

Брюс М. Ван Хорн II, Куан Нгуен

PyCharm:

профессиональная
работа на Python



Брюс М. Ван Хорн II
Куан Нгуен

PyCharm: профессиональная работа на Python

Hands-On Application Development with PyCharm

Build applications like a pro with the ultimate
python development tool

Bruce M. Van Horn II
Quan Nguyen



BIRMINGHAM—MUMBAI

PyCharm: профессиональная работа на Python

Брюс М. Ван Хорн II
Куан Нгуен



Москва, 2024

УДК 004.438Python

ББК 32.973

B17

Брюс М. Ван Хорн II, Куан Нгуен

B17 PyCharm: профессиональная работа на Python / пер. с англ. И. Л. Люско. – М.: ДМК Пресс, 2024. – 618 с.: ил.

ISBN 978-5-93700-274-7

PyCharm – лучшая профессиональная среда разработки для программистов Python среди множества доступных интегрированных сред. Независимо от того, в каких целях используется Python – для общих задач автоматизации, создания утилит, веб-приложений, анализа данных, машинного обучения или бизнес-приложений, – инструменты PyCharm упрощают выполнение сложных задач и оптимизируют общую производительность.

Вы изучите базовые и расширенные функции PyCharm, узнаете как разрабатывать веб-приложения с помощью Flask, Django, FastAPI и Pyramid, освоите автоматизацию написания кода, отладку и удаленную разработку в PyCharm, а также научитесь выполнять задачи по обработке данных с помощью блокнотов Jupyter, библиотек NumPy, pandas и других.

Издание адресовано как опытным разработчикам на Python, так и новичкам.

УДК 004.438Python

ББК 32.973

Все права защищены. Любая часть этой книги не может быть воспроизведена в какой бы то ни было форме и какими бы то ни было средствами без письменного разрешения владельцев авторских прав.

Оглавление

Предисловие от издательства	16
Авторы	17
Предисловие	19
ЧАСТЬ I. ОСНОВЫ PYCHARM	25
Глава 1. Введение в PyCharm – самую популярную IDE для Python.....	27
Технические требования.....	30
Продолжающийся успех Python	31
Философия IDE.....	32
PyCharm как интегрированная среда разработки Python	37
Интеллектуальная помощь в кодировании.....	39
Оптимизированные инструменты программирования.....	43
Опции веб-разработки.....	50
Поддержка научных вычислений.....	51
Особенности редакций Professional, Community и Educational	52
Краткое содержание	53
Вопросы.....	53
Дальнейшее чтение	54
Глава 2. Установка и настройка PyCharm	55
Технические требования.....	56
Загрузка PyCharm традиционным способом.....	57
JetBrains Toolbox	58
Установка Toolbox в Windows	60
Установка Toolbox в macOS	60
Установка PyCharm с помощью Toolbox	60
Запуск PyCharm с помощью Toolbox.....	62
Установка альтернативной версии или ее удаление	62
Обновление PyCharm с помощью Toolbox.....	63
Запуск и регистрация PyCharm.....	64
Настройка PyCharm	65
Внешний вид и свойства.....	67

Работа с проектами	76
Создание нового проекта.....	77
Запуск проекта PyCharm	81
Клонирование кода этой книги с GitHub	83
Настройка учетной записи GitHub	84
Клонирование репозитория книги	85
Краткое содержание	86
Вопросы.....	87
Дальнейшее чтение	88

ЧАСТЬ II. ПОВЫШЕНИЕ ПРОИЗВОДИТЕЛЬНОСТИ 89

Глава 3. Настройка интерпретаторов

и виртуальные среды 91

Технические требования.....	93
Виртуальные среды	94
Создание виртуальной среды вручную.....	95
Создание проекта в PyCharm (повторно).....	98
Использование существующей виртуальной среды	100
Смена интерпретатора для проекта.....	102
Активация виртуального окружения	104
Использование встроенного терминала	104
Работа с REPL в окне консоли	105
Работа со сторонними библиотеками пакетов.....	107
Добавление сторонних библиотек в PyCharm	110
Удаление сторонних библиотек в PyCharm	111
Использование файла requirements.txt.....	112
Новое окно Python Packages	113
Функции версии Professional, важные для виртуальных сред	115
Импорт проектов в PyCharm.....	116
Импорт клонированного проекта из репозитория.....	119
Работа с неисправными интерпретаторами	121
Работа с конфигурациями запуска.....	123
Файлы проекта PyCharm	128
Краткое содержание	129
Вопросы.....	131
Дальнейшее чтение	131

Глава 4. Редактирование и форматирование

с легкостью в PyCharm 132

Технические требования.....	133
-----------------------------	-----

Анализ кода, проверка и предложения.....	133
Пешыте бес ашыбок	135
Это понимает ваш код.....	136
Завершение постфиксного кода	137
Завершение хиппи	138
Индексация	140
Энергосберегающий режим.....	141
Настройка завершения кода	143
Сопоставление регистров	143
Сортировка предложений по алфавиту	144
Завершения с помощью машинного обучения.....	145
Отображение всплывающего окна документации через [...] мс	146
Инфо параметров	146
Анализ кода и автоматические исправления.....	147
Обнаружение проблем	148
Синтаксические ошибки.....	151
Дублированный код.....	152
Проблемы PEP-8	152
Мертвый код	152
Несоответствие сигнатуры метода.....	153
Дорога к хорошему коду вымощена намерениями PyCharm.....	153
Рефакторинг.....	155
Что такое рефакторинг?.....	155
Инструменты рефакторинга в PyCharm	156
Документация	169
Работа со строками документации	170
Краткое содержание	175
Вопросы.....	175
Дальнейшее чтение	175

Глава 5. Контроль версий в PyCharm с помощью Git..... 176

Технические требования.....	177
Контроль версий и основы Git	177
Настройка Git на вашем компьютере.....	179
Установка имени пользователя и адреса электронной почты по умолчанию	180
Генерация ключа SSH	180
Добавление ключа SSH в вашу учетную запись GitHub	181
Настройка репозитория вручную.....	182
Мастер-ветви и главные ветви в GitHub	183
Ручная инициализация репозитория	184
Работа с удаленным доступом.....	186

Добавление удаленного доступа на GitHub	186
Первая передача	187
Внесение, принятие и передача изменений	187
Работа с Git в IDE	188
Контроль версий в PyCharm.....	188
Создание нового проекта с нуля с использованием инструментов VCS в PyCharm.....	192
Инициализация локального репозитория Git	193
Добавление удаленного доступа на GitHub	194
Добавление файлов проекта	197
Добавление файла .gitignore	197
Получение и отправка	200
Ветвление и слияние	202
Создание ветки	203
Переключение между ветками	204
Слияние	204
Просмотр диаграммы ветвей	205
Различия и разрешение конфликтов	206
Просмотр различий.....	210
Краткое содержание	211
Вопросы.....	211
Дальнейшее чтение	212

Глава 6. Бесшовное тестирование, отладка

и профилирование	213
Технические требования.....	214
Тестирование, тестирование, 1-2-3	214
Модульное тестирование на Python с использованием PyCharm	215
Выбор тестовой библиотеки	218
Добавление класса банковского счета	219
Тестирование класса банковского счета	219
Запуск тестов	221
Исправление неудачных тестов	223
Тестирование на отказ	224
Автоматическое создание тестов	225
Создание теста транзакции	226
Работа с отладчиком PyCharm	234
Использование отладчика для поиска и устранения проблемы, выявленной тестом	238
Проверка тестового покрытия.....	240
Выходные данные тестового покрытия.....	245
Профилирование производительности	246

Профилирование в PyCharm.....	247
Сравнение производительности и встроенной функции sum().....	249
Просмотр графа вызовов	250
Навигация с помощью профиля производительности.....	251
Скриншоты профиля производительности cProfile.....	251
Краткое содержание	252
Вопросы.....	253
Дальнейшее чтение	253

ЧАСТЬ III. ВЕБ-РАЗРАБОТКА В PYCHARM 255

Глава 7. Веб-разработка с JavaScript, HTML и CSS 257

Технические требования	258
Введение в HTML, JavaScript и CSS	258
Написание кода с помощью HTML.....	259
Создание HTML в PyCharm	261
Создание пустого проекта	261
Предварительный просмотр веб-страниц.....	264
Перезагрузка представления браузера при сохранении	265
Использование предварительного просмотра HTML в PyCharm.....	266
Настройка доступных браузеров.....	267
Навигация по коду структуры с помощью окна структуры.....	267
Добавление CSS	268
Использование селекторов цвета.....	270
Добавление JavaScript.....	270
Добавляем немного кода JavaScript	270
Добавление элементов в HTML-файл	271
Отладка JavaScript на стороне клиента	272
Работа с шаблонами Emmet	274
Типы HTML-проектов в PyCharm Professional.....	275
Шаблон HTML 5.....	276
Предварительный просмотр и редактирование графики с помощью внешних инструментов.....	277
Загрузка вашего сайта на сервер.....	279
Создание проекта Bootstrap.....	286
Работа с современными JavaScript и NodeJS.....	287
Создание проекта NodeJS.....	287
Создание проекта React	287
Другие фреймворки для фронтенда	288
Краткое содержание	289
Вопросы.....	289

Глава 8. Создание динамических сетевых приложений с Flask..... 291

Технические требования.....	292
Основы веб-технологий – архитектура клиент–сервер.....	293
Изучение механизма запроса–ответа в HTTP – как взаимодействуют клиенты и серверы	296
Что такое Flask?.....	297
Обработка и маршрутизация запросов и ответов с помощью Werkzeug	298
Создание шаблонов с помощью Jinja2	299
Примечание о наименовании файлов и папок	301
Создание приложения Flask в PyCharm Professional	302
Создание динамического веб-приложения.....	304
Настройка статических частей	304
Запуск приложения Flask	309
Давайте сделаем, чтобы это выглядело немного лучше.....	313
Добавляем немного CSS	314
Делаем страницу динамичной	316
Улучшения редактора для работы с Flask и Jinja2	318
Краткое содержание	319
Дальнейшее чтение	320

Глава 9. Создание RESTful API с помощью FastAPI 321

Технические требования.....	324
В жестоком мире стейтлесс нет REST.....	324
Создание проекта FastAPI в PyCharm Professional	328
Запуск проекта FastAPI	330
Работа с HTTP-запросами PyCharm	333
Рассмотрение деталей возврата	335
Мы только что создали новую конфигурацию запуска	338
Использование Before в конфигурациях запуска.....	339
Работа со средами HTTP-запросов Request	344
Операции CRUD	346
Больше тестирования.....	349
Создание тестов.....	353
Редактирование и отладка полнофункционального приложения путем соединения проектов	355
Создание приложения React в отдельном проекте	356
Привязка проекта к проекту FastAPI, который мы создали ранее	357
Краткое содержание	358
Вопросы.....	359
Дальнейшее чтение	359

Глава 10. Полнофункциональные фреймворки – Django и Pyramid..... 361

Технические требования	362
Что за суматоха вокруг Джанго?	362
Компоненты фреймворка Django	364
Создание проекта Django	365
Структура проекта Django	367
Начальная конфигурация	368
Запуск проекта Django	369
Создание моделей Django	371
Миграция с помощью manage.py	375
Интерфейс администратора Django	378
Создание суперпользователя и вход в систему	378
Добавление моделей Author и Book в интерфейс администратора	380
Создание представлений Django	382
Что за странный значок Python в желобе шаблона?	385
Запустим его!	386
Создание приложений Pyramid с помощью PyCharm	386
Создание проекта Pyramid	387
Краткое содержание	390
Вопросы	391
Дальнейшее чтение	391

Глава 11. Понимание управления базами данных в PyCharm.... 392

Технические требования	394
Оболочки реляционных баз данных	395
Язык структурированных запросов	395
Две половины SQL	396
Взаимосвязи	397
Больше реляционных структур	399
В терминологии базы данных используются простые английские формы множественного числа	400
Инструментарий баз данных в PyCharm	401
Настройка сервера базы данных MySQL с помощью Docker	401
Установка и запуск контейнера MySQL	402
Остановка и запуск контейнера	404
Подключение к источникам данных с помощью PyCharm	404
Создание новой базы данных	408
Установка диалекта SQL (это важно)	409
Группирование и цветовое кодирование источников данных	411
Создание базы данных и манипуляции с ней	416
Создание таблицы	416

Изменение существующих структур.....	426
Генерация скриптов	426
Запрос к источнику данных с помощью SQL.....	429
Специальные запросы.....	429
Генерация операторов SQL	431
Запуск запроса	433
Экспорт результатов запроса.....	435
Работа с файлами SQL	436
Краткое содержание	437
Дальнейшее чтение	438

ЧАСТЬ IV. ОБРАБОТКА ДАННЫХ С ПОМОЩЬЮ PYCHARM 439

Глава 12. Включаем научный режим..... 441

Технические требования.....	442
Создание научного проекта в PyCharm	442
Дополнительная конфигурация для научных проектов в PyCharm	445
Плагин Markdown.....	447
Добавление изображений	448
Установка плагина CSV	449
Установка плагина режима ячеек.....	450
Установка пакетов	452
Заполните файл requirements.txt	453
Добавляем научный код.....	453
Переключение научного режима	455
Понимание расширенных возможностей научных проектов PyCharm	457
Окно просмотра документации	457
Использование ячеек кода в PyCharm.....	459
Использование ячеек кода PyCharm	459
Плагин режима ячеек	461
Краткое содержание	462
Вопросы.....	463

Глава 13. Динамический просмотр данных с помощью SciView и Jupyter 464

Технические требования.....	464
Просмотр данных с помощью панели SciView PyCharm – легко и просто.....	465
Просмотр диаграмм и работа с ними.....	466
Тепловые карты и корреляционные данные	468
Просмотр данных и работа с ними	470
Фильтрация на вкладке Data	474

Понимание IPython и магических команд	478
Установка и настройка IPython.....	479
Знакомство с Ipython и магическими командами	481
Использование блокнотов Jupyter	485
Понимание основ Jupyter	486
Идея итеративной разработки	486
Блокноты Jupyter в PyCharm	488
Создаем блокнот и добавляем наш код	488
Документирование с помощью Markdown и LaTeX	491
Добавляем наши диаграммы	492
Запуск ячеек.....	492
Нечетности и завершения	495
Краткое содержание	497
Вопросы.....	498

Глава 14. Создание конвейера данных в PyCharm..... 499

Технические требования	500
Работа с наборами данных	500
Начнем с вопроса	501
Архивированные пользовательские данные.....	502
Тарру-данные.....	504
Сбор данных.....	506
Загрузка из внешнего источника	506
Ручной сбор данных и веб-скрейпинг	506
Сбор данных через третьих лиц	507
Экспорт баз данных.....	507
Контроль версий наборов данных.....	507
Использование поддержки больших файлов Git.....	508
Очистка и предварительная обработка данных.....	512
Пример токсичных данных с участием ninja	513
Исследовательский анализ в PyCharm	514
Очистка данных.....	521
Изучение второго набора данных.....	527
Рефакторинг для масштабирования	531
Анализ данных и insight	536
Запускаем блокнот и считываем обработанные данные.....	536
Использование диаграмм и графиков	538
Аналитика на основе машинного обучения.....	542
Скрипты против блокнотов в науке о данных.....	545
Краткое содержание	546
Вопросы.....	547
Дальнейшее чтение	547

ЧАСТЬ V. ПЛАГИНЫ И ЗАКЛЮЧЕНИЕ 549**Глава 15. Больше возможностей с плагинами 551**

Технические требования.....	552
Плагины в комплекте и JetBrains Marketplace	552
Окно плагинов	552
Связанные плагины.....	553
JetBrains Marketplace	554
Создание собственных плагинов	554
Плагины, необходимые для ваших проектов	555
Разные полезные плагины.....	556
Кодируйте со мной (и вы никогда больше не будете одиноким).....	560
Удаленная разработка	562
Настройка удаленной разработки в PyCharm.....	564
Создание удаленного проекта	570
Давайте попробуем еще раз	574
Создание виртуальной среды на удаленном компьютере	575
Другие соображения.....	576
Работа с Docker	577
Входящий в комплект PyCharm плагин Docker	578
Создаем проект.....	578
Добавьте конфигурацию запуска Docker	580
Краткое содержание	584

Глава 16. Ваши следующие шаги с PyCharm 586

Разнообразие возможностей PyCharm.....	587
Удаленные виртуальные среды	587
Работа с HashiCorp Vagrant.....	588
Отслеживание вашего времени.....	594
TODO – список дел	596
Макросы	597
Уведомления.....	599
Новые возможности версии 2023.2	600
Интеграция с Black	602
Интеграция с GitLab	604
Запускайте все что угодно!	605
AI-помощник	606
Поддержка Jupyter Notebook для Polars.....	610
Резюме и заключительные замечания.....	610
Дальнейшее чтение	613

Предметный указатель 614

Моим дочерям Китти и Фиби и моей жене Карине. Для моей команды в Visual Storage Intelligence. Во имя моего Господа и Спасителя Иисуса Христа. «Служите друг другу каждый тем даром, какой получил, как добрые домостроители многообразной благодати Божией». – 1 Петра 4:10.

Брюс М. Ван Хорн II

Двум моим великим учителям в жизни: моей матери Чи Лан и моему отцу Бангу. В память о моей бабушке и двух моих дорогих дедушках.

Куан Нгуен

Предисловие от издательства

Отзывы и пожелания

Мы всегда рады отзывам наших читателей. Расскажите нам, что вы думаете об этой книге – что понравилось или, может быть, не понравилось. Отзывы важны для нас, чтобы выпускать книги, которые будут для вас максимально полезны.

Вы можете написать отзыв на нашем сайте www.dmkpress.com, зайдя на страницу книги и оставив комментарий в разделе «Отзывы и рецензии». Также можно послать письмо главному редактору по адресу dmkpress@gmail.com; при этом укажите название книги в теме письма.

Если вы являетесь экспертом в какой-либо области и заинтересованы в написании новой книги, заполните форму на нашем сайте по адресу http://dmkpress.com/authors/publish_book/ или напишите в издательство по адресу dmkpress@gmail.com.

Список опечаток

Хотя мы приняли все возможные меры для того, чтобы обеспечить высокое качество наших текстов, ошибки все равно случаются. Если вы найдете ошибку в одной из наших книг – возможно, ошибку в основном тексте или программном коде, – мы будем очень благодарны, если вы сообщите нам о ней. Сделав это, вы избавите других читателей от недопонимания и поможете нам улучшить последующие издания этой книги.

Если вы найдете какие-либо ошибки в коде, пожалуйста, сообщите о них главному редактору по адресу dmkpress@gmail.com, и мы исправим это в следующих тиражах.

Нарушение авторских прав

Пиратство в интернете по-прежнему остается насущной проблемой. Издательство «ДМК Пресс» очень серьезно относится к вопросам защиты авторских прав и лицензирования. Если вы столкнетесь в интернете с незаконной публикацией какой-либо из наших книг, пожалуйста, пришлите нам ссылку на интернет-ресурс, чтобы мы могли применить санкции.

Ссылку на подозрительные материалы можно прислать по адресу dmkpress@gmail.com.

Мы высоко ценим любую помощь по защите наших авторов, благодаря которой мы можем предоставлять вам качественные материалы.

Авторы

ОБ АВТОРАХ

Брюс М. Ван Хорн II – ведущий главный инженер-программист подразделения Visual Storage Intelligence. Он специализируется на разработке программного обеспечения и веб-разработке на Python, C# и JavaScript. Он также отвечает за DevOps и является сертифицированным Scrum-мастером продвинутого уровня (A-CSM). Имея более чем 30-летний опыт создания и распространения успешного программного обеспечения, он также имеет 25-летний опыт преподавания, полученный в ходе вечерних занятий в колледжах и университетах недалеко от его дома в Далласе, штат Техас. Ван Хорн – автор нескольких книг и серий видео, опубликованных Packt, Skillsoft, Lynda.com и LinkedIn Learning, включая оригинальный видеокурс LinkedIn по PyCharm. Его проекты за прошедшие годы завоевали множество престижных наград, но самым большим успехом на сегодняшний день является достижение его команды, получившее награду IS&T BMC США. Вы можете связаться с ним в LinkedIn по адресу <https://www.linkedin.com/in/brucevanhorn2/>.

Куан Нгуен, автор первого издания этой книги, – программист на Python, страстно увлеченный машинным обучением. Он имеет двойную степень – по математике и информатике, а также степень по философии, полученную в Университете ДеПау. Куан активно участвует в сообществе Python и является автором нескольких книг по Python, вносит свой вклад в Python Software Foundation и регулярно делится своими мыслями на DataScience.com. В настоящее время работает над докторской диссертацией по компьютерным наукам в Вашингтонском университете в Сент-Луисе, вы можете найти его в LinkedIn по адресу <https://www.linkedin.com/in/quan-m-nguyen/>.

О РЕЦЕНЗЕНТАХ

Доктор Говришанкар С. Натх – профессор и декан факультета компьютерных наук и инженерии Технологического института доктора Амбедкара в Бангалоре, Индия. Он получил степень доктора инженерных наук в Университете Джадхавпур, Калькутта, Индия, в 2010 году, а также степень магистра технических наук в области разработки программного обеспечения и степень бакалавра компьютерных наук и инженерии в Технологическом университете Висвесварая (VTU) в 2005 и 2003 годах соответственно. Его исследовательские интересы включают применение машинного обучения, интеллектуального анализа данных и анализа больших данных в здравоохранении. Вы найдете его в LinkedIn по адресу <https://www.linkedin.com/in/gowrishankarnath/>.

Уокер Кристал – мастер, творческий человек в душе, который любит разysкивать и решать проблемы. Он получил степень бакалавра в области автомо-

бильных инженерных технологий в Университете Бригама Янга, штат Айдахо. Работал тестировщиком системной интеграции у производителя автомобилей, разработчиком программного обеспечения в компании из списка Fortune 500, а также DevOps-инженером-разработчиком программного обеспечения в крупной автотранспортной компании и в других областях. Его пристальное внимание к деталям и природное любопытство помогают ему задавать правильные вопросы и разрабатывать инновационные и творческие решения проблем. Он считает, что технологии – это мультипликатор силы, который может увеличить вашу способность творения добра и расширить ваше положительное влияние на клиентов по всему миру. В свободное время он создает 3D-модели CAD и печатает их на 3D-принтере, играет с конструктором Lego, ремонтирует старые и новые автомобили, исследует новые технологии и проводит время со своей семьей. Вы можете найти его в LinkedIn по адресу <https://www.linkedin.com/in/nitrospaz/>.

Карина Ван Хорн имеет двойную степень в области политологии и творческого письма Южного методистского университета, а также степень доктора юридических наук Техасского Уэслианского университета. Несмотря на то что ей не нравится все, что связано с технологиями, у нее есть талант писать великолепные статьи.

Предисловие

Добро пожаловать в мир программирования на Python с PyCharm! В этой книге мы отправляемся в путешествие по универсальной и динамичной сфере разработки Python, чему способствует интегрированная среда разработки PyCharm. Независимо от того, являетесь ли вы программистом-новичком, только начинающим свое приключение в программировании, или опытным разработчиком, желающим улучшить свои навыки Python, эта книга станет вашим надежным помощником.

Python стал одним из самых популярных и универсальных языков программирования, известных своей простотой и читабельностью. Благодаря своей богатой экосистеме библиотек и фреймворков Python используется в широком спектре приложений: от веб-разработки и анализа данных до искусственного интеллекта и научных вычислений. PyCharm, разработанная JetBrains, – это ведущая среда разработки Python, которая предоставляет программистам надежный набор инструментов и функций для эффективной разработки кода, отладки и совместной работы.

В следующих главах мы изучим основы PyCharm, углубимся в расширенные возможности настройки и воспользуемся профессиональной версией PyCharm для оптимизации рабочего процесса кодирования. Независимо от того, стремитесь ли вы создавать веб-приложения, автоматизировать задачи, анализировать данные или разрабатывать модели машинного обучения, эта книга предоставит вам знания и навыки, позволяющие воплотить ваши идеи в реальность.

Наша цель – сделать ваше путешествие по программированию на Python не только познавательным, но и приятным. На протяжении всей книги мы приводим практические примеры, даем упражнения и представляем реальные проекты, чтобы укрепить ваше понимание и разжечь ваш творческий потенциал. К тому времени, как вы дойдете до последней страницы, у вас будет уверенность и опыт для реализации проектов Python любого масштаба и сложности с использованием PyCharm в качестве единственного инструмента.

Итак, давайте вместе отправимся в это захватывающее приключение, раскрывая красоту программирования на Python и используя возможности PyCharm, чтобы превратить ваши стремления к кодированию в осязаемые достижения. Приятного кодирования!

Для кого эта книга

Эта книга предназначена для разнообразной аудитории людей, которые интересуются программированием на Python и хотят использовать интегрированную среду разработки (IDE) PyCharm для улучшения своего опыта программирования. Вот основные группы людей, для которых предназначена эта книга.

- **Начинающие программисты.** Если вы новичок в программировании или у вас ограничен опыт программирования, эта книга представля-

ет собой краткое введение в Python и PyCharm. По своей природе PyCharm упрощает изучение Python, предоставляя большую помощь в настройке шаблонов проектов, автозаполнении и автоматическом форматировании PEP-8¹ для вашего кода.

- **Разработчики Python среднего уровня.** Если у вас уже есть некоторый опыт работы с Python, но вы хотите углубить свои знания и навыки, эта книга вместе с PyCharm может вам помочь. Настраиваемая технология линтера² и анализа кода PyCharm обеспечит соответствие вашей работы самым высоким стандартам. Рефакторинг, практика, которой слишком часто пренебрегают, становится тривиальной благодаря инструментам индексации и рефакторинга PyCharm. Вы научитесь использовать интегрированные инструменты тестирования, покрытия и профилирования, чтобы обеспечить быструю и надежную работу вашего кода.
- **Опытные разработчики, работающие на других языках.** Если вы опытный программист на другом языке и хотите перейти на Python или включить Python в свой набор навыков, наша книга поможет вам преодолеть этот пробел и освоить программирование на Python с помощью PyCharm. Это особенно актуально, если вы использовали другие IDE PyCharm. Если вы обычно пользуетесь IntelliJ Idea, WebStorm, Rider или PHP Storm, вы будете чувствовать себя как дома, поскольку будете использовать те же сочетания клавиш. Если вы работаете на Visual Studio, то можете легко настроить PyCharm для использования привычных вам сочетаний клавиш, и мы думаем, что обычные рабочие процессы, такие как работа с Git, будут более удобными и интуитивно понятными и в PyCharm.
- **Студенты и преподаватели.** Python – популярный язык для преподавания и изучения программирования. Эта книга может служить ценным ресурсом для студентов, изучающих Python в рамках своей курсовой работы, а также для преподавателей, которые ищут подробное руководство по эффективному обучению Python с помощью PyCharm.
- **Специалисты по данным и аналитики.** Python широко используется в области анализа данных и машинного обучения. В этой книге рассматриваются расширенные библиотеки и инструменты для исследования, манипулирования, очистки и анализа данных, что делает ее ценной для специалистов по данным, стремящихся улучшить свои навыки Python в контексте PyCharm. PyCharm содержит мощный набор инструментов для работы с реляционными и нереляционными базами данных, которые полностью описаны.

¹ PEP-8 – это документ, описывающий стандарты, которым разработчики должны следовать при написании кода на Python, аналог Clean Code для Java. Следование этим рекомендациям значительно улучшает качество кода, делает его более читаемым и понятным для других разработчиков. Рекомендации касаются отступов, длины строки, использования пробелов, соглашения о наименованиях переменных, функций и методов и т.д. – *Прим. ред.*

² Линтер (англ. *linter*) – это инструмент программирования, который используется для анализа исходного кода программного обеспечения с целью выявления потенциальных проблем, структурных ошибок, стилевых нарушений и других недостатков. – *Прим. ред.*

- **Веб-разработчики.** Для тех, кто интересуется веб-разработкой с использованием Python, в этой книге рассматриваются веб-фреймворки и инструменты, позволяющие создавать динамические веб-приложения с использованием PyCharm в качестве инструмента разработки. Вы научитесь создавать проекты в популярных средах веб-разработки, таких как Flask, FastAPI и Django. Большинство людей этого не знают, но PyCharm Professional содержит полную интегрированную среду разработки, предназначенную для разработки JavaScript и HTML. В этой книге все это описано очень подробно.
- **Все, кто интересуется Python и PyCharm.** Если у вас есть общий интерес к программированию, технологиям или Python в частности, эта книга предлагает увлекательное исследование возможностей Python и функций PyCharm, что делает ее доступной и информативной для широкого круга читателей.

Независимо от вашего опыта или уровня знаний, эта книга призвана стать ценным ресурсом для всех, кто хочет изучить программирование на Python и использовать возможности PyCharm для написания эффективного, читаемого и поддерживаемого кода.

О ЧЕМ ЭТА КНИГА

Глава 1 «Знакомство с PyCharm, самой популярной IDE для Python». В этой начальной главе мы обсуждаем весь наш дальнейший путь.

Глава 2 «Установка и настройка». В этой главе представлен процесс установки, а также инструкции по настройке PyCharm в соответствии с вашим конкретным стилем разработки.

Глава 3 «Настройка интерпретаторов и виртуальных сред». Одной из очень полезных функций экосистемы Python является возможность помещать ваши проекты в «песочницу». PyCharm предоставляет проектно-ориентированный графический инструмент для управления вашими проектами и соответствующими интерпретаторами и виртуальными средами.

Глава 4 «Простое редактирование и форматирование в PyCharm». Сердце любой великолепной IDE – ее редактор. Эта глава дает четкую ориентацию в теме.

Глава 5 «Контроль версий с помощью Git в PyCharm». Все, что вы обычно делаете в командной строке, можно сделать графически в среде IDE. Здесь показано, как это делается.

Глава 6 «Бесшовное тестирование, отладка и профилирование». PyCharm поддерживает различные среды модульного тестирования непосредственно в среде IDE. Вы научитесь писать тесты и визуализировать результаты в PyCharm.

Глава 7 «Веб-разработка с использованием JavaScript, HTML и CSS». PyCharm – это полноценная среда для полнофункциональной разработки. Таким образом, вы научитесь разрабатывать HTML, JavaScript и CSS в PyCharm. Мы кратко рассмотрим несколько интерфейсных фреймворков, таких как HTML Boilerplate, Bootstrap и React.

Глава 8 «Создание динамического веб-приложения с помощью Flask». Flask – это нестандартная среда для создания веб-приложений, способных обслуживать динамический контент. PyCharm делает это очень легко.

Глава 9 «Создание RESTful API с помощью FastAPI». В этой главе вы научитесь создавать RESTful API с помощью FastAPI. Вы также научитесь тестировать API с помощью встроенной среды HTTP-запросов и тестирования PyCharm.

Глава 10 «Дополнительные полнофункциональные фреймворки: Django и Pyramid». PyCharm содержит специализированные инструменты для Django, одной из самых популярных веб-фреймворков на Python. Мы также коснемся Pyramid, фреймворка, который менее сложен, чем Django, но более полный, чем Flask.

Глава 11 «Понимание управления базами данных в PyCharm». PyCharm содержит полнофункциональную интегрированную среду разработки баз данных, упрощающую вашу работу с десятками реляционных и нереляционных (NoSQL) платформ данных.

Глава 12 «Включаем научный режим». Вы познакомитесь с основами научного режима PyCharm, который является основой его инструментов для обработки данных.

Глава 13 «Динамический просмотр данных с помощью SciView и Jupyter». Вы научитесь использовать возможность видеть данные на каждом этапе многофазного конвейера данных, что неоценимо. PyCharm поддерживает расширенное окно просмотра, которое визуализирует структуры данных NumPy и Pandas.

Глава 14 «Построение конвейера данных в PyCharm». В PyCharm есть все необходимое для выполнения расширенного научного анализа данных. В этой главе мы анализируем научное исследование, призванное предсказать раннее начало болезни Альцгеймера.

Глава 15 «Больше возможностей с плагинами». Многие функции IDE JetBrains реализованы с помощью плагинов. Торговая площадка JetBrains позволяет вам дополнить установку PyCharm еще более специализированными функциями.

Глава 16 «Будущие разработки». JetBrains не стоит на месте. PyCharm быстро развивается. В этой главе показаны некоторые функции, которые на момент написания находились в активной разработке.

Чтобы получить максимальную пользу от этой книги

Я предполагаю, что вы знаете основы программирования на Python, а также имеете базовые навыки работы с командной строкой для вашей любимой операционной системы. Следует всегда помнить, что мы рассматриваем PyCharm, а не одно из руководств по глубокой разработке платформ, упомянутых в различных главах. Например, в главе 8 рассматриваются функции PyCharm, предназначенные для разработки на Flask. Это не полный учебник по Flask.

Программное/аппаратное обеспечение, описанное в книге	Требования к операционной системе
Python 3	Windows, macOS или Linux
PyCharm Professional	Windows, macOS или Linux
Docker Desktop	Windows, macOS или Linux
Git	Windows, macOS или Linux

Для большей части книги требуется профессиональная версия PyCharm. Первые шесть глав будут работать с версией Community, но после этого вам понадобится версия Professional.

Если вы используете цифровую версию этой книги, мы советуем вам ввести код самостоятельно или получить доступ к коду из репозитория книги на GitHub (ссылка доступна в следующем разделе). Это поможет вам избежать любых потенциальных ошибок, связанных с копированием и вставкой кода.

ЗАГРУЗИТЕ ФАЙЛЫ ПРИМЕРОВ КОДА

Вы можете загрузить файлы примеров кода для этой книги с GitHub по адресу <https://github.com/PacktPublishing/Hands-On-Application-Development-with-PyCharm--Second-Edition>. Если есть обновление кода, оно будет обновлено в репозитории GitHub. В главе 2 рассказывается о клонировании репозитория с помощью встроенного клиента Git PyCharm.

У нас также есть другие пакеты кода из нашего богатого каталога книг и видео, доступных на <https://github.com/PacktPublishing/>. Просмотрите их!

ИСПОЛЬЗУЕМЫЕ СОГЛАШЕНИЯ

В этой книге используется ряд текстовых соглашений.

`Code in text`: указывает кодовые слова в тексте, имена таблиц базы данных, имена папок, имена файлов, расширения файлов, имена путей, фиктивные URL-адреса, пользовательский ввод и дескрипторы Twitter. Вот пример: «Мы вычисляем корреляционную матрицу этого набора данных с помощью метода `corr()`».

Блок кода задается следующим образом:

```
# Compute and show correlation matrix
corr_mat = df.corr()

plt.matshow(corr_mat)
plt.show()
```

Любой ввод или вывод командной строки записывается следующим образом:

```
$ mkdir css
$ cd css
```

Жирный шрифт: обозначает новый термин, важное слово или слова, которые вы видите на экране. Например, слова в меню или диалоговых окнах выделяются **жирным шрифтом**. Вот пример: «Если вы кликнете ссылку **View as Array**, которую также можно активировать, кликнув правой кнопкой мыши переменную, то увидите таблицу, похожую на электронную таблицу, на панели **Data**».

Советы или важные примечания

Представлены так.

Часть I

Основы PyCharm

Эта часть знакомит читателей с PyCharm и предлагает подробное описание того, как загрузить, установить и начать использовать PyCharm для своих проектов Python.

Эта часть состоит из следующих глав:

- глава 1 «Введение в PyCharm – самую популярную IDE для Python»,
- глава 2 «Установка и настройка PyCharm».

Введение в PyCharm – самую популярную IDE для Python

Добро пожаловать во второе издание «*PyCharm: профессиональная работа на Python*» Целью большинства программистов является создание надежного, высококачественного программного обеспечения, способного выдержать испытание временем. Самый важный шаг на пути к этой цели – выбор правильного языка программирования. Какой из языков выбрать лучше, если их существует так много? Выдающийся программист примет во внимание множество особенностей языка. Одним из наиболее важных аспектов языка программирования, который следует учитывать, являются инструменты поддержки, необходимые на этапах разработки. Ходят слухи, что язык программирования **Python** обеспечивает большую производительность по сравнению со многими другими языками. Знаменитая философия «*батарейки в комплекте*»¹ языка Python воплощает эту идею, объединяя мощную стандартную библиотеку, редактор кода и отладчик. Все встроено в обычный инсталлятор языка, который доступен по адресу <https://www.python.org>. Есть только одна маленькая проблема, по крайней мере для меня, – это Microsoft.

Я знаю, о чем вы подумали. Вы только что мысленно подготовились к продолжительной напыщенной речи одного из тех парней из **Unix/Linux**, что жалуются на большое зло, которым является Microsoft. Я не собираюсь этого делать, потому что не уверен, что я фанат Linux. Я имею в виду, что у меня в шкафу очень много брюк с карманами. Я ничего не могу с этим поделать. Они такие вместительные, что можно носить с собой все свои вещи, не таская сумку. Я также признаю, что у меня есть много футболок с эмблемами, логотипами или утверждениями, которые, возможно, поймут только 5 % людей, с кем я сталкиваюсь. Эти футболки очень забавные, но единственные улыбки, которые я получаю, – от моих коллег. Чем больше я об этом думаю, тем больше я не сторонник Linux. Для меня Linux – это инструмент. Иногда это правильно, но иногда – нет. Поскольку я не фанат Linux, это не может быть причиной моего

¹ Это означает, что Python поставляется с набором полезных модулей и пакетов, включенных в его стандартную библиотеку, и разработчикам Python не нужно искать и устанавливать внешние библиотеки для общих задач. – *Прим. ред.*

заявления о том, что проблема в Microsoft. Настоящая причина совершенно противоположна. Около 30 лет назад Microsoft сделала что-то совершенно правильное. Они создали первую действительно хорошую коммерчески доступную **интегрированную среду разработки (IDE)**.

По правде говоря, возможно, это произошло более 30 лет назад, а до этого могли быть и другие попытки. Однако сегодня многие «старшие разработчики» в сфере программного обеспечения начали свою карьеру с продукта Microsoft под названием **Visual Basic (VB)**. Допустим, это тот самый момент, когда языковые снобы ухмыляются и зажимают носы, как будто им только что подарили тарелку брюссельской капусты или, может быть, грязный подгузник, но давайте отмотаем обратно. 30 лет назад большинство людей, юзающих домашние компы, имели только несколько вариантов. **Универсальный код символьных инструкций для начинающих (BASIC)** поставлялся практически с каждым компьютером, выпущенным с 1978 года. Это была эпоха, когда даже у Apple не было **графического пользовательского интерфейса (GUI)** в их **операционной системе (ОС)**. Его не было до 1983 года, когда Apple выпустила компьютер Lisa. У нас были мыши, и мы могли создавать программы, способные работать с указывающими устройствами, но в ОС не было оконной системы. Им не нужна была оконная система, потому что тогда компьютеры могли запускать только одну программу одновременно.

Написание программного обеспечения для десктоп-компьютеров, в которых не было поддержки Windows на уровне ОС, было трудным. Не существовало **комплектов разработки программного обеспечения (SDK)** или **интерфейса прикладного программирования (API)**, которые могли бы справиться с какой-либо серьезной работой. Написание программного обеспечения было в основном утомительным занятием. Вам приходилось писать сотни строк шаблонного кода для рисования блоков в инструменте, который был немногим лучше Блокнота (Notepad). Затем, в 1991 году, когда я окончил Университет Оклахомы, все изменилось.

Microsoft выпустила версию BASIC, которая включала возможность создания **графических пользовательских интерфейсов** прямо в среде разработки. Они называли это *Visual Basic*. Первые версии работали в **дисковой операционной системе Microsoft (MS-DOS)**, но позже появилась поддержка Windows, Windows 2, а затем и Windows 3.1. Последняя имела большое значение, потому что именно тогда мы получили настоящую многозадачность, если наш компьютер был оснащен процессором 80386. ПК больше не были ограничены запуском одной программы за один раз, а ОС Windows сделала взаимодействие с компьютером с помощью мыши общеупотребимым¹.

С VB все стало интересно. Вместо того чтобы кодировать интерфейс, вы его рисовали. IDE включала в себя палитру компонентов и окно. Вы могли рисовать кнопки, текстовые поля и все остальное, что нужно, прямо в окне. После того как вы их нарисовали, вы «скрепляли их вместе» с помощью хендлеров² событий. То, что вы нарисовали, появлялось при запуске программы. **Пользовательский ин-**

¹ Автор не упоминает альтернативные разработки – например, AmigaDOS/Workbench фирмы Commodore. – *Прим. ред.*

² Хендлеры – это функции, которые вызываются при получении определенного события от пользователя. – *Прим. ред.*

терфейс VB (UI) в конечном итоге был перенесен в Microsoft Visual Studio. Даже сегодня Visual Studio 2022 сохраняет те же функции, которые были столь революционными в 1991 году. На рис. 1.1 показан набор инструментов, используемый для рисования визуальных пользовательских интерфейсов для Windows:

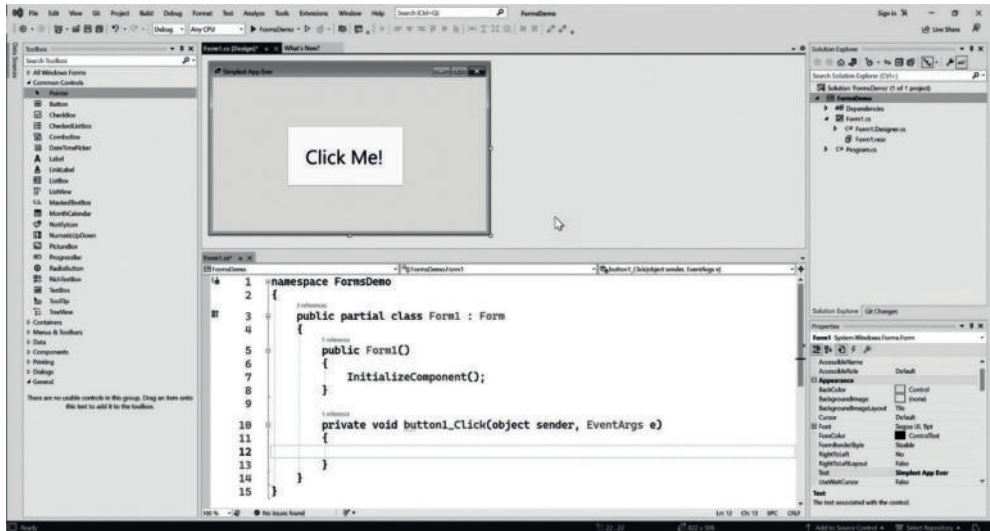


Рис. 1.1. Visual Studio IDE возникла как продукт под названием Visual Basic в 1991 году. Она определяла стандарты того, какой должна быть хорошая IDE

IDE VB3, с чего началась моя карьера, представила еще больше революционных функций, о которых мои самодовольные коллеги из Unix в брюках карго могли только мечтать. Они все еще боролись за превосходство VI над Emacs или наоборот, в зависимости от того, кого вы спрашивали. Между тем VB3 имел цветную подсветку синтаксиса, поддержку редактирования нескольких файлов, редактор графического интерфейса для рисования кнопок и других экранных виджетов, а также инструмент визуального программирования, который связывал вместе код, события и элементы графического интерфейса. У него была система отладки, которую можно было использовать, просто кликнув номер строки. Это приводит к созданию красной точки – точки останова в коде, где выполнение во время тестового запуска остановится, что позволит разработчику проверить состояние работающей программы. Это был чистый ботан-кодер! Нравится вам это или нет, но IDE Microsoft VB определили, какими должны быть IDE сегодня. Никто из тех, кто научился программировать с использованием Microsoft IDE, будь то устаревший язык или современный, не готов принять что-то меньшее, чем этот опыт.

С каждым языком, который я выучил с тех пор, первое, что я всегда делаю, – это нахожу самую лучшую доступную IDE, предлагающую те функции, без которых я не могу жить. Когда я начал работать с Python 3 около шести лет назад, я нашел PyCharm. Я использовал его, чтобы полностью переписать сложный продукт «**Программное обеспечение как услуга**» (SaaS), на завершение которого у меня ушло около 18 месяцев. Это было испытание огнем. В этой книге я намерен поделиться тем, что узнал, включая рассказы о полученных ожогах.

На протяжении всей этой книги мы узнаем об общем интерфейсе PyCharm IDE, а также о настройках и расширениях, которые помогут вам адаптировать ваши инструменты к типу работы, которую вы будете выполнять с Python. В этой первой главе обсуждаются достоинства IDE в целом. Я приведу сравнение наиболее распространенных инструментов, используемых для разработки на Python. Некоторые из них очень хороши, а другие, несмотря на широкое использование, достаточно примитивны.

В этой главе мы рассмотрим следующие основные темы:

- назначение PyCharm как среды разработки Python и некоторые примечательные подробности о компании-разработчике JetBrains;
- использование PyCharm внутри сообщества и список профессий, которые чаще всего используют PyCharm;
- подробное описание преимуществ и недостатков использования PyCharm по сравнению с другими редакторами IDE Python;
- различия между профессиональной и общественной редакциями PyCharm и дополнительными функциями, которые предлагает платная версия.

Однако, если вы уже решили, что PyCharm – это IDE Python для вас, смело переходите к главе 2 «Установка и настройка PyCharm», чтобы пройти процесс установки и регистрации. Если вы уже загрузили и успешно настроили PyCharm в своей системе, возможно, вам стоит начать со второго раздела этой книги, начиная с главы 3 «Настройка интерпретаторов и виртуальных сред».

ТЕХНИЧЕСКИЕ ТРЕБОВАНИЯ

Эта глава является вводной, поэтому мы пока не будем писать код, а технические требования равны нулю. Это глава 1, и я знаю, что вы все воодушевлены и готовы к работе, а ноль – это скучно. Итак, давайте двигаться!

Во-первых, вот что вам понадобится для успеха с этой книгой.

- Компьютер. Я знаю! Это очевидно, но я горжусь тем, что я последователен и не оставляю ничего на волю случая!
- ОС. Лучше всего это работает, если она уже установлена на вашем компьютере, поскольку в этой книге мы не будем рассказывать, как это сделать. Windows, macOS, Linux – для этой книги все одинаково, потому что PyCharm работает во всех трех ОС, а пользовательский интерфейс в каждой из этих сред практически идентичен.
- Инсталляция Python. В этой книге мы будем использовать исключительно Python 3. Существует несколько различных «разновидностей» Python 3, но по большей части подойдет старый добрый Python 3 с <https://www.python.org>. Мы вернемся к этим «разновидностям» позже, когда начнем говорить о виртуальных средах в главе 3 «Настройка интерпретаторов и виртуальные среды». Если вас утешают номера редакций, то на момент написания этой книги последней версией была 3.11.1. Версия Python, которую я использую в упомянутом ранее производственном SaaS-приложении, – 3.8. Если ваша установка Python 3 старше этой, вам следует обновить ее.
- В какой-то момент мне может пригодиться учетная запись **GitHub**, поскольку я буду делиться кодом из книги, используя репозиторий **Git**.

Продолжающийся успех Python

В первом издании этой книги автор назвал этот раздел «Недавний успех Python». Прошло время, и я продолжаю с того места, где он остановился. Я думаю, важно отметить, что *недавний* успех более или менее продолжился с момента публикации первого издания этой книги. Python продолжает оставаться одним из самых популярных и широко распространенных языков по нескольким очень веским причинам. Одна из этих причин заключается в том, что Python уделяет особое внимание читабельности и использует простой синтаксис. Это позволяет новичкам в языке, да и в области разработки программного обеспечения быстрый путь к успеху. Сравните это с ранее обычным опытом, когда студентов колледжей и университетов заставляли изучать C или C++ в качестве основного языка программирования. Эти языки кратки и сложны и, как правило, имеют плохую репутацию, когда дело касается производительности. Конечно, C и C++ – мощные языки, на которых можно создавать самое производительное программное обеспечение. Однако, по моему опыту, язык, который поможет вам перейти от «Hello, World» к созданию полезного программного обеспечения за короткий период времени, превосходит прирост производительности во всех случаях, кроме самых крайних. **Гвидо ван Россум**, создатель Python, сравнивает скорость Python с другими языками в своей статье «OMG-DARPA-MCC Семинар по композиционной архитектуре программного обеспечения». В статье ван Россум утверждает, что разработка на Python примерно в 3–5 раз быстрее, чем на Java, и в 5–10 раз быстрее, чем на C/C++. Помня об этой разнице, мы можем легко понять, почему Python так широко распространен. Ведь время – деньги. Вы можете найти полное эссе Гвидо ван Россума здесь: <http://www.python.org/doc/essays/omg-darpa-mcc-position/>.

Сравнение Python с Java или C/C++ некорректно, поскольку эти языки разработаны и используются для разных приложений. C и C++ используются, когда требуется очень высокая производительность. Большинство ОС написаны на C++, как и системы реального времени, подобные тем, которые можно найти в автомобиле Tesla или современном космическом корабле. Не всегда справедливо сравнивать производительность Python и C++, поскольку они не используются для создания приложений одного и того же типа.

С другой стороны, Java используется для разработки тех же типов приложений, для которых вы можете использовать Python: корпоративных и веб-приложений. Однако Java требует большого количества шаблонов (или «паттернов»). Это означает, что разработчик должен создать много страниц кода и структур только для поддержки существования приложения, прежде чем он сможет даже подумать о написании кода для самого приложения. Этот подход практически отсутствует в Python. Более того, Java опирается на очень жесткую статическую объектно ориентированную парадигму разработки¹. Python, напротив, гораздо более гибок, предлагая модель динамического программирования. Несмотря на то что эти два языка используются для создания приложений одного и того же типа, Python дает нам несколько серьезных сокращений благодаря своей более гибкой парадигме.

¹ Парадигма в программировании – это совокупность идей и понятий, определяющих стиль написания компьютерных программ (подход к программированию). – Прим. ред.

Эти факторы, составляющие сильные стороны Python, наряду со многими другими, были объединены, чтобы сформировать очень доступный язык разработки, поддерживаемый сообществом восторженных поклонников. Это сообщество продолжает расти, внедряя программирование в целый ряд областей и профессий, отличных от тех из нас, кто исторически фокусировался исключительно на разработке традиционных приложений. На момент написания книги **TIOBE**, система ранжирования популярности языков программирования, оценивает Python как язык номер один, что показано на рис. 1.2:

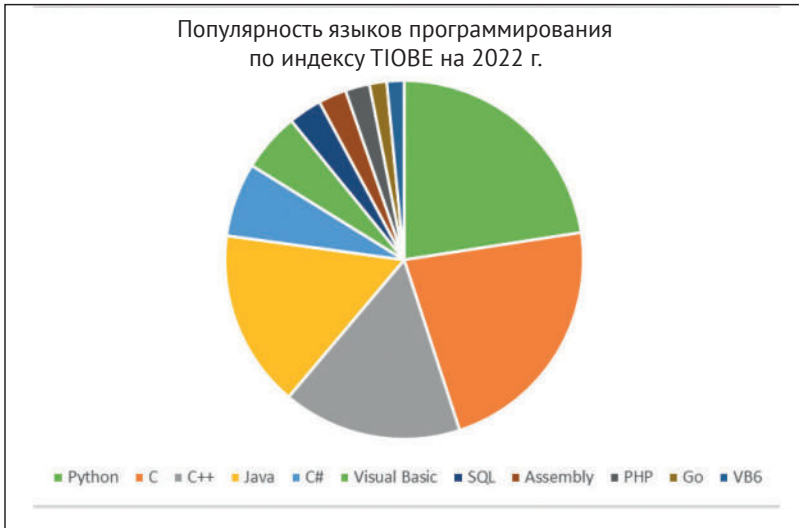


Рис. 1.2. Рейтинги TIOBE показывают, что Python является самым популярным языком программирования

Python имеет огромную стандартную библиотеку, предоставляющую все, что вам может понадобиться для создания любого программного обеспечения, которое вы только можете себе представить. Если это утверждение окажется ложным для вашего конкретного проекта, существует обширная сторонняя экосистема с открытым исходным кодом, состоящая из сотен тысяч библиотек, на которых вы можете строить свой проект. Каталог этих библиотек можно найти на <https://pypi.org>. Объединив все это, новый разработчик программного обеспечения может очень быстро перейти от идеи и нулевого опыта работы с Python к промышленному приложению. Этот процесс можно значительно ускорить с помощью хорошей IDE.

Философия IDE

Когда я был в вашем возрасте, все было по-другому. Это, конечно, если мы не одного возраста, в этом случае все было одинаково. У нас не было интернета. Когда мы хотели изучить новые языки и методы программирования или понять историю нашего ремесла, нам приходилось совершать священное паломничество, и я тайно провез с собой Полароид. Вы можете увидеть фотографии, которые я сделал, на рис. 1.3. Вы должны понимать, что все, что я собираюсь

рассказать, является одновременно правдой и тщательно охраняемой отраслевой тайной. Чтобы прояснить ситуацию: вы слышали это не от меня.

Спрятанные где-то в мистических горах, искатели великой мудрости кодирования поднимались по 10 000 ступеням при свете полной луны в поисках Мастера. Путешествие было нелегким, переданную мудрость пришлось заслужить тяжелым трудом. Именно во время одного из таких крестовых походов я понял, почему хорошие IDE так важны. Мастер сказал: «Если вы знаете язык и знаете IDE, вам не нужно бояться результата сотни развертываний».



Рис. 1.3. Высоко в священных горах, на высоте 10 000 ступеней, находится монастырь, где я научился программировать

Мастер часто говорил загадками, поэтому позвольте мне объяснить. *Развертывание* относится к опубликованной итерации или **приращению** вашего программного обеспечения. В большинстве профессиональных случаев целью является публикация вашего программного обеспечения. Если нашей целью является публикация, следующим камнем преткновения будет то, что мы должны знать язык программирования. Я предполагаю, что у вас есть хотя бы некоторое понимание программирования на Python. Это намек на учение Мастера об IDE.

Существует несколько классов инструментов, которые разработчик может использовать для разработки кода Python. Язык Python можно считать интерпретируемым языком. Мы могли бы утверждать, что при запуске некоторая часть кода оптимизируется в код C и кешируется, но на данном этапе нас не беспокоит такой уровень детализации. Дело в том, что программа Python существует в виде простых текстовых файлов и может быть выполнена в такой форме. Сравните это со статически скомпилированными языками, такими как C, C++, C#, Java или Go. Эти и многие другие языки требуют, чтобы код в текстовых файлах прошел этап компиляции, на котором создается новый исполняемый файл. В C# вы не можете просто выполнить файл .cs. Вам нуж-

но скомпилировать его в двоичный файл, а затем выполнить его. Поскольку Python выполняет свой код непосредственно через интерпретатор Python, уровень инструментов, необходимых для работы на Python, может быть очень простым. По сути, подойдет любой текстовый редактор. На выбор доступны три уровня возможностей редактора.

Первый – простой текстовый редактор. Простые текстовые редакторы обычно ограничиваются открытием, редактированием и сохранением текста. Это универсальные инструменты, предназначенные для работы с любыми текстовыми файлами, от списков покупок до конфигураций `systemd`. В Windows вы можете знать его как *Notepad* (Блокнот). На Mac вы можете использовать *TextPad*, а если вы используете рабочий стол Linux, например **Ubuntu**, вы легко найдете *Text Editor*. Если вы не являетесь поклонником графических интерфейсов в своей ОС, то вы, несомненно, слышали о таких редакторах, как *vi*, *vim* (улучшенный *vi*), *Emacs* и *nano*. Все эти программы относятся к категории простых текстовых редакторов. Если вы не уверены, что такое конфигурация `systemd`, не переживайте; это файл системного администрирования в Linux. Мне просто нужно было что-то, что звучало бы сложно, чтобы охарактеризовать более сложную часть диапазона текстовых файлов.

Вторая эволюция программных редакторов называется *расширенными редакторами*. Эти редакторы специально созданы для работы с техническими файлами. Некоторые популярные примеры включают следующее:

- Visual Studio Code,
- Atom,
- Notepad++,
- UltraEdit,
- Sublime Text,
- JetBrains Fleet,
- Bluefish Editor,
- IDLE (редактор, поставляемый с Python)¹.

Эти инструменты предназначены для работы с широким спектром языков программирования, и, как правило, их можно легко настроить для добавления поддержки новых языков. Расширенные редакторы предлагают некоторые общие функции, которые делают жизнь разработчика немного приятнее. Это:

- подсветка синтаксиса: цветовая кодировка ключевых слов и других семантических элементов вашего кода;
- макросы, которые позволяют разработчику записывать и воспроизводить стандартные нажатия клавиш;
- организация проектов и файлов, позволяющая легко переключаться между несколькими файлами, составляющими проект;
- элементарное завершение кода для уменьшения количества ввода, необходимого для написания кода;
- поддержка плагинов для других тонкостей, таких как линтеры, проверка орфографии, предварительный просмотр файлов вашего кода и многое другое.

¹ Здесь явный юмор разработчиков Python – `idle` переводится как «бездельник». – Прим. ред.

Со временем некоторые из этих расширенных редакторов стали очень надежными, поскольку их возможности можно настраивать и расширять. Если рассматривать эти инструменты в том виде, в котором они есть, прямо из коробки, они более полезны и специализированы, чем обычные текстовые редакторы, но они не соответствуют требованиям IDE.

На вершине пищевой цепочки редактора кода находится **IDE**. Если бы вы заглянули в кабину истребителя времен Первой мировой войны, вы бы увидели несколько простых приборов, элементов управления и ничего более. Если это текстовый редактор, то IDE – это кабина Боинга-747. Каждый инструмент, который когда-либо мог понадобиться разработчику или который он только мог пожелать, втиснут в сравнительно сложный пользовательский интерфейс. IDE содержат все функции расширенного текстового редактора, но также предлагаются следующие дополнительные улучшения:

- несколько простых способов запустить код прямо из редактора;
- инструменты для управления репозиторием исходного кода, например **Git** или **Subversion**;
- интегрированный, простой в использовании отладчик, который позволяет приостанавливать выполнение работающей программы и проверять или изменять ее текущее состояние;
- инструменты, которые помогут вам писать автоматические тесты, такие как модульные тесты, а также запускать программу и визуализировать результаты;
- комплексное завершение кода основано на самоанализе или индексировании кода вашего проекта. В современных IDE это усиливается с помощью **искусственного интеллекта (AI)**;
- инструменты профилирования, которые помогут вам найти узкие места при выполнении;
- интегрированные инструменты для помощи с дополнительными системами, такими как базы данных;
- инструменты для развертывания вашего кода на сервере или в облачной среде прямо из IDE.

Некоторые популярные примеры IDE включают следующее:

- Visual Studio (это отличается от Visual Studio Code),
- PyCharm,
- IntelliJ IDEA,
- NetBeans,
- Apple Xcode,
- Xamarin Studio,
- Eclipse.

Как видите, IDE – самое мощное оружие в вашем арсенале кодировщика. Важно использовать лучший из доступных вам IDE. Если вы новичок в разработке программного обеспечения или, возможно, даже не новичок, вы можете задаться вопросом, почему расширенные редакторы так популярны. На момент написания примерно 50 % разработчиков используют Visual Studio Code, которого нет в моем списке IDE.

Многие разработчики предпочитают «облегченную» среду разработки. Это особенно актуально для веб-разработчиков внешнего интерфейса, которые

доверяют Sublime Text и Visual Studio Code. По правде говоря, им нужны все возможности IDE, и они их используют, но они распределены по различным инструментам, которые они используют в течение рабочего дня. Разработчик внешнего интерфейса полагается на профилировщики и отладчики, работающие в веб-браузерах, и им не нужны эти инструменты в IDE. Вместо этого они могут получить более простой редактор, который быстро загружается, легко устанавливается и запускается мгновенно, когда они кликают значок в своей ОС.

Я утверждаю, что, если вы занимаетесь полнофункциональной веб-разработкой или мобильной разработкой или вам нужно работать с серверами или контейнерами, IDE – лучший выбор.

Существует определенный класс разработчиков программного обеспечения, которые клянутся, что вы никогда не должны использовать ничего, кроме самых простых инструментов. Они считают, что использование инструмента для кодирования и связанной с ним тяжелой работы снижает общее мастерство и уровень достигнутых результатов, необходимые для того, чтобы считаться профессионалом. Я не могу с этим не согласиться. Однажды, находясь в монастыре, Учитель рассказал мне историю великого мастера меча из Японии по имени Миямото Мусаси. В его времена каждый самурай знал Мусаси как величайшего из ныне живущих фехтовальщиков, и все самураи хотели победить его. В те времена поединки обычно велись насмерть. Однажды соперник встретил Мусаси, когда тот выходил из лодки. Мусаси был безоружен. Претендент ждал, пока Мусаси сможет вытесать из одного из весел лодки деревянный меч, называемый *боккен*, который он намеревался использовать в поединке. Легенда гласит, что Мусаси одурачил этого претендента и оставил его в живых к большому позору претендента. Мусаси, по словам Мастера, был лучшим воином, когда-либо жившим на свете, и с тех пор его мастерство владения мечом не имело себе равных. Однако, если бы целью было просто победить его, я бы легко сделал это, расстреляв из автомата.

По моему мнению, глупо ограничивать инструменты, которые вы используете, из-за чувства гордости за свои возможности. Целью разработчика программного обеспечения является поставка программного обеспечения, обычно в неумолимые сроки. Это не предполагает трату времени на попытки доказать, что вы соответствуете чьим-то стандартам, если, конечно, вы не студент, соискатель степени. Я уверен, что некоторые из вас, читающих эту книгу, именно таковы. Играйте в игру и делайте то, что говорят ваши профессора. Вы должны понимать, что, как только вы закончите учебу, все изменится. От вас ожидают, что вы будете создавать код быстро, точно и последовательно. Лучше всего этого достичь с помощью автоматизации, доступной в хорошей IDE. Вам следует выбрать лучший инструмент для выполняемой работы. Я обнаружил, что PyCharm помог мне повысить продуктивность во время изучения языка Python. Когда вы начинаете и не используете редактор, исправляющий межстрочный интервал и отступы, вы наделаете много глупых ошибок. Это расстраивает. Я думал про себя: «Если бы я использовал C#, я бы уже закончил». У меня даже возникло искушение отказаться от Python и PyCharm в пользу чего-то более удобного. Однако это не то, что я хотел сделать.

PyCharm подчеркнет за вас все эти глупые ошибки и исправит их одним нажатием кнопки! После того как я увидел эти ошибки, подчеркнутые снова и снова, я понял, что делать, когда я использую редактор без проверки кода. Се-

годня, работая с другими языками, я по-прежнему использую правила Python. Изучив Python с помощью PyCharm, я смог быстрее выпускать, быстрее учиться и улучшать свой код на других языках и в других инструментах. Сделайте мне одолжение и никогда не позволяйте никому говорить вам, что вы не *настоящий разработчик*, потому что вы сделали что-то не так, как они. Если они будут упорствовать, скажите им, что nano лучше, чем vi или Emacs, и просто уходите. Такое заявление, вероятно, заставит их взорваться.

Я хотел бы сделать еще один комментарий по поводу Visual Studio Code. Этот редактор развился за счет плагинов до такой степени, что может конкурировать с полнофункциональной IDE. Однако за это приходится платить, если сравнивать с профессионально разработанной IDE, такой как PyCharm. Чтобы получить те же функции, что и в PyCharm, вам потребуется установить большое количество плагинов. Все эти плагины написаны сообществом, а это означает, что все они являются независимыми разработками. Эти плагины никогда не будут работать так слаженно, как базовые функции, которые вы найдете в PyCharm. Это также верно при сравнении Visual Studio с Visual Studio Code. Попробуйте создать проект C# в Visual Studio и Visual Studio Code, и вы обнаружите, что в Visual Studio этот процесс чудный, плавный. Visual Studio Code, с другой стороны, потребует большого количества работы в командной строке и установки множества странных плагинов. Опыт работы совершенно разный. То же самое можно сказать и о других редакторах, таких как vim, которые можно настраивать. Вы потратите неделю на возню с плагинами и скриптами с открытым исходным кодом, чтобы в лучшем случае добиться частичного соответствия готовым функциям IDE.

PYCHARM КАК ИНТЕГРИРОВАННАЯ СРЕДА РАЗРАБОТКИ PYTHON

Хорошо говорить о сравнении инструментов, обычном для других языков. Но нас это не волнует, не так ли? Мы хотим знать наши возможности для Python! Лучшие IDE обычно специализированы. PyCharm специализируется на работе с Python. Вот и все. Создавая новый проект в PyCharm, вы увидите варианты проектов Python, и ничего больше. Сравните этот опыт с Visual Studio. По моему мнению, Visual Studio – единственный близкий конкурент PyCharm, когда дело доходит до работы с проектами Python. Когда проект создается в Visual Studio, вы, скорее всего, потратите добрых пять минут, пытаясь разобраться в множестве вариантов. IDE поддерживает десятки языков, и это усугубляется дюжиной типов проектов, таких как веб-сайты, десктопы и др. Visual Studio пытается быть всем для всех разработчиков. PyCharm хочет играть только с разработчиками Python.

Сам по себе PyCharm был создан с учетом нескольких целей разработки.

- Интеллектуальная помощь в кодировании.
- Оптимизированные инструменты программирования.
- Варианты веб-разработки.
- Поддержка научных вычислений.
- Визуальная отладка и профилирование производительности.

Мы рассмотрим каждую из этих целей разработки по очереди, но сначала мне нужно кое-что отметить. На момент написания этой статьи PyCharm вот-вот претерпит большие изменения. JetBrains работает над совершенно новым пользовательским интерфейсом. К моменту публикации этой книги велика вероятность,

что он станет стандартом. Если вы новичок в PyCharm, то должны понимать, что какое-то время вы будете видеть его двумя разными способами. Классический пользовательский интерфейс будет доступен в продукте какое-то время, что позволит нам легко освоить новый интерфейс. Я решил, что воспользуюсь новым пользовательским интерфейсом, учитывая, что между первым изданием и этим прошло несколько лет. Тем не менее стоит отметить, что вы увидите классический пользовательский интерфейс наряду с новым, вероятно, до конца 2024 года, когда старый перестанет поддерживаться. Он устареет и однажды исчезнет в песках времени, как легендарная статуя Озимандиаса¹ в сонете Перси Биши Шелли.



Рис. 1.4. Новый пользовательский интерфейс (вверху) по сравнению с классическим пользовательским интерфейсом (внизу)

¹ В древности Озимандиас было греческим именем фараона Рамсеса II. – Прим. ред.

На рис. 1.4 показаны два пользовательских интерфейса. Целью разработки нового пользовательского интерфейса является уменьшение беспорядка. В этом разработчики не ошиблись. По мере того как инструмент с годами развивался, в меню добавлялось все больше и больше функций, что сделало пользовательский интерфейс несколько сложным для новых пользователей. Самое главное, что нужно понимать, – это то, что большинство вещей, которые содержались в меню, остались, но сама система меню скрыта под значком гамбургера в верхнем левом углу экрана. Не волнуйтесь – я расскажу об этом подробно позже. Пока я пишу это, есть параметр, который мы рассмотрим в главе 2 «Установка и настройка PyCharm», он позволяет переключаться между классическим и новым пользовательским интерфейсом.

Я хотел указать на это сейчас, потому что вы скоро увидите несколько скриншотов, и если вы видели старый пользовательский интерфейс, вы можете подумать, что взяли не ту книгу. Это не так. Как раз наоборот. Если я правильно рассчитал время, вы будете единственным программистом, у кого будет правильная книга.

Интеллектуальная помощь в кодировании

Я собираюсь сказать вам то, что моя жена постоянно говорит. Лени – мой недостаток. Подождите. Вышло неправильно. Она говорит, что это я ленивый. Я не говорю, что она говорит, что она ленивая. Ох. Как это объяснить! Теперь мне конец, да?

Она не ошибается. Как разработчик, я, по сути, очень ленив. Я отказываюсь тратить часы или даже минуты на то, чтобы сделать что-то, требующее долгого времени. У греков существовала легенда о парне по имени Сизиф, который был проклят и потому должен был закатывать камень на крутой холм. Как только он достигал вершины, камень катился обратно с холма, вниз. Сизиф застрял в бесконечном цикле, так как на его клавиатуре не было команды `Cmd/Ctrl + C`.

Я знаю одно: Сизиф не был разработчиком программного обеспечения! Любой разработчик программного обеспечения закатил бы этот камень ровно дважды, после чего потратил бы вечность на разработку системы шкивов и кранов, управляемых устройством IoT¹. Естественно, микроконтроллер будет запускать скрипт Python. Я отвлекся.

То, что *некоторые* (говорю я про себя, глядя в сторону ничего не подозревающей жены) могут назвать ленью, я называю *эффективностью*. Как разработчик, я хочу добиться максимального эффекта с минимальными усилиями во всем, что делаю. Написание кода – это сложно. Вы пишете инструкции для самого упрямого и неразумного объекта, когда-либо созданного. Программировать хуже, чем пытаться научить двухлетнего ребенка завязывать шнурки. Поверьте мне, я делал и то и другое! Программисты должны быть предельно конкретными и многословными в объяснении любых операций, которые они хотят выполнить. Более того, ситуацию усугубляют разработчики языка,

¹ Интернет вещей (Internet of Things, IoT) – это множество физических объектов, подключенных к интернету и обменивающихся данными. Концепция IoT может существенно улучшить многие сферы нашей жизни и помочь нам в создании более удобного, умного и безопасного мира. Примеры Интернета вещей варьируются от носимых вещей, таких как умные часы, до умного дома, который умеет, например, контролировать и автоматически менять степень освещения и отопления. – Прим. ред.

которые стремятся заставить пользователей писать кучу шаблонного кода. Я говорю о бесполезном коде, не имеющем ничего общего с кодом, который вы хотите написать, или с задачей, которую хотите решить. Python обычно избегает этого, поэтому позвольте мне привести пример, возможно, худшего нарушителя: Java.

Когда я был маленьким мальчиком, Java была в моде. Была каста корпоративных программистов, занимавшихся Java, которые придумали что-то под названием **Enterprise Java Beans (EJB)**¹. EJB должны были стать воплощением модульного программирования с повторно используемыми объектами. Это была настоящая катастрофа при реализации. Вместо того чтобы просто создавать класс (а это все, что вам нужно), вам нужно было создать специальную файловую структуру с различными папками и файлами манифеста², чтобы раскрыть то, что было в файле BEAN³, и все это было скомпилировано в специальный формат. Оказалось, что специальный формат – это не что иное, как ZIP-файл. Создание EJB потребовало много работы, а это означало, что разработчикам приходилось создавать массу файлов и писать много кода только для того, чтобы приступить к работе над функциональностью, которую им нужно было выразить для выполнения своей работы. Вот что мы подразумеваем под *шаблоном*⁴. Шаблон обычно бесполезен, но необходим, потому что без него код не работает.

Все IDE развивались благодаря этому феномену. PyCharm произошел от Java IDE JetBrains IntelliJ. В Python обычно не так уж много шаблонов, необходимых для работы вашего кода, но они все же есть. Существует два вида шаблонов. Шаблон, необходимый для работы EJB старой школы, является плохим. Шаблон, созданный как средство для запуска вашего проекта, является хорошим вариантом. Как мы увидим, PyCharm, как и большинство IDE, генерирует структуру папок, набор файлов и некоторый базовый код для начала работы. Это можно считать шаблоном. Но в данном случае этот код не сохраняется. Он заменяется реальным кодом вашего проекта. Код, сгенерированный IDE, – это всего лишь подсказка, которая поможет вам приступить к работе. Это избавит вас от необходимости создавать отправную точку вашего проекта вручную.

Все это здорово, но генерация шаблонного кода – это не то, о чем мы обычно думаем, когда слышим «интеллектуальная помощь в кодировании». Обычно мы думаем о функции **IntelliSense**, впервые разработанной Microsoft. Если вы позволите мне на мгновение антропоморфизировать IDE, эта функция наблю-

¹ Enterprise Java Beans – спецификация технологии написания и поддержки серверных компонентов, содержащих бизнес-логику. – *Прим. ред.*

² Файл манифеста – это файл, содержащий метаданные для группы сопутствующих файлов, которые являются частью набора или согласованного блока. – *Прим. ред.*

³ Bean-файл – это обычный файл класса Java, реализующий бизнес-интерфейс EJB. – *Прим. ред.*

⁴ Шаблон (или «паттерн») – нетворческий программный код, который программисту приходится писать вследствие требований языка программирования, операционной системы, библиотеки подпрограмм, манеры программирования и прочего. Название «шаблон» говорит, что он повторяется из функции в функцию, из программы в программу с минимальными изменениями. – *Прим. ред.*

дает за тем, как вы вводите свой код. Все это время IDE думает о том, что вы пытаетесь сделать. Когда она видит способ помочь, например автоматически дополнив за вас слово или строку, она представляет это как вариант. У меня есть умный человек, который завершает за меня все предложения, – это моя жена. Когда она заканчивает за меня предложения, они обычно более организованны и умны, чем они были бы, если бы я был один. (Возможно, это еще одна причина, по которой она думает, что я ленив.)

Хочу отметить, что не все инструменты с функцией IntelliSense созданы одинаково. Когда вы видите эту функцию в расширенном редакторе, она обычно работает иначе, чем в IDE. В расширенных редакторах списки ключевых слов используются для выделения и автозаполнения элементов языка. Действительно хорошие расширенные редакторы могут индексировать ваш код, распознавать имена переменных и функций и использовать статистику, чтобы попробовать дать вам наиболее вероятное завершение. За этим вариантом обычно следует длинный список помех, включающий все возможные варианты данного завершения. Завершение кода становится все более продвинутым с появлением инструментов искусственного интеллекта, и это делает разницу между IDE и расширенными редакторами немного более запутанной, по крайней мере в этом отношении. Такие инструменты, как Copilot от GitHub, могут не только автоматически заполнять имена переменных и ключевые слова, но также автоматически заполнять целые разделы вашего кода.

Важно помнить, по крайней мере, сейчас, когда я пишу это, что эти функции AI не являются частью IDE или расширенного редактора. Они реализованы в виде плагинов. Поскольку это правда, я продолжу отстаивать достоинства IDE и PyCharm в частности, основываясь исключительно на достоинствах самого программного обеспечения. Мы обсудим плагины в главе 16.

Хотя расширенные редакторы могут предоставить вам длинный список возможностей для завершения кода, PyCharm может анализировать ваш код и выполнять более интеллектуальное автодополнение. Вы также получаете анализ кода, например предупреждения о дублировании кода. Распространенным антипаттерном (антишаблоном) в разработке программного обеспечения является копирование и вставка кода в один и тот же проект или даже в сам этот код. Это ужасно, но это обычное явление. PyCharm обнаружит дублированный код и пометит его, чтобы вы могли напомнить о необходимости реорганизации дублированного кода в функцию или модуль, которые можно будет повторно использовать и поддерживать в одном месте.

PyCharm также может выполнять статический анализ вашего кода. Этот тип анализа ищет антипаттерны внутри самого кода; например, PyCharm обнаружит мертвый код, как показано на рис. 1.5. Что касается разработки Python, PyCharm автоматически отформатирует ваши отступы и предоставит вам критические отзывы о том, насколько ваш код соответствует соглашениям **PEP-8**, представляющим собой стилистические требования, которым вы должны соответствовать, чтобы код мог считаться *питоническим* (и это хорошо).

Например, если вы введете следующий код в новый файл в PyCharm, то увидите предупреждение PyCharm о том, что вы создали недоступный код в строке 13. Текст в этой строке будет выделен. Наведя указатель мыши на эту выделенную строку, вы увидите, что не так:

```
def print_hi(name):
    print(f'Hi, {name}')
    for x in range(25):
        print(str(x))
        if x == 12:
            return
        print("You'll never make it here")
if __name__ == '__main__':
    print_hi('PyCharm')
```

Функция `print_hi` запускается достаточно безобидно, выдавая на консоль все, что передается в функцию в аргументе `name`. После этого мы создаем цикл, который будет выполняться 25 раз. При каждом запуске цикла мы выводим `x`, который содержит текущую итерацию. Когда переменная-счетчик `x` достигает 12, цикл завершается через функцию `return`, которая, как назло, находится в строке 12. Уверяю вас, это чистое совпадение. Поскольку цикл возвращается в строке 12, код в строке 13 никогда не будет достигнут:

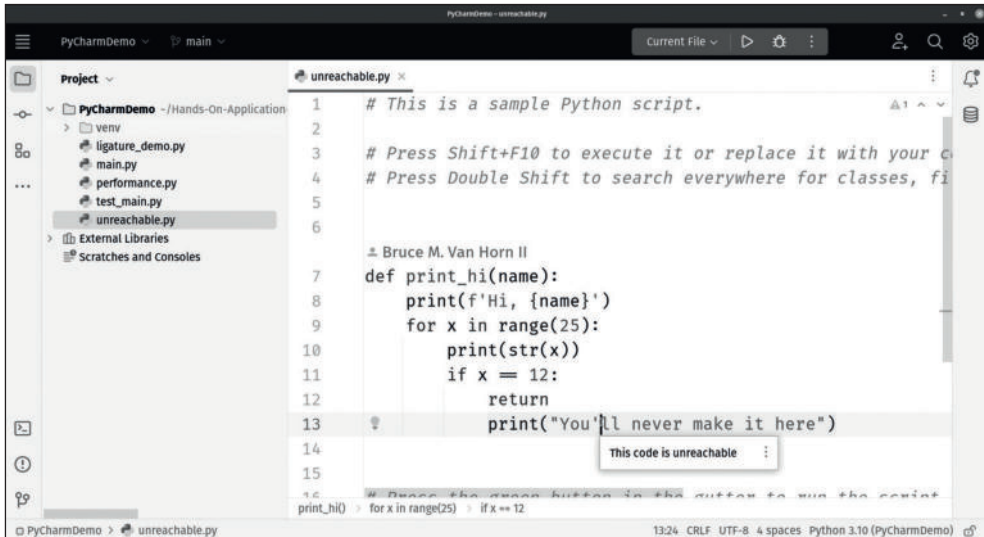


Рис. 1.5. PyCharm выделит многие распространенные ошибки кодирования, подобные этой. Код в строке 13 недоступен, на что указывает наведение курсора мыши на выделенный код

PyCharm также позволяет вам перемещаться между файлами в сложном проекте, помогая вам найти, где определены функции, переменные и классы, а также где они используются. Со временем вы выучите набор нажатий клавиш, которые позволяют вам перемещаться в любое место вашего проекта, не отрывая пальцев от клавиатуры.

По сути, интеллектуальная помощь при кодировании PyCharm позволяет вам меньше беспокоиться об ошибках и больше о вашей логике и о том, что вы хотите реализовать, и это позволяет быстрее завершить свой код с меньшим количеством ошибок.

Оптимизированные инструменты программирования

Написание кода – это всего лишь одно из действий, которое разработчик выполняет каждый день, чтобы уложиться в сроки проекта. Хорошие разработчики также тратят время на отладку, тестирование и профилирование своих продуктов, чтобы добиться наилучшего результата. Нам также нужно заняться отправкой кода на серверы тестирования, рефакторингом плохого кода (посторонних разработчиков), работой с базами данных и контейнерами. В PyCharm есть инструменты для каждого из этих процессов и многого другого. Когда я пишу сложные веб-приложения в PyCharm, единственные инструменты, которые у меня обычно открыты, – это PyCharm и веб-браузер: два инструмента, каждый на своем мониторе.

Отладчик PyCharm

Моя любимая функция, которая взволновала меня при первом использовании PyCharm, – это отладчик (дебаггер). Отладчик PyCharm великолепен. Он намного лучше, чем стандартный отладчик, который поставляется с самим Python. Python поставляется с отладчиком **Python Debugger (pdb)**. По моему скромному мнению, лучше я буду есть жуков, ползающих по тротуару, чем использовать этот инструмент¹. Я уже упоминал об этом ранее в этой главе. Я вырос на отладчиках Microsoft и больше ничего не могу представить. Отладчик PyCharm работает именно так, как я и ожидал. Кликните мышью на строке, где вы хотите остановить выполнение, чтобы создать точку останова, и нажмите кнопку отладки в IDE, и программа запустится и остановится на указанной строке. Вы получите экран, на котором сможете проверить как состояние стека, так и вывод терминала. Его очень просто использовать, и я покажу вам, как это сделать, в главе 6 «Бесшовное тестирование, отладка и профилирование».

Графический интерфейс тестирования

Инструменты тестирования интегрированы в виде средств запуска тестов. PyCharm поддерживает все основные среды тестирования, включая **pytest**, **nose** и обычные функции модульного тестирования из стандартной библиотеки. Опять же, я имел дело с некоторыми очень хорошими IDE, и в данном случае я вспоминаю Eclipse и Visual Studio, которые включают в себя средства запуска графических тестов. Пословица «Если полоска зеленая, код чист» визуально реализована в PyCharm. Пример вы можете увидеть на рис. 1.6. Вы можете запустить тесты и увидеть список, показывающий, что прошло успешно, а что нет, хотя это скорее список, а не полоса. Затем можете повторно запускать неудачные тесты, пока они не заработают.

Я приведу простой пример. В файле `main.py` исходного кода этой главы есть один файл с именем `main.py` и другой с именем `test_main.py`. Содержимое `main.py` представляет собой простую функцию, которая складывает два числа:

```
def add_two_numbers(a: int, b: int) -> int:
    return a + b
```

¹ Баг (англ. *bug*, жук) – это «ошибка в коде» на арго программистов, так что «debugger» можно перевести как «обезжучиватель». – Прим. ред.

В файле `test_main.py` есть простой модульный тест:

```
from unittest import TestCase
from main import add_two_numbers

class Test(TestCase):
    def test_add_two_numbers(self):
        self.assertTrue(add_two_numbers(a=5, b=6) == 11, \
                        "Should be 11")

    def test_show_a_fail(self):
        self.fail()
```

Класс `Test` содержит два теста: один пройдет успешно, а другой автоматически завершится неудачно. Обычно я сначала провожу автоматически провальный тест, просто чтобы убедиться, что мой тестовый класс настроен правильно. Затем, позже, я удаляю неудачу из-за моей дофаминовой зависимости, которая удовлетворяется только зелеными галочками в программе запуска тестов, как показано на рис. 1.6. Если я кликну правой кнопкой мыши по `test_main.py`, как показано на рис. 1.5, то получу возможность запустить тесты внутри файла:

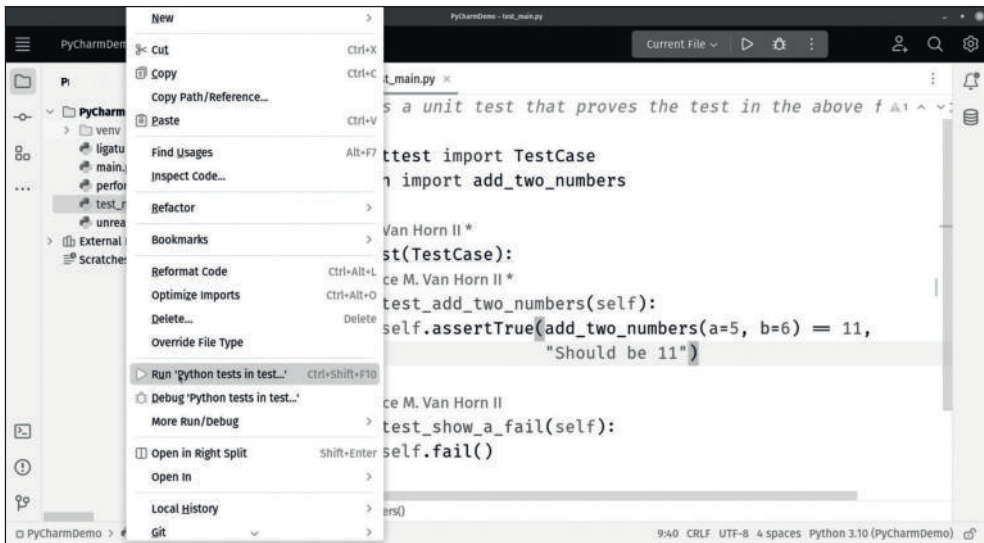


Рис. 1.6. Кликните правой кнопкой мыши файл `test_main.py` и выберите `Run 'Python tests in test...'`, чтобы запустить модульные тесты, содержащиеся в файле

Посмотрите на нижний левый угол рис. 1.7, где показано завершение тестового прогона, и вы увидите список тестов, которые прошли или не прошли, либо с зеленой галочкой, либо с желтым крестиком, обозначающим неудачу. Как и все рисунки в этой книге, которые напечатаны в черно-белом цвете, вы не увидите цвета. Цветные чернила дороги, и ваш отец прав, деньги на деревьях не растут:

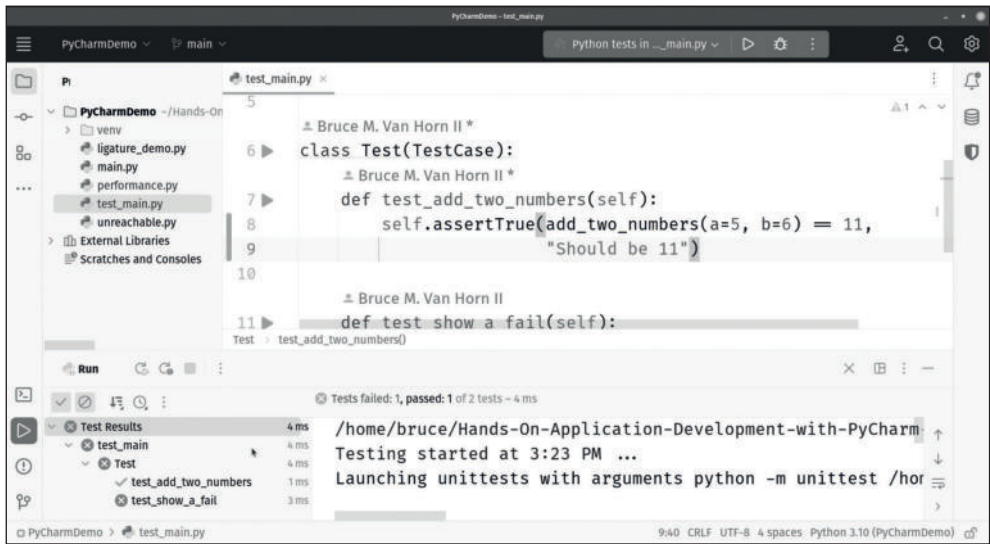


Рис. 1.7. Встроенный инструмент запуска тестов PyCharm отображает традиционный список пройденных/непройденных тестов (нижняя левая панель) для обозначения пройденных и непройденных тестов

Инструменты профилирования PyCharm

Аналогично профилирование кода встроено и просто в использовании. Вы можете нажать кнопку **Profile**, чтобы запустить код. Когда программа завершит работу, вы получите график каждого вызова функции, а также время и ресурсы, затраченные на вызов. Это позволяет легко выявить нереализованные возможности для улучшения скорости выполнения и потребления ресурсов.

Учтите возможность того, что в вашей программе есть алгоритм, который может работать не так, как вам хотелось бы. Я знаю, знаю, с вами такого никогда бы не случилось, так что предположим, что вас только что наняли, и человек, которого они уволили, написал этот ужасно эффективный алгоритм. Может быть, представим, что на дворе 1956 год, и парнем, которого уволил ваш новый работодатель, Нью-Йоркская компания по страхованию жизни, был некий **Эдвард Гарри Френд**. Френд написал статью под названием «Сортировка в электронных компьютерных системах», вероятно, являющуюся первым опубликованным примером алгоритма, который мы сегодня знаем как **сортировка пузырьком (bubble sort)**¹. Если бы Френд написал свой алгоритм на Python 3, он мог бы выглядеть примерно так:

```
def bubble_sort(input_array):
    length_of_array = len(input_array)
```

¹ Сортировка пузырьком – один из самых известных алгоритмов сортировки. Здесь нужно последовательно сравнивать значения соседних элементов и менять их местами, если предыдущее оказывается больше последующего. Таким образом, элементы с большими значениями оказываются в конце списка, а с меньшими остаются в начале. – *Прим. ред.*

Френд только что создал функцию, принимающую в качестве аргумента список, который в нашем случае будет массивом целых чисел. Цель состоит в том, чтобы отсортировать эти числа. Для этого создадим два цикла, один внутри другого:

```
for i in range(length_of_array):
    for j in range(0, length_of_array - i - 1):
        if input_array[j] > input_array[j + 1]:
            input_array[j], input_array[j + 1] = \
                input_array[j + 1], input_array[j]
```

Внутри этих циклов каждое число сравнивается с числом перед ним. Если будет установлено, что эти два числа не в порядке, они меняются местами. При следующем запуске цикла это происходит снова со следующими двумя числами, и так продолжается до тех пор, пока не будет достигнут конец списка.

Если вы вообще изучали алгоритмы, вы, вероятно, слышали о сортировке пузырьком и вас предупреждали, почему она не используется. Это очень медленно. У нас есть цикл `for` внутри другого цикла `for`, и это нормально, если размер вашего несортированного списка невелик. Но этот алгоритм замедляется с логарифмической скоростью по мере роста списка чисел. Производительность алгоритма измеряется с использованием обозначения «**O**» **большое**¹. Я не хочу превращать это в книгу по алгоритмам, поэтому просто скажу вам, что цикл внутри другого цикла плохо масштабируется с точки зрения производительности. В обозначении «**O**» большого мы классифицируем этот алгоритм как $O(n^2)$. Это плохо.

На простом языке это означает, что если вы удвоите количество чисел для сортировки (n), то обработка вашего алгоритма займет в 2^2 (или 4) раза больше времени. Если вы умножите размер вашего списка на 5, то он станет в 5^2 (или 25) раз медленнее. Чем больше список, тем медленнее скорость его сортировки.

Чтобы продемонстрировать инструмент производительности, мы собираемся дать этому тесту список из 100 000 чисел для сортировки. Сейчас самое время отметить, что я использую процессор Intel i9. Если вы студент или другой потребитель с ограниченным бюджетом, использующий процессор i3 (или того хуже), возможно, вам стоит сократить список чисел на несколько нулей, если вы хотите попробовать это. На моем i9 это занимает много времени:

```
test_array = []
for x in range(100000):
    test_array.append(random.randint(1, 10000))
```

Давайте закончим тестовый код, вызвав функцию и распечатав результаты:

¹ «**O**» большое – математическое обозначение для сравнения асимптотического поведения функций. – *Прим. ред.*

```
bubble_sort(test_array)
print("The result of the sort is:")
for i in range(len(test_array)):
    print(test_array[i])
```

Мы подробно рассмотрим инструмент профилирования в главе 6 «Бесшовное тестирование, отладка и профилирование», а пока давайте просто запустим этот код с помощью профилировщика и посмотрим результат. Чтобы запустить профилировщик для файла `performance.py`, просто кликните файл правой кнопкой мыши и выберите **More Run/Debug**, затем **Profile 'performance'**, как показано на рис. 1.8:

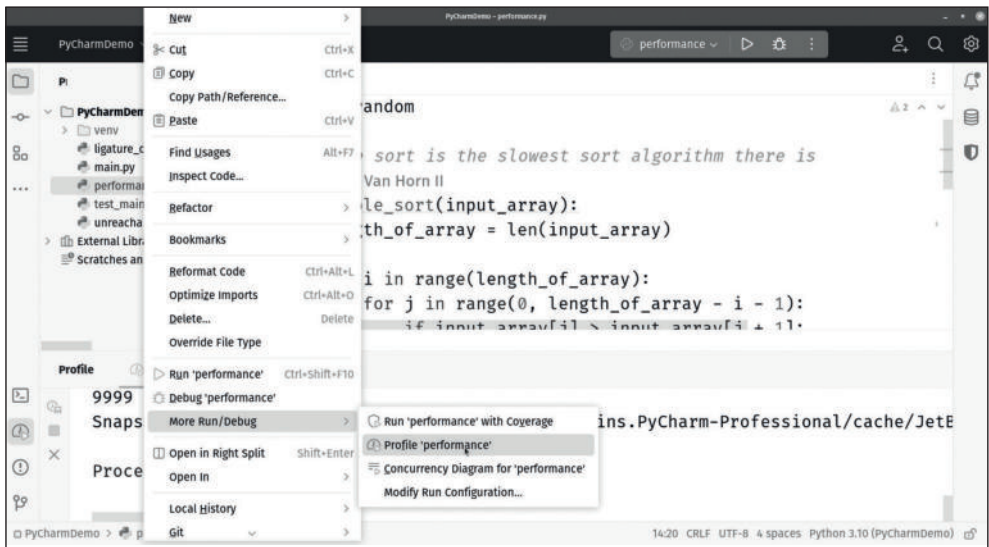


Рис. 1.8. Кликните правой кнопкой мыши файл, который вы хотите профилировать, и выберите **More Run/Debug | Profile 'performance'**, чтобы просмотреть профиль производительности

Помните: если вы используете тот же код, что и я, его выполнение займет много времени, особенно на более медленном компьютере. Не стесняйтесь уменьшать размер списка, если его запуск занимает слишком много времени. Результатом является файл `.pstat`, который отображается в виде таблицы в PyCharm. Опять же, мы рассмотрим это более подробно в главе 6 «Бесшовное тестирование, отладка и профилирование». Вы можете увидеть отчет о производительности на рис. 1.9:

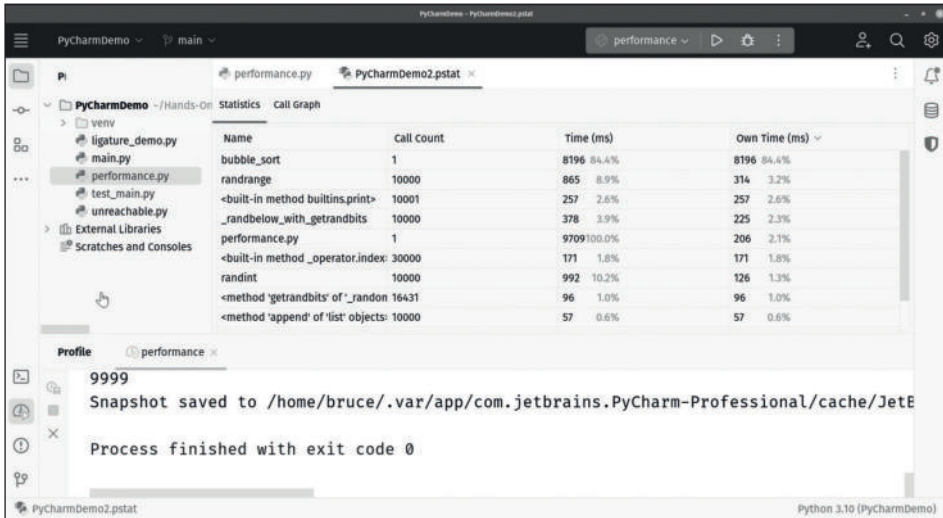


Рис. 1.9. Профилировщик ресурсов PyCharm показывает узкие места производительности работающей программы

Как видите, 84,4 % времени программы тратится на функцию `bubble_sort`, которая является узким местом. PyCharm рассказал вам, на чем сосредоточить усилия по рефакторингу, чтобы улучшить производительность вашей программы.

Публикация из IDE

Если вам нужно опубликовать свой код на тестовом сервере и вы не используете для этого систему непрерывной интеграции, можете использовать PyCharm. Чтобы внести ясность, вам следует использовать систему непрерывной интеграции, но я часто использую функции PyCharm на ранних этапах проекта, прежде чем система непрерывной интеграции заработает, чтобы получить код, с которым заинтересованные стороны могут поиграть. Вы можете выполнить развертывание с использованием **протокола передачи файлов (FTP)** или **протокола безопасной передачи файлов (SFTP)** или скопировать непосредственно в общий сетевой ресурс, чтобы быстро и легко поделиться своим прогрессом со всеми, кто захочет на него посмотреть.

Инструменты рефакторинга

PyCharm имеет надежные инструменты рефакторинга¹, которые можно ожидать от подходящей IDE. Если вы хотите изменить имя переменной или даже

¹ Рефакторинг – процесс изменения внутренней структуры программы, не затрагивающий ее внешнего поведения и имеющий целью облегчить понимание ее работы. В основе его лежит последовательность небольших эквивалентных (т. е. сохраняющих поведение программы) преобразований. Поскольку каждое преобразование маленькое, программисту легче проследить за его правильностью, и в то же время вся последовательность может привести к существенной перестройке программы и улучшению ее согласованности и четкости. – *Прим. ред.*

сигнатуру метода функции, кликните правой кнопкой мыши и выберите инструмент **Refactor**. Будьте уверены, внесенные вами изменения будут перенесены на все связанные экземпляры вашего проекта, а не только на файл, который вы редактируете. На рис. 1.10 показан пример этого в действии:

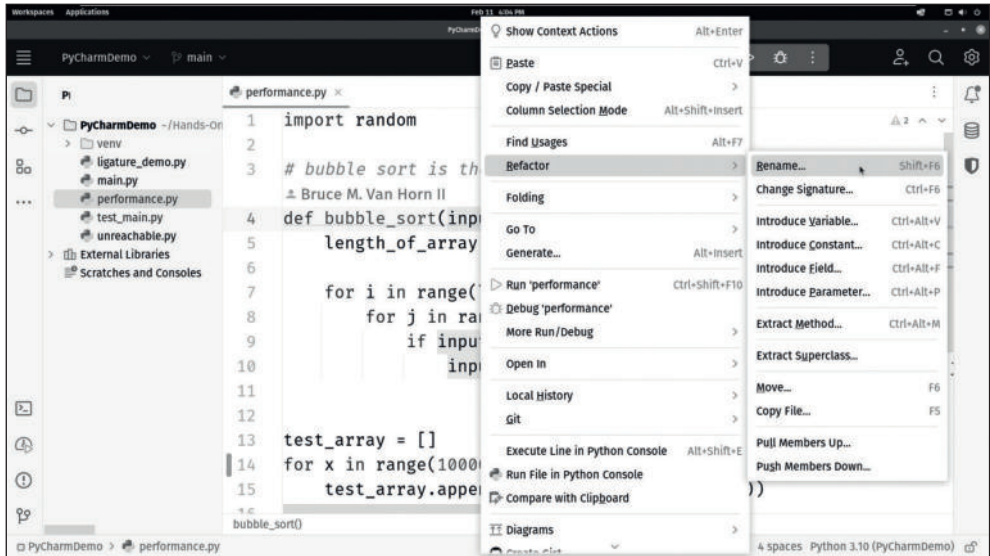


Рис. 1.10. PyCharm имеет полный набор доступных инструментов рефакторинга

Помимо переименования переменной или функции, вы можете выполнить и другие действия, например изменить сигнатуру метода и изменить структуру объектно ориентированных классов.

Работа с базами данных в PyCharm

Если вы работаете с базами данных, профессиональная версия PyCharm включает графический редактор таблиц и поддержку SQL для десятков популярных баз данных. Посмотрите это на рис. 1.11. Я расскажу подробнее о профессиональной версии PyCharm чуть позже в этой главе; у нас будет целая глава о функциях базы данных (глава 11 «Понимание управления базами данных в PyCharm»).

Как видите, очень многие реляционные базы данных и базы данных NoSQL поддерживаются напрямую.

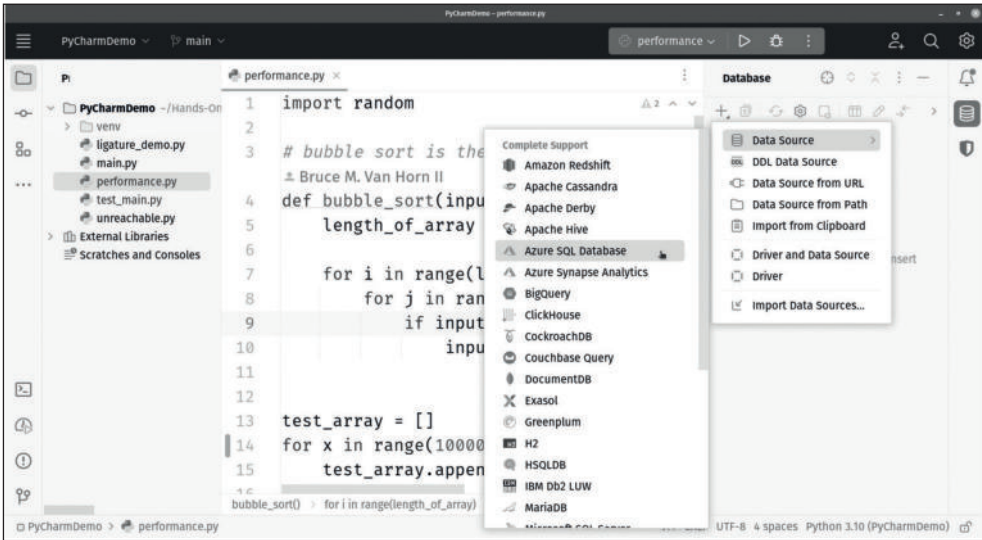


Рис. 1.11. PyCharm имеет надежный и полный набор инструментов для работы с реляционными базами данных, такими как Oracle, SQL Server, Postgres и многие другие

Удаленная разработка, контейнеры и виртуальные машины

Наконец, но не исчерпывающе PyCharm имеет функции для работы с удаленными системами через **SSH**, локальными виртуальными машинами (VM) с использованием **Vagrant** от HashiCorp и обширную поддержку контейнеров **Docker**.

Это не исчерпывающий список того, что PyCharm может для вас сделать, но теперь вы поняли суть. Все инструменты, которые могут вам понадобиться, интегрированы в среду разработки. Вероятно, поэтому они и назвали это *интегрированной средой разработки*.

Опции веб-разработки

Я готов поспорить, что более чем половине разработчиков, работающих на Python, в какой-то момент понадобится разработать веб-проект. Независимо от того, делаете ли вы, как и я, SaaS-предложение в виде полностью реализованного веб-приложения, или занимаетесь ракетостроением, и вам нужен способ визуализировать и в интерактивном режиме поделиться своим последним быстрым преобразованием Фурье по данным радиоизлучения в дальнем космосе, – веб-проекты обычно неизбежны. Я не ученый, и не я это придумал. Если последнее предложение и имело для кого-то смысл, то это было чистое совпадение.

Работа с веб-проектами предлагает новый, отдельный уровень сложности. Большинство из них используют трехуровневые конструкции, обычно представляемые с помощью шаблона **модель–представление–контроллер (MVC)**. Если вы не понимаете, что это значит, читайте внимательно, потому что в этой книге целый раздел посвящен веб-разработке. На данный момент это означает, что приложение имеет интерфейс, с которым пользователь может взаимодействовать, средний уровень, содержащий соединительную логику, и уровень базы данных для хранения и извлечения структурированных данных. Только средний уровень выполнен на Python. Мы подробно рассмо-

трим веб-разработку в последующих главах, а сейчас я хочу рассказать вам об уровне инструментов, которые вы получаете с PyCharm.

Компания JetBrains, создавшая PyCharm, производит множество IDE, ориентированных на разные языки. Одна из их IDE специально предназначена для веб-разработки. Она называется **WebStorm**. Ранее я говорил, что хорошие IDE ориентированы на один язык. WebStorm ориентирован на JavaScript; в частности, мы говорим о полнофункциональном JavaScript. Современное выполнение JavaScript происходит в двух местах. Традиционно JavaScript всегда выполнялся в браузере. Около 10 лет назад был выпущен **Node.js**, а JavaScript был выпущен за пределы окна браузера и разрешен для запуска на серверной части.

Ранее я упоминал о мощном наборе функций PyCharm для работы с базами данных. У JetBrains также есть IDE, предназначенная для разработчиков баз данных SQL, под названием **DataGrip**. Так получилось, что профессиональная версия PyCharm включает в себя весь набор функций, доступных в WebStorm и DataGrip. Приобретая версию Professional, вы получаете три продукта JetBrains в одном пакете: PyCharm, WebStorm и DataGrip. Когда вы начнете использовать PyCharm для работы над веб-проектами, вам потребуются все три набора функций, и они есть в профессиональной версии.

Поддержка научных вычислений

Рост области науки о данных сыграл важную роль в развитии самого Python, и сейчас Python является наиболее распространенным инструментом программирования, используемым в научных проектах (даже более распространенным, чем **R**). Примечательными функциями, включенными в PyCharm и облегчающими работу с данными, являются интеграция **IPython**, блокнотов **Jupyter** и интерактивной консоли. Поддержка научных вычислений в PyCharm подробно описана в четвертом разделе этой книги, начиная с главы 12 «Включаем научный режим». PyCharm также предоставляет настраиваемое представление, оптимально организующее рабочее пространство в научном проекте под названием **SciView**, который показан на рис. 1.12:

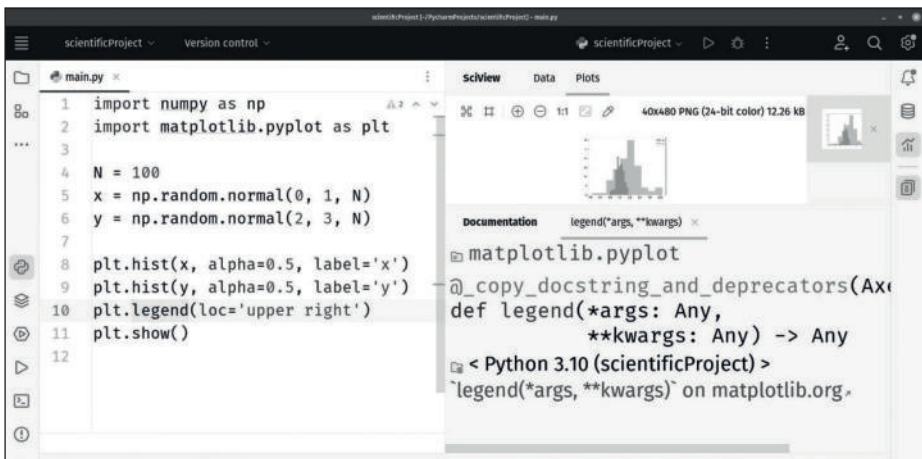


Рис. 1.12. SciView в PyCharm через удобный интерфейс предоставляет доступ к инструментам научной визуализации

Особенности редакций Professional, Community и Educational

Существует три редакции PyCharm. Я упомянул две, потому что третья – это специальная версия, полезная только учителям, а цель этой книги – разработка приложений, а не инструкции по разработке программного обеспечения. Я расскажу вам о каждой, но я знаю, что вам нужна сравнительная таблица функций. Вы найдете ее на рис. 1.13:

	PyCharm Professional	PyCharm Community
Стоимость	Платно	Бесплатно
Интеллектуальный редактор Python	✓	✓
Графический отладчик и средство запуска тестов	✓	✓
Инструменты навигации по коду и рефакторинга	✓	✓
Проверка кода	✓	✓
Git, Subversion и другие инструменты контроля версий	✓	✓
Научные инструменты	✓	
Веб-разработка с использованием HTML, JavaScript, CSS и т. д.	✓	
Поддержка веб-фреймворка Python	✓	
Профилирование производительности	✓	
Удаленная разработка, контейнеры и т. д.	✓	
Поддержка баз данных и SQL	✓	

Рис. 1.13. Сравнительная таблица функций, показывающая возможности бесплатной версии Community и платной версии Professional

Версия **Community** бесплатна, но предлагает лишь ограниченный набор функций по сравнению с версией Professional. Она идеально подходит для работы над проектами, которые предполагают работу только с Python. Продукт, над которым я работаю, содержит набор скриптов Python, выполняющих пакетную обработку больших объемов данных. Все происходит на Python, и это идеальный вариант использования версии Community. Если все, что вам нужно, – это потрясающая среда разработки Python, используйте бесплатную версию. Версия Community также идеально подходит, если вы просто работаете над скриптами автоматизации, такими как графические конвейеры для компьютерной 3D-графики, или над общей автоматизацией ИТ-задач.

Версия **Professional**¹ имеет все функции бесплатной версии, но к ней добавлены типы веб-разработки, баз данных, удаленной разработки, контейнеризации и научных проектов. Она предназначена для профессионалов, создающих

¹ С российского аккаунта покупка PyCharm Professional на февраль 2024 года недоступна. – *Прим. ред.*

публикуемые программные проекты. Хотя она и не бесплатна, JetBrains делает ее более доступной, предлагая несколько вариантов цен в зависимости от того, как вы используете этот инструмент. Индивидуальные разработчики могут получать лицензии по более низкой цене, чем корпоративные разработчики. Есть также способы получить профессиональную версию бесплатно, например доказав, что вы используете PyCharm в проекте **программного обеспечения с полностью открытым исходным кодом (FOSS)**. Начинающие компании могут получить скидку 50 %, а если вы преподаете в учебном лагере по кодированию или в университете, вы также можете претендовать на бесплатные профессиональные лицензии. Поскольку эти параметры могут меняться со временем, вам следует посетить веб-сайт JetBrains для получения полной информации по адресу <https://www.jetbrains.com/pycharm/>.

Ранее я говорил, что существует три редакции PyCharm, а мы рассмотрели только две. Версия **Educational** предназначена для преподавателей и преподавателей университетов, разрабатывающих учебные программы по преподаванию Python. В этой версии можно создавать и воспроизводить интерактивные уроки прямо в IDE. Это ценно только для учителей, инструкторов и создателей контента.

В этой книге я сосредоточусь на функциях, присутствующих в версиях Community и Professional.

КРАТКОЕ СОДЕРЖАНИЕ

В этой главе мы представили сам язык Python, а также историю создания IDE Python в целом и PyCharm в частности.

Мы также обсудили удобство использования PyCharm для программистов Python. В частности, чтобы иметь возможность в полной мере воспользоваться всеми преимуществами и возможностями, которые предлагает PyCharm, не становясь слишком зависимыми от IDE, программист должен сначала освоить основы языка Python и его основной синтаксис. Мы также рассмотрели сравнение самого PyCharm с другими редакторами и IDE Python и причину, по которой PyCharm считается лучшей средой разработки из всех.

Наконец, мы сравнили две версии PyCharm, доступные для загрузки: платную версию Professional и бесплатную версию Community. Если вы работаете с большими и сложными проектами со множеством взаимодействующих частей, включая управление базами данных, языки веб-разработки и возможность просмотра в научных отчетах, то вам, вне всякого сомнения, понадобится версия Professional.

В следующей главе вы узнаете, как загрузить PyCharm, настроить его в своей операционной системе и настроить его среду для ваших проектов Python. Это послужит первым шагом в начале работы с PyCharm, после чего мы начнем обсуждать конкретные функции PyCharm, которые рассматриваются в этой книге.

Вопросы

Ответьте на следующие вопросы, чтобы проверить свои знания, полученные в этой главе.

1. Программисты обычно разрабатывают свой код с помощью редактора или IDE. В чем разница между ними и к какому из них относится PyCharm?

2. Почему некоторые могут подумать, что IDE для разработки на Python может быть неуместной или ненужной?
3. Каковы ключевые особенности PyCharm? Которые из этих функций дают PyCharm преимущество перед другими редакторами и IDE?
4. Какое преимущество имеет PyCharm перед такими редакторами, как Visual Studio Code или vim, которые можно настроить для выполнения многих функций, предлагаемых PyCharm?
5. Какие существуют редакции PyCharm? Каковы главные различия между ними?

ДАЛЬНЕЙШЕЕ ЧТЕНИЕ

Чтобы узнать больше о темах, рассмотренных в этой главе, посетите следующие ресурсы.

- Friend, Edward H. *Sorting on Electronic Computer Systems. Journal of the ACM (JACM)* 3.3 (1956): 134-168.
- Nguyen, Quan. *Hands-On Application Development with PyCharm: Accelerate your Python applications using practical coding techniques in PyCharm*. Packt Publishing Ltd, 2019.
- Shelley, Percy B. *Ozymandias*. <https://www.poetryfoundation.org/poems/46565/ozymandias>.
- Wikipedia contributors. (2022, December 19). *Bubble sort*. Wikipedia. https://en.wikipedia.org/wiki/Bubble_sort.

Установка и настройка PyCharm

В предыдущей главе мы ознакомились с наиболее популярными функциями PyCharm и рассмотрели не только то, что делает PyCharm отличной IDE, но и то, что делает любую IDE великолепной и общепризнанной. Существует базовый набор функций, который нам, как разработчикам, необходим для того, чтобы работать по-настоящему продуктивно. В этой главе мы сосредоточим внимание на установке PyCharm. Вы можете подумать, что просто скачаете и установите его. Вы можете это сделать, но есть другие способы установки PyCharm, которые могут вам понравиться больше. Существуют также другие варианты в зависимости от вашей операционной системы.

Помимо простой загрузки инсталлятора, его запуска и нажатия кнопки Next, пока диалоговые окна инсталляции не исчезнут, есть и другие соображения, необходимые для правильной установки и работы инструмента. PyCharm обладает широкими возможностями настройки, и некоторые из этих параметров настройки вам будут доступны, как только программа запустится в первый раз. Некоторые из этих опций интересны, а некоторые могут вызвать затруднения, если вы просто выбираете каждую опцию и каждую настройку в процессе.

Вот что вас ждет в этой главе:

- загрузка JetBrains Toolbox и использование его для установки и управления PyCharm. Это мой рекомендуемый метод установки, поскольку вы получаете простой способ обновления и удаления. Вы даже можете установить и управлять несколькими версиями PyCharm, если вам это понадобится;
- запуск PyCharm в первый раз и знакомство с параметрами настройки, которые программное обеспечение предоставляет при первом запуске. Естественно, вы можете изменить их в любое время, и мы это тоже рассмотрим;
- клонирование репозитория этой книги с GitHub, используя инструменты **интегрированной системой контроля версий (VCS)** PyCharm.

ТЕХНИЧЕСКИЕ ТРЕБОВАНИЯ

Чтобы добиться успеха в этой главе, вам понадобится следующее.

- Компьютер. На тот случай, если вы пропустили этот прикол в главе 1, я горжусь тем, что я последователен и ничего не оставил на волю случая! Компьютер, если он у вас есть, должен соответствовать следующим системным требованиям для PyCharm:
 - ◆ 64-разрядные версии Microsoft Windows 8 или более поздней версии, macOS 10.14 или более поздней версии, или Linux с рабочим столом GNOME или KDE;
 - ◆ официальные системные требования указывают минимум 4 Гб ОЗУ и рекомендуемую 8 Гб. Но если вы собираетесь выполнять нетривиальную работу и думаете о характеристиках нового компьютера, я бы предложил поставить оперативную память не менее 16 Гб, а еще лучше 32 Гб. Это именно то, что необходимо для PyCharm. Большинство разработчиков используют не только IDE;
 - ◆ 2,5 Гб места на жестком диске; SSD рекомендуется. Опять же, это рекомендация PyCharm для нижнего уровня. Если вы делаете настоящие покупки, приобретите диск NVMe, а не SSD. Его скорость раз в 10 выше, а стоимость вполне доступна;
 - ◆ минимальное разрешение экрана 1024×768. Это бюджетная спецификация, и это шутка. На экране такого размера многого не сделаешь, и даже дешевые современные компьютеры легко поддерживают разрешение 1920×1080. Для профессиональной работы вам действительно нужен 4K, если это возможно, а еще лучше два (или более) монитора с разрешением 1920×1080. Чем больше у вас экранного пространства, тем более продуктивно вы будете работать с любой IDE. На мониторе 4K я могу использовать PyCharm, который покажет мне окно моего проекта, два окна с открытым кодом рядом, окно базы данных и окно сеанса терминала.
- Операционная система. Лучше всего, если она уже установлена на вашем компьютере, поскольку в этой книге мы не рассматриваем, как это сделать. Windows, macOS, Linux – для этой книги все одинаково, поскольку PyCharm работает во всех трех средах, а пользовательский интерфейс в каждой из них практически идентичен.
- Подключение к интернету.
- Установка Python 3. В этой книге мы будем использовать исключительно Python 3. Существует несколько «разновидностей» Python 3, но по большей части подойдет старый добрый Python 3 с <https://www.python.org>. Мы вернемся к этим «разновидностям» позже, когда начнем говорить о виртуальных средах в главе 3 «Настройка интерпретаторов и виртуальные среды». Если номера обновлений вас утешают, то последняя версия на момент написания статьи – 3.11.1. Версия Python, которую я использую в производственном SaaS-приложении, упомянутом в главе 1, – 3.8. Если ваша установка Python 3 старше этой, следует ее обновить.

- В какой-то момент может пригодиться учетная запись **GitHub**, поскольку я буду делиться кодом из книги, используя репозиторий **Git**. Поскольку вы будете клонировать некоторый код, но не отправлять его, входить в систему не обязательно, т. е. если только вы не хотите войти в систему, а лишь просмотреть репозиторий книги и поставить ей звездочку. Это было бы здорово и доказало бы всему миру, что вы достойный человек.

ЗАГРУЗКА PYCHARM ТРАДИЦИОННЫМ СПОСОБОМ

Сначала я собираюсь показать самый простой, прямой и распространенный способ установки PyCharm. Вполне вероятно, что вы уже сделали это до того, как купили эту книгу, и в этом нет ничего плохого. После того как мы это рассмотрим, я покажу вам предпочтительный способ установки с помощью бесплатного приложения от JetBrains под названием Toolbox. Вы можете выбрать любой из методов установки, который вам нравится, зная, что этот выбор не повлияет на результат всего, что мы делаем в этой книге.

Обратите внимание, что для версии Professional доступна 30-дневная бесплатная пробная версия, если вы хотите ее опробовать. По истечении 30 дней вам придется заплатить за нее или перейти на версию Community. Более того, не заикивайтесь на номере версии, показанном на рис. 2.1. JetBrains довольно часто выпускает обновления для PyCharm, и их число, вероятно, изменится несколько раз, прежде чем эта книга успеет появиться на прилавках. С последней версией трудно ошибиться.

Чтобы установить PyCharm, откройте в браузере <https://www.jetbrains.com/pycharm>. Сайт определит вашу операционную систему и предложит вам правильный вариант загрузки.

Страница загрузки состоит из трех основных частей, как показано на рис. 2.1.

1. Вы можете выбрать операционную систему (Windows, macOS или Linux). Если хотите использовать Linux, убедитесь, что вы используете GNOME или KDE, поскольку они поддерживаются оконными менеджерами.
2. Вы можете выбрать между версиями Professional или Community.
3. Независимо от первых двух опций, обязательно просмотрите раскрывающийся список инсталлятора. Windows и Linux позволяют выбирать между Intel и ARM. В свою очередь, macOS позволяет выбирать между Intel и Apple Silicon:

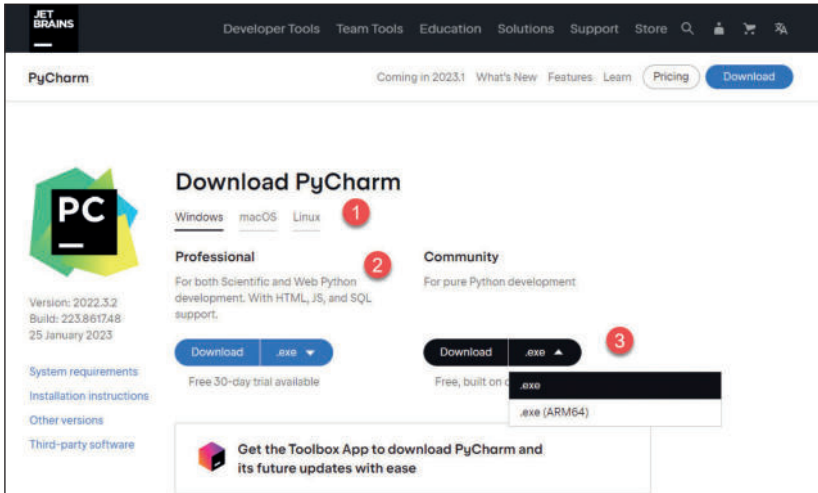


Рис. 2.1. Окно загрузки на сайте JetBrains; убедитесь, что выбрана правильная операционная система и процессор

После загрузки нужной версии выполните обычную процедуру установки для вашей операционной системы. Если вы выбрали версию Linux, вы не загружаете установщик, а загружаете сжатый файл в формате `.tar`. Можете просто извлечь его и запустить PyCharm из полученной папки.

JETBRAINS TOOLBOX

Я только что представил наиболее распространенный среди программистов способ установки программного обеспечения, включая PyCharm. Я предпочитаю другой способ: установить отдельный продукт под названием JetBrains Toolbox. Toolbox особенно полезен, если у вас есть несколько продуктов JetBrains, как у меня. У меня есть подписка на все их инструменты, и многими из них я регулярно пользуюсь. Моя любимая среда разработки для C# – JetBrains Rider, которую я использовал исключительно в своей книге «Практическая реализация шаблонов проектирования на C#», доступной в Packt Publishing.

Даже если вы не используете несколько продуктов JetBrains, Toolbox предоставляет некоторые полезные функции, такие как простой способ инсталляции, удаления и обновления вашей PyCharm. Вы даже можете использовать его для установки нескольких версий PyCharm, если вам когда-нибудь понадобится, включая выпуски **программы раннего доступа (EAP)**. Релизы EAP предоставляют доступ к самым передовым функциям JetBrains еще до того, как они станут общедоступными. Как руководитель разработки, я предпочитаю тестировать новейшие IDE, прежде чем дать разрешение *своей* команде разработчиков. Toolbox делает это очень легко.

Toolbox можно загрузить отдельно и бесплатно. Давайте начнем с повторного посещения страницы загрузки PyCharm. Внизу вы найдете ссылку на JetBrains Toolbox, как показано на рис. 2.2:

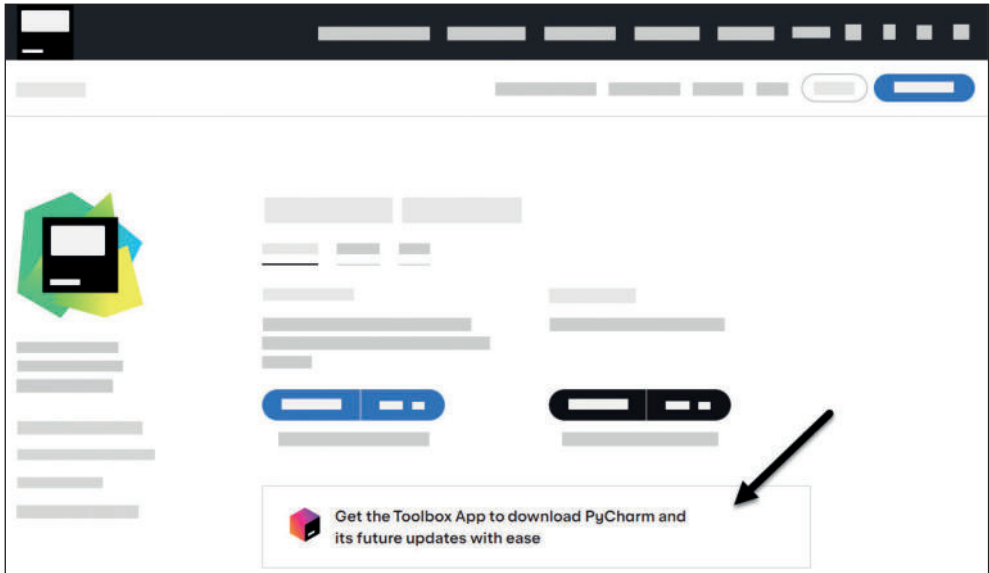


Рис. 2.2. Пропустите обычные ссылки для загрузки, которые мы рассматривали ранее, и вместо этого кликните ссылку на Toolbox

Вполне возможно, что JetBrains реорганизует свой сайт. Если внизу нет ссылки, вы можете просто выполнить поиск *Toolbox* и найти самую последнюю версию для загрузки. Вы попадете на страницу, подобную той, что показана на рис. 2.3:

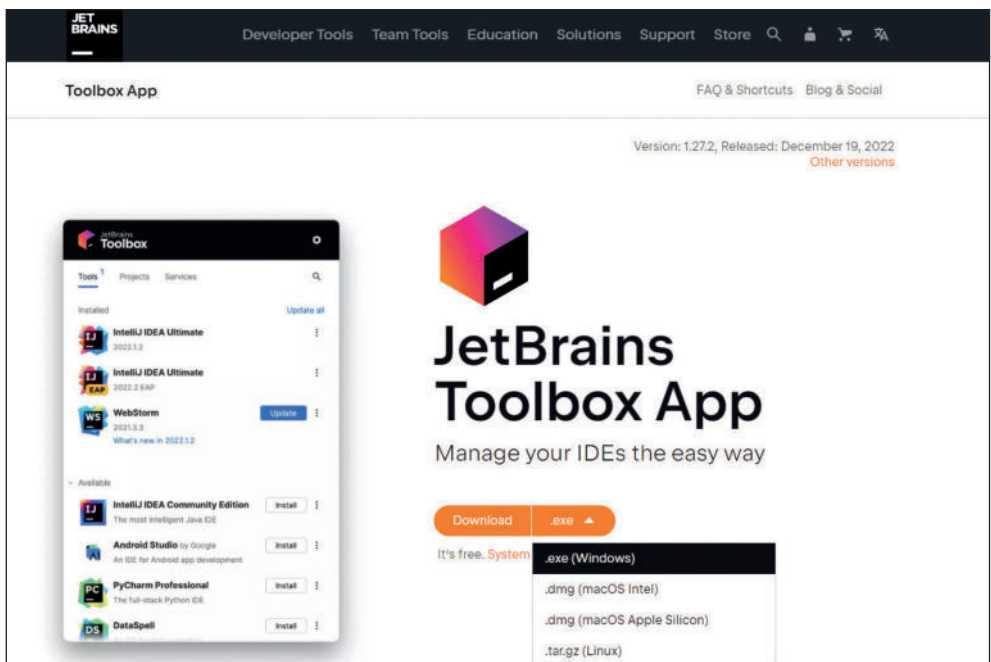


Рис. 2.3. Экран загрузки Toolbox

Как и в случае с обычной загрузкой PyCharm, которую мы рассматривали ранее, эта веб-страница определит используемую вами операционную систему. Варианты немного проще, но, если вы используете Mac, обязательно нажмите кнопку форматирования, чтобы выбрать Intel или Apple Silicon, как показано на рис. 2.3.

Установка Toolbox в Windows

Процесс установки Toolbox прост. Это тот случай, когда вы запускаете установщик и нажимаете кнопку **Next**, пока диалоговое окно не исчезнет.

Обратите внимание, что, когда Toolbox запущен, вы можете найти его на панели задач Windows.

Установка Toolbox в macOS

Как и все, что касается Mac, установка macOS очень проста. Убедившись, что вы загрузили правильную версию файла .dmg (Intel или Apple Silicon), найдите загруженный файл .dmg в папке Downloads в вашей папке home. Дважды кликните, чтобы открыть файл .dmg. Перетащите значок Toolbox в папку Applications. Все готово!

Установка PyCharm с помощью Toolbox

Независимо от того, как вы сюда попали, – будь то Mac, Windows или Linux, – теперь у вас должен быть запущен Toolbox. На данный момент этот опыт почти универсален. В macOS Toolbox – это обычное приложение, такое же, как и любое другое. Однако в Windows он запускается в системном трее.

При первом запуске Toolbox возникает довольно стандартное **лицензионное соглашение с конечным пользователем (EULA)**. Вы знаете, что делать. Прочтите его и убедитесь, что JetBrains не требует выдачи вашего первенца, а затем сделайте свой выбор: согласиться или не согласиться с лицензионным соглашением. Естественно, если вы не согласны, наше время общения подошло к концу, если только вы не настроитесь на случайные папины шутки. Я буду исходить из того, что вы согласились с лицензионным соглашением.

Установив Toolbox, можете устанавливать IDE. Вы увидите экран, подобный показанному на рис. 2.4, на котором перечислены все доступные продукты JetBrains:

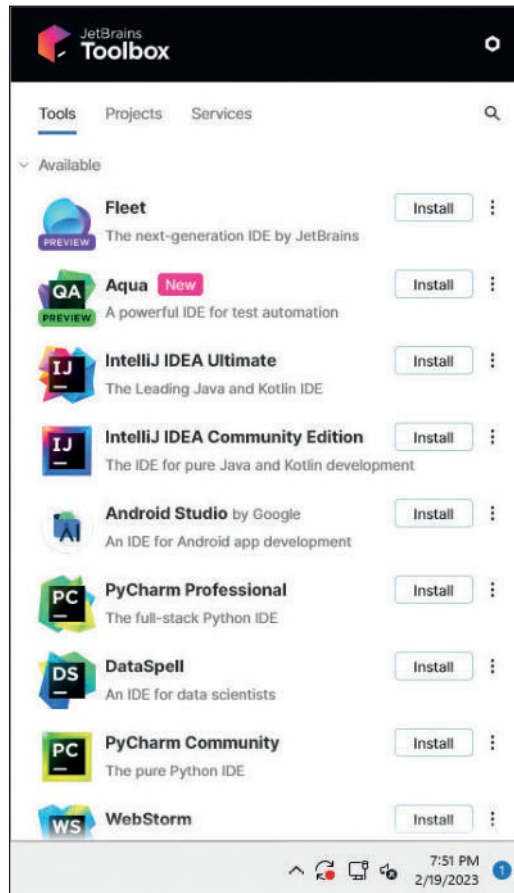


Рис. 2.4. JetBrains Toolbox, работающий на панели задач Windows

Естественно, нас интересует только PyCharm. На рис. 2.4 в списке показаны PyCharm Professional и PyCharm Community. Если вас интересует образовательная версия PyCharm, она находится ниже в списке.

Чтобы установить IDE, нажмите кнопку **Install** и подождите, пока Toolbox загрузит и установит IDE. После установки меню Toolbox немного изменится, показывая, какие инструменты вы установили. Вы можете увидеть мой Toolbox после установки PyCharm Professional на рис. 2.5:

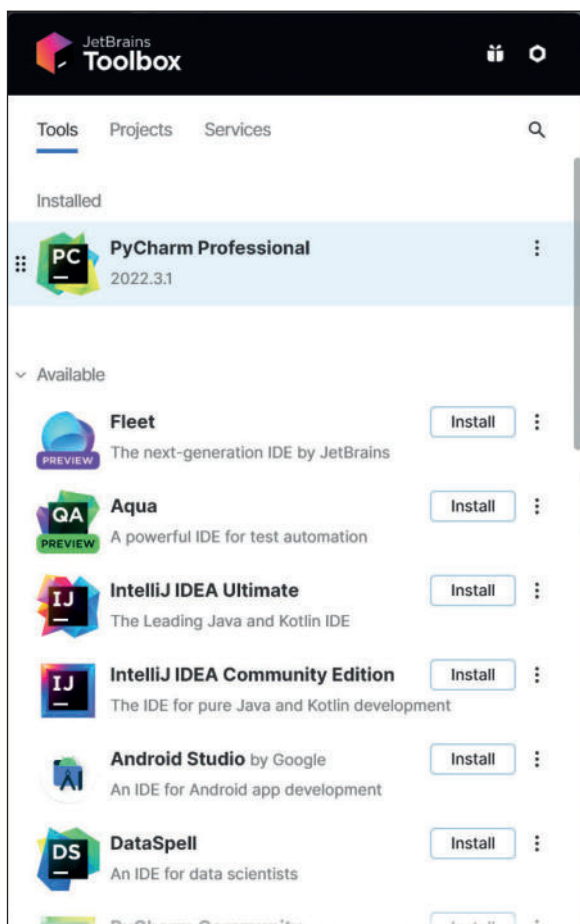


Рис. 2.5. Приложение Toolbox после установки PyCharm Professional

Обратите внимание, что Toolbox перечисляет установленные вами приложения вверху списка, а доступные для установки – ниже. Преимущество Toolbox, по сравнению с обычной установкой, – это возможность запускать, обновлять и удалять ваши IDE, а также легко экспериментировать с различными версиями.

Запуск PyCharm с помощью Toolbox

После установки PyCharm вы можете запустить ее или любую установленную IDE, просто кликнув запись в меню. Прежде чем мы покинем Toolbox, позвольте мне показать вам еще несколько интересных функций.

Установка альтернативной версии или ее удаление

Рядом с каждой установленной IDE в списке вы увидите три точки, как показано на рис. 2.6:

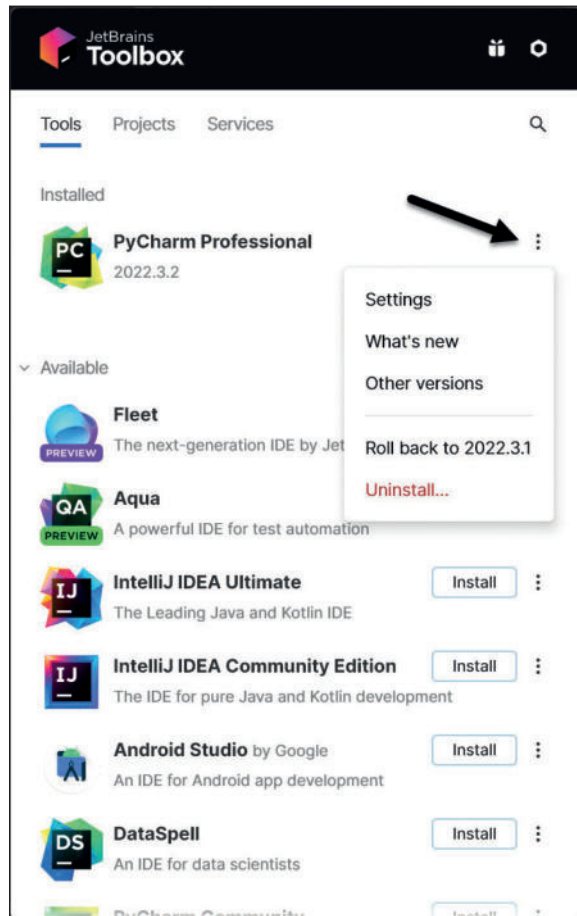


Рис. 2.6. Три точки рядом с каждым приложением обозначают вложенное меню

Точки – это меню. Если вы нажмете на них, то увидите несколько вариантов. Вы можете получить доступ к настройкам PyCharm. Мы собираемся сделать это позже внутри самого PyCharm, поэтому сейчас нам не нужно этого делать. Чтобы просмотреть последние новости, нажмите **What's new**. Ниже есть возможность установки различных версий PyCharm, кроме последней. Можете легко вернуться к последней установке, если у вас возникнут проблемы с последней версией. Наконец, под разделителем, выделенным устрашающим красным шрифтом, находится опция удаления. После пункта меню даже есть устрашающее и загадочное многоточие. Наведение курсора на эту опцию обычно все-ляет дурное предчувствие у всех, кто ее пробует. Вас предупредили.

Обновление PyCharm с помощью Toolbox

Toolbox поможет вам оставаться в курсе последних версий PyCharm и любых других IDE, которые вы используете. Сам Toolbox имеет свои настройки и собственный механизм обновления.

Чтобы перейти к этим настройкам, кликните шестиугольный значок в правом верхнем углу экрана Toolbox, как показано на рис. 2.7:

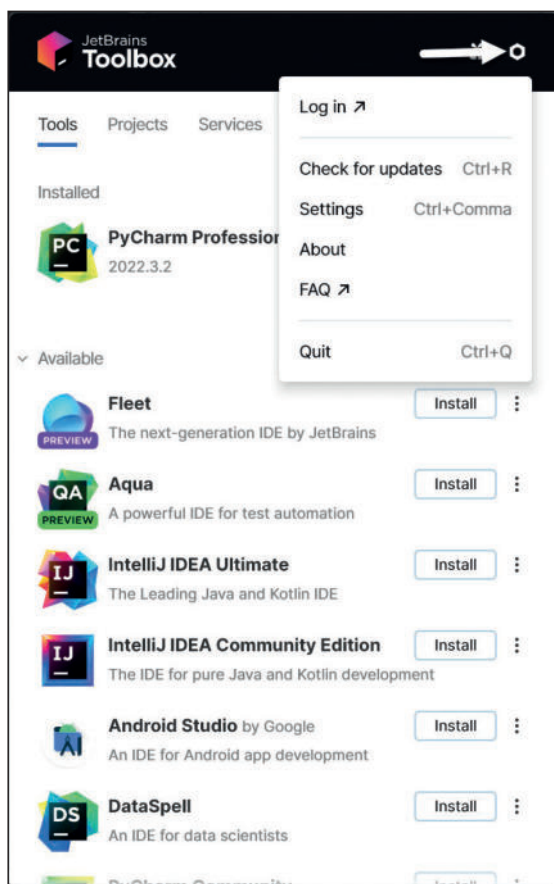


Рис. 2.7. Используйте шестиугольный значок в верхней части Toolbox, чтобы получить доступ к параметрам обновления

Опция **Log in** позволяет подключить Toolbox к вашей учетной записи JetBrains. Опции **About** и **FAQ** отображают информацию о продукте. Последний приведет вас на веб-сайт, где хранятся **часто задаваемые вопросы (FAQ)**. Опция **Quit** закрывает программу Toolbox.

ЗАПУСК И РЕГИСТРАЦИЯ PYCHARM

При запуске PyCharm, независимо от того, как вы это делаете, отображается заставка, а затем открывается набор типичных экранов первого запуска. Если вы установили версию Professional, первое, что вы увидите, – это экран регистрации, показанный на рис. 2.8:

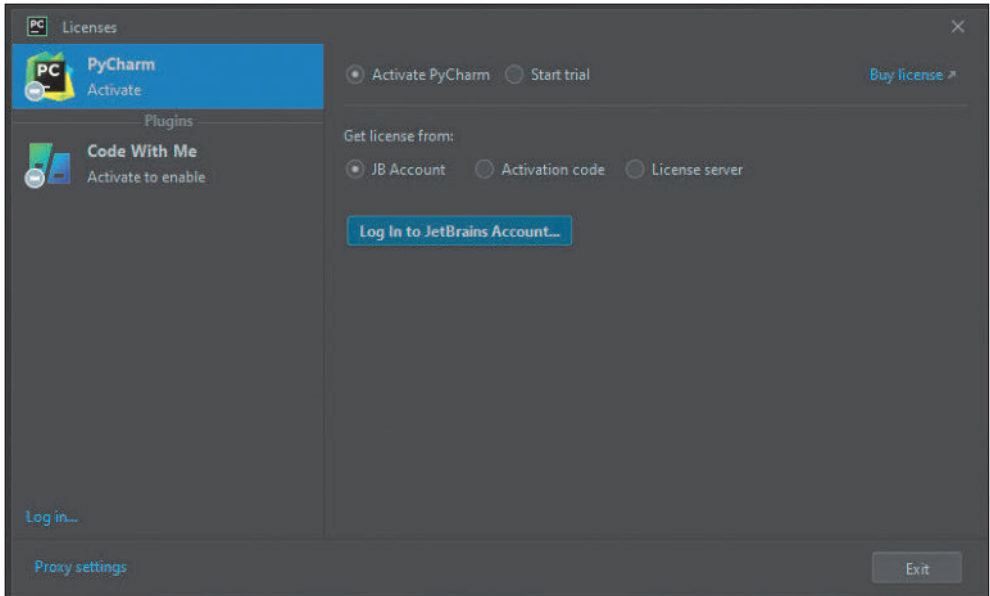


Рис. 2.8. Экран лицензирования в PyCharm Professional появляется при первом же запуске

Самый распространенный способ продолжения – войти в свою учетную запись JetBrains. Нажатие кнопки **Log In to JetBrains Account...** запустит ваш браузер. Вы можете войти в систему или создать учетную запись. Если вы приобрели лицензию, при входе в систему ваша копия будет связана с приобретенной лицензией.

Если вы работаете в компании, которая владеет множеством лицензий, может потребоваться войти на сервер лицензий JetBrains. Также существует возможность регистрации с использованием регистрационного кода. Этот код вы найдете в своей учетной записи магазина. Это может быть полезно, если у вас нет хорошего доступа в интернет, он потребует закрыть одну копию.

У вас более одного компьютера?

Обратите внимание, что установка на нескольких компьютерах разрешена, если вы не используете две копии одновременно с одной и той же лицензией. IDE обнаружит это и потребует закрыть одну копию.

Если у вас нет лицензии и вы не готовы принять на себя ответственность, можете выбрать опцию **Start trial**. Вам все равно необходимо войти в учетную запись JetBrains, чтобы активировать пробную версию.

НАСТРОЙКА PYCHARM

Когда вы запускаете PyCharm в первый раз и проходите лицензирование и регистрацию, следующее, что вы видите, – это небольшое окно, представляющее

PyCharm без загруженного проекта. Вы можете увидеть версию этого экрана со светлой цветовой схемой на рис. 2.9:

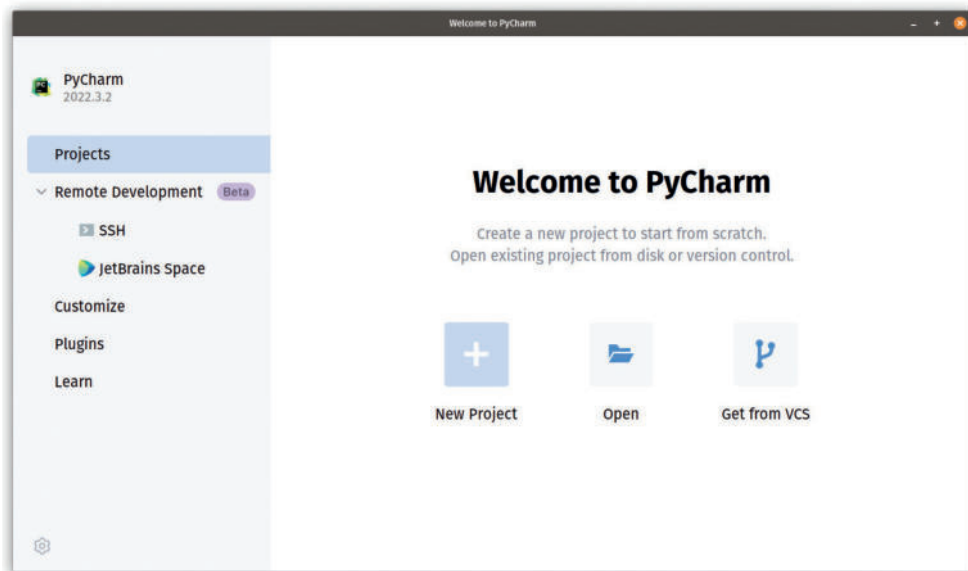


Рис. 2.9. PyCharm без загруженного проекта

В этом окне есть несколько очевидных опций. Я называю их *очевидными*, потому что они находятся прямо посередине большого открытого пространства по центру окна. Вы можете создать проект, открыть существующий проект или клонировать проект из системы управления версиями, например Git или **Subversion (SVN)**. Однако моя первая остановка – в серой зоне слева на экране, где вы найдете пункт меню **Customize**. Давайте продолжим и рассмотрим варианты настройки PyCharm в соответствии с вашим стилем работы. Нажатие кнопки **Customize** приведет вас к экрану настроек, как показано на рис. 2.10.

Этот экран позволяет нам изменять наиболее часто используемые настройки. Мы можем изменить цветовую тему, шрифт IDE и раскладку клавиатуры.

Внизу мы можем импортировать наши настройки. Это полезно, когда ваш начальник наконец-то разорился на новый ноутбук, и вы не хотите тратить кучу времени на перенастройку своей IDE. Как мы все знаем, хорошо сконфигурированная персонализированная среда разработки похожа на старый диван. Каждому, кто попытается на него сесть, будет ужасно неудобно. Кроме того, он пахнет. Ваши друзья не скажут вам, но это так. Однако для вас он идеально подстраивается под вашу форму благодаря, возможно, годам сидячего образа жизни. Это ваша IDE. Вам могут потребоваться годы, чтобы довести ее до совершенства, но придется начинать все заново каждый раз, когда вы получаете новый комп. Не сегодня, друзья. Не с PyCharm. Мы можем экспортировать, импортировать и даже делиться нашими настройками и легко переносить их в новые установки.

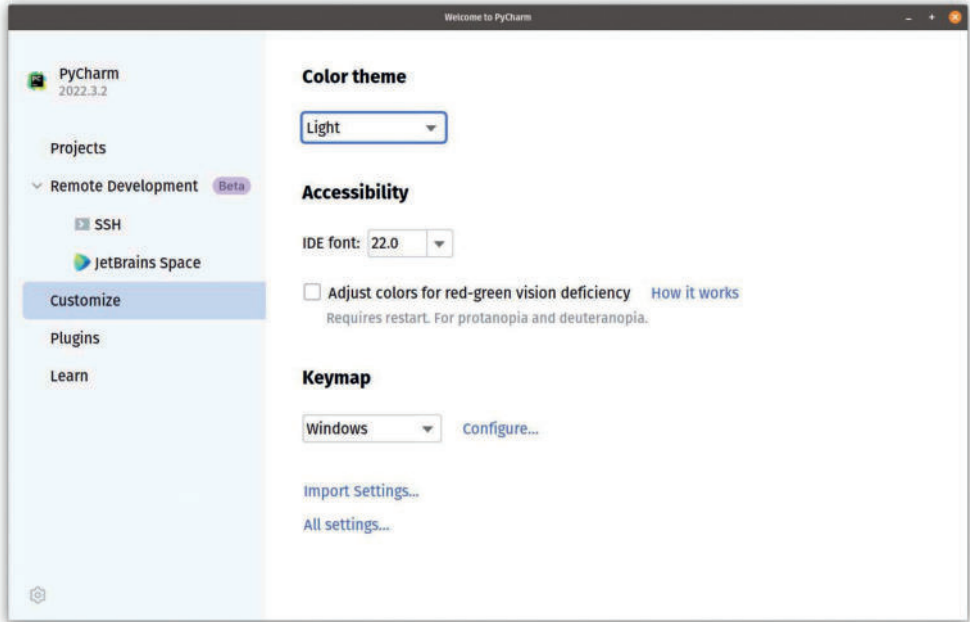


Рис. 2.10. Небольшое окно для настройки некоторых важных параметров в PyCharm

Мне нравятся большие шрифты, и неважно почему!

Возможно, вы заметили, что на всех этих снимках размеры моего шрифта огромны. Я из Техаса, а в Техасе все больше. Однако в данном случае я сделал это для вас. Снимки экрана легко читаются благодаря увеличенному размеру шрифта, поэтому на протяжении всей книги в моих настройках вы увидите безумные кегли шрифтов.

На данный момент вы можете подумать, что PyCharm имеет довольно скудный набор параметров настройки. Вы ошибаетесь. Это обманчиво простое окно призвано облегчить вам работу, представляя самые пустяковые настройки. Многие пользователи останавливаются на этом. Но не вы. Нет, не вы. Вы викинг! Другие разработчики видят обманчиво миниатюрную кнопку **All settings...** внизу экрана и думают: «Здесь обитают драконы». Вы же видите богатую возможность для приключений! Итак, нажмите кнопку **All settings...**, если осмелитесь, и мы исследуем этот дивный новый мир вместе.

Внешний вид и свойства

Нажатие кнопки **All settings...** приведет вас к экрану, показанному на рис. 2.11. Я, к счастью, не буду пытаться охватить все доступные настройки, поскольку список возможностей обширен до утомления. Вывод при перемещении по настройкам заключается в том, что вы можете настроить практически каждый пиксель, генерируемый работающей IDE:

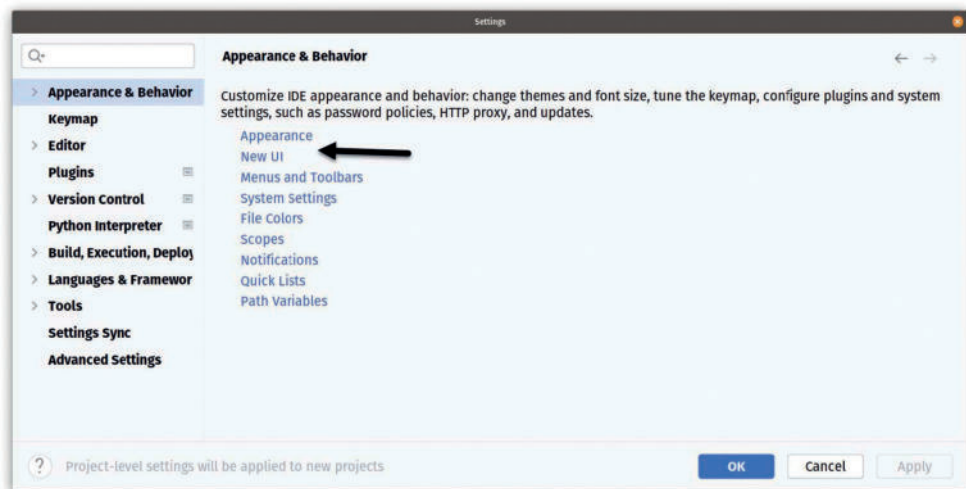


Рис. 2.11. Экран настроек в PyCharm позволяет вам изменить каждый аспект взаимодействия с пользователем в PyCharm

Возможно, самый полезный способ начать – указать, что вверху списка категорий настроек есть поле поиска. Обычно более практично искать нужную настройку, чем пролистывать все экраны, пытаясь ее найти.

Прежде чем мы пойдем дальше, я хочу, чтобы вы обратили внимание, что стрелка на рис. 2.11 указывает на два элемента на экране: **Appearance** (внешний вид) вверху (о котором мы поговорим чуть позже) и **New UI** (настройку нового пользовательского интерфейса). Пока пишется эта книга, пользовательский интерфейс PyCharm претерпевает полную переработку. По мере роста приложения меню и панели инструментов становились все более перегруженными. Чтобы исправить это, JetBrains работает над более простым макетом пользовательского интерфейса, который, как мы надеемся, облегчит изучение и внедрение этого инструмента. Классический пользовательский интерфейс, вероятно, останется доступным до 2024 года, но новый пользовательский интерфейс станет стандартом в конце 2023 года, что должно совпасть с выходом этой книги в печать. Все, что я покажу вам в этой книге, будет с включенной настройкой **New UI**. Возможно даже, что эта опция даже не отображается на экране в более поздних выпусках PyCharm.

Первое, что большинство разработчиков хотят сделать с новой IDE, – это настроить внешний вид. Мы уже нашли настройки цветовой темы и размера шрифта, но есть и другие настройки, которые вам стоит посетить. Начнем с нажатия на **Appearance**, как показано на рис. 2.11. Откроется меню слева и отобразится экран настроек внешнего вида.

Внешний вид

Открыв настройки **Appearance**, вы увидите окно, похожее на то, что показано на рис. 2.12:

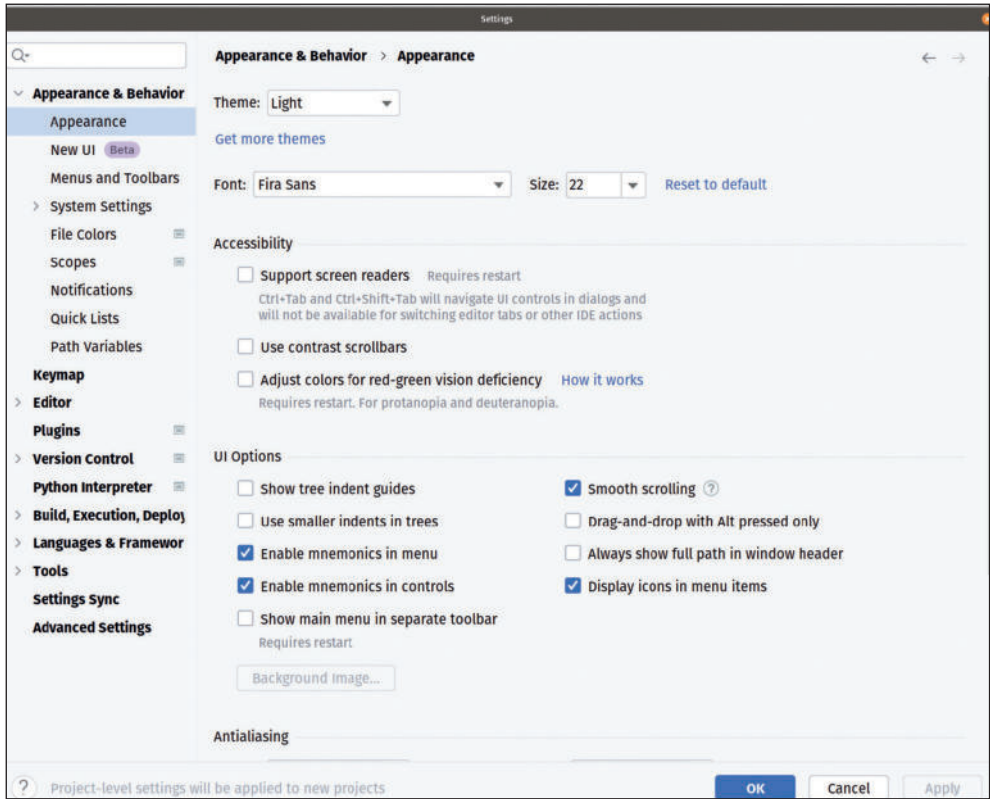


Рис. 2.12. Настройки внешнего вида для PyCharm

Некоторые из этих настроек мы уже видели. Мы можем установить тему так же, как в более простом диалоговом окне настроек, которое мы видели ранее. Однако в этом окне вы найдете ссылку с надписью **Get more themes**, которая позволит вам изучить рынок JetBrains в поисках тем, выходящих за рамки стандартной проблемы, возникающей при установке PyCharm. О плагинах мы поговорим ближе к концу книги, но, если вы хотите найти что-то более уникальное, не помешает осмотреться.

Подождите!

Здесь вы не устанавливаете шрифт, который отображается в редакторе кода. Этот параметр управляет шрифтами самого пользовательского интерфейса. Это повлияет на кнопки, пункты меню, заголовки окон и т. д., а не на редактор, который, вероятно, вы и хотите изменить.

Не волнуйтесь: мы настроим шрифт редактора буквально через минуту.

Настройки редактора

Как и в большинстве разделов окна **Settings**, здесь имеется множество параметров настройки. Я просто останавлиюсь на нескольких основных моментах. Проходя по экранам, вы увидите, пределы возможности настроек. Если это всего лишь пиксель на вашем экране, отображаемый с помощью PyCharm, вы, вероятно, сможете каким-то образом настроить и его! На рис. 2.13 показан раздел **Editor** окна **Settings**:

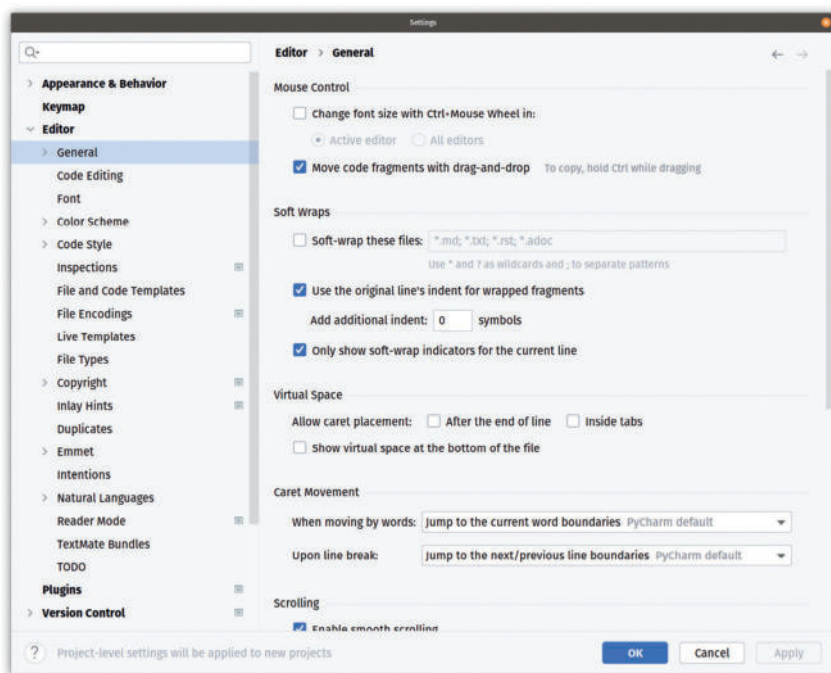


Рис. 2.13. Настройки редактора в окне настроек PyCharm

Если хотите, просмотрите варианты слева и посмотрите, есть ли смысл что-то изменять. Раздел **Color Scheme** позволяет настроить цвет синтаксиса в среде IDE. Я собираюсь перейти к настройке, которую считаю наиболее полезной для изменения. Кликните раздел **Font**, как показано на рис. 2.14.

Этот экран выполняет одну функцию: управляет внешним видом текста в окне редактора IDE. Остальные настройки шрифта, которые мы видели, обычно применяются ко всей IDE. Этот параметр является наиболее важным для нас, пользователей инструмента. Это настройка той части инструмента, на которую вы будете смотреть весь день. Вы можете установить шрифт, размер и высоту строки. У меня установлен большой размер шрифта, потому что мне нужны красивые, большие шрифты для всех этих снимков экрана, но, честно говоря, мой ежедневный драйвер на рабочем ноутбуке установлен на 16 или 18. Шрифты большего размера легче читать в зависимости от того, насколько далеко ваш монитор. В идеале он не должен быть намного дальше, чем на расстоянии вытянутой руки. Если размер вашего шрифта слишком мал, вы будете неосознанно щуриться на монитор, и это может привести к утомлению.

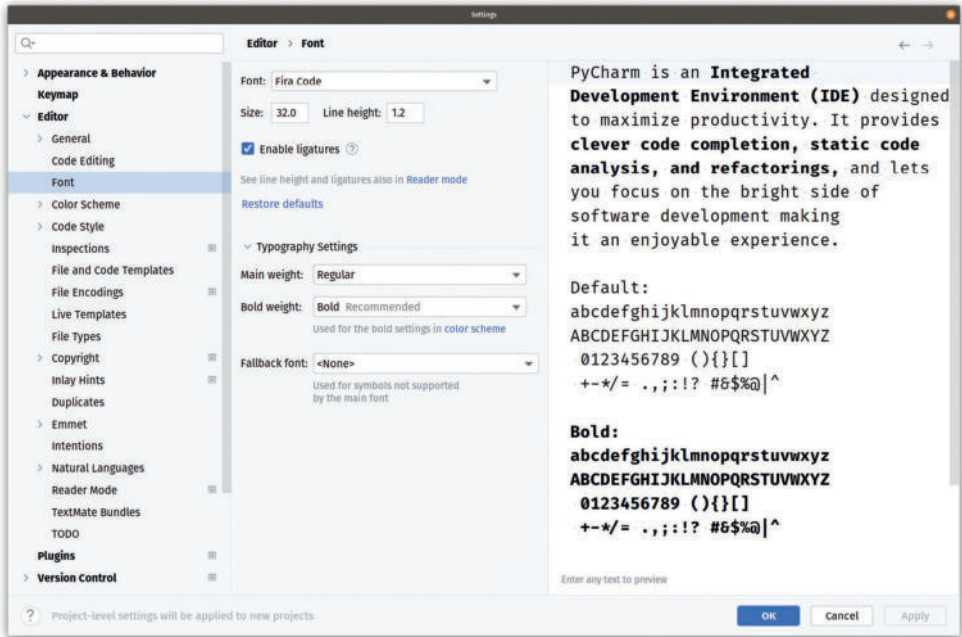


Рис. 2.14. Настройки шрифта редактора в PyCharm

Здесь есть настройка включения лигатур¹ шрифтов. Это усовершенствования стандартных шрифтов, позволяющих символам, которые мы используем в нашем коде, выглядеть более элегантно. Например, если включены лигатуры шрифтов, условное выражение со знаком «не равно», например `if a != b`, будет отображаться, как показано на рис. 2.15:

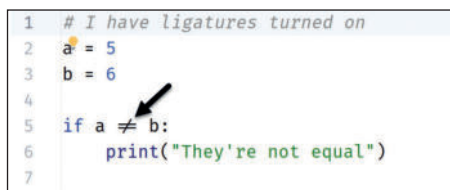


Рис. 2.15. Включение лигатур шрифта позволяет отображать определенные символы, которые обычно недоступны, например символ неравенства в строке 5

Чтобы это работало, вам необходимо выбрать шрифт, поддерживающий лигатуры. PyCharm поставляется со шрифтом *Fira Code* именно потому, что это хороший шрифт IDE, поддерживающий лигатуры. Если вы хотите изучить больше шрифтов с лигатурами, откройте в браузере <https://nerdfonts.com>. На этом сайте есть множество шрифтов, подходящих для редакторов, терминалов и консолей, многие из которых поддерживают лигатуры.

¹ Лигатура – знак любой системы письма, образованный путем соединения двух и более графем. – Прим. ред.

Ниже настроек **Font** в настройках **Editor** находится **Color Scheme**. Вы уже видели это раньше, но приятно знать, где оно находится, поскольку в последний раз мы его видели в окне настройки.

Если вам нужны дополнительные настройки цветовой схемы, передвиньте курсор вниз рядом с пунктом меню **Color Scheme**, и вы увидите, как я это вижу на рис. 2.16, что существует гораздо больше опций:

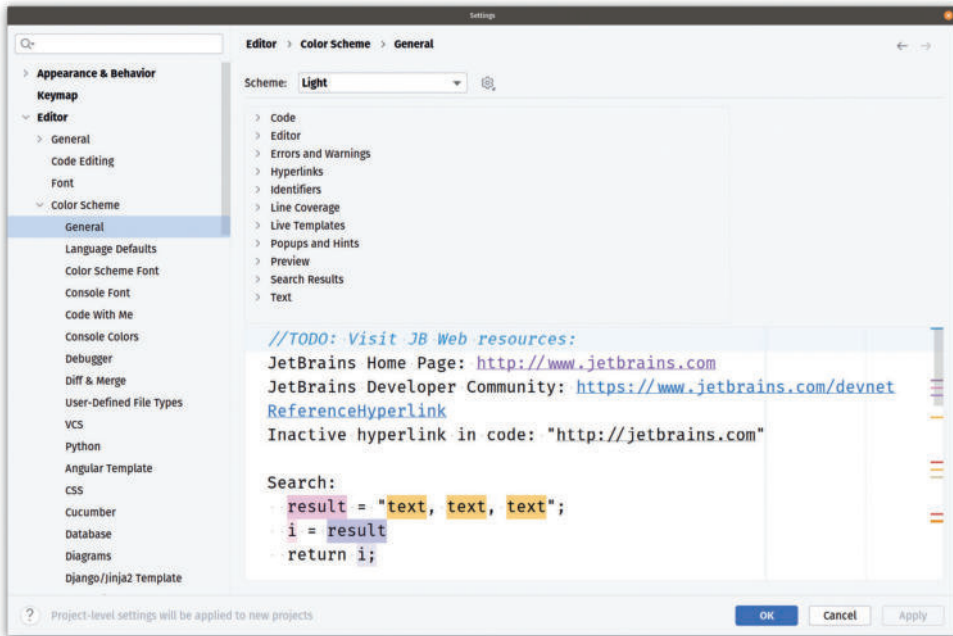


Рис. 2.16. Вы можете изменить цвет практически всего в PyCharm, используя параметры в меню настроек цветовой схемы

Вы можете управлять цветами каждого окна приложения в каждом контексте, в котором они используются. Например, у вас могут быть разные цветовые схемы для файлов SQL, файлов JavaScript и файлов Python.

Возможно, самая полезная настройка в этом лоте – это возможность настроить шрифт, который отображается во встроенном окне терминала PyCharm. Вы можете видеть это на рис. 2.17.

По умолчанию консоль использует тот же шрифт, что и редактор. Если вы хотите это изменить, можете это сделать. Можно дополнительно настроить внешний вид консоли, используя настройки **Console Colors**, как показано на рис. 2.18:

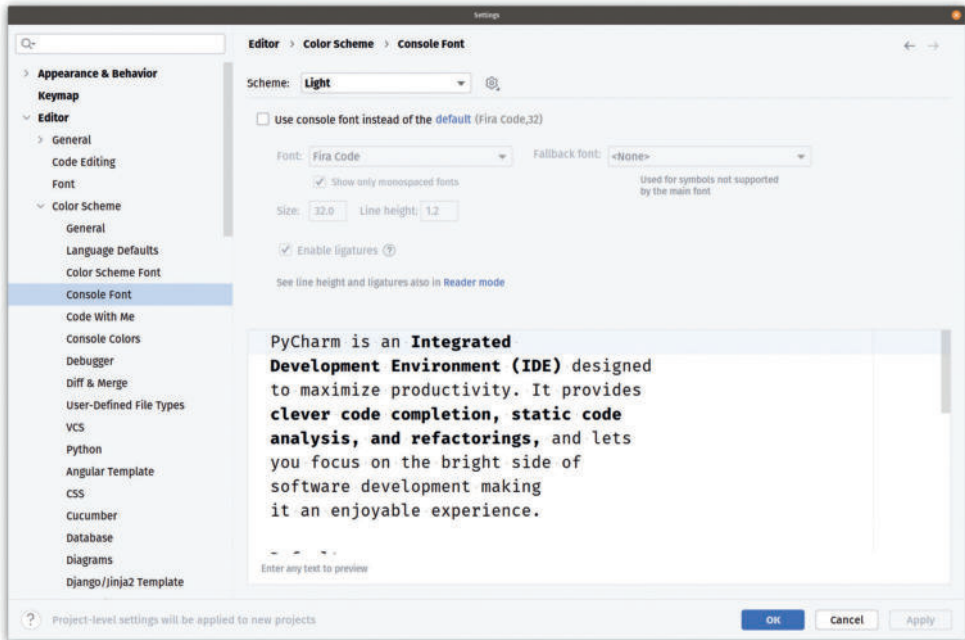


Рис. 2.17. Вы можете настроить шрифт, который отображается во встроенном окне терминала PyCharm

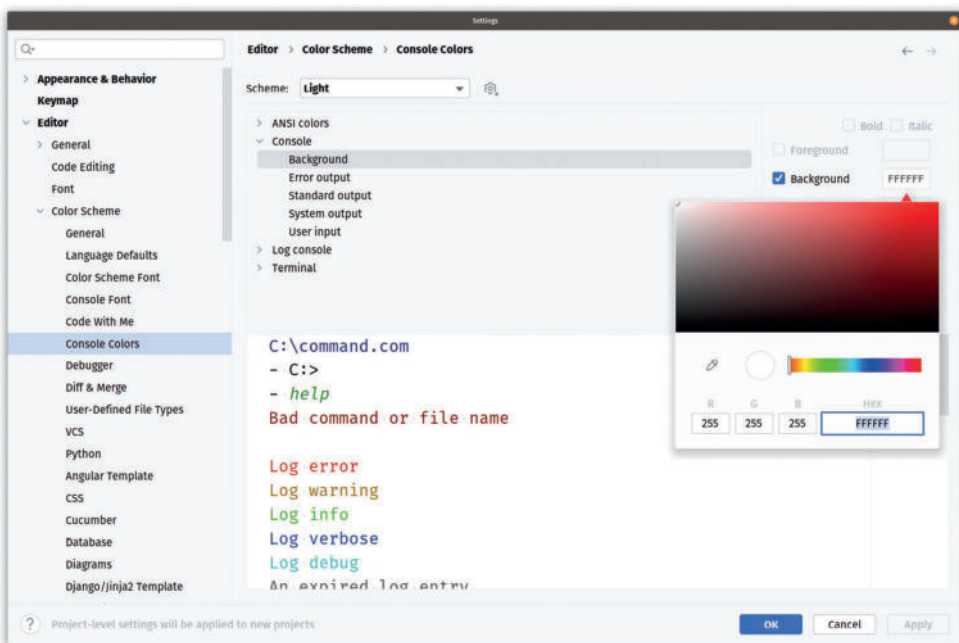


Рис. 2.18. Настройки цветов консоли в PyCharm

Лично я провожу в терминале много времени, поэтому ценю такой уровень доступных настроек.

Закроем меню **Color Scheme** и перейдем к меню **Code Style**. Вы можете увидеть меню на рис. 2.19:

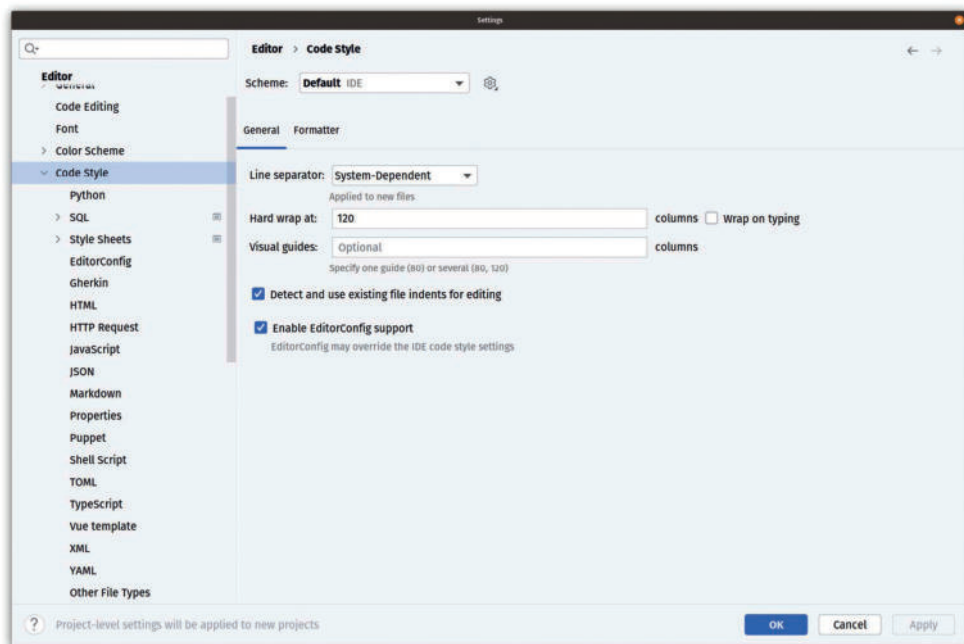


Рис. 2.19. Меню стиля кода Code Style в окне настроек PyCharm

Здесь я нашел нечто, что меня несколько озадачило. Стандарт *PEP 8* определяет, что ни одна строка Python не должна быть длиннее 79 символов, однако в редакторе установлен принудительный перенос после 120 символов. К счастью, мы можем это исправить. Как я уже сказал, *PEP 8* говорит, что строки кода должны быть ограничены 79 столбцами или меньше. Далее стандарт рекомендует ограничивать строки комментариев и документации 72 столбцами. Чтобы убедиться, что я соблюдаю эти правила, я меняю настройки **Code Style**, как показано на рис. 2.20.

Я изменяю настройку **Hard Wrap at** на 79 и проверяю перенос, устанавливая флажок. Затем я добавляю 72 и 79 к настройке **Visual guides**, чтобы видеть линии в правом гаттере¹. Хорошо написанный код Python всегда соответствует стандартам. Установка этих визуальных ориентиров помогает мне избежать этих надоедливых красных волнистых линий правки. Я их ненавижу.

Если вы не понимаете, что я имею в виду, в Python есть встроенный линтер, который предупредит вас, если вы нарушите правила *PEP 8*. Через некоторое время это становится похоже на надоедливую тещу, сидящую на заднем сиденье, пока вы управляете автомобилем. Установка настроек **Code Style** таким образом позволяет избежать ярости кода, которая неизбежна по отношению к «водителям на заднем сиденье» и, возможно, для свекровей или тещ в общем.

¹ Гаттер (здесь) – промежуток между двумя полосами набора. – Прим. ред.

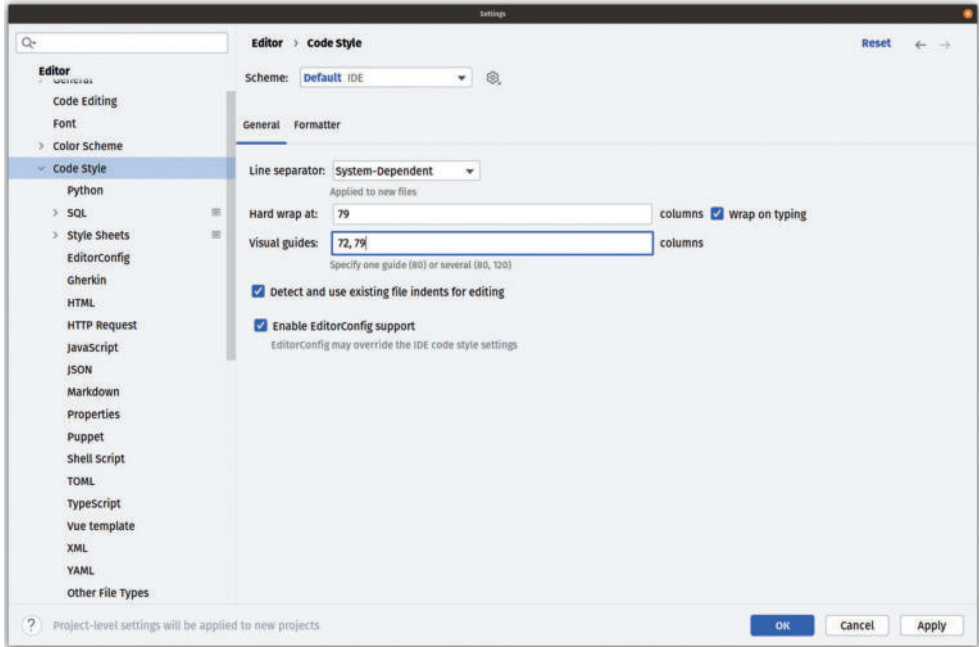


Рис. 2.20. Это мои рекомендуемые настройки для переносов и направляющих, соответствующие PEP 8

Применение этих настроек меняет внешний вид редактора, как показано на рис. 2.21. Справа мы видим две строки, показывающие наши направляющие в 72-м и 79-м столбцах соответственно:

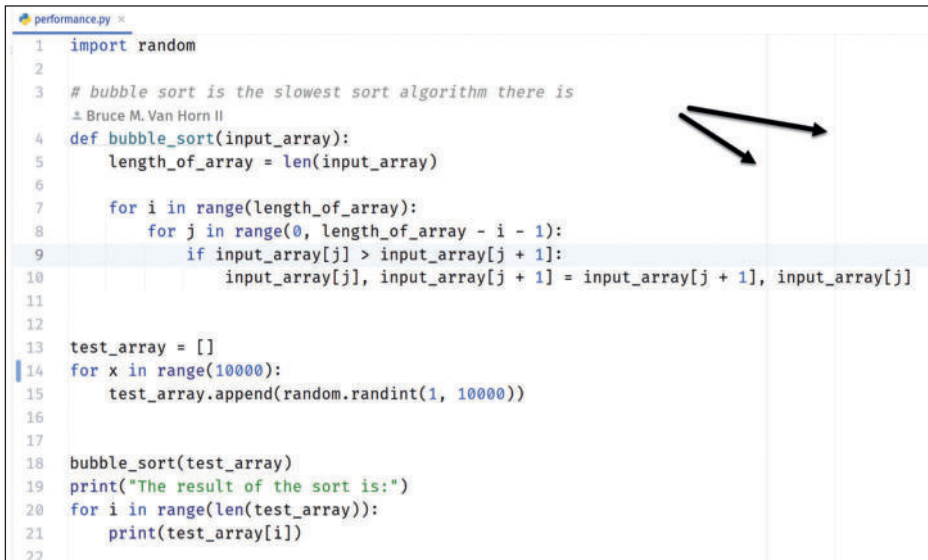


Рис. 2.21. Стиль кода применен, и мы видим две слабые серые линии справа в редакторе

Если вы помните, мы также включили принудительный перенос в строке 79. Чтобы продемонстрировать что получается, обратите внимание, что строка 10 кода, которая была введена до того, как я изменил настройки, явно выходит за рамки нашего руководства по 79 столбцам. Чтобы проверить настройку принудительного переноса, я снова набрал строку сразу под строкой 11. Как только я достиг предела жесткого переноса, PyCharm вставил продолжение строки (\) и перенес меня на следующую строку с соответствующим отступом. Теперь я не могу ввести строку кода, нарушающую стандарт PEP 8.

Обратите внимание, что изменение настроек не приводит к автоматическому переформатированию вашего кода. Есть утилита, которая это делает. Мы узнаем, как автоматически форматировать и переформатировать код, в главе 4.

Другие настройки

В окне **Settings** отображаются дополнительные настройки, включая **Remote Development** и **Plugins**. Они будут рассмотрены в последующих главах.

Экспорт индивидуальных настроек

Индивидуальные настройки – это здорово! Но всегда тяжело переключиться на новый компьютер. Вам бы не хотелось переделывать все ваши настройки. С другой стороны, может быть, вы захотите поделиться своими настройками с другими разработчиками в вашей команде. PyCharm делает это легко. Просто кликните меню **File** и найдите параметр **Manage IDE Settings**, как показано на рис. 2.22:

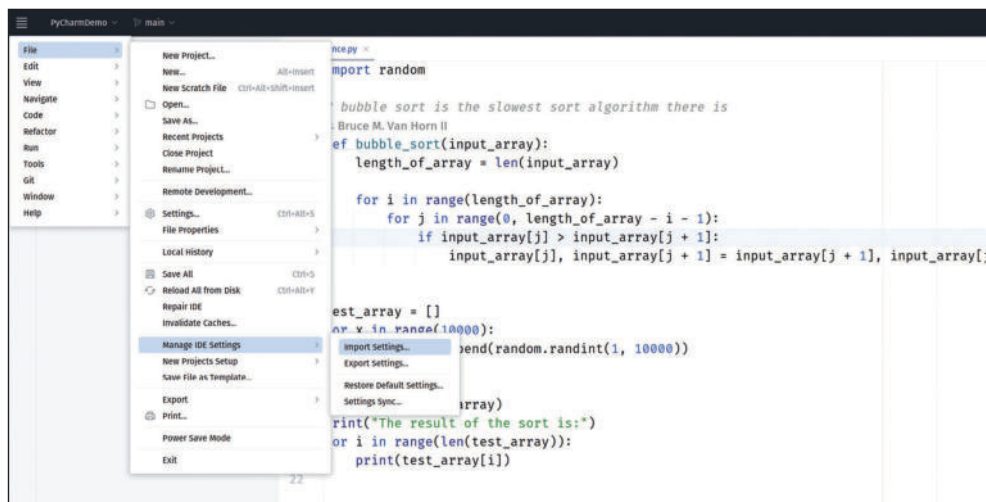


Рис. 2.22. PyCharm позволяет экспортировать и импортировать настройки. Вы можете импортировать, экспортировать и сбрасывать настройки по своему усмотрению

РАБОТА С ПРОЕКТАМИ

Теперь, когда мы рассмотрели все параметры конфигурации и настроили некоторые из наиболее популярных настроек, давайте быстро пробежимся по

PyCharm. Мы создадим быстрый проект Python и научимся запускать наш основной скрипт в пользовательском интерфейсе.

На этом этапе вам потребуется установить Python 3 на ваш компьютер. В документации Python по адресу <https://docs.python.org/3/using/index.html> объясняется, как надо производить инсталляцию. Я считаю, что эта документация является полной, но вам придется многое просмотреть, чтобы выйти на оперативный простор. Чтобы облегчить задачу новичкам, я включу несколько дополнительных ссылок в раздел «Дальнейшее чтение» в конце этой главы.

Создание нового проекта

Вы, вероятно, начинаете понимать, что PyCharm – довольно мощный и гибкий инструмент. Есть несколько способов его настройки и несколько мест в приложении, откуда можно запустить процесс настройки. Аналогичная гибкость существует и при создании проектов.

Если вы никогда раньше не создавали проект в PyCharm (а это, скорее всего, на данный момент так и есть), вы увидите окно, подобное тому, которое показано на рис. 2.23. В этом окне имеется кнопка для создания нового проекта:

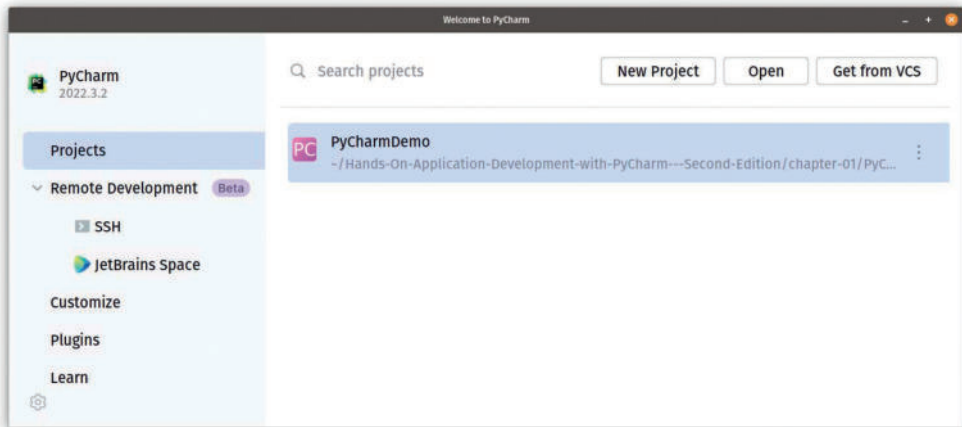


Рис. 2.23. На начальном экране PyCharm есть кнопка, которую можно использовать для создания нового проекта

Если вы уже создали проект в PyCharm, откроется окно PyCharm с последним загруженным проектом. Именно этот случай я показываю на рис. 2.24. Здесь у меня есть открытый демонстрационный проект для наших примеров. Чуть позже в этой главе я покажу вам, как клонировать этот код с помощью встроенного клиента Git PyCharm, но давайте вернемся к созданию нового проекта. На рис. 2.24 у меня уже открыт проект. В этом случае вы можете использовать меню **File** для создания нового проекта. Нажмите **File | New Project...**:

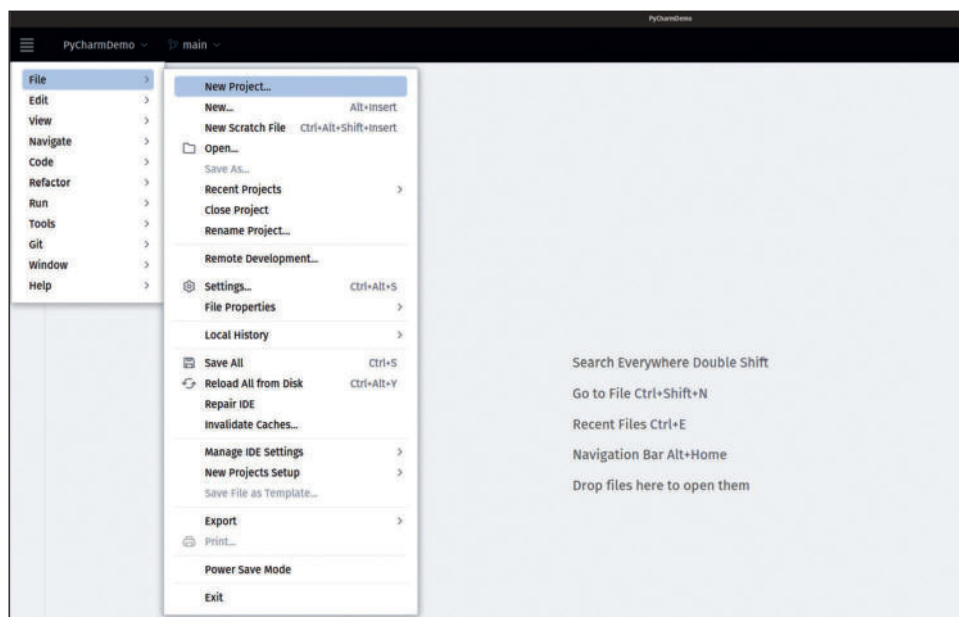


Рис. 2.24. Вы можете использовать меню File для создания нового проекта, кликнув File | New Project...

Независимо от того, какой вектор вы выберете, результат будет одинаковым. Вы получите окно **New Project**, показанное на рис. 2.25:

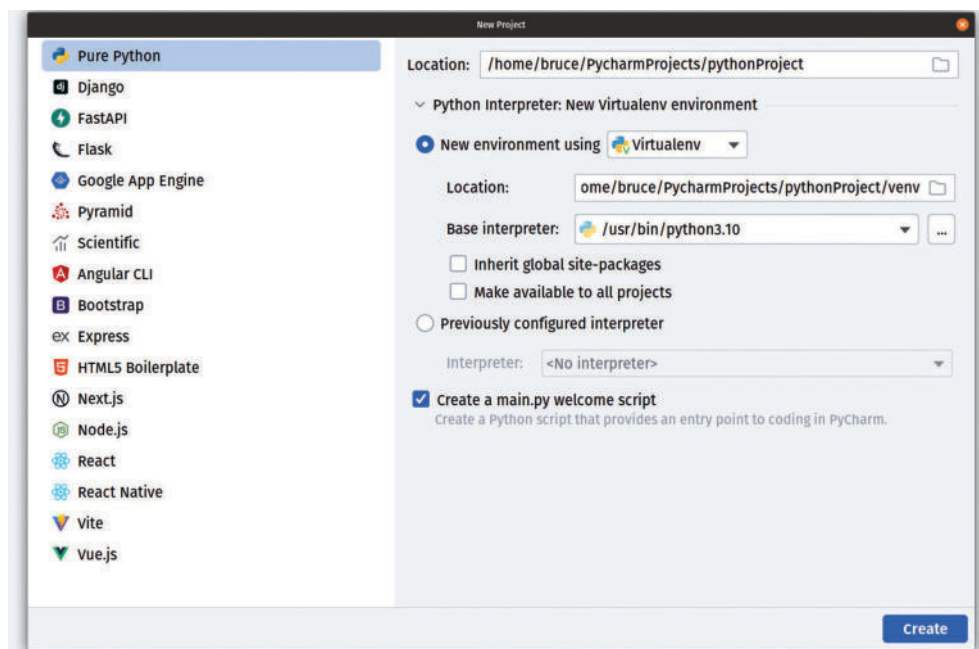


Рис. 2.25. Окно New Project в PyCharm Professional

Здесь обратите внимание на большую разницу между версией Professional и версией Community. На рис. 2.26 вы увидите окна **New Project** для версии Professional (слева) и версии Community (справа). Сосредоточьте свое внимание на левой стороне окна профессиональной версии. Профессиональная версия предлагает гораздо больше типов проектов по сравнению с всего лишь одним вариантом в бесплатной версии. Версия Community создает только проекты на чистом Python. Она даже не дает вам выбора, поскольку это все, что она может сделать:

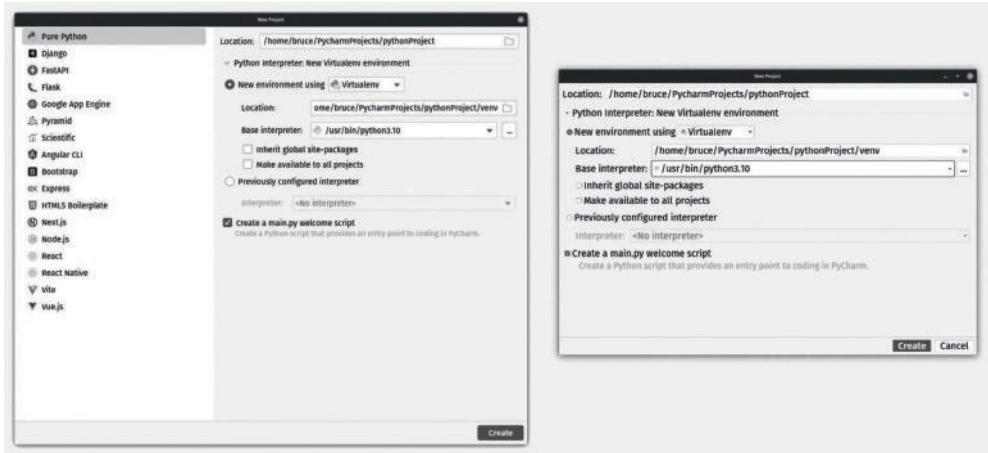


Рис. 2.26. Параметры создания проекта в PyCharm Professional (слева) рядом с версией Community (справа)

Важно отметить, что в версии PyCharm Community невозможно создавать проекты любого типа. Например, если я хочу создать проект Flask для создания веб-приложения, я, в принципе, могу сделать это в версии Community. Однако мне придется делать это вручную, с нуля. Версия Professional ускоряет работу благодаря улучшенным инструментам, но не мешает вам делать то, что вы не могли бы сделать в любом другом редакторе, бесплатном или платном.

Я собираюсь выбрать проект, общий для обеих редакций: проект на чистом Python. Это делается стандартным образом по умолчанию в обеих редакциях.

Первая опция в верхней части экрана позволяет вам установить имя и локацию вашего проекта. Я собираюсь оставить значение по умолчанию в `pythonProject`. Помимо этого, есть множество вариантов создания виртуальной среды Python. Это тема главы 3, поэтому я пока оставлю все по умолчанию.

Примечание

Если вы еще не установили Python 3 или PyCharm не может обнаружить вашу установку Python 3, вам может быть предложено установить Python 3 автоматически. Имейте в виду, что PyCharm может не установить вам последнюю версию Python 3 и не даст вам никакого контроля над тем, где она установлена. Это также значительно увеличит время, необходимое для создания вашего первого проекта.

Удостоверьтесь, что флажок установлен на **Create a main.py welcome script** (Создать скрипт приветствия main.py), и нажмите кнопку **Create**. Если у вас уже был открыт проект, когда вы начали процесс создания нового проекта, вы увидите диалоговое окно с вопросом, что вы хотите сделать с открытым в данный момент проектом, как показано на рис. 2.27:

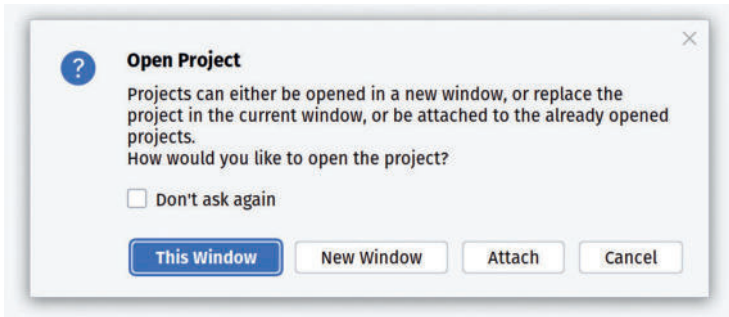


Рис. 2.27. Если у вас уже открыт проект, у вас есть варианты того, что вы хотите с ним сделать при создании нового проекта

А пока давайте выберем кнопку **This Window**, которая закроет открытый проект и заменит его новым проектом. Естественно, ваш старый проект остается нетронутым, и его можно снова открыть с помощью команды **File | Open** меню.

В результате получается новый проект Python 3 с одним файлом. Файл будет содержать вариант вездесущей идиомы «Hello, World», что избавит меня от необходимости включать ее в исходный код книги. Понятно? Мы все победим! Давайте посмотрим на наш новый проект в IDE на рис. 2.28:

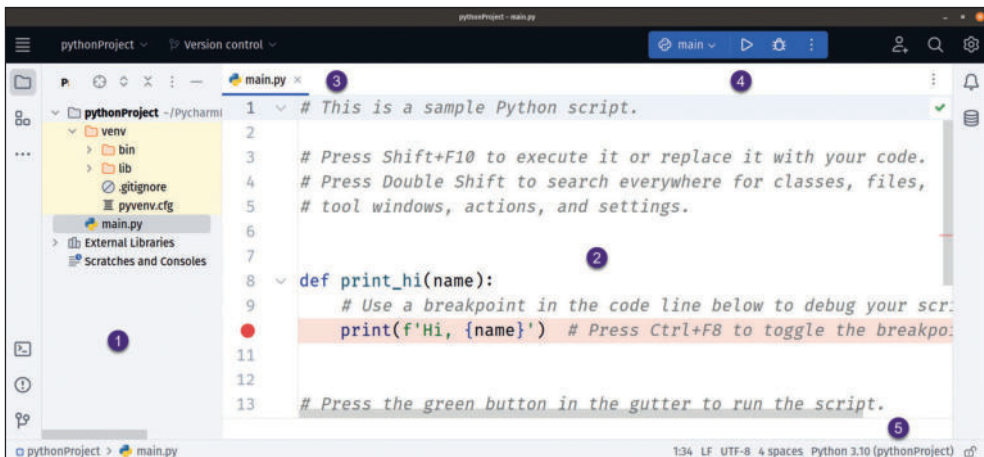


Рис. 2.28. Новый проект создан; теперь давайте осмотримся!

Как видите, PyCharm создал для нас структуру проекта в проводнике проектов (1). Обзорщик проекта позволяет вам видеть и взаимодействовать со всеми файлами вашего проекта. Вы можете видеть, что он создал несколько папок для виртуальной среды. Мы подробно рассмотрим это в главе 3.

Редактор хорошо виден посередине (2). Мы потратим здесь много времени, поэтому отметим, что PyCharm, как и большинство IDE и редакторов, использует интерфейс с вкладками (3). Есть панель инструментов (4) для запуска, отладки, тестирования и профилирования вашей программы, которой мы вскоре воспользуемся.

В правом нижнем углу (5) вы можете увидеть несколько важных настроек, касающихся текущего открытого файла и самого проекта. Мы видим, что текстовая кодировка файла – UTF-8, поскольку я использую Windows, а мой символ новой строки – это **перевод строки (carriage return line feed, CRLF)**. Вероятно, это будет отличаться в macOS и Linux. Мы настроены использовать четыре пробела для стандартных отступов, что соответствует спецификациям *PEP 8*. Когда вы нажмете *Tab* на клавиатуре, это будет интерпретировано как четыре пробела, что положит конец битве между тем, что лучше. Я скажу это еще раз. Вы не сможете ввести реальный символ табуляции (`\t`) в PyCharm, если не настроите его соответствующим образом. Но с какой стати вам пойти и сделать такое?

Совет

Если PyCharm не отвечает, посмотрите вниз в нижнюю часть окна рядом с *area 6*, и вы, вероятно, увидите индикатор выполнения, указывающий, что PyCharm выполняет какую-то операцию, например индексацию файла проекта. Возможно, вам придется подождать, пока он завершится, прежде чем ваша IDE начнет полностью реагировать.

Запуск проекта PyCharm

Я чувствую себя назойливым продавцом автомобилей, который не умолкает о достоинствах машины. Все, что вам хочется сделать, – это проехать на нем по реальной дороге. Давайте я принесу вам ключи и скажу моему несуществующему менеджеру, что у нас рыба на крючке. Я скоро вернусь.

В случае PyCharm клавиши находятся в верхней части окна: *area 4*, показанная на рис. 2.28. Вы можете легко найти ее – это синяя кнопка со стрелкой на ней. Я покажу вам, какая это кнопка, на рис. 2.29. Вам придется довериться мне в отношении цвета. Это синий. Это кнопка **Run**. Нажмите на нее:

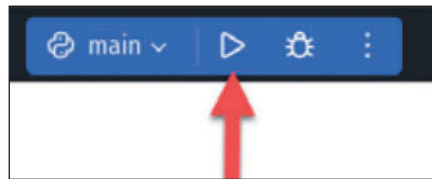


Рис. 2.29. Нажмите синюю стрелку, чтобы запустить основной скрипт

А теперь выпустите большой старый техасский ЙИИИ-ХУ! Или можете просто делать все, что можно назвать криком или улюлюканьем, где бы вы ни ро-

дились. Вы только что запустили свою программу. Мы сделали большой шаг к освоению новой IDE.

Окно IDE изменится, как показано на рис. 2.30:

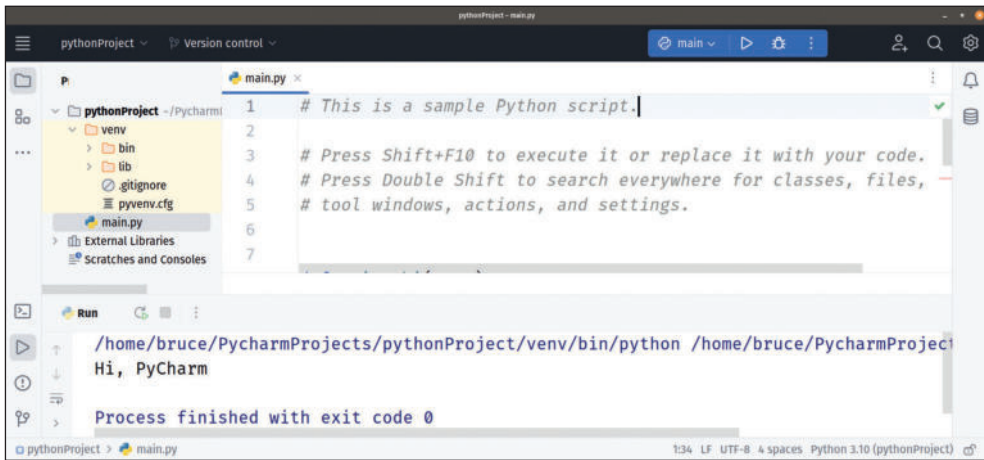


Рис. 2.30. Когда вы запускаете программу, можете увидеть результат вашего запуска на консоли

Помимо самого окна запуска, обратите внимание, что на вкладке имеются кнопки, позволяющие перезапустить программу или остановить запущенную в данный момент программу.

В следующем разделе вы узнаете, как клонировать проект кода с GitHub. Прежде чем мы это сделаем, давайте закроем текущий проект. Найдите меню **File** и выберите **File | Close Project**, как показано на рис. 2.31:

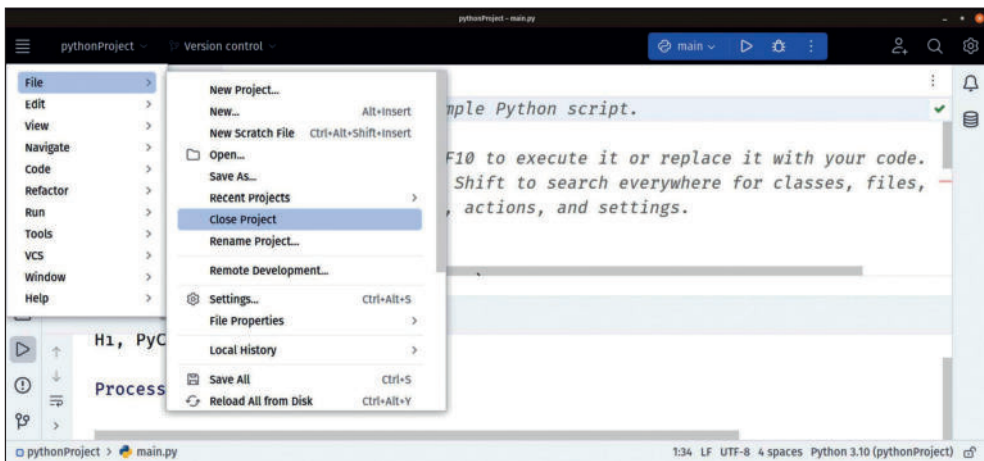


Рис. 2.31. Закройте проект из меню File

Отлично! Вы готовы к следующему приключению! Давайте возьмем пример исходного кода книги с GitHub.

Клонирование кода этой книги с GitHub

PyCharm имеет надежный набор функций для работы с такими системами контроля версий, как Git, SVN, Mercurial и Perforce. На протяжении всей этой книги мы будем работать с Git и GitHub, поскольку они стали фактическим стандартом в отрасли. Если вы используете одну из других поддерживаемых систем контроля версий, процесс в основном тот же. Фактически пользовательский опыт в основном такой же, за исключением различий в работе некоторых систем контроля версий. Например, Git использует четырехэтапный процесс для фиксации и отправки файлов.

1. Внесите локальные изменения (`git add`).
2. Зафиксируйте изменения локально (`git commit`).
3. Извлеките данные из GitHub или из центрального репозитория, чтобы убедиться в наличии последних изменений и устранить любые конфликты (`git pull`).
4. Отправьте изменения в центральный репозиторий (`git push`).

Напротив, SVN имеет только два шага.

1. Загрузите последнюю версию и исправьте все конфликты (`svn update`).
2. Зафиксируйте локальные изменения в центральном репозитории (`svn commit`).

Я хочу сказать, что разные системы контроля версий могут иметь разные рабочие процессы, которые меняют взаимодействие с пользователем в PyCharm. В этой книге мы будем использовать только Git.

Поскольку вы закрыли свой проект в конце последнего раздела, вы должны увидеть окно, похожее на то, что показано на рис. 2.32:

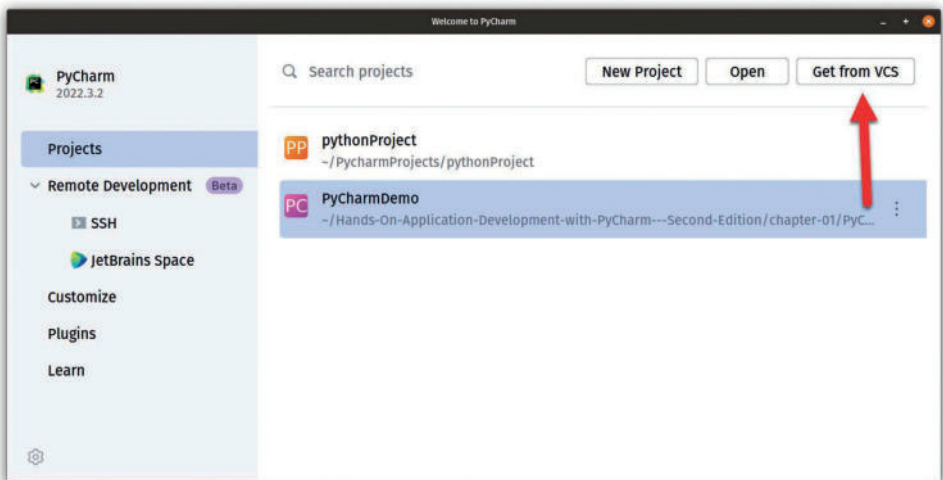


Рис. 2.32. Найдите кнопку Get from VCS

В этом разделе нужно выполнить две задачи. Сначала мы собираемся настроить вашу учетную запись GitHub в PyCharm. Это не является строго обязательным требованием, если у вас в настоящее время нет учетной записи, но рано или поздно в вашей карьере вам понадобится учетная запись GitHub. С тем же успехом вы могли бы сорвать этот пресловутый пластырь¹. Я не буду здесь рассказывать о том, как создать учетную запись GitHub. Инструкции для этого можно найти на <https://github.com/signup>.

Найдите и нажмите кнопку **Get from VCS**. Вы увидите экран, подобный показанному на рис. 2.33:

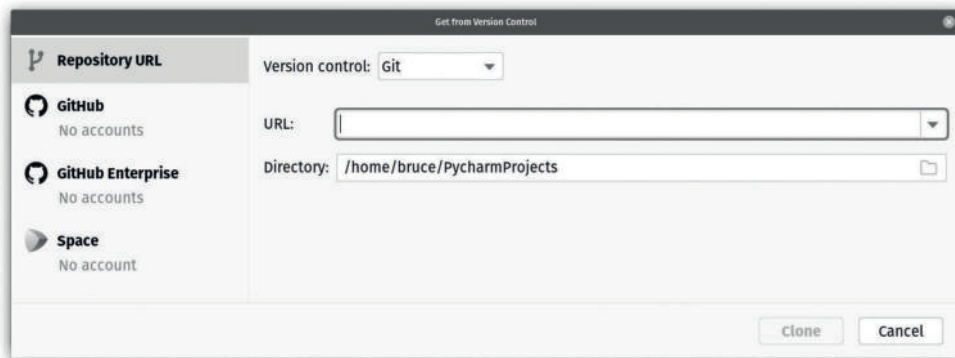


Рис. 2.33. Окно Get from Version Control в PyCharm

Настройка учетной записи GitHub

Обратите внимание, что в правой части окна вы можете увидеть все свои учетные записи GitHub, GitHub Enterprise и JetBrains Space. Я собираюсь продолжить и настроить свою учетную запись GitHub. Кликните элемент **GitHub** в меню слева. Вы увидите экран, подобный показанному на рис. 2.34:

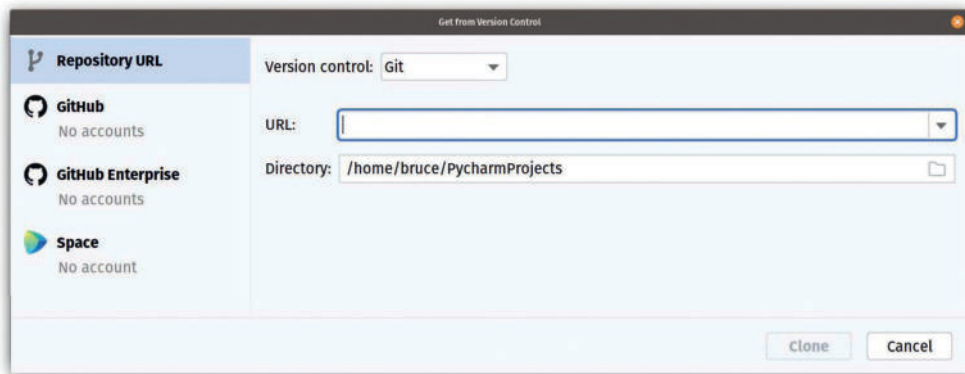


Рис. 2.34. Вход в GitHub прост, его нужно выполнить только один раз

¹ Сделать что-то болезненное или неприятное, но очень быстро. – Прим. ред.

Найдите и нажмите кнопку **Login via GitHub**. Прежде чем нажать на нее, я рекомендую вам войти в GitHub по адресу <https://github.com/login>, используя ваш любимый браузер. Этот процесс будет более простым, если вы уже вошли в систему и преодолели все препятствия **двухфакторной аутентификации (2FA)**.

После авторизации вы можете закрыть браузер и вернуться в PyCharm. Через несколько минут вы увидите в окне все свои репозитории GitHub. Это значительно упрощает работу с собственными репозиториями. Однако репозитория, который мы собираемся клонировать, нет в вашем списке, поэтому нам нужно еще немного поработать.

Клонирование репозитория книги

Чтобы клонировать репозиторий, которым вы не владеете, вернитесь к параметру **Repository URL** в левом меню, как показано на рис. 2.35:

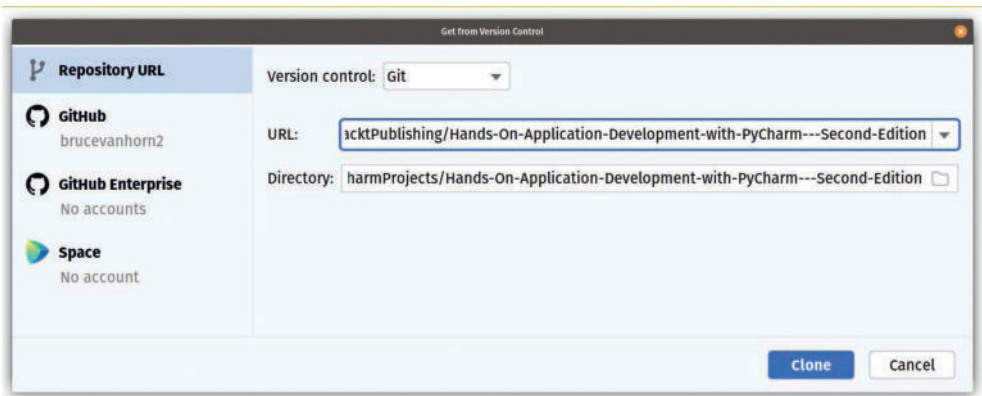


Рис. 2.35. Чтобы клонировать репозиторий, которым вы не владеете, выберите параметр Repository URL в левом меню

В разделе **Repository URL** вам нужно будет ввести URL: <git@github.com:PacktPublishing/Hands-On-Application-Development-with-PyCharm---Second-Edition.git>. Здесь я использую адрес SSH. Если вы предпочитаете использовать версию HTTPS, используйте это: <https://github.com/PacktPublishing/Hands-On-Application-Development-with-PyCharm---Second-Edition.git>. Выберите папку для хранения вашего клона. По умолчанию он находится в вашей домашней папке PyCharmProjects. Я оставил свой по умолчанию. Нажмите кнопку с надписью **Clone**.

После клонирования вы получаете обычное дурацкое диалоговое окно, широко распространенное сегодня в инструментах разработки программного обеспечения, как показано на рис. 2.36:

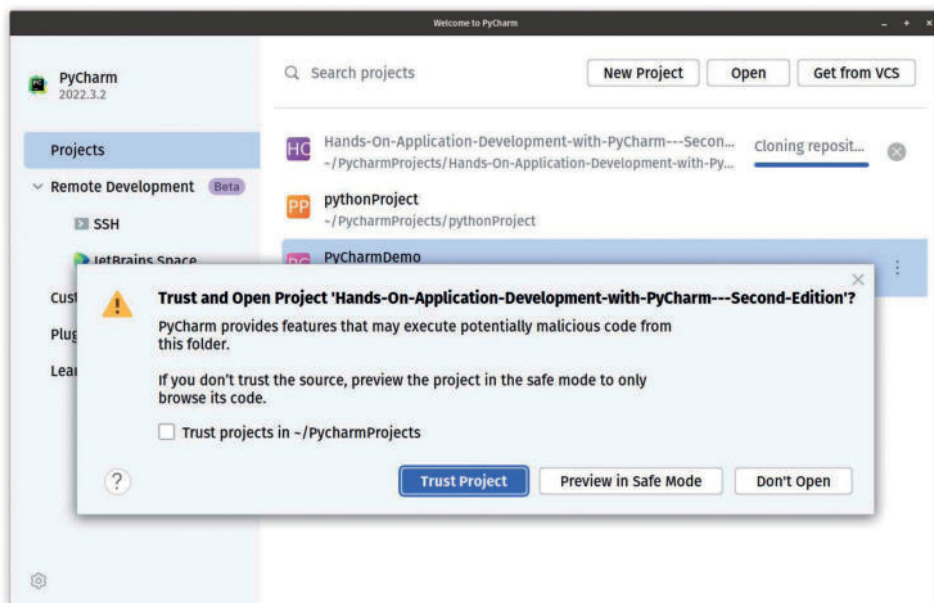


Рис. 2.36. Вы мне доверяете?

Я знаю, что мы только что встретились и все такое, но, если вам действительно нужен исходный код книги, следует нажать кнопку **Trust Project**. Как только вы это сделаете, все готово. Теперь у вас есть весь пример кода, показанный в книге.

КРАТКОЕ СОДЕРЖАНИЕ

Вся эта глава была посвящена предварительным действиям. Мы установили и настроили ваш новый экземпляр PyCharm на вашем компьютере. Существует несколько способов установки PyCharm. Вы можете пойти традиционным путем и просто загрузить установщик. Я предпочитаю установить приложение JetBrains Toolbox и использовать его для установки и управления PyCharm, а также любых других инструментов JetBrains, которые мне могут понадобиться или кие я хочу. Использование приложения Toolbox позволяет нам легко обновить PyCharm или даже выполнить полное удаление, если возникнет такая необходимость.

Мы узнали о различиях между тремя версиями PyCharm. Бесплатная версия Community ограничена в типах проектов, которые PyCharm поддерживает со встроенными инструментами. Когда мы создавали новый проект, нам не было предоставлено никаких вариантов шаблона проекта. Поддерживаются только проекты «чистого Python». Это не означает, что мы не можем создать какой-либо проект; это просто означает, что утомительная часть настройки различных типов проектов не выполняется за вас IDE.

Версия Professional поддерживает более широкий набор инструментов, включая полный набор шаблонов проектов и расширенные инструменты для

веб-разработки и научных задач. Хотя версия Professional не является бесплатной, существуют варианты лицензирования для частных лиц и отдельная цена для корпоративных разработчиков. Также существуют варианты получения бесплатной профессиональной лицензии путем подачи заявления. Примеры включают Microsoft MVP и других признанных специалистов, разработчиков открытого исходного кода и профессоров университетов. На веб-сайте JetBrains часто проводятся скидки для стартапов, образовательных учреждений и других организаций. К сожалению, авторы книг не указаны.

Третья версия PyCharm – образовательная. Это специальная версия, позволяющая создавать и воспроизводить интерактивные уроки прямо в IDE. Если вы преподаете Python с помощью PyCharm, вам, вероятно, это не понадобится. Лучше использовать версию Community или Professional и воспользоваться предложением бесплатного программного обеспечения JetBrains для учителей.

После установки PyCharm мы приступили к настройке приложения. Был выделен ряд важных опций конфигурации. Мы можем настроить практически каждый аспект работы PyCharm. Популярные настройки включают настройку цветовой темы IDE, размера шрифта и параметров разработки кода. В нашем случае мы настроили наш редактор на использование шрифта с лигатурами под названием Fira Code, который поставляется с PyCharm. Мы также настроили параметры **Code Style** для выполнения принудительного переноса на 79-м символе, чтобы обеспечить соответствие стандартам *PEP 8*.

После того как все настроили, мы приступили к созданию проекта на чистом Python и запустили его с помощью кнопки **Run** в среде IDE.

Наконец, мы настроили PyCharm с нашей учетной записью GitHub и клонировали репозиторий примеров кода этой книги. Мы успели многое сделать за короткий промежуток времени. На этом этапе у вас все готово для разработки программного обеспечения с использованием PyCharm, а мы только переходим к главе 2!

Оставайтесь со мной, и мы углубимся в работу с виртуальными средами Python в следующей главе. Виртуальные среды считаются лучшей практикой и позволяют разделить требования ваших проектов в локальной среде разработки. Как вы увидите, PyCharm избавляет от необходимости запоминать около полудюжины команд, а также устанавливать дополнительные библиотеки для более быстрого запуска вашего проекта.

Вопросы

1. Каковы преимущества использования приложения JetBrains Toolbox для установки PyCharm и управления ими?
2. Каковы основные различия между версией Community и профессиональной версией PyCharm?
3. Каковы основные ограничения сообщества PyCharm?
4. Если вы используете версию PyCharm Community, можете ли вы по-прежнему разрабатывать проекты с использованием таких фреймворков, как Flask, которые, по-видимому, поддерживаются только в версии Professional?

5. Каковы преимущества привязки вашей учетной записи GitHub к PyCharm?
6. Вы уже зашли в репозиторий книги и поставили ей звезду? Если вы создадите форк репозитория, то получите уведомление, если я когда-нибудь что-нибудь изменю. Я сомневаюсь, что это произойдет, поскольку, как и ваш, мой код часто оказывается идеальным с первой попытки. Неа. Это была шутка. Не могу сохранить серьезное выражение лица. Но я пытался.

ДАЛЬНЕЙШЕЕ ЧТЕНИЕ

- Инструкции по настройке Python: <https://www.maddevskilz.com/pages/python-landing-page>.
- PEP 8 – Style Guide for Python Code: <https://peps.python.org/pep-0008/#maximum-line-length>.

Часть II

Повышение производительности

В этой части рассматриваются концепции и методы от начинающих до продвинутых, которые повышают производительность при работе над проектами Python в PyCharm. Читатели смогут узнать о динамических параметрах управления проектами, интерпретаторах Python и виртуальных средах, а также о том, как применять хорошие методы программирования, такие как контроль версий, тестирование, отладка и профилирование, а также о том, как PyCharm оптимизирует эти процессы.

Эта часть состоит из следующих глав:

- глава 3 «Настройка интерпретаторов и виртуальные среды»,
- глава 4 «Редактирование и форматирование с легкостью в PyCharm»,
- глава 5 «Контроль версий в PyCharm с помощью Git»,
- глава 6 «Бесшовное тестирование, отладка и профилирование».

Настройка интерпретаторов и виртуальные среды

Данная глава посвящена настройке конфигурации. Параметры конфигурации предназначены для того, чтобы помочь вам настроить рабочую среду в соответствии с проектом, а также обеспечить соответствие ее с вашими личными вкусами. Эти параметры – одна из многих причин, по которым PyCharm – отличный инструмент. Еще одна замечательная функция, которая делает PyCharm полезным, – это настройка интерпретаторов и виртуальных сред.

В этой главе будут рассмотрены следующие темы:

- важность работы с виртуальными средами в Python,
- как создать виртуальную среду вручную с помощью `virtualenv`,
- как создать виртуальную среду с помощью PyCharm,
- как работать с существующей виртуальной средой в PyCharm,
- как добавлять и удалять сторонние библиотеки с помощью PyCharm,
- как импортировать проекты, созданные вне PyCharm,
- как работать с настройками виртуальной среды в конфигурациях запуска.

На этом этапе у многих из вас будет установлен хотя бы один интерпретатор Python. Он либо поставляется с вашей системой, как в случае с macOS или Linux, либо вы его установили, как в случае с Windows. Python – это **интерпретируемый язык**. Это означает, что написанный код не оценивается по-настоящему, пока он не будет запущен. Давайте на минутку опишем, что это значит, сравнив Python с несколькими другими языками и тем, как они выполняются.

Существует три распространенных способа выполнения написанного кода на вашем компьютере:

- компиляция,
- интерпретация,
- промежуточная компиляция со средой выполнения.

Прежде чем мы углубимся в это, давайте совершим небольшое путешествие назад во времени. Я знаю, что это уже история, но оставайтесь со мной. В 1522 году преобладающей религией на большей части территории Западной

Европы был католицизм. Это было время великих потрясений в истории церкви, но мы на мгновение оставим это в стороне. Вместо этого сосредоточимся на немецком священнике по имени Мартин Лютер. В те дни немецкий народ в удовлетворении своих духовных потребностей полностью зависел от церковного клира. Библия в 1522 году была доступна только на греческом и латыни. Лютер, находившийся в то время в заключении, перевел Ветхий Завет с греческого на немецкую Вульгату¹. Это изменило правила игры для немцев. Типографии того времени подхватили ее и напечатали сотни экземпляров, чтобы ее могла прочитать каждая семья. Я собираюсь использовать это как аналогию.

Первый метод выполнения написанного кода – *компиляция*. Это похоже на перевод Мартина Лютера с греческого на немецкий. Большинство людей, которые не умеют программировать, увидят язык программирования, почешут затылки и подумают про себя: «Для меня это греческий!» То же самое происходит и с компьютерами, потому что компьютер тоже не может понять ваш язык кода. Компьютеры не «говорят» на C, C++ или Java. Вместо этого программист написал что-то на языке, который проще и легче перевести по сравнению с человеческим языком. Компиляция – это процесс, при котором файлы текстового кода один раз переводятся в двоичный формат, понятный компьютеру. Примеры компилируемых языков включают C и C++. Файлы кода пропускаются через компилятор, который переводит код в отдельный новый формат только один раз. Процесс создает новый файл, отдельный от кода, который полезен только компьютеру. Подводя итог нашей аналогии, отметим, что после перевода с греческого на немецкий печатные книги стали результатом этапа компиляции. Умение читать Библию пригодилось немцам.

Сравните это со второй формой выполнения кода: *интерпретируемым языком*. До перевода Лютера Библию переводили на ходу в церкви. Священник открывал Библию и переводил текст на немецкий язык, читая его вслух. Именно это происходит с интерпретируемыми языками, включая Python. *Переводчиком* в нашем случае является *интерпретатор* Python – исполняемый файл Python, который вы установили на каком-то этапе ранее в процессе установки. Когда вы запускаете выполнение своего кода с помощью такой команды, как `python main.py`, интерпретатор Python открывает файл кода, считывает код, переводит (компилирует) этот код и выполняет инструкции по мере их чтения построчно.

Для тех из вас, кто глубоко знаком с теорией компиляторов, это может показаться чрезмерным упрощением, но для большинства из нас аналогия будет справедливой. Во время процесса Python выполняет дополнительную работу, кешируя и оптимизируя часть перевода, чтобы его можно было использовать снова. По большому счету, каждый запуск программы представляет собой новую интерпретацию. Примеры интерпретируемых языков включают Python, JavaScript и Lua.

¹ Перевод Библии на немецкий язык, сделанный Лютером, был далеко не первым – до 1522 года было множество полных изданий Библии на немецком языке, переведенных с Вульгаты, латинского перевода. Перевод Лютера называют «Библия Лютера», это был первый полный перевод Библии на немецкий язык (включая Новый Завет и апокрифы), в котором использовалась не только латинская Вульгата, но и греческие переводы. Источник – К. Бергер, Оксфордский справочник по теологии, 2014. – Прим. ред.

Последняя категория выполнения кода представляет собой нечто среднее между двумя другими. В таких языках, как C# и Java, используется этап *компиляции*. В отличие от обычных компиляторов результатом не является файл, который запускается сам по себе на «голом железе» вашего компьютера. Вместо этого он создает промежуточный формат, который может быть прочитан **средой выполнения (runtime)**.

Вспомните время, когда вы были студентом. Обычной практикой является то, что профессор или учитель читает лекции в классе, а класс делает записи. Хороший студент может воспроизвести содержание лекции по конспектам. Эти конспекты представляют собой сокращенную форму лекции. В этой аналогии учитель – это программист, а ученик – промежуточный компилятор. Надеемся, что компилятор выдаст великолепный набор заметок, который позволит студенту получить хорошую оценку на экзамене. Ваши файлы кода (лекция) переводятся в компактную двоичную версию кода (ваши заметки). Когда этот промежуточный файл выполняется, среда выполнения, по существу, «просматривает заметки» и позволяет этому коду запускаться на компьютере. Плюсом этого метода является то, что промежуточную компиляцию можно выполнять на различных платформах без необходимости перекомпиляции. Программы, написанные на C, будут работать на различных платформах: от Intel и ARM до старых платформ, таких как мейнфреймы. Однако для этого вам необходимо перекомпилировать программу на этой платформе. Исполняемые файлы запускаются только на платформе, которая их компилирует. Затем промежуточная компиляция запускается на любой платформе, поддерживающей среду выполнения. Вы можете скомпилировать свой код один раз и распространить его на что угодно.

Для наших целей Python – это интерпретируемый язык, и нам нужен интерпретатор. Кроме того, PyCharm необходимо кое-что знать об интерпретаторе. В этой главе рассказывается о средствах, с помощью которых вы можете представиться. PyCharm, встречайте интерпретатора. Интерпретатор, познакомьтесь с PyCharm.

ТЕХНИЧЕСКИЕ ТРЕБОВАНИЯ

Чтобы пройти эту главу, как и остальную часть книги, вам понадобится следующее:

- установленный и работающий интерпретатор Python. Я буду использовать последнюю версию <https://python.org>;
- установленные копии `pip` и `virtualenv`. Вы получаете их автоматически при установке Python в Windows, а в macOS они включены в каждую систему. Если вы используете Linux, необходимо установить менеджеры пакетов, такие как `pip`, и инструменты виртуальной среды, такие как `virtualenv`, отдельно. В наших примерах будут использоваться `pip` и `virtualenv`;
- установленная и рабочая копия PyCharm. Установка была описана в главе 2 «Установка и настройка PyCharm»;
- пример исходного кода этой книги с GitHub. Мы рассмотрели клонирование кода в главе 2 «Установка и настройка PyCharm». Соответствующий код этой главы вы найдете на <https://github.com/PacktPublishing/Hands-On-Application-Development-with-PyCharm---Second-Edition/tree/main/chapter-03>.

Существует несколько разновидностей Python. Я использовал и ссылался на Python 3 с <https://www.python.org>. Помните: поскольку это проект с открытым исходным кодом, можно создать альтернативную версию Python. Это делалось много раз с разной степенью успеха. Некоторые из этих вариантов включают следующее:

- **Anaconda** – это вариант Python, ориентированный на научную работу и работу с большими данными. По сути, это обычный Python с предварительно установленными наиболее популярными и полезными внешними библиотеками, такими как `numpy`, `matplotlib` и `pandas`. Обратной стороной является объем места на ваших дисках, занимаемый этой более крупной установкой, но, учитывая цену хранилища в наши дни, это, вероятно, не имеет большого значения. Эти библиотеки, скорее всего, являются наиболее важным вариантом, который следует учитывать, если вы собираетесь заниматься наукой о данных;
- **IronPython** – это вариант Python, предназначенный для работы в среде выполнения .NET в Microsoft. Последствия этого выходят за рамки этой книги. Однако IronPython интересен тем, что, помимо работы в среде .NET, эта реализация не ограничена **глобальной блокировкой интерпретатора (GIL)**. GIL предотвращают эффективную многопоточность в вашем коде. Если вы не знаете об этом ограничении Python, я оставлю ссылку в разделе «Дальнейшее чтение» этой главы;
- **Jython** – этот вариант позволяет вашему коду Python выполняться на **виртуальной машине Java (JVM)**;
- **MicroPython** – этот вариант используется для запуска кода Python с микроконтроллерами для использования в **Интернете вещей (IoT)** и потенциально встроенных проектах;
- **ActiveState ActivePython** – это коммерчески поддерживаемая реализация Python, в которой особое внимание уделяется совместимости и выполнению Windows. Традиционно Python разрабатывался с учетом того, что он будет работать в средах Unix или Linux. Если вы собираетесь запустить свой код Python в рабочей среде на сервере Windows, можете рассмотреть этот вариант.

Вы найдете все это официально по адресу <https://www.python.org/download/alternatives/>. Хотя любой из этих вариантов должен работать в PyCharm, большинство разработчиков, использующих IDE, работают на стандартном Python (тот, что с [python.org](https://www.python.org)) или Anaconda. Работая с виртуальными средами в Python, вы, несомненно, увидите различные варианты этих альтернативных реализаций. В общем, я всегда использую ванильную версию¹ Python 3.

ВИРТУАЛЬНЫЕ СРЕДЫ

Мы ориентировались на создание проекта в главе 2 «Установка и настройка»; однако мы умолчали о деталях создания или настройки **виртуальной среды**. Виртуальные среды – это особенность Python, а не PyCharm. Виртуальная

¹ «Ваниль» – программное обеспечение или алгоритмы, которые не были настроены или изменены. – Прим. ред.

среда – это копия интерпретатора Python и связанных с ним файлов, специфичная для проекта. Большинство языков программирования позволяют каким-то образом отделить требования одного проекта от любого другого на вашем компьютере. В начале проекта вы обычно создаете виртуальную среду из своей основной, также называемой *глобальной*, установки Python. Преимущество появляется, когда вы работаете над несколькими проектами с разными требованиями. Возможно, для одного проекта требуется Python 3.6, а версия там заморожена, чтобы соответствовать требованиям клиентов по контролю изменений. Или, может быть, нужная вам библиотека, например `matplotlib`, работает только на Python 3.6 на вашем Mac, но в Windows она стабильно работает только на Python 3.9. Пока этот проект выполняется, запускается другой проект, требующий Python 3.10.

Если бы вы работали исключительно с глобальными или общесистемными установками Python, было бы очень сложно переключаться между этими двумя проектами. Вам придется возиться с переменными среды и исправить переменную среды `PATH` вашего компьютера. Затем нужно будет попытаться вспомнить, какие сторонние библиотеки установлены глобально, и надеяться, что не возникнет противоречия между требованиями к этим библиотекам для отдельных проектов. Неприятно.

Виртуальные среды позволяют легко решать эти проблемы. Хотя их использование не является обязательным, рекомендуется создавать виртуальную среду для каждого проекта. Прежде чем мы вернемся к PyCharm, я подумал, что было бы интересно создать виртуальную среду вручную. Вы можете пропустить это, если хотите, но, если вы не сделали этого раньше, я думаю, это даст вам признательность за часть работы, которую PyCharm берет на себя в начале нового проекта.

Создание виртуальной среды вручную

Откройте терминал своего компьютера и найдите место на диске, где вы сможете начать работать. Мы не будем откладывать это на потом, мы просто собираемся пройти через это. Затем переключимся на PyCharm, чтобы увидеть более автоматизированную версию того же рабочего процесса.

Я использую Windows, и в моей домашней папке есть место, где я храню свои проекты; папка просто называется *Projects*. Открыв на своем компьютере терминальную программу, – в данном случае это **Windows Terminal с PowerShell**, – я могу ввести команды для этого эксперимента. Вам не обязательно использовать Windows или PowerShell. Обычное приглашение терминала **zshell (zsh)** в macOS или приглашение оболочки **Bourne Again Shell (Bash)** в Linux работает одинаково. Большинство команд идентичны во всех терминалах и операционных системах. Я начну с создания новой папки следующим образом:

```
mkdir python-environment-demo
```

При этом создается новая папка с именем `python-environment-demo`. Далее мне нужно изменить каталог (`cd`) на него, вот так:

```
cd python-environment-demo
```

Теперь я внутри этой папки и создаю свою виртуальную среду. Если вы используете macOS или Linux, велика вероятность, что у вас установлены как Python 2, так и 3, и мы хотим обязательно создать виртуальную среду на основе Python 3. Чтобы отличить друг от друга, вам нужно ввести следующее:

```
python3 -m venv venv
```

Если вы используете Windows, вероятно, у вас установлен только Python 3, поэтому команда такая:

```
python -m venv venv
```

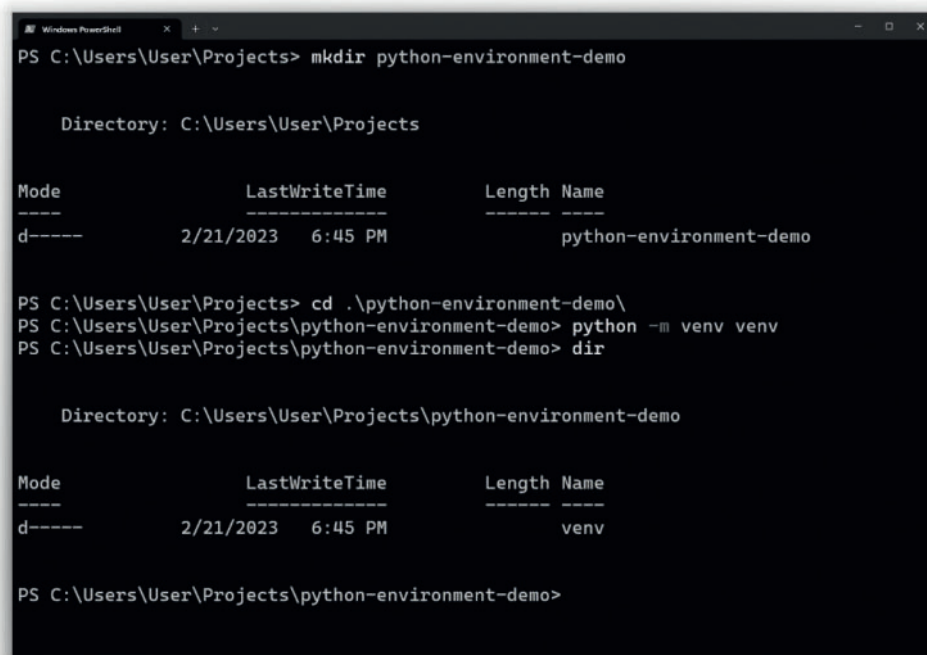
Здесь мы запускаем команду `python3` и передаем ключ `(-m)`, который запустит пакет `venv` для создания новой виртуальной среды на основе Python 3 в новой папке с именем `venv`. Как только команда завершится, я могу убедиться, что она работает в Windows, с помощью этого:

```
dir
```

А в macOS/Linux я могу использовать это:

```
ls -a
```

Вы должны увидеть выходные данные вашей системы. Я использую Windows, поэтому мой вариант выглядит так, как показано на рис. 3.1:



```
PS C:\Users\User\Projects> mkdir python-environment-demo

Directory: C:\Users\User\Projects

Mode                LastWriteTime         Length Name
----                -
d-----           2/21/2023   6:45 PM                python-environment-demo

PS C:\Users\User\Projects> cd .\python-environment-demo\
PS C:\Users\User\Projects\python-environment-demo> python -m venv venv
PS C:\Users\User\Projects\python-environment-demo> dir

Directory: C:\Users\User\Projects\python-environment-demo

Mode                LastWriteTime         Length Name
----                -
d-----           2/21/2023   6:45 PM                venv

PS C:\Users\User\Projects\python-environment-demo>
```

Рис. 3.1. Вывод терминала, проверяющий успешность создания виртуальной среды

Не обязательно, чтобы ваша виртуальная среда находилась в той же папке, что и остальная часть вашего проекта, но я обычно организую все таким образом, чтобы виртуальную среду было легко найти позже.

Если я собираюсь использовать Git или какую-либо другую систему контроля версий, было бы целесообразно, чтобы система контроля версий игнорировала эту папку. Вам *не следует* возвращать эту папку в свой репозиторий.

Последним шагом в работе с виртуальной средой является ее активация. Команда в Windows немного отличается от macOS и Linux. Команда для активации виртуальной среды в Windows выглядит так:

```
.\venv\Scripts\activate
```

В macOS и Linux это будет так:

```
source ./venv/bin/activate
```

Если вы успешно активировали свою виртуальную среду, приглашение должно измениться и отобразить имя виртуальной среды, которая активна в данный момент. Имейте в виду, что, если вы настроили свой терминал таким образом, чтобы предотвратить это, он может не работать. На рис. 3.2 можно видеть, что все у меня работает, на что указывает (venv), появляющийся в начале моего приглашения. В верхнем примере показана активация в Windows 11, а в нижнем – команда активации в Linux, такая же, как и в macOS:



Рис. 3.2. Моя виртуальная среда активирована

Когда я буду готов прекратить работу в своей виртуальной среде, я могу деактивировать ее, введя в терминале следующую команду:

```
deactivate
```

Это была небольшая работа. Если вы занимаетесь этим какое-то время, это не так уж и плохо – возможно, всего несколько минут. Однако, если вы

делаете это не очень часто, придется все просмотреть, а это может занять время. На самом деле вы делаете это только в начале нового проекта. Некоторые люди могут делать это всего несколько раз в год. Теперь давайте вернемся к PyCharm и посмотрим, как этот шаг интегрирован в рабочий процесс создания нового проекта.

СОЗДАНИЕ ПРОЕКТА В PYCHARM (ПОВТОРНО)

Если мы вернемся к PyCharm и создадим новый проект на чистом Python, вы увидите, где происходит процесс создания виртуальной среды. Давайте создадим новый проект в PyCharm, кликнув **File | New Project...**, как показано на рис. 3.3:

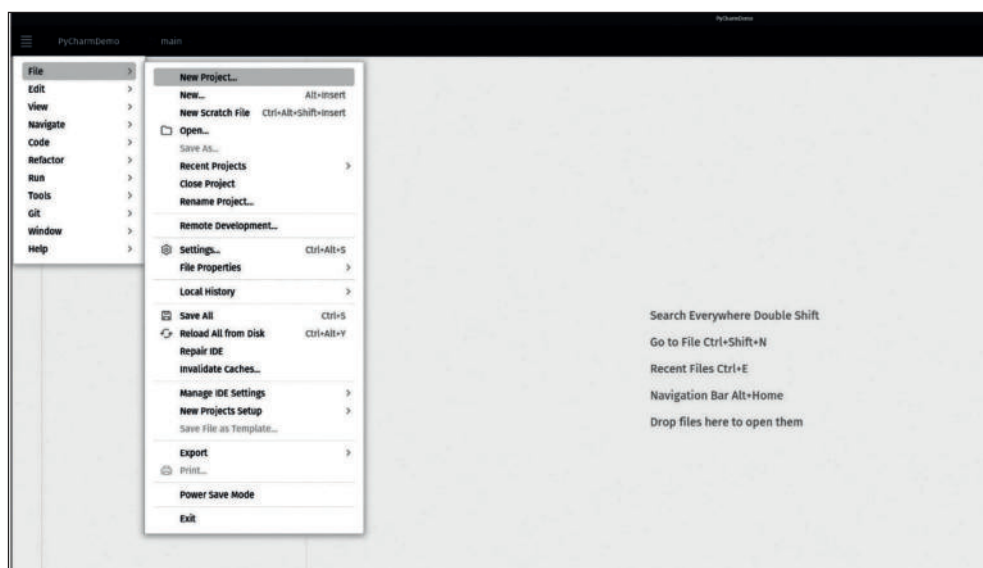


Рис. 3.3. Создание нового проекта в PyCharm

Это будет чистый проект Python. Профессиональная версия не является обязательной для дальнейшего использования. Диалоговое окно нового проекта мы уже видели в главе 2, но на этот раз сосредоточимся на некоторых деталях, которые пропустили. На рис. 3.4 слева показано окно нового проекта PyCharm в профессиональной версии. В сообщении PyCharm отсутствует меню типов проектов, поскольку оно может создавать только проекты «на чистом Python».

Раздел диалогового окна **New Project**, на которое я хотел бы обратить ваше внимание, выделен на рис. 3.5. Именно этот раздел экрана позволяет вам настроить виртуальную среду. В главе 2 мы прошли мимо этого и просто приняли значения по умолчанию. Как оказалось, значения по умолчанию почти соответствуют ручному процессу, который мы завершили в последнем разделе:

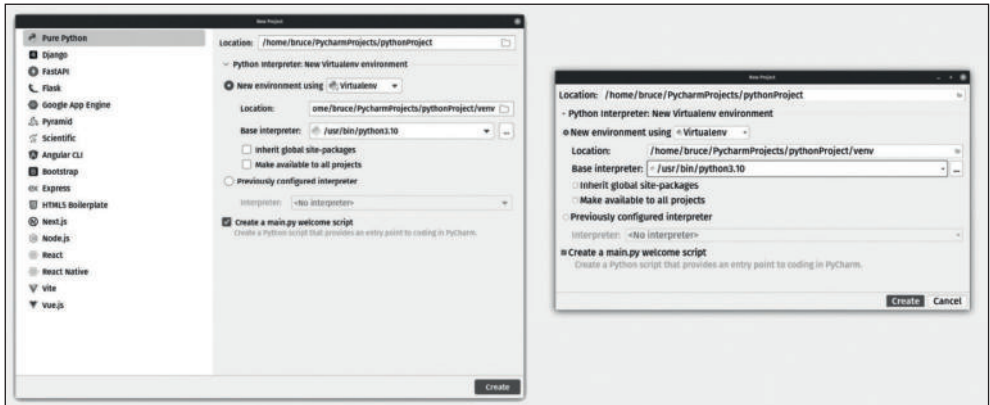


Рис. 3.4. Сравнение окна New Project для версии Professional (слева) и версии Community (справа)

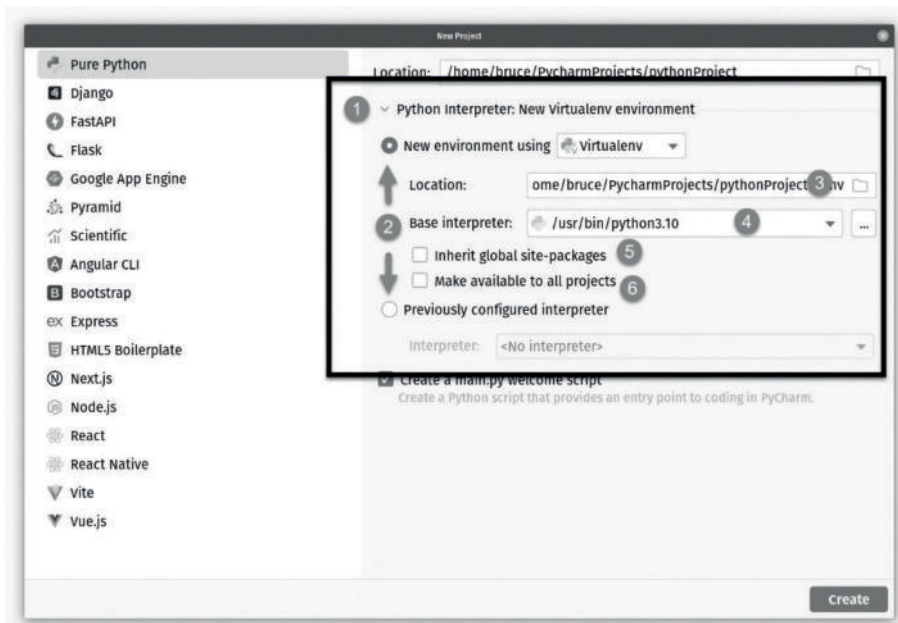


Рис. 3.5. Настройки виртуальной среды из PyCharm, перечисленные для пояснения

Давайте рассмотрим пронумерованные метки на рис. 3.5.

1. Это раздел настроек интерпретатора Python в окне **New Project**. Вы можете скрыть его, если хотите, повернув треугольник рядом с названием раздела, но, если бы мы это сделали, мы не смогли бы продолжать говорить о том, что будет дальше.
2. Здесь есть два варианта. Вы можете либо создать новую виртуальную среду, либо указать на уже созданную. Давайте сосредоточимся на создании новой. Рядом с опцией **New environment** есть переключатель и раскрывающийся список, позволяющий выбрать механизм создания

виртуальной среды. В нашем примере руководства в предыдущем разделе мы использовали функцию виртуальной среды Python `virtualenv`. Изменение механизма с помощью раскрывающегося списка приведет к изменению содержимого экрана в соответствии с настройками выбранного вами скрипта виртуальной среды. А пока давайте оставим его на **Virtualenv**, поскольку он самый старый и, вероятно, наиболее широко используемый.

3. Поле **Location** позволяет вам указать локацию вашей виртуальной среды. По умолчанию используется папка с именем `venv` внутри папки проекта. Вы можете установить локацию в любом месте вашего компьютера; он не обязательно должен находиться в папке проекта. Если у вас есть несколько проектов, которые совместно используют виртуальную среду, поскольку они имеют общие зависимости, имеет смысл создать центральное расположение для хранения среды. Большую часть времени я предпочитаю хранить виртуальную среду в той же папке, что и проект, чтобы не возникало вопросов относительно ее местоположения.
4. **Base interpreter** позволяет выбрать, какая установка Python будет использоваться для создания виртуальной среды. Если на вашем компьютере более одной инсталляции Python, PyCharm, вероятно, сможет найти ее автоматически. Выбор представлен в виде раскрывающегося списка мест, где PyCharm обнаружил установку Python. Если он каким-то образом пропустил один из них, можете нажать кнопку с многоточием (...) и перейти к инсталляции Python, которую вы хотите использовать. Если вы это сделаете, нужно будет перейти к исполняемому файлу Python и дважды кликнуть его.
5. Флажок **Inherit global site-packages** касается любых сторонних библиотек, которые вы могли установить глобально. Если установить этот флажок, они будут скопированы в вашу виртуальную среду и будут доступными локально в вашем проекте.
6. Флажок **Make available to all projects** позволяет легко находить и повторно использовать эту виртуальную среду в других проектах.

ИСПОЛЬЗОВАНИЕ СУЩЕСТВУЮЩЕЙ ВИРТУАЛЬНОЙ СРЕДЫ

Иногда вам нужно создать проект с точно такими же требованиями, которые используются в другом проекте. Вы можете легко поделить или повторно использовать виртуальную среду в PyCharm. На рис. 3.5 показано диалоговое окно **New Project** в PyCharm.

Чтобы использовать существующую виртуальную среду, вам необходимо изменить настройку по умолчанию **New Projects** использованием **Previously configured interpreter**, как показано на рис. 3.6.

Как только вы это сделаете, ваши возможности выбора существующей среды станут активными. Для выбора интерпретатора доступен раскрывающийся список. Он работает так же, как раскрывающийся список **Base interpreter**, который мы видели ранее при создании новой виртуальной среды. Если вы создали существующую среду в PyCharm, IDE запомнит ее. В этом случае я ранее создал виртуальную среду с использованием PyCharm, когда создавал код демонстрационного проекта для главы 1 «Введение в PyCharm – са-

мую популярную IDE для Python». PyCharm запомнит виртуальные среды, созданные с помощью PyCharm, и предложит их в списке.

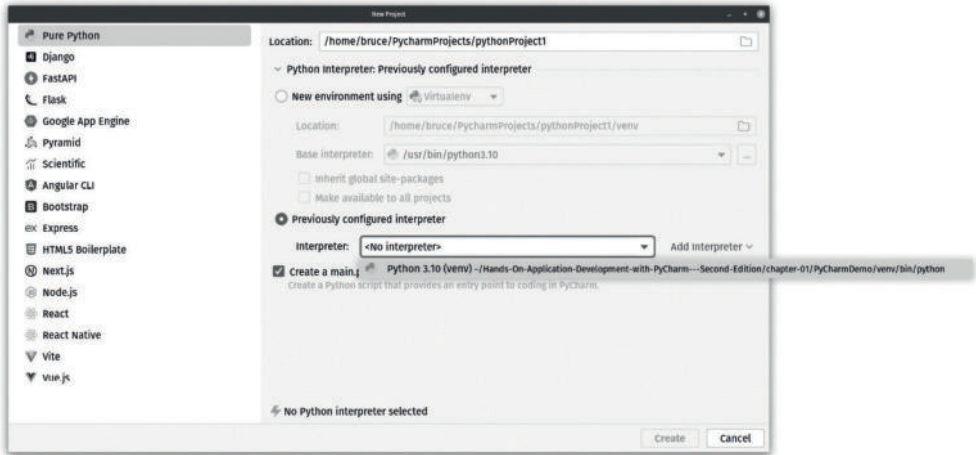


Рис. 3.6. Вы можете указать для своего проекта существующую виртуальную среду, изменив настройки проекта

Если вы использовали ручной метод или какой-либо другой инструмент, нужно будет добавить интерпретатор в список с помощью кнопки **Add Interpreter**. Нажав кнопку, обратите внимание на некоторые возможности, как показано на рис. 3.7:

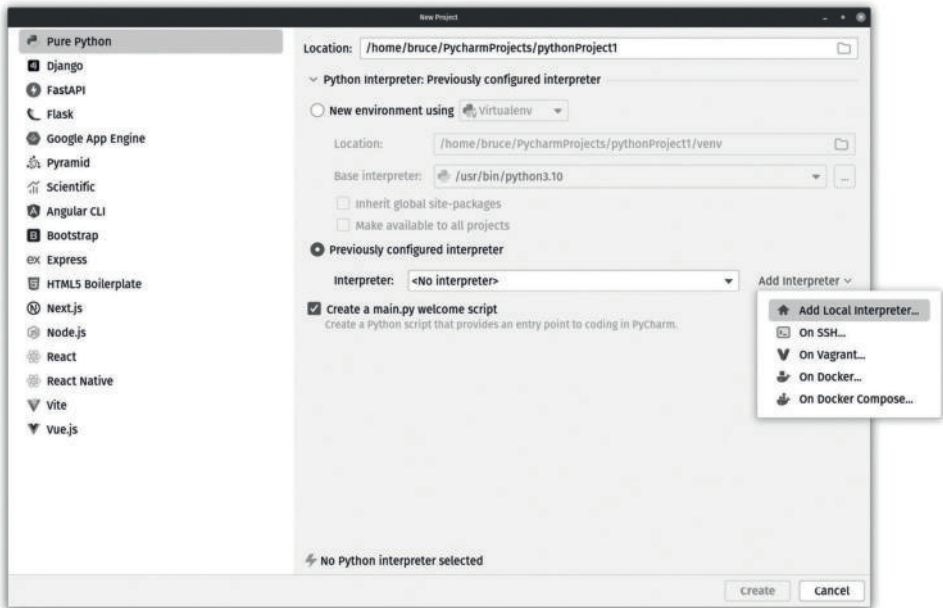


Рис. 3.7. Когда вы нажимаете Add Interpreter, у вас появляется множество вариантов источника происхождения среды, которую вы, возможно, захотите добавить

Пока что мы строго ограничиваем наше обсуждение добавлением виртуальной среды, существующей на вашем локальном компьютере. Также можно добавить среду на удаленном компьютере, **виртуальной машине (ВМ)**, **подсистеме Windows для Linux (WSL)**, которая является виртуальной машиной, или контейнере **Docker**. Мы поговорим об этих вариантах гораздо позже в книге.

После того как вы выберете **Add Local Interpreter...**, вам будет предложено перейти к исполняемому файлу Python в виртуальной среде, которую вы хотите использовать, как показано на рис. 3.8. Помните: структура виртуальных сред в Windows отличается от любой другой. В виртуальных средах macOS и Linux будет папка `bin`, содержащая исполняемый файл Python. В Windows в виртуальной среде есть папка `Scripts`, содержащая файл, указывающий на исполняемый файл Python. Ваша цель в любом случае – выбрать исполняемый файл:

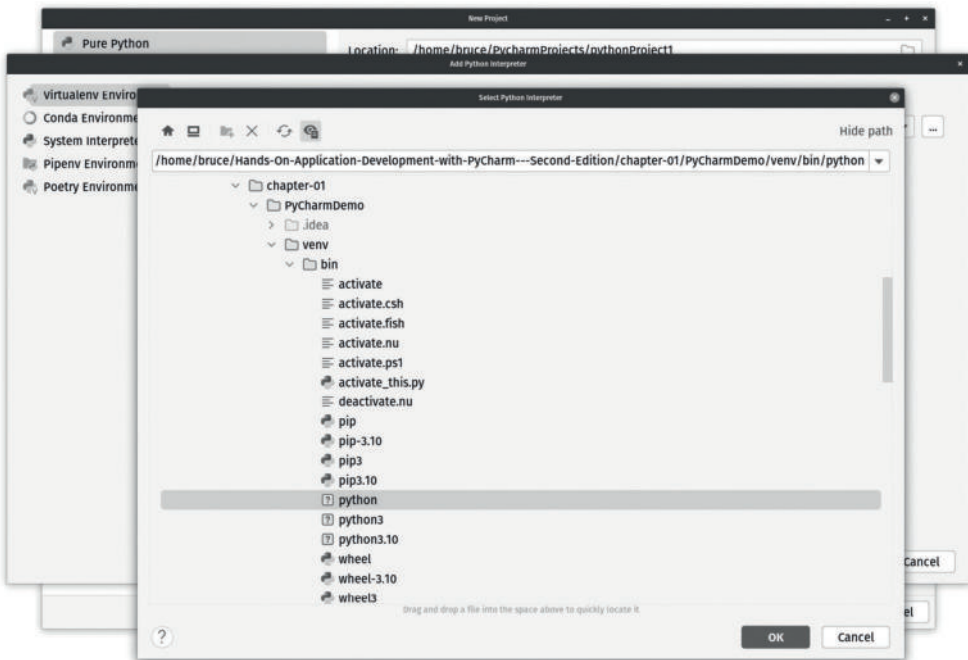


Рис. 3.8. Вам необходимо выбрать исполняемый файл Python в папке виртуальной среды, которую вы хотите использовать

СМЕНА ИНТЕРПРЕТАТОРА ДЛЯ ПРОЕКТА

Работая над проектом, который продолжается уже 6 месяцев или дольше, я использую один прием: создаю новую виртуальную среду с полностью обновленными пакетами. Таким образом, я могу протестировать программу с обновленными зависимостями, не повреждая мою готовую к работе виртуальную среду. Мы рассмотрим управление пакетами чуть позже в этой главе. А сейчас я хочу показать вам, где настройки интерпретатора вашего проекта существуют независимо от процесса его создания. Мне кажется это не очень интуитивно понятным. Это в **Settings**. Тот же параметр **Settings**, который вы используете для

глобальной настройки IDE во всех проектах, используется для установки параметров конкретного проекта, таких как параметры интерпретатора.

Независимо от вашей причины, можно изменить интерпретатор и, как следствие, виртуальную среду, используемую в вашем проекте. Вы найдете настройки проекта, кликнув значок шестеренки в верхнем левом углу пользовательского интерфейса, как показано на рис. 3.9:

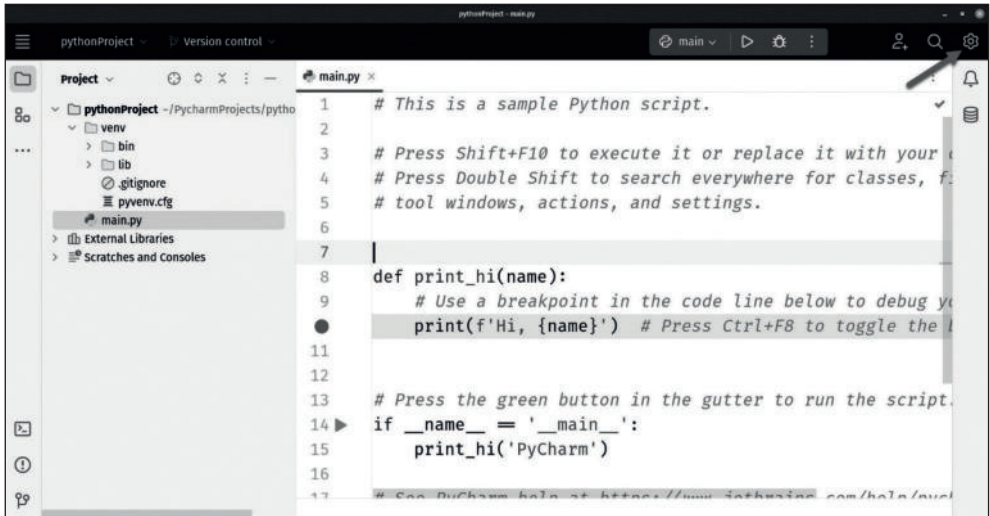


Рис. 3.9. Вы найдете глобальные настройки и настройки проекта, нажав на шестеренку

После того как вы выберете **Settings**, вы попадете на экран глобальных настроек, который мы изучали в главе 2. Здесь также хранятся свойства проекта, как показано на рис. 3.10:

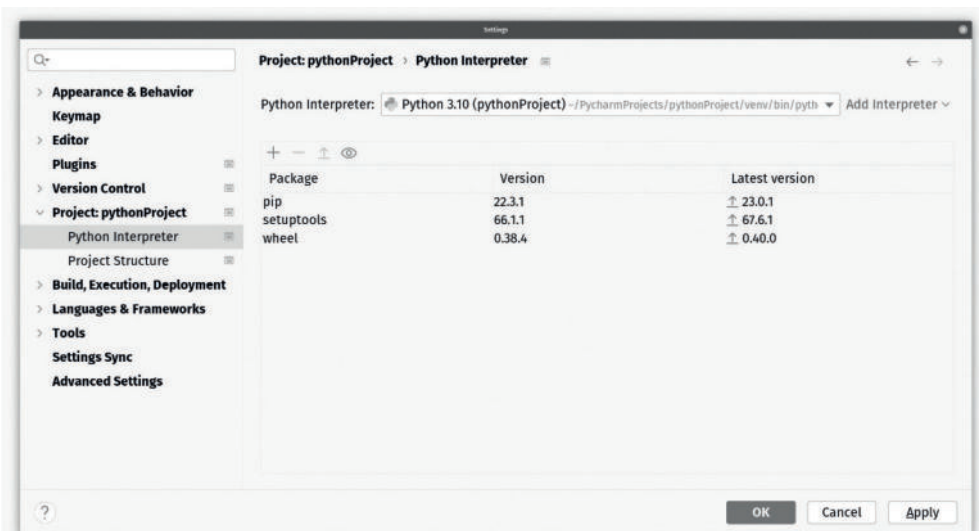


Рис. 3.10. Настройки проекта находятся в середине списка

Выберите настройки интерпретатора в левой части экрана и измените интерпретатор, используя тот же механизм, который мы использовали при первой настройке виртуальной среды. Вы можете создать новую среду или перейти к новой. Не забывайте, что при переходе в новую среду вам необходимо выбрать исполняемый файл Python, а не папку, в которой находится установка Python. Иногда я забываю об этом и недоумеваю, почему диалоговое окно выбора не исчезает. Оно ищет файл, который является исполняемым файлом Python, а не папкой.

Активация виртуального окружения

В нашем ручном упражнении ранее в этой главе мы создали виртуальную среду с нуля, используя `virtualenv`. После создания среды вы должны активировать ее, чтобы использовать. Очевидный вопрос может быть: «Как мне активировать среду?» Вот прекрасный ответ: это делать не обязательно. Пока вы находитесь в IDE, параметры среды всегда присутствуют и всегда соблюдаются. Когда вы добавляете новые сторонние пакеты из **индекса пакетов Python (PyPI)**, о котором мы вскоре расскажем, они автоматически попадают в нужное место в соответствии с настройками вашей среды.

Когда мы запускаем нашу программу, мы можем переопределить **конфигурацию запуска**, но по умолчанию она использует настройки среды проекта. Честно говоря, переопределение в конфигурации запуска редко бывает полезным. С учетом вышесказанного позже в этой главе я покажу вам, как переопределить конфигурацию запуска.

Использование встроенного терминала

Еще одно место, где следует беспокоиться об активности виртуальной среды, – это **терминал**. Это правда, что PyCharm не может помочь вам с вашим системным терминалом. Однако у PyCharm есть собственная вкладка окна терминала. Если вы используете терминал PyCharm, настройки интерпретатора проекта применяются автоматически. Вам не придется ничего активировать вручную. Раз уж мы об этом заговорили, давайте посмотрим на встроенный терминал в PyCharm. Пункт меню для показа терминала вы можете найти в меню **View**, как показано на рис. 3.11. Лично я провожу много времени в терминале, поэтому закрепил сочетание клавиш в мышечной памяти.

Если наличие `(venv)` в командной строке не является достаточным доказательством того, что это сработало, я предоставил дополнительные сведения на рис. 3.12. Если вы используете macOS или Linux, можете использовать команду `that`, чтобы найти путь к любому исполняемому файлу. Извините, пользователи Windows: в вашей ОС нет эквивалентной команды. Результатом команды `that` является то, что она показывает, что, когда я запускаю `python3`, он использует тот, который находится по указанному пути, а не общесистемную установку, которая обычно находится в `/usr/bin/python3`.

Вызов этой опции меню вызывает терминал, и, как было обещано и вы можете видеть на рис. 3.12, он учитывает настройки виртуальной среды нашего проекта.

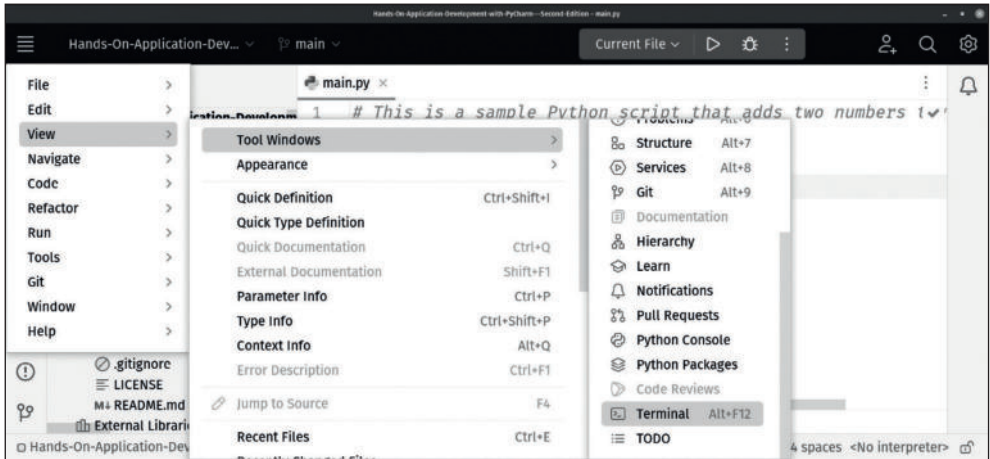


Рис. 3.11. Встроенный терминал PyCharm можно найти в меню View

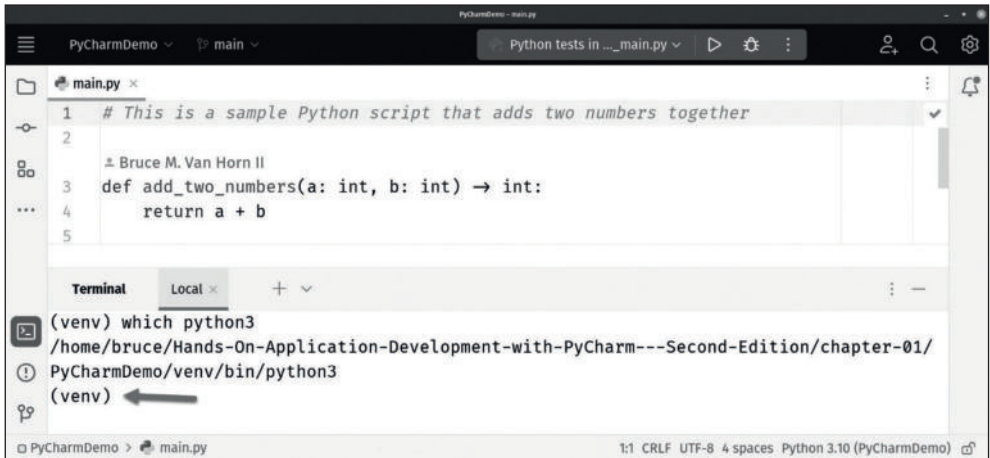


Рис. 3.12. Вы можете видеть, что терминал запускается с уже активной виртуальной средой

Работа с REPL в окне консоли

Последний инструмент в PyCharm, который может заставить вас задуматься, соблюдаются ли настройки вашей среды, – это **консоль (console)**. Консоль отличается от терминала. Terminal – это всего лишь оболочка вашей операционной системы. Консоль представляет собой работающую версию Python, основанную на настройках вашего интерпретатора. Вам представлена среда Python **Read Execute Print Loop (REPL)**, которая удобна для тестирования небольших фрагментов синтаксиса или для тестирования импорта. Если у вас не было этого инструмента, вы могли бы получить ту же функциональность в терминале, просто запустив `python3`. PyCharm предоставляет вам интегрированный инструмент, благодаря которому вам не придется этого делать. Чтобы попасть в консоль, используйте то же меню, которое вы использовали для входа в терминал. Я показал это вам на рис. 3.13:

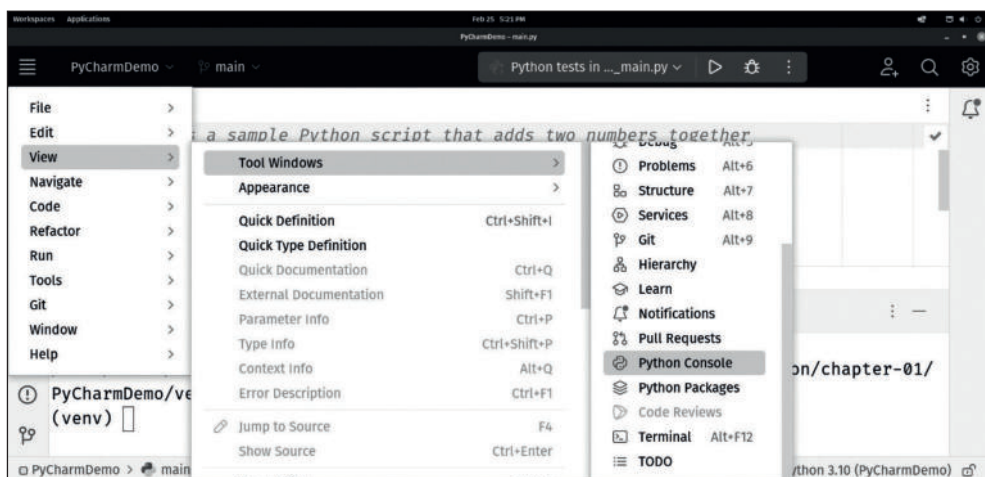


Рис. 3.13. Консоль Python можно вызвать через меню View

Ваша консоль не только знает настройки вашего интерпретатора, но также знает содержимое вашего проекта. Вы можете импортировать и тестировать что угодно в своем проекте прямо в консоли. Я нахожу это полезным, когда отслеживаю страшное сообщение «Модуль не найден? Что значит, что его не нашли?! Это прямо здесь! Я смотрю прямо на это!», – проблема, с которыми мы иногда сталкиваемся. Это также полезно для экспериментов с некоторыми более краткими синтаксисами, которые мы встречаем в Python, такими как **регулярные выражения** (regexes) или понимание списков. На рис. 3.14 показан небольшой пример:

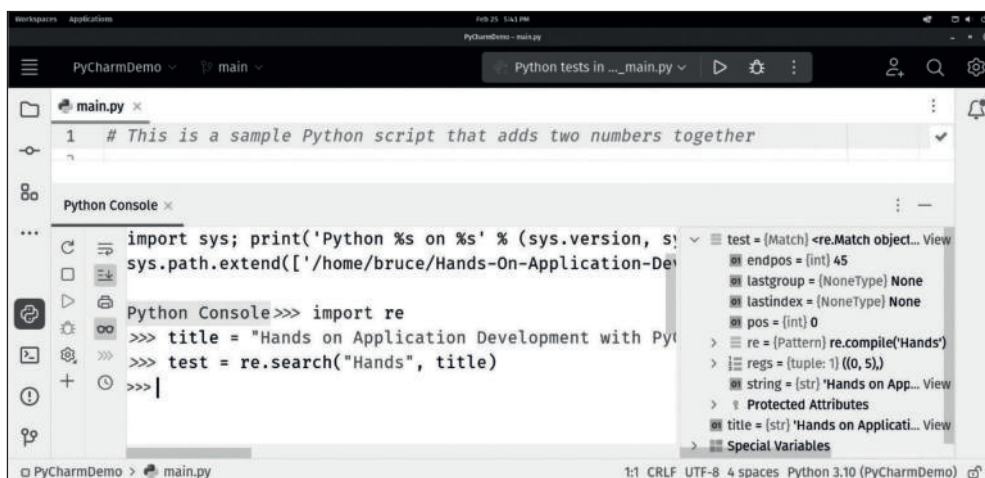


Рис. 3.14. Окно консоли, используемое для проверки простого регулярного выражения

Код, который я опробую на рис. 3.14, никого не впечатлит на моем большом собеседовании по Python в Google:

```
import re
title = "Hands on Application Development with PyCharm"
test = re.search("Hands", title)
```

Я печатаю каждую строку в консоль. Помните: это представление живого кода, а не файл, который вы выполняете. Каждый раз, когда я нажимаю *Enter*, вычисляется только что введенное выражение, и результаты появляются в отладочном окне с правой стороны, когда это необходимо. Думайте об этом как о сеансе отладки, когда справа у вас есть окно просмотра, и все, что вы вводите рядом с промптом `>>>`, мгновенно становится точкой останова.

Нажав *Enter* для оценки третьей строки, я вижу, что переменная `test` имеет в качестве значения объект `re.Match`. Посмотрите на окно еще раз, и вы увидите, что символы были найдены в позиции (0, 5) или символы от 0 до 5 представлены в одном кортеже (`regs`).

Испытание сложных фрагментов кода в консоли экономит немного времени по сравнению с обычным процессом, состоящим из запуска программы, чтения выводов `print`, внесения изменений и повторной попытки, что занимает большую часть нашего времени как разработчиков программы. Затем вы забываете удалить операторы печати, а вам следует это сделать, потому что они могут серьезно замедлить работу большого скрипта.

Консоль выполняет все мгновенно и возвращает вам результаты. Она гораздо более подробна, чем вывод `print`, поскольку вы можете увидеть внутренности любого объекта, используя окно, похожее на отладку, а не просто видеть строку. После некоторых манипуляций с окном консоли можете вернуться к своему коду и просто использовать то, что работало.

РАБОТА СО СТОРОННИМИ БИБЛИОТЕКАМИ ПАКЕТОВ

Python известен своей философией «батарейки прилагаются», которая противоречит многим другим языкам. Создатель Python Гвидо ван Россум считает, что надежная и полная стандартная библиотека важна и что язык должен быть способен выполнять практически любую задачу без какой-либо зависимости от сторонних источников.

Под сторонними зависимостями я подразумеваю внешние по отношению к Python библиотеки, предназначенные для выполнения специализированных функций, которые не реализованы в Python «из коробки». Другими словами, Python ставит перед собой очень высокую цель. Он должен иметь возможность делать буквально все самостоятельно, используя так называемую стандартную библиотеку Python.

Естественно, эта цель никогда не может быть достигнута в совершенстве. Стандартная библиотека Python очень полная. Однако рано или поздно вы обнаружите, что стандартная библиотека либо не может что-то сделать, либо реализация задачи с ее помощью не так проста, как могла бы быть. Давайте рассмотрим пару примеров широко используемых сторонних библиотек и поговорим о том, почему вы можете захотеть их использовать.

Начнем с библиотеки `requests`. Это сторонняя библиотека, которую вы найдете в PyPI по адресу <https://pypi.org/project/requests/>. В каждом современном языке

есть что-то подобное. В JavaScript имеется **менеджер пакетов Node (NPM)** по адресу <https://npmjs.com>. PHP использует **Composer** для установки пакетов с <https://packagist.org/>. Языки .NET используют **NuGet** с пакетами, зарегистрированными на <https://nuget.org>. У Python есть <https://pypi.org>. Это централизованный список сторонних модулей и библиотек, которые вы можете добавить в свой проект, часто бесплатно. Образно говоря, идея состоит в том, чтобы избавить нас от необходимости изобретать велосипед каждый раз, когда мы хотим построить новый автомобиль. Поскольку я занимаюсь проектом **программного обеспечения как услуги (SaaS)** в качестве своей повседневной работы, мне нужен простой способ выполнения запросов **протокола передачи гипертекста (HTTP)**. Возможность моих скриптов Python вызывать какой-либо веб-сервис и обрабатывать результаты имеет решающее значение, и это очень распространенная задача.

В стандартной библиотеке Python есть несколько вариантов достижения этой цели. Я думаю, что наиболее очевидным является `urllib` из стандартной библиотеки. Вот пример кода из документации `urllib2` по адресу <https://docs.python.org/3/howto/urllib2.html>:

```
import urllib.parse
import urllib.request

url = 'http://www.someserver.com/cgi-bin/register.cgi'
values = {'name': 'Michael Ford',
          'location': 'Northampton',
          'language': 'Python' }
Working with third-party package libraries 87
data = urllib.parse.urlencode(values)
data = data.encode('ascii') # data should be bytes
req = urllib.request.Request(url, data)
with urllib.request.urlopen(req) as response:
    the_page = response.read()
```

Сначала мы импортируем две части пакета `urllib`, помня, что на самом деле это `urllib2`. Далее мы создаем переменную URL. Если вам интересно, какие ссылки на `cgi` содержатся в примере, это **общий интерфейс шлюза (CGI)**. Никто не использовал его в производстве примерно с 1991 года, но он все еще присутствует в документации и действует до сих пор. CGI – прародитель современной веб-разработки. По сути, это относится к программе, написанной на C, которая выполняется веб-сервером в ответ на конечную точку¹, имеющую в адресе `cgi-bin`. Этот стиль разработки был давно вытеснен такими, как **PHP**, классические **Active Server Pages (ASP)**, **Java ServerPages (JSP)** и **ColdFusion**. Даже они с тех пор превратились в современные системы, которые используют более сложные, но простые в кодировании реализации очень гибких библиотек маршрутизации. Мы рассмотрим их позже в книге, когда будем говорить о функциях разработки веб-приложений в PyCharm Professional.

Как только мы установим переменную URL, нам понадобятся некоторые данные. В этом примере мы используем HTTP POST для отправки данных на наш

¹ Конечная точка – это имя, используемое для обратного поиска правил URL-адресов с помощью `url_for`, и по умолчанию оно соответствует имени функции просмотра. – Прим. ред.

веб-сервер для обработки. Это делается с помощью переменной `values`, содержащей словарь, имитирующий поля и значения, которые вы можете найти в форме **языка гипертекстовой разметки (HTML)**.

После того как мы это настроим, нужно сделать парсинг этого словаря в формат, подходящий для передачи. Это делается с помощью `urllib.parse.urlencode`. Далее нам нужно дополнительно закодировать данные в виде **американского стандартного кода обмена информацией (ASCII)**. Это очередной анахронизм в документации. Современные системы используют 8-битный универсальный текстовый формат (UTF-8), поскольку ASCII кодирует только буквы романских языков, таких как английский, французский, немецкий, итальянский или португальский. Остальному миру не повезло. UTF-8 обрабатывает все общепринятые сегодня глобальные алфавиты; таким образом, игнорируются только древние египтяне, поскольку UTF-8 не поддерживает египетские иероглифы – по крайней мере, я об этом ничего не знаю.

После того как мы закодировали текст, нужно создать объект `request`. Мы передаем переменную URL и данные конструктору для `urllib.request.Request`. Только тогда мы готовы отправить запрос в последних двух строках примера.

Мы установили, что существует способ сделать запрос `POST` в стандартной библиотеке Python. Однако до сих пор мы рассмотрели только самое основное требование. Что делать, если мне нужно представить проверенный запрос? Как мне поступить с сессиями или файлами `cookie`? Можем ли мы сделать это еще проще? Мы можем это сделать с помощью библиотеки `request`. Вот пример кода:

```
import requests

requests.post('https://httpbin.org/post', data={'key': 'value'})
```

Наш предыдущий код из `urllib` можно было бы выразить в виде одной строки, используя `requests`! Более сложные требования, такие как сесансы, файлы `cookie` и т. д., столь же просты. Документация современная и содержит полезные примеры. Популярные библиотеки PyPI, такие как `requests`, очень хорошо документированы, а учебные пособия можно найти на таких сайтах, как <https://realpython.com>. Я позаимствовал предыдущий пример с <https://realpython.com/python-requests/>, который является частью большого и очень полного руководства по использованию этой мощной библиотеки.

Цель всего этого – указать на то, что система сторонних библиотек кода необходима или, по крайней мере, полезна в контексте Python – языка, который стремится включить в себя все, что вам когда-либо может понадобиться, но никогда этого не достигнет. Стандартная библиотека может быть близка к этому, но она никогда не будет включать в себя все, что бы я хотел.

Еще один важный пример необходимости сторонних библиотек касается научной и финансовой работы. Стандартные математические возможности Python примерно такие же, как и в любом другом языке. Большинство языков не очень точно справляются с вычислениями с плавающей запятой. Они также не очень разбираются в обработке матриц числовых данных, что представляет собой общее требование. К счастью, существуют сторонние библиотеки, такие как `numpy` и `pandas`, которые могут заполнить этот пробел. Эти библиотеки открывают новые возможности для разработчиков Python и являются одной из основных причин, по которым Python продолжает привлекать новых пользователей.

Добавление сторонних библиотек в PyCharm

PyCharm включает экран пользовательского интерфейса, который позволяет вам управлять пакетами в вашем текущем проекте. Однако, если IDE не использовалась, вы могли бы использовать менеджер пакетов, чтобы сделать это вручную. Например, чтобы установить `numpy` в ваш проект, можете использовать диспетчер пакетов `pip` с помощью такой команды:

```
pip install numpy
```

Вам следует убедиться, что вы активировали виртуальную среду для проекта, чтобы все работало правильно. В противном случае `numpy` будет установлен глобально. Установщик `pip` не единственный доступный установщик. Есть `easy_install`, `pipenv`, `poetry`, и, конечно же, `conda`. Я не собираюсь обсуждать относительные преимущества каждого из них, поскольку это вопрос предпочтений. PyCharm поддерживает их все.

Менеджер пакетов PyCharm вызывает менеджер пакетов, установленный в настройках вашей среды. Он представляет собой графический интерфейс, в котором вы можете искать пакеты, просматривать их описания, а также устанавливать, обновлять и удалять любой пакет в вашем проекте. Давайте взглянем.

Возвращаясь к PyCharm, давайте вернемся к настройкам нашего проекта. Нажмите **Python Interpreter**, как показано на рис. 3.15.

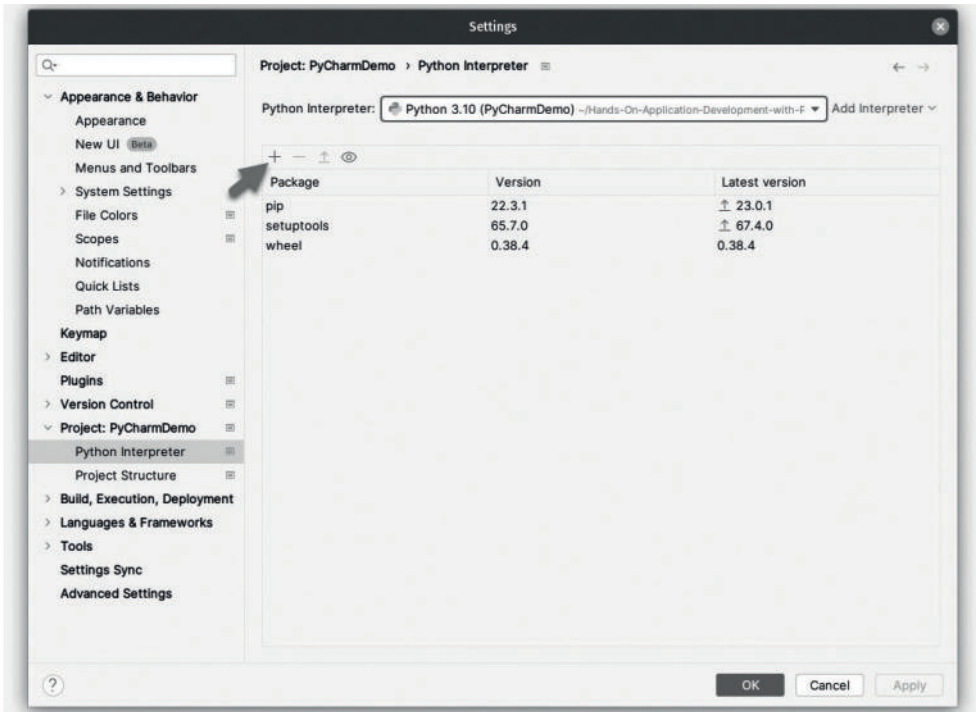


Рис. 3.15. В настройках интерпретатора отображается не только ваш интерпретатор, но и список пакетов, ему доступных

На экране отображается интерпретатор, используемый в текущем проекте. Ранее мы видели, что можем изменить эту среду в любой момент. Теперь сосредоточимся на пустом списке пакетов, который занимает большую часть экрана. Здесь вы сможете работать с пакетами после их установки.

Давайте установим пакет `numpy`. Начните с нажатия кнопки **+**, отмеченной стрелкой, вверху списка пакетов, как показано на рис. 3.15. Откроется еще одно диалоговое окно, открывающее поиск пакетов. В текстовом поле поиска, показанном на рис. 3.16, введите `numpy`. По мере ввода вы увидите список пакетов PyPI, соответствующих вашему запросу. Нажмите ввод `numpy` и просмотрите описание.

Рекомендуется внимательно изучить описание, поскольку, если вы неправильно напишете поисковый запрос, можете получить неправильные пакеты. Это определено библиотека `numpy`, поэтому я устанавливаю ее, нажав кнопку **Install Package** в нижней части диалогового окна. После небольшого ожидания я увижу сообщение, показанное на рис. 3.16 (чуть выше неактивной кнопки **Install Package**, о том, что пакет успешно установлен:

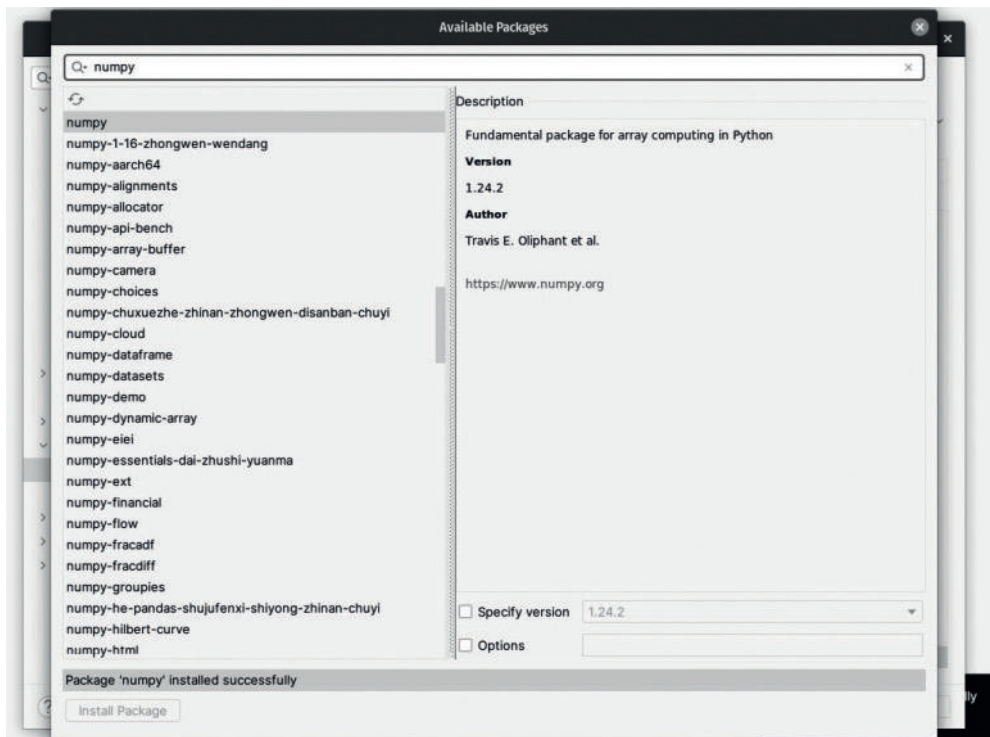
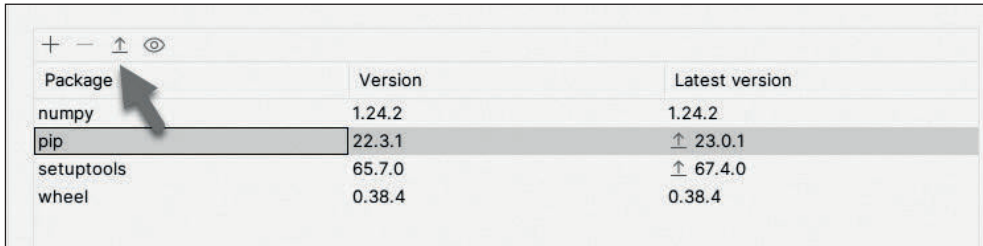


Рис. 3.16. Дисплей внизу показывает, что пакет был успешно установлен

Продолжайте и закройте диалоговое окно, чтобы вернуться к настройкам проекта.

Удаление сторонних библиотек в PyCharm

Рядом с кнопкой **+**, которую мы только что использовали для поиска и установки пакета `numpy`, мы видим еще несколько кнопок возле стрелки на рис. 3.17:



Package	Version	Latest version
numpy	1.24.2	1.24.2
pip	22.3.1	↑ 23.0.1
setuptools	65.7.0	↑ 67.4.0
wheel	0.38.4	0.38.4

Рис. 3.17. Инструменты управления установками пакетов

Вы, вероятно, можете догадаться, что они делают. Кнопка — удалит выбранный пакет. Кнопка со стрелкой вверх позволит вам обновить пакет до более новой версии. На рис. 3.17 мы видим, что `pip` можно обновить до версии 23.0.1. Для этого кликните стрелку вверх. Кнопка с глазом покажет вам предварительные версии пакетов, если вы захотите опробовать новейшие версии пакетов.

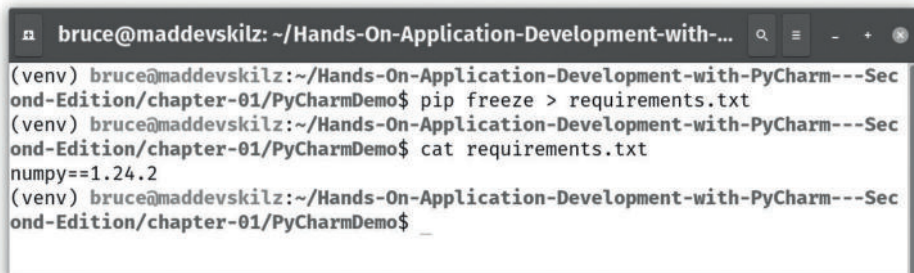
Теперь, когда мы понимаем, что делает пользовательский интерфейс для управления пакетами, я хотел бы на мгновение отвлечься и поговорить о том, с чем он работает не очень хорошо. На мой взгляд, в нынешней версии PyCharm нет хорошего способа работы с файлом `requirements.txt`.

Использование файла `requirements.txt`

Лучшая практика использования проектов Python — использовать файл с именем `requirements.txt` для перечисления зависимостей сторонних библиотек для вашего проекта. Если вы начинаете свой проект с нуля, можете сгенерировать файл `requirements.txt`, используя `pip` в своем терминале, с помощью этой команды:

```
pip freeze > requirements.txt
```

На рис. 3.18 показан пример содержимого только что созданного нами файла `requirements.txt`. Возможно, вам придется присмотреться, чтобы увидеть это, поскольку оно состоит всего из одной строки. Вы можете увидеть имя пакета, а также требования к версии для `numpy==1.24.2`. Если вы используете другой менеджер пакетов, просмотрите конкретный процесс создания файла `requirements.txt`:



```

bruce@maddevskilz: ~/Hands-On-Application-Development-with-...
(venv) bruce@maddevskilz:~/Hands-On-Application-Development-with-PyCharm---Second-Edition/chapter-01/PyCharmDemo$ pip freeze > requirements.txt
(venv) bruce@maddevskilz:~/Hands-On-Application-Development-with-PyCharm---Second-Edition/chapter-01/PyCharmDemo$ cat requirements.txt
numpy==1.24.2
(venv) bruce@maddevskilz:~/Hands-On-Application-Development-with-PyCharm---Second-Edition/chapter-01/PyCharmDemo$ _

```

Рис. 3.18. Содержимое образца файла `requirements.txt`, показывающее сторонние зависимости для нашего проекта

Если вы клонировали существующий проект и у него есть файл `requirements.txt`, PyCharm распознает его, когда вы откроете проект, и предложит установить содержимое в вашу виртуальную среду. PyCharm также будет контролировать ваш проект во время работы. Если вы добавите новые зависимости либо через графический интерфейс, который мы обсуждали, либо вручную из терминала, PyCharm предложит автоматически добавить эти зависимости в ваш файл `requirements.txt`.

Новое окно Python Packages

Новым в последних выпусках PyCharm, использующих новый пользовательский интерфейс, является окно **Python Packages**. Оно добавляет аналогичную функциональность в окно настроек интерпретатора в другом месте и макете. Очевидно, они читали через мое плечо, когда я писал ранее о том, что настройки среды Python, находящиеся внутри окна общих настроек, не очень интуитивно понятны. Хотя я сомневаюсь, что мне дадут премию за эту наводку. Вы найдете новое окно **Python Packages** в многоточии **More Tool Windows**, как показано на рис. 3.19:

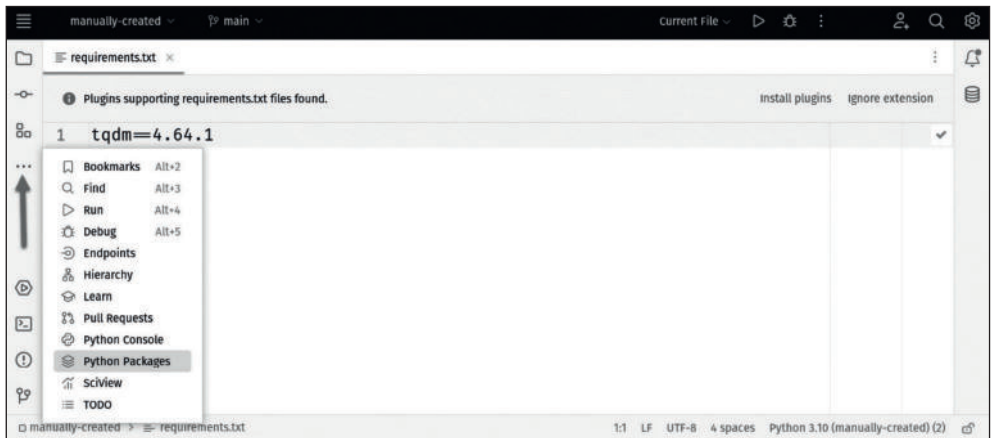


Рис. 3.19. Меню More Tools содержит новое окно Python Packages

Когда вы кликнете значок **Python Packages**, появится новое окно, как показано на рис. 3.20.

Как и в окне **Environment Settings**, в окне **Python Packages** отображается список пакетов, установленных в виртуальной среде, а также соответствующий номер версии. Есть кнопка **Add Package**, которая позволяет добавить пакет из PyPI или из репозитория. Вы можете искать пакеты, используя диалоговое окно поиска, в котором отображается список соответствующих пакетов. Нажав на один из них, вы увидите документацию для этого пакета. Это шаг вперед, по сравнению с окном **Environment Settings**, поскольку в нем отображается документация по оценке проекта. Вы можете увидеть пример на рис. 3.21:

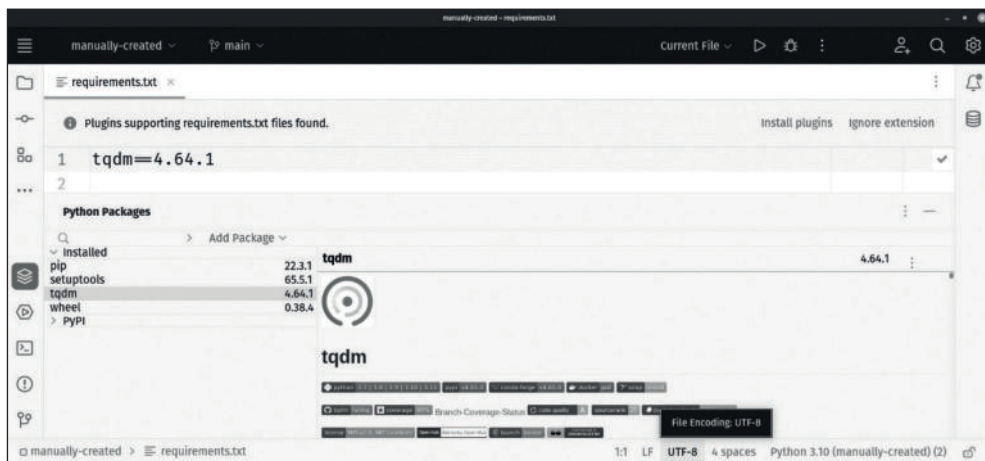


Рис. 3.20. Новое окно Python Packages в PyCharm

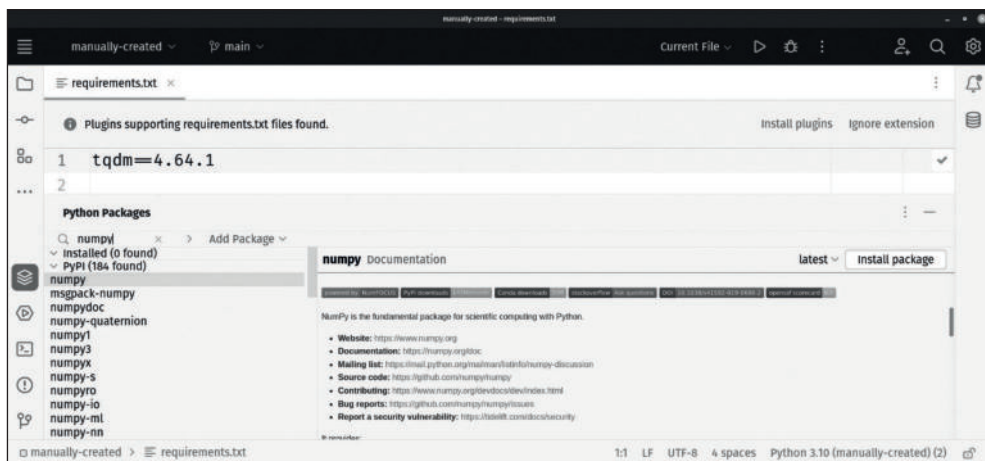


Рис. 3.21. Поиск пакета numpy в новом окне пакета Python

Как видите, я искал пакет, который был найден в PyPI и нашел 184 совпадения. Я выбрал пакет `numpy`, и PyCharm отображает документацию по проекту. Есть красивая большая кнопка **Install package** и селектор для выбора версии, которую я хочу установить.

Я могу удалить установленный пакет, кликнув установленный пакет в списке. Номер версии отображается вместе с документацией пакетов. Под многоточием рядом с номером установленной версии находится кнопка **Delete Package**, как показано на рис. 3.22.

На момент написания этой статьи эта функция настолько нова, что почти не вошла в эту главу. Случайно заметил это на днях на работе. На мой взгляд, она представляет собой менее запутанный и более удобный способ работы с пакетами Python.



Рис. 3.22. Удаление пакета можно выполнить с помощью пункта меню под многоточием рядом с номером версии в окне Python package

ФУНКЦИИ ВЕРСИИ PROFESSIONAL, ВАЖНЫЕ ДЛЯ ВИРТУАЛЬНЫХ СРЕД

Одной из основных особенностей профессиональной версии PyCharm является простота создания проектов по сравнению с менее автоматизируемым подходом. Рассмотрим новое окно проекта версии Professional, в котором есть набор типов проектов, как показано на рис. 3.23:

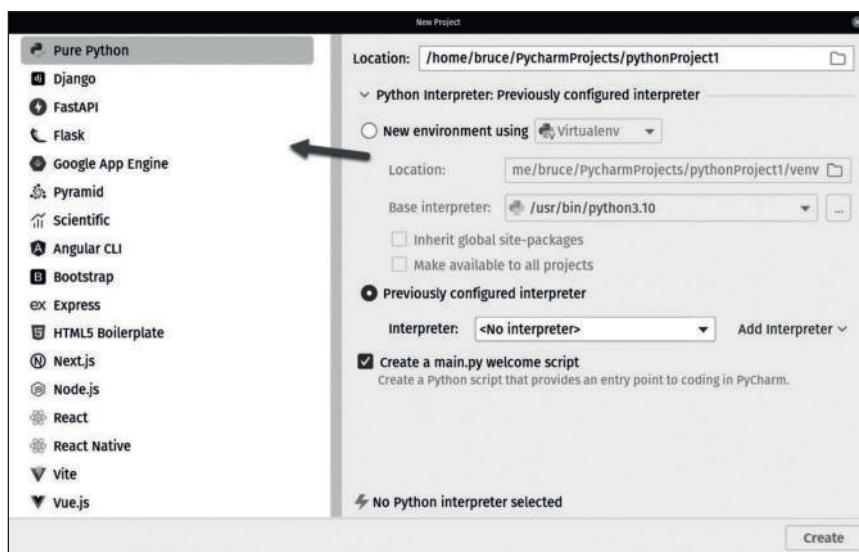


Рис. 3.23. В версии Professional в диалоговом окне New Project перечислено множество шаблонов проектов

В версии Python Community нет ни одного из этих вариантов проекта. Вы можете создавать только проекты на чистом Python, но, честно говоря, вы все равно можете создать проект Flask или любой другой тип проектов, перечисленных в версии Community. Однако IDE не поможет вам в этом.

Давайте рассмотрим шаги по созданию проекта Flask. Flask – это библиотека, которая позволяет нам легко создавать динамические веб-приложения.

Мы подробно рассмотрим это в главе 8 «Создание динамических сетевых приложений с Flask». А сейчас мы собираемся сравнить уровень работы, необходимый для этого, без версии Professional (т. е. вообще без IDE).

Вот шаги для ручной настройки проекта.

1. Создайте новую папку для вашего проекта.
2. Создайте новую виртуальную среду для своего проекта.
3. Активируйте виртуальную среду.
4. Используйте свой любимый менеджер пакетов, чтобы установить библиотеку Flask вместе с ее зависимостями.
5. Создайте файл Python для хранения стартового кода.
6. Исследуйте интернет, найдите простой пример «Hello, world» и поместите его в свой файл Python.
7. Создайте скрипт, который может устанавливать любые необходимые переменные среды, такие как `PYTHONPATH`, а затем запустите программу.

Могу поспорить, что многие из вас смогут сделать это максимум за 20–30 мин. Вот моя проблема: когда вы больше всего воодушевлены своей новой проектной идеей? Это в начале проекта! Вы готовы к работе, но затем должны остановиться и потратить 30 мин на эту утомительную настройку. Это не так уж и много времени, но это перерыв. Есть вероятность столкнуться с неожиданными препятствиями, которые могут подорвать ваш энтузиазм. В крайних случаях иногда неожиданные препятствия задерживают вас на часы или дни. PyCharm автоматизирует весь процесс. Вы проходите через пару диалоговых окон, обычно принимая значения по умолчанию, и ваш проект готов менее чем за 30 с. Вы можете сразу же приступить к воплощению своей идеи в коде, изменив начальный шаблон, сгенерированный IDE.

Версия Professional поможет вам создавать более сложные проекты с минимальными усилиями с вашей стороны. Среда IDE создаст структуру папок, базовый набор файлов с подсказками для редактирования и виртуальную среду. PyCharm даже установит для вас требования.

ИМПОРТ ПРОЕКТОВ В PYCHARM

Возможно, это само собой разумеется, но я все равно это скажу. Вы можете импортировать в PyCharm существующие проекты, созданные вручную или с помощью другой IDE. Это кажется очевидным, потому что, в конце концов, это всего лишь вопрос открытия папки на вашем компьютере. Процесс импорта на самом деле так же прост, как открытие папки в PyCharm. Однако в этой истории есть нечто большее. PyCharm – ваш разумный союзник, когда дело доходит до запуска на вашем компьютере проекта, который был запущен или полностью создан на другом компьютере без участия PyCharm. Давайте посмотрим на это в действии.

Исходный код этой книги мы клонировали в конце главы 2, а в папке `Chapter-03` есть проект, созданный полностью из командной строки. Вы можете увидеть процесс, который я использовал, на рис. 3.24:

в текстовый файл, который я могу включить в свой репозиторий Git, я передал вывод с помощью `> requirements.txt`, в результате чего полная команда выглядела следующим образом:

```
pip freeze > requirements.txt.
```

Такая работа по настройке из командной строки нам не по плечу. На самом деле я рекомендую вам отпраздновать это, купив себе новую пару брюк карго и приняв новый образ беззаботности! Если вы идентифицируете себя как разумное существо, не носящее брюки карго, я надеюсь, что вы воспользуетесь своим воображением, чтобы воплотить мои намерения. Если серьезно, то, рискуя обобщить, вам, вероятно, не следует принимать от меня советов по поводу гардероба и социального поведения. Традиционно *советы по гардеробу* – не моя *сильная сторона*.

Предупреждение

Если вы читаете первую из моих книг, я придерживаюсь одного основного правила: все каламбуры приводятся умышленно.

К счастью, работа с проектами, созданными за пределами PyCharm, является сильной стороной этой среды. Поскольку проект уже создан вне PyCharm, давайте посмотрим, что из него сделает PyCharm. Давайте откроем папку `Chapter-03/ manually-created` в PyCharm. Чтобы все было предельно ясно, я показываю вам диалоговое окно открытия проекта на рис. 3.25:

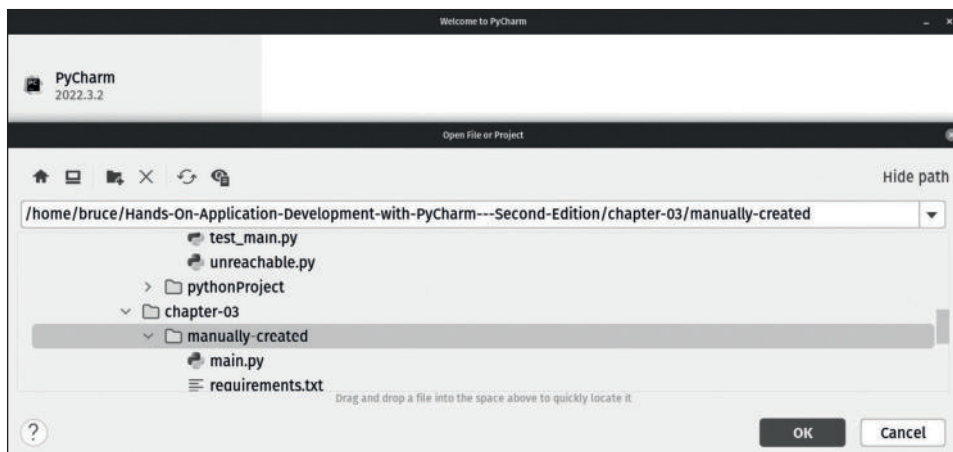


Рис. 3.25. Открытие проекта, созданного вручную вне PyCharm

Для этого проекта я сделал то, чего обычно не делал. Папка виртуальной среды (venv) включена в репозиторий git. Обычно этого не должно быть, но она здесь, чтобы вы могли увидеть следующее маленькое волшебство.

Откройте папку проекта и посмотрите, что произойдет. Если вы ищете широкую демонстрацию возможностей программного обеспечения, это может показаться немного разочаровывающим. Это даже не достойно скриншота. Проект открылся, и он просто сидит там. Это самая крутая часть. PyCharm может видеть папку вашей виртуальной среды и автоматически настраивает ваш проект для вас. Вы можете проверить все это в настройках вашего проекта, как показано на рис. 3.26:

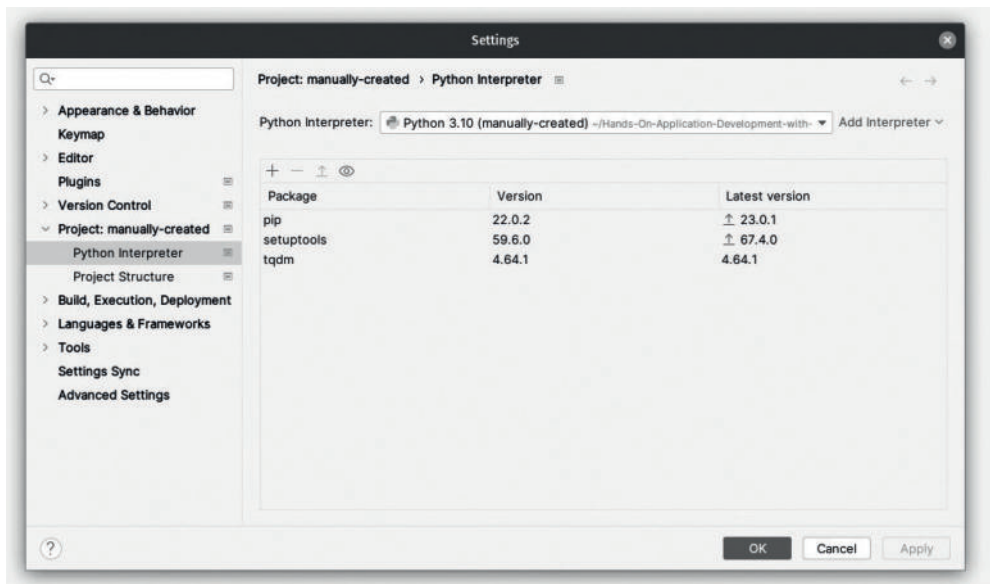


Рис. 3.26. Импортированный проект имеет правильные настройки интерпретатора без каких-либо усилий с нашей стороны

Импорт клонированного проекта из репозитория

В нашем предыдущем примере был показан импорт проекта, который я создал вручную на своем компьютере. Скорее всего, вы импортируете клонированный проект из GitHub или другого репозитория. В этих проектах при клонировании папка виртуальной среды не будет присутствовать. Таким образом, вам придется проделать немного больше работы для настройки проекта.

Я собираюсь повторно использовать тот же проект, но удалю несколько файлов, чтобы PyCharm ничего не помнил о проекте.

Сначала закройте проект в PyCharm, как показано на рис. 3.27:

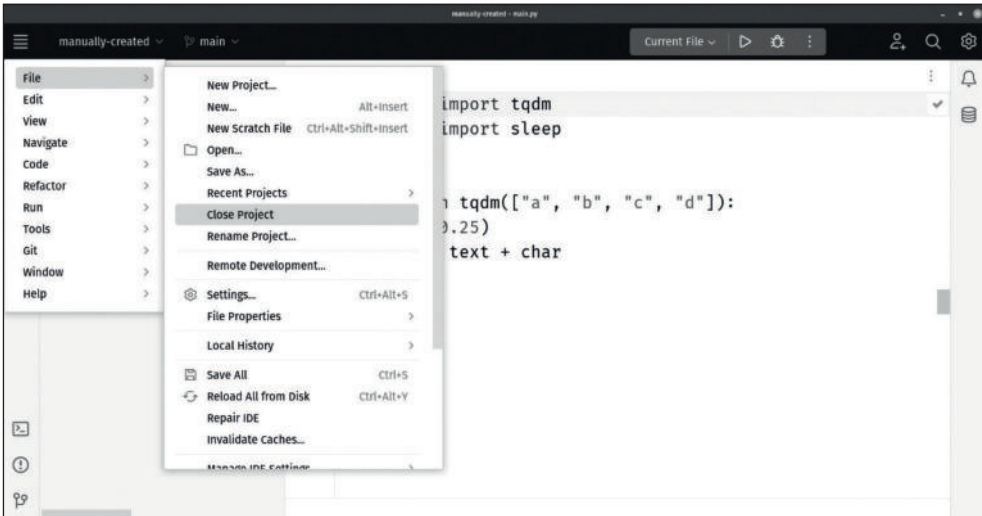


Рис. 3.27. Закрытие проекта

Затем используйте свою операционную систему для удаления папок, показанных на рис. 3.28:



Рис. 3.28. Удаление папок venv и .idea для имитации проекта, который вы, возможно, клонировали из системы контроля версий (VCS)

Мы избавляемся от папки venv и папки .idea. Последние будут скрыты в системах MacOS и Linux, поэтому обязательно включите возможность просмотра скрытых папок. Папка .idea – это папка проекта PyCharm, которая автоматически создается при открытии папки проекта.

После исчезновения этих двух папок PyCharm практически не знает, открывал ли когда-либо проект. Папка по-прежнему будет отображаться в папке последних проектов **Recent**, но это нормально. Откройте папку еще раз с помощью PyCharm. Результат открытия проекта в PyCharm показан на рис. 3.29:

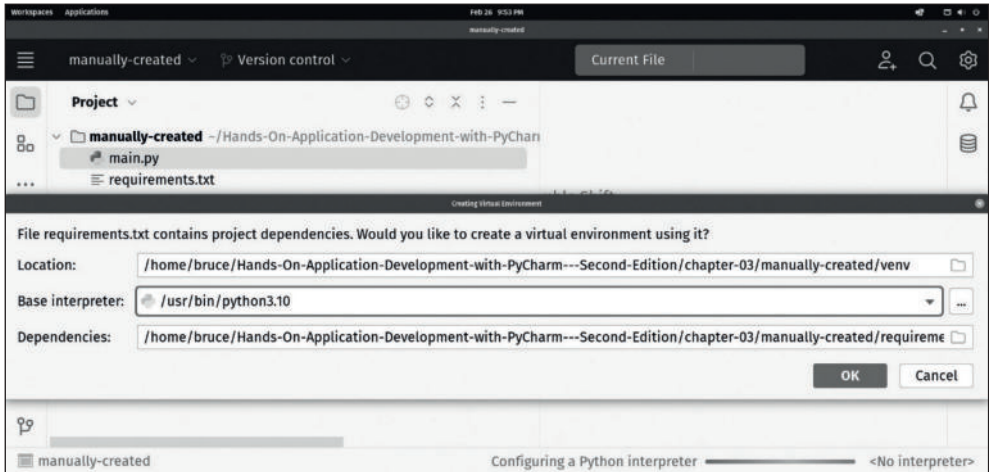


Рис. 3.29. Импорт проекта, созданного вне PyCharm, предлагает создать виртуальную среду

PyCharm видит файл `requirements.txt`, но не может найти подходящую виртуальную среду, поскольку в проекте нет подпапки, содержащей исполняемый файл Python. PyCharm предложит вам создать среду для вашего проекта. В моем случае по умолчанию используется локальная папка с именем `venv`. Она будет использовать исполняемый файл Python 3.10, найденный в `/usr/bin/python3.10`, в качестве основы для виртуальной среды и видит файл `requirements.txt`. Если я нажму **ОК**, PyCharm настроит виртуальную среду, а затем установит для меня мои зависимости.

Работа с неисправными интерпретаторами

Ни одна IDE не идеальна. Иногда PyCharm может запутаться в выборе виртуальной среды. На самом деле это происходит только тогда, когда вы создаете новый проект с кодом, созданным не в PyCharm. Это крайний случай, но рано или поздно вы с этим столкнетесь, поэтому я хочу поговорить об этом скрипте. Возможно, вы готовитесь написать книгу о PyCharm. Хорошо, но это, думаю, чаще случается со мной. Допустим, вы готовите демоверсию. Возможно, вы готовите учебный курс и возитесь с чем-то слишком много или слишком много раз. Если вы похожи на меня, вам нравится, чтобы все было идеально. Вы работали над настройкой проекта и видите списки недопустимых сред, например показанный на рис. 3.30.

Великолепно! Появляется ужасное красное сообщение о том, что базовый интерпретатор неисправен. Нам нужно это исправить. Хорошая новость в том, что вы знаете, что нужно делать. Для начала давайте решим вашу насущную задачу. Вам необходимо настроить базовый интерпретатор, чтобы PyCharm мог завершить импорт вашего проекта. Первое, что стоит попробовать, – это раскрывающийся список базового интерпретатора. По всей вероятности, в списке появится ваша глобальная установка Python. Если да, выберите ее и нажмите **ОК**, и PyCharm сделает свое дело.

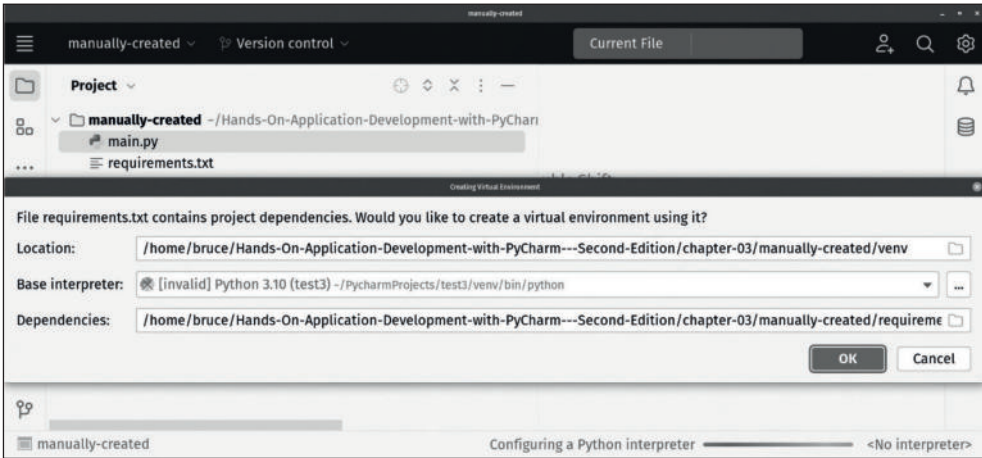


Рис. 3.30. С ошибками неисправного интерпретатора сталкиваются лучшие из нас

Если вы не можете ее найти, вы все равно можете создать работающую среду. Просто нужно создать виртуальную среду так же, как мы это делали, когда создавали наш проект с нуля. Вам нужно будет выйти из диалогового окна, которое вы видите на рис. 3.32, и использовать настройки проекта, чтобы добавить новую среду.

Нажмите **File | Settings**, чтобы попасть в диалоговое окно **Settings** и найти настройки проекта, как показано на рис. 3.31:

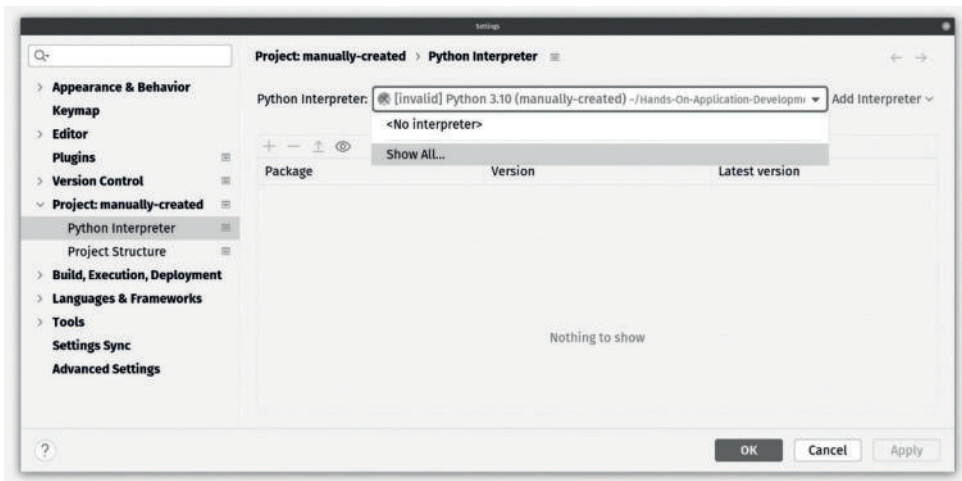


Рис. 3.31. Наихудший скрипт (это случается нечасто, но случается)

Это худший вариант развития событий! В списке нет базового интерпретатора, и единственным вариантом является наша неисправная виртуальная среда. По общему признанию, вам придется приложить немало усилий, чтобы заставить PyCharm сделать это, но обычно это происходит ровно за 5 мин до

того, как вам понадобится PyCharm для работы перед большим количеством людей. Я здесь для вас.

Внизу раскрывающегося списка находится опция **Show All....** Это позволит нам все исправить!

Нажмите на эту кнопку! Вы увидите очень полезное диалоговое окно, показанное на рис. 3.32:

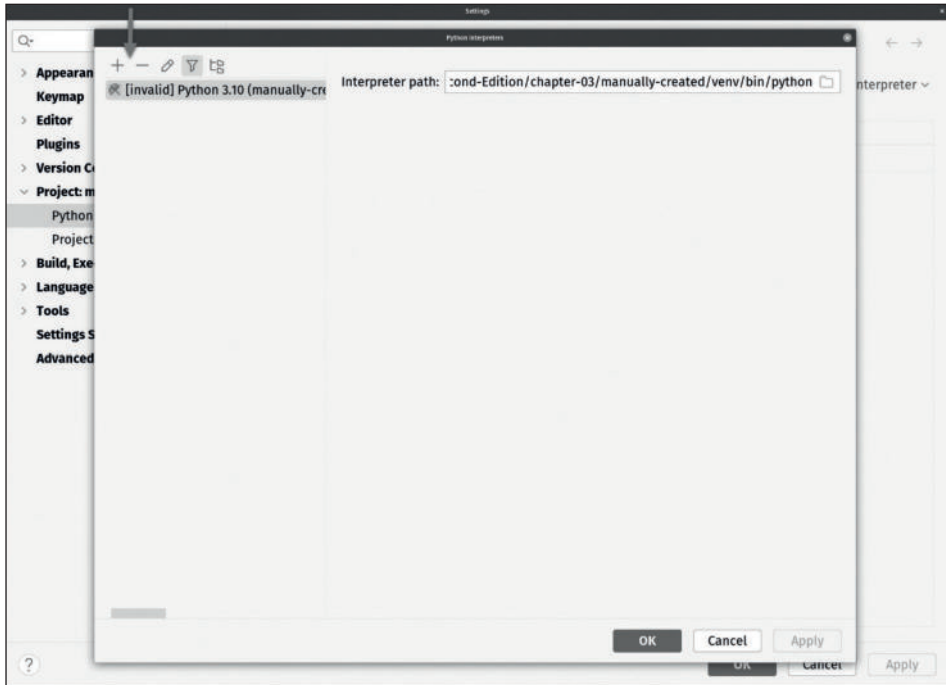


Рис. 3.32. Диалоговое окно интерпретаторов Python позволяет вам управлять всеми средами Python в одном месте

Обратите внимание на стрелку на рис. 3.32. Вы можете использовать значки + и – для добавления и удаления сред; выбрать недопустимую среду и удалить ее с помощью значка –. Затем можно добавить новую действительную среду с помощью значка +. Рабочий процесс добавления нового интерпретатора такой же, как и при его создании с нуля в новом проекте. Мы уже рассмотрели этот процесс, поэтому я не буду повторять его снова.

РАБОТА С КОНФИГУРАЦИЯМИ ЗАПУСКА

Конфигурации запуска в PyCharm позволяют вам настраивать различные способы запуска ваших программ из IDE. Сердцем конфигураций запуска является группа синих кнопок в верхней части главного окна. Следуйте стрелке на рис. 3.33:

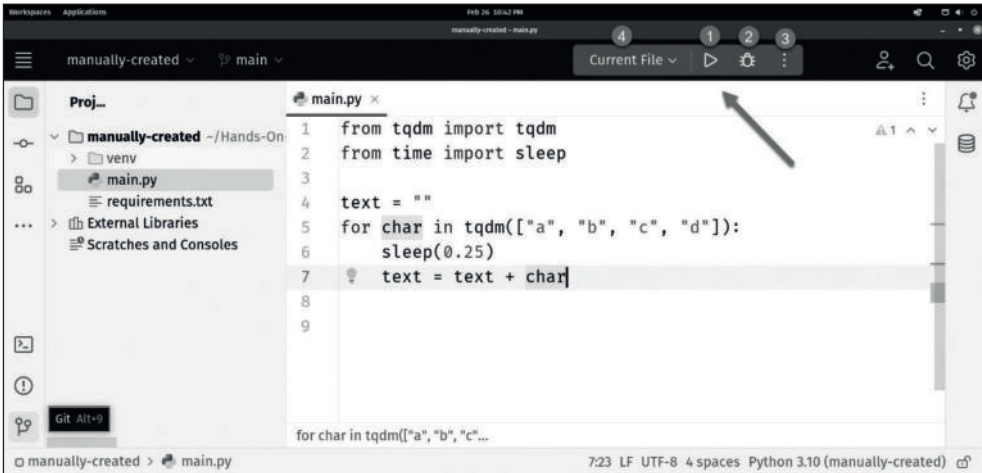


Рис. 3.33. В верхней части экрана имеется несколько кнопок запуска, а также раскрывающийся список, который позволяет работать с различными конфигурациями запуска

Поскольку ранее `manually-created` проект уже был открыт, почему бы нам не продолжить изучение использования этого проекта? Группа кнопок, которую я указал на рис. 3.33, состоит из кнопки **Run**, кнопки **Debug** и раскрывающегося списка, который позволяет вам устанавливать конфигурации запуска и управлять ими. Когда мы импортировали проект, он сгенерировал конфигурацию запуска под названием **Current File**.

Если этот параметр установлен в раскрывающемся меню, файл на вкладке **Focused** будет выполняться при нажатии кнопки **Run** или **Debug**.

1. Это обычная кнопка **Run**. Он просто запускает то, что указано в текущей выбранной конфигурации запуска. В этом случае будет запущен скрипт `main.py`, поскольку он находится на текущей вкладке.
2. Это кнопка **Run** (дебаггер). Нажав на эту кнопку, вы запустите выбранную конфигурацию запуска с подключенным отладчиком. Мы увидим это в действии в главе 6 «Бесшовное тестирование, отладка и профилирование».
3. Кнопка с многоточием содержит дополнительные параметры для запуска профилировщика и проверки покрытия тестами. Мы рассмотрим их в последующих главах.
4. **Run configuration** позволяет выбрать конфигурацию запуска. Существует возможность создавать и редактировать существующие конфигурации, а также создавать новые.

А пока давайте сосредоточимся на самих конфигурациях запуска, поскольку они имеют свои собственные настройки среды, независимые от остальной части проекта.

Кликните раскрывающееся меню с надписью **Current File** и выберите **Edit Configurations...**, как показано на рис. 3.34. Откроется диалоговое окно для управления конфигурациями запуска. Здесь вы можете добавлять, редак-

тировать и удалять свои конфигурации. Обратите внимание, что в версии Professional в этом окне будет доступно больше инструментов, чем в версии Community. Я остановлюсь на версии Professional:



Рис. 3.34. Добавление новой конфигурации запуска для нашего проекта, созданного вручную

В диалоговом окне имеется множество подсказок, позволяющих создать новую конфигурацию запуска. Давайте проигнорируем их, потому что, как только вы их добавите, эти параметры исчезнут. Нажмите значок + в верхней части диалогового окна, как показано на рис. 3.35.

Если вы используете версию Professional, вы увидите длинный список опций, как показано на рис. 3.35. Список редакций Community значительно короче, поскольку в нем есть инструменты только для проектов на чистом Python.

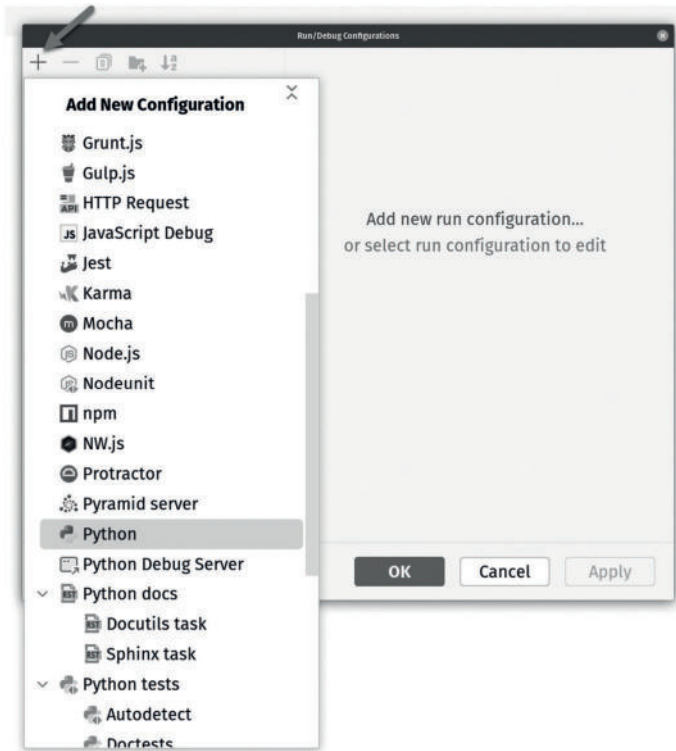


Рис. 3.35. Нажмите кнопку +, чтобы добавить новую конфигурацию запуска (показанный список относится к версии Professional, и вам нужно будет прокрутить вниз, чтобы найти показанный элемент Python)

Я выберу тип проекта Pure Python, поскольку его мы все можем использовать. Каждый вариант может представлять собой отдельное диалоговое окно с разными настройками в зависимости от того, что подходит для вашего выбора. Параметры Pure Python выглядят так, как показано на рис. 3.36.

Здесь очень много настроек. Давайте рассмотрим наиболее важные настройки по номерам, представленным на рис. 3.36.

1. Вы должны дать конфигурации запуска имя. Если вы этого не сделаете, по умолчанию будет использовано имя скрипта. Здесь вы можете использовать понятное для человека имя; оно не должно каким-либо образом соотноситься с запущенным файлом.
2. Эта настройка, вероятно, самая очевидная. Она позволяет выбрать скрипт, который вы хотите запустить. Можете перейти к нему или ввести путь. Вы также можете изменить это имя с файла скрипта на имя модуля, кликнув треугольник рядом с полем ввода текста пути к скрипту. Если вы измените настройку запуска модуля, кнопка **Browse** в папке изменится на многоточие, и вы сможете просматривать модули в своей программе, чтобы найти тот, который хотите запустить.

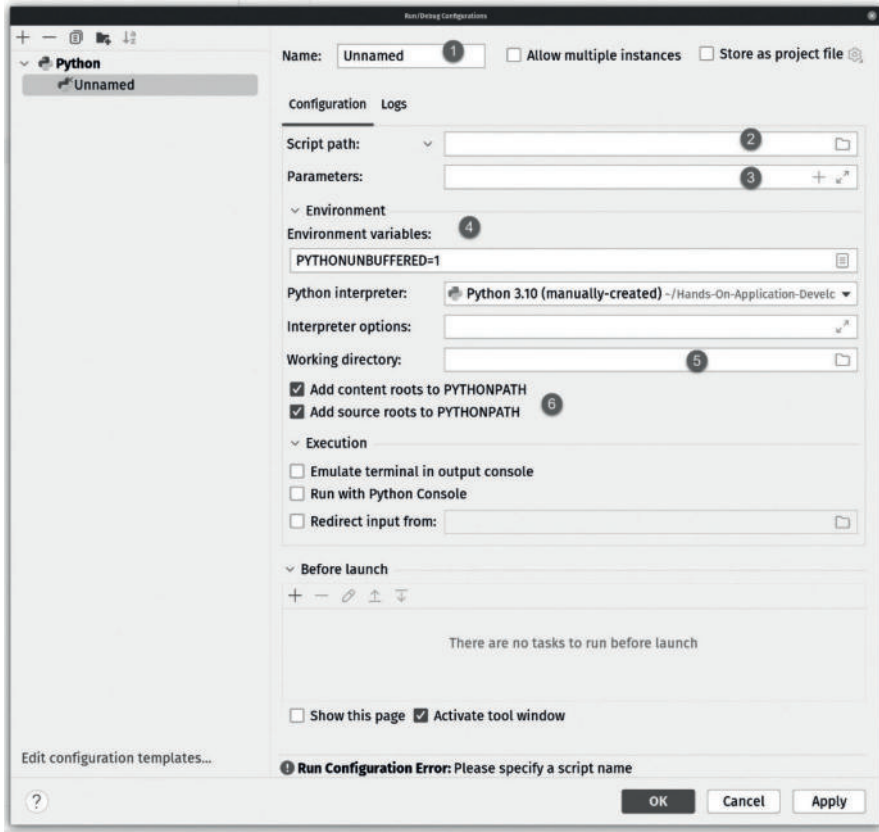


Рис. 3.36. Варианты конфигурации запуска с использованием шаблона Pure Python

3. Установка **Parameters** позволяет передавать аргументы. Если ваша программа ожидает аргументы, используя такую библиотеку, как `argparse`, вы можете передать их сюда. Это текстовое поле довольно интерактивно. Обратите внимание на значок **+**. Нажав на эту кнопку, вы сможете вызвать некоторые макросы, доступные в инструменте. Например, если вам нужен путь к текущему каталогу, вы найдете соответствующую опцию в списке макросов.
4. Это настройки среды. По умолчанию используются настройки проекта. Вы можете установить для конфигураций запуска другую виртуальную среду, отличную от виртуальной среды вашего проекта, если вам необходимо протестировать свое программное обеспечение в разных средах. Вы также можете установить здесь переменные среды, не переходя в операционную систему. Это удобно, если вы полагаетесь на переменные среды для настроек вашей программы. Если вы используете переменные среды уровня ОС для конфиденциальных данных, таких как пароли или строки подключения, я не рекомендую устанавливать их здесь. Эти настройки могут оказаться в файлах конфигурации проекта, а это значит, что они могут попасть в ваш исходный репозиторий. Нет ниче-

го хуже, чем осознать, что файлы вашего проекта на Хакатоне содержат ваш личный пароль или токены доступа к вашему любимому облачному провайдеру. SuperH4xr1337@someevil-hacker-org.darkweb.net (*примечание*: надеюсь, это не настоящий адрес) любит, когда вы это делаете. Есть боты, которые троллят GitHub в поисках подобных вещей, так что будьте осторожны. Лично я использую для этого файлы конфигурации, используя одну из множества библиотек `env`. Я помещу ссылку в раздел «Дальнейшее чтение», если вам это интересно.

5. Вы можете установить рабочий каталог, если какая-либо часть вашего скрипта использует относительные пути к файлам.
6. Этот флажок определяет, внедряются ли корни контента из вашего проекта в `PYTHONPATH` при запуске программы. Переменная среды `PYTHONPATH` управляет тем, где интерпретатор Python ищет модули. Корни контента относятся к настройкам структуры проекта, в которых вы определяете папки контента. Обычно это используется только с веб-проектами. Папка содержимого может содержать изображения или другой статический контент. Параметр **Add source roots to PYTHONPATH** позволяет вам вставлять папки исходного кода, настроенные проектом, в переменную среды `PYTHONPATH`. Я работал над проектами микросервисной архитектуры, где это было полезно. Иногда у вас может быть модуль в другом проекте, который нужно использовать, но он принадлежит отдельному проекту, а не управляется напрямую в текущем проекте. Вы можете настроить исходные корни для включения других проектов с зависимостями, а установка этого флажка сделает их доступными для вашей работающей программы.

Как видите, существует множество опций и способов настройки запуска программы с помощью PyCharm. Если в вашем приложении имеется несколько исполняемых скриптов, можете настроить для каждого из них собственную конфигурацию запуска.

ФАЙЛЫ ПРОЕКТА PYCHARM

В большинстве IDE есть своего рода файл проекта, предназначенный для хранения настроек уровня проекта. Если вы когда-либо использовали какую-либо из IDE Microsoft, вы, возможно, помните такие папки, как `.vscode` для кода Visual Studio и `.vs` в проекте Visual Studio. Java IDE, такие как NetBeans и Eclipse, также используют набор файлов для хранения настроек проекта. PyCharm также имеет набор файлов, хранящихся в папке внутри каждого проекта, которая называется `.idea`. Это название может показаться странным, пока вы не вспомните, что JetBrains начиналась только с одного проекта IDE – IntelliJ IDEA. IntelliJ IDEA заслужила репутацию лучшей IDE для разработки на Java, вне всякого сомнения. Как хорошо, что Google заключила контракт с JetBrains на создание Android Studio; это истинно так – все приложения Android написаны на Java. Все IDE от JetBrains имеют одну и ту же родословную. Все они являются потомками IntelliJ IDEA, поэтому папка проекта называется `.idea`. Помните: пользователи Windows будут ясно видеть эту папку, в то время как в Linux и Mac любое имя папки, начинающееся с точки (`.`), скрыто.

По умолчанию внутри этих папок нет ничего особенно интересного, если у вас когда-нибудь возникнет желание попытаться их отредактировать. На самом деле, если бы вы удалили их, PyCharm просто создал бы их заново при повторном открытии проекта.

Тем не менее в PyCharm есть опции, позволяющие хранить конфигурации запуска в этих файлах, как показано на рис. 3.37:

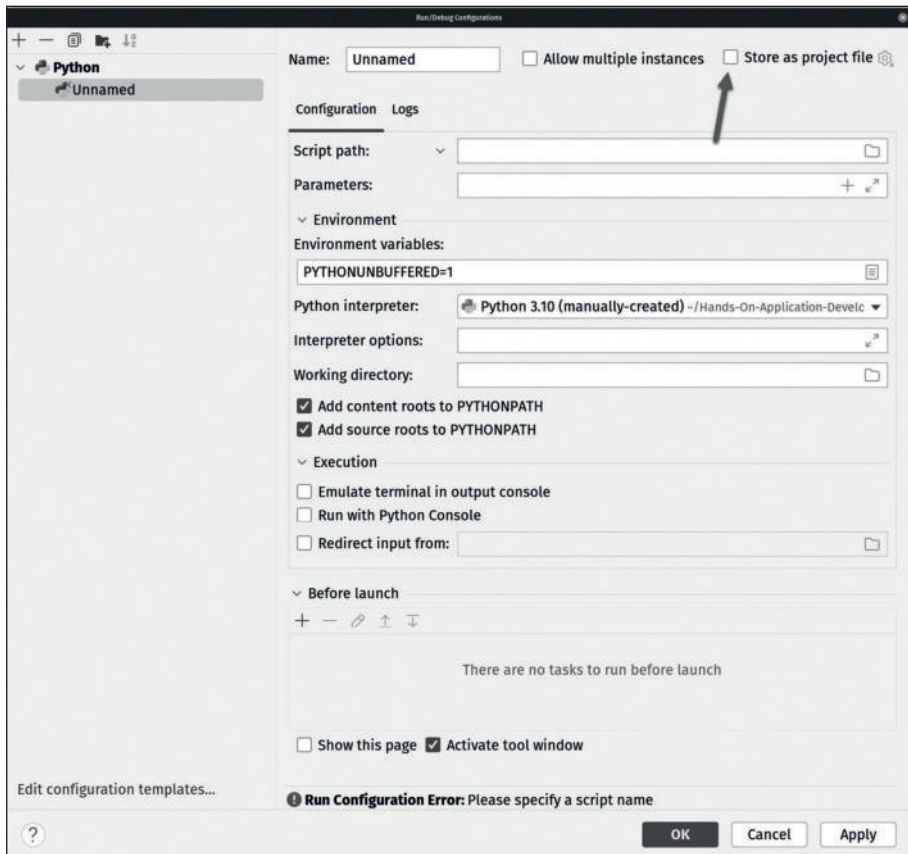


Рис. 3.37. Вы можете выбрать сохранение конфигураций запуска как часть файлов проекта в папке `.idea`

Если вы решите это сделать, у вас будет возможность вернуть файлы `.idea` в свой репозиторий VCS и поделиться ими со своей командой. Это означает, что у всех будут одинаковые конфигурации, поэтому можете провести встречу и стандартизировать структуру вашего проекта, чтобы вы не наступали друг другу на ноги.

КРАТКОЕ СОДЕРЖАНИЕ

В этой главе речь шла о настройке виртуальной среды для проекта Python. Каждый проект обычно имеет свою собственную виртуальную среду. Мы используем виртуальные среды, чтобы изолировать наши проекты от требований дру-

гих проектов и не допустить загрязнения нашей глобальной установки Python множеством глобальных пакетов, которые использовались только для одного текущего проекта.

Виртуальная среда – это копия интерпретатора Python вместе со всеми вспомогательными инструментами, такими как менеджер пакетов и сторонние библиотеки, необходимые для вашего проекта. PyCharm имеет встроенные функции для создания виртуальных сред и управления ими.

Впервые мы видим эти инструменты при создании нового проекта. PyCharm каждый раз предлагает нам создать новую виртуальную среду. Мы также можем выбрать существующую среду, если это целесообразно. PyCharm дает нам возможность в любой момент изменить виртуальную среду проекта.

В нашей виртуальной среде также находятся все сторонние библиотеки, необходимые для нашего проекта. Вы можете найти тысячи готовых модулей на <https://pypi.org>. Эти модули можно установить и использовать в ваших проектах с помощью менеджера пакетов. Самый распространенный менеджер пакетов – `pip`, но существуют и другие, например `conda`, `hipenv` и `poetry`. PyCharm поддерживает все эти инструменты, и вам предоставляется возможность выбрать нужный инструмент при создании своей виртуальной среды.

Выбрав менеджер пакетов, PyCharm дает графический интерфейс, который позволяет вам просматривать, добавлять, удалять и обновлять зависимости вашей библиотеки от PyPI. Вы найдете этот графический интерфейс в настройках вашего проекта, которые включены в общие настройки самого PyCharm.

PyCharm может легко импортировать проекты, созданные вне PyCharm. Все, что вам нужно сделать, – это открыть папку, в которой находится код. PyCharm создает набор файлов проекта в этой папке с именем `.idea`, а затем предлагает вам установить интерпретатор, если он не может найти его в папке проекта. PyCharm также найдет файл `requirements.txt` и предложит вам установить зависимости.

Когда вы будете готовы запустить некоторый код в IDE, PyCharm предлагает мощную систему запуска, которая использует конфигурации запуска, созданные в графическом интерфейсе. PyCharm Professional поставляется со многими типами проектов, каждый из которых имеет свои собственные настройки конфигурации запуска. Мы рассмотрели обширные настройки, имеющиеся в чистом проекте Python, поскольку они доступны пользователям версии Community. Конфигурация запуска может управлять (а в некоторых случаях имитировать) широким спектром настроек, которые обычно приходится задавать вручную на уровне операционной системы.

В целом PyCharm предоставляет вам полный набор инструментов для управления виртуальными средами для разработки и запуска вашего кода. В следующей главе мы сосредоточим внимание на основных инструментах, которые вы будете использовать каждый день для управления, написания и редактирования кода.

Вопросы

1. Что такое виртуальные среды и почему они важны?
2. Какие инструменты использует сообщество Python для создания виртуальных сред?
3. Какие инструменты виртуальной среды поддерживаются в PyCharm?
4. Может ли конфигурация запуска использовать виртуальную среду, отличную от основного проекта? Чем это может быть полезно?
5. Где PyCharm хранит файлы конфигурации проекта?

Дальнейшее чтение

- *Альтернативные реализации Python:* <https://www.python.org/download/alternatives/>.
- Библиотека env: <https://pypi.org/project/env/>.
- Ajitsaria, A. *What is the Python Global Interpreter Lock (GIL)?*: <https://real-python.com/python-gil/>.

Редактирование и форматирование с легкостью в PyCharm

Леонардо да Винчи, великий художник и скульптор, размышлял, что его скульптуры были полностью сформированы внутри каменных глыб из карьера еще до того, как он увидел их мрамор. Да Винчи объяснил, что все, что он сделал, — это удалил лишние куски мрамора, чтобы освободить форму. Другими словами, его шедевры были завершены в его воображении еще до того, как долото коснулось грубого камня. Во многом вы и есть да Винчи. У вас в голове полностью сформированный проект, и вы жаждете показать миру свой шедевр. Вместо того чтобы использовать молоток и долото для написания кода, вы используете PyCharm. В предыдущих главах мы рассмотрели процесс установки и настройки PyCharm. Мы также настроим интерпретатор для вашего проекта. Далее: исследование основных инструментов, которые вы будете использовать для создания своего шедевра, которые в основном находятся в редакторе.

К настоящему моменту вы открыли для себя многие очевидные возможности редактора. Мы знаем, что он автоматически обрабатывает многие правила синтаксиса PEP-8. Мы знаем, что получаем синтаксис с цветовой кодировкой. Мы также заметили, что **интегрированная среда разработки (IDE)** вносит предложения по нескольким различным областям кодирования, начиная от правил стиля проверки и заканчивая автодополнением.

В этой главе основное внимание будет уделено менее очевидным функциям редактора. В документации продукта представлены сочетания клавиш и основные сведения о редакторе, которые не будут обсуждаться в этой здесь. Вместо этого будут рассмотрены следующие темы:

- проверка кода в реальном времени с автоматическими исправлениями, что позволяет вам сосредоточиться на целях разработки, а не на правилах написания кода на Python;

- различные функции поддержки автодополнения кода в PyCharm и способы их использования. Используя их, вы сможете писать код быстрее и точнее. Мы сосредоточимся только на тех инструментах, которые поставляются с PyCharm, а не на сторонних улучшениях **искусственного интеллекта (AI)**, для которых требуются такие плагины, как Kite или GitHub Copilot. Они будут рассмотрены в главе 15;
- инструменты рефакторинга, которые позволяют вам отшлифовать и усовершенствовать ваш код, превратив его в шедевр, которым он может стать благодаря терпению, дисциплине и хорошему инструментарию;
- инструменты документирования, которые переведут вас с уровня «хорошего разработчика» на уровень «мастера-разработчика». Одно дело изобрести потрясающий код. Документирование, чтобы другие могли извлечь из этого пользу, выводит вашу работу на новый уровень.

ТЕХНИЧЕСКИЕ ТРЕБОВАНИЯ

Чтобы продолжить работу с этой главой и остальной частью книги, вам понадобится следующее:

- установленный и работающий интерпретатор Python. Я буду использовать последнюю версию <https://python.org>;
- установленные копии `pip` и `virtualenv`. Вы получаете их автоматически при установке Python в Windows, а в MacOS они включены в каждую систему. Если вы используете Linux, вам необходимо отдельно установить менеджеры пакетов, такие как `pip`, и инструменты виртуальной среды, такие как `virtualenv`. В наших примерах будут использоваться `pip` и `virtualenv`;
- установленная и рабочая копия PyCharm. Установка была описана в главе 2;
- пример исходного кода этой книги с GitHub можно найти по адресу <https://github.com/PacktPublishing/Hands-On-Application-Development-with-PyCharm---Second-Edition/tree/main/chapter-04>. Мы рассмотрели клонирование кода в главе 2.

АНАЛИЗ КОДА, ПРОВЕРКА И ПРЕДЛОЖЕНИЯ

Интеллектуальное завершение кода необходимо для внедрения любого инструмента программирования. Хороший механизм завершения кода – это тот, который учитывает высокоуровневые аспекты программирования, включая особенности синтаксиса языка. Движок также должен учитывать особенности нижнего уровня программы, которую вы пишете. Многие расширенные текстовые редакторы поддерживают автодополнение кода, но им не хватает такого уровня сложности. PyCharm выделяется как исключительно сложный редактор кода, охватывающий как исторические, так и современные аспекты редакторов кода и предлагающий уровень сложности, который превосходит многие другие расширенные текстовые редакторы с точки зрения интеллектуального завершения кода.

Наиболее распространенной формой завершения кода является большой список слов, которые соответствуют вводимому. Список возможностей сужается по мере ввода большего количества букв. Notepad++ является расширенным текстовым редактором, широко используемым разработчиками. Я считаю, что

это обязательный инструмент для быстрого и легкого редактирования, когда мне слишком не терпится дождаться полной загрузки IDE. На рис. 4.1 показан сеанс, в котором я начал набирать код Python:

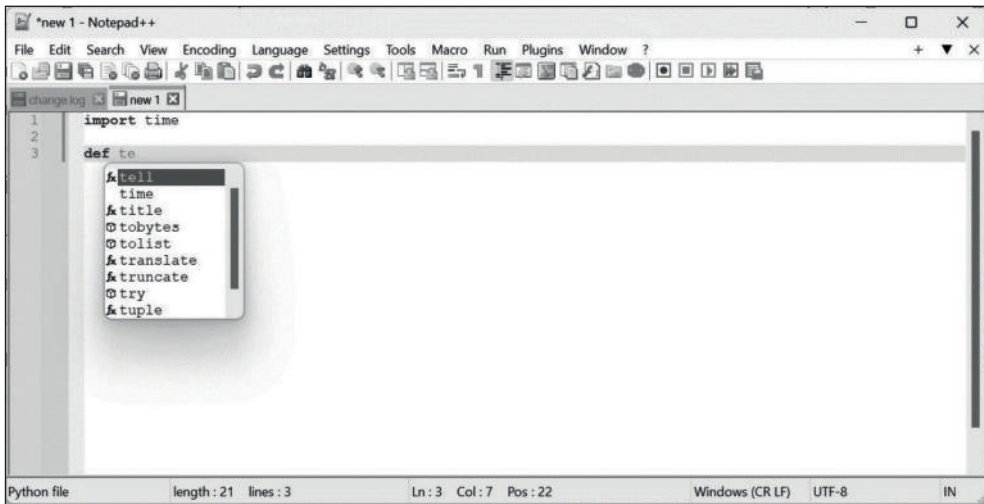


Рис. 4.1. Notepad++ использует очень простой механизм выделения и завершения кода

Инструмент интуитивно не знает, что набирается; поэтому я должен сказать ему, что пишу код на Python. После того как язык установлен, он попытается все автозаполнить, хотя такое упражнение явно бесполезно. На рис. 4.1 я собирался ввести следующее:

```
import time

def test_code():
    pass
```

Это явно не принесет мне никаких наград Jolt¹. На скриншоте вы видите, что список фильтрует известный список слов. Единственная контекстуальная точка отсчета – это знание того, что файл является файлом Python. Это не очень эффективно, но это лучше, чем ничего.

Подобные системы представляют собой не более чем средства проверки правописания. Хотя опытный профессионал может посмеяться над полезностью такой простой системы, самые ранние IDE, демонстрировавшие этот уровень волшебства, впервые использовали функцию, без которой мы теперь не хотим жить: завершение кода.

PyCharm находится на противоположном конце спектра с точки зрения сложности. Как и Notepad++, PyCharm знает ключевые слова, используемые PyCharm. Однако PyCharm способен получить представление о структуре объектов, составляющих стандартную библиотеку. На рис. 4.2 показано, как я ввожу некоторый код в PyCharm, создав для этого примера простой файл:

¹ Jolt Award, награда в индустрии программного обеспечения от Dr. Dobb's Journal. – Прим. ред.

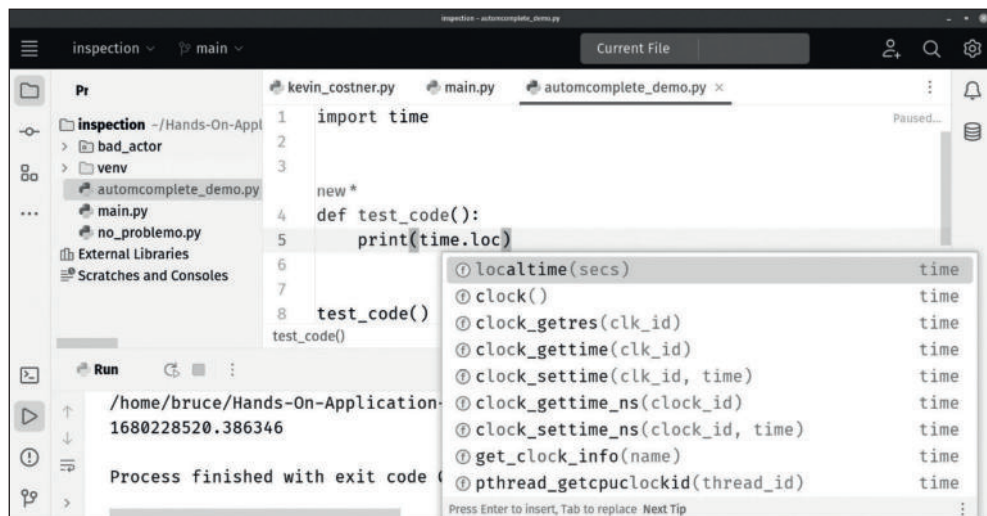


Рис. 4.2. PyCharm выполняет автозаполнение на основе понимания библиотеки времени

В данном случае я импортировал библиотеку `time`¹ точно так же, как в примере Notepad++ на рис. 4.1. Здесь я продвинулся немного дальше. У меня определена функция, и я собираюсь просто распечатать текущее местное время, используя уже импортированную библиотеку `time`. Как вы можете видеть на рис. 4.2, PyCharm предлагает завершение содержимого библиотеки времени.

Завершение автозаполнения

Когда появится список автозаполнения, вы можете нажать *Tab* или *Enter*, чтобы выбрать выделенный вариант. Для перемещения по списку можно использовать клавиши со стрелками вверх и вниз или кликать мышью по любому элементу списка. Однако вы будете работать быстрее всего, если будете держать пальцы на клавиатуре.

К этому моменту вы начинаете ценить систему завершения кода, предлагаемую PyCharm. Давайте подробнее рассмотрим возможности этого важного инструмента.

Пешыте бес ашыбок

Наш упрощенный пример со списком слов из Notepad++, представленный ранее, уже резко контрастирует с более сложными функциями механизма автозаполнения PyCharm. Давайте копнем немного глубже. Если ваш инструмент

¹ Библиотека времени – это стандартная библиотека для обработки времени, предоставляемая Python. Библиотека времени предоставляет функцию синхронизации точного таймера системного уровня, которую можно использовать для анализа производительности программы, а также приостановить выполнение программы. – *Прим. ред.*

основан на списке слов, то в ту самую секунду, когда ваше правописание выходит за рамки правописания, список предложений иссякает. По сути, метод списка слов требует, чтобы вы знали, что именно вы ищете, и одновременно – чтобы это слово было написано правильно.

На рис. 4.3 вы увидите немного другое:

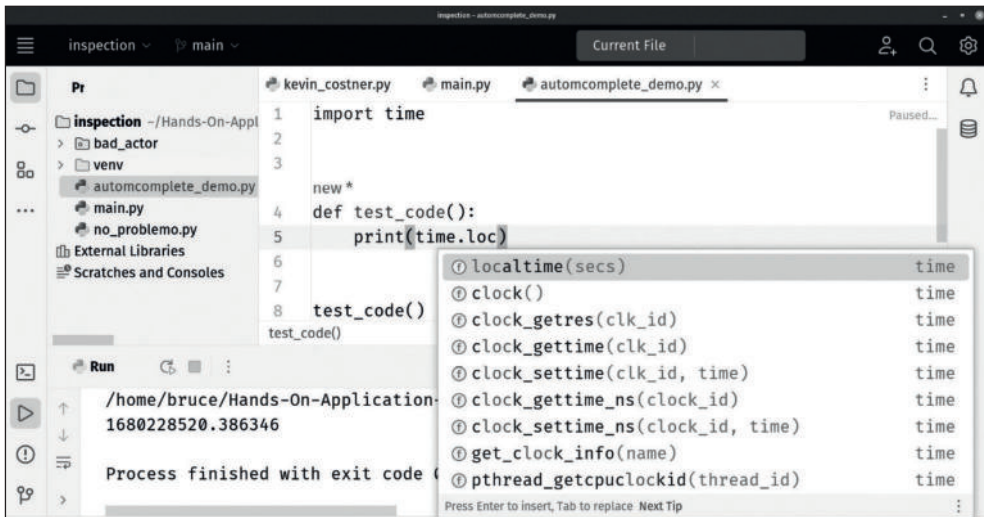


Рис. 4.3. PyCharm находит все варианты, содержащие введенный вами текст

PyCharm предназначен для предоставления предположений на основе набираемых букв. Другими словами, когда вы вводите *cl*, может появиться слово *clock*. Слово *clock* также появится, если вы наберете *lo*, *sk* или любую последовательность букв, содержащуюся в списке совпадающих ключевых слов. Идеально написанные слова не обязательны. Просто зайдите на бейсбол, и слово, которое вы ищете, скорее всего, рано или поздно, появится на табло.

Это понимает ваш код

Написание хорошего кода подразумевает его понимание и умение делать его автозаполнение в зависимости от используемого языка и библиотек. В нашем случае это Python, который имеет огромную стандартную библиотеку по сравнению с другими языками. Python разработан по принципу «батарейки прилагаются». Сравните это с JavaScript, реализованным в Node.js, где единственные библиотеки, которые вы получаете, – это файловые и HTTP-библиотеки. Языки .NET дают вам небольшое ядро. То же самое можно сказать и о Golang. Большинство языков требуют использования менеджера пакетов. Тот факт, что PyCharm может это сделать, сам по себе впечатляет.

PyCharm, будучи отличной IDE, также может понять написанный вами код. На рис. 4.4 показан файл `autocompletem_demo.py`, в который я добавил импорт в файл `no_problemo.py`. В файле `no_problemo.py` есть одна функция, называемая `Perfection()`. Как видите, PyCharm способен заглянуть внутрь файла и обеспечить автодополнение написанного мной кода вместо простого автозаполнения из списка слов на основе языка:

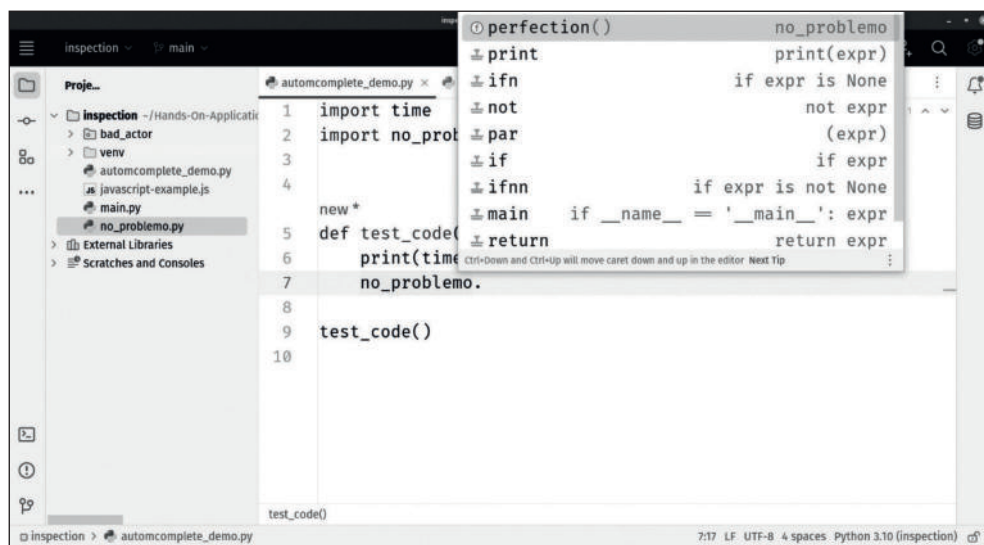


Рис. 4.4. PyCharm предоставляет предложения по автодополнению написанного вами кода, а также стандартного языка Python и стандартной библиотеки

Мне предлагают автодополнение по имени функции, а также подсказки по **сигнатуре метода (method signature)**. Если вы не знакомы с термином «*сигнатура метода*», он просто относится к имени, списку аргументов и возвращаемым значениям функции или метода. Если вы включили подсказки по типам, PyCharm напомним вам имена и типы аргументов, которые требуются функции или методу. Это работает как с модулями, так и с классами, если вы используете **объектно ориентированное программирование (ООР)**.

Завершение постфиксного кода

Традиционное завершение кода в PyCharm вышло на новый уровень, но мы еще далеки от завершения. Обычно нажатие клавиши точки (.) на клавиатуре вызывает появление списка. Теперь мы привыкли к этому списку, содержащему то, что может идти после точки. Однако что, если бы PyCharm мог подсказать вам, что может стоять *перед* точкой? На рис. 4.5 мы видим пример завершения постфиксного кода¹, который вы найдете в файле `post-fix_example.py` в примере кода главы 4:

¹ Постфиксный код – это код, в котором ни одно кодовое слово не совпадает с концом другого кодового слова. Сообщения при использовании такого кода декодируются однозначно и только с конца. – *Прим. ред.*

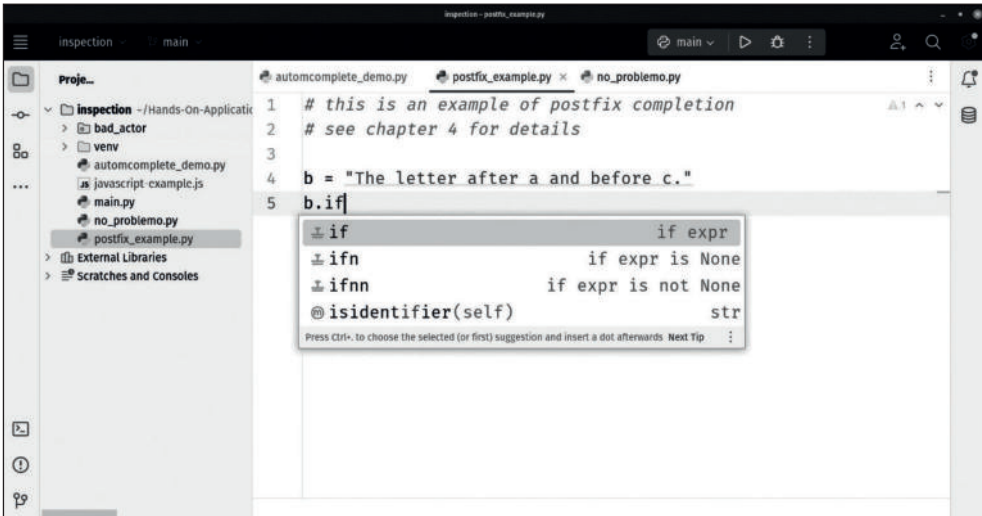


Рис. 4.5. Завершение Postfix в PyCharm может подсказать, что может быть до точки, а не просто что может быть после нее

Я не могу винить вас, если вы сбиты с толку, рассматривая `.if` (`if expr`) как возможность, следующую за явной строковой переменной. Предложение `.if` не является частью Python. Это **постфиксное предложение**. Если вы выполните это предложение, ваш код изменится. Следующий код не является жизнеспособным синтаксисом Python:

```
b.if
```

Поэтому он преобразуется в следующее:

```
If b:
```

Только представьте себе возможности! Однако, если вы не обладаете богатым воображением, взгляните на рис. 4.6, на котором показаны параметры конфигурации **Postfix Completion** в PyCharm.

Естественно, эти параметры полностью настраиваются. Вы даже можете добавить свой собственный! Шаблоны не ограничиваются Python. В списке вы можете видеть, что имеются конфигурации для TypeScript, JavaScript и **языка структурированных запросов (SQL)**. Это важно, поскольку разработка приложений редко ограничивается рамками только языка Python.

Завершение хиппи

Эй, чел! Хочешь увидеть кое-что реально крутое? Это называется – «**циклическое расширение слов**»! Однако только зануды называют это так. Если ты хочешь быть в струе, ты назовешь это **завершением хиппи (hippie completion)**!

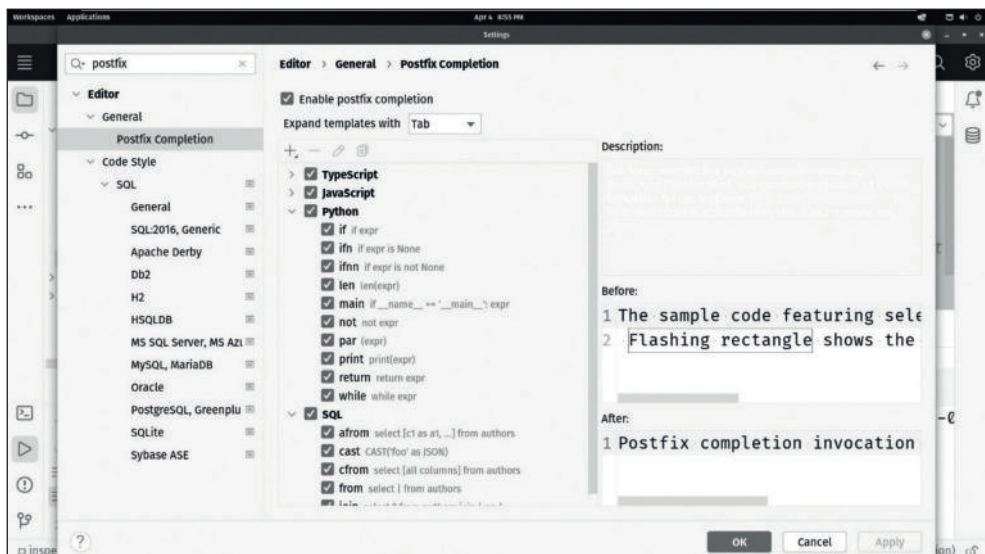


Рис. 4.6. Параметры конфигурации PyCharm Postfix Completion

Завершение хиппи вызывается нажатием `Alt + /` (Windows и Linux) или `⌘ + /` в macOS. После запуска PyCharm проиндексирует все файлы, которые в данный момент открыты, и предоставит предложения по автозаполнению на основе слов в этом контексте. По сути, вы используете простейшую форму автозаполнения – список слов. Список слов генерируется на лету из слов в открытых файлах. Они не обязательно должны быть кодом. Обычные текстовые файлы, отметки, разметка или вообще любой текст будут отображаться в списке предложений на основе простого прямого сопоставления. По мере ввода список сужается. Вы можете увидеть пример на рис. 4.7:

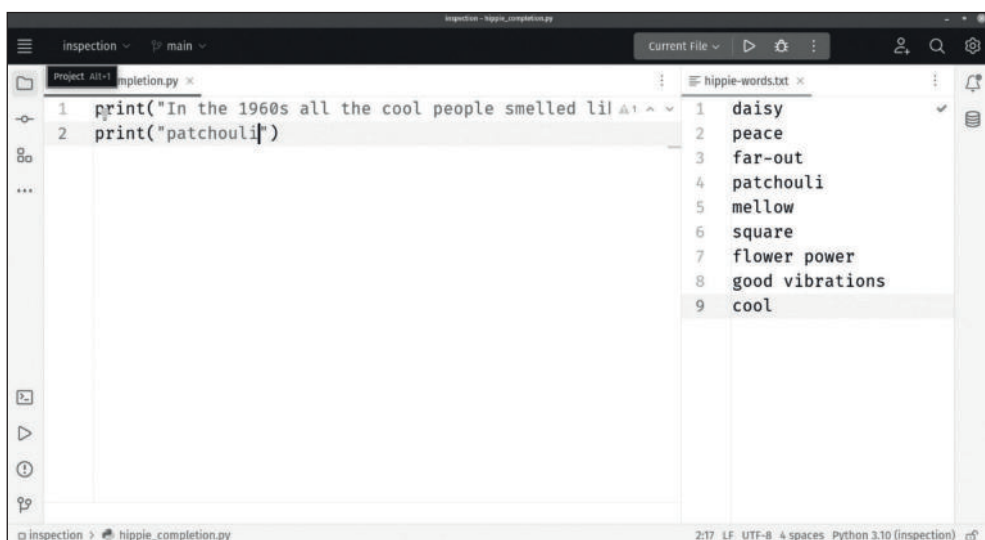


Рис. 4.7. Завершение хиппи – это круто!

Текстовый файл справа содержит список слов, которые я сгенерировал на <https://fungenerators.com/lorem-ipsum/hippie/>. К сожалению, оказывается, что многие знаковые слова поколения хиппи в Соединенных Штатах не могут быть напечатаны в книге такого жанра. У нас самые высокие стандарты! Итак, я отредактировал список, и это все, что осталось. Эти слова случайны, поэтому, пожалуйста, не пытайтесь истолковать из них какой-либо смысл, несмотря на все слухи о том, что на альбомах The Beatles есть секретные сообщения, написанные задом наперед. Код справа можно расширить из этого списка, используя автодополнение хиппи. Чтобы это сработало, я набрал `print("pat`, а затем нажал `Alt + /`. Волшебным образом появилось слово *пачули*¹! Надеюсь, этот пример для вас того стоил. Возможно, я никогда не смогу избавиться от этого запаха на своей клавиатуре!

Вы можете использовать `Alt + Shift + /` или `⌘ + Shift + /`, чтобы включить обратное циклическое расширение слов, которое соответствует от конца слов назад к началу. Если бы сильные мира сего попросили меня (а они этого не сделали, потому что они никогда этого не делают), я бы назвал это «**Завершением Джинджер Роджерс**». Все думают, что Фред Астер был потрясающим танцором во всех великих голливудских фильмах 1940-х годов. Однако никогда не забывайте, что его партнерша по танцам, Джинджер Роджерс, должна была делать все то же, что и Фред, за исключением того, что она делала это задом наперед и на каблуках. Это мое предложение. Кто знает? Если многие из вас твитнут `#GingerRogersCompletion` и сошлутся на эту книгу, она может стать популярной!

Совет от профессионалов хиппи

Повторное нажатие `Alt + /` (Windows и Linux) или `⌘ + /` в macOS позволяет циклически перемещаться по списку. Вы можете просто нажимать комбинацию клавиш, пока не появится искомое слово.

Индексация

Существуют различные механизмы, обеспечивающие различные методы завершения кода PyCharm. Вам может быть интересно, как это работает. Это не черное колдовство, уверяю вас. Ключом к пониманию этого является внимание к моменту загрузки проекта PyCharm. Рисунок 4.8 обращает ваше внимание на нижнюю часть окна приложения PyCharm. PyCharm запускает несколько фоновых процессов, которые прочесывают ваш код и индексируют каждый символ. Затем индекс преобразуется в базу данных в памяти, которая используется различными движками.

Обычно меня не особо волнует, как работает эта магия, но об этом стоит упомянуть, потому что бывают случаи, когда PyCharm тормозит или не отвечает. Если PyCharm работает медленно или автозаполнение не работает, проверьте область экрана, указанную на рис. 4.8, и посмотрите, выполняются ли процессы индексирования. Вы, вероятно, также заметите всплеск производительности процессора, если будете следить за такими вещами. Это временно. После завершения процесса индексирования PyCharm снова будет бегать.

¹ Имеются в виду ароматические палочки, атрибут культуры хиппи. – Прим. ред.

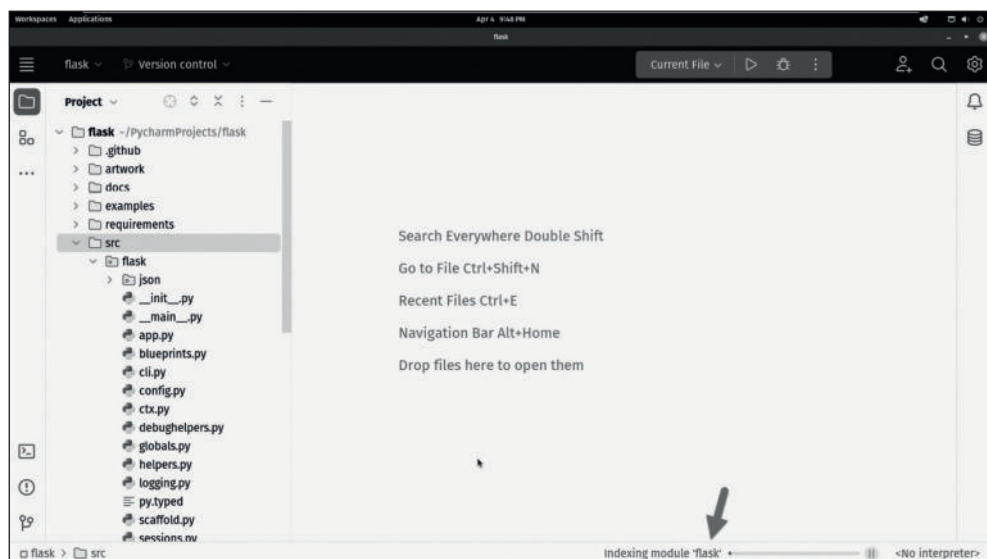


Рис. 4.8. Следите за нижней частью окна PyCharm, чтобы знать, когда выполняются фоновые процессы, такие как индексирование

ЭНЕРГОСБЕРЕГАЮЩИЙ РЕЖИМ

Одна из наиболее загадочных записей в меню PyCharm, показанная на рис. 4.9, – **Power Save Mode**:

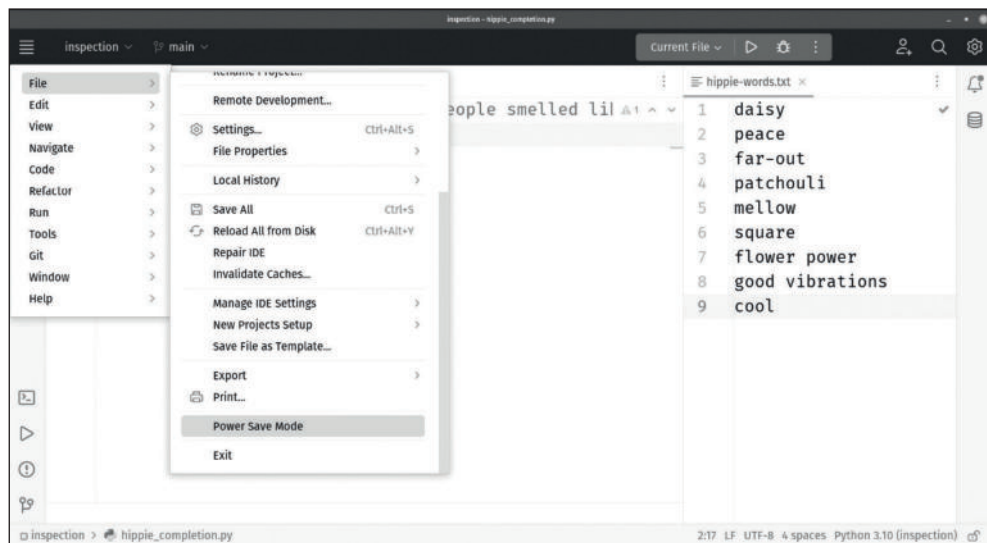


Рис. 4.9. Пункт меню Power Save Mode находится в меню File

Я помню, как впервые нажал на нее. Уличные фонари во всем квартале, где я живу, мгновенно стали ярче. Мой электросчетчик, который еще несколько

минут назад вращался, как циркулярная пила, стал вращаться лениво, еле-еле. Однажды парень из энергетической компании даже позвонил мне и поблагодарил за вклад в спасение планеты.

Хорошо, я все это выдумываю, кроме той части о парне из энергетической компании. Это реально произошло. Параметр **Power Save Mode** предназначен для ограничения энергопотребления PyCharm путем отключения всех фоновых процессов, включая индексацию, используемую для генерации предложений по завершению кода. PyCharm напомнит вам об этом с помощью сообщения, как показано на рис. 4.10. Вы можете не только увидеть предупреждающее сообщение, но также увидеть, что я не получаю никакого завершения в строке 5. Я бы, по крайней мере, ожидал `b.if` из предыдущего примера:

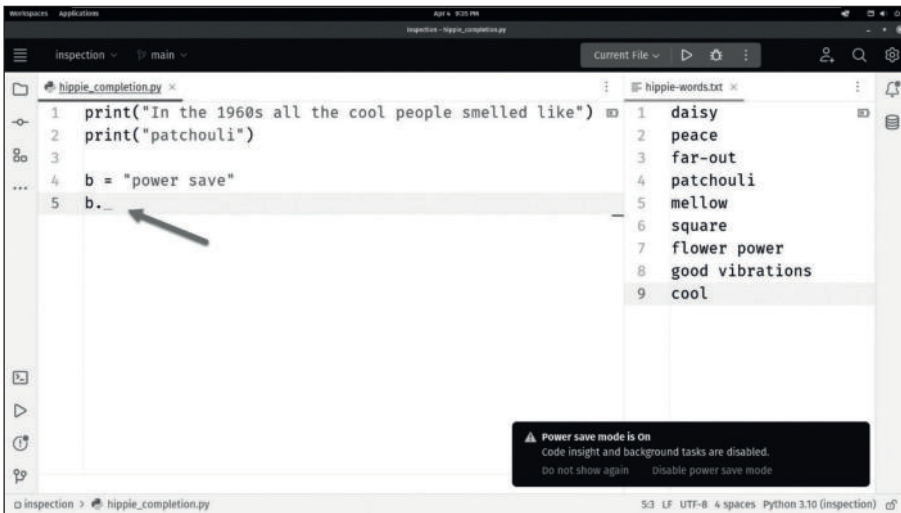


Рис. 4.10. Сообщение напоминает вам, что режим энергосбережения включен и вы не будете получать обычный уровень помощи от PyCharm

Лично я считаю это полнейшим варварством! Как мы можем работать таким образом? Далее вы скажете мне, что у меня нет доступа к интернету, Kite, GitHub Copilot, ChatGPT или Stack Overflow! Вы лишите меня моей подписки на [Packtpub.com](https://packtpub.com), где я смогу получить все электронные книги, которые смогу прочитать, по одной низкой цене. Тогда вы заберете мое кресло Herman Miller Aeron Chair и ограничите меня только одним латте меццо-гранде с полужирной тройной пеной за 14 долл. и посыпкой, наполненной радужным жасмином, в день! Может быть, вы еще заставите меня раскатать свои джинсы?! Бороться! Бороться!

Извините. Я немного увлекся. Излишне говорить, что это, пожалуй, наименее любимая функция IDE. Возможно, это будет полезно, если вы сожрали заряд батареи ноутбука, играя в *Ghost Recon: Breakpoint* во время заграничного перелета, а приземлившись, вам вдруг позвонил начальник, которому нужно что-то срочно исправить. Внезапно вам стало нужно выжать каждую секунду из тех 5 % уровня заряда батареи, которые у вас еще остались. Ненавижу, когда это происходит.

НАСТРОЙКА ЗАВЕРШЕНИЯ КОДА

Настройка в PyCharm – это постоянная тема. Возможно, было бы проще показать вам то, что вы не можете настроить, но я еще ничего не нашел. Завершение кода ничем от ней не отличается. Независимо от того, предпочитаете ли вы очень легкий интерфейс практически без помощи или хотите ее на каждой строке кода, есть способ превратить PyCharm в редактор, который вы захотите использовать каждый день.

Чтобы открыть настройки автодополнения кода, вернитесь к диалоговому окну настроек, которое мы рассмотрели в главе 2. Вы можете легко добраться до него, кликнув пункт меню **File** и выбрав **Settings**. Это вызовет диалог настроек. Если вы помните главу 2, это диалоговое окно огромно! Мы ищем **Editor | General | Code Completion**, как показано на рис. 4.11:

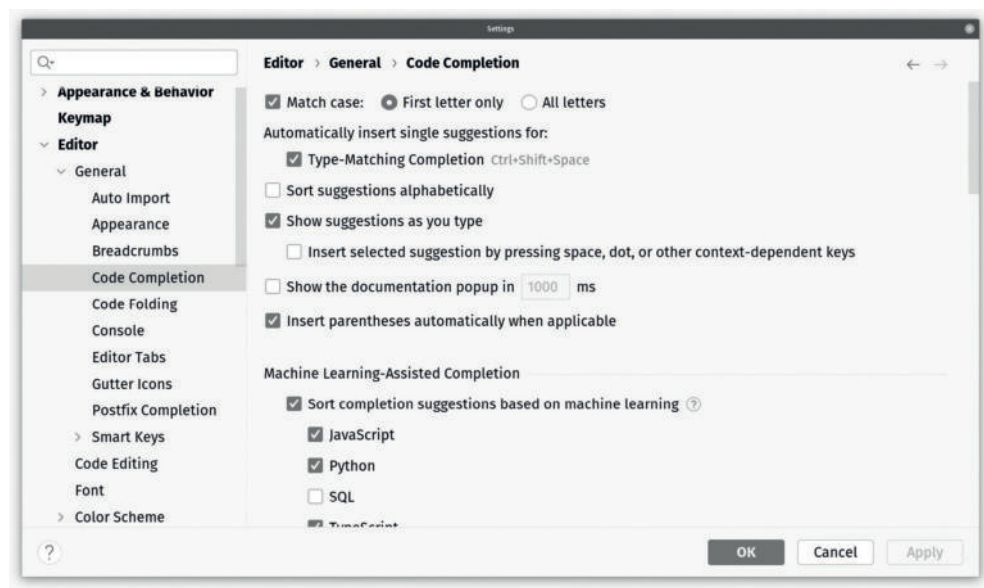


Рис. 4.11. Настройки Code Completion позволяют вам настроить поведение механизма завершения кода PyCharm

Вам следует потратить некоторое время на изучение всего, что вы можете делать на этом экране. PyCharm дает вам отличный контроль! Я обещал, что это не будет утомительным рассмотрением всех вариантов. Вместо этого обратите внимание на самые популярные и потенциально самые полезные настройки.

Сопоставление регистров

Этот параметр, расположенный в верхней части окна, определяет, должны ли элементы в списке предложений соответствовать регистру того, что вы вводите. Например, если бы я хотел ввести исключение для `KeyboardInterrupt` в Python и была включена опция **Match case**, мне пришлось бы ввести заглавную букву K, чтобы правильное имя класса было включено в список предложений. Рядом с флажком

Match case вы также можете выбрать, должен ли сопоставляться только регистр первой буквы, или это должно применяться ко всем буквам.

Лично я всегда отключаю этот флажок, поэтому мне нужно только ввести, например, строчную букву *k*, чтобы воспользоваться преимуществами автодополнения кода. Когда я изучаю новый язык или API, этот параметр может помочь мне в практике, которую я называю *покупкой недвижимости*. Я не знаю, какие свойства и методы доступны, поэтому список в алфавитном порядке может оказаться полезным. Например, в каждом языке программирования есть какой-то класс или библиотека для обработки строк. Можно с уверенностью сказать, что в указанной библиотеке будут функции обрезки, т. е. удаления лишних пробелов с начала, конца или обоих концов строки. Всегда есть какие-то методы `toUpperCase` и `toLowerCase`. Все это жизненно важные части защиты программирования. Ввод пароля, при котором пользователь случайно включает пробел в начале или конце пароля, приводит к разочарованию пользователя. Любую условную логику, которую вы используете, включающую пользовательский ввод, легче обрабатывать, если вы напишете все в верхнем или нижнем регистре. Учитывая, что они настолько важны для нашей работы, мы знаем, что они будут в списке, но в каждом языке они называются по-своему. Я работаю над проектами, которые требуют переключения между двумя или тремя языками, и очень легко ввести неправильное имя функции. Возьмем метод, который преобразует строку в верхний регистр. В JavaScript эта функция выглядит так:

```
let foo = "some user option";
if(foo.toUpperCase() === "SOME USER OPTION"){
  console.log("It matched!");
}
```

Тот же код в PyCharm, который я, возможно, создам всего несколько минут спустя, будет выглядеть так:

```
foo = "some user option"
if foo.upper() == "SOME USER OPTION":
    print("It matched")
```

Чтобы быть эффективным, мне нужно, чтобы слово «upper» совпадало независимо от того, написано ли оно в верхнем или нижнем регистре.

Как и все, что мы видели в завершении кода, и, если уж на то пошло, как и все в программировании, у этой практики есть компромисс. В частности, если параметр **Match case** отключен, иногда список предложений может содержать гораздо больше нерелевантных параметров, что затрудняет поиск правильного API. Однако в то же время вы увидите полный список возможностей, который может помочь вам освоиться, а иногда и обнаружить функции API, о которых вы даже не подозревали.

Сортировка предложений по алфавиту

Как следует из названия, эта опция позволяет сортировать элементы в списке предложений в алфавитном порядке. Эта функция полезна для длинных

списков предложений, которые требуют от разработчика внимательно их прокручивать, чтобы найти то, что они ищут, если они не упорядочены в алфавитном порядке.

Мы неоднократно видели динамическую природу PyCharm, и эта функция еще раз демонстрирует ее. В частности, взаимодействуя со списком предложений в редакторе, вы можете в любой момент изменить порядок элементов в списке, кликнув значок, расположенный в правом нижнем углу окна предложений, как показано на рис. 4.12:

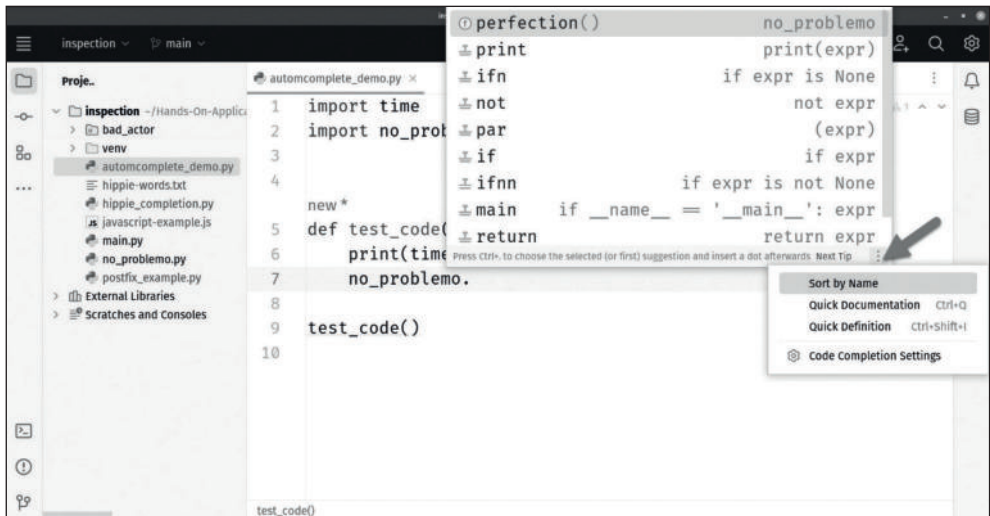


Рис. 4.12. Вы можете изменить способ сортировки предложений внутри самого списка

Нажатие на элемент ... (многоточие) позволяет изменить способ сортировки предложений: по релевантности или в алфавитном порядке по имени.

Завершения с помощью машинного обучения

Этот новый вариант одновременно и волшебный, и пугающий. Включенный по умолчанию, PyCharm будет обучать модель AI на основе вашего кода. Это позволяет PyCharm вносить предложения, основываясь не только на вашем коде, но и на коде тысяч других разработчиков. Традиционное автодополнение кода обычно предлагает варианты следующего ключевого слова, свойства, имени метода или параметра, который вы собираетесь ввести. Не удивляйтесь, если PyCharm предложит вам полные функции или блоки кода с завершением с помощью машинного обучения. Вы увидите это для распространенных задач, таких как подключение к базам данных, работа с датафреймами pandas или проверка ввода пользователя.

Настройки позволяют вам включить различные языки, поддерживаемые PyCharm. Python, JavaScript и TypeScript включены по умолчанию. По умолчанию не включен только SQL – специальный язык, используемый для работы с реляционными базами данных. Я подозреваю, что это связано с тем, что для вашего предпочтительного диалекта SQL существуют дополнительные настройки как на глобальном уровне, так и на уровне проекта, которые содей-

ствуют предложениям. Было бы неловко ожидать сжатого списка предложений, учитывая количество баз данных, поддерживаемых PyCharm, и различия в их реализации языковых элементов, не определенных стандартизированным SQL. Мы рассмотрим SQL и реляционные базы данных позже в главе 11. Я обещаю, что этот опыт не оставит вас банкротом.

Отображение всплывающего окна документации через [...] мс

Когда вы включите эту функцию, то увидите документацию в дополнение к коду предложения. Вы сможете понять, что делает код, который вы вводите, вместо того чтобы слепо принимать предложения. Это отлично подходит для новых разработчиков, независимо от того, являются ли они новичками в кодировании или просто новичками в Python. Преимущество этой функции заключается в том, что вы можете динамически просматривать документацию по всем предлагаемым элементам, просто перемещая курсор вниз по элементам.

Это особенно полезно при работе с классами и методами, имеющими схожие API. Мы обсудим эту функцию, а также другие функции, связанные с документацией, в последнем разделе этой главы.

Инфо параметров

Прокрутите вниз раздел JavaScript, как показано на рис. 4.13, и вы увидите предложение по информации о параметрах:

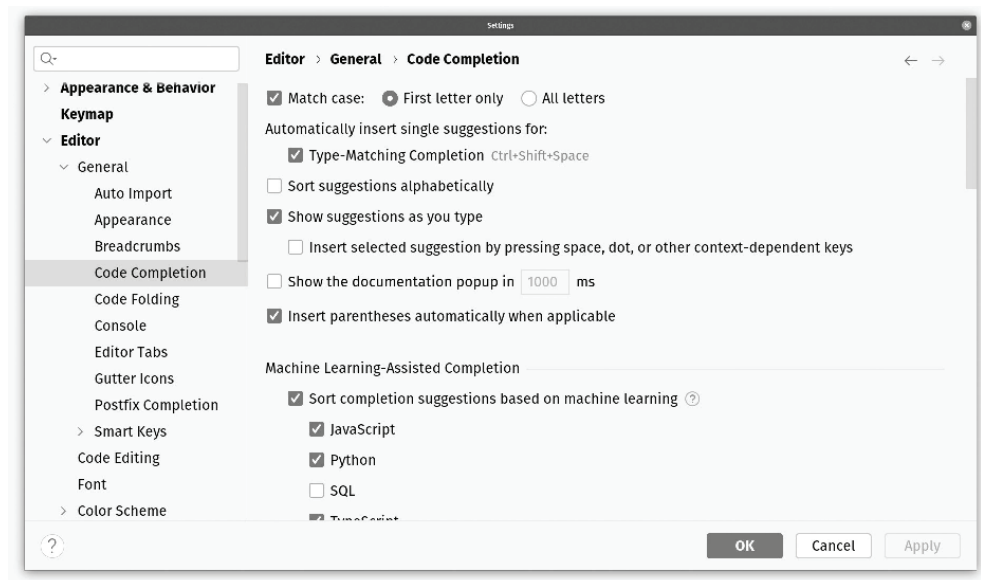


Рис. 4.13. Прокрутите вниз страницу JavaScript, чтобы найти варианты отображения информации о параметрах

Первый вариант прост. Он контролирует количество времени, которое должно пройти до появления предложения. В целом предложения хороши,

если только вы не преподаете, не проверяете код или не проводите какую-то демонстрацию, в этом случае они могут загромождать ваш экран. Иногда полезно увеличить ограничение по времени, чтобы предложения или документация отображались только в том случае, если вы задержитесь на несколько секунд.

Второй вариант позволяет переключаться, показывая полную сигнатуру метода. Мне нравится эта функция. Подсказка по коду покажет вам всю сигнатуру метода, чтобы вы могли видеть все аргументы одновременно.

Сигнатура метода однозначно определяет функцию или метод в области, где она объявлена. Она состоит из имени функции наряду с именами и, желательно, подсказками для типов аргументов функции, а также некоторыми подсказками о типе возвращаемого значения. Они не уникальны в Python. На самом деле в Python они немного нечеткие. Теперь сравните его со статическим языком, таким как C#, который использует более строгие структуры программирования. В Python вы можете использовать подсказки типов, которые помогают разработчикам запоминать ожидаемые типы передаваемых аргументов. Давайте рассмотрим пример сигнатуры метода без подсказок:

```
a = 5
b = 6

def add_two_ints(first, second):
    return a + b
```

Хорошо. Этот код будет работать так, как задумано, и намерение разработчика ясно. Давайте посмотрим на ту же функцию с подсказками:

```
a = 5
b = 6

def add_two_ints(first: int, second: int) -> int:
    return a + b
```

Это намного лучше! Теперь мы точно знаем типы, которые ожидаются в качестве входных параметров, и знаем тип, который будет возвращен. Ранее я говорил, что сигнатуры методов в Python «нечеткие». Я сказал это потому, что оба примера кода будут работать. Python полностью игнорирует подсказки во время компиляции. Подсказки просто используются инструментами и облегчают чтение и понимание вашего кода Python. Добавление подсказок в ваш код, когда это возможно, улучшит способ просмотра сигнатур ваших методов вами и вашими товарищами по команде, если вы включите параметр **Show full method signatures**.

Анализ кода и автоматические исправления

Завершение кода – стандартная функция большинства редакторов кода и IDE. Как мы видели, не все механизмы завершения созданы одинаково. То же самое можно сказать и о механизмах анализа. Механизм анализа кода – это

расширение автодополнения кода в концепции, если не в реализации. Автодополнение кода пытается предсказать код, который вы пишете, и помогает вам закончить его быстрее. Анализ кода проверяет написанный вами код и пытается определить, будет ли он работать при запуске. Как и в случае с завершением кода, здесь существуют разные уровни сложности и разные процессы, исследующие разные вещи.

Самая простая форма анализа называется *линтингом*. Практически в каждом языке программирования есть инструмент для проверки, и Python не является исключением. Хотя выбор есть из чего, PyCharm по умолчанию использует популярную библиотеку `pylint`.

Линтер проводит неявный анализ вашего кода посредством процесса сопоставления с образцом. Существует две операции проверки: **логическая** и **стилистическая**. Логическая проверка ищет ошибки кода, код с потенциально непреднамеренными результатами или побочными эффектами, а также опасные шаблоны кода. Стилистическая проверка ищет код, который не соответствует общепринятым соглашениям. Для Python это меньшая проблема, поскольку в языке уже есть строгий набор правил форматирования кода, называемый **Python Enhancement Proposal #8**. Никто это так не называет. Специалисты называют его просто **PEP-8**.

В совокупности вы можете думать о `pylint` и, как следствие, обо всех линтерах, что он очень похож на средство проверки орфографии и грамматики для обычного текста. Линтер ищет ключевые слова с ошибками, неверный код и очевидные синтаксические ошибки. Линтер также может обеспечивать соблюдение правил стиля, хотя на самом деле Python уже разработан для обеспечения соблюдения правил, чтобы сделать ваш код максимально удобочитаемым для человека.

Одно дело указать на проблемы в вашем коде, но гораздо полезнее, если инструмент также предлагает и даже реализует исправления этих проблем. То же самое относится и к людям. Легко указать на недостатки. Это может сделать кто угодно. Хороший совет о том, как исправить свои недостатки, полезнее критики. Итак, помимо линтинга, PyCharm предлагает систему, которая поможет вам решить проблемы, выявленные линтером.

Обнаружение проблем

Обнаружение проблем выполняется PyCharm в режиме реального времени по мере ввода кода. Здесь играет роль процесс индексации, о котором было упомянуто ранее, но мы к этому еще вернемся. Во-первых, давайте сосредоточимся на видимом интерфейсе редактора, который показывает, в чем заключаются ваши проблемы. Есть четыре места для поиска, как показано на рис. 4.14.

Правая полоса редактора (1) покажет вам, где находятся все проблемные строки в открытом в данный момент файле. Этот разделитель представляет собой сжатое миниатюрное представление вашего файла. Довольно часто файл состоит из сотен или даже тысяч строк. Можете кликнуть область в желобе, где вы видите цветные метки, и редактор прокрутит ее до этого места.

PyCharm классифицирует проблемы на три основные категории: ошибки (красный), предупреждения (желтый) и слабые предупреждения (серый). Об этом сообщается для всего файла в верхней части столбца желоба с указа-

нием количества проблем каждого типа (2). Помимо цветов, эта область дает вам разные значки. Значок ошибки представляет собой круглый красный круг с восклицательным знаком внутри него. Предупреждения представляют собой треугольник с восклицательным знаком. Слабые предупреждения также обозначаются треугольником с восклицательным знаком, но они выглядят значительно тусклее. Если проблем не обнаружено, вы получаете зеленую галочку. На рис. 4.15 показаны два файла. У одного нет проблем (1), а у другого есть ряд проблем в разных категориях (2):

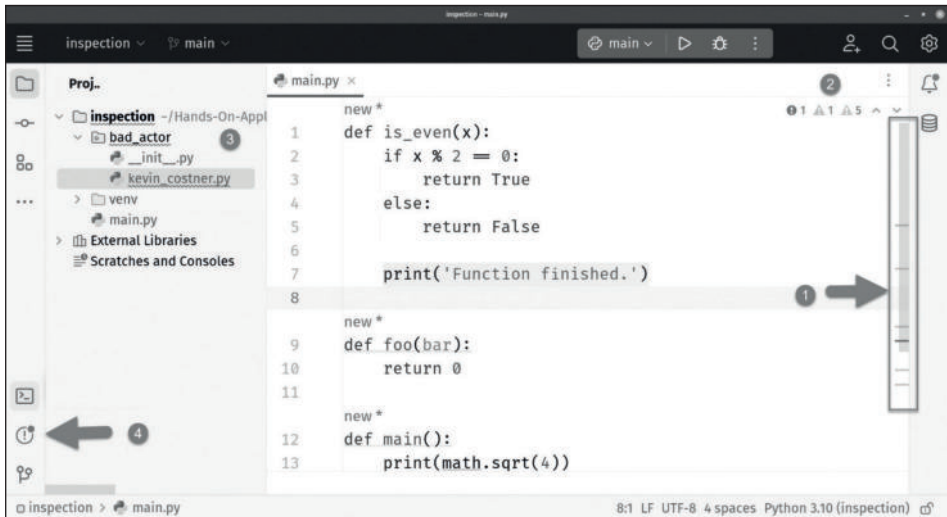


Рис. 4.14. Четыре места пользовательского интерфейса сообщают вам о наличии проблем

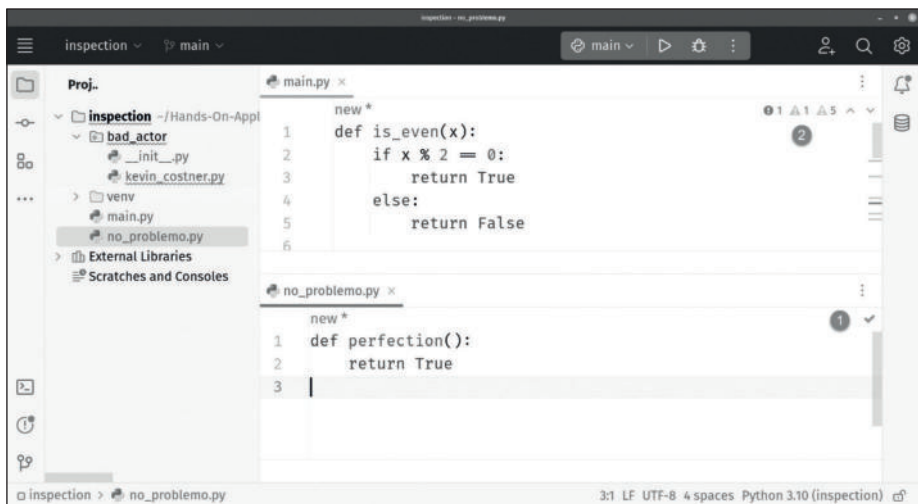


Рис. 4.15. Файл внизу лишен каких-либо проблем, а файл сверху не так удачен

Все, что отмечено красным, является ошибкой, которая, вероятно, помешает вашей программе работать правильно или работать вообще. Предупрежде-

ние означает, что ваш код, вероятно, будет работать, но в некоторых очевидных случаях он не будет работать должным образом. Слабое предупреждение обычно представляет собой незначительный недостаток, например имя переменной, не соответствующее слову из английского словаря. Если для вашего языкового стандарта выбран язык, отличный от английского, PyCharm будет отмечать слова с вашего местного языка.

Второе место, где следует искать проблемы, – это само окно редактора. Вернитесь к рис. 4.15, строка 13. Вы увидите (поверьте мне) волнистое красное подчеркивание под словом `math`. Цвет линий соответствует серьезности проблемы. Правый желоб показывает, где находится проблема, а подчеркивание показывает проблему непосредственно на проблемной линии. Если навести указатель мыши на подчеркнутое, вы получите описание проблемы. Я подробнее расскажу об этом позже в этой главе, когда мы будем говорить о **намерениях (intentions)**, которые представляют собой предложения по устранению проблем в вашем коде.

Третье место, где вы найдете индикаторы проблем, – это проводник проекта. Вернитесь к рис. 4.15, область 3. Существует пакет Python под названием `bad_actor`. Этот файл не открыт на рис. 4.15, но я открыл его, когда создавал, набрал неполный код, а затем закрыл файл. Вы можете увидеть открытый файл на рис. 4.16:

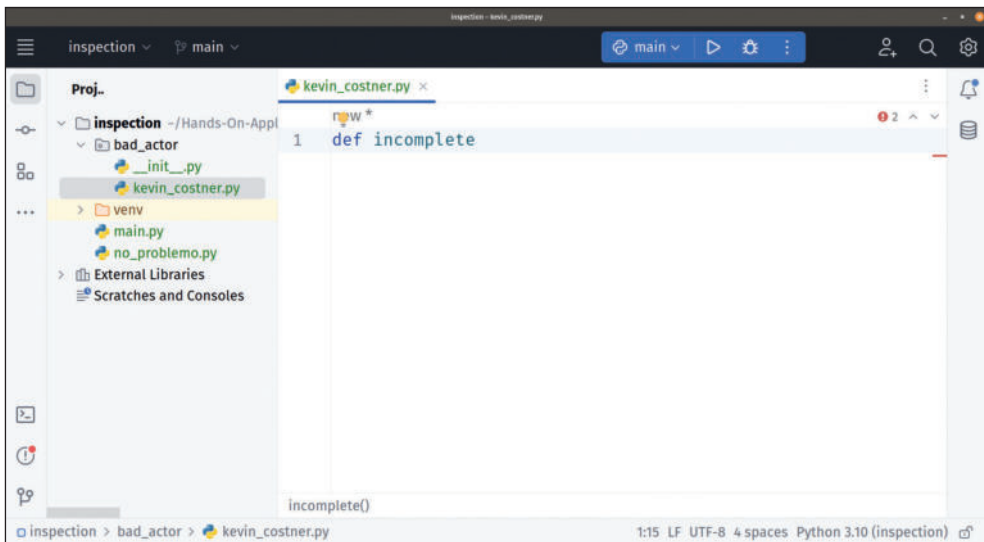


Рис. 4.16. Вот я накосячил! Начал набирать, но так и не закончил

Если в файле, над которым вы сейчас не работаете, есть ошибка, вы увидите ее в списке файлов. Предупреждение распространится по системе каталогов. В верхней папке проекта есть красная волнистая линия, еще одна в папке пакета и еще одно подчеркивание под файлом, в котором возникла проблема. PyCharm не проверяет закрытые файлы. Он проверяет только файлы, которые вы открываете, но как только обнаружит проблему, запомнит, где она находится, и продолжит предупреждать вас, пока вы ее не устранили.

Плохой актер

Мне нужен был образцовый плохой актер для предыдущего визуального каламбура. Кевин Костнер – плохой американский актер. Он, конечно, не самый худший, но легко входит в топ-10 из них. Если не верите, посмотрите фильм «Танцы с волками». Если вы все еще мне не верите, посмотрите «Телохранитель». Если вы все еще считаете его великим, посмотрите «Водный мир» и тогда поймете, ведь он финансировал этот фильм из собственного кармана, потому что был убежден, что он станет мегахитом.

Четвертое место – окно проблем. На рис. 4.17 вы можете увидеть красную точку над значком в области 4 на скриншоте. Кликните значок, и откроется окно проблем со списком неисправностей, как показано на рис. 4.17:

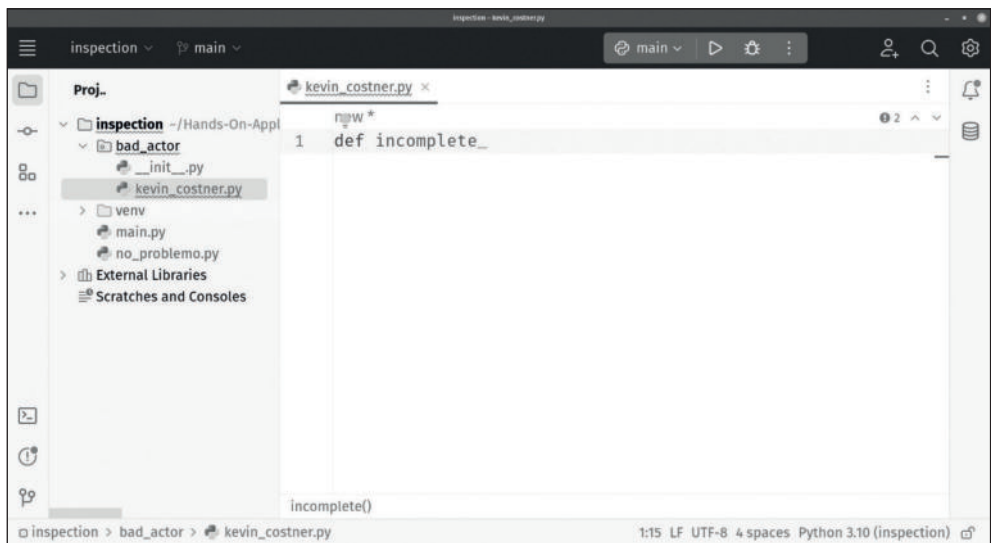


Рис. 4.17. Откройте окно проблемы, чтобы увидеть список всех ваших проблем. Отображаются только ваши проблемы с кодом. Жизненно важные проблемы не показаны

Синтаксические ошибки

Синтаксические ошибки обычно являются самыми мелкими ошибками, обнаруживаемыми в IDE. Мы уже видели несколько. На рис. 4.17 показано неполное определение (дефиниция) функции, поэтому оно помечено красным как синтаксическая ошибка. На рис. 4.14 в строке 13 слово `math` подчеркнуто красной линией. PyCharm распознает это как ссылку на библиотеку Python `math`, которую я не импортировал. Это вызывает неразрешенную ошибку ссылки. Ошибки такого типа всегда отмечаются красным цветом как серьезные ошибки, поскольку они препятствуют запуску программы.

Дублированный код

Если у вас есть привычка копировать и вставлять код внутри своего проекта или даже между разными проектами, вы можете ожидать предупреждения от PyCharm. Дублированный код – признак того, что у вашего проекта проблемы.

Лучше всего следовать концепции «**Не повторяйся**» (DRY). Я скажу это еще раз. Вы хотите, чтобы ваш код всегда был DRY (сухим). Никогда не повторяйте свой код путем копирования и вставки. Убедитесь, что он DRY. Хорошо, я остановлюсь, если вы пообещаете прислушаться к предупреждениям PyCharm о том, что код не является DRY.

Когда вы обнаруживаете эту проблему в одном проекте, вы сможете исправить ее, поместив дублированный код в функцию и вызвав функцию из частей вашего кода, где находятся дубликаты.

Если вас пометили за копирование и вставку между проектами, вам следует передать дублированный код в пакет Python, который можно будет использовать совместно между проектами.

Проблемы PEP-8

Линтер PyCharm предупреждает вас о проблемах со стилем в вашем коде, которые нарушают PEP-8. Самая большая проблема для разработчиков, начинающих знакомиться с Python, – это правила, касающиеся пробелов. Отступы и пустые строки между функциями – все это часть правил PEP-8, разработанных для обеспечения читабельности вашего кода. Большинство проблем PEP-8 помечены как предупреждения.

Мертвый код

Это моя личная любимая мозоль. Кто-то пишет код, который в конечном итоге заменяется другой функцией. И старая, и неиспользуемая функция находятся в файле кода вместе с новой. Возможно, у них похожие имена. Они могут даже находиться в разных файлах. Когда я был ребенком, у меня на стене висел плакат под названием «Законы технологий Мерфи». Плакат отражал пессимистическое, но, по моему опыту (и, возможно, вашему, если вы занимаетесь этим какое-то время), совершенно точное мировоззрение. Вот примеры законов Мерфи применительно к технологиям.

- Глядя на рельсы, невозможно определить, в каком направлении ехал поезд.
- Логика – это систематический метод, позволяющий уверенно прийти к неправильному выводу.
- Всякий раз, когда система становится полностью определенной, какой-то дурак обнаруживает что-то, что либо упраздняет систему, либо расширяет ее до неузнаваемости.
- В технологиях доминируют те, кто управляет тем, чего не понимает: если бы строители строили здания так, как программисты пишут программы, то первый же появившийся дятел уничтожил бы цивилизацию.
- Область внимания компьютера равна длине его электрического шнура.
- Эксперт – это тот, кто знает все больше и больше о все меньшем и меньшем, пока не узнает абсолютно все ни о чем.

Это актуально, потому что, по крайней мере, для меня вероятность того, что я найду и попытаюсь изменить мертвый код (считая его живым и очевидным источником всех моих проблем) в программной системе, асимптотически приближается к 100 %. Незначительная разница между 99 и 100 %, похоже, зависит от моего нынешнего уровня кофеина и от того, пропустил ли я завтрак. Эти эффекты оказываются обратно пропорциональными.

Я благодарю бога за систему, которая предупреждает меня, что я смотрю на мертвый код. Моя обычная напыщенная речь заключается в том, что вам следует удалить мертвый код. Вам это не понадобится, а если и понадобится, то для этого и нужны системы контроля версий.

Несоответствие сигнатуры метода

Несоответствие сигнатуры метода возникает, когда функция требует больше или меньше аргументов, чем указано вами. PyCharm предупредит вас, когда это произойдет.

Дорога к хорошему коду вымощена намерениями PyCharm

Теперь, когда мы потратили время на изучение наших недостатков, давайте рассмотрим некоторые инструменты, которые помогут нам их исправить. В PyCharm имеется механизм под названием «намерения» (intentions), предназначенный для автоматизации исправления и улучшения вашего кода. Взгляните на рис. 4.18:

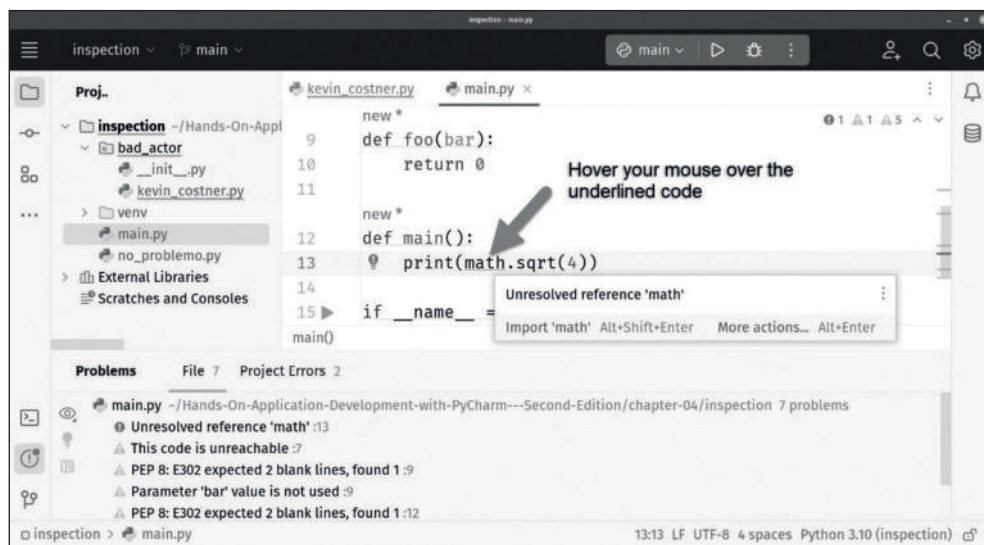


Рис. 4.18. Наведите указатель мыши на любой подчеркнутый код, чтобы узнать, почему он подчеркнут, а также возможные исправления

В показанном коде проблема в том, что я вызвал метод `sqrt()`, который находит квадратный корень. Этот метод является статическим методом класса `math`. Проблема в том, что мне не удалось импортировать этот класс. Описание проблемы появляется чуть ниже подчеркивания при наведении курсора. Ниже описания

проблемы указано наиболее вероятное решение. Нажатие **Alt + Shift + Enter** автоматически устранит проблему, добавив `import math` в верхнюю часть файла.

Если вы хотите опробовать это, можете использовать проект `inspections` в папке `Chapter-04` примера кода, который мы клонировали в главе 2.

Обратите внимание, что это может быть не единственное возможное решение. На рис. 4.18 мы также видим **More actions...**, предлагающее либо кликнуть ссылку, либо нажать **Alt + Enter**, чтобы увидеть больше возможностей.

По-настоящему проницательные читатели могли заметить лампочку. Это альтернативный вектор того же объекта. Посмотрите на рис. 4.19, чтобы увидеть лампочку в действии:

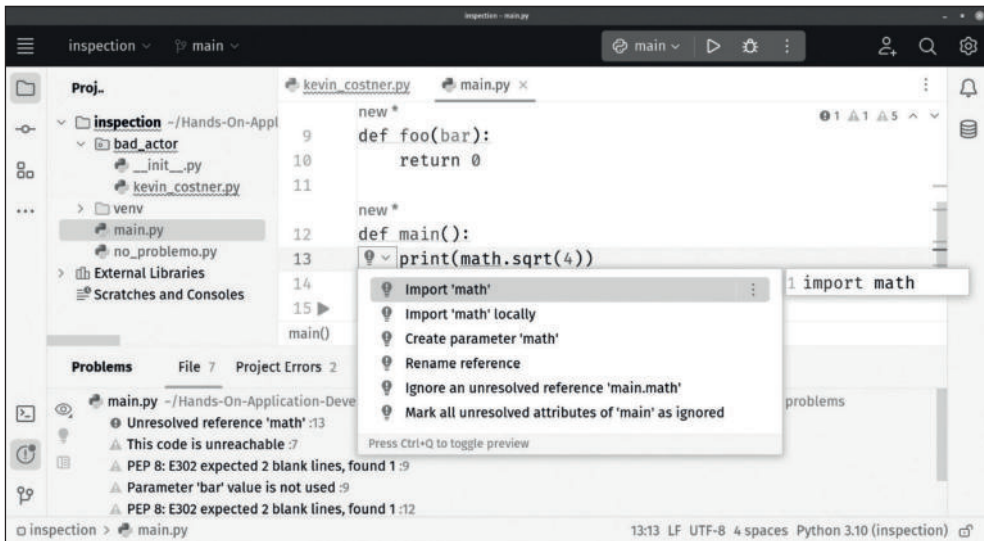


Рис. 4.19. Лампочка – еще один способ понять намерения

При нажатии на лампочку отображается список возможных намерений. На этот раз мы видим предварительный просмотр. Планируется добавить `import math` в первую строку файла.

Лампочки любят играть с вами в прятки – они исчезают, если вы сойдете со строки, на которой они изначально появились. Если вы хотите использовать лампочку, просто кликните в любом месте подчеркивания и дождитесь галочки. Она появится в начале строки, где кроется проблема.

В случае, который мы исследуем, проблема заключается в законной ошибке. Программа не запустится, пока мы не устраним проблему. На двух предыдущих рисунках этого не видно, но лампочка красная. Вы также увидите желтые лампочки, обозначающие менее серьезные проблемы.

Теперь, если вы похожи на меня, вы хотите видеть чистый файл с зеленой галочкой и без подчеркиваний. Позвольте мне сказать вам, что этого, вероятно, никогда не произойдет. PyCharm почти всегда найдет, что изменить. Иногда предлагаемые изменения не очень полезны. Вы можете прислушаться к предложению, которое каким-то незначительным образом изменит ваш код. Сразу после того, как вы это сделаете, лампочка возвращается, и PyCharm предлагает

изменить код обратно в прежний вид. Желтые лампочки вам не враги, если в них нет восклицательного знака.

РЕФАКТОРИНГ

Большинство хороших IDE и текстовых редакторов, ориентированных на разработку, имеют определенный уровень инструментов для рефакторинга. Рефакторинг – очень важная практика, которой часто пренебрегают. В моей книге «Практическая реализация шаблонов проектирования C#», доступной на Amazon (или там, где продаются лучшие технические книги), я рассказываю о некоторых энтропийных силах, которые приводят к краху хорошо задуманного проекта кодирования. Ваш код изначально чистый, и вся команда обязуется поддерживать нулевую техническую задолженность. Но это никогда не длится долго. Такие факторы, как нехватка времени, уровень квалификации разработчиков, неизбежные изменения, прозрачность и сложность, вызывают процесс вырождения. Ваш код превратится из хорошо построенного, идеально спроектированного шедевра в большой ком грязи на тарелке спагетти!

Я понимаю, что указываю здесь на книгу по C#, но, если у вас есть подписка на Packt, я настоятельно рекомендую вам прочитать первые две главы книги. В главах обсуждаются общие проблемы и способы предотвращения регресса вашего проекта. Единственное, что вы можете сделать, – это быть бдительными и никогда не игнорировать ценность рефакторинга вашего кода как регулярной части практики разработки.

Что такое рефакторинг?

Проще говоря, рефакторинг – это улучшение кода без изменения его функциональности. Если у вас есть модульные тесты (а они есть, ВЕРНО!?!?!), они должны пройти до и после рефакторинга без каких-либо изменений в самих тестах. Вы ищете способы оптимизировать свой код с точки зрения читабельности и производительности. Возможно, вы пропустили некоторые тонкости кода, например добавление **строк документации**, о которых мы поговорим позже в этой главе. Возможно, вы не добавили подсказки типов в сигнатуры методов. Возможно, есть возможности использовать шаблоны проектирования или принципы SOLID, чтобы сделать ваш код более гибким.

Идея рефакторинга заключается в том, что вы еще раз просматриваете свой код, желательно по прошествии некоторого времени. Вы когда-нибудь смотрели на код, который написали месяц или даже год назад, и задавались вопросом, что заставило вас напечатать это ужасное тело функции? Почему, черт побери, я сделал _____? (Заполните пропуск какой-нибудь глупостью, которую вы сморозили.) Вы не можете поверить, что это были вы. Вы умнее этого. Рассмотрите возможность проведения регулярных экспертных проверок вашего кода. Это может произойти в любой момент в процессе написания. Кроме того, если кто-то менее заинтересован в вашей работе, вы сможете обнаружить нереализованные возможности для улучшения. Почти всегда это потребует рефакторинга. Переверните ваши строки. Сколько раз вам передавали код, написанный другим разработчиком? Может быть, это был человек, который покинул компанию? Вы просматриваете его код и приходите к выводу, что тот,

кто писал его, ранее сбежал из местного дурдома. Его код необходимо полностью переписать. Вам самому придется стать рефакторингом чужого кода.

Инструменты рефакторинга в PyCharm

PyCharm имеет набор пунктов меню, предназначенных для рефакторинга. Некоторые из них явно не называются инструментами рефакторинга.

Очистка вашего кода

PyCharm имеет очень тщательный инструмент **очистки кода**. Этот инструмент, по сути, выполняет те же проверки, которые вы видели до сих пор, но делает их массово. Вы также можете попросить PyCharm попытаться исправить все проблемы. Эта функция полезна, когда вы импортируете проект, созданный за пределами PyCharm, скажем, с помощью инструмента, который не предлагает той помощи, которую вы видели в PyCharm.

Вы можете очистить открытый файл или все файлы в проекте. Честно говоря, я не рекомендую вам делать это на уровне проекта, поскольку вы не можете точно предсказать, что движок сделает с большим набором файлов, которые вы даже не открывали. На рис. 4.20 показано расположение меню **Code Cleanup...**:

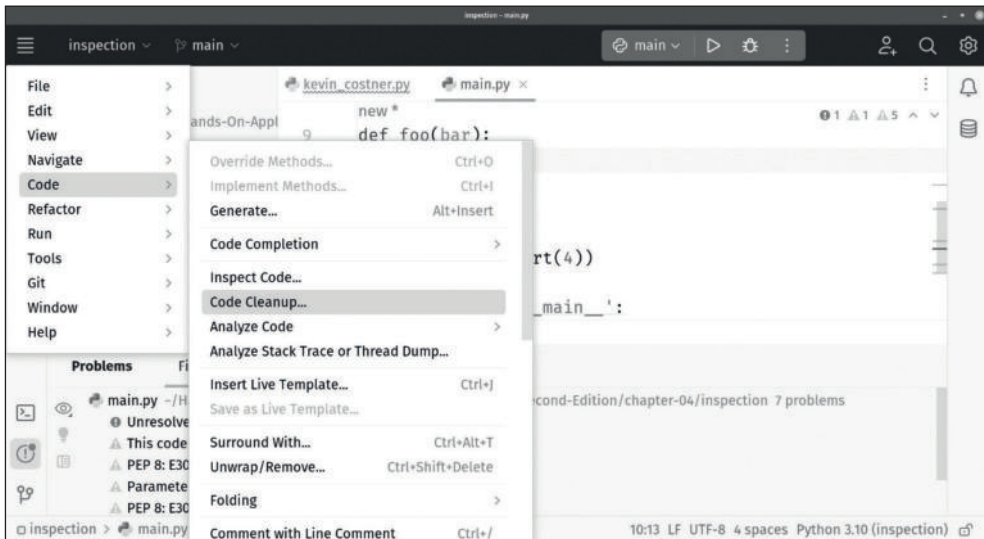


Рис. 4.20. Инструменты очистки кода можно найти в меню Code

Кликнув опцию меню, вы увидите диалоговое окно с вопросом об объеме очистки, как показано на рис. 4.21.

Вы можете очистить весь проект, незафиксированные файлы, текущий файл (в моем случае это `main.py` из проекта `inspection` в папке `Chapter-04` примера исходного кода) или пользовательскую область. Я рекомендую не пытаться вскипятить океан. Не делайте полную очистку большого проекта. Обычно разумнее позволить PyCharm творить чудеса с небольшими пакетами файлов. Параметр **Uncommitted files** – хороший шаг, который вы можете предпринять, прежде чем вносить изменения в свою систему контроля версий.

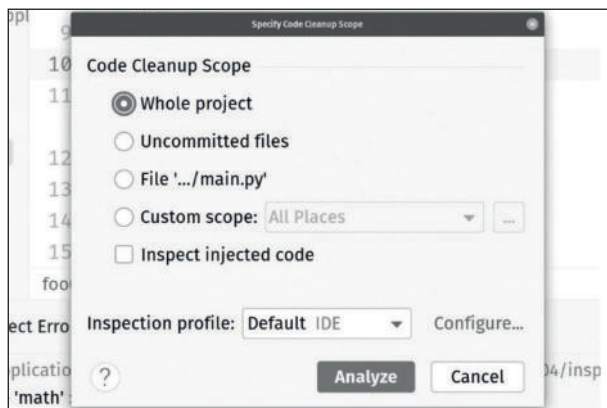


Рис. 4.21. Укажите объем очистки кода

Переименование

Дональд Кнут, один из самых уважаемых разработчиков программного обеспечения всех времен, написал в своей книге «Искусство компьютерного программирования», что в программировании есть две сложные вещи: присвоение имен вещам (например, переменным, функциям, классам, файлам и т. д.) и аннулирование ваших кешей. Как это верно! Это форма искусства – писать осмысленные определения переменных, которые являются самодокументируемыми и при этом ясно проясняют ваши намерения. Обычно требуется несколько попыток, чтобы сделать все правильно. В PyCharm есть инструмент, который позволяет легко изменить имя. Вы можете кликнуть правой кнопкой мыши все, что вы назвали, и выбрать **Rename**. Введите новое имя объекта, который необходимо переименовать. PyCharm сделает изменение везде, где используется это имя. Он даже найдет и изменит ссылки в комментариях и строках документации.

Если область изменения невелика, например вы переименовываете что-то, ограниченное локальной функцией, операция переименования происходит немедленно. Если вы пытаетесь переименовать более широкую область, например которая влечет за собой несколько файлов, PyCharm предварительно просмотрит изменение, показывая вам все файлы, которые будут затронуты. Можете проверить все изменения до их внесения, чтобы убедиться, что они подходят. После того, как вы будете удовлетворены, можете применить эти изменения.

Для файлов нет пункта меню Rename!

Это сбивает с толку, если вы используете другой инструмент, например Visual Studio Code, где для переименования файлов достаточно просто выбрать файл в представлении проводника и дважды кликнуть мышью или нажать *F2*. Вы будете искать опцию переименования файлов, но не найдете ее. Это потому, что он находится в меню **Refactor**. PyCharm считает переименование файла рефакторингом. В отличие от других инструментов PyCharm гарантирует, что переименование файла не повлияет на ваш код.

Встраивание переменных

PyCharm дает вам возможность автоматически встраивать переменные. Фактически это одно из наиболее распространенных предложений PyCharm. Рассмотрим этот код:

```
a = 5
b = 6

def add_two(num1: int, num2: int) -> int:
    sum = num1 + num2
    return sum

add_two(a, b)
```

Переменная `sum` на самом деле не нужна. Если вы встроите переменную, код станет следующим:

```
a = 5
b = 6

def add_two(num1: int, num2: int) -> int:
    return num1 + num2

add_two(a, b)
```

Мы удалили строку, в которой объявлялась переменная `sum`, и присвоили ей значение `num1 + num2`.

Методы извлечения

Ранее я упоминал концепцию под названием DRY, или «Не повторяй себя». Да, я понимаю, что, упоминая об этом еще раз, я нарушаю правило. Я делаю это с определенной целью. Помните, что IDE указывает на ошибки, а затем предоставляет советы и инструменты для устранения проблемы. Хочу представить очень полезную функцию. PyCharm предоставляет вам инструмент для легкого извлечения кода в отдельную функцию. Вам может понадобиться сделать это при нескольких обстоятельствах. Во-первых, если вы копируете и вставляете код в свой проект. Вероятно, вам нужно сделать этот код функцией, а затем вызывать ее из тех мест, куда вы вставляете скопированный код. Во-вторых, когда вы находите функцию, которая нарушает принцип единой ответственности. Если вы никогда об этом не слышали, то можете догадаться, что это значит. Хорошо написанная программа должна содержать функции, выполняющие только одно действие. Написанный код, содержащий функции или методы, выполняющие несколько функций, лучше разбить на отдельные функции.

Взгляните на простой пример такой возможности. Откройте файл `Character-04/not_dry.py` в примере кода. Код внутри поистине вопиющий! Чувствительным зрителям лучше сесть, прежде чем открыть файл. Вот! Это то, что каждый делал хотя бы один раз, прежде чем понял, что это плохая идея:

```
computer_science_grades = {
    "Guido van Rossum": 100,
    "Ewa Jodlwska": 99,
    "Fabrizio Romano": 88,
    "Henrich Kruger": 87,
    "Rick van Hattem": 83,
    "Steven Lott": 72,
    "Dusty Phillips": 72,
    "Quan Nguyen": 92
}
```

Хорошо, пока все в порядке. У нас есть словарь людей, которые посещали уроки информатики, а также их оценки. Кстати, это не случайные имена. Прочитав эту главу, посмотрите, сможете ли вы выяснить, кто эти выдающиеся личности. Я прошу прощения у некоторых выдающихся личностей за сами цифры. Предполагалось, что это будут более или менее случайные ключи, за исключением мистера ван Россума, который, очевидно, получил бы высшую оценку. Я уверен, что они все очень хорошо справились в реальной жизни. После этого у нас есть еще один набор оценок классов:

```
advanced_theoretical_and_applied_recess_grades = {
    "Bruce Van Horn": 100,
    "Prajakta Naik": 92,
    "Kinnari Chohan": 88,
    "Pooja Yadiv": 86
}
```

Хотя это другая тема и другая группа людей, идея та же. Теперь предположим, что нам нужно вычислить среднее значение для каждого класса. Я могу создать функцию для вычисления среднего значения по уроку информатики:

```
def computer_science_average(grades: dict) -> float:
    raw_total = 0
    for grade in grades.values():
        raw_total += grade

    average = (raw_total / len(grades))
    return average
```

Сигнатура нашего метода дает много полезных подсказок. У нас есть описательное имя функции. Функция принимает один параметр, и наша подсказка сообщает нам, что мы ожидаем словарь. Ожидается, что функция вернет число с плавающей запятой.

Тело функции создает переменную с именем `raw_total` и устанавливает для нее значение 0. Затем мы перебираем значения `dict` и на каждой итерации добавляем `value` в `raw_total`. Получив сумму, мы делим ее на количество ключей (`len`) в `dict` и – вуаля! У нас среднее по классу. В нижней части файла мы видим, где вызывается эта функция:

```
boring_class_average = computer_science_average(computer_science_grades)
print(f"Boring average is {boring_class_average}")
```

Замечательно! У нас есть вызов нашей функции `Computer_science_average` вместе с очень субъективным (и, вероятно, неточным, поскольку это мог быть ваш любимый класс) присвоением переменной. Так что же во всем этом плохого? Ничего. То, что происходит дальше, представляет собой проблему и возможность извлечь метод. Следующая функция вычисляет другой класс: расширенные теоретические и прикладные углубления. Это область, в которой я лично был пионером, и в этой области у меня нет соперников. К сожалению, поскольку я проводил больше времени на игровой площадке, совершенствуя свое искусство, и меньше времени на уроках информатики, я практически продублировал функцию, которую мы написали ранее:

```
def advanced_recess_average(grades: dict) -> float:
    raw_total = 0
    for grade in grades.values():
        raw_total += grade

    average = (raw_total / len(grades))
    return average
```

Это та же функция с другим именем. Нам нужно консолидироваться! Для этого вам нужно выделить все, что находится между двоеточием, завершающим сигнатуру метода, и оператором `return`. См. рис. 4.22. Не включайте оператор `return`, иначе PyCharm не сгенерирует оператор `return` в вашей извлеченной функции:

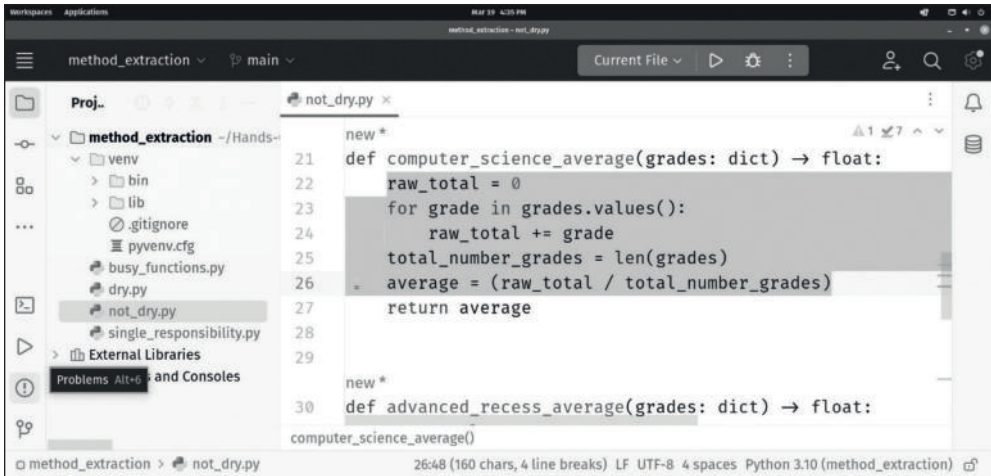


Рис. 4.22. Выберите код для извлечения в новую функцию или метод

Затем кликните правой кнопкой мыши выбранный код и выберите **Refactor | Extract Method**, как показано на рис. 4.23:

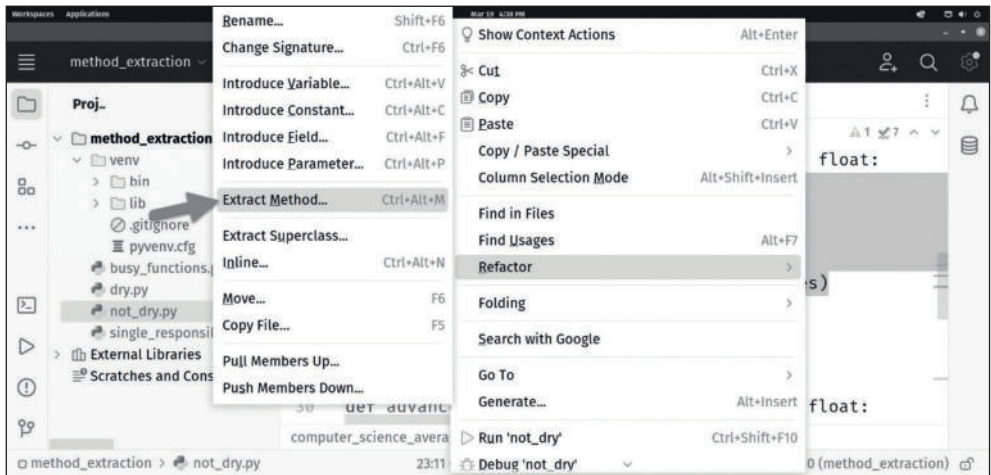


Рис. 4.23. Кликните правой кнопкой мыши выбранный код, затем выберите Refactor, затем Extract Method

При этом откроется пользовательский интерфейс, позволяющий определить новый метод, как показано на рис. 4.24:

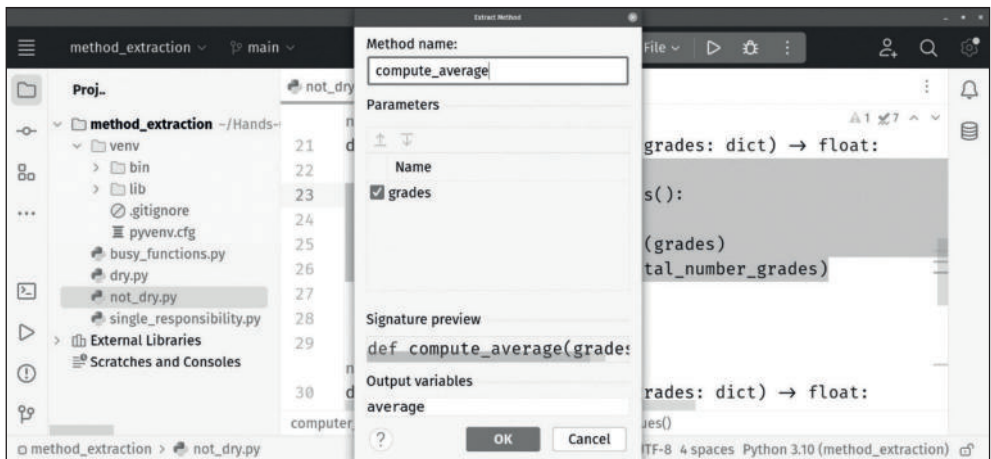


Рис. 4.24. Диалоговое окно Extract Method в PyCharm

Задайте имя метода извлечения. Я установил для себя значение `compute_average`. Остальное PyCharm заполнил автоматически. Нажмите **ОК**, и ваш код изменится. На рис. 4.25 показан результат моего рефакторинга:

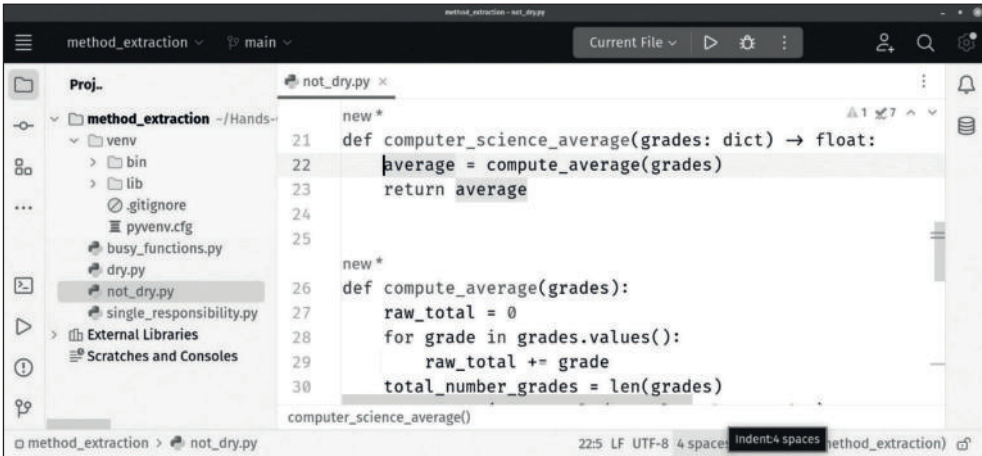


Рис. 4.25. Результат рефакторинга. Обратите внимание, что функция `Compute_average` была сгенерирована автоматически из выбранного кода

Давайте посмотрим на результирующий код PyCharm, сгенерированный в результате рефакторинга. Во-первых, `computer_science_average` изменилось на следующее:

```
def computer_science_average(grades: dict) -> float:
    average = compute_average(grades)
    return average
```

Эта функция теперь вызывает извлеченную функцию. Извлеченная функция выглядит следующим образом:

```
def compute_average(grades):
    raw_total = 0
    for grade in grades.values():
        raw_total += grade
    total_number_grades = len(grades)
    average = (raw_total / total_number_grades)
    return average
```

Это код, который мы выбрали для извлечения. PyCharm сгенерировал для меня функцию. На данный момент я должен вам сказать, что обычно я настороженно отношусь к сгенерированному коду. Он редко бывает идеальным. Здесь я бы предпочел подсказку о типе параметра `grades` и подсказку о типе возвращаемого значения. Однако это незначительные проблемы, и результат избавил меня от необходимости печатать.

Возможно, останется еще один нерешенный вопрос. Почему PyCharm не обнаружил и не пометил дублированный код? Быстрый ответ: наш пример кода слишком короткий. Если бы я добавил еще несколько строк к дублируемым функциям, они выглядели бы как дубликат. Давайте попробуем. Измените код обеих функций, чтобы он выглядел следующим образом:

```
def computer_science_average(grades: dict) -> float:
    raw_total = 0
    fake_var_1 = 1
    fake_var_2 = 2
    fake_var_3 = 3
    fake_var_4 = 4
    fake_var_5 = 5
    fake_var_6 = 6
    fake_var_7 = 7
    print(f"{fake_var_1}{fake_var_2}{fake_var_3}{fake_var_4}")
    print(f"{fake_var_5}{fake_var_6}{fake_var_7}")
    for grade in grades.values():
        raw_total += grade

    average = (raw_total / len(grades))
    return average
```

Все, что я сделал, – это добавил кучу поддельных объявлений переменных. Они не делают ничего важного, кроме как удлиняют дублирующийся фрагмент кода. По умолчанию PyCharm ищет только дублированные фрагменты длиной 10 строк и более. Короткие дубликаты не подходят. Я говорю об этом, потому что магия извлечения методов автоматически обрабатывает дубликаты. Давайте сделаем то же упражнение. Сначала посмотрите на PyCharm с внесенными изменениями. Вы должны увидеть некоторые индикаторы того, что у нас есть проблемы, как показано на рис. 4.26:

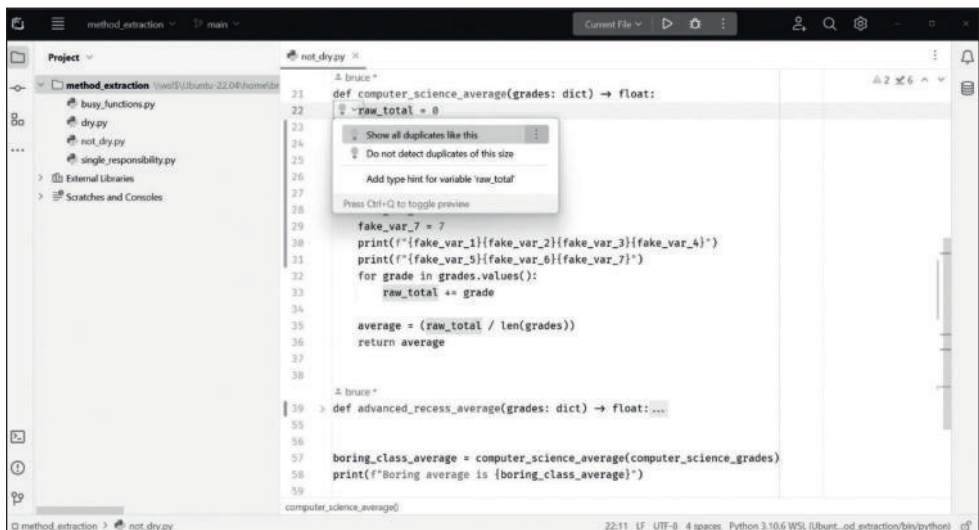


Рис. 4.26. Теперь, когда наш дублированный код стал длиннее, он обнаруживается и помечается

Мы видим, что PyCharm заметил наш дублированный код. Выделите код для извлечения, как показано на рис. 4.27:

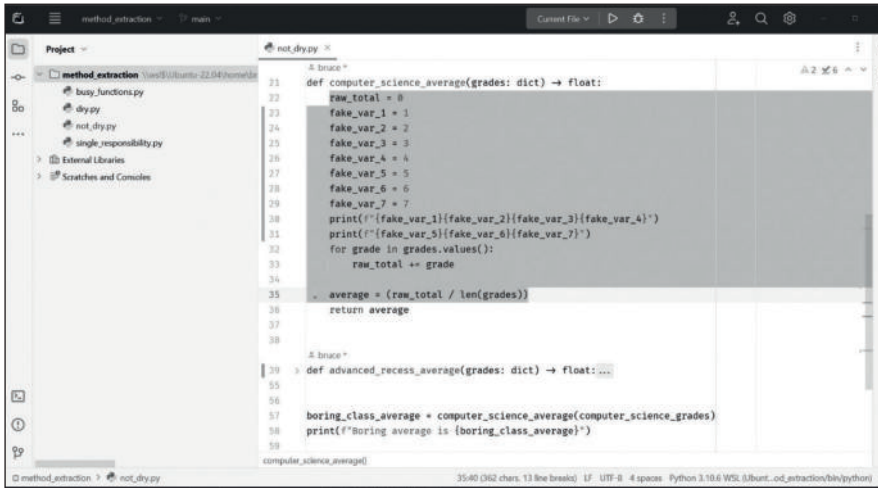


Рис. 4.27. Отметьте код для извлечения

Кликните правой кнопкой мыши выделенный код и выберите **Refactor | Extract Method**, как показано на рис. 4.28:

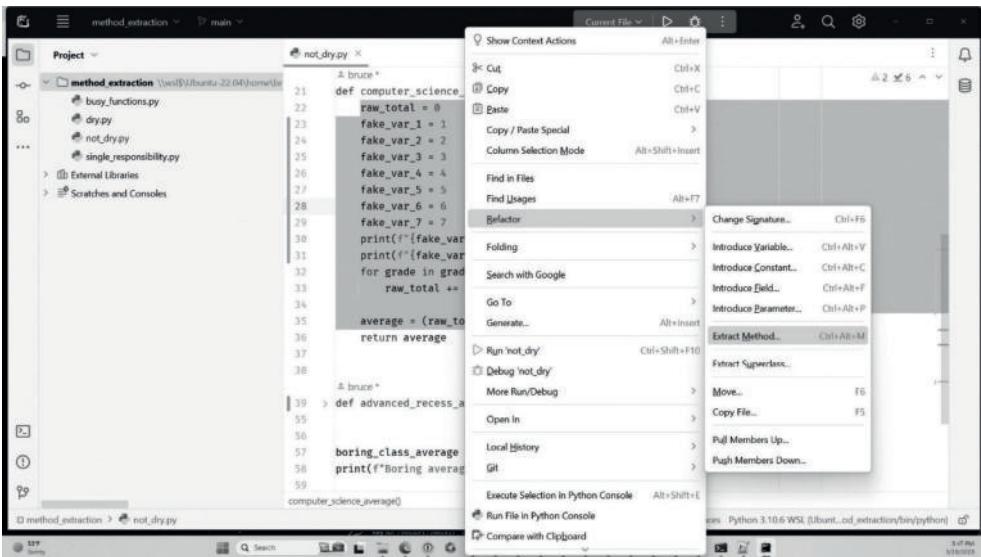


Рис. 4.28. Извлеките метод, кликнув правой кнопкой мыши, выбрав Refactor, затем Extract Method

Назовите извлеченную функцию `Compute_average`, как показано на рис. 4.29:

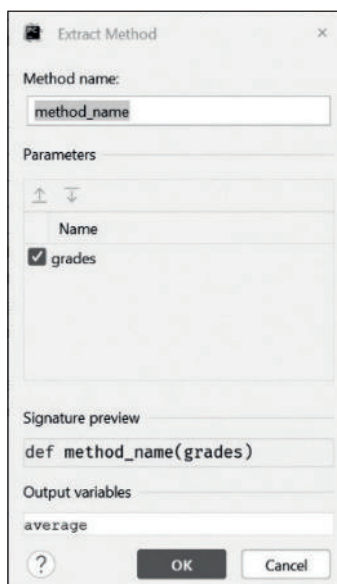


Рис. 4.29. Назовите извлеченную функцию `Compute_average`

Нажмите **OK**. На этот раз все немного по-другому. Вы обнаружите, что PyCharm создает извлечение функции, как и раньше, но на этот раз вам также будет предложено заменить дублирующий код, как показано на рис. 4.30:

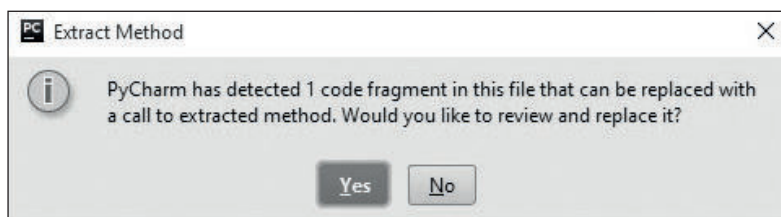


Рис. 4.30. PyCharm спрашивает, хотите ли вы заменить дублированный фрагмент ссылкой на извлеченную функцию

Экспорт функции в другой файл

Как часто вы писали изящную функцию полезности не в том месте? Возможно, вы поместили ее в модуль или класс, предназначенный для выполнения чего-то конкретного, но оказывается, что ваша служебная функция обычно используется во многих местах. Принцип единой ответственности, применимый к функциям, также применим к модулям и классам. Отличным примером является функция, которая подключается к базе данных. Допустим, вы только что устроились на работу на конфетную фабрику Билли Бланки. Им нужно, чтобы вы написали несколько скриптов, которые будут импортировать составленные ими списки конфет из нескольких различных текстовых форматов и сохранять их в базе данных. Первое требование: вам нужно прочитать простой текстовый файл и записать его в базу данных SQLite.

Откройте проект в `Chapter-04/move_function/read_input_file_a.py`. Давайте рассмотрим содержимое:

```
import sqlite3
CANDY_DB = "candy.db"
```

Эти первые две строки импортируют библиотеку `sqlite3` из стандартной библиотеки. Если вы раньше не работали с `sqlite3` – вот все, что вам нужно знать на данный момент: это реляционная база данных на основе файлов. Под этим мы подразумеваем, что вам не нужно устанавливать сервер, как при использовании таких баз данных, как `Postgres` или `MariaDB`. Это делает ее хорошей базой данных для обучения и работы с прототипами. Мы подробно рассмотрим базы данных в главе 11. Давайте продолжим определение функции, которая откроет файл, прочитает его содержимое и вставит его в базу данных:

```
def read_input_file_type_a(file_path: str) -> None:
    with open(file_path, "r") as data:
        for line in data:
            cleaned = line.strip("\n")
            write_to_database(cleaned)
            print("Processing Complete!")
```

Мы открыли файл. Для каждой строки в файле мы читаем ее как текст и удаляем символ новой строки. Это необходимо для правильной работы вставки базы данных. После очистки строки мы вызываем функцию, которая записывает в базу данных:

```
def write_to_database(datum: str) -> None:
    connection = sqlite3.connect(CANDY_DB)
    cursor = connection.cursor()
    sql = f"INSERT INTO candy(name) VALUES ('{datum}')"
    print(sql)
    cursor.execute(sql)
    cursor.close()
    connection.close()
```

Я включил файл базы данных в репозиторий кода, поэтому для создания базы данных код не требуется. Эта функция просто открывает базу данных, а затем создает курсор. Курсор используется для выполнения команд базы данных с использованием SQL. Даже если вы не знаете SQL, я уверен, вы сможете разобраться, что происходит. В базе данных есть таблица под названием `candy`. В таблице есть только одно поле: `name`. Мы делаем это очень просто. Я не стал создавать первичный ключ, потому что на данный момент база данных не имеет особого значения. Мы должны сосредоточиться на функции больше, чем на том, как она работает.

Сгенерировав оператор SQL для вставки названия `candy` из названия `candy` в текущую строку текстового файла, я выполняю оператор SQL, который вставит одну строку в таблицу `candy`.

Общее правило программирования: все, что вы создаете, следует уничтожить, а все, что вы открываете, – закрывать. Итак, я закрываю курсор и соеди-

нение с базой данных, чтобы избежать блокировки ресурсов в будущем. Наконец, я использую общепринятое соглашение об именах dunder для запуска файла для тестирования:

```
if __name__ == "__main__":
    read_input_file_type_a("../input_file_a.txt")
```

В PyCharm я могу выполнить это, установив для конфигурации запуска значение **Current File** и нажав кнопку **Run**. Это работает! Файл читается, ошибок нет.

Двойное подчеркивание

Двойное подчеркивание (dunderscore) относится к элементу в коде Python, которому предшествуют и продолжают два символа подчеркивания. Это решает проблему произношения при разговоре о коде. Как бы вы рассказали о методе `__init__`? Вы могли бы сказать «*подчеркивание подчеркивание init подчеркивание подчеркивание*». Но это обременительно. Если вы просто скажете «init», вы не будете достаточно конкретны, поскольку может быть другая функция или метод с именем `init` без подчеркиваний. Итак, вы говорите «*dunderscore init*» или даже просто «*dunder init*», и все понимают, о чем.

На следующий день мы приходим на работу и узнаем о новом требовании. Нам нужен еще один скрипт, который считывает данные из файла **JavaScript Object Notation (JSON)**. Файл JSON просто содержит такой массив:

```
{
  "data": [
    "truffles",
    "turtles",
    "dark chocolate bark"
  ]
}
```

Это по-прежнему просто список конфет, но нам нужно обработать его по-другому. Откройте `Chapter-04/move_function/input/read_input_file_b.py`. Вы обнаружите, что его код похож на другой код:

```
import json
from read_input_file_a import write_to_database
```

Мы знаем, что нам нужно работать с JSON, поэтому я импортировал пакет `json`. Я также знаю, что мне нужно будет писать в ту же базу данных, что и раньше. Я знаю, что повторное использование кода – это хорошо, поэтому я импортирую функцию из другого скрипта. Затем я приступил к созданию версии кода, читающей файлы JSON:

```
def read_input_file_type_b(file_path: str) -> None:
    with open(file_path, "r") as json_data:
        data = json.load(json_data)
        candies = data["data"]
        for candy in candies:
            write_to_database(candy)

    print("Processing Complete!")

if __name__ == "__main__":
    read_input_file_type_b("../input_file_b.json")
```

Метод `json.load` принимает необработанный текст и преобразует его в обычный словарь Python 3 `dict`. Как вы можете видеть из предыдущего листинга файлов, в `dict` будет одна вещь: массив конфет с ключом данных. Итак, я беру это и помещаю в переменную `candies`, затем перебираю этот массив и вызываю функцию `write_to_database` для каждой `candies` в массиве. Ух ты! Еще даже не время обеда! Может, мне прогуляться по фабрике? Я слышал, какая-то маленькая девочка тестирует жвачку, которая превращает людей в малину.

Не так быстро! Этот код можно улучшить. Наши скрипты ввода действительно предназначены для чтения данных из текста. Нет смысла содержать в одном из скриптов функцию базы данных, поскольку она просто не принадлежит скрипту, считывающему текстовые входные файлы. Это действительно должно быть в отдельной упаковке. Давайте извлечем ее в отдельный файл.

Откройте `Chapter-04/move_function/input/read_input_file_a.py`. Кликните правой кнопкой мыши имя функции, которую мы собираемся переместить, как показано на рис. 4.31:

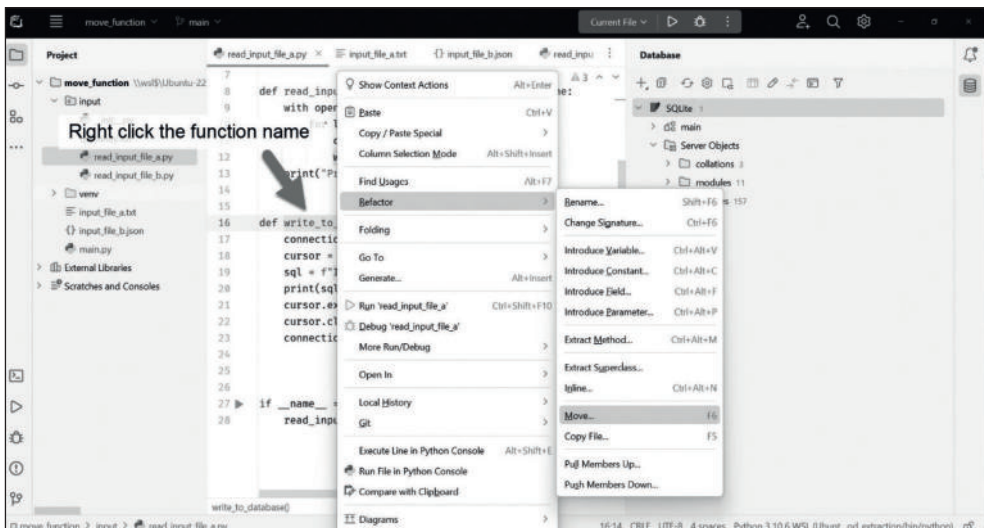


Рис. 4.31. Кликните правой кнопкой мыши функцию, которую хотите переместить, затем нажмите Refactor, затем Move

В контекстном меню выберите **Refactor | Move**. Вы увидите диалоговое окно с запросом имени целевого файла. Я ввел имя файла `data_helper.py`. Нажмите **ОК**. Смотрите внимательно, потому что многое должно произойти.

Я получаю новый файл с именем `data_helper.py`:

```
import sqlite3

from input.read_input_file_a import CANDY_DB
def write_to_database(datum: str) -> None:
    connection = sqlite3.connect(CANDY_DB)
    cursor = connection.cursor()
    sql = f"INSERT INTO candy(name) VALUES ('{datum}')"
    print(sql)
    cursor.execute(sql)
    cursor.close()
    connection.close()
```

Это извлеченная функция, перемещенная в отдельный файл. PyCharm обнаружил соответствующий оператор импорта и также переместил его сюда. У меня была константа для имени файла `CANDY_DB`. Это не сдвинуло с места, что было бы моим предпочтением. К сожалению, текущая версия PyCharm не является экстрасенсорной. Мне просто придется переместить это самому. В остальном этот файл выглядит идеально.

Если вы проверите содержимое `input_file_a.py`, вы увидите, что оно изменилось. Теперь первая строка файла выглядит следующим образом:

```
from input.database_helper import write_to_database
```

Аналогично, если я открою `input_file_b.py`, я также найду там входные данные. PyCharm извлек функцию в собственный файл, а затем изменил каждую ссылку в каждом файле, чтобы она указывала на новую локацию.

Теперь я понимаю, что мне, вероятно, следовало бы поместить это в отдельный модуль, и я понимаю, что вы, вероятно, сделали бы это совершенно по-другому. Прежде чем вы начнете критиковать меня в Твиттере, помните, что эта книга посвящена PyCharm, а не архитектуре программного обеспечения. Я намеренно пытаюсь дать все просто.

Документация

Ни один программист не усомнится в важности документации в разработке программного обеспечения. Тем не менее процесс создания документации для программы может быть довольно утомительным. Более того, конечный результат может оказаться малоэффективным, если человек, создающий документацию, не будет следовать стандартным практикам.

Помня об этом, PyCharm стремится упростить процесс документирования и сделать его максимально простым и гладким. Что касается документации, то в этом процессе мы рассмотрим два компонента: просмотр и создание документации. Мы узнаем, что PyCharm предлагает отличную поддержку обоих процессов.

Работа со строками документации

Документация в Python известна как **докстринги (docstrings)**, определяемые как строковый **литерал**, который помещается перед любым оператором в модуле, функции, классе или методе Python. Вы можете просмотреть примеры строк документации Python, заглянув в исходный код различных встроенных функций Python. Также рекомендуется, чтобы любой написанный вами пользовательский API имел соответствующие строки документации для удобства чтения и обслуживания.

Наиболее примечательной тонкостью при создании строк документации является практика использования тройных кавычек для заключения строки документации (примеры этого мы увидим в следующем подразделе). Более подробную информацию о соглашениях по строкам документации можно найти в этой статье PEP: www.python.org/dev/peps/pep-0257/.

Создание документации

В этом подразделе мы рассмотрим процесс написания строки документации для функций с помощью PyCharm. Давайте начнем.

Откройте проект в примере исходного кода в chapter-04/documentation. Откройте файл `prime_test.py`, который выглядит следующим образом:

```
import sys
from math import sqrt

def prime_check(n: int) -> bool:
    # TODO: docstring goes here

    if n < 2:
        return False

    limit = int(sqrt(n)) + 1
    for i in range(2, limit):
        if n % i == 0:
            return False # return False if a divisor is found

    return True # return True if no divisor is found
```

После этого файл продолжается, но мне нужно, чтобы вы сосредоточили внимание на строке прямо под сигнатурой метода функции `prime_check`. Там есть `TODO`. Если вы из испаноязычной страны, поймите, что это еще не *все*. Это относится к элементу «to-do» в коде. В данном случае первоначальный разработчик, которым оказался Куан Нгуен, автор первого издания этой книги, заявляет, что он не писал строку документации для этой функции. Здесь он анализирует, что знает это и намерен вернуться и исправить это позже. Давайте поможем ему с помощью небольшого волшебства PyCharm. Прежде чем вы будете слишком взволнованы, я с грустью сообщу, что в PyCharm нет инструмента, который мог бы прочитать ваш код и сгенерировать строку документации. Учитывая, как сильно разработчики ненавидят писать документацию, я готов поспорить, что где-то предпринимаются быстрые и яростные усилия, чтобы

заставить AI делать это. Но мы придерживаемся того, что позволяет нам делать PyCharm «из коробки».

Удалите строку `# TODO` и вместо нее введите три двойных кавычки (`"""`) и нажмите *Enter*. Вы увидите сгенерированный шаблон строки документации:

```
"""

:param n:
:return:
"""
```

Этот шаблон требует некоторого заполнения, чтобы стать полноценной строкой документации. Обратите внимание на пробел под первым набором тройных кавычек. Здесь вы должны написать о том, что делает функция. Может быть, что-то вроде этого:

```
"""
Check whether an integer is a prime number of not.
Generally, the function goes through all odd numbers
less than the square root of the input integer, and
checks to see if the input is divisible by that number.

:param n:
:return:
"""
```

Ниже находится раздел параметров, ожидаемых функцией. Здесь функция принимает один аргумент с именем `n`. Нам следует немного написать об этом параметре, включая его тип:

```
"""
Check whether an integer is a prime number of not.
Generally, the function goes through all odd numbers
less than the square root of the input integer, and
checks to see if the input is divisible by that number.

:param n: the integer to prime check
:return:
"""
```

Последняя часть – это документация по возвращаемому значению:

```
"""
Check whether an integer is a prime number of not.
Generally, the function goes through all odd numbers
less than the square root of the input integer, and
checks to see if the input is divisible by that number.

:param n: the integer to prime check
:return: boolean
"""
```

Рассмотрим сгенерированный шаблон строки документации после того, как мы нажмем *Return/Enter*, чтобы раскрыть пару тройных двойных кавычек. Строки `:param` и `:return:` являются частью шаблона и будут включаться каждый раз, когда мы таким же образом расширяем строку документации. PyCharm позволяет нам изменять этот формат шаблонов строк документации, делая его легко настраиваемым и удобным.

Настройка шаблонов документации

Как обычно, шаблоны строк документации легко настраиваются. Вы найдете параметры настройки, перейдя в окно **Settings**, подробно описанное в главе 3. Просто выполните поиск в строке документации, и вы найдете области, которые должны привлечь ваше внимание.

Первый показан на рис. 4.32:

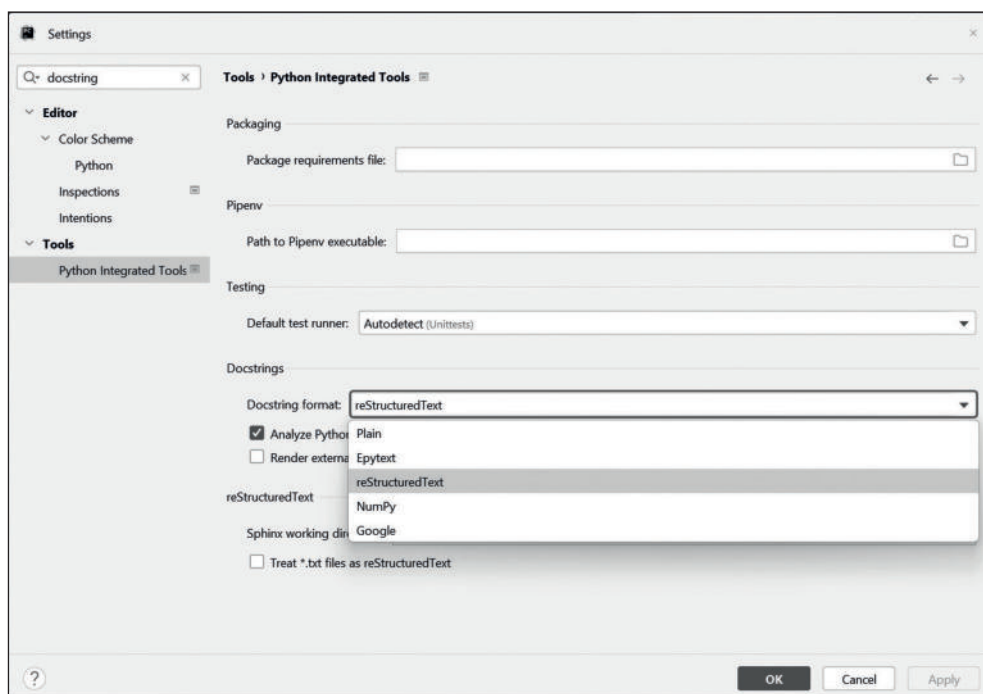


Рис. 4.32. Вы можете изменить общий формат, используемый для отображения строк документации, на один из нескольких стандартных отраслевых форматов

Другая группа настроек является частью настроек цветовой схемы, которые позволяют вам настроить цвет, используемый для отображения строки документации в редакторе PyCharm.

Просмотр документации

Представьте себе ситуацию, когда вы используете определенный метод из одного пакета, но не совсем уверены, какие параметры принимает этот метод или каков тип возвращаемого значения. Поэтому вам необходимо выйти в интернет и просмотреть документацию пакета для этого конкретного метода.

Как пользователь PyCharm, вы можете добиться того же с помощью двух простых действий: **Quick Definition** и **Quick Documentation**. Продолжая использовать скрипт `prime_check.py` из предыдущего раздела, переместите курсор на строку, где мы используем функцию `math.sqrt()` в функции `prime_check()`; это должно быть около строки 19.

Quick Documentation

Допустим, мы хотели бы увидеть документацию по этой функции. Вы можете просто навести указатель мыши на вызов функции и немного подождать. Альтернативно вы можете выбрать **View | Quick Documentation** для этого или соответствующего сочетания клавиш. Вы увидите всплывающее окно с документацией, похожее на рис. 4.33:

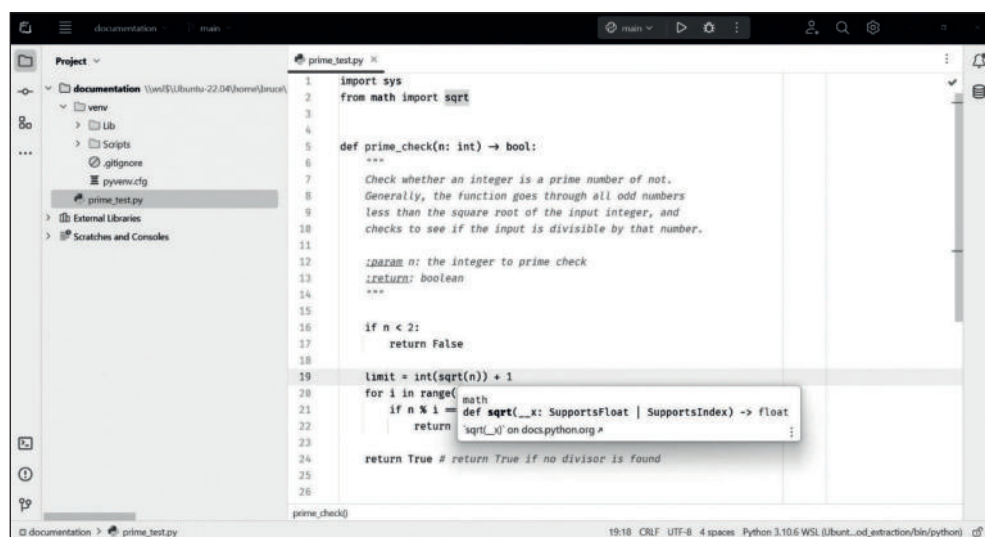


Рис. 4.33. Quick Documentation показывает документацию для выбранной функции

Более того, вы также можете просмотреть документацию для своих собственных функций, методов, классов и т. д., используя то же действие. Процесс индексирования PyCharm находит и генерирует эту информацию при открытии проекта.

Если вы переместите курсор на вызов `prime_check()` в основной области видимости в следующей строке (которая должна быть около строки 38):

```
if prime_check(num):
```

Подождав немного, вы сможете увидеть ту же строку документации, которую мы ввели ранее, как показано на рис. 4.34.

Обратите внимание, что формат строки документации в документации соответствует документации, показанной в окне.

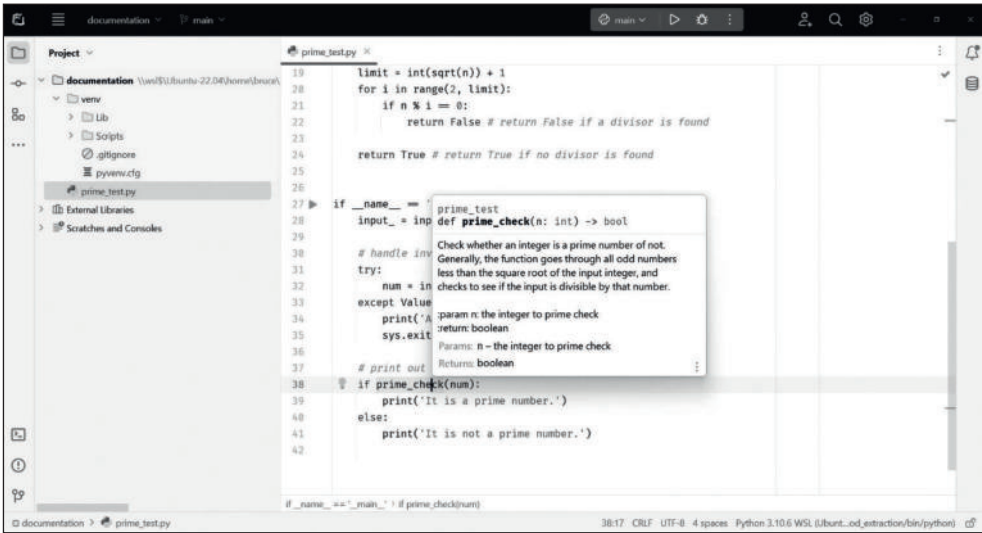


Рис. 4.34. Quick Documentation показывает документацию для выбранной функции в строке 38

Quick Definition

Quick Definition работает так же, как **Quick Documentation**. Это полезно, когда документация не предоставляет достаточно информации, и вы хотите увидеть, как определенная функция определена в исходном коде. Для этого поместите курсор на определенный вызов API и выберите **View | Quick Definition**, чтобы вызвать действие.

Например, на рис. 4.35 показано быстрое определение, вызываемое вызовом `prime_check()` в нашем примере:

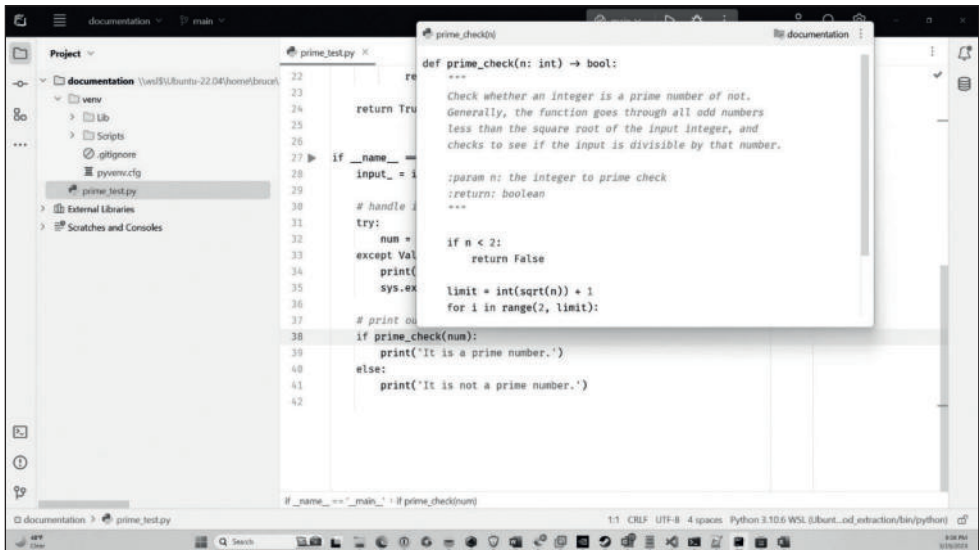


Рис. 4.35. Quick Definition показывает фактическое определение кода для функции, которое, естественно, включает строку документации, если она присутствует

В целом PyCharm предоставляет мощные возможности для динамического просмотра документации и определений в среде IDE. Можно сэкономить значительное время и энергию, если программистам не придется переключаться из среды разработки для поиска документации.

КРАТКОЕ СОДЕРЖАНИЕ

В этой главе мы рассмотрели возможности PyCharm, касающиеся различных аспектов программирования, включая анализ кода, завершение кода, рефакторинг и документирование. Во всех этих процессах интеллектуальный анализатор кода PyCharm предоставляет интеллектуальные и удобные возможности для редактирования и устранения проблем в вашем коде в реальном времени и в динамическом режиме.

Помимо большого количества опций, которые может поддерживать интеллектуальный анализатор кода, PyCharm также позволяет пользователям настраивать поведение анализатора по своему вкусу. Этого можно добиться в различных разделах общих настроек. В целом эти функции поддержки направлены на повышение вашей производительности как разработчика, причем таким образом, чтобы это было индивидуально и выгодно для вас.

В следующей главе мы сосредоточимся на конкретном аспекте программирования: контроле версий. Мы узнаем об особенностях процесса контроля версий с помощью Git и о том, как PyCharm поддерживает и оптимизирует этот процесс.

Вопросы

1. Какие уровни серьезности проблем в программе Python определяются анализатором кода PyCharm?
2. Какие распространенные проблемы PyCharm может обнаружить и помочь устранить с помощью своего интеллектуального анализатора кода?
3. Чем поддержка завершения кода PyCharm отличается от других?
4. Какие распространенные варианты завершения кода предлагает PyCharm?
5. Каковы распространенные причины, по которым поддержка завершения кода PyCharm не работает?
6. Какие распространенные варианты рефакторинга предлагает PyCharm?

Дальнейшее чтение

- Jolt Awards: *The Best Books For Developers* (informationweek.com).
- Домашняя страница Pylint: <https://www.pylint.org/>.
- Законы Мерфи о технологиях: <https://www.netlingo.com/word/murphys-laws-on-technology.php>.

Контроль версий в PyCharm с помощью Git

Контроль версий – это важная передовая практика в области разработки программного обеспечения. Однако механика этого процесса пугает новых разработчиков. Легко совершать ошибки. Будучи инструктором учебного курса в Южном методистском университете (Пони, вперед!)¹, я видел довольно много грубых ошибок, таких как случайное добавление всей вашей домашней папки в репозиторий Git, создание репозитория внутри других репозиториях и уничтожение результатов тяжелой работы различными правками. Я предпочитаю, чтобы мои ученики осваивали **Git** в командной строке. Фактически это самый первый навык, которому я обучаю, и, на мой взгляд, он один из самых сложных.

После того как разработчик обретает уверенность во всем процессе Git, ему приходится постоянно выходить из IDE, чтобы выполнить четыре или пять команд, а затем снова возвращаться в IDE, чтобы продолжить работу. Это правда, что вы можете использовать встроенное окно терминала, доступное в PyCharm, но IDE предлагает лучший вариант: встроенный графический интерфейс управления версиями.

PyCharm поддерживает ряд основных **система контроля версий (VCS)**, включая следующие:

- Git,
- Mercurial,
- Subversion,
- Perforce,
- Microsoft **Team Foundation Server**.

Несмотря на то что существует множество других систем контроля версий, они, пожалуй, самые популярные, и среди тех, что в этом списке, Git стал фактическим стандартом в отрасли.

¹ Это – девиз университета. – *Прим. ред.*

В этой главе сначала будет рассмотрена некоторая базовая информация о контроле версий и системах контроля версий на тот случай, если вы новичок в этой теме. После этого введения мы полностью сосредоточимся на инструментах для Git, поскольку, как я только что намекал, он лидирует с точки зрения доли рынка.

К концу главы вы изучите следующее:

- преимущества использования VCS,
- работу с Git для выполнения операций добавления, фиксации, отправки, слияния и ветвления,
- создание и поддержку файла .gitignore,
- работу с инструментами Git в IDE, на которые можно ссылаться из нескольких векторов.

ТЕХНИЧЕСКИЕ ТРЕБОВАНИЯ

Чтобы успешно изучить эту главу, вам понадобится следующее:

- рабочая установка Python 3.10 или новее;
- рабочая установка PyCharm;
- клиентское программное обеспечение Git для вашего компьютера. В системах Mac и Linux это обычно устанавливается в стандартной комплектации. Пользователи Windows могут пройти на <https://gitforwindows.org>, чтобы загрузить бесплатное программное обеспечение;
- бесплатная учетная запись GitHub. Зарегистрируйтесь на <https://github.com>. Обратите внимание: хотя на GitHub есть платные функции, мы не будем их использовать.

КОНТРОЛЬ ВЕРСИЙ И ОСНОВЫ GIT

Икона индустрии программного обеспечения **Джоэл Спольски**, создатель Stack Overflow, еще в 2000 году написал знаменитый блог. Среди его многочисленных влиятельных постов была статья под названием «Тест Джоэла: 12 шагов к лучшему коду». Публикация была создана для того, чтобы дать разработчикам программного обеспечения возможность легко оценить уровень зрелости любой организации, занимающейся разработкой программного обеспечения. Он задал 12 вопросов, по одному баллу за каждый вопрос. Хорошая группа разработчиков программного обеспечения должна иметь оценку не менее 11. Самый первый пункт в списке посвящен этой главе: «Используете ли вы систему контроля версий?» Если вам интересно остальное, в разделе «Дальнейшее чтение» этой главы есть ссылка на сообщение в блоге, а также ссылка на книгу г-на Спольски «Джоэл о программном обеспечении».

Джоэл назвал это **контролем источников (source control)**. Я называю это **контролем версий (version control)**. Термины взаимозаменяемы. Я назову это *контролем версий*, чтобы соответствовать пользовательскому интерфейсу в PyCharm, относящемуся к группе функций, о которых мы говорим как о **системе контроля версий (VCS)**. Строго говоря, контроль версий – это процесс, используемый для отслеживания изменений в файлах, составляющих вашу программу, с течением времени. Цели VCS включают в себя следующее:

- возможность вернуться к любой ранее сохраненной версии любого файла вашего проекта;
- возможность автоматически объединять работу нескольких разработчиков в одной команде, если их работа не конфликтует. Конфликт возникает, когда два разработчика изменяют один и тот же файл в одном и том же месте;
- предоставить простые средства для рассмотрения конфликтов и разрешения их посредством сотрудничества;
- возможность отслеживать, какие разработчики внесли каждое изменение в код с течением времени;
- обеспечить систему ветвления, позволяющую исправлять ошибки, улучшать и экспериментировать, не жертвуя стабильностью вашего производственного кода.

Таким образом, VCS – это программная система, предназначенная для реализации этого процесса. Системы контроля версий бывают двух разновидностей в зависимости от того, как хранятся версии. **Централизованные системы контроля версий** работают примерно так же, как ваша местная публичная библиотека. Ваш проект зарегистрирован в **репозитории**. Затем разработчики могут проверить код, чтобы поработать над созданием новой версии. При этом будет получена последняя версия проекта. Некоторые системы, такие как **Perforce**, традиционно требуют, чтобы вы явно извлекали файлы, с которыми хотите работать, точно так же, как если бы вы извлекали книгу из библиотеки. Пока вы извлекли эти файлы, никому другому не разрешается изменять их, пока вы не вернете их обратно со своими изменениями. Другие системы, такие как **Subversion**, не имеют этого требования. Любой может работать над своей локальной рабочей копией. Когда работа завершена, разработчик **передает** свою работу в центральный репозиторий. В процессе фиксации VCS проверяет наличие **конфликтов**. Если конфликтов нет, входящая работа **объединяется**, а новая версия сохраняется на сервере VCS, которая затем становится доступной другим разработчикам для возможного обновления. Если обнаружены конфликты, фиксация отклоняется, и разработчик должен работать с другими разработчиками в команде, чтобы разрешить конфликт, прежде чем работа может быть объединена и окончательно зафиксирована. Во всех централизованных системах каждый разработчик в команде имеет на своем локальном компьютере только последнюю версию кода.

Напротив, **распределенные системы контроля версий (DVCS)**, такие как и **Mercurial**, сохраняют все изменения, когда-либо выполненные в проекте, на компьютере каждого разработчика. Преимущество здесь в том, что нет единой точки отказа, которая могла бы привести к полной потере кода проекта. В централизованной системе, если сервер взломан, история изменений проекта может быть потеряна. В распределенной системе нет центральной системы для хранения истории изменений проекта. У каждого разработчика на своем компьютере хранится вся история проекта, поэтому потеря любого компьютера не представляет собой серьезной проблемы, если не считать обычной боли, связанной с заменой оборудования.

При этом распределенные системы используют центральный **концентратор (hub)** или **удаленный компьютер (remote)**, чтобы обеспечить легкую

синхронизацию между рабочими копиями на компьютере каждого разработчика. Хаб-сервер предоставляет дополнительные функции, такие как управление тем, кому разрешено просматривать или изменять код, а также функции совместной работы, такие как обзоры кода, отслеживание ошибок, документирование и обсуждение. DVCS также представляет собой простой способ контролировать публикацию новых выпусков веб-приложений. Многие системы непрерывного развертывания, такие как **Travis CI**, **CircleCI** и **Beanstalk**, просто используют функции DVCS для контроля процесса управления релизами.

Самая известная DVCS – Git, созданная **Линусом Торвальдсом**, который также создал операционную систему **Linux**. Самым известным хабом DVCS является, как вы уже догадались, GitHub, принадлежащий Microsoft. Многие люди используют термины «Git» и «GitHub» как взаимозаменяемые. Это ошибка. GitHub – это место в интернете. Программное обеспечение, используемое для доступа к *GitHub*, называется *Git*. Многие проекты, использующие Git в качестве DVCS, размещают свой центр репозитория в сервисах, отличных от GitHub, таких как **GitLab**, **Microsoft Azure DevOps**, **Atlassian Bitbucket** и **Beanstalk** (<https://beanstalkapp.com>, не путать с сервисом Beanstalk на **Amazon Web Services**, не имеющим никакого отношения к контролю версий).

Когда разработчики вносят изменения в свою рабочую копию, они могут фиксировать изменения в своей локальной копии. Позже они смогут синхронизировать свой код, извлекая изменения из центрального узла, разрешая любые конфликты, а затем распространяя свою объединенную работу. Опять же, большая разница заключается в DVCS; все происходит на компьютере разработчика, а не на сервере с централизованной системой.

НАСТРОЙКА GIT НА ВАШЕМ КОМПЬЮТЕРЕ

Независимо от того, установили вы Git для Windows на свой компьютер с Windows или работаете с предустановленным клиентом Git на своем компьютере Mac или Linux, вам необходимо выполнить некоторые дополнительные настройки, помимо простой установки программного обеспечения. Предстоящие задачи по настройке включают в себя следующее:

- 1) установите имя пользователя и адрес электронной почты по умолчанию;
- 2) создайте ключ **Secure Shell (SSH)**, чтобы вы могли безопасно взаимодействовать с удаленными концентраторами, такими как GitHub;
- 3) добавьте ключ SSH в свою учетную запись на GitHub.

Давайте рассмотрим процесс для каждой задачи.

Для начала вам необходимо запустить терминал. Если вы используете Mac или Linux, вы сможете найти в своей системе приложение, которое называется просто *Terminal*. Команды, которые я представлю, будут использовать оболочку **Bash shell**. Программа терминала Mac по умолчанию использует оболочку **Z shell (zsh)**, которая напрямую совместима с Bash. В большинстве установок Linux по умолчанию используется Bash, так что все уже готово. Если вы используете Windows, вам необходимо установить программное обеспечение Git с <https://gitforwindows.org>. Одна из установленных программ называется *gitbash*. Запуск **gitbash** позволит нам всем использовать одни и те же команды независимо от того, какую операционную систему мы используем.

Установка имени пользователя и адреса электронной почты по умолчанию

Запустив терминал, вам необходимо выполнить следующую команду, чтобы установить имя пользователя по умолчанию:

```
git config --global user.name "FIRST_NAME LAST_NAME"
```

Естественно, вы укажете свое настоящее имя. Далее заполните свой адрес электронной почты. Если вы работаете в корпорации, я настоятельно рекомендую создать отдельную учетную запись GitHub, чтобы отделить вашу работу от работодателей и избежать неприятных споров об интеллектуальной собственности. Имея это в виду, используйте подходящий адрес электронной почты для своей работы с помощью следующей команды:

```
git config --global user.email "MY_NAME@example.com"
```

Помимо использования этих двух глобальных переменных, существуют более обширные параметры конфигурации, но это наиболее распространенный способ настройки Git, и без этих настроек вы не сможете отправить код на GitHub. Если вам нужна более подробная информация о работе с Git, я рекомендую ознакомиться с дополнительной литературой в разделе «Дальнейшее чтение» этой главы.

Генерация ключа SSH

Одна из целей удаленного хаба – защитить ваш код от злонамеренного вмешательства. Один из способов сделать это – обеспечить шифрование всех сообщений между вашим и удаленным компьютером. Вы можете работать с шифрованием HTTPS или SSH. Хотя оба являются эффективными инструментами шифрования, SSH считается гораздо более совершенным и является признаком настоящего профессионала. Для работы с SSH вам необходимо сгенерировать уникальную пару ключей шифрования: один открытый ключ и соответствующий закрытый ключ. Я не буду здесь вдаваться в подробности того, как работает SSH, но, если вам интересно, оставлю несколько рекомендаций для чтения на эту тему в разделе «Дальнейшее чтение» этой главы.

Генерировать ключ довольно просто. Открыв сеанс Bash, просто введите следующее:

```
ssh-keygen
```

Программа генерации ключей задаст вам несколько вопросов, например где хранить файлы ключей и как их называть. По умолчанию они называются `id_rsa` (закрытый ключ) и `id_rsa.pub` (открытый ключ), оба из которых хранятся в вашей домашней папке в подпапке с именем `.ssh`.

Последний заданный вопрос – это фраза пароля для ключа. Вы можете ввести пароль для ключа для дополнительной безопасности или просто нажать *Enter*, чтобы ввести пустой пароль. Это не рекомендуется, поскольку при создании пароля безопасность становится более строгой. Недостатком наличия пароля является то, что вам постоянно приходится вводить некую фразу. Пустая

фраза-пароль обходит эти прерывания, но подвергает риску безопасность, поскольку ваш ключ становится легче взломать. Я рекомендую ввести пароль, поскольку одним из преимуществ использования инструмента PyCharm является то, что он сохраняет ваш пароль в собственной зашифрованной базе данных. PyCharm ответит за вас на все ваши задачи, устраняя неудобства работы с ключом шифрования, защищенным паролем.

После ввода ключа доступа вы получите дополнительную информацию. Вы можете проверить свой ключ, набрав следующее:

```
ls ~/.ssh -a
```

Здесь перечислены (ls) файлы в вашей домашней папке (~/) в подпапке .ssh, включая все скрытые файлы (-a для «всех файлов»). Вы должны увидеть как минимум файлы id_rsa и id_rsa.pub. Содержимое файла id_rsa.pub понадобится вам буквально через мгновение, поскольку вы собираетесь вставить его содержимое на экран GitHub.

Мы подошли к той части книги, где я читаю отеческую проповедь о безопасности. Никогда, никогда, никогда (умножьте на бесконечность в степени Google плюс один) не вставляйте свой закрытый ключ ни во что. Никогда (до тошноты) не позволяйте никому копировать ваш закрытый ключ. Не записывайте его на флеш-накопитель. Не копируйте его на сетевой диск или в облачную папку для безопасного хранения. Ради всего святого, никогда не загружайте это в репозиторий! Это касается любых паролей для любой системы. Ваши репозитории могут в конечном итоге стать общедоступными если не для всего мира, то, по крайней мере, для остальной части вашей команды.

Открытый ключ предназначен для обмена. Закрытый ключ должен храниться в секрете и скрыто. Если ваш закрытый ключ скомпрометирован, только пароль защитит вашу работу. Если вы оставили фразу пароля пустой, у вас больше нет защиты!

Чтобы отобразить содержимое вашего открытого ключа, введите следующее:

```
cat ~/.ssh/id_rsa.pub
```

Содержимое вашего открытого ключа появится в окне терминала. Выделите весь ключ, кликните правой кнопкой мыши и выберите **Copy**.

Добавление ключа SSH в вашу учетную запись GitHub

Я расскажу об этом довольно кратко, так как подозреваю, что многие из вас делали это раньше. Если вам нужно более подробное руководство по добавлению ключа SSH в вашу учетную запись GitHub, см. раздел «Дальнейшее чтение» в конце этой главы, где приведен подробный список шагов, специфичных для Git.

Войдите в GitHub и кликните свое лицо. Под этим я имею в виду ваш аватар на GitHub в правом верхнем углу, предполагая, что они не меняли дизайн своего сайта с момента публикации. Найдите опцию работы со своим профилем и найдите **Settings**. В настройках вашего профиля есть возможность управлять ключами SSH. Нажмите эту ссылку, затем найдите возможность добавить ключ SSH. Вставьте открытый ключ, который вы скопировали минуту назад, дайте ему имя, которое легко запомнить, и нажмите кнопку **Add**.

Поздравляем! Вы полностью настроены для безопасной связи между вашим компьютером и удаленными устройствами GitHub. Ключ SSH можно повторно использовать на разных компьютерах, которыми вы владеете. Просто скопируйте папку `.ssh` на все компьютеры. Только будьте осторожны и не скомпрометируйте ключ, храня его на портативном диске, который вы можете потерять, или в облачном файловом ресурсе, который может быть скомпрометирован.

НАСТРОЙКА РЕПОЗИТОРИЯ ВРУЧНУЮ

В главе 3 мы вручную создали виртуальную среду, чтобы ознакомиться с процессом. Мы сделаем то же самое здесь. Давайте воспользуемся командной строкой, чтобы создать репозиторий и выполнить все основные функции, доступные в Git. Позже мы увидим, как инструменты PyCharm позволяют нам выполнять те же операции прямо из IDE в удобном графическом интерфейсе.

Не используйте пробелы в именах файлов и папок

Я буду часто поднимать этот вопрос, так как знаю, что многие читатели пропускают такие темы в книгах, подобных этой. Если вы новичок в разработке программного обеспечения, возможно, вы не знаете, что использование имен файлов с пробелами может вызывать проблемы. Избегайте использования имен файлов и папок с пробелами, включая любые папки, созданные вашей операционной системой. Вместо этого используйте разделители вместо пробелов, например тире или подчеркивание, или используйте имена в верблюжьем регистре, в которых пробелы отсутствуют, а границы слов обозначаются заглавной буквой¹.

Когда сеанс терминала Bash открыт, первое, что мы собираемся сделать, – это создать папку для нашего проекта. Мы сделаем это в вашей домашней папке с помощью команды:

```
mkdir first-repository
```

Далее давайте изменим каталог на вновь созданную папку:

```
cd first-repository
```

Здесь будет жить наш код! Однако, прежде чем мы пойдем дальше, нужно поговорить на деликатную тему.

¹ Верблюжий регистр – это стиль именования, используемый в программировании, особенно в JavaScript, где начальная буква каждого слова в имени, состоящем из нескольких слов, пишется с большой буквы, за исключением первого слова, которое остается строчным. Это соглашение называется «camelCase», потому что заглавные буквы напоминают горбы на спине верблюда. – *Прим. ред.*

Мастер-ветви и главные ветви в GitHub

В течение первых 12 лет после создания GitHub при каждом создании нового репозитория в GitHub инструмент создавал стартовую ветку по умолчанию под названием *master*. В 2020 году организация *The Software Freedom Conservancy* посоветовала GitHub изменить название ветки по умолчанию с *master* на *main* из-за ассоциации термина *master* с рабством. Хотя Git не обязывает вас использовать какое-либо конкретное имя, значение по умолчанию на GitHub было изменено на *main*. Поскольку GitHub не контролирует Git, это изменение имени применяется по умолчанию только для репозиториях, созданных GitHub. Мы создавали наши репозитории с помощью командной строки Git, которая используется инструментами Git PyCharm. Независимо от того, используете ли вы команду Git или интеграцию PyCharm, есть большая вероятность, что вы получите репозиторий, инициализированный с помощью ветки под названием *master*. Когда вы создаете удаленный сервер на GitHub, этот репозиторий будет создан с веткой по умолчанию под названием *main*.

Вам следует позаботиться о том, чтобы имена ветвей по умолчанию совпадали и при первом нажатии ваши ветки синхронизировались должным образом. Вы можете либо изменить имя в GitHub, либо настроить команду Git на своем компьютере на использование *main* по умолчанию.

Если вы хотите изменить его на GitHub, можете зайти в настройки своего профиля и нажать **Repositories**. Вы найдете настройку вверху, как показано на рис. 5.1.

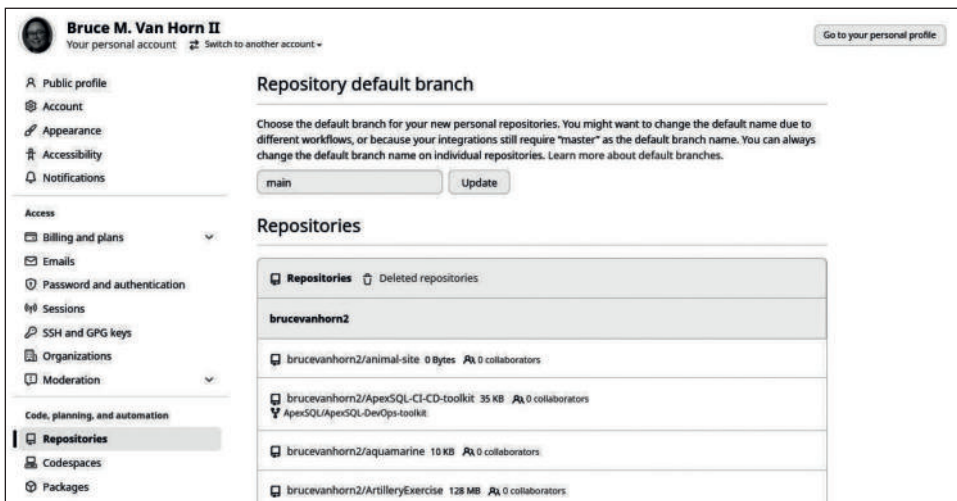


Рис. 5.1. Вы можете установить имя репозитория по умолчанию в GitHub, чтобы оно соответствовало имени по умолчанию на вашем компьютере

Вы также можете установить глобальные настройки на своем компьютере с помощью этой команды:

```
git config --global init.defaultBranch main
```

Если вы собираетесь много работать с GitHub в качестве удаленного устройства, рекомендуется синхронизировать имя ветки по умолчанию. Если вы этого не сделаете, при первой попытке отправить репозиторий, созданный на вашем компьютере, на удаленный компьютер, созданный на GitHub, вы столкнетесь с ошибкой. Локально по умолчанию установлено значение *master*, а на удаленном компьютере – *main*. На удаленном компьютере не будет ветки с именем *master*, поэтому вы получите сообщение о том, что нет вышестоящей ветки с именем *master*. Вам придется поработать, чтобы уладить это. Оставляю ссылку на статью о том, как решить эту ситуацию, если вы с ней столкнетесь. Я советую вам просто избегать этого, используя предыдущую команду Git, чтобы установить для локальной глобальной настройки значение *main*, чтобы она соответствовала GitHub.

Ручная инициализация репозитория

Давайте создадим наш репозиторий. Вполне возможно и даже нормально создать репозиторий после того, как вы создали проект кода. Однако мы начнем с репозитория, поскольку это будет очень простая демонстрация. Чтобы создать новый репозиторий Git, или сокращенно *repo*, введите следующее:

```
git init
```

Это было легко! Может показаться, что ничего особенного не произошло. Вы получите сообщение о том, что репозиторий создан. Посмотрим на изменения в папке:

```
ls -a
```

Команда `ls` выводит список всех файлов. Переключатель `-a` показывает скрытые файлы и папки. Когда вы инициализировали репозиторий Git, программное обеспечение Git создало папку с именем `.git`. Поскольку имя папки начинается с точки (`.`), в Linux-подобных системах она скрыта. Если вы выполните эти действия в Windows с помощью PowerShell, то сможете увидеть папку, но в других операционных системах она не видна.

В папке `.git` хранятся все ваши версии, а также все настройки вашего проекта. В общем, вам никогда не понадобится изменять содержимое этой папки.

У нас есть репозиторий на нашем локальном компьютере. Добавим новый файл. Введите следующее:

```
echo "hello world" > test.txt
```

Это создало новый файл с именем `test.txt`. В этом файле есть строка `hello world`. Какой отличный способ начать, правда? Вы можете проверить содержимое файла, набрав следующее:

```
cat test.txt
```

Кому нужна IDE, верно? Подождите, нет, забудьте, что я это сказал. Порядок следующих нескольких шагов важен. Процесс выглядит следующим образом.

1. Добавьте файл(ы) в репозиторий. Добавляемые вами файлы – это файлы, которые вы создали или изменили с момента вашего последнего коммита. У нас пока нет такого. Добавление файлов в репозиторий также называется промежуточным размещением файлов. Мы информируем репозиторий о новых и измененных файлах. Не всегда следует добавлять в проект каждый файл. Некоторые из них следует держать подальше от своего репозитория. Мы рассмотрим это чуть позже.
2. Зафиксируйте свои файлы. Это создаст новую версию вашего кода на вашем локальном компьютере. Действие фиксации требует, чтобы вы включили комментарий, объясняющий характер внесенных вами изменений. Это жизненно важно! Вам следует постараться написать краткое изложение ваших изменений и написать эти комментарии так, как будто ваш самый важный клиент однажды прочитает все эти комментарии. Это не то место, где можно выразить свое разочарование! Поверьте мне. Я видел, как проекты попадали под неожиданную проверку, когда появлялись грубые или неуместные комментарии и разрушались рабочие отношения. Представьте себе – все, что вы напишете, будет прочитано и оценено всем миром.
3. Извлеките изменения из удаленного хаба. Вам всегда захочется сделать прием (**Pull**), прежде чем сделать передачу (**Push**)¹. Это позволяет вам обнаружить любые конфликты между вашей работой и любой работой, которая могла быть реализована другим разработчиком проекта. Это важно! Если вы не сделаете **Pull** перед отправкой, можно перезаписать версию другого разработчика! Это, конечно, не катастрофично, поскольку все изменения хранятся в репозитории. Однако это сделает завтрашнее стендап-совещание очень неловким для вас, поскольку другой разработчик, скорее всего, не будет доволен тем, что вы задаете дополнительную работу для группы! Это просто. Просто не забудьте принять, прежде чем передать. Сделайте это, даже если вы единственный человек, работающий над проектом, и просто сделайте это привычкой!
4. Предположим, что конфликтов нет, – тогда перенесите свою работу на удаленный хаб.

Ваши изменения теперь доступны остальной команде и надежно хранятся на удаленном сервере. Давайте рассмотрим команды для этого процесса.

Сначала добавьте новые и измененные файлы в Git. Помните, это также называется **размещением** ваших файлов (**staging**):

```
git add test.txt
```

Если в вашем проекте есть несколько файлов, их можно добавить все сразу, выполнив следующие действия:

```
git add .
```

После подготовки последних изменений вы можете зафиксировать их в локальном репозитории. Это создаст новую версию вашего проекта на компью-

¹ Для синхронизации с другими репозиториями в Git используются команды **Push** (передача) и **Pull** (прием). – Прим. ред.

тере. Необходимо добавить комментарий, в котором суммируются ваши изменения:

```
git commit -m "initialized repository and added test.txt which might be the best code in the world."
```

Ключ `-m` – это наш комментарий, заключенный в двойные кавычки. Просто запомните это как *сообщение о фиксации*, и ключ `-m` легко запомнить. Конечно, к концу главы вам не нужно будет запоминать всю эту информацию о командной строке, поскольку у вас будет отличный графический интерфейс.

Работа с удаленным доступом

Работать с проектом можно полностью на локальном компьютере. Естественно, вы потеряете все преимущества совместной работы, предоставляемые удаленным хабом, включая преимущества удаленного резервного копирования истории изменений. Ранее я представил четырехэтапный процесс.

1. Добавьте или разместите измененные файлы.
2. Зафиксируйте изменения, чтобы создать новую версию.
3. Извлеките изменения с удаленного компьютера.
4. Перенесите объединенную работу на удаленный компьютер.

Мы прошли половину списка, выполнив первые два шага в нашей локальной копии репозитория. На этом этапе мы должны прервать процесс, потому что в настоящее время у нас нет удаленного доступа. Мы собираемся создать его на GitHub, но, естественно, вы можете использовать любой совместимый сервис.

Добавление удаленного доступа на GitHub

Войдите в GitHub и найдите кнопку **Add Repository**. Обычно это очень приметная зеленая кнопка. Если мне повезет, ко времени публикации они уже переделают сайт. Тем не менее они обычно делают кнопку **Create repository** очень заметной в пользовательском интерфейсе.

Давайте создадим репозиторий, соответствующий тому, который мы создали локально. Я сделаю свой общедоступным, а это значит, что все смогут его увидеть. Если вы застенчивы, можете сделать его приватным. Для этого упражнения это не будет иметь значения.

После создания репозитория GitHub отобразит инструкции по добавлению версии репозитория GitHub в качестве удаленного. Просто скопируйте и вставьте код, сгенерированный GitHub, в свой терминал, убедившись, что ваш терминал в настоящее время находится в папке репозитория в качестве текущего рабочего каталога.

Вы можете проверить удаленный доступ, набрав следующее:

```
git remote -a
```

Команда, сгенерированная GitHub, устанавливает имя копии репозитория GitHub в качестве *источника*. У вас должен быть один удаленный репозиторий с именем *origin*, а на локальную версию репозитория ссылается текущая ветка, обычно это *main*, как описано ранее.

Первая передача

Теперь, когда у вас есть удаленный компьютер, можете перенести на него свой локальный репозиторий. Введите следующее:

```
git push origin main -u
```

Это отправит вашу копию основной ветки на удаленный компьютер и установит отслеживание восходящей ветки для основной ветки. Это просто причудливый способ сказать, что и удаленный, и локальный репозиторий знают об основной ветке, и любое нажатие здесь приведет к обновлению удаленной основной ветки. Если на удаленном компьютере нет ветки с именем `main`, она будет создана, и все будет хорошо синхронизироваться.

Внесение, принятие и передача изменений

Теперь, когда у нас есть полностью рабочий репозиторий с удаленным доступом на GitHub, давайте внесем один полный цикл изменений в наш код. На этот раз мы рассмотрим четырехэтапный процесс без перерыва. Это процесс, которому вы будете следовать много раз в день на протяжении всего проекта.

Следующий шаг очень важен. Сделайте это привычкой!

```
git pull
```

Это осуществляет прием любых изменений, находящихся на удаленном компьютере, и позволяет исправлять любые конфликты до того, как они выйдут наружу! Естественно, в этой ситуации никаких конфликтов не будет, поскольку над проектом работаем только мы.

Далее давайте изменим содержимое нашего текстового файла. Введите следующее:

```
echo "this is the best code in the world" > test.txt
```

Проверить результат можно следующим образом:

```
cat test.txt
```

Вы должны увидеть измененный файл. Теперь давайте выполним полный цикл добавления, фиксации, извлечения и отправки. Введите каждую из следующих строк по одной и нажмите *Enter* в конце каждой:

```
git add test.txt
git commit -m "changed the text in test.txt"
git pull
git push origin main
```

Переключитесь в браузер, в котором вы создали свой репозиторий, и обновите страницу. Вы должны увидеть свой код. Вы можете кликнуть файл `test.txt` в браузере, чтобы убедиться, что новое содержимое было отправлено. Вы также должны увидеть, что есть две версии.

Ваш репо настроен и работает правильно! Давайте переключим внимание с командной строки на работу в IDE.

РАБОТА С GIT В IDE

Мы уже рассмотрели некоторые рабочие процессы Git в PyCharm. В главе 2 мы использовали PyCharm для клонирования репозитория примера кода для этой книги. Поскольку мы рассмотрели клонирование – не будем делать этого снова. Вместо этого давайте рассмотрим все, что мы сделали вручную минуту назад.

1. Мы использовали `git init` для инициализации нового локального репозитория.
2. Внесли изменения в наш код.
3. Использовали `git add`, чтобы добавить изменения в виде промежуточных файлов при подготовке к началу.
4. Зафиксировали наши изменения в локальном репозитории.
5. Мы извлекли данные из удаленной версии, чтобы убедиться, что у нас есть последняя версия кода в этой ветке и что нет конфликтов между тем, что есть в нашей рабочей копии, и тем, что существует на удаленном компьютере. Версия на удаленном компьютере могла недавно измениться из-за того, что какой-то другой разработчик внес свои изменения.
6. Наконец, мы перенесли наши изменения на удаленный компьютер.

Этот список представляет собой самый простой рабочий процесс для сбора изменений в проекте и управления ими. Все это и многое другое мы можем сделать в графическом интерфейсе PyCharm. Также довольно часто часть работы выполняется в PyCharm, а часть – в командной строке. Использование того или другого ничего не исключает. Например, я часто разветвляюсь и объединяюсь в командной строке, потому что для меня это быстро и легко. Если возникает конфликт, я считаю, что его разрешение в PyCharm во всех отношениях лучше, чем ручной подход. Графический интерфейс для этой задачи великолепен. Мы доберемся до всего этого в свое время. А пока откройте папку, которую вы использовали в PyCharm, и давайте посмотрим, что IDE может сделать для нас в отношении этих первых шагов, которые мы предприняли вручную.

КОНТРОЛЬ ВЕРСИЙ В PYCHARM

PyCharm имеет несколько мест в пользовательском интерфейсе, которые позволяют вам получить доступ к вашей системе контроля версий, для нас это Git. Они не являются лишними, а вместо этого каждая область предоставляет визуальный инструмент для конкретной задачи. Первое – это главное меню, как показано на рис. 5.2.

Второе обеспечивает быстрый и простой способ фиксации (коммита) файлов. Его можно найти в меню боковой панели, как показано на рис. 5.3:

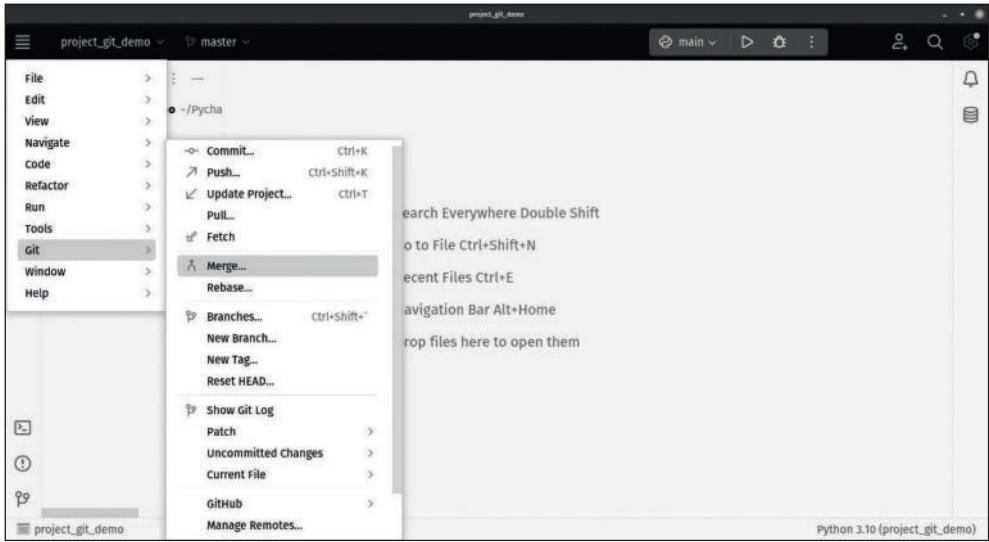


Рис. 5.2. Меню Git позволяет получить доступ ко всем командам, которые вы обычно используете в командной строке

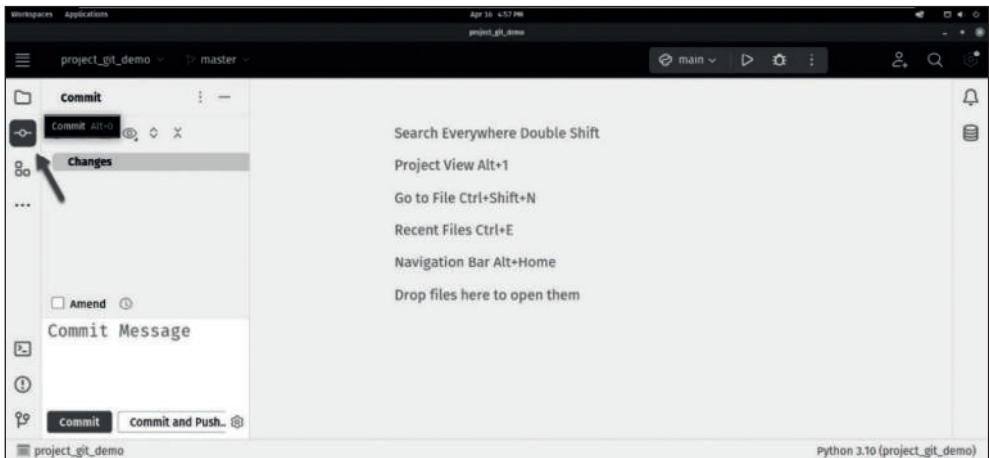


Рис. 5.3. Инструменты фиксации Git можно найти на боковой панели

Третье позволяет визуализировать историю ветвей и коммитов. Его можно найти в окне инструментов, расположенном в нижней части экрана. Эта часть экрана сворачивается, но инструмент выбора окна инструментов всегда доступен в нижней части боковой панели, как показано на рис. 5.4.

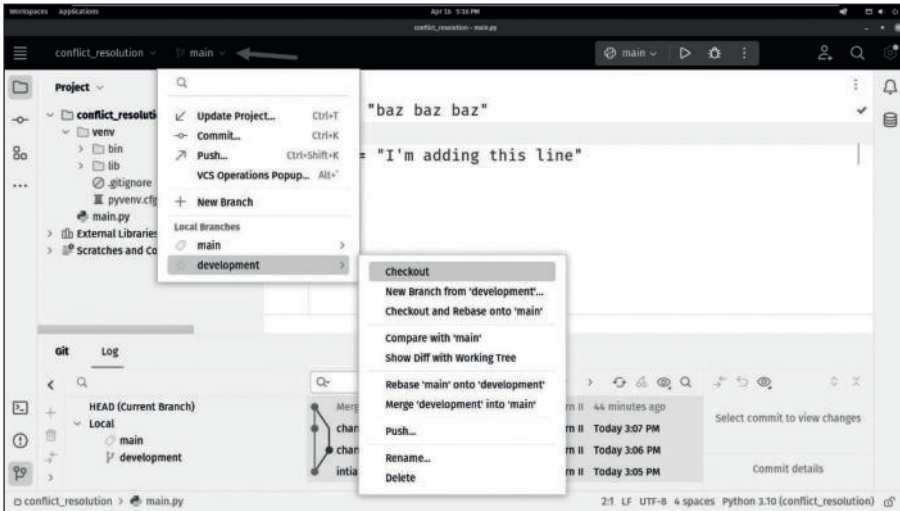


Рис. 5.4. При нажатии на окно инструмента Git откроется окно, в котором отображается история commit, а также все ветки

Четвертое находится на верхней панели инструментов рядом с главным меню. Эта область показывает вам ветку, над которой вы сейчас работаете, и позволяет легко управлять веткой. Вы можете увидеть это на рис. 5.5.

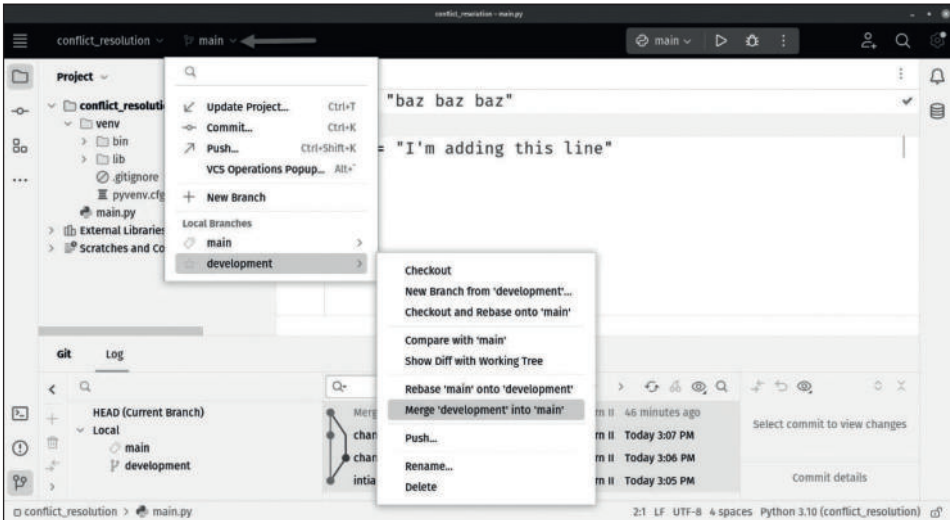


Рис. 5.5. Текущая ветка отображается рядом с главным меню. Нажав здесь, вы получите доступ к нескольким командам

Пятое – всплывающее окно операций VCS, как показано на рис. 5.6. Оно существует уже довольно давно, и я бы посчитал это наследием, хотя оно все еще существует. Вы можете перейти к всплывающему окну, используя *Ctrl* + *`*. Обратите внимание, что это не кавычка. Клавиша «`», называемая знаком глубокой печати, расположена слева от клавиши *1* (один) на американской раскладке клавиатуры.

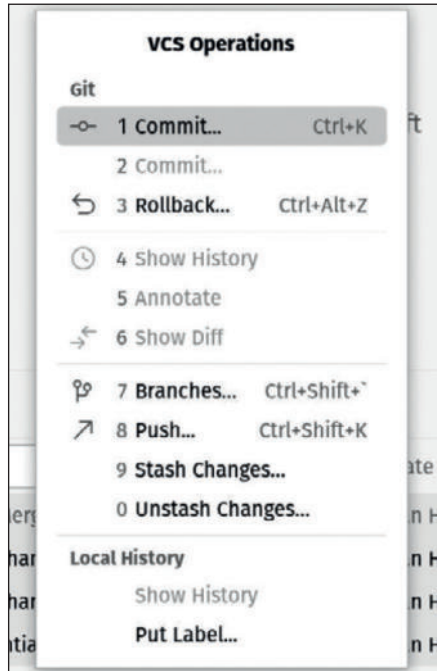


Рис. 5.6. Всплывающее окно операций VCS активируется с помощью Ctrl/Cmd + `

Наконец, вы можете кликнуть правой кнопкой мыши окно редактора или вкладку в окне редактора, чтобы получить доступ к тому же меню Git, которое находится в главном меню, как показано на рис. 5.7.

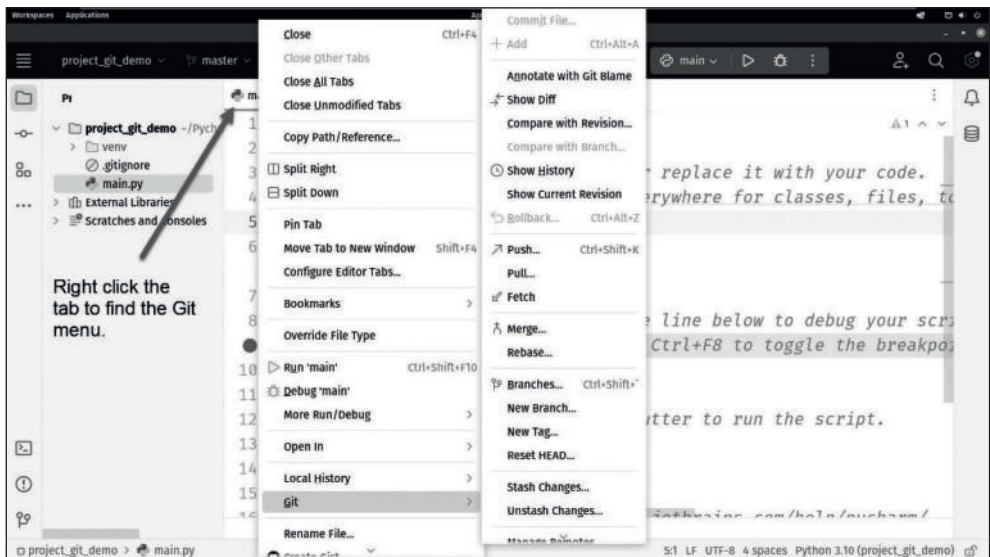


Рис. 5.7. Вы можете кликнуть правой кнопкой мыши в окне редактора или на вкладке, чтобы увидеть меню Git

Как видите, инструменты для контроля версий аккуратно вплетены в IDE организованным и интеллектуальным образом. Поскольку мы открыли папку, в которой уже был репозиторий Git, PyCharm просто распознает этот факт и представляет интерфейс. Когда вы открываете папку или создаете проект без репозитория GitHub, большая часть инструментов по-прежнему будет видна, но она не будет специфичной для Git. Поскольку PyCharm поддерживает множество систем контроля версий, проект, который не был инициализирован с помощью Git или любой другой системы контроля версий, будет предоставлять общие инструменты до тех пор, пока такая инициализация не будет завершена.

СОЗДАНИЕ НОВОГО ПРОЕКТА С НУЛЯ С ИСПОЛЬЗОВАНИЕМ ИНСТРУМЕНТОВ VCS В PYCHARM

Давайте сделаем новый проект. Код этого проекта находится в репозитории книги, но вам не стоит просто открывать эту копию, поскольку она уже связана с репозиторием, и вы не увидите всего, что я собираюсь вам показать. Для этого упражнения вам следует создать новый проект с нуля, как я собираюсь это сделать.

Нажмите значок гамбургера в верхнем левом углу приложения, чтобы активировать главное меню, затем нажмите **File | New Project**. Я назову свой проект `project_git_demo`. Все остальное оставлю по умолчанию. Диалоговое окно создания моего проекта вы можете увидеть на рис. 5.8.

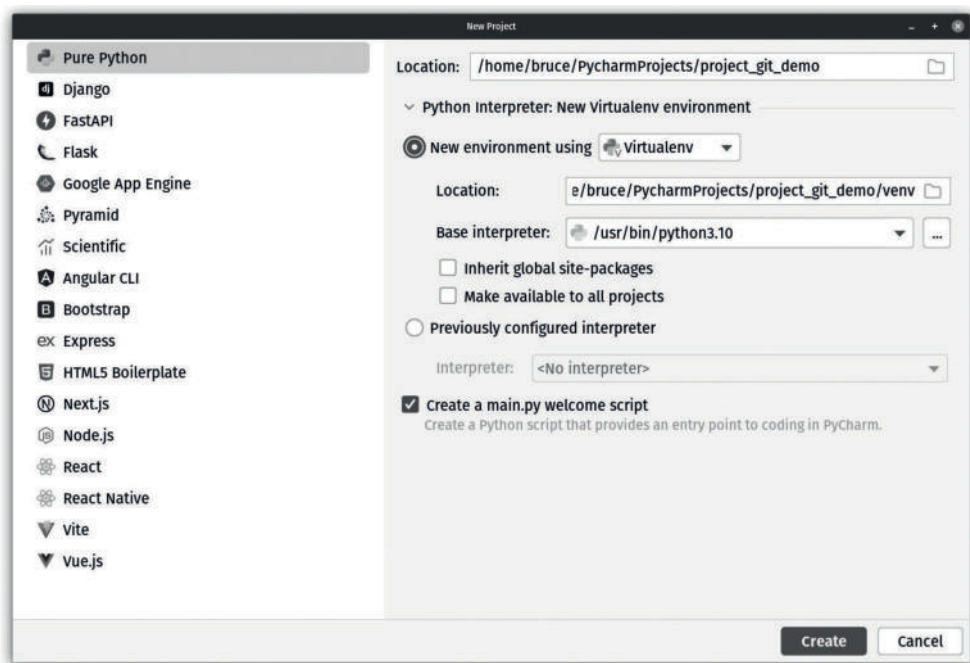


Рис. 5.8. Настройки моего демонстрационного проекта

Теперь у нас есть проект, с которым мы можем работать.

Инициализация локального репозитория Git

После создания моего проекта мне нужно инициализировать репозиторий Git. Мы сделали это вручную с помощью команды `git init`. В PyCharm мы можем использовать опцию **VCS** в главном меню. Прямо сейчас там написано **VCS**, потому что мы еще не инициализировали репозиторий с помощью какой-либо системы VCS, поэтому терминология в пользовательском интерфейсе является общей. Как вы можете видеть на рис. 5.9, у нас есть варианты контроля версий в Mercurial, Git, GitHub, Perforce и Subversion. Появилась новая возможность поделиться проектом в новом продукте для совместной работы JetBrains под названием **Space**, который содержит службу хостинга Git. Естественно, по мере роста продукта Space можно ожидать появления все большей и большей интеграции во всех IDE. Я собираюсь нажать **Create Git Repository...**, как показано на рис. 5.9.

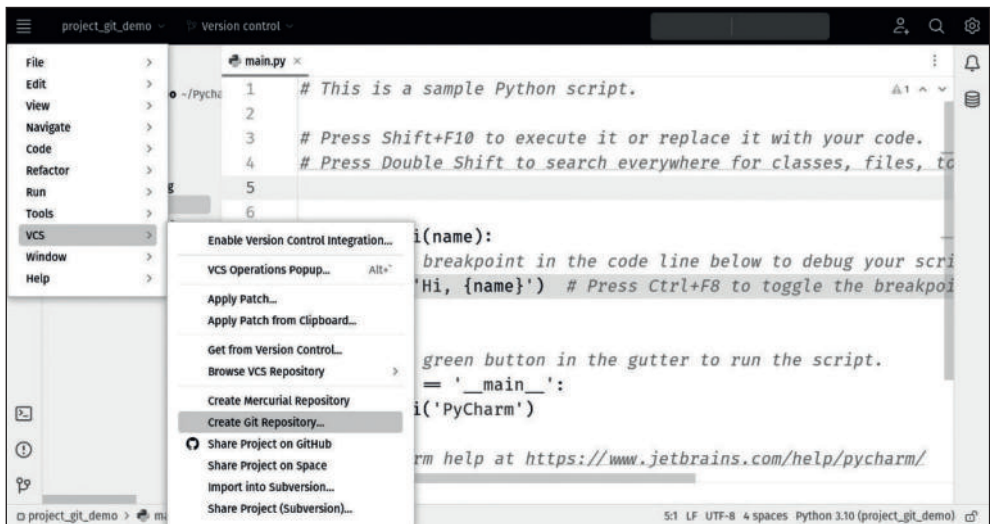


Рис. 5.9. Создание репозитория Git из меню PyCharm VCS

Как можно видеть на рис. 5.10, вам будет предложено выбрать папку, в которой вы хотите создать репозиторий. По умолчанию используется текущая папка, и это правильно.

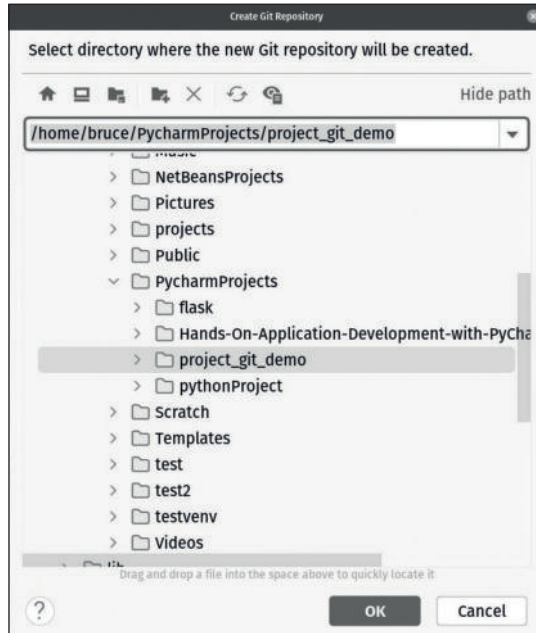


Рис. 5.10. Выберите папку, в которой вы хотите создать репозиторий Git

Нажатие кнопки **OK** создает репозиторий в выбранной папке. Вы можете убедиться, что это работает, проверив, отображается ли ветка по умолчанию (master) на верхней панели инструментов с раскрывающимся списком, содержащим несколько операций Git. Теперь, когда мы инициализировали репозиторий Git, в меню больше не используется термин **VCS**, а вместо этого перечисляются параметры, специально предназначенные для Git.

Добавление удаленного доступа на GitHub

В PyCharm нет инструментов для создания удаленного доступа. Вам нужно войти в GitHub и создать репозиторий, как вы это делали ранее. На рис. 5.11 показано, как я настраиваю удаленный репозиторий в своей личной учетной записи. Обратите внимание, что это отличается от репозитория книги, который предназначен для размещения всего кода книги, а не для демонстрации создания репозитория.

Create a new repository
A repository contains all project files, including the revision history. Already have a project repository elsewhere? Import a repository.

Owner * brucevanhorn2 / **Repository name *** project_git_demo ✓

Great repository names are short and memorable. Need inspiration? How about ideal-broccoli?

Description (optional)
This is a demo project used in my book Hands on Application Development with PyCharm 2nd Edition

☒ **Public**
Anyone on the internet can see this repository. You choose who can commit.

☐ **Private**
You choose who can see and commit to this repository.

Initialize this repository with:
Skip this step if you're importing an existing repository.

☐ **Add a README file**
This is where you can write a long description for your project. Learn more.

Add .gitignore
Choose which files not to track from a list of templates. Learn more.
.gitignore template: None

Choose a license
A license tells others what they can and can't do with your code. Learn more.
License: None

ⓘ You are creating a public repository in your personal account.

Create repository

Рис. 5.11. Настройки моего удаленного доступа

Обратите внимание, что установлен только самый минимум. Нет файла .gitignore, нет файла README и т. д. Я собираюсь добавить все это в PyCharm. Когда я нажимаю кнопку **Create repository**, GitHub генерирует URL-адрес, который мне нужен для добавления удаленного устройства. Я указал на это на рис. 5.12.

Затем вернитесь к PyCharm и найдите меню **git** в главном меню. Помните, минуту назад в меню было написано **VCS**, потому что мы не определили репозиторий, а теперь написано **git**.

В меню **git** вы найдете опцию **Manage Remotes**. Нажмите на нее, и вы увидите модальное диалоговое окно, показанное на рис. 5.13, которое позволяет вам добавить только что созданный вами удаленный доступ на GitHub.

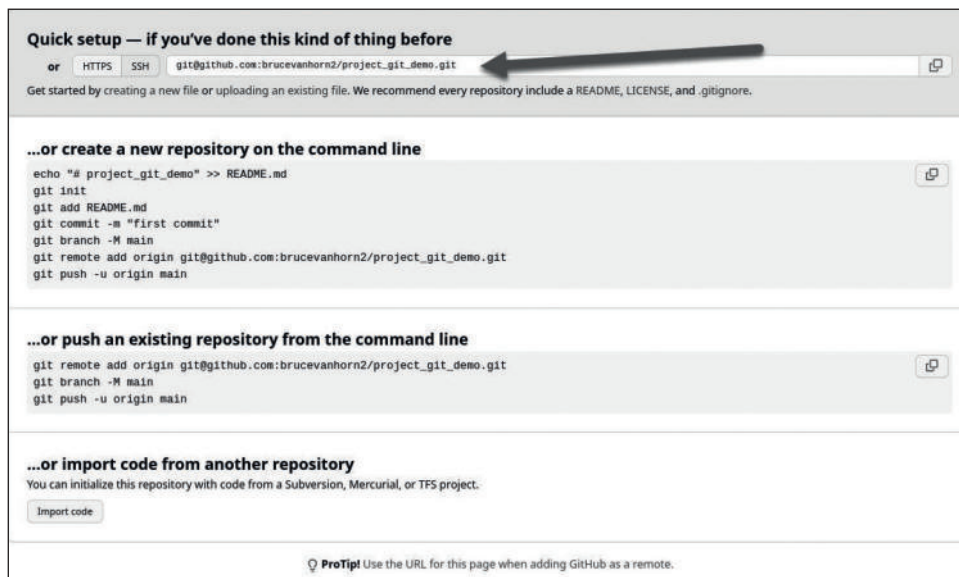


Рис. 5.12. Вам нужен URL-адрес удаленного устройства, чтобы добавить его в локальный репозиторий

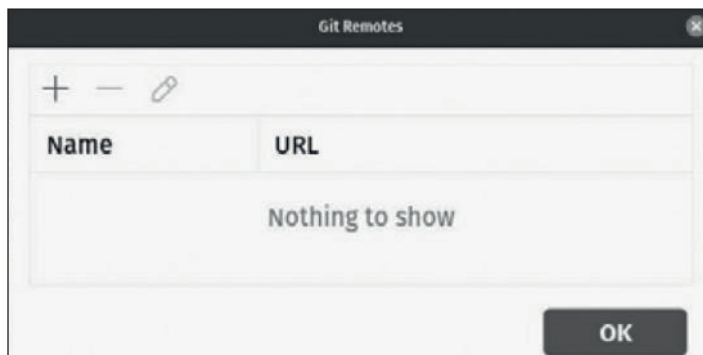


Рис. 5.13. Диалоговое окно Git Remotes позволяет добавить удаленный доступ, созданный вами на GitHub

Нажмите кнопку +, чтобы добавить удаленный доступ. Поверх этого диалогового окна появится еще одно, как показано на рис. 5.14.

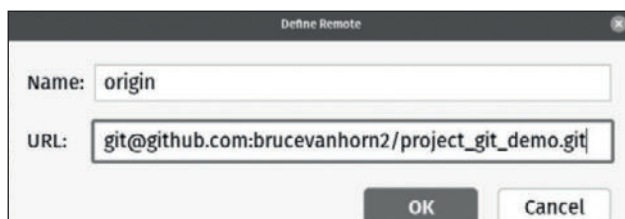


Рис. 5.14. Добавьте URL-адрес, скопированный со страницы GitHub

Нажмите **ОК** в диалоговом окне **Define Remote** и еще раз в диалоговом окне **Manage Remotes**. Вы увидите некоторые тонкие изменения. Файл `main.py` имеет красный цвет. Вам просто придется мне поверить, что он красный. Он красный, потому что это неотслеживаемый файл. Нам нужно добавить наши файлы в наш репозиторий, чтобы мы могли их отслеживать.

Добавление файлов проекта

Настройка проекта и удаленного управления обычно выполняется только один раз за весь проект. Если вы присоединяетесь к существующей команде, возможно, это было сделано задолго до того, как вы присоединились. Однако со следующим набором действий вы будете работать каждый день.

Commit окажется очень полезным. Вернитесь к рис. 5.3, если вы забыли, как найти окно **Commit**.

Добавление файла .gitignore

Если вы еще не знакомы с этим, файл `.gitignore` определяет файлы и папки, которые вы не хотите включать в свой репозиторий. Как правило, все, что может быть сгенерировано из вашего кода, не обязательно должно находиться в вашем репозитории. При работе с проектами Python краткий список вещей, которые не нужны в вашем репозитории, может включать следующее:

- папку `venv`, поскольку она создается при запуске `pip`,
- папки кеша Python (вы увидите их как папки `__pycache__`),
- байт-компилированные файлы `DLL`,
- расширения `C`,
- сборку файлов из дистрибутива и упаковки,
- манифесты `PyInstaller`,
- файлы журналов приложений,
- отчеты о тестовом покрытии.

Это ни в коем случае не исчерпывающий список. Список на самом деле будет зависеть от того, какое приложение вы создаете и что именно генерируется во время выполнения вашего проекта. Существует множество предложений по содержимому файла `.gitignore`. В GitHub есть такой список. Это также файл `.gitignore` в репозитории Git книги. Вы можете использовать файл `.gitignore`, если хотите, но для краткости и простоты я просто включу несколько записей в файл `.gitignore`, который мы создадим.

Вернитесь в режим просмотра файлов, как показано на рис. 5.15. Это можно сделать, кликнув значок папки (1). Затем кликните правой кнопкой мыши заголовок проекта `project_git_demo` (2) и создайте новый файл. Назовите это `.gitignore`. Никакое творчество здесь недопустимо. Его имя должно быть `.gitignore`, все строчные.

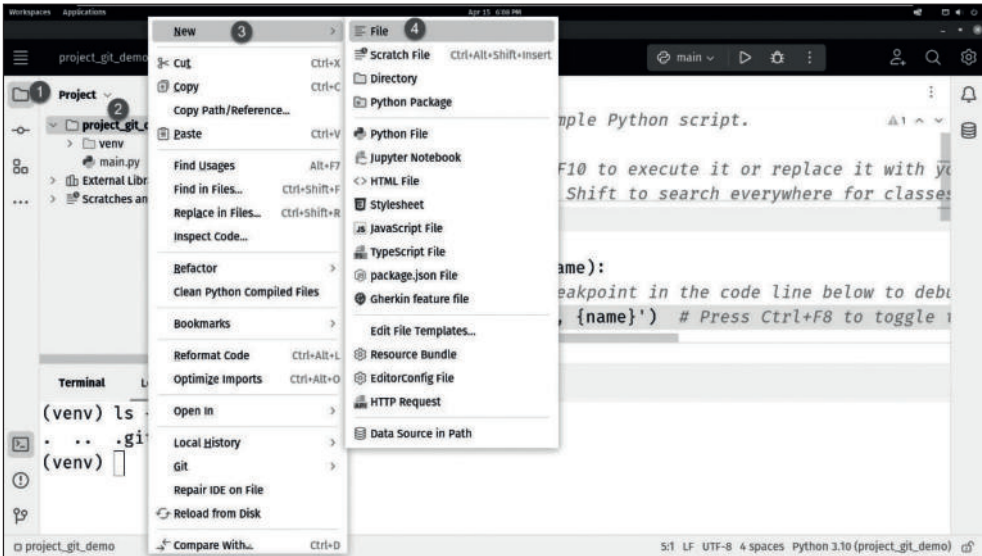


Рис. 5.15. Вернитесь в режим просмотра файлов и создайте новый файл с именем .gitignore

Когда вы создадите новый файл, PyCharm предложит вам добавить его в Git, как показано на рис. 5.16. Если вы хотите добавить новые файлы, чтобы они стали файлами по умолчанию, можете установить флажок **Don't ask again**, и все созданные вами файлы будут автоматически добавлены в GitHub.

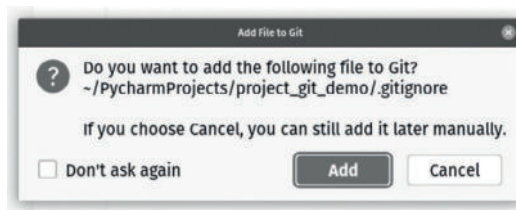


Рис. 5.16. Каждый раз, когда вы создаете файл в PyCharm, вам предлагается добавить его в репозиторий

Независимо от того, что вы делаете с флажком, нажмите кнопку **Add**, чтобы добавить файл. Обратите внимание, что файл .gitignore в проводнике проекта отображается зеленым, поскольку он был добавлен, и пока ничего не изменилось. В файл .gitignore добавьте следующие строки:

```
venv
__pycache__
```

Это исключит всю папку venv и любые папки кеша, созданные Python. PyCharm сохраняет ваши файлы по ходу работы, поэтому давайте добавим этот файл и файл main.py в репозиторий. Переключитесь в окно **Commit**, и вы увидите что-то вроде рис. 5.17.

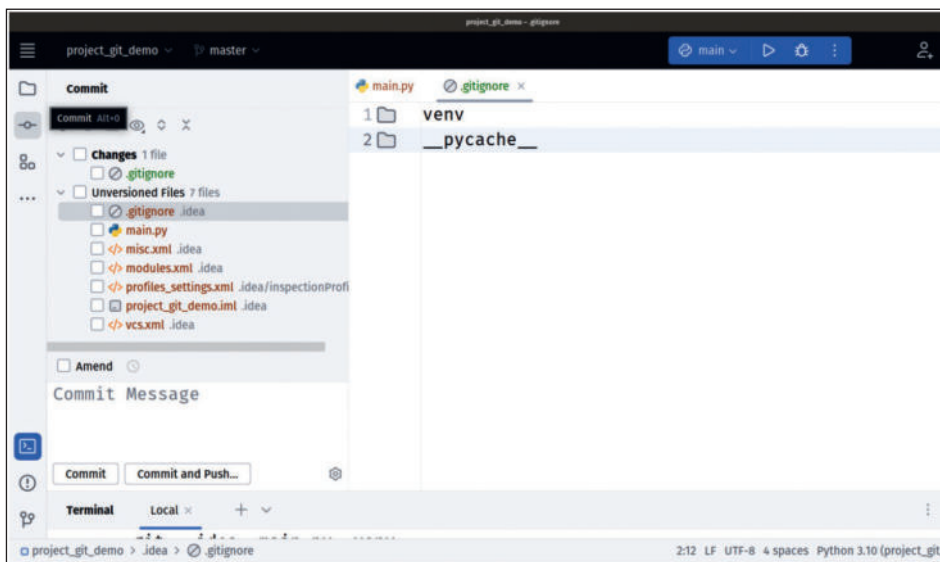


Рис. 5.17. Окно фиксации после добавления файла .gitignore

Ух ты! Откуда все это взялось? Мы только что добавили один файл! Как видите, файл .gitignore был добавлен в ответ на диалоговое окно, которое мы видели при создании файла. Единственный другой файл кода в проекте – это файл main.py, созданный PyCharm при создании проекта. А что насчет остального? Почему в списке неверсионных файлов указан второй файл .gitignore?

Все они находятся в папке .idea, созданной PyCharm. Есть некоторые споры о том, принадлежит ли эта папка системе контроля версий. В документации JetBrains есть страница <https://intellij-support.jetbrains.com/hc/en-us/articles/206544839> с обсуждением этого вопроса. Подводя итог статьи, JetBrains рекомендует хранить в папке .idea содержимое, относящееся к проекту, а не любые файлы, содержащие пользовательские настройки, а именно:

- workspace.xml,
- usage.statistics.xml,
- все в папке shelf.

Цель статьи – охватить все IDE на базе **IntelliJ IDEA**, поэтому большая часть статьи будет относиться к вещам, не имеющим отношения к работе на Python, например артефактам из проектов кодирования Android. Поскольку я пока не сделал ничего, что могло бы создать эти файлы или папки, их нет в списке. Хотя это возможно в будущем. Любой из моей команды также может внести свой вклад в проект и добавить файлы в репозиторий, даже не осознавая этого, что может изменить способ действия PyCharm в соответствии с чьими-либо предпочтениями. Поскольку это может быть предметом беспокойства, рекомендуется добавить их в файл .gitignore. Мы еще ничего не зафиксировали, поэтому добавьте эти строки в свой файл .gitignore:

```
.idea/workspace.xml
.idea/usage.statistics.xml
.idea/shelf
```

Давайте отметим все флажки, чтобы добавить все эти файлы, как показано на рис. 5.18.

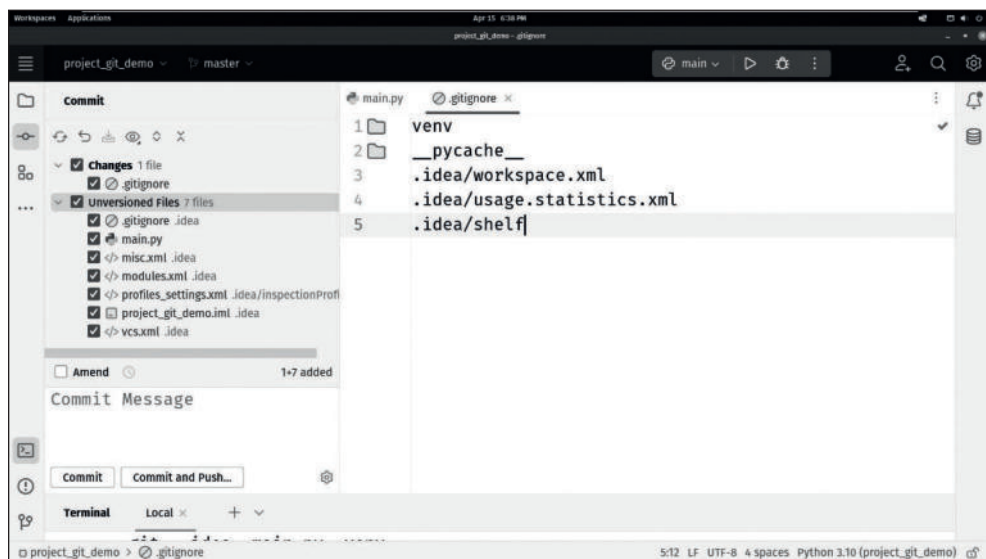


Рис. 5.18. Все добавлено, и мы готовы к фиксации

Добавьте сообщение о фиксации в поле. Вы можете нажать кнопку **Commit**, чтобы зафиксировать изменения в локальном репозитории, или **Commit and Push...**, чтобы отправить изменения прямо на GitHub. Как правило, не следует использовать кнопку **Commit and Push...**, если вы не полностью уверены, что никто не мог отправить изменения с момента последнего нажатия. Вероятно, это безопасно, потому что мы ничего не отправляли; однако я собираюсь нажать кнопку **Commit**, чтобы продемонстрировать весь процесс, изложенный ранее. Вы добавляете свои изменения, фиксируете, извлекаете, а затем отправляете. На данный момент мы добавили наши изменения и теперь фиксируем их.

Если все пойдет хорошо, в правом нижнем углу экрана появится всплывающее сообщение о том, что ваши файлы были успешно зафиксированы. Если вы не указали свое глобальное имя пользователя и адрес электронной почты во время выполнения упражнения вручную, вам будет предложено настроить их в диалоговом окне.

Получение и отправка

Теперь, когда первая фиксация уже сделана, мы готовы выполнить вторую половину процесса. Мы всегда делаем проверку, прежде чем отправлять, чтобы убедиться, что у нас есть последняя версия текущей ветки. Это дает нам шанс разрешить любые конфликты, которые могли возникнуть с момента нашей последней попытки. В Git мы использовали команду `pull`. В инструментах Git есть опция `pull`, но, прежде чем мы перейдем к ней, я укажу на другую опцию: **Update Project...** Ее можно найти в раскрывающемся списке на панели инструментов, обозначенном текущей ветвью, как показано на рис. 5.19.

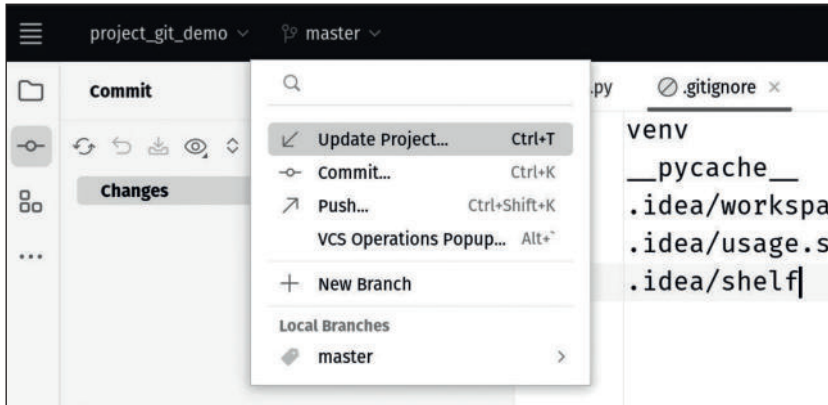


Рис. 5.19. Команда Update Project... на самом деле не является командой Git. Она относится к стратегии обновления

Когда вы нажмете **Update Project...**, вам будет предложено указать предпочтения для обновления локального репозитория Git. В появившемся диалоговом окне есть две возможности, как показано на рис. 5.20.

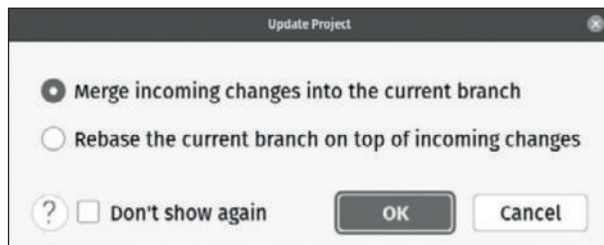


Рис. 5.20. Существует две стратегии обновления на выбор

Команда `git pull`, которую мы выполнили ранее, осуществляет первый вариант. Операция извлечения (fetch) будет *извлекать* любые изменения с удаленного компьютера и автоматически *объединять* эти изменения (merge) с нашей локальной копией. Операция `git rebase` представляет собой другую стратегию. Она меняет структуру вашего репозитория, чтобы показать более четкую временную шкалу изменений. Я не буду вдаваться в аргументы в пользу того, какую стратегию использовать, но включу ссылку в раздел «Дальнейшее чтение» этой главы, если вы захотите погрузиться в тему глубже.

Лично я предпочитаю первый вариант, поскольку он проще и безопаснее. Мне нужно только нажать кнопку **OK**. Я могу установить флажок **Don't show again**, если не хочу, чтобы меня запрашивали перед использованием. PyCharm всегда будет использовать выбор, который я делаю здесь и сейчас.

Если у вас есть проблемы с фиксацией, я не имею в виду общие проблемы, но технически, когда дело доходит до выполнения `git commit`, вы можете полностью пропустить этот диалог. Просто используйте пункт **Pull...** в меню Git, как показано на рис. 5.21.

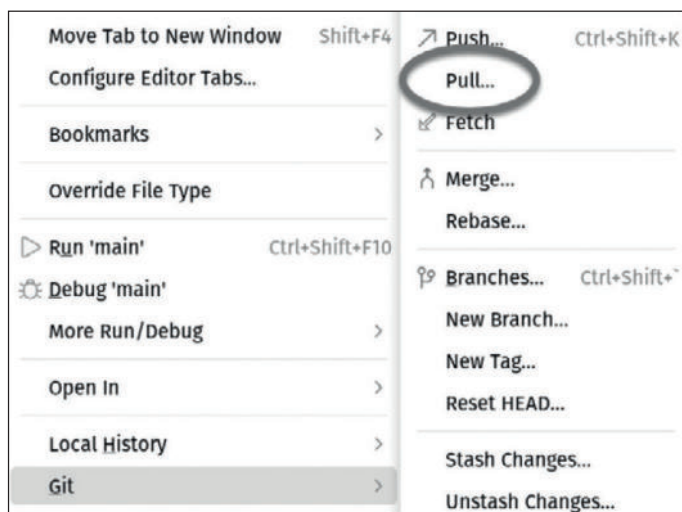


Рис. 5.21. Вы можете выполнить традиционное получение Git из любого меню Git

Ничего не остается, кроме как сделать отправку. Возможно, самый быстрый способ был показан ранее на рис. 5.19, где мы столкнулись с командой **Update Project...**

Ветвление и слияние

Мы обсудили самые основные функции любой системы контроля версий, которая защищает код проекта, отслеживая изменения с течением времени. Второй набор очень важных функций предполагает разделение проектной работы на несколько копий. Эта практика называется **ветвлением (branching)**. Ветвление имеет множество преимуществ; я не могу охватить их все здесь. Если вам нужна дополнительная информация о контроле версий с помощью Git, прочтите книгу Packt «Git для программистов», указанную в разделе «Дальнейшее чтение» этой главы. Тем не менее я предложу то, что считаю наиболее важными преимуществами.

Рассмотрим типичный веб-стартап, который проходит цикл создания репозитория и разработки до первого выпуска. Релиз находится на рабочем сервере где-то в облаке, и клиенты начали использовать приложение. Начинаящая компания не только хочет расширить свое предложение дополнительными функциями, но также сталкивается с ошибками и проблемами, которые необходимо исправить. Давайте представим, что команды разработки и обеспечения качества проделали настолько потрясающую работу, что в релизе практически нет ошибок. Довольная собой, компания хочет начать добавлять новые функции. Вся работа выполняется в основной ветке, потому что она единственная, которая существует. Чтобы добавить новую функцию, нам нужно изменить что-то глубоко внутри кода. С того момента, как вы начнете изменение, потребуется неделя, чтобы завершить его, и в течение этого времени разрабатываемая версия продукта будет очень нестабильной.

Внезапно мираж рассеивается. Практически невозможно выпустить версию программного обеспечения без ошибок, если, конечно, программное обеспечение не было написано единорогами или Чаком Норрисом. Скажем так, это не просто ошибка. Это плохая ошибка. Что-то вроде того, что кто-то забыл `WHERE customer_id=@customerId` в инструкции SQL, и ваше приложение ошибочно отображает все данные о клиентах всем. Это конец карьеры! Я видел, как это происходит, и это некрасиво. Нужно исправить это *немедленно!* За исключением того, что вы не сможете этого сделать, не отменив все внесенные вами изменения, которые могут сделать приложение нестабильным. Это один из вариантов. Другой вариант заключается в том, что вы можете вернуться к выпущенному вами коду, затем внедрить исправление и сохранить свою карьеру. Тогда придется потратить много времени, пытаться выбрать коммиты, чтобы спасти свою функциональную работу. Вы также можете написать весь свой код в Блокноте, используя клавиатуру, настроенную на раскладку Дворака¹, с полностью пустыми клавишами. Вы могли бы это сделать, но это не будет продуктивным использованием вашего времени.

Здесь в игру вступает ветвление. Стартап может выполнять всю свою работу в основной ветке. Мы не должны, но давайте ограничимся тем, что у нас есть на данный момент. Как только программное обеспечение будет выпущено, можно будет создать новую ветку и, возможно, назвать ее *веткой разработки*. Имя не важно. Операция ветвления создает копию того, что было выпущено. Именно в этой ветке наша бесстрашная команда разработчиков начинает вносить изменения, которые могут сделать версию разработки нестабильной.

Бум! Случилось страшное. Вы можете переключиться с *ветки разработки* обратно на *основную*. Теперь у вас есть выпущенный код, и вы можете решить проблему. После того как исправление будет установлено, можете объединить его с веткой разработки и продолжить работу над нестабильным приложением в *ветке разработки*. Разделение работы позволяет вам работать над новыми функциями независимо от срочного исправления ошибок. Я сильно упрощаю типичную стратегию ветвления, используемую большинством команд на этом примере. Я бы предпочел оставить освещение этой темы работам, посвященным контролю версий. Я оставляю предложения для дальнейшего чтения по стратегиям ветвления в разделе «Дальнейшее чтение» этой главы. Продолжим механику работы с ветками. Они жизненно важны для вашей повседневной практики разработки программного обеспечения.

Создание ветки

Создать новую ветку очень просто. Кликните меню ветки в верхней строке меню и выберите **New Branch**. Вы увидите диалоговое окно, показанное на рис. 5.22.

Введите имя для новой ветки. Ветка создается локально, и вы автоматически переключаетесь на новую ветку.

¹ Раскладка клавиатуры, запатентованная профессором Вашингтонского университета Августом Двораком и Вильямом Дилли в 1936 году для набора английских символов как альтернатива раскладке QWERTY, была разработана для устранения неэффективности и усталости после долгой работы – недостатков раскладки QWERTY. Дворак – англоязычный вариант чешской фамилии Дворжак. – *Прим. ред.*

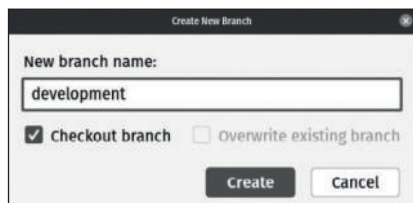


Рис. 5.22. Диалоговое окно Create New Branch позволяет создать новую ветку Git

Переключение между ветками

Это тоже очень легко. Вернитесь в меню ветки. Локальные ветки перечислены в раскрывающемся списке ветвей, как показано на рис. 5.23.

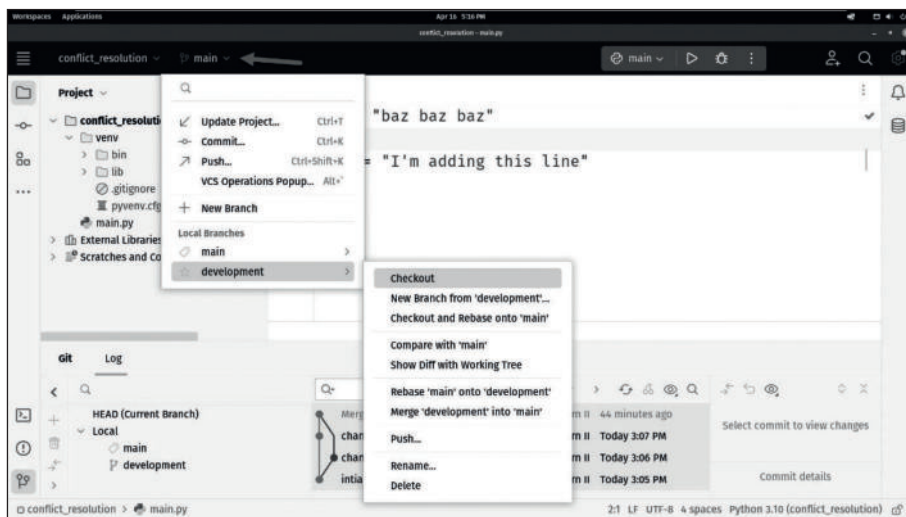


Рис. 5.23. Локальные ветки перечислены в меню ветвей

Для переключения на другую ветку достаточно кликнуть нужную ветку из списка и затем нажать **Checkout**. Как видите, над веткой можно выполнять и другие операции.

Слияние

Когда вы будете готовы объединить ветку обратно с основной или любой другой веткой, просто переключитесь на эту ветку, а затем используйте команду **Merge** в меню ветки, как показано на рис. 5.24.

Диалоговое окно **Merge**, показанное на рис. 5.24, позволит вам выбрать ветвь, которую вы хотите объединить с текущей выбранной ветвью.

Если вам нужно что-то более экзотическое, чем стандартная операция слияния, есть раскрывающийся список **Modify options**, который позволяет выполнить действие, эквивалентное нескольким переключателям командной строки, обычно используемым во время процесса слияния вручную.

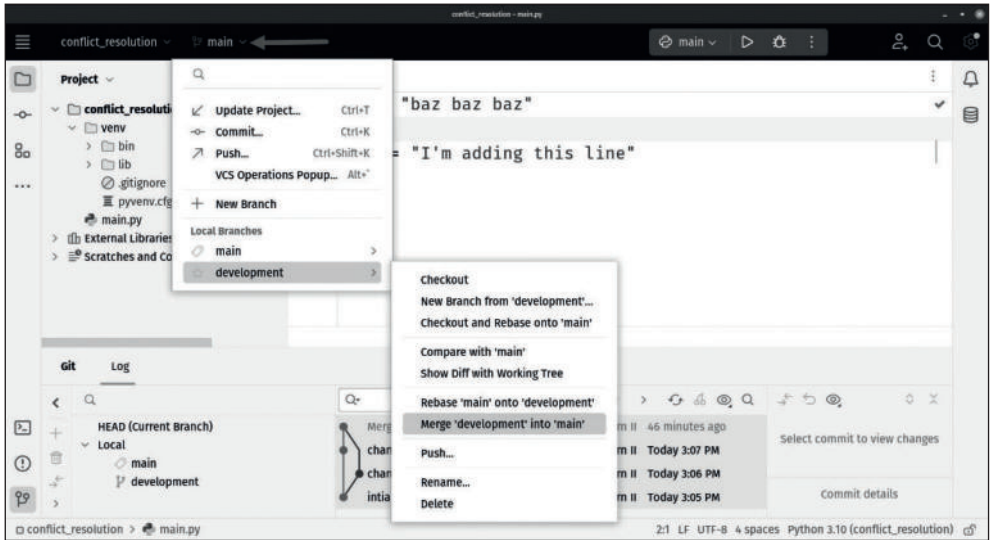


Рис. 5.24. Команду Merge можно найти в меню ветки

Просмотр диаграммы ветвей

Инструмент Git в окне инструментов обеспечивает графическое представление различных ветвей вашего репозитория. Вы можете увидеть это на рис. 5.25. Это может быть полезно, когда вам нужно просмотреть, что недавно изменилось в репозитории. Это то, что вы можете сделать первым делом утром, чтобы просмотреть изменения в вашей команде за последние 24 ч.

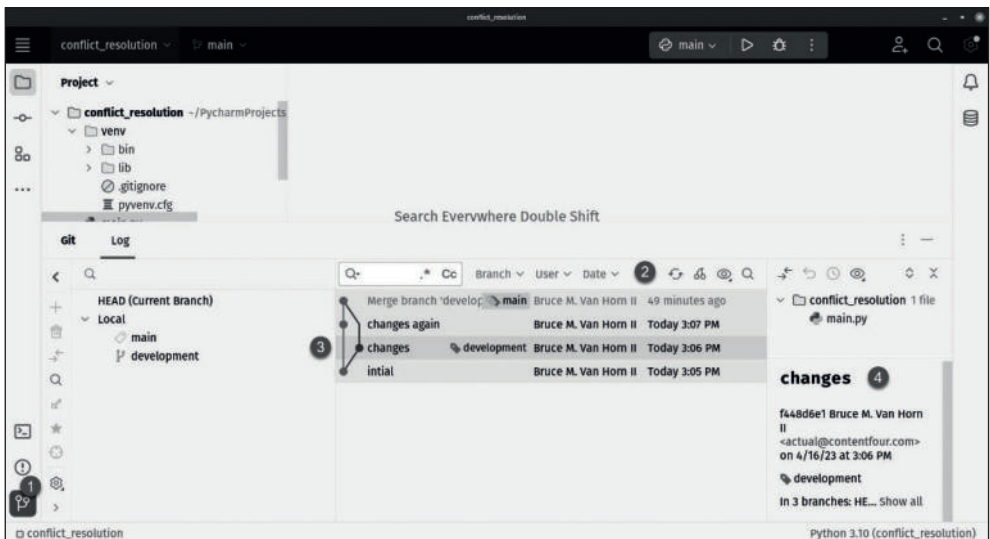


Рис. 5.25. Инструмент Git можно активировать, кликнув панель инструментов в левом нижнем углу окна PyCharm

Чтобы активировать его, кликните значок Git в окне инструмента (1). Вы можете искать историю изменений и фильтровать ее, используя различные инструменты на панели инструментов (2). Выбор коммита (3) позволяет вам увидеть его детали (4) вместе с сообщением о фиксации.

РАЗЛИЧИЯ И РАЗРЕШЕНИЕ КОНФЛИКТОВ

Рано или поздно вам придется выполнить запрос (pull) перед отправкой некоторых изменений только для того, чтобы обнаружить, что существует конфликт. Это может пугать и вызывать стресс даже у опытных разработчиков, поскольку вы рискуете нарушить чей-то недавний вклад. Тем не менее такое случается, и вам нужен способ справиться с этой проблемой. Это подводит нас к одной из моих любимых функций PyCharm. В главе 1 я рассказал вам о своем опыте работы с отладчиком Microsoft. Поскольку я освоил это в начале своей карьеры, мне больше ничего не надо. Я так же отношусь к инструменту merge в PyCharm. Разрешение конфликтного слияния в чем-либо другом кажется болезненным.

В проекте только один файл. К настоящему моменту вы знаете это как файл `main.py`, который PyCharm генерирует с новыми проектами. Мы собираемся создать конфликт, и нужно выполнить следующие шаги:

- создайте новый проект в PyCharm,
- измените содержимое файла `main.py` на одну строку кода, чтобы гарантировать возникновение конфликта,
- инициализируйте репозиторий Git в папке проекта,
- добавьте и зафиксируйте все файлы проекта,
- создайте новую ветку под названием *development*,
- измените строку кода, которая отображает сообщение «Hello, World»,
- зафиксируйте изменение в ветке,
- вернитесь в основную ветку,
- внесите другое изменение в одну строку кода,
- зафиксируйте изменение,
- теперь попробуйте объединить ветку разработки с основной веткой.

Результатом этих 11 шагов станет конфликт. Давайте его разрешим!

Сначала создайте в PyCharm новый проект с именем `conflict_resolution`, как показано на рис. 5.26. Убедитесь, что вы создаете проект в месте на вашем диске вне структуры папок хранилища книги. Помните, код книги уже находится в репозитории. Вы не можете создать новый репозиторий внутри другого репозитория.

PyCharm генерирует проект с файлом `main.py`. Удалите все строки в `main.py` и замените их следующим кодом:

```
foo = «foo»
```

Для ясности: ваше окно PyCharm должно выглядеть так же, как мое на рис. 5.27.

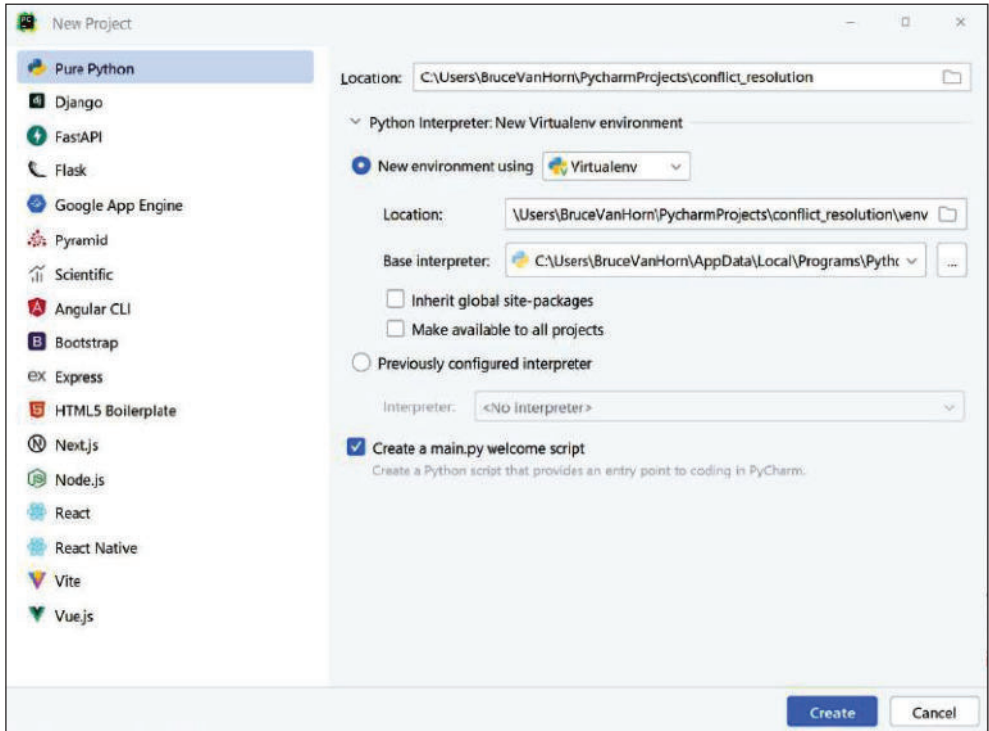


Рис. 5.26. Настройки моего проекта conflict_resolution по умолчанию



Рис. 5.27. Файл main.py уменьшен до одной строки кода

Откройте меню **VCS** в меню **File** и выберите **Create Git repository**. Если вы не помните, где это находится в меню, вернитесь к рис. 5.9. Запишите созданную ветку по умолчанию. Для этого упражнения нам не понадобится удаленный доступ, поэтому не имеет значения, называется ли она *main*, или *master*,

или как-то еще; вам просто нужно запомнить ее название. Я предполагаю, что она называется *main*.

Используйте окно `commit`, чтобы добавить файлы проекта в репозиторий. Рисунок 5.3 напомнит вам, если вы забыли, как это сделать. Продолжайте и зафиксируйте файлы, введя сообщение о коммите.

Далее нам нужно создать ветку. Кликните раскрывающийся список ветки и выберите **Create branch**. На рис. 5.4 показано расположение меню, если вам нужно обновить информацию. Я вызову свою ветку `development`.

Откройте файл `main.py`. Замените код одной строкой:

```
foo = "bar"
```

Ограничивая себя одной линией, мы можем быть уверены, что наши действия приведут к конфликту. Зафиксируйте это изменение в ветке `development`.

Вернитесь к основной ветке, используя раскрывающийся список ветвей. Измените содержимое `main.py` на этот код:

```
foo = "baz baz baz"
```

Зафиксируйте это изменение в основной ветке. Вы только что смоделировали, как два разработчика изменяют один и тот же файл в одной и той же области. У Git не будет возможности примирить различия. Когда основная ветка активна, кликните раскрывающийся список ветки, кликните ветку разработки и нажмите **Merge development into main**. Вы увидите сообщение о конфликте, как показано на рис. 5.28.

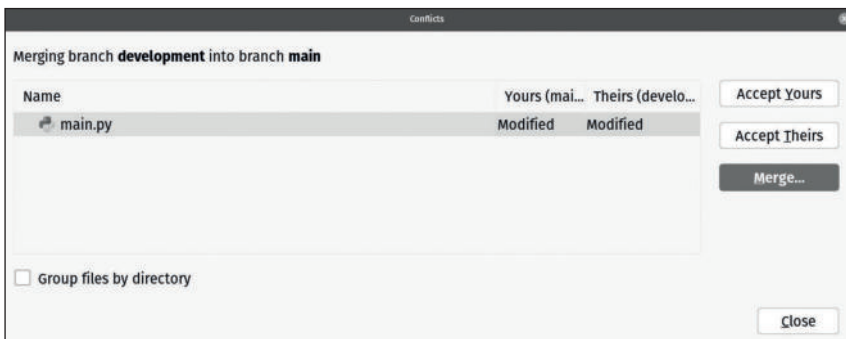


Рис. 5.28. У нас есть конфликт, который необходимо разрешить, прежде чем мы сможем сделать слияние

Обычно у вас есть три варианта разрешения конфликта. Вы можете игнорировать входящие изменения и выбрать свою локальную работу в качестве правильного кода, нажав кнопку **Accept Yours**. Аналогичным образом вы можете выбрать входящие изменения кода и отказаться от локальной копии, нажав **Accept Theirs**. Однако часто вам придется объединить части вашей локальной версии с поступающими изменениями. Для этого нужно будет активировать инструмент слияния, как показано на рис. 5.29, нажав кнопку **Merge....**

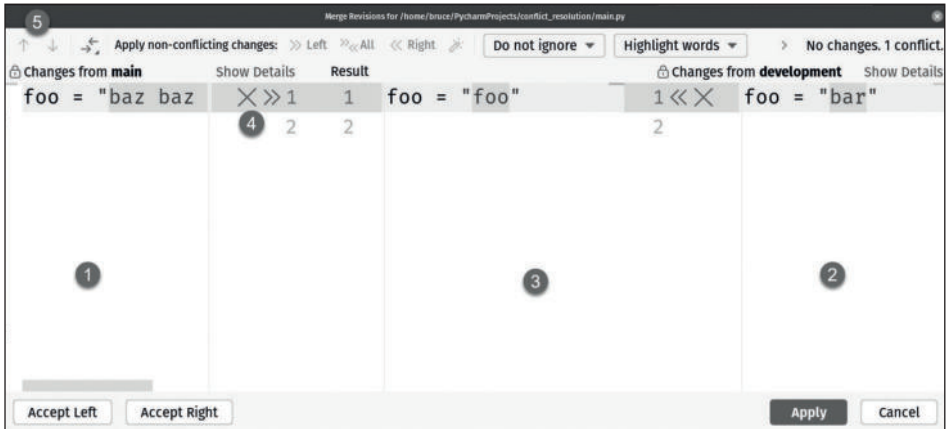


Рис. 5.29. Активируйте инструмент merge, чтобы выбрать части входящего кода для слияния с вашим

Если инструмент merge активен, вы увидите три панели. Панель слева (1) представляет конфликтующий код из основной ветки. Код справа (2) – это ревизия из ветки разработки. Панель посередине (3) представляет собой слияние этих двух. Вы можете выбирать части своего кода и части входящего кода, чтобы составить наилучшую возможную комбинацию. Поэкспериментируйте с кнопками >> (4), которые копируют код в центральный объединенный результат. Кнопки с надписью X (4) будут игнорировать конфликтующую строку. Вы можете использовать стрелки вверх и вниз (5), чтобы перейти к следующему неразрешенному конфликту в файле.

Давайте представим, что лучшее решение моего конфликта – это объединить строку из левой панели и строку из правой панели. В реальной жизни это не работает, потому что мы получим два присвоения переменных рядом друг с другом, но это демонстрирует, как работает инструмент. Чтобы использовать строку из левой панели, я могу нажать кнопку, обозначенную стрелкой на рис. 5.30.

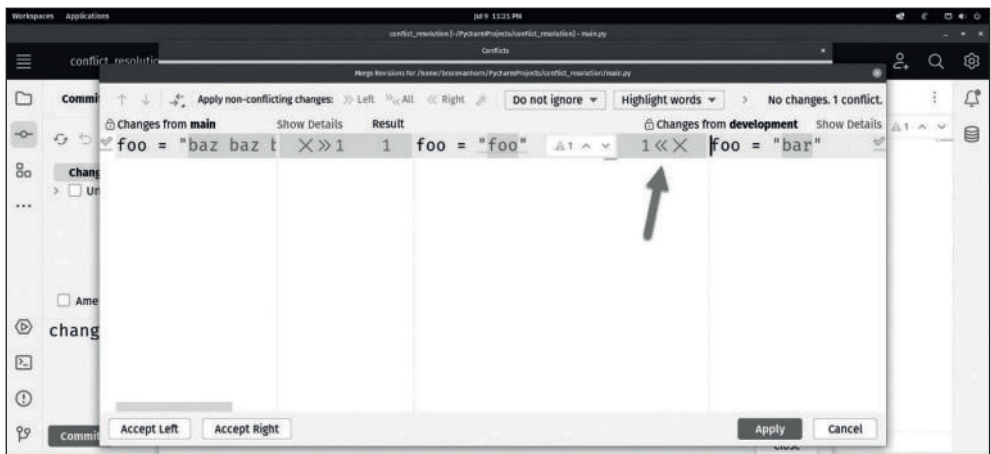


Рис. 5.30. Эта кнопка переместит код из левой панели в рабочее решение слияния

Далее я хотел бы поместить строку из правой панели под строку, которую я только что переместил. Зачастую вам хочется использовать часть кода с обеих сторон, а не просто считать его правильным или неправильным с одной стороны. Нажмите кнопку, указанную стрелкой на рис. 5.31, чтобы скопировать код с левой панели под текущей строкой в средней панели.

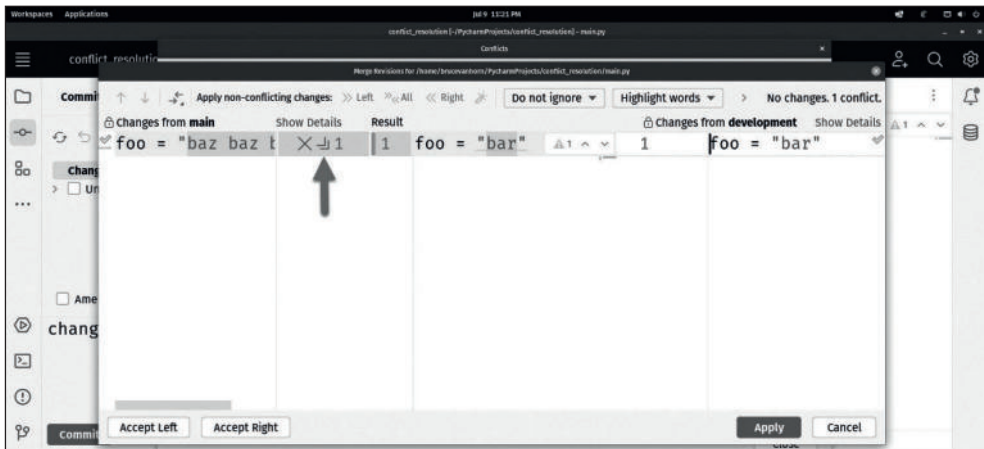


Рис. 5.31. Щелчок по двойной стрелке копирует эту линию под ту, которую мы выбрали справа

Выполнив это действие, вы разрешили конфликт! Вы будете вознаграждены красивым зеленым сообщением, сообщающим вам об этом.

Ваша цель – использовать левую и правую панели для создания лучшей версии кода в средней панели, которая представляет собой разрешение конфликта. Когда закончите, вы можете нажать кнопку **Apply**. Если в файле остались неразрешенные конфликты, PyCharm сообщит вам об этом. Если вы получили их все, слияние, которое вы попытались выполнить, завершено. Если вы работаете с удаленным компьютером, следует отправить результирующее разрешение конфликта после успешной фиксации.

Просмотр различий

Вам не нужен конфликт, чтобы использовать окно сравнения, которое мы только что видели. Окно различий – это окно, в котором рядом отображаются две (или более) версии кода. Вы можете различать ветки или версии. Допустим, я быстро вношу изменения в наш объединенный код, добавив следующую строку:

```
text = "I'm adding this line"
```

Я также удаляю строку 2, которая гласит:

```
text = "I'm adding this line"
```

Мой код теперь совсем другой. Если я хочу сравнить свой новый код с последним кодом ветки, я могу кликнуть правой кнопкой мыши вкладку `main`.

ру и выбрать **Git | Show Diff**. Я увижу разницу между двумя файлами, как показано на рис. 5.32.



Рис. 5.32. Представление различий между тем, что у вас есть в редакторе, и последней зафиксированной версией

Это различие отображается в параллельном формате. Вы можете изменить его на встроенное представление, которое часто отображается на GitHub при просмотре коммитов, изменив настройку раскрывающегося списка с меткой (1). Вы также можете редактировать файл, используя те же инструменты перемещения строк, которые вы только что видели при слиянии, обозначенном (2).

КРАТКОЕ СОДЕРЖАНИЕ

В этой главе мы рассмотрели две основные темы – идею контроля версий при разработке и программировании приложений и ее важность, а также способы ее реализации с использованием Git и GitHub в PyCharm. В частности, мы научились осуществлять контроль версий с помощью Git и GitHub двумя разными способами: вручную и с помощью PyCharm.

Обладая этими знаниями, пользователи PyCharm могут гибко применять контроль версий к своим собственным проектам, пропуская утомительный ручной процесс в терминале или командной строке. Мы видим, что, предлагая эти функции, PyCharm позволяет нам сосредоточиться на реальном процессе решения задачи в любом конкретном проекте разработки программного обеспечения.

Помимо контроля версий, существуют и другие методы разработки приложений, для облегчения которых PyCharm предоставляет интуитивно понятные и простые команды. Без этих команд разработка приложений была бы довольно сложной и пугающей. Этими процессами являются тестирование, отладка и профилирование, и все они будут обсуждаться в следующей главе.

Вопросы

1. Что включает в себя термин «*контроль версий*», особенно в контексте программирования?
2. Каковы преимущества использования контроля версий?
3. Каковы основные шаги по контролю версий ваших собственных проектов с помощью Git и GitHub?
4. Что такое *ветвление* и почему оно важно?
5. Назовите различные окна и места инструментов в PyCharm, которые дают вам доступ к Git и другим командам VCS.

ДАЛЬНЕЙШЕЕ ЧТЕНИЕ

- *Merging vs. Rebasing*: <https://www.atlassian.com/git/tutorials/merging-vs-rebasing>.
- How Secure Shell (SSH) works: https://en.wikipedia.org/wiki/Secure_Shell.
- *Why GitHub renamed its master branch to main*: <https://www.theserverside.com/feature/Why-GitHub-renamed-its-master-branch-to-main>.
- *Git Merge Strategy Options and Examples*: <https://www.atlassian.com/git/tutorials/using-branches/merge-strategy>.
- Git branching guidance: <https://learn.microsoft.com/en-us/azure/devops/repos/git/git-branching-guidance>.
- Git Essentials for Beginners: <https://www.packtpub.com/product/mastering-git/9781783553754>.
- Liberty, J. (2021). *Git for Programmers: Master Git for effective implementation of version control for your programming projects*. Packt Publishing Limited.
- Narebski, J. (2016). *Mastering Git*. Packt Publishing Ltd.
- *The Joel Test: 12 Steps to Better Code*: <https://www.joelonsoftware.com/2000/08/09/the-joel-test-12-steps-to-better-code/>.

Бесшовное тестирование, отладка и профилирование

В главе 5 я говорил о *тесте Джозла*. Этот тест представляет собой всего лишь список лучших практик. На вершине списка находится использование контроля версий, которому была посвящена предыдущая глава. Если вы просмотрели список, вы, вероятно, не удивились, увидев, что в списке также есть тестирование. Формализованные методы тестирования программного обеспечения, такие как **разработка через тестирование (TDD)** и **разработка через поведение (BDD)**, являются краеугольными камнями контроля качества программного обеспечения. Работа с этими методологиями помогает создавать программное обеспечение, которое с меньшей вероятностью выйдет из строя в рабочей среде. Если все сделано правильно, это также имеет побочные преимущества, такие как предотвращение расширения объема и возможность эффективного рефакторинга проектов, которые могли пренебречь лучшими практиками и взять на себя большой технический долг.

Сегодня на практике существует несколько уровней тестирования, в том числе следующие:

- **модульное тестирование**, цель которого – протестировать базовую низкоуровневую функциональность на уровне функций или классов;
- **интеграционное тестирование**, цель которого – проверить, как компоненты в более крупной системе работают вместе;
- **тестирование пользовательского интерфейса**, целью которого является проверка работы интерактивных элементов системы;
- **сквозное тестирование**, при котором тестируется вся система в производственной среде.

Как и все известные языки программирования, Python имеет богатый набор библиотек тестирования. А поскольку Python работает «на приложенных батарейках», в стандартную библиотеку встроено несколько прекрасных инструментов тестирования. Естественно, сторонние решения развивались и доступны через [PyPi.org](https://pypi.org).

Я освоил Java-библиотеку **JUnit**, а позже и ее порт .NET под названием **NUnit**. Я обнаружил, что это делает разработку программного обеспечения очень увлекательной. Есть что-то забавное в том, чтобы начинать свой день с набора тестов, которые не проходят, и в течение дня писать код, чтобы каждый из них прошел. Если вы дисциплинированы, вы напишете необходимый минимум кода и постепенно увидите прогресс, поскольку ваш инструмент тестирования меняет цвет с красного на зеленый. Вы не должны использовать ярлыки и не должны поддаваться искушению написать функциональность, которая кажется крутой, но которая может вам не понадобиться позже. Когда несколько лет назад я перешел на Python, я был рад увидеть так много возможностей для тестирования библиотек и фреймворков. Мне также было приятно видеть, что PyCharm поддерживает большинство популярных прямо в IDE.

В этой главе мы рассмотрим создание модульных тестов в коде Python, следуя принципам TDD. В TDD вы обычно создаете набор тестов, призванных доказать, что ваше программное обеспечение соответствует набору требований. Эти тесты пишутся до того, как вы создадите какую-либо функциональность в своей программе, и начинаются с неудачных попыток. Ваша задача – обеспечить прохождение тестов с использованием максимально простого кода.

Попутно вам придется использовать отладчик для пошагового выполнения проблемного кода, который либо необъяснимым образом дает сбой, либо, что еще хуже, необъяснимым образом работает. Как только ваш код заработает и пройдет тесты, вам обычно нужно учитывать скорость выполнения. Национальная служба здравоохранения (NHS) Великобритании разработала алгоритм, который сопоставляет донорские органы пациентам в своей системе. Сложный алгоритм должен был быть быстрым, поскольку существует ограниченный период времени, в течение которого извлеченный орган пригоден для трансплантации. Подобные ограничения по времени существуют и во многих других типах приложений. Нам, разработчикам, нужны инструменты, которые помогут нам выявить узкие места эффективности.

В этой главе будут рассмотрены следующие темы:

- модульное тестирование на Python с помощью PyCharm,
- использование мощного визуального отладчика PyCharm,
- работа с инструментами профилирования PyCharm для поиска узких мест в производительности вашего кода.

ТЕХНИЧЕСКИЕ ТРЕБОВАНИЯ

Ниже приведены предварительные технические условия для этой главы:

- рабочая установка Python 3.10 или новее,
- рабочая установка PyCharm,
- пример кода для этой главы можно найти по адресу <https://github.com/PacktPublishing/Hands-On-Application-Development-with-PyCharm---Second-Edition/tree/main/chapter-06>.

ТЕСТИРОВАНИЕ, ТЕСТИРОВАНИЕ, 1-2-3

Модульное тестирование – это практика, призванная доказать, что ваш код работает так, как задумано. Хороший набор тестов будет соответствовать функ-

циональной спецификации. Для этого подойдет большой набор тестов, а также учтет любые очевидные пути сбоя. Для начала давайте разберемся с чем-то простым: вашим банковским счетом. Хорошо, он не обязательно должен быть вашим. Рассмотрим типичную транзакцию, когда вы покупаете что-то в магазине с помощью своей банковской карты.

Вы посещаете свой любимый книжный магазин, чтобы приобрести очередную превосходную книгу в области разработки программного обеспечения. Допустим, вы нашли копию моей первой книги «Практическая реализация шаблонов проектирования C#», опубликованной издательством Packt. Учитывая ее статус настоящей классики, вы не сможете устоять перед покупкой за любую цену. Вы подносите свою карту к кассовой системе книжного магазина, и происходят две вещи:

- 1) эквивалент 39,95 долл. США – что, кстати, настоящий грабеж – снимается с вашего банковского счета,
- 2) такая же сумма переводится на банковский счет книжного магазина.

Это транзакционная операция. Формально говоря, транзакция – это многоэтапная операция, каждый шаг которой должен завершиться без ошибок. Это должен быть набор операций по принципу «все или ничего». Если первый шаг завершится, а второй не удастся, то 39,95 долл. просто исчезнут с вашего банковского счета, и вы не сможете пойти домой со своей книгой. Если второй шаг срабатывает, а первый окажется неудачным, вы получите бесплатную книгу, но местный книготорговец разорится. Нам нужно, чтобы оба шага завершились или, в худшем случае, полностью провалились, чтобы деньги не перешли из рук в руки.

Этот уровень критичности – хороший скрипт для изучения модульного тестирования.

Модульное тестирование на Python с использованием PyCharm

Создайте новый проект в PyCharm, используя простой шаблон проекта Python. Назовем его `bank_account`. Завершенный пример вы найдете в репозитории исходного кода этой главы, но, если вы хотите попрактиковаться в создании и тестировании необходимого кода, просто следуйте инструкциям.

PyCharm создал файл с именем `main.py`. Мы воспользуемся им позже, но давайте вынесем код нашей банковской транзакции в отдельный модуль. Одним из принципов написания хорошего кода является написание **тестируемого кода**, а лучший способ написать тестируемый код – следовать **принципу единственной ответственности (SRP)**, когда вы создаете единицы кода, которые несут только одну ответственность. SRP является частью более широкого набора правил для создания устойчивой архитектуры кодирования под названием **SOLID**, что является аббревиатурой следующих принципов:

- принципа единственной ответственности (SRP),
- принципа открытости-закрытости (OCP),
- принципа подстановки Лискова (LSP),
- принципа разделения интерфейса (ISP),
- принципа инверсии зависимостей (DIP).

SOLID обычно учитывается при разработке **полностью объектно ориентированной (FOO)** архитектуры с использованием статических языков, которые строго объектно ориентированы. Java, C++ и C# – классические примеры таких языков. Python допускает множество различных парадигм разработки, и его реализация **объектно ориентированного программирования (ООП)** не такая полная или, возможно, не такая традиционная, как во многих других. Если вы никогда не слышали о SOLID как о разработчике Python, возможно, именно поэтому. Существуют книги и блоги, где люди пытались адаптировать код Python под себя, но, на мой взгляд, это часто кажется вынужденным.

SRP – это то, чему вы должны обязательно следовать. Он вписывается в любой язык и любую парадигму. Проще говоря, создаваемые вами элементы, будь то функции, пакеты Python или объекты, должны делать только одно, и делать это хорошо. Разделив обязанности вашего кода, вы можете создавать повторно используемые элементы, которые можно легко тестировать и, следовательно, легко обслуживать. Все должно делать что-то одно. Конечно, будет что-то, связывающее все это вместе, – возможно, функция `main` в программе, единственная цель которой – вызывать все остальное и обеспечить общий поток вашей программы.

ОСР утверждает, что после того, как вы отправили класс в производство, вам никогда не следует его менять. Вы должны писать свой код таким образом, чтобы ваши классы были открыты для расширения, но закрыты для модификации. Этот принцип предназначен для защиты уже протестированных и выпущенных вами функций. Если вы откроете класс и измените его, вы рискуете получить ошибки, и придется повторно тестировать всю программу. Если вы ограничиваете свои изменения расширением, вам нужно беспокоиться только о тестировании расширения.

LSP нелегко перевести на Python. В нем говорится, что любой подкласс должен иметь возможность заменить свой суперкласс, не влияя на корректность программы. Другими словами, если программа использует базовый класс, она должна работать правильно, если вы замените базовый класс производным классом. Придерживаясь LSP, вы продвигаете концепцию полиморфизма в своих классах. Это позволяет единообразно обрабатывать различные объекты через их общий супертип, что приводит к более гибким и модульным конструкциям. Реализация LSP сложна в динамических языках, таких как Python, поскольку эти языки допускают динамическую типизацию и позднее связывание вызовов методов. По этой причине LSP даже более важен, чем в статических, строго типизированных языках. Проблема связана с отсутствием строгой проверки типа компиляции, которая имеется в C#, C++ или Java. Любые ошибки проектирования, которые вы допустите, не всплывут до начала выполнения. Как разработчик Python, вы должны очень тщательно проектировать и тестировать свои программы с большей тщательностью, чем на других языках.

ISP утверждает, что классы или модули должны иметь интерфейсы, адаптированные к их конкретным потребностям. Интерфейс, определяющий структуру и поведение класса, не должен содержать ничего, что не требуется этому классу. Это не очень хорошо переводится на Python, поскольку в Python отсутствуют традиционные интерфейсы, имеющиеся в таких языках, как Java и C#. Слово интерфейс можно понимать как обычный суперкласс, и в этом случае

суперкласс не должен содержать свойства и методы, которые никогда не используются внутри подкласса.

DIP – это фундаментальный принцип объектно ориентированного программирования, который касается зависимостей между классами и модулями. В нем говорится, что модули высокого уровня не должны зависеть от модулей низкого уровня, но оба должны зависеть от абстракций. Кроме того, он подчеркивает, что абстракции не должны зависеть от деталей; скорее, детали должны зависеть от абстракций¹.

Вот ключевые идеи DIP.

- **Модули высокого уровня не должны зависеть от модулей низкого уровня:** модули высокого уровня представляют логику или функциональность приложения более высокого уровня, тогда как модули низкого уровня имеют дело с деталями реализации и операциями более низкого уровня. Согласно DIP, модули высокого уровня не должны напрямую зависеть от модулей низкого уровня. Вместо этого оба должны зависеть от абстракций.
- **Абстракции не должны зависеть от деталей:** абстракции, такие как интерфейсы или абстрактные классы, определяют контракты, которые определяют поведение и функциональность, ожидаемую от взаимодействующих объектов. DIP утверждает, что эти абстракции не должны зависеть от конкретных деталей реализации модулей нижнего уровня. Этот принцип продвигает идею программирования для интерфейсов, а не конкретных реализаций.

Чтобы придерживаться DIP, важно вводить абстракции, такие как интерфейсы или абстрактные классы, и программировать на основе этих абстракций, а не конкретных реализаций. Это способствует слабой связи и обеспечивает большую гибкость и удобство сопровождения базы кода. Убедитесь, что вы не путаете это с **внедрением зависимостей (DI)**. Они связаны, но это не одно и то же.

DI – это шаблон или метод проектирования, который облегчает реализацию DIP. DI – это способ предоставить зависимости, необходимые классу, из внешнего источника, вместо того чтобы класс создавал свои зависимости или управлял ими внутри себя.

В DI ответственность за создание и предоставление зависимостей делегируется внешнему объекту, обычно называемому «инжектором» или «контейнером». Контейнер отвечает за создание экземпляров классов и внедрение их зависимостей. Это обеспечивает лучшую развязку, гибкость и упрощает тестирование, поскольку зависимости можно легко заменить или имитировать во время модульного тестирования.

DI можно рассматривать как стратегию реализации принципов, изложенных в DIP. Это помогает придерживаться DIP, предоставляя механизм, который инвертирует контроль над зависимостями и отделяет создание объектов от их использования.

Подводя итог, DIP – это руководство по проектированию модульных, слабосвязанных систем, а DI – это метод или шаблон, используемый для реализации

¹ Абстракция в программировании – это способ снизить сложность и повысить эффективность проектирования и реализации программного обеспечения за счет сокрытия технической сложности за более простым API. – *Прим. ред.*

DIP, который осуществляется путем передачи ответственности за управление зависимостями.

Если вы заинтересованы в такого рода архитектуре, вам следует ознакомиться с той книгой, которую я упомянул ранее, поскольку SOLID подробно описан там, хотя и с использованием в качестве языка C#. Однако SRP прекрасно сочетается с любым языком или парадигмой, включая Python.

Когда вы придерживаетесь функций и классов, которые делают только одну вещь и делают это хорошо, их тестирование становится проще простого, поскольку функциональность изолирована. Функции или классы, которые пытаются сделать слишком много, труднее тестировать из-за взаимодействия между зависимостями. Давайте создадим что-нибудь, чтобы прояснить это.

Выбор тестовой библиотеки

Для Python 3 доступно несколько популярных библиотек модульного тестирования. Некоторые из наиболее широко используемых фреймворков включают следующие:

- `unittest`: это встроенная среда модульного тестирования Python. Она предоставляет набор классов и методов для написания и запуска тестов. Среда `unittest` следует стилю модульного тестирования `JUnit` и предлагает такие функции, как обнаружение тестов, тестовые приспособления и методы утверждения;
- `pytest` – это популярная, многофункциональная среда тестирования, которая обеспечивает более краткий и выразительный способ написания тестов по сравнению с `unittest`. Она поддерживает обнаружение тестов, фикстуры, параметризованные тесты и мощные методы утверждения¹. `pytest` известна своей простотой и гибкостью;
- `nose` – еще одна популярная среда тестирования, расширяющая возможности `unittest`. Она предоставляет дополнительные функции, такие как автоматическое обнаружение тестов, генераторы тестов, плагины, а также расширенные параметры выбора и фильтрации тестов. Хотя `nose` широко используется, ее популярность в последние годы снизилась в пользу `pytest`;
- `doctest` – это уникальная среда тестирования, которая позволяет писать тесты в виде интерактивных примеров в строках документации или комментариях к документации. Она извлекает и выполняет примеры в качестве тестов, проверяя, соответствует ли фактический результат ожидаемому результату. `doctest` хорошо подходит для тестирования документации и примеров кода.

Это всего лишь несколько примеров популярных библиотек модульного тестирования в Python. Каждая библиотека имеет свои особенности, стиль и сильные стороны, поэтому стоит изучить их, чтобы найти ту, которая лучше всего соответствует требованиям вашего проекта и вашим личным предпочтениям. Особенность работы с PyCharm заключается в том, что она поддерживает все эти библиотеки тестирования, а пользовательский интерфейс для запуска тестов и просмотра результатов всегда один и тот же.

¹ `Assert` – это метод для определения статуса, «пройден» или «не пройден» тестовый пример. – Прим. ред.

Поскольку это книга о PyCharm, а не описание платформ тестирования, я буду использовать библиотеку `unittest`, которая является частью стандартной библиотеки Python. Это защитит наш пример кода от внешних зависимостей.

Добавление класса банковского счета

Кликните правой кнопкой мыши заголовок проекта в окне проекта и выберите **New | Python File**. Назовите файл `bank_account.py`.

Затем добавьте следующий код:

```
class BankAccount:
    def __init__(self, name: str, account_number: str, \
        balance: float):
        self.name = name
        self.account_number = account_number
        self.balance = balance
```

На данный момент мы создали класс `BankAccount`, создали конструктор и инициализировали три переменные-члена: `name`, `account number`, и `balance`. Далее мы добавим метод, предназначенный для списания денег, но только если сумма меньше баланса:

```
def withdraw(self, amount: float) -> None:
    new_balance = self.balance - amount
    if new_balance > 0:
        self.balance = new_balance
    else:
        raise ValueError("Account overdrawn!")
```

Если снятая сумма превышает `balance`, мы выдаем сообщение о ошибке `ValueError` и сообщение "Account overdrawn!" («Превышен допустимый лимит списания средств»). Далее, нам нужен метод пополнения счета. Это должно быть положительное число; в противном случае мы будем осуществлять списание средств, а не зачисление на депозит:

```
def deposit(self, amount: float):
    if amount > 0:
        self.balance += amount
    else:
        raise ValueError("Deposit amount must be greater than 0.")
```

Пока все хорошо, да? Поскольку наши методы содержат некоторую бизнес-логику, следует создать для них модульный тест.

Тестирование класса банковского счета

Кликните правой кнопкой мыши заголовок проекта в окне проекта и выберите **New | Python File**, но на этот раз сделайте его модульным тестом Python, как показано на рис. 6.1:

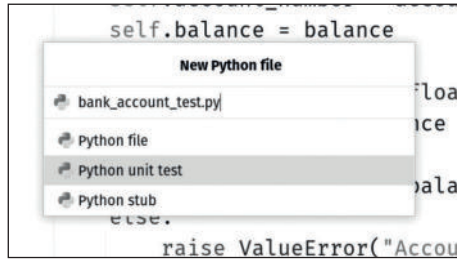


Рис. 6.1. Существует несколько шаблонов для нового файла Python

Существует несколько соглашений по работе с тестовыми файлами. Некоторые считают, что иметь папку, содержащую только тесты, – это хорошая идея. Другие считают, что тестовый файл должен находиться рядом с тестируемым файлом. Мне нравится это соглашение, потому что оно позволяет мне легко увидеть, какие файлы в моем проекте не проверены. По аналогичному соглашению я назову файл `back_account_test.py`. Соглашения требуют, чтобы я либо начинал, либо заканчивал имя моего файла словом *test*. Я поместил его в конец, потому что в противном случае тестовый файл не будет находиться рядом с файлом, который он тестирует в проводнике.

PyCharm создает файл, но он не пустой. Код в файле выглядит так:

```
import unittest
class MyTestCase(unittest.TestCase):
    def test_something(self):
        self.assertEqual(True, False) # add assertion here
if __name__ == '__main__':
    unittest.main()
```

Среда IDE предоставила нам шаблон, содержащий класс тестирования, который наследуется от встроенного фреймворка модульного тестирования Python. Среда просто, но незамысловато названа `unittest`. Шаблон содержит необходимый импорт в верхней части файла, класс тестирования, один метод тестирования и блок `dundermain`, который позволяет скрипту запускаться автономно. Чтобы тест заработал, вам необходимо изменить этот файл. Начните с добавления импорта в файл, содержащий класс, который вы хотите протестировать.

```
import unittest
from bank_account import BankAccount
```

Затем измените имя класса на `BankAccountTestCase`. Затем полностью удалите метод `test_Something(self)` и замените его следующим:

```
def test_init(self):
    test_account = BankAccount("Bruce Van Horn", \
        "123355-23434", 4000)
    self.assertEqual(test_account.name, "Bruce Van Horn")
    self.assertEqual(test_account.account_number, \
        "123355-23434")
    self.assertEqual(test_account.balance, 4000)
```

Честно говоря, это какой-то смешной тест, потому что логика конструктора чрезвычайно проста. Даже у меня возникнет соблазн пропустить это. Однако это не всегда так. Если вы делаете что-то сложное в конструкторе, следует провести модульное тестирование. Здесь пример служит для того, чтобы побудить нас двигаться дальше. Все, что мы здесь сделали, – это создали новый экземпляр класса `BankAccount` и передали ему имя, номер счета и начальный баланс. Затем мы использовали метод `AssertEqual` класса `unittest`, чтобы проверить каждую из переменных-членов и убедиться, что они установлены правильно. Вряд ли это было так, если только вы не допустили ошибку, в этом и есть суть.

Остерегайтесь надоедливых самонаборов `self`:

Прочитав целую главу о чудесах автозаполнения, я должен признать, что иногда это может раздражать. Это – как-раз тот случай. В тот момент, когда вы вводите открывающую скобку вашего тестового метода, PyCharm заполняет слово `self` вместе с закрывающей скобкой и двоеточием в конце строки. Поскольку я печатаю быстро, мне часто приходилось получать что-то вроде `test_init(selfself):`, прежде чем я понимал, что произошло. Я научил свою правую руку находить клавишу *End* на клавиатуре, как только я нажимаю на открывающую скобку. Это перенесет вас в конец автоматически заполняемой строки. Нажмите *Enter*, и вы окажетесь там, где хотите.

Мы добавим еще два теста ниже первого. Первый тест проверит метод `withdraw`. Введите новый метод под первым тестом, но над строкой с тестом `dunder-main`:

```
def test_withdraw(self):
    self.fail()
```

Введите тест для депозита ниже метода `test_withdraw`:

```
def test_deposit(self):
    self.fail()
```

Если вы еще не догадались, эти два теста провалятся. Это нормально. Мне нравится видеть, как они терпят неудачу, чтобы знать, что вся установка для тестирования работает. Одним из преимуществ и побочных эффектов работы опытного разработчика программного обеспечения является то, что вы не предполагаете, что что-то просто так будет работать независимо от того, кто это написал и сколько это стоит. Назовите это инстинктом выживания. Если вы пропустите этот шаг, пусть шансы всегда будут на вашей стороне.

Запуск тестов

Давайте проведем наши тесты. Как и во многих других случаях, существует множество способов запуска тестов. Вы, несомненно, заметили появление зеленых стрелок в вашем тестовом коде, как показано на рис. 6.2:

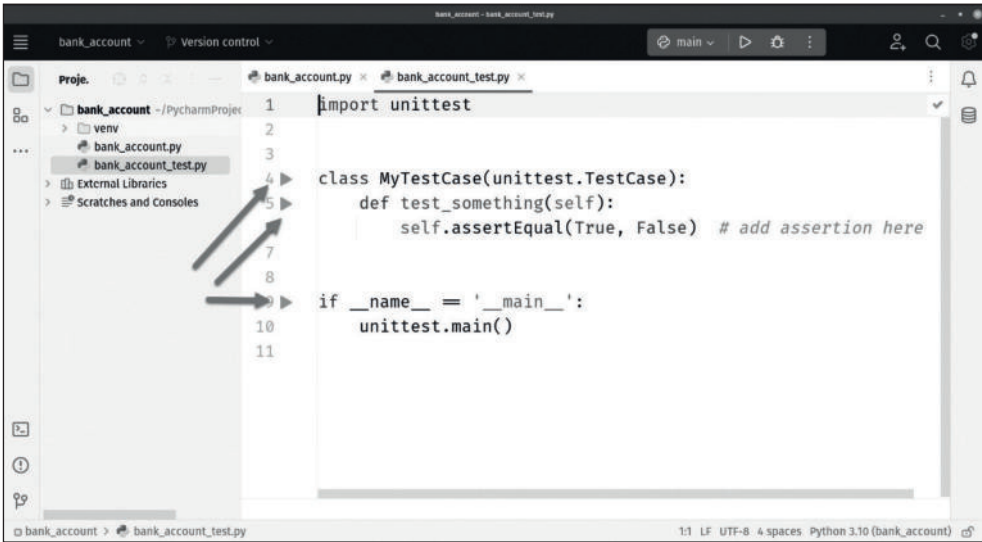


Рис. 6.2. Зеленые стрелки запуска будут появляться в IDE по мере создания тестов

Эти зеленые стрелки вызывают меню, когда вы нажимаете на них. На данный момент мы кликнем первый элемент, который называется **Run 'Python tests for...'**. Если кликнуть зеленую стрелку рядом с определением класса тестов, будут запущены все тесты этого класса. Если кликнуть зеленую стрелку рядом с любым из трех методов тестирования, будет запущен только этот тест.

Если кликнуть зеленую стрелку рядом с `BankAccountTestCase`, средство выполнения тестов появится в окне инструментов в нижней части окна IDE. Вы можете увидеть мое на рис. 6.3:

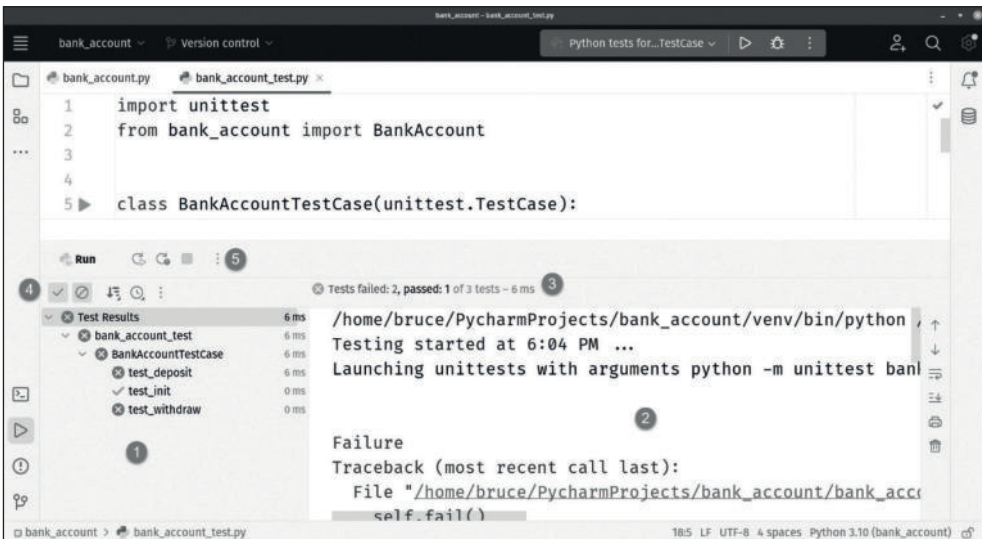


Рис. 6.3. Средство запуска тестов показывает выполненные тесты, в том числе пройденные и неудачные, а также выходные данные консоли

Сам тест-раннер имеет полный набор инструментов, встроенных в его окно. Я пронумеровал их на рис. 6.3.

1. На этой панели показаны пройденные и непройденные тесты. Они отображаются в иерархии, соответствующей иерархии вызовов.
2. На этой панели показаны выходные данные консоли самого тестового запуска.
3. Над панелью вывода отображается сводная информация о количестве пройденных тестов, а также о продолжительности работы набора тестов.
4. Слева на той же панели инструментов находится набор из пяти кнопок, за которыми следует вертикальное многоточие. Кнопки ✓ и ☒ отфильтруют все пройденные и непройденные тесты соответственно. Фильтрация пройденных тестов позволяет сосредоточиться исключительно на том, что не удалось. Фильтрация неудавшихся тестов уменьшает общее недоумение и полную безнадежность, которые вы почувствуете, если у вас будет 5 удачных из 100 пройденных тестов. Когда это происходит, я обычно съедаю бутерброд, и мне становится лучше. Посмотрите на это с такой стороны: пока у вас есть провальные тесты, ваша работа, вероятно, безопасна, потому что отладка замены займет больше времени, чем ожидание, пока все начнет работать. Воспринимайте это как наполовину полный стакан. Следующие три параметра перед многоточием позволяют сортировать результаты тестов, импортировать тесты из другого файла и просматривать историю выполнения тестов. Предположим, у вас есть тест, который прошел успешно, а затем провалился, и вы захотели вернуться и посмотреть, когда он был пройден в последний раз. Эта история есть, если она вам нужна. Многоточие содержит еще несколько опций, включая некоторые другие настройки для самого средства запуска тестов.
5. Эта панель инструментов позволяет вам перезапустить все тесты, перезапустить только те, которые завершились неудачно, и остановить длительно выполняемый тест. Опять же, у нас есть вертикальное многоточие, но здесь есть интересная опция переключения **Auto Test**. Включение этой опции приведет к непрерывному запуску ваших тестов для тех из вас, кто не может выдержать время перемещения курсора обратно к кнопке повторного запуска.

Когда вы запускаете тесты в первый раз, PyCharm автоматически создаст для вас конфигурации запуска. Вы можете увидеть ее в раскрывающемся списке **Run configuration** на верхней панели инструментов.

Исправление неудачных тестов

У нас есть два теста, которые всегда провалятся, что бы мы ни делали. Начнем с изменения метода `test_draw(self)` в файле `bank_account_test.py`. Измените это на следующее:

```
def test_withdraw(self):
    test_account = BankAccount("Bruce Van Horn", "123355-23434", 4000)
    test_account.withdraw(2000)
    self.assertEqual(test_account.balance, 2000)
```

Первая строка создает экземпляр класса `BankAccount` с некоторыми проверяемыми значениями. Затем мы вызываем метод `withdraw` и выводим 2000 долл. Надеюсь, чтобы потратить их на чего-нибудь забавное! Обычно мои дочери одалживают мой кошелек, чтобы купить одежду или, возможно, материалы для ремонта. Я имею право на это надеяться, да? Теперь я ожидаю, что мой баланс упадет с 4000 до 2000 долл. Итак, я использую метод `AssertEqual` в классе `unittest`, который является суперклассом для моего класса `BankAccountTestCase`. Я передаю `test_account.balance`, который будет сравниваться с ожидаемым результатом.

Я совершенно уверен, что этот тест пройдет успешно! Нажмите кнопку повторного запуска неудачных тестов, показанную на рис. 6.3. Он прошел! Теперь давайте напишем метод `test_deposit`:

```
def test_deposit(self):
    test_account = BankAccount("Bruce Van Horn", "123355-23434", 4000)
    test_account.deposit(5000)
    self.assertEqual(test_account.balance, 9000)
```

Объяснение здесь такое же, как и предыдущее, за исключением того, что на этот раз мы вносим на мой счет 5000 долл. В реальной жизни такое случается редко, так что дайте мне минутку порадоваться.

Перезапустите неудачные тесты. Сейчас они все должны пройти! Но мы еще не закончили, не так ли?

До сих пор эти тесты проходили без ошибок. Это означает, что до сих пор я тестировал методы только во время их запуска в том виде, в каком я их разработал. Пользователи в реальном мире никогда этого не сделают. Нам также необходимо проверить варианты отказа.

Тестирование на отказ

Есть одна очевидная ошибка, которую мы заложили в систему: овердрафт. Что произойдет, если мы попытаемся вывести больше денег, чем имеется на текущем балансе? Или, как могли бы сказать мои дочери, как нам создать сигнал, который сообщит нам, что пора вернуться домой из торгового центра и спрятать чеки?

Мы учитываем это в нашем коде:

```
def withdraw(self, amount: float) -> None:
    new_balance = self.balance - amount
    if new_balance > 0:
        self.balance = new_balance
    else:
        raise ValueError("Account overdrawn!")
```

Как видите, мы проверяем, будет ли новый баланс отрицательным числом. Если это так, мы выдаем `ValueError`. Этот тест будет немного другим. Вместо использования `AssertEquals` для проверки результата без ошибок мы хотим убедиться, что при наличии этого условия мы не только выдаем ошибку, но и выдаем ошибку правильного типа. Это важно, потому что мы ожидаем `ValueError`, но если возникнет какая-то другая ошибка, тесты дадут нам ложный

положительный результат, если мы проверим только общее `Exception`. Добавьте следующий тест класса `BankAccountTestCase`:

```
def test_overdraft(self):
    test_account = BankAccount("Bruce Van Horn", "123355-23434", 4000)
    self.assertRaises(ValueError, test_account.withdraw, 5000)]
```

Как и прежде, мы создаем экземпляр класса `BankAccount` с некоторыми проверяемыми значениями. Для теста мы хотим утверждать, что метод вывода вызывает ошибку `ValueError`, если мы передаем больше денег, чем имеется на балансе. Здесь мы используем `self.assertRaises`, который принимает три аргумента. Первый аргумент – это ожидаемый тип ошибки.

Второй аргумент – это тестируемый метод. Обратите внимание, что мы передаем ссылку на функцию в стиле лямбда. Мы не выполняем эту функцию, поскольку для этого нам нужна функция `AssertRaises`. Наконец, нужно передать значение любых аргументов – в данном случае некоторых чисел, превышающих те четыре тысячи, которые я использовал для создания экземпляра. В данном случае я передаю 5000. Когда я запускаю этот тест, он должен пройти, потому что функция завершится ошибкой с ожидаемым исключением `ValueError`.

Остался всего один тест: нам нужно быть уверенными, что при передаче отрицательного числа в метод `deposit` мы получим `ValueError`. Я оставлю это для вас, чтобы вы могли попрактиковаться. Полный рабочий код находится в репозитории кода этой главы.

Автоматическое создание тестов

До сих пор мы потратили некоторое время на написание класса `BankAccount`, но вернемся к нашей первоначальной идее варианта использования для модульного тестирования – финансовой транзакции. На этот раз мы собираемся написать код, который необходимо протестировать, но вместо общего шаблона теста мы собираемся создать более точный тест.

Начнем с кода, который мы будем тестировать. Создайте в своем проекте новый файл под названием `transaction.py`. Содержимое этого файла должно выглядеть следующим образом:

```
from bank_account import BankAccount
```

Нам понадобится класс `BankAccount`, поскольку вся идея состоит в том, чтобы написать код, который переводит деньги с одного счета на другой в ответ на продажу предмета. Говоря об *item*, давайте создадим класс, который будет представлять то, что мы будем покупать:

```
class Item:
    def __init__(self, name: str, price: float):
        self.name = name
        self.price = price
```

Здесь нет ничего необычного – всего две переменные экземпляра, называемые `name` и `price`. Теперь самое сложное: нам нужен класс для представления транзакции. Помните, что транзакция – это атомарная операция. Все шаги

должны быть выполнены. Если по пути возникнут ошибки, то все, что было до ошибки, нужно откатить:

```
class Transaction:
    def __init__(self, buyer: BankAccount, seller: \
BankAccount, item: Item):
        self.buyer = buyer
        self.seller = seller
        self.item = item
```

Мы начали класс с конструктора, который инициализирует два банковских счета и элемент. После этого следует логика самой транзакции:

```
def do_transaction(self):
    original_buyer_balance = self.buyer.balance
    original_seller_balance = self.seller.balance
```

Нам нужно сохранить исходные балансы. Если что-то пойдет не так, нам понадобится эта информация, чтобы вернуть все как было. Далее следует этап, когда деньги переходят из рук в руки. Я попробую:

```
try:
    self.buyer.withdraw(self.item.price)
    self.seller.deposit(self.item.price)
except ValueError:
    self.buyer.balance = original_buyer_balance
    self.seller.balance = original_seller_balance
    raise ValueError("Transaction failed and was \
        rolled back")
```

Мы пытаемся снять деньги со счета покупателя, а затем внести ту же сумму на счет продавца. Если выдается ошибка `ValueError`, мы возвращаем все деньги, восстанавливая балансы до их исходных значений. После того как деньги были восстановлены, мы все равно должны выдать сообщение об ошибке, чтобы основное приложение знало, что произошла ошибка. Эта функция должна будет сообщить результат пользовательскому интерфейсу, чтобы сообщить пользователю, что произошло с транзакцией. Последняя строка делает это за нас. В реальном приложении вы можете захотеть создать свою собственную ошибку, которая может предоставить больше информации, но эта ошибка хорошо подходит для демонстрационных целей.

Создание теста транзакции

Ранее мы создали новый тест с помощью меню **File**. Это дало нам очень общий тест, который не имел ничего общего с нашей работой на тот момент. На этот раз мы сгенерируем тест из самого определения класса. Кликните правой кнопкой мыши строку, которая начинается с `class Transaction`. Затем выберите пункт меню **Generate...**, как показано на рис. 6.4:

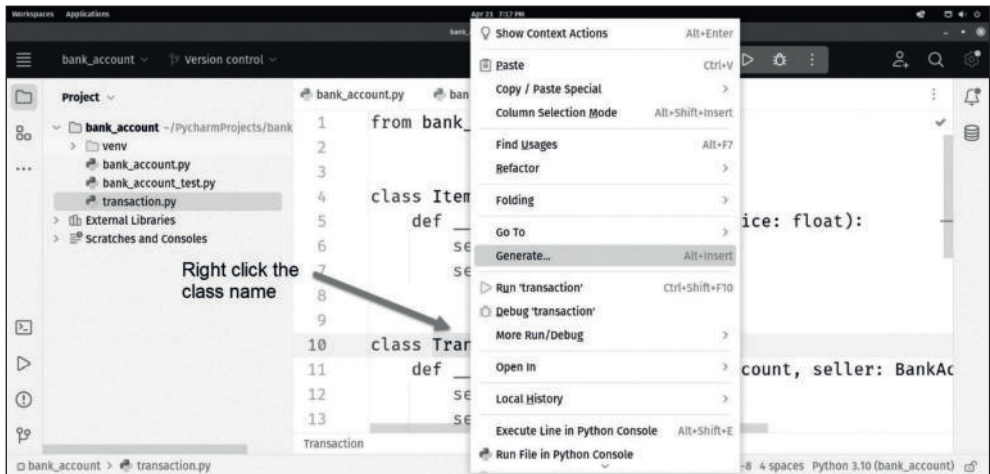


Рис. 6.4. Пункт меню **Generate...** можно найти, кликнув правой кнопкой мыши определение класса

Затем нажмите **Test...**, как показано на рис. 6.5:

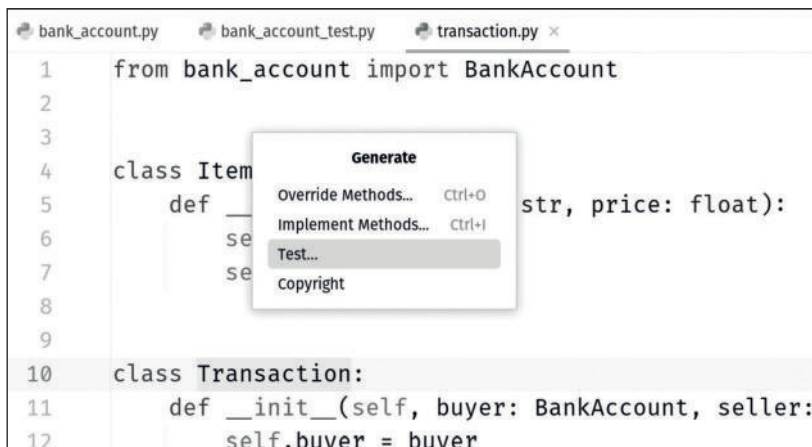


Рис. 6.5. Нажмите кнопку **Test...**, чтобы сгенерировать тест

На этом этапе появится диалоговое окно, в котором вы сможете управлять создаваемым тестом, как показано на рис. 6.6:

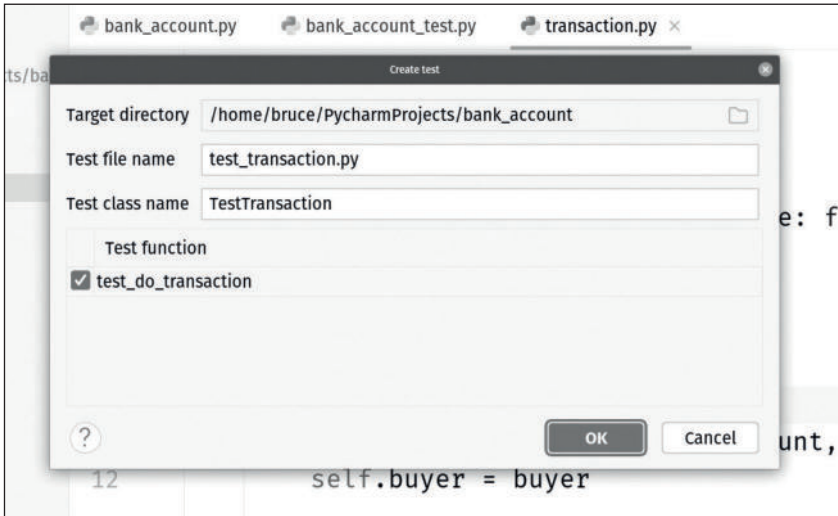


Рис. 6.6. PyCharm собирается сгенерировать файл модульного теста на основе этих настроек

PyCharm собирается создать файл с именем `test_transaction.py`. В этом файле вместо общего имени тестового класса будет определение класса `TestTransaction`. Наконец, внутри файла, если вы оставите флажок установленным, будет сгенерирована заглушка (stub) тестового метода с именем `test_do_transaction`.

Результирующий файл содержит этот код:

```
from unittest import TestCase
class TestTransaction(TestCase):
    def test_do_transaction(self):
        self.fail()
```

Еще в главе 1 я говорил вам, что одним из преимуществ IDE является то, что она позволяет сократить количество шаблонов. В первый раз PyCharm сгенерировала для нас некий общий шаблон. По крайней мере, нам не пришлось его вводить, но на изменение того, что он генерировал, потребовалось почти столько же усилий. На этот раз работы еще меньше. Если бы в моем классе было много методов, для каждого из них была бы правильно именованная заглушка. Все, что мне нужно сделать сейчас, – это написать код, который обеспечивает передачу метода `test_do_transaction`. Вот!

```
from unittest import TestCase
from bank_account import BankAccount
from transaction import Transaction, Item
```

Мы начали с необходимого импорта. Я знаю, что мне понадобятся два теста, а не только один, сгенерированный PyCharm. PyCharm сгенерировала один тестовый метод, который я использую для пути без ошибок. Я передам что-то, что работает так, как задумано для этого метода. Поскольку я знаю, что у меня есть два теста, я могу повторно использовать учетную запись

продавца, чтобы мой тест оставался **DRY**. Если вы не знакомы с этой аббревиатурой, она означает «**Не повторяйте себя**» (Don't Repeat Yourself). Подняв этот код в начало файла, мне нужно ввести его только один раз. Этот код инициализирует банковский счет продавца с балансом в 4000 долл. США. Он также устанавливает покупаемый нами предмет, который не будет меняться между тестами:

```
initial_seller_balance = 4000
seller_account = BankAccount("PacktPub", "839423-38402", initial_seller_balance)
item = Item("Python book", 39.95)
```

Далее перейдем к самому тестовому классу, который для нас был сгенерирован. У нас уже есть эта часть:

```
class TestTransaction(TestCase):
    def test_do_transaction(self):
```

Я заменяю сгенерированный `self.fail()` кодом, который, как я надеюсь, обеспечит прохождение теста:

```
buyer_account = BankAccount("Bruce Van Horn", "123355-23434", 99)
item = Item("Python book", 39.95)
test_transaction = Transaction(buyer_account, \
    seller_account, item)
```

Как обычно, я создаю экземпляры классов, которые буду использовать в тесте. На данный момент я сделал два аккаунта и товар с ценой. Далее я запущу тестируемый метод:

```
test_transaction.do_transaction()
```

Затем проверю свои результаты:

```
self.assertEqual(buyer_account.balance, 99 - 39.95)
self.assertEqual(seller_account.balance, initial_seller_balance + 39.95)
```

У вас может возникнуть соблазн пофантазировать с тестовым кодом. Будьте осторожны с этим. Причудливый тестовый код с такой же вероятностью сломается, как и намеренно причудливый код, который предназначен для тестирования. Если вы тестируете сложные математические вычисления, не дублируйте вычисления в тесте, а затем сравнивайте их с тестируемым кодом. Вам следует подключить известные входы и проверить известные выходы. Больше ничего!

Этот тест представляет собой путь без ошибок. Я ожидаю, что это пройдет, поскольку это упражнение просто предполагает, что все работает в идеальных условиях. Посмотрим, прав ли я. Нажмите любую из зеленых кнопок запуска. Мой результат показан на рис. 6.7:

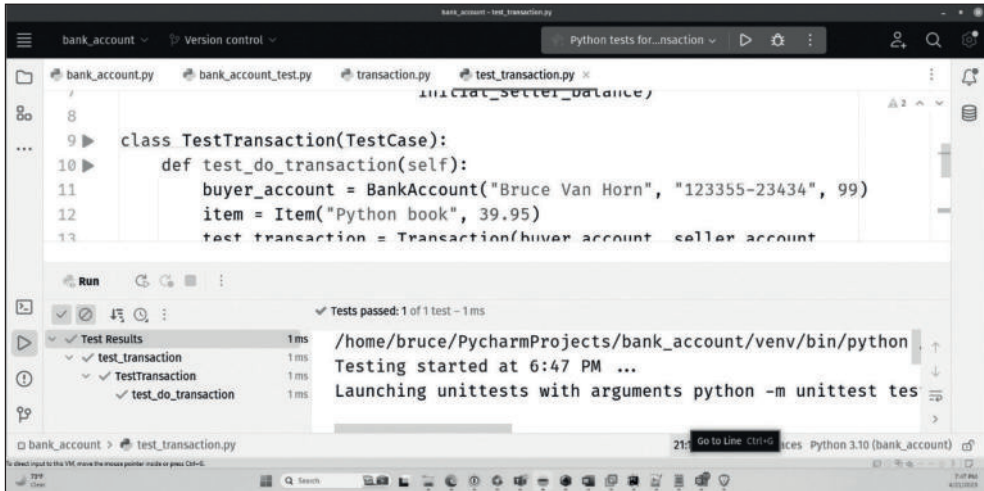


Рис. 6.7. Пока все хорошо! Мой тест проходит!

Нам нужен тест хотя бы на один путь неисправности. В данном случае это будет проверка того, что произойдет, если на моем счету не будет достаточно средств для покупки книги.

Вот мой тест для этого случая:

```
def test_transaction_overdraw_fault(self):
    initial_buyer_balance = 5
    buyer_account = BankAccount("Bruce Van Horn", \
        "123355-23434", initial_buyer_balance)
    test_transaction = Transaction(buyer_account, \
        seller_account, item)
```

Когда я сегодня ушел на работу, на моем счету было как минимум 9000 долл. Но моя дочь Фиби «одолжила» мою визитку из кармана куртки. Она сказала, что собирается создать завод по производству роботизированных велосипедов. Я ничего об этом не думал. Она шутила, да? Итак, после работы я иду в книжный магазин, намереваясь купить последний шедевр:

```
test_transaction.do_transaction()
```

Транзакция происходит. Знаете ли вы этот звук, который издает Пакман, когда его съедает призрак? Я сейчас издаю этот звук. Продажа не удастся; посмотрим, правильно ли откатывается транзакция:

```
self.assertEqual(buyer_account.balance, initial_buyer_balance)
self.assertEqual(seller_account.balance, initial_seller_balance)
```

Этот последний фрагмент кода проверяет, что балансы покупателя и продавца возвращаются к исходным значениям. Запустите тесты – они оба должны пройти! Смотрите рис. 6.8, где представлен мой триумфальный тестовый запуск. Мне не терпится пойти домой и расслабиться!

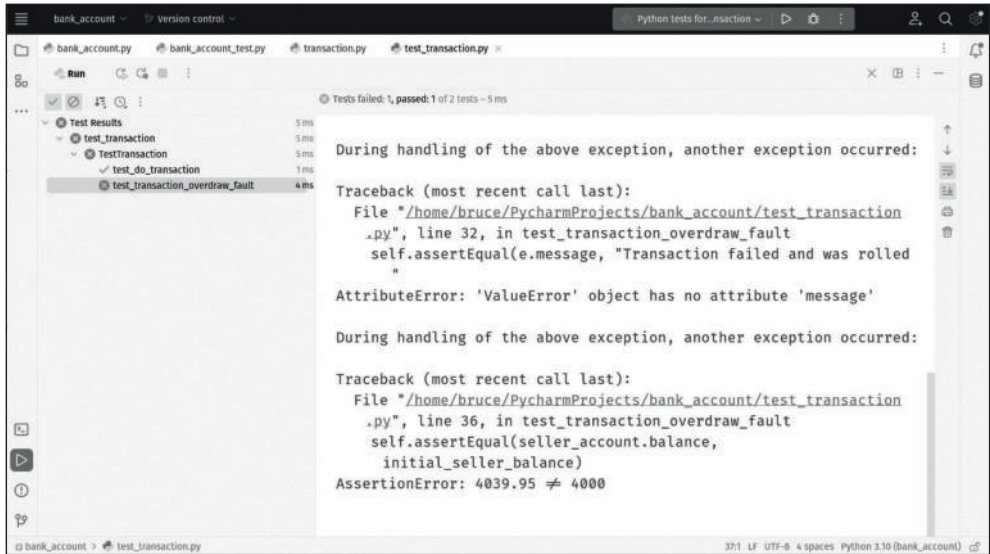


Рис. 6.8. Мне лучше позвонить жене и сказать ей, чтобы она оставила для меня ужин теплым

Похоже, я оказался слишком самоуверен. В окне вывода отображается набор трассировок стека для всего, что пошло не так. Он настолько длинный, что мне пришлось немного прокрутить вниз, чтобы добраться до основной части этого скриншота. В трассировке (которая не показана) я вижу, что несколько ошибок, которые, как я думал, будут выданы, были, и это нормально. Те две, которые мы видим здесь, таковыми не являются. Во-первых, я намеревался убедиться, что сообщение, поступающее из исключения, соответствует значению, которое я присвоил в определении. Опять же, я делаю это, чтобы убедиться, что выброшенная мною ошибка – это именно та ошибка, которую мы видим, а не какая-то другая, возникшая в результате более ранней ошибки. Похоже, я не понял структуру ошибки, да и атрибута `message` на самом деле нет.

Другая, более тревожная ошибка заключается в том, что моя транзакция не была отменена! Посмотрев на трассировку, вы увидите, что повсюду имеются гиперссылки, позволяющие перейти непосредственно к коду неисправности, указанному в трассировке. Очень легко передвигаться и искать проблемы. Я могу найти строку для первой проблемы в списке трассировок стека, как показано на рис. 6.9:

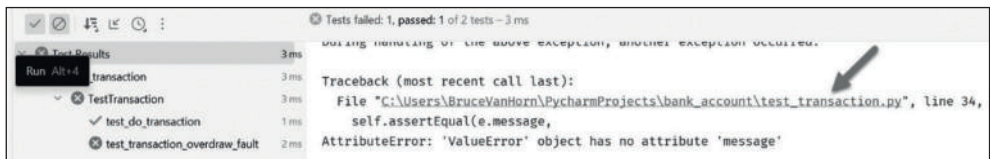


Рис. 6.9. Трассировки стека пронизаны гиперссылками, которые перенаправляют вас к проблемному разделу вашего кода

Кликнув эту ссылку, я попаду к проблемному коду, показанному на рис. 6.10:

```

31 | try:
32 |     test_transaction.do_transaction()
33 | except ValueError as e:
34 |     self.assertEqual(e.message,
35 |                     "Transaction failed and was rolled back")
36 | finally:
37 |     self.assertEqual(buyer_account.balance, initial_buyer_balance)
38 |     self.assertEqual(seller_account.balance, initial_seller_balance)
39 |

```

Рис. 6.10. Древний ужас пробудился! IDE говорила мне, что строка 34 неверна, но я не послушался!

У меня есть несколько вариантов, не так ли? Я мог бы использовать функции документации в PyCharm, наведя курсор на переменную `e`. Мы говорили о функциях автоматического документирования в главе 4. На рис. 6.11 показано, как это выглядит, если вы пропустили:

```

31 | try:
32 |     test_transaction.do_transaction()
33 | except ValueError as e:
34 |     self.assertEqual(e.message,
35 |                     "
36 | finally:
37 |     self.assertEqual(b
38 |     self.assertEqual(s
39 |

```

builtins

class ValueError(_StandardError)

Inappropriate argument value (of correct type).

["ValueError\(_StandardError\)" on docs.python.org](https://docs.python.org/3/library/exceptions.html#ValueError)

Рис. 6.11. Функция автоматического документирования предоставит мне ссылку на официальную документацию

Здесь нет простого ответа, не так ли? Конечно, я мог бы нажать на ссылку внизу, перейти на сайт Python и прочитать документацию. Однако если я это сделаю, я потеряю всякое доверие к вам, читатель. Читать инструкцию под просмотром? Никогда! Я уверен, что найду ответ, но не ценой самоуважения.

У меня есть еще одна идея! Я уже говорил о консоли PyCharm. Хочу попробовать кое-что. Посмотрите на рис. 6.12.

Стрелка, указывающая на 2, откроет консоль PyCharm. Если вы никогда этого не делали, этого значка не будет на панели инструментов. Вам нужно будет кликнуть многоточие в *цифре 1* и кликнуть область **Python Console**. Это добавит его на вашу панель инструментов. Мой сеанс с этой консолью показан на рис. 6.13:

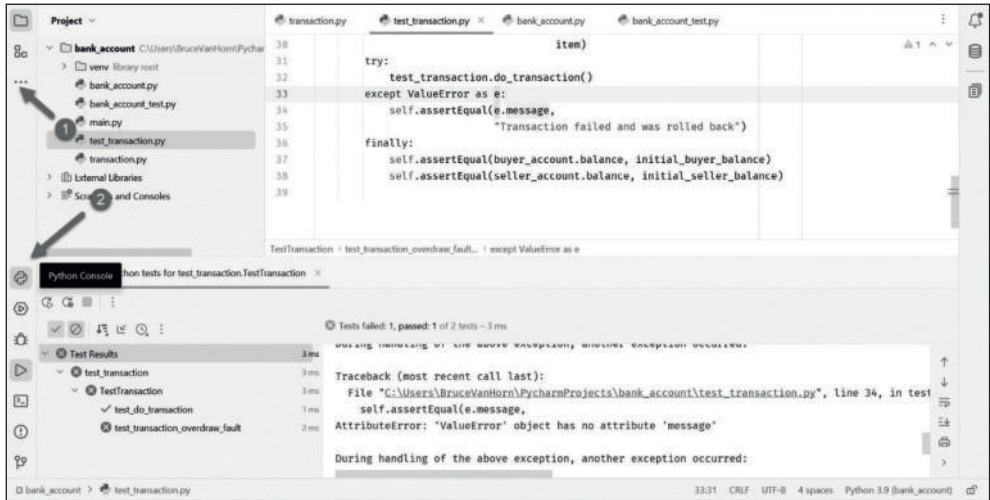


Рис. 6.12. Если кнопка консоли PyCharm (2) отсутствует на панели инструментов, кликните эллипс (1), чтобы включить ее



Рис. 6.13. Повторное посещение консоли позволит нам провести быстрый эксперимент, чтобы устранить ошибку

В консоли я сначала нажал *Enter* в первой строке. Я сделал это для вас. Если бы я этого не сделал, консоль бы все перепутала, и это выглядело бы не так красиво. Далее я набрал следующее:

```
check = ValueError("This is a test")
```

Я подозреваю, что если я преобразую чек в строку, то получу искомое сообщение. Назовите это интуицией. Или скажите: «Ты подсмотрел это с помощью ChatGPT, пока я отвернулся». Я руководствуюсь интуицией.

Если я введу `str(check)`, Python REPL оценит выражение и распечатает результат. Идея работает. Я могу исправить свой код. Строка 34 в `test_transaction.py` теперь будет выглядеть следующим образом:

```
self.assertEqual(str(e), «Transaction failed and was rolled back»)
```

Теперь, если я запущу тест еще раз, он завершится неудачно, как показано на рис. 6.14:

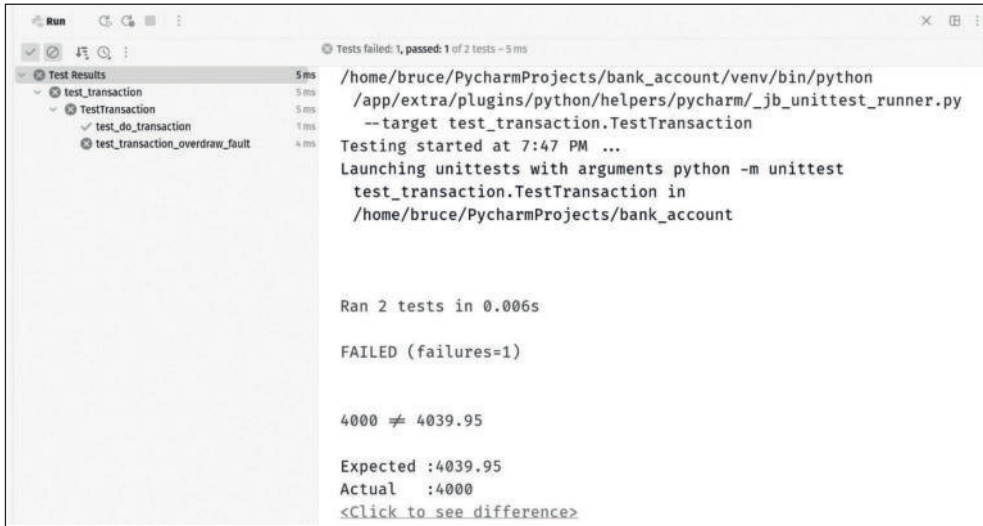


Рис. 6.14. Прогресс в разработке программного обеспечения можно измерить по скорости сокращения списка ошибок

Мы этого ожидали. Список задач стал короче, так что это победа! Давайте проясним последнюю проблему. Транзакция не может правильно сбросить стоимость счета продавца после сбоя транзакции. Мы могли бы посмотреть на него какое-то время или могли бы применить более активный подход, запустив отладчик PyCharm и пройдя весь тест.

РАБОТА С ОТЛАДЧИКОМ PYCHARM

В главе 1 я хвалил отладчик PyCharm как единственную главную причину использовать IDE вместо отладчика командной строки, такого как стандартный отладчик Python, который называется **pdb**. Поймите меня правильно: вам следует научиться использовать **pdb**, потому что бывают случаи, когда IDE недоступна. Однако я подозреваю, что, воспользовавшись PyCharm, вы предпочтете его чему-либо еще. Посмотрим, прав ли я.

У нас есть проблема в нашем классе `Transaction`, которая не совсем точно определена. Когда дело доходит до тестирования, всегда есть две возможности:

- код не работает из-за ошибки в тестируемом коде,
- код не работает из-за тестового кода.

Поскольку на данный момент мы не знаем, какая из возможностей правильна, отладчик позволит нам пройти по коду по одной строке за раз и проверить его внутреннюю работу. Для этого нам нужно установить точку останова. Точка останова отмечает место в вашем коде, где вы хотели бы остановить его выполнение и проверить содержимое переменных, стека и т. д. Вы можете создать точку останова, кликнув номер строки в поле редактора, как показано на рис. 6.15. Я собираюсь добавить точку останова в начало теста, чтобы мы могли пройти его. Тест начинается со строки 25, поэтому я нажимаю на этот номер строки; обратите внимание, что номер строки заменен красной точкой:

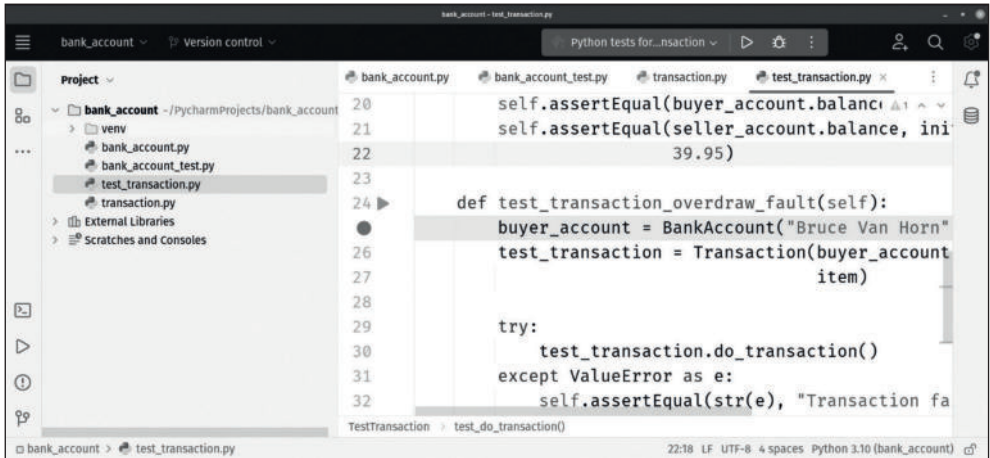


Рис. 6.15. Кликните номер строки, чтобы создать точку останова, которая заменит номер красной точкой

Далее нам нужно запустить отладчик. Кликните зеленую стрелку в окне редактора рядом с определением метода `test_transaction_overdraw_fault(self)`. На этот раз выберите опцию **Debug 'Python tests for tes...'**, показанную на рис. 6.16, чтобы запустить неудавшийся тест:

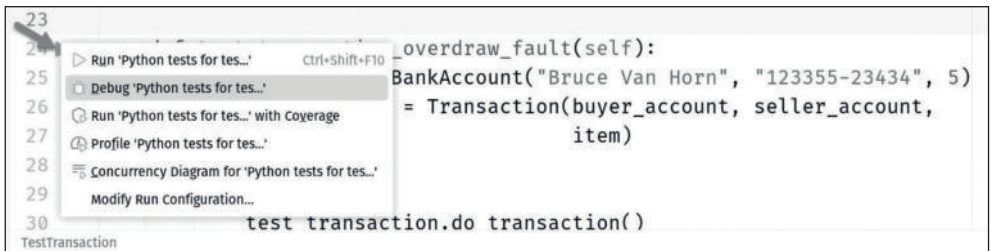


Рис. 6.16. При нажатии на зеленые стрелки открывается меню, которое можно использовать для внесения изменений в выполняемый тест, включая запуск отладчика

При запуске отладчика программа запустится, а затем остановится на строке 25 нашего теста. IDE значительно преобразилась. Давайте посмотрим на рис. 6.17.

Есть несколько вещей, которые вы можете заметить сразу. Во-первых, кнопки запуска в верхней части IDE теперь зеленые, а красная кнопка остановки подсвечивается (1). Все это визуальные подсказки, показывающие, когда что-то работает.

Нижняя половина IDE теперь занята инструментами отладчика (2). Также имеется панель вкладок (3), которая позволяет одновременно запускать несколько сеансов отладки. Это может быть полезно при разработке микросервисных архитектур RESTful, о которых мы поговорим в нескольких следующих главах, особенно в главе 9 «Создание RESTful API с помощью FastAPI».

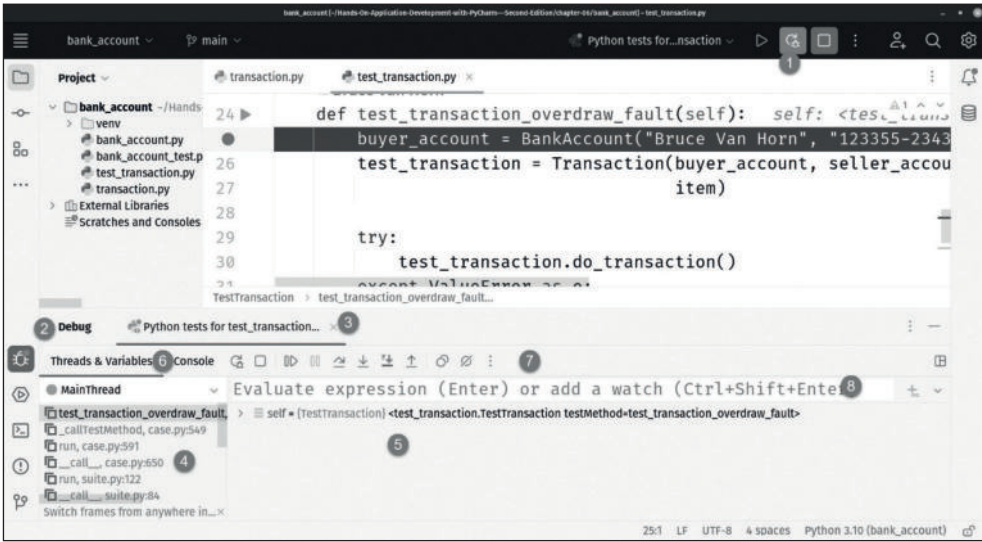


Рис. 6.17. Приостановленный отладчик в PyCharm

С правой стороны находится список потоков (4), который позволяет вам переключаться между различными потоками и проверять их. Однако в большинстве случаев вы приземлитесь в нужном месте и будете использовать это лишь изредка. Область в позиции 5 показывает все, что в данный момент находится в области видимости. Сейчас это просто `self`, который, как вы можете видеть, является экземпляром класса `TestTransaction`.

Отметка 6 показывает две вкладки, которые позволяют вам переключаться между видами, видимыми прямо сейчас, что позволяет вам проверять состояние программы в области 5. Если вы переключите эту вкладку на **Console**, в области 5 отобразятся выходные данные терминала вашей программы. Любые операторы вывода или `print` будут отображаться, чтобы вы могли просматривать вывод во время работы программы.

Панель инструментов, отмеченная цифрой 7, содержит набор очень полезных инструментов, а окно выражений (8) позволяет вам добавлять наблюдения или оценивать выражение, используя все, что в данный момент находится в области видимости.

Наиболее полезными частями окна отладки являются область проверки (5), переключатель вкладок, который можно использовать для переключения между инспектором переменных и потоков и выводом консоли (6), а также панель инструментов отладки (7). Давайте подробнее рассмотрим панель инструментов отладки:



Рис. 6.18. Панель инструментов отладки в PyCharm

Я пронумеровал каждую кнопку. Давайте рассмотрим их.

1. Эта кнопка перезапускает отладку. Вы можете найти дублирующую кнопку перезагрузки в верхней части окна IDE рядом с кнопкой запуска.
2. Эта кнопка останавливает выполнение отладки. Вы можете найти дубликат кнопки остановки в верхней части окна IDE рядом с кнопкой перезагрузки.
3. Это кнопка **Continue**. Отладчик остановится в любой точке останова и будет ждать, пока вы не воспользуетесь одной из кнопок шага (5–8) или не нажмете эту кнопку, чтобы продолжить выполнение.
4. Кнопка **Pause** приостановит прогон. Это может быть полезно, если вы запускаете цикл или алгоритм, выполнение которого занимает некоторое время, и вы хотите приостановить выполнение.
5. Кнопка **Step Over** выполнит текущую строку, на которой отладчик остановился. Если эта строка является вызовом функции в вашей программе, функция выполнится нормально и вернется, после чего вы перейдете к следующей строке вашего кода, где отладчик останется на паузе. Здесь вы переступаете через выполнение следующей строки.
6. Кнопка **Step Into**. Если ваш отладчик остановился на строке, содержащей вызов функции, нажатие этой кнопки позволит вам войти в эту функцию и пройти ее так, как если бы вы поместили точку останова в начале функции. **Step Over** пропускает это выполнение, а эта кнопка переходит к нему.
7. **Step Into My Code** меняет правила игры! Эта кнопка аналогична кнопке **Step Into** (6), за исключением того, что она не позволяет войти в код, который вы не создавали. Под этим я подразумеваю, что кнопка **Step Into** с радостью приведет вас в недра кода вашей сторонней библиотеки или в код, содержащий сам Python. Это редко бывает полезно. Кнопка **Step Into My Code** позволяет перейти только к коду, который является частью вашего проекта.
8. Это кнопка **Step Out**. Если вы обнаружите, что вступаете в какой-то код, который явно не является проблемой, или, возможно, вы ввели код библиотеки, который не создавали, код выхода вернет вас обратно в точку, в которую вы вошли.

Внимание пользователей Visual Studio

Кнопки в отладчике PyCharm работают иначе, чем в Visual Studio! Мне потребовалось время, чтобы к этому привыкнуть. В Visual Studio вы можете нажать зеленую кнопку на верхней панели инструментов, чтобы начать сеанс отладки. Когда вы достигнете точки останова, можете нажать ту же кнопку, чтобы продолжить. В PyCharm кнопка продолжения находится на панели инструментов отладки, в области 3 на рис. 6.18. Если бы вы нажали ту же кнопку, которую использовали для запуска отладчика, вы бы запустили второй сеанс отладки. PyCharm обычно будет жаловаться, когда вы это сделаете, если вы не установите флажок в конфигурации запуска, который разрешает несколько запусков одновременно.

Использование отладчика для поиска и устранения проблемы, выявленной тестом

Наш модульный тест выявил проблему в нашем коде. Когда наша транзакция терпит неудачу из-за ошибки овердрафта, мы ожидаем, что балансы нашего покупателя и продавца вернутся к своим первоначальным значениям. На данный момент продавец получает кредит в размере 39,95 долл. США после неудачной транзакции. Давайте воспользуемся отладчиком и посмотрим, сможем ли мы выяснить, почему это происходит.

Как показано на рис. 6.19, мы запустили отладчик для нашего модульного теста и остановились на строке 25 в `test_transaction.py`. На этом этапе ничего из тестового метода не выполнено. Когда вы смотрите на выделенную строку в отладчике PyCharm, вам нужно помнить, что выделенная строка еще не была выполнена. Чтобы выполнить строку, нажмите кнопку **Step Over**, которая на рис. 6.18 обозначена цифрой 5. Окно переменных отладчика обновится после создания экземпляра `buy_account`, а выделение переместится на строку 26 и остановится на ней, как показано на рис. 6.19:

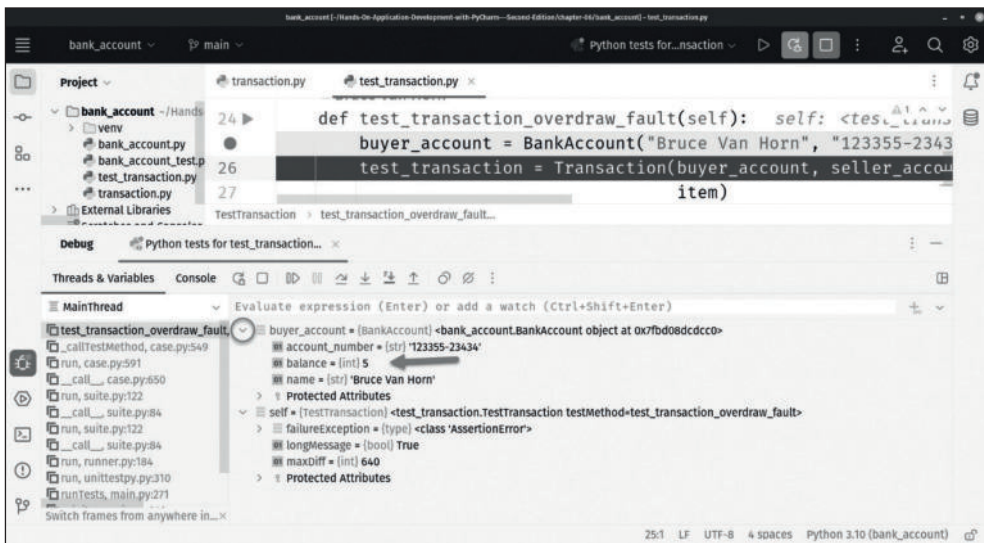


Рис. 6.19. После нажатия кнопки Перейти отладчик был остановлен на строке 26

Чтобы увидеть содержимое объектов, вам нужно открыть курсор, который я обвел на рис. 6.19. Вы увидите, что баланс `buyer_account` составляет 5 долл. США. Однако нас здесь интересует учетная запись продавца, поскольку именно здесь кроется проблема.

Кликните строку 30, чтобы добавить туда точку останова, затем нажмите кнопку **Continue**. Отладчик остановится на строке 30. Мы собираемся войти в метод `do_transaction()` и посмотреть, как он выполняется. Нажмите кнопку **Step Into My Code**. Обратитесь к рис. 6.18 и посмотрите на цифру 7, если вы не помните, какую кнопку я имею в виду. Это приведет нас к строке 17 в `transaction.py`. Перейдите через строки 17 и 18, чтобы добраться до стро-

ки 19 и проверить наше состояние. Вы увидите проблему, показанную на рис. 6.20:

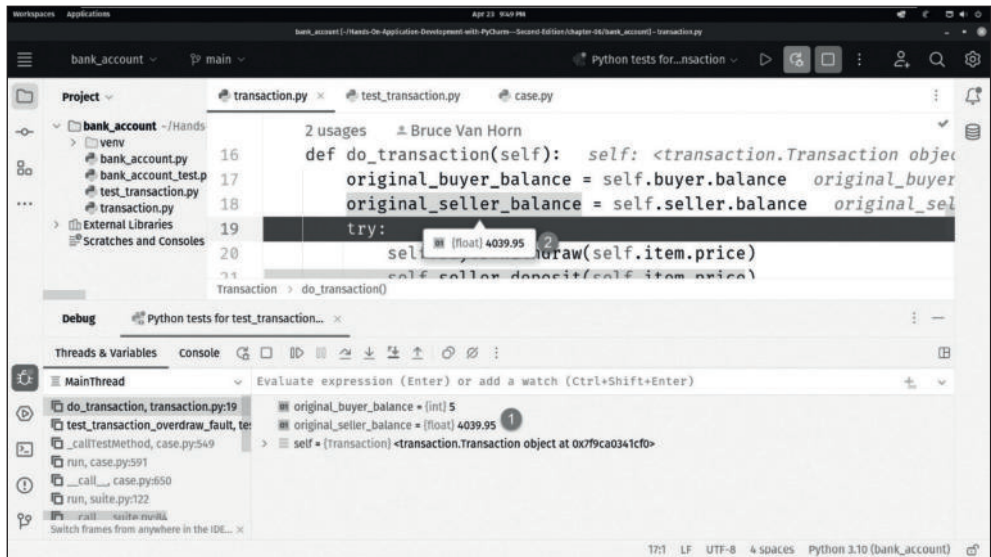


Рис. 6.20. Отладчик показывает, что начальное значение баланса продавца неверно

Отладчик показывает, что первоначальный баланс продавца составляет 4039,95 долл. США, тогда как мы ожидали, что он составит 4000 долл. Вы можете увидеть это значение в двух местах. Окно переменных показывает это нам (1), но вы также можете навести курсор на любую переменную в области видимости в окне редактора (2) и увидеть ее значение.

Итак, почему наш стартовый баланс может быть неправильным? Это проблема масштаба! Поскольку я поднял переменную `seller_account` в строке 5 в `test_transaction.py` до глобального значения, первый тест успешно изменил баланс до 4039,95 долл., как и должно быть. Поскольку оно глобальное, это число остается. Чтобы это исправить, нам нужно сбросить баланс аккаунта продавца в начале метода `test_transaction_overdraw_fault(self)`. Мы начали отладку со строки 25. Давайте просто внесем изменения именно туда. Нажмите кнопку «stop» на панели инструментов отладчика, затем добавьте эту строку в строку 25:

```
seller_account.balance = 4000
```

Повторно запустите тесты без отладки. Смелее! Предположим, это сработало. Если вы не следуете инструкциям, любезно подойдите к краю сиденья и начните нервно грызть ногти. Победит ли наш герой на рис. 6.21? Органная музыка: ду-ду-ду-дуууу!

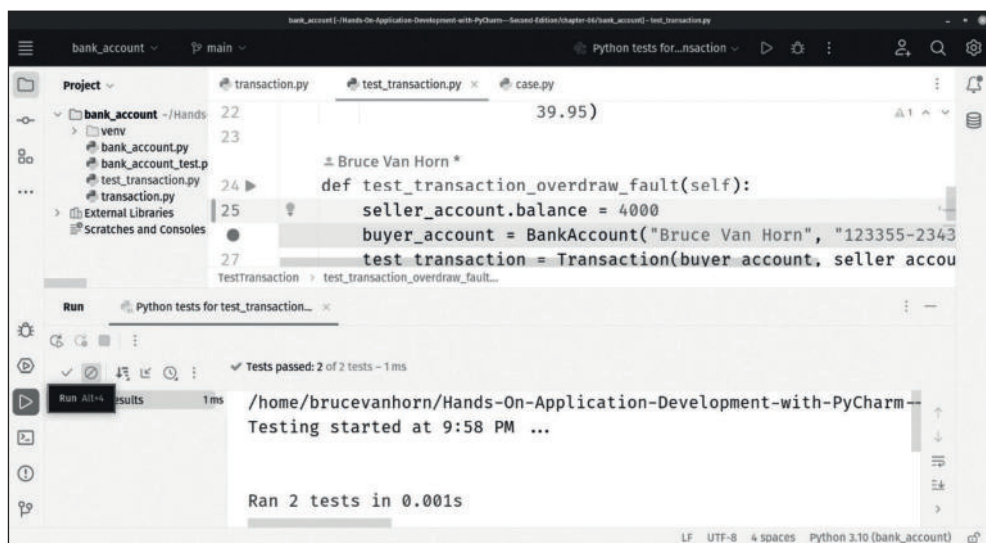


Рис. 6.21. Победа!

Это работает! Теперь пришло время отправиться домой и разогреть ужин, поскольку мы восстановили всеобщую веру в международный банковский бизнес.

ПРОВЕРКА ТЕСТОВОГО ПОКРЫТИЯ

Модульные тесты наиболее эффективны, когда они охватывают каждый класс, метод, функцию или модуль в вашей программе. По мере роста вашего программного кода легко забыть о написании тестов или, возможно, отложить их до тех пор, пока у вас не появится больше времени. В PyCharm есть инструмент, который может подсказать вам, каково ваше тестовое покрытие, и поможет найти неиспользованные возможности для тестирования большего количества вашей работы, чем вы сделали бы сами.

Чтобы проверить тестовое покрытие, просто нужно запустить тесты немного по-другому. Мы запускали наши тесты индивидуально из тестовых файлов. Нам необходимо провести все тесты вместе, чтобы получить исчерпывающий отчет о том, где нам не хватает покрытия. Для этого создадим новую конфигурацию запуска. Кликните раскрывающийся список «run configurations» на панели инструментов и нажмите **Edit Configurations**. Добавьте новую конфигурацию, используя шаблон Python unittest. Убедитесь, что вы используете мои настройки, как показано на рис. 6.22.

В качестве пути скрипта введите папку, в которой находятся ваши тесты. Установите шаблон *_test.py. Это заставит программу запуска тестов найти все файлы, заканчивающиеся на _test.py, что отличается от значений по умолчанию. По умолчанию ищутся файлы, начинающиеся с «test». Мне это не особенно нравится, потому что все тесты группируются в окне файла проекта, а не рядом с тестируемым файлом.

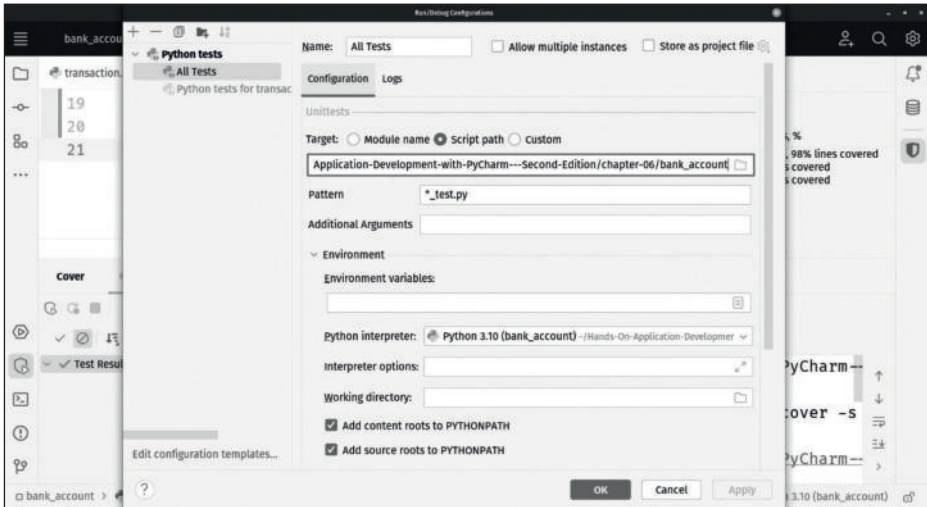


Рис. 6.22. Создайте конфигурацию запуска, которая запускает все ваши тесты одновременно

Установив шаблон и установив средство запуска тестов в папку, а не в один файл, средство запуска найдет все файлы, соответствующие шаблону, и запустит их как тесты. Говоря о запуске, вы можете сделать это, кликнув многоточие рядом с кнопками запуска и отладки. Смотрите рис. 6.23, чтобы найти пункт меню **Run 'All Tests' with Coverage**:

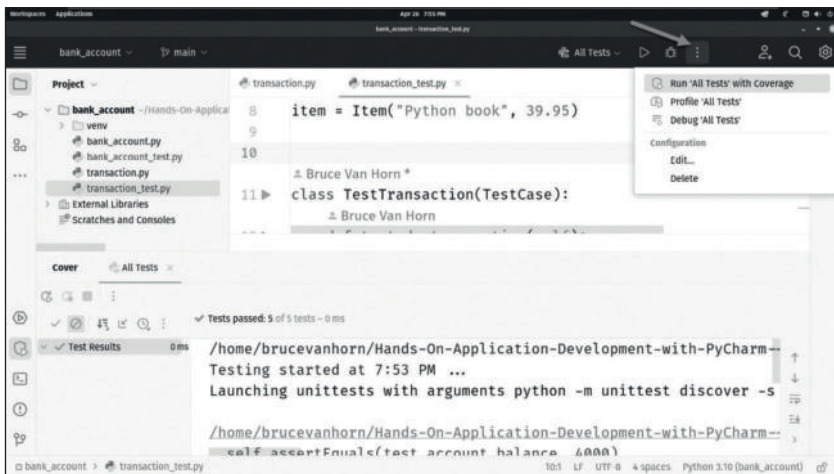


Рис. 6.23. Run 'All Tests' with Coverage позволяет запускать тесты и определять, какая часть вашего приложения не охвачена модульными тестами

Когда вы сделаете это в первый раз, вы, скорее всего, увидите сообщение об ошибке – не из вашего кода, а из PyCharm. Смотрите рис. 6.24, чтобы понять, что я имею в виду:

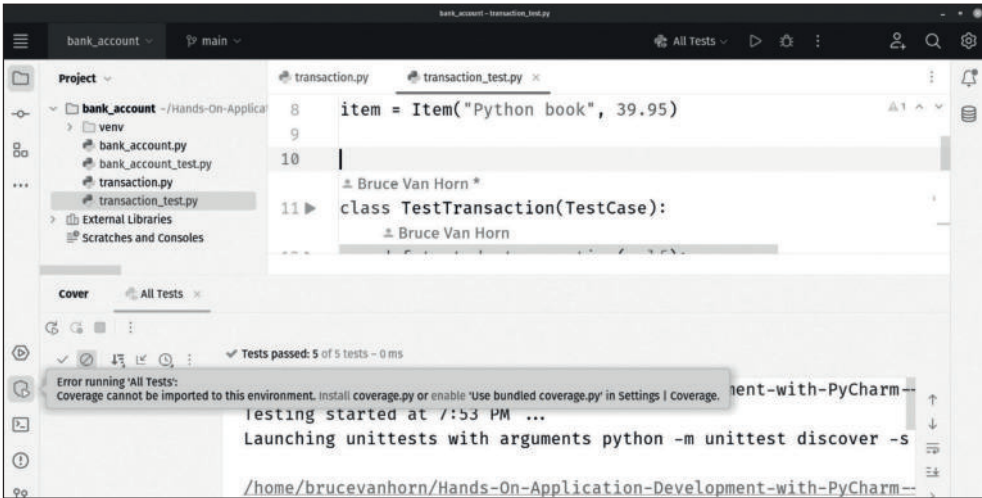


Рис. 6.24. При первом запуске тестового покрытия вы получите предупреждение, если вы не установили программное обеспечение для покрытия или не включили связанную копию

Для работы с покрытием требуется некое программное обеспечение, `coverage.py`, которое мы не установили. Здесь есть два варианта: вы можете либо добавить `coverage.py` в свой проект, либо использовать встроенную версию, поставляемую с PyCharm. Я предпочитаю использовать пакетную версию. Вы можете кликнуть слово *enable* в сообщении об ошибке, которое отображается в виде синей гиперссылки, и PyCharm включит этот параметр для вас. Если вы хотите управлять этим параметром самостоятельно, см. рис. 6.25, где можно найти этот параметр:

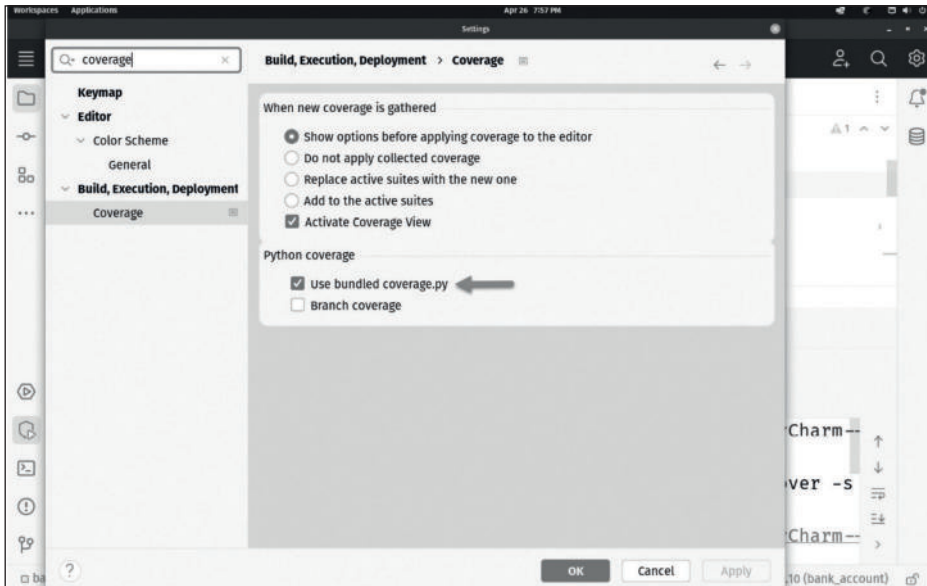


Рис. 6.25. Настройка использования встроенного файла `coverage.py` позволяет использовать его без необходимости добавления в проект

Включив `coverage.py`, перезапустите тест покрытия. Давайте посмотрим, как мы это делаем:

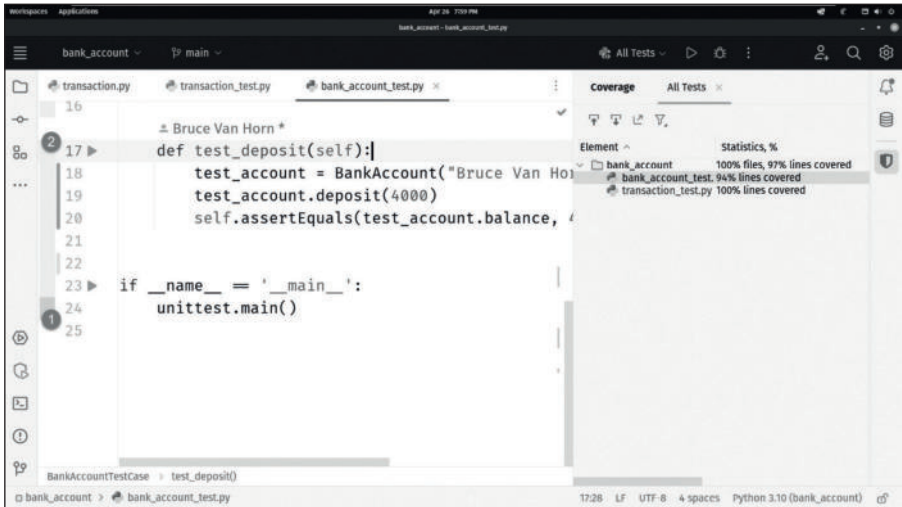


Рис. 6.26. Мне бы хотелось, чтобы мои оценки в колледже были так же хороши!

Боже правый! У нас стопроцентное покрытие в тестах транзакций, но жалкий провал в файле `bank_account_test.py` – т. е., если считать провалом покрытие в 94 %. Будучи перфекционистом, мне хотелось бы увидеть, как я упустил эти моменты. Я могу дважды кликнуть строку, показывающую 94 % в `bank_account_test.py`, и я окажусь в желобе с цветовой кодировкой. Здесь я еще раз должен извиниться за то, что книга напечатана в черно-белом режиме. Область 1 на рис. 6.26 окрашена в красный цвет. Это линии, которые не охвачены тестами. Честно говоря, я не помню, чтобы их вводил. Они мне не нужны, так как мой тестировщик выполняет мои тесты за меня. Я могу просто удалить эти строки и переформатировать файл с помощью `Ctrl + Alt + L / Cmd + Opt + L`. Повторите тест с покрытием. Мои результаты на рис. 6.27 показывают, что мы рядом.

Файл `bank_account_test.py` теперь имеет 100%-ное покрытие, но вверху я все еще вижу, что папка `bank_account` имеет только 98%-ное покрытие. Это никуда не годится! Прямо сейчас у меня закрыто окно обозревателя проекта, чтобы освободить место для окна редактора и покрытия. Если я открою его опять и запущу покрытие, то получу больше информации. На рис. 6.28 показано, куда нам следует смотреть. Файл `bank_account.py` имеет покрытие только 92 %. После двойного щелчка мыши, чтобы открыть его, я увижу пропущенные строки красного цвета, как показано на рис. 6.28:

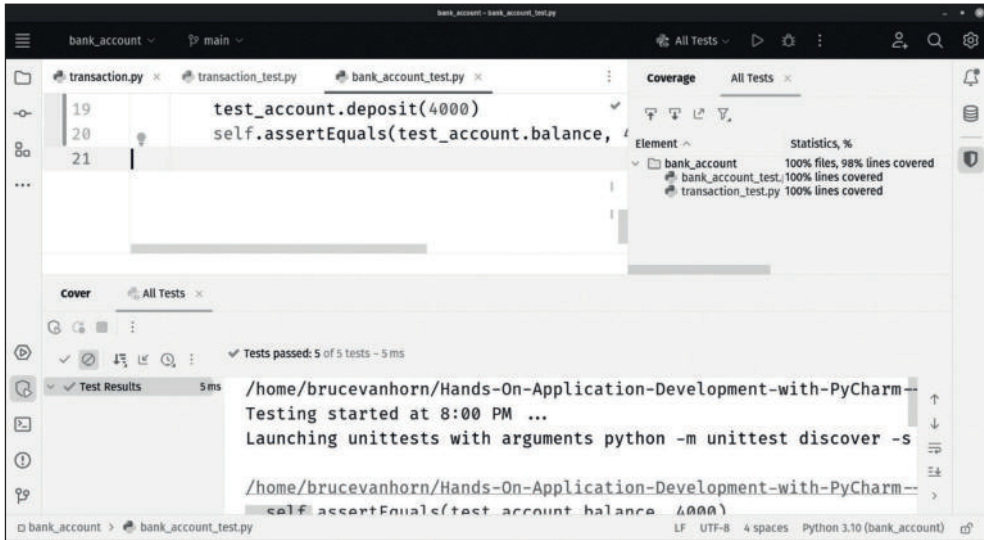


Рис. 6.27. Мы добились покрытия 100%! Продолжайте в том же духе, и вы можете получить повышение

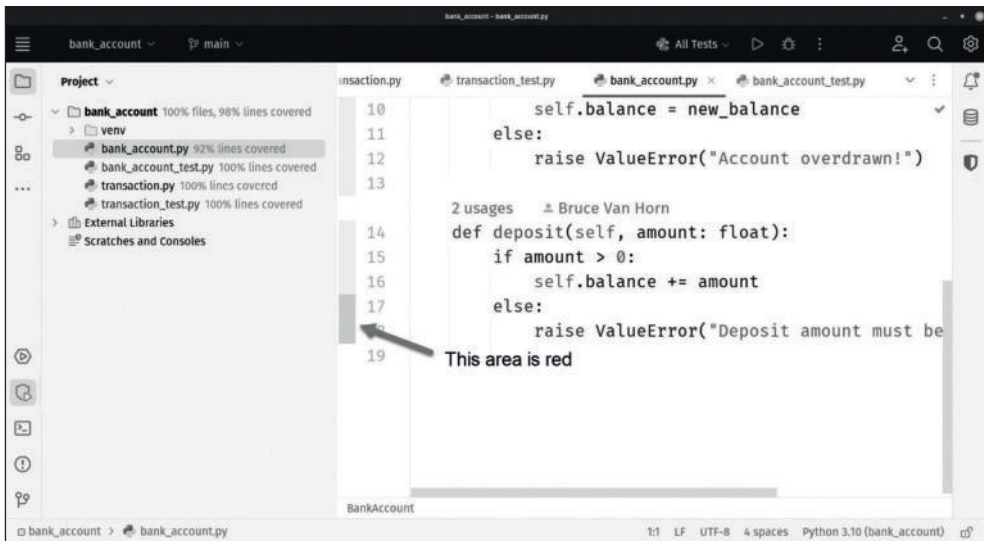


Рис. 6.28. Серовато-красная область в поле означает, что строки 17 и 18 не покрыты каким-либо модульным тестом

Похоже, нам нужно написать еще один модульный тест. Я забыл написать тест на наличие ошибки. Как вы помните, я оставил проверку депозита как вызов для вас. Я написал это в своем коде для этой книги, но забыл написать тест на ошибки. Покрытие нас спасет!

Откройте `bank_account_test.py` и добавьте следующий тест, который покроет случай попытки внести на счет отрицательную сумму:

```
def test_deposit_negative_number_fail(self):
    test_account = BankAccount("Bruce Van Horn", "123355-23434", 4000)
    self.assertRaises(ValueError, test_account.deposit, -2000)
```

Запустите тест и убедитесь, что он пройден, а затем повторно запустите конфигурацию **All Tests** с покрытием. Мой результат показан на рис. 6.29:

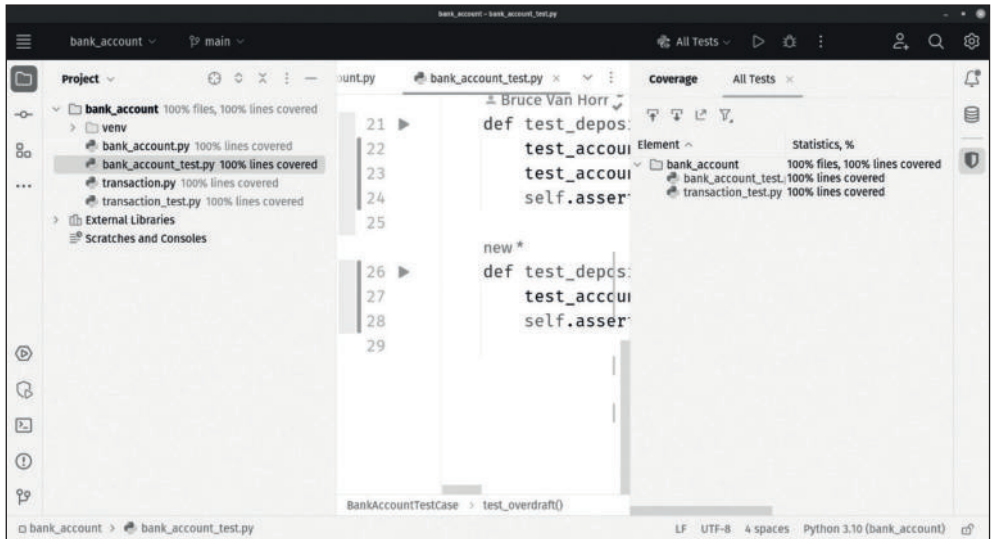


Рис. 6.29. Мы добились 100%-ного покрытия для всех файлов

На этот раз у нас – высший балл! Теперь, когда мы удовлетворены, я укажу на окно покрытия в правой части экрана. Оно показывает список результатов, которые мы уже видели. Обратите внимание на значок щита на правой панели инструментов. Вы можете показать или скрыть окно покрытия, кликнув этот щит.

Выходные данные тестового покрытия

В дополнение к графическому отображению PyCharm выводит отчет о прогоне покрытия. Вы увидите вывод, указанный в окне вывода, рядом с обычными тестовыми выводами. Мой утверждает следующее:

```
Wrote XML report to /home/brucevanhorn/.cache/JetBrains/PyCharm2023.1/coverage/
bank_account&All_Test.xml
```

XML-файл создается с помощью `coverage.py`, который мы включили ранее. Как вы уже могли догадаться, `coverage.py` – популярный инструмент Python для измерения покрытия кода во время тестовых запусков. Это инструмент с открытым исходным кодом, который помогает вам определить, какие части вашего кода Python используются вашими тестами, а какие нет. Инструмент работает путем сбора информации о том, какие строки кода выполняются во время тестового запуска, а затем создания отчета, показывающего

процент покрытия кода. Вывод XML используется PyCharm для рендеринга отображений пользовательского интерфейса с цветовой кодировкой, которые мы использовали. Вывод XML также может использоваться вашей системой непрерывной интеграции **continuous integration (CI)** для создания отчетов и отображений.

Компания JetBrains создала отличную систему CI под названием **TeamCity**, которая может использовать coveralls.com для ошибки сборки, если тестовое покрытие ниже установленного порога.

ПРОФИЛИРОВАНИЕ ПРОИЗВОДИТЕЛЬНОСТИ

Первый шаг в создании отличной программы – заставить ее работать полностью. Второй шаг – провести автоматическое тестирование, чтобы доказать, что программа работает именно так, как задумано. Последним шагом должна стать настройка кода так, чтобы программа работала как можно быстрее и эффективнее. Плохо работающие программы в лучшем случае рискуют иметь низкий уровень применения, а в худшем могут быть просто непригодны для использования. В Англии у Национальной службы здравоохранения есть алгоритм, разработанный для сопоставления реципиентов трансплантированных органов с недавно извлеченными органами. Алгоритм сложен, но чрезвычайно чувствителен ко времени. Собранные органы необходимо пересадить быстро; в противном случае их ткани погибнут и станут бесполезными. Короче говоря, алгоритм должен быть чрезвычайно точным; в противном случае пересаженный орган может быть отторгнут, что приведет к смерти пациента. Он также должен быть быстрым, так как орган потеряет жизнеспособность, что также может привести к смерти пациента. И я очень рад своей работе по планированию и прогнозированию мощности аппаратных систем. Никто еще не умирал из-за того, что мои запросы к базе данных были слишком медленными. По крайней мере, я не знаю об этом.

Помимо возможности запускать тесты с покрытием, вы также можете запускать их с профилированием производительности. В то время как отчет о покрытии графически показывает, какие области вашего кода остаются непроверенными, профилировщик PyCharm предоставляет отчеты о том, какие части вашего кода занимают львиную долю общего времени выполнения. Это позволяет выявлять узкие места и сосредоточить усилия по рефакторингу на повышении эффективности кода и его выполнения.

Подобно тому, как существует несколько библиотек тестирования, которые широко используются разработчиками Python, существует также множество инструментов профилирования, включая Yappi, cProfile и VMProf. PyCharm поддерживает их все, но они работают по-разному. cProfile встроен в Python, как и профилировщик по умолчанию. Yappi – это улучшение по сравнению с cProfile, поскольку оно позволяет профилировать многопоточные прило-

жения и поддерживает профилирование процессорного времени. VMProf поддерживает статистическую выборку. Когда вы профилируете с помощью этого инструмента, он не просто отслеживает время одного запуска вашей программы; вместо этого он выполнит несколько прогонов и выберет образцы, предоставляя вам более реалистичный профиль производительности. PyCharm будет использовать VMProf, если он доступен. Если нет, он будет искать Yarpri. Если он не сможет найти Yarpri, он будет использовать решение cProfile, встроенное в Python. В этой книге я буду использовать инструмент cProfile по умолчанию.

Профилирование в PyCharm

Код, который мы будем профилировать, можно найти в репозитории этой книги, в папке Chapter-06. Файл `profiling.py` содержит следующий код:

```
def custom_sum(n=1000000):
    result = 0
    for i in range(n):
        result += i
    return result

def built_in_sum(n=1000000):
    result = sum(range(n))
    return result

if __name__ == '__main__':
    print(custom_sum())
    # print(built_in_sum())
```

Этот код сравнит два способа вычисления суммы целых чисел в диапазоне от единицы до верхнего предела, выраженного как n , значение которого по умолчанию равно 1 000 000. Функция `custom_sum` перебирает все элементы, добавляя каждый к текущей сумме. Функция `built_in_sum` использует встроенный метод `sum()` Python.

В основной области мы будем использовать комментарии для переключения между двумя вызовами функций и тестирования обоих методов. Сначала рассмотрим нашу пользовательскую функцию суммирования, поэтому вызов `built_in_sum` пока закомментирован.

Типичное утверждение состоит в том, что встроенные функции обычно работают быстрее, чем любой написанный вами код. В этом примере мы сможем проверить это утверждение и дополнительно уточнить его с помощью статистики времени выполнения в процессе профилирования. Давайте начнем.

Как и в случае с тестированием и покрытием, мы можем запустить профилирование, используя либо зеленые стрелки в редакторе, либо кнопку с многоточием в верхней части экрана. На рис. 6.30 показаны оба варианта:

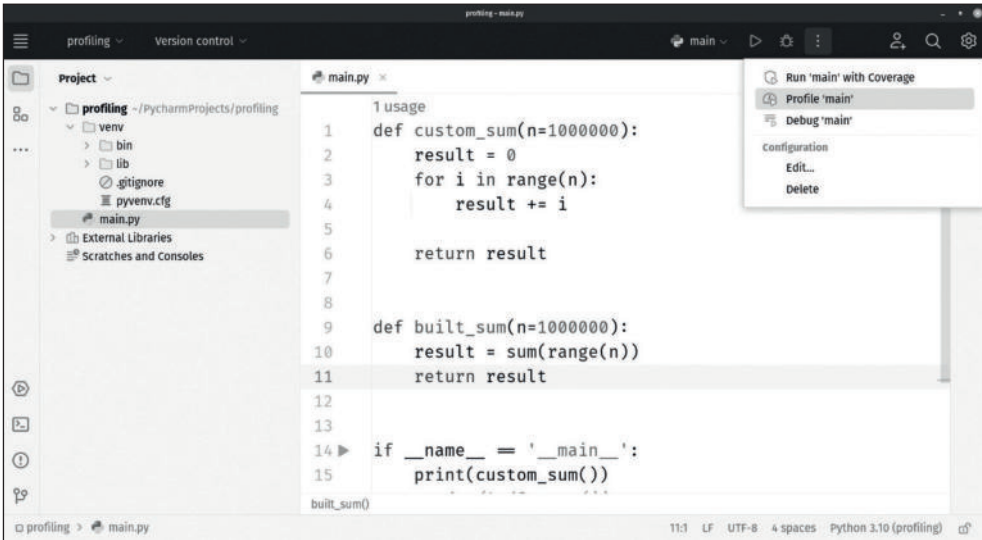


Рис. 6.30. Вы можете запустить профиль, используя меню с многоточием в правом верхнем углу или кликнув зеленую стрелку рядом с точкой входа dunder-main в строке 14

Когда запуск профиля будет завершен, нам будет предоставлен отчет о производительности, как показано на рис. 6.31:

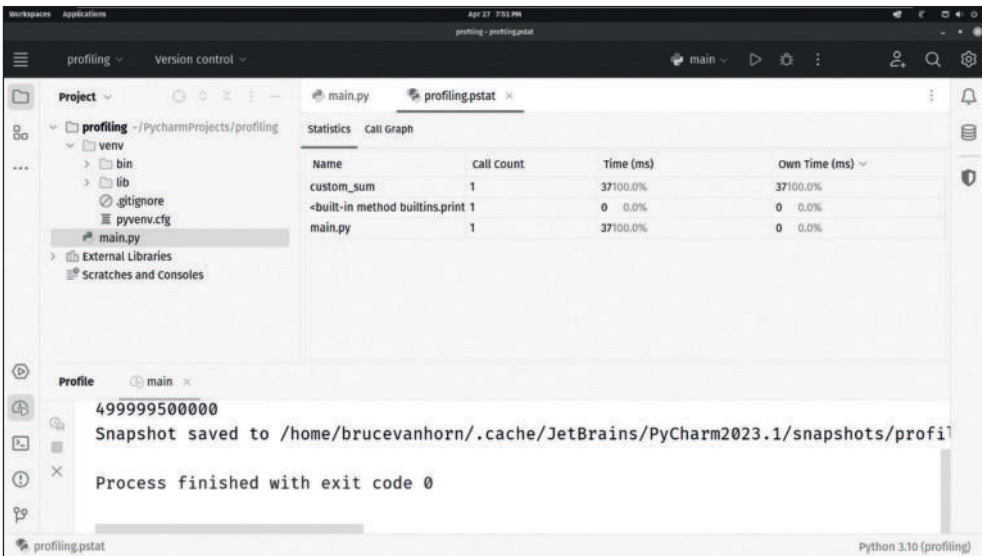


Рис. 6.31. Профиль производительности для функции custom_sum

На моем компьютере, который в данном случае представляет собой виртуальную машину VMWare с очень скромной конфигурацией (2 ядра, 4 Гб ОЗУ и вращающийся диск со скоростью 7200 об/мин), функция `custom_sum` была выполнена за 41 мс. На моем дисплее время и процент сгруппированы вместе, но мы видим, что 100 % времени было потрачено на функцию `custom_sum`. Если бы это была более сложная программа с множеством функций, вызываемых во время выполнения, мы бы увидели полный список каждой функции и количество времени, затраченного на каждую. Обратите внимание на столбец **Own Time** и столбец **Time**.

В профилировщике производительности PyCharm столбец **Time** показывает общее время, затраченное на выполнение определенной функции или метода, включая время, затраченное на выполнение любых подфункций или методов, вызванных внутри нее.

С другой стороны, в столбце **Own Time** показано время, затраченное на выполнение только кода внутри самой функции или метода, исключая любое время, затраченное на выполнение подфункций или методов. Это означает, что столбец **Own Time** может дать вам лучшее представление о производительности кода внутри конкретной функции или метода, независимо от каких-либо внешних факторов, таких как производительность других функций или методов, которые она вызывает.

Чтобы проиллюстрировать разницу, рассмотрим функцию `A()`, которая вызывает две другие функции: `B()` и `C()`. Если вы посмотрите на столбец **Time** для `A()`, он будет включать время, затраченное на выполнение как `B()`, так и `C()`, в дополнение к времени, затраченному на выполнение кода внутри самого `A()`. Однако, если вы посмотрите на столбец **Own Time** для `A()`, он покажет только время, затраченное на выполнение кода внутри `A()`, а не время, затраченное на выполнение `B()` и `C()`.

В общем, столбец **Time** может дать вам представление об общем влиянии на производительность конкретной функции или метода, а столбец **Own Time** может помочь вам сосредоточиться на производительности кода внутри этой функции или метода.

Сравнение производительности и встроенной функции `sum()`

Давайте посмотрим, как мое время выполнения 72 мс соотносится со встроенной функцией `sum()` в Python. Измените нижнюю часть файла `main.py`, закомментировав функцию `custom_sum` и функцию `built_in_sum`, например:

```
if __name__ == '__main__':
    # print(custom_sum())
    print(built_in_sum())
```

Запустите профиль с этой конфигурацией. Вы можете увидеть мой результат на рис. 6.32:

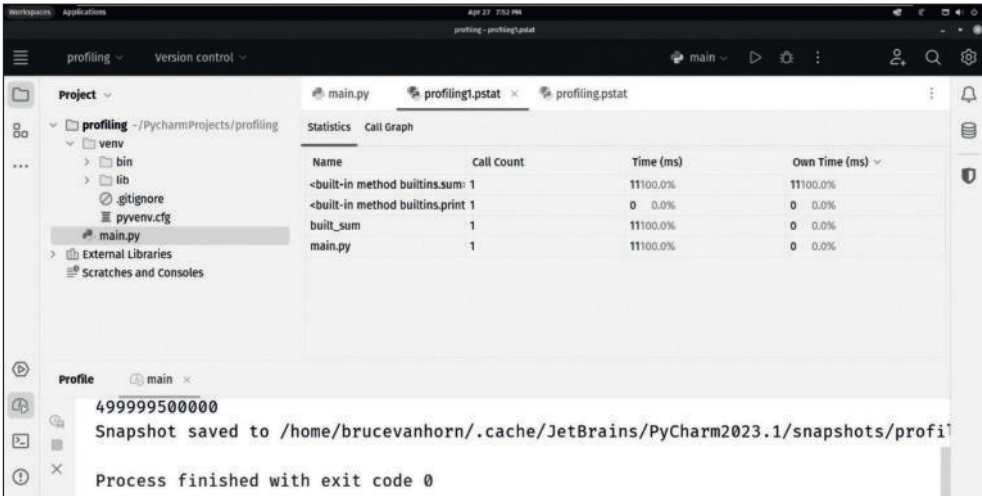


Рис. 6.32. Встроенная функция суммы работает значительно быстрее – 11 мс

Никакого сравнения! На моем компьютере встроенная функция `sum()` бежит в семь раз быстрее! В реальной жизни я советую запускать каждый профиль несколько раз и брать среднее значение, поскольку время выполнения может варьироваться. В моем случае последующие запуски функции `built_in_sum` варьировались от 11 до 26 мс, что является довольно большим разбросом.

Просмотр графа вызовов

Помимо таблицы статистики, вы также можете просмотреть профиль в виде графа вызовов. Эта схема представляет собой древовидное представление выполнения вашей программы, как показано на рис. 6.33:

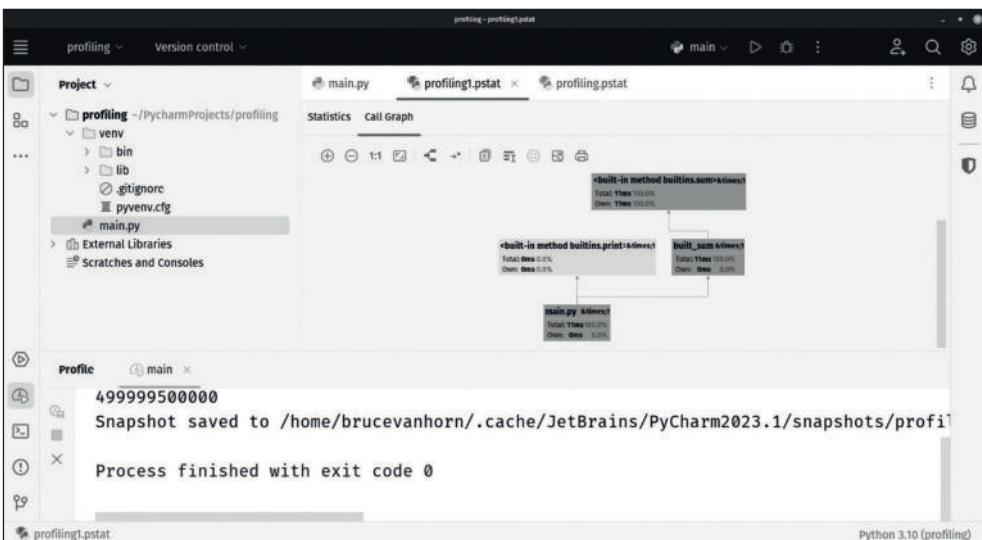


Рис. 6.33. Граф вызовов показывает древовидное представление выполнения программы

Узлы графа вызовов заштрихованы зеленым и красным. Чем темнее оттенок красного, тем больше времени было потрачено на функцию, указанную этим узлом. На рис. 6.33 почти все время тратится на функцию `custom_sum`, которая имеет темно-красный цвет (поверьте мне). Встроенный метод `print` занимает небольшое, но ненулевое время, когда печатает сумму в основной функции.

Навигация с помощью профиля производительности

Вы можете перейти к функции, используя либо таблицу статистики, либо соответствующий узел графа вызовов. Просто кликните правой кнопкой мыши, как показано на рис. 6.34:

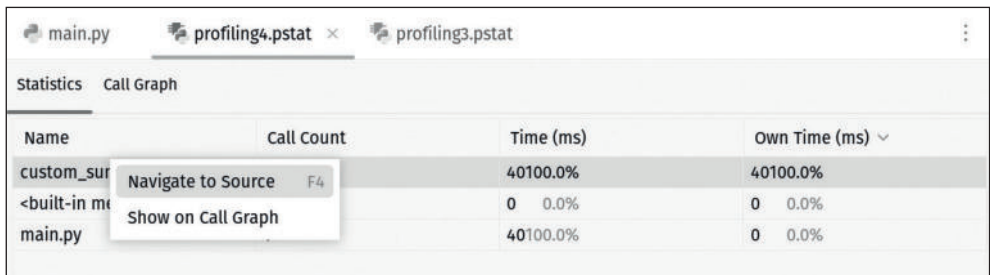


Рис. 6.34. Вы можете перейти к своему коду, кликнув функцию правой кнопкой мыши и выбрав `Navigate to Source`

Вы можете сделать то же самое на графе вызовов. Кликнув правой кнопкой мыши узел на графе вызовов, вы получите ту же опцию навигации, которая приведет вас к источнику. Это поможет вам перейти прямо к любому коду, который вы захотите проверить.

Скриншоты профиля производительности cProfile

Когда вы запускаете профиль с помощью `cProfile`, PyCharm сохранит для вас **cProfile snapshot** или файл **pstat**. Вы можете увидеть это в окне вывода. В моем случае файлы `.pstat` создаются в моей домашней папке:

Скриншот сохранен в `/home/brucevanhorn/.cache/JetBrains/PyCharm2023.1/snapshots/profiling4.pstat`.

Когда я занимаюсь серьезной работой по профилированию, я часто копирую эти файлы в более удобную папку и изменяю их имена, чтобы указать условия, при которых они запускались. Например, в нашем примере я мог бы назвать первый файл `.pstat` примерно так: `custom_sum_Performance_1.pstat`; второй мог бы называться `built_in_sum_performance_1.pstat`.

Я делаю это для того, чтобы у меня был базовый профиль производительности для каждого из них. Я подозреваю, что в реальной жизни у вас редко будет такая простая альтернатива тому, что мы здесь представили. Скорее всего, у вас будет несколько версий функции, использующих разные подходы к разработке алгоритма. В таких случаях сохранение ваших файлов `.pstat`, чтобы вы могли сравнить их с будущими запусками, может быть очень удобным хотя бы по одной причине, кроме как похвалиться на следующей аттестации сотрудников.

Вы можете открыть свои старые файлы `.pstat` с помощью меню **Tools**, как показано на рис. 6.35:

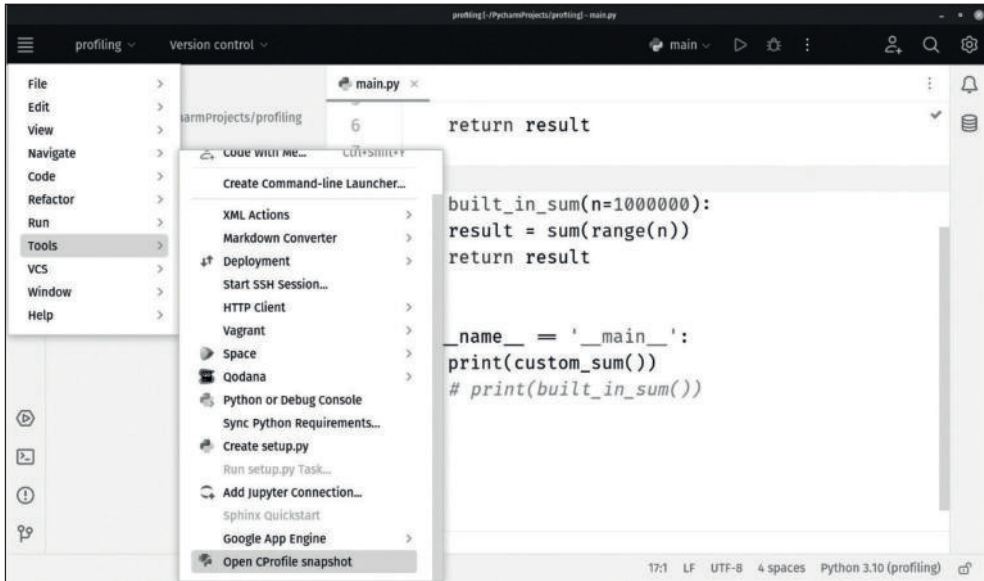


Рис. 6.35. Вы можете открыть свои старые скриншоты через меню Tools

Открытие этого файла `.pstat` покажет таблицу статистики и график вызовов. Если вы провели рефакторинг имен функций, не стоит ожидать, что навигация по-прежнему будет работать; однако вы можете просмотреть старые результаты и сравнить их с новыми.

В целом способность PyCharm открывать и сравнивать старые версии файлов `.pstat` может быть полезным инструментом для отслеживания производительности вашего кода с течением времени и определения областей, где можно улучшить производительность.

КРАТКОЕ СОДЕРЖАНИЕ

Тестирование, отладка и профилирование – это задачи высокого уровня, которые мы можем использовать для анализа приложений в поисках улучшений в корректности и производительности, но они могут сбивать с толку начинающих разработчиков. PyCharm предлагает простые и интуитивно понятные интерфейсы для этих процессов, что делает их более доступными и оптимизированными.

Модульное тестирование – это процесс проверки того, что отдельные компоненты большой системы работают должным образом. PyCharm имеет удобные команды для создания тестовых скелетов и шаблонного кода, написание которых вручную разработчикам обычно требует от разработчиков конкретного времени. При тестировании программы важно учитывать ожидаемые ошибки, а также очевидные проверки предполагаемой функциональности.

В сеансе отладки разработчики пытаются сузить круг поиска и выявить причины ошибок и багов, обнаруженных во время тестирования. Благодаря гра-

фическому интерфейсу в сочетании с различными опциями для отслеживания значений переменных в программе PyCharm позволяет нам динамически отлаживать наши программы со значительной свободой. Различные пошаговые функции также предоставляют нам гибкий способ пошагового продвижения по программе, которую мы отлаживаем.

Наконец, цель профилирования – анализ производительности программы и поиск способов ее улучшения. Это может включать поиск более быстрых способов вычисления значений или выявление узкого места в этом вычислении. Благодаря возможности генерировать исчерпывающую статистику о времени выполнения каждой выполняемой функции, а также графов вызовов PyCharm помогает разработчикам с легкостью перемещаться по различным компонентам профилируемой программы.

Эта глава также отмечает завершение второй части книги, в которой мы сосредоточились на повышении продуктивности нашей разработки. Отсюда мы будем рассматривать использование PyCharm в более специализированных областях, а именно в веб-разработке и проектах науки о данных.

В следующей главе рассмотрим основы трех универсальных языков веб-разработки – JavaScript, HTML и CSS – в контексте PyCharm.

Вопросы

Ответьте на следующие вопросы, чтобы проверить свои знания, полученные в этой главе.

1. Что такое тестирование в контексте разработки программного обеспечения? Каковы различные методы тестирования?
2. Как PyCharm поддерживает процессы тестирования?
3. Что такое отладка в контексте разработки программного обеспечения?
4. Как PyCharm поддерживает процессы отладки?
5. Что такое профилирование в контексте разработки программного обеспечения?
6. Как PyCharm поддерживает процессы профилирования?
7. Каково значение стрелок запуска в редакторе PyCharm?

Дальнейшее чтение

Чтобы узнать больше о темах, рассмотренных в этой главе, посетите следующие ресурсы:

- *Agile Software Development, Principles, Patterns, and Practices*, Martin, R. C. (2003). Prentice Hall;
- *Clean Architecture: A Craftsman's Guide to Software Structure and Design*, Martin, R. C. (2017). Prentice Hall;
- *Real-World Implementation of C# Design Patterns*, Van Horn, B and Symons, V. (2022).
- Packt Publishing.

Часть III

Веб-разработка в PyCharm

Эта часть книги посвящена процессам веб-разработки в программировании на Python и поддержке, которую PyCharm предлагает для веб-проектов. Читатели смогут использовать PyCharm и ее функции для эффективной разработки своих веб-приложений.

Эта часть состоит из следующих глав:

- глава 7 «Веб-разработка с JavaScript, HTML и CSS»,
- глава 8 «Создание динамических сетевых приложений с Flask»,
- глава 9 «Создание RESTful API с помощью FastAPI»,
- глава 10 «Полнофункциональные фреймворки – Django и Pyramid»,
- глава 11 «Понимание управления базами данных в PyCharm».

Веб-разработка с JavaScript, HTML и CSS

Эта глава знаменует собой начало серии из пяти глав, посвященных веб-программированию с помощью PyCharm, каждая из которых будет посвящена разработке общих веб-приложений. Я с нетерпением ждал возможности написать этот раздел, потому что это мой хлеб с маслом. Я занимаюсь разработкой веб-приложений с тех пор, как существует интернет. Профессиональная версия PyCharm – настоящее удовольствие для веб-разработчиков: полная копия **WebStorm** от JetBrains, которая поставляется через предустановленный плагин. Это означает, что мы получаем гораздо больше, чем просто подсветку синтаксиса для **HTML**, **CSS** и **JavaScript**! У нас также есть полный доступ к инструментам **Node.js** и современным платформам веб-интерфейса, таким как **React**. У меня может возникнуть соблазн назвать это излишним, но это не так. Продукт, над которым я работаю каждый день, представляет собой смесь **микросервисов**, написанных на **Python 3**, **NodeJS** и **React** для внешнего интерфейса. Я использую три разные базы данных: **Microsoft SQL Server**, **MongoDB** и **Redis**. Мне никогда не придется покидать PyCharm! Как я только что упомянул, у меня есть надежные встроенные инструменты для JavaScript, NodeJS и React. Все базы данных поддерживаются в PyCharm, о чем мы поговорим в главе 11. Я могу создавать конфигурации запуска отладки в PyCharm, запускать несколько сервисов, которые взаимодействуют друг с другом через вызовы REST, и выполнять межсервисную отладку. Я также могу разместить точку останова во внешнем интерфейсе React, а другую – в отдельном проекте, используя **Flask** или **FastAPI**; когда я работаю над рабочим процессом приложения, мои точки останова останавливают работу независимо от того, в каком проекте я работаю и какой язык используется в проекте. Я бы не стал пытаться сделать это ни в какой другой IDE. Я уже упоминал, как был рад начать писать эту группу глав? Я забегаю вперед.

Темы, которые будут обсуждаться в этой главе, включают интеграцию распространенных языков веб-программирования (JavaScript, HTML и CSS) в PyCharm

и простые и интуитивно понятные способы разработки с их помощью. К концу этой главы вы получите всесторонние знания о том, как использовать эти три языка, и сможете начать проект веб-разработки с использованием PyCharm.

В этой главе будут рассмотрены следующие темы:

- внедрение JavaScript, HTML и CSS в процесс веб-разработки,
- варианты работы с кодом JavaScript, HTML и CSS в PyCharm,
- как реализовать интерактивное редактирование и отладку веб-проектов,
- как работать с шаблонными параметрами HTML в PyCharm.

ТЕХНИЧЕСКИЕ ТРЕБОВАНИЯ

Ниже приведены предварительные условия для этой главы:

- рабочая установка Python 3.10 или новее;
- рабочая установка PyCharm Professional. Если вы используете версию Community, большая часть того, что мы рассмотрим в этой главе, не будет работать, поскольку вы получите лишь ограниченную поддержку HTML. Вы по-прежнему сможете работать с файлами CSS и JavaScript, но возможности будут очень ограничены по сравнению с профессиональной версией;
- веб-браузер Chrome. Он понадобится вам, если вы хотите отлаживать код JavaScript, работающий в браузере;
- рабочая установка NodeJS и **менеджер пакетов узлов (npm)**. Это не обязательно. Они понадобятся вам только в том случае, если вы хотите работать с React или современными платформами JavaScript, такими как Angular, Vue или Express. React – это сложная тема, и она не будет подробно рассмотрена, кроме настройки и работы с проектом React;
- я покажу вам, как использовать функции развертывания PyCharm. Для этого я предполагаю, что вы понимаете, как развернуть веб-проект на удаленном хосте с помощью такого инструмента, как WinSCP или FileZilla, или как передавать файлы с помощью инструментов командной строки, таких как **защищенное копирование (SCP)** или **протокол передачи файлов (FTP)**;
- вы можете найти пример кода для этой главы по адресу <https://github.com/PacktPublishing/Hands-On-Application-Development-with-PyCharm---Second-Edition/tree/main/chapter-07>. Мы рассмотрели, как клонировать репозиторий, в главе 2.

ВВЕДЕНИЕ В HTML, JAVASCRIPT И CSS

Я называю эти языки *триумvirатом веб-разработки*. Они дают вам самые базовые навыки, которые вы можете освоить по мере продвижения к тому, чтобы стать полноценным веб-разработчиком. Термин **фулстек** применительно к фулстек-разработчику просто означает, что вы обладаете навыками разработки клиентской части приложения, а также серверной части и базы данных. Следующие шесть глав посвящены полнофункциональной веб-разработке с помощью PyCharm.

Строго говоря, только один из трех языков, которые мы здесь рассмотрим, является языком программирования. **Язык гипертекстовой разметки (HTML)** используется для создания структуры и макета веб-страницы или пользовательского интерфейса приложения. **Каскадные таблицы стилей (CSS)** используются для управления внешним видом пользовательского интерфейса, соблюдая при этом разделение задач: мы разделяем расположение кнопок, текста и интерактивных элементов на основе определений визуального внешнего вида. Ни HTML, ни CSS сами по себе не способны создать какой-либо уровень интерактивности, за исключением нескольких трюков CSS, таких как изменение цвета кнопки, когда пользователь наводит на нее курсор.

Настоящая интерактивность в веб-интерфейсе обеспечивается JavaScript, или, как его правильно называют, ECMAScript. Около миллиона лет назад компания Netscape боролась за сердца и умы зарождающегося сообщества пользователей веб-браузеров. Это была битва Давида против Голиафа, где Голиафом была Microsoft. Лицензия на браузер Netscape стоила немного, в то время как Microsoft Internet Explorer был бесплатным, встроенным в операционную систему Windows и имел набор инструментов, которые позволяли IT-менеджерам централизованно настраивать поведение своего браузера, одновременно расширяя его среди крупных корпоративных пользователей. Подобного предложения от Netscape не существовало. Им нужен был переломный момент.

Примерно в это же время компания Sun Microsystems активно продвигала свой новый флагманский язык программирования под названием **Java**. Как и Netscape, Sun страдала от конкуренции со стороны Microsoft, поэтому они объединились. Netscape начала создавать то, что впоследствии стало языком программирования для браузеров, под названием ECMAScript, в то время как Sun запатентовала название Java, понимая, что недавно переименованный JavaScript никогда не будет работать где-либо, кроме как в браузере. Конечная цель Sun заключалась в том, чтобы убедиться, что все используют Java для создания серверной части своих веб-приложений, а Netscape завоевала долю рынка внешнего рендеринга, предлагая свой интерактивный опыт. JavaScript не имеет ничего общего с Java, кроме маркетинга. Это очень разные языки, и их никогда не следует путать.

Я предполагаю, что у вас есть некоторые знания в огромном мире разработки HTML. И также хочу напомнить вам, что эта книга посвящена PyCharm, а не веб-разработке. Учитывая то, как я зарабатываю на жизнь, мой опыт здесь глубок, и я довольно быстро перейду от «это HTML-тег» (это абсолютные основы) к «а вот React» (т. е. там будут драконы, если вы работали только на Python). Единственное, о чем я сожалею, – это то, что Packt не позволил мне написать «там будут драконы» шрифтом в стиле фильмов ужасов. Я оставляю вам ссылки на книги и другие ресурсы в разделе «Дальнейшее чтение» этой главы на случай, если вы захотите узнать больше об интерфейсной части или полнофункциональной разработке в целом.

Написание кода с помощью HTML

HTML, как и любой другой код, создается в текстовом файле. В случае HTML текст строго структурирован с использованием набора HTML-тегов. Вы мо-

жете думать о тегах как о ключевых словах в языке программирования, за исключением того, что они отделяются угловыми скобками. Помните, что HTML предназначен для структуры и макета контента. Он был создан Тимом Бернерсом-Ли (а не Элом Гором) как способ представления статей в научных журналах о предшественнике интернета, называемом **Сетью Агентства перспективных оборонных исследовательских проектов (DARPANET)**. Это подразделение Министерства обороны США. Одной из проблем в научном мире было время, необходимое для публикации рецензируемой журнальной статьи. В тех случаях, когда наука должна была быстро развиваться, нам нужен был формат электронной публикации исследований без прохождения рецензирования и процесса публикации в печатном виде, который может занять месяцы. HTML был разработан для электронной имитации бумажной публикации. Поскольку это так, неудивительно, что структура элементов HTML относится к **объектной модели документа (DOM)**. Документ организован по абзацам, заголовкам, подзаголовкам, разделам, рисункам, изображениям и т. п. Рассмотрим следующий пример:

```
<html>
  <head>
    <title>Advances in the Application of Time Travel</title>
  </head>
  <body>
    <h1>Introduction</h1>
    <p> Lorem ipsum dolor sit amet, consectetur adipiscing
      elit. Vestibulum tincidunt tempus lectus vitae
      euismod.</p>
  </body>
</html>
```

Это HTML-документ, структурированный как типичная исследовательская работа. У него есть заголовок, который отображается на вкладке браузера. `<title>` находится в теге `<head>` документа, который также может содержать метаданные и ссылки на файлы CSS и JavaScript, используемые на странице.

Тег `<body>` включает в себя содержимое документа. Голова и тело заключены в HTML-тег. Помните, что теги заключаются в угловые скобки, например `<body>`. У каждого тега есть открывающий и закрывающий тег, между которыми находится контент. Например, заголовок открывается с помощью `<h1>`, вставляется содержимое *Введение*, и тег закрывается с `</h1>`. В закрывающем теге добавляется косая черта к соответствующему элементу.

Эта структура позволяет веб-браузеру легко анализировать документ и отображать его содержимое в виде электронной страницы. Однако в современном HTML определена только структура. Расположение элементов на странице, а также визуальные определения шрифтов, цветов, размеров и т. д. контролируются внешним связанным документом CSS.

Если мы добавим этот код в наш предыдущий документ в тег `<head>`, мы получим страницу, которая в браузере выглядит совсем по-другому:

```
<head>
  <title>Advances in the Application of Time Travel</title>
  <link rel="stylesheet" href="mystyle.css">
</head>
```

Обратите внимание, что добавленный тег ссылки немного отличается: у него нет закрывающего тега. В HTML есть несколько подобных исключений. Кроме того, к тегу ссылки прикреплены некоторые атрибуты. Вы можете думать об атрибутах как об аргументах функции. Они определяют дополнительные входные данные, используемые тегом. В этом случае в HTML возможно несколько различных типов тегов ссылок. Мы определяем это как таблицу стилей, используя атрибут `rel`. Атрибут `href` сообщает HTML-странице, где найти файл CSS. Здесь страница будет искать файл с именем `mystyle.css` в той же папке, что и файл HTML.

Создание HTML в PyCharm

Опыт разработки HTML-интерфейса можно получить только в PyCharm Professional. Версия Community позволяет создавать HTML-файлы в проекте Python, но это все. В этой главе я буду иметь в виду исключительно профессиональную версию.

Мы собираемся создать новый проект, но на этот раз не проект Python. На самом деле мы вообще не собираемся создавать новый проект в PyCharm. Раньше в PyCharm была возможность создать пустой проект. В какой-то момент его удалили, и это нормально. Проекты PyCharm – это просто папка с папкой `.idea` внутри нее.

Создание пустого проекта

Создать пустой проект, который будет обходить процесс настройки ненужной нам виртуальной среды, несложно. Просто создайте папку где-нибудь на своем компьютере. Я создал свой на рабочем столе и назвал его `html-project`. Мне нравится называть свои проекты, которые не используют Python, в кебаб-регистре (регистре шашлыка), а не в змеином регистре¹, чтобы я мог легко заметить разницу. Естественно, вы можете называть свои проекты как угодно. Убедитесь, что у PyCharm нет открытого проекта. Если это так, используйте **File | Close**, чтобы закрыть текущий проект.

Затем просто перетащите пустую папку в PyCharm. Вы увидите обычное диалоговое окно **Trust and Open Project**, как показано на рис. 7.1:

¹ Kebab case, Snake case – стили именования с использованием различных регистров букв, например: camelCase (dromedaryCase), PascalCase (CamelCase, StudlyCase), snake_case (pothole_case), SCREAMING_SNAKE_CASE (MACRO_CASE, CONSTANT_CASE), kebab-case (dash-case, lisp-case), TRAIN-CASE (COBOL-CASE, SCREAMING-KEBAB-CASE), Train-Case (HTTP-Header-Case), flatcase. – *Прим. ред.*

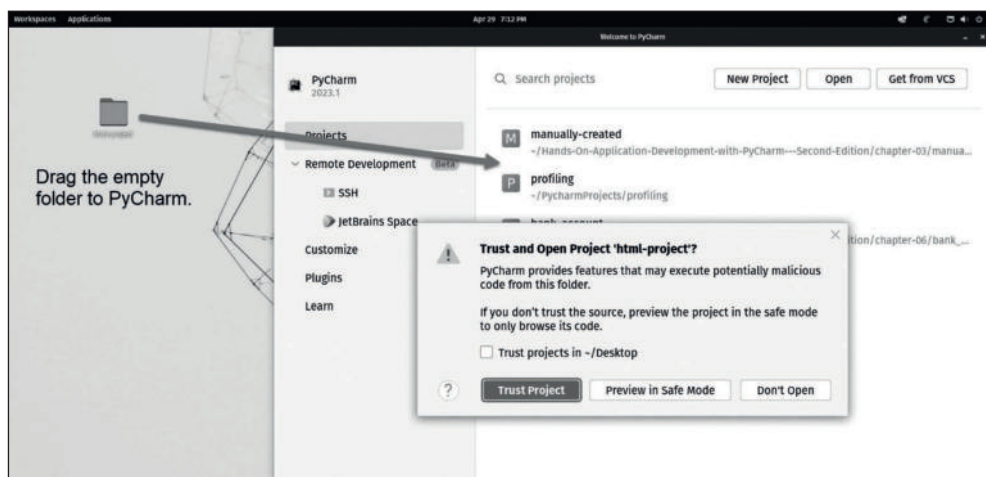


Рис. 7.1. Перетащите пустую папку в окно проекта PyCharm, чтобы создать пустой проект без среды Python

Возможно, вы заметили, что в новом окне проекта PyCharm Professional есть типы проектов HTML, и задаетесь вопросом, почему я их игнорирую. Я бы предпочел начать с простого примера, прежде чем покажу шаблонный проект HTML, который создает целый сайт на основе идеалов, установленных Google. Давайте прогуляемся, прежде чем полетим. Если вы уже являетесь фулстек-ветераном, не пропускайте этот раздел. Поскольку это не книга по HTML, я не собираюсь задерживаться на основах. Я собираюсь рассмотреть несколько функций в этом простом примере.

Когда вы создаете проект таким образом, это фактически проект Python без интерпретатора. Вы получите сообщение о том, что в качестве среды используется установка Python по умолчанию, и это нормально. Нам не понадобится это:

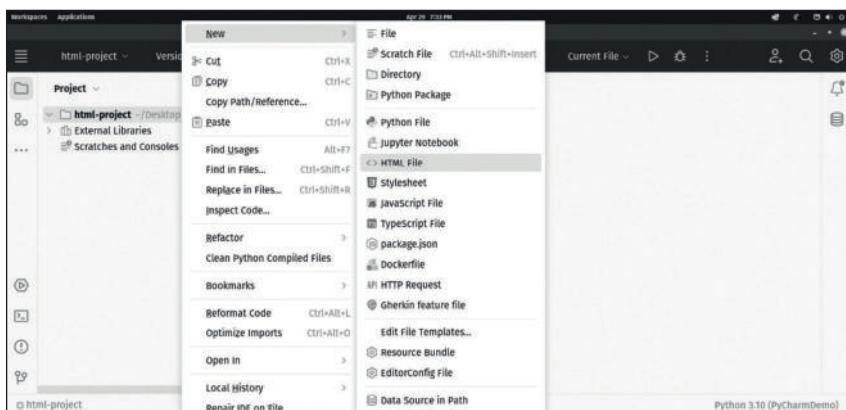


Рис. 7.2. Кликните правой кнопкой мыши папку проекта, чтобы создать новый HTML-файл

Вам будет предложено дать файлу имя. Назовите его `index.html`. Это файл по умолчанию, который будет отображаться на веб-сервере, поэтому это наи-

более распространенный файл, который вам нужно создать в первую очередь. PyCharm сгенерирует базовую структуру HTML-страницы на основе шаблона. Более того, PyCharm предложит вам заполнить основные элементы шаблона в процессе создания. Взгляните на рис. 7.3:

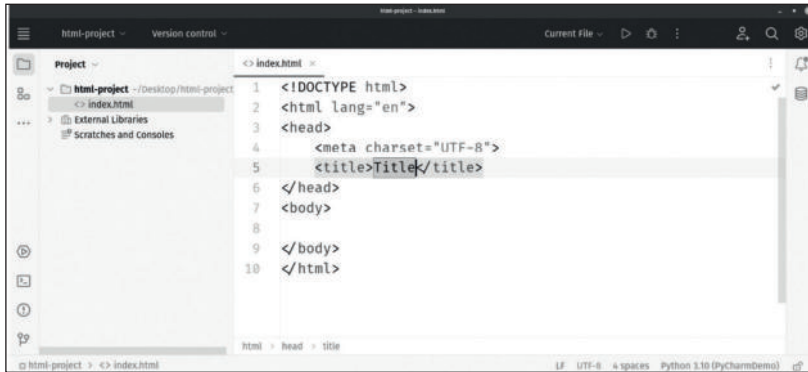


Рис. 7.3. PyCharm представляет шаблон для заполнения

В этом шаблоне есть только одна переменная шаблона: `Title`. Как видите, строка 5 выделена, а слово `Title` выделено еще больше. PyCharm ожидает, что вы введете заголовок, а затем нажмете клавишу `Tab`. При этом задается заголовок, а затем вы сразу попадаете внутрь тега `body`.

После добавления этого кода становятся очевидными несколько вещей.

1. Все функции автозаполнения, которые вы ожидаете от Python, также применяются к тегам HTML. Допустимые атрибуты также предоставляются при дополнении кода. Обратите внимание это, когда введете тег ссылки для таблицы стилей.
2. PyCharm автоматически создаст для вас закрывающий тег. Когда вы вводите `<h1>Introduction`, как только вы заполняете тег `<h1>`, PyCharm вставит за вас `</h1>` ближе. Если вы не закроете теги, это может привести к проблемам с макетом представления или потере контента.
3. Те же функции документации для ключевых слов и библиотек Python существуют и для HTML. Наведите курсор на любой элемент, и вы увидите документацию для этого элемента.
4. Тот же линтинг, который мы видели в коде Python, применяется и в коде HTML. На рис. 7.4 показаны те же предупреждения в тех же областях, которые мы видели, когда говорили об этих функциях в редакторе кода Python.

Сумма проблем в верхней части экрана (1) указывает на наличие нескольких опечаток. На самом деле их нет, просто словарь предполагает английский, а мой заполнитель – на латыни. Однако есть предупреждение. Если кликнуть желтый треугольник рядом с 1, откроется окно проблемы (3). Я также вижу желтый предупреждающий маркер в позиции 2. Все это указывает на то, что нам еще предстоит создать файл CSS, указанный в строке 6.

В дополнение к тому, что мы обычно видим в редакторе, мы можем увидеть кое-что новое в 4: здесь есть несколько значков, соответствующих веб-браузерам, с которыми PyCharm взаимодействует на вашем компьютере.

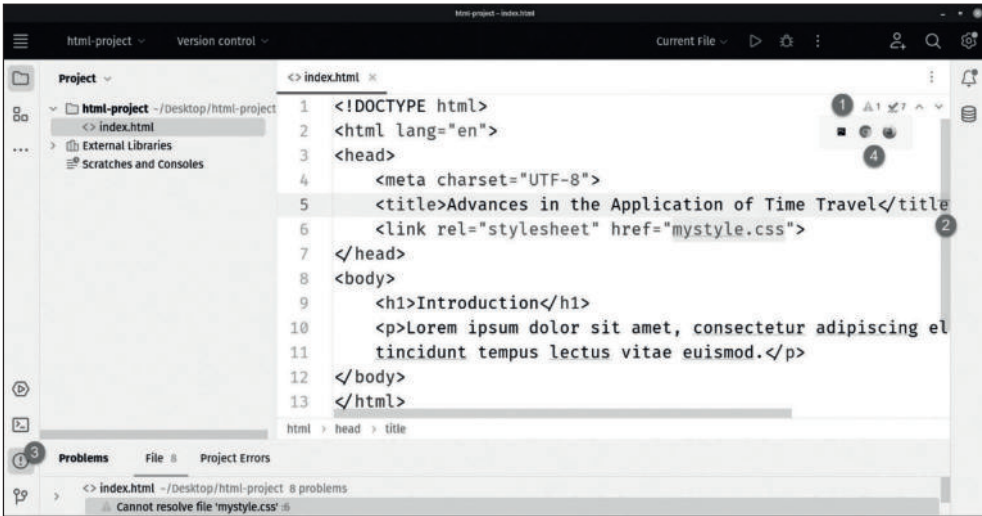


Рис. 7.4. Мы видим много общего между тем, как редактор работает в Python, и как с HTML

Предварительный просмотр веб-страниц

Нажав на любой из значков браузера, вы сможете просмотреть свой код в этом браузере. В моем случае у меня три браузера: Chrome, Firefox и встроенное окно предварительного просмотра PyCharm. Эти значки будут появляться и исчезать по мере приближения курсора мыши к правому верхнему углу окна редактора.

В качестве похвалы Netscape в начале этой главы давайте взглянем на Firefox на рис. 7.5, поскольку он является преемником Netscape:

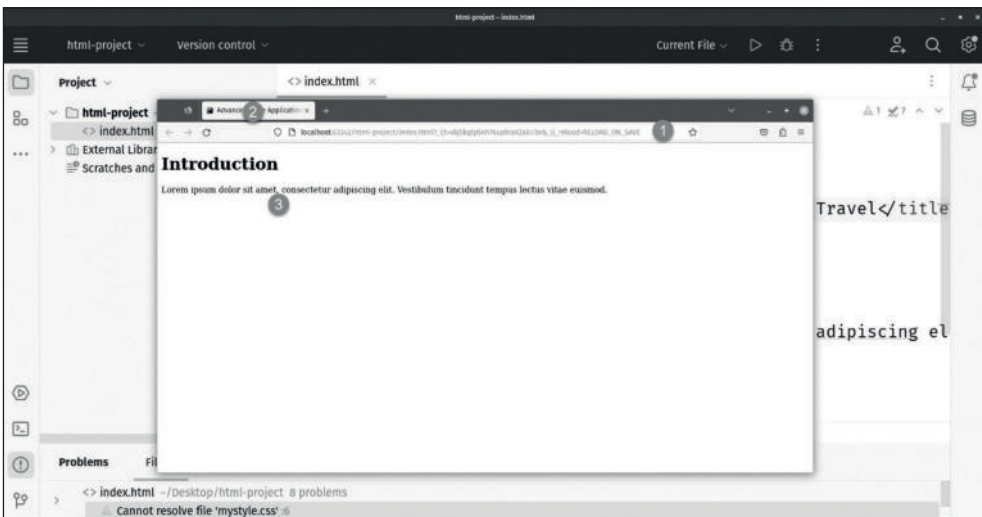


Рис. 7.5. Браузер запускается и отображает нашу страницу

На рис. 7.5 стоит отметить три вещи.

- PyCharm не просто открывает страницу в браузере с URL-адресом `file:///`. Если вы хотите это сделать, вы можете удерживать *Alt* или *Option* при нажатии на значок браузера. В нашем случае PyCharm запустил свой внутренний веб-сервер. Это удобно, поскольку предварительный просмотр вашей работы в браузере с URL-адресом `file:///` очень ограничен. Многие функции просто не будут работать.
- Содержимое тега `<title>` отображается как заголовок на вкладке браузера.
- Вызывается 1991 год, и они хотят вернуть свой сайт. HTML просто создает структуру. Эта страница довольно уродлива без CSS. Мы должны это исправить.

Однако, прежде чем мы это сделаем, я хочу отметить изящную функцию, которую предоставляет нам PyCharm: автоматическую перезагрузку.

Перезагрузка представления браузера при сохранении

Если вы изучите URL-адрес в браузере на рис. 7.5, обратите внимание на атрибут `RELOAD_ON_SAVE`. Вы, скорее всего, догадаетесь, что это дает, но давайте все равно рассмотрим это. На рис. 7.6 я расположил окна плиткой так, что PyCharm находился слева, а мой браузер – справа. Это обычная конфигурация, используемая веб-разработчиками, хотя обычно она воспроизводится на нескольких мониторах:

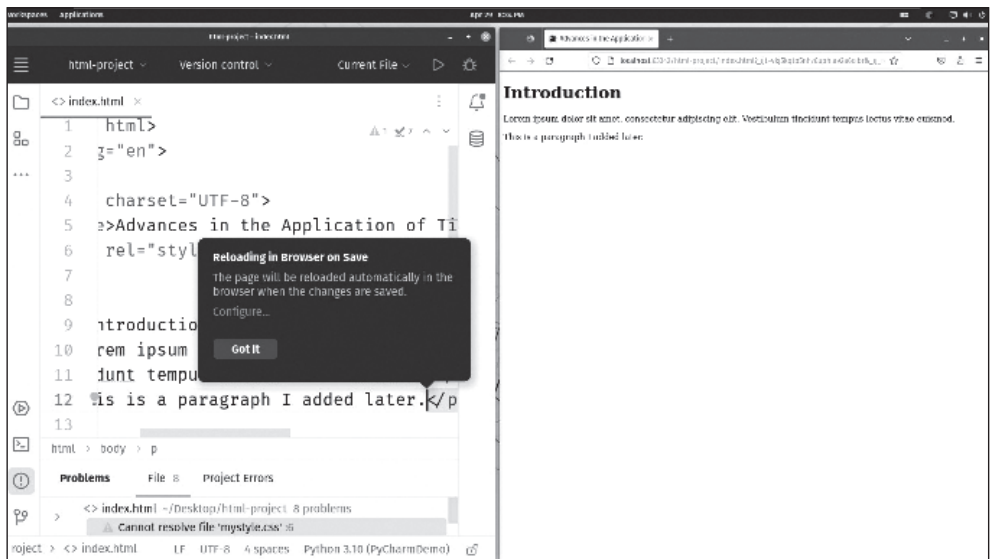


Рис. 7.6. PyCharm автоматически перезагружает страницу в браузере каждый раз, когда вы сохраняете изменения в файле

Я добавил второй тег `<p>`, который читается следующим образом:

```
<p>This is a paragraph I added later.</p>
```


В тот момент, когда я нажал *Ctrl + S* (*Cmd + S* на Mac), браузер обновился новым содержимым. Это серьезно ускоряет разработку! Вряд ли это революционно; большинство редакторов так или иначе поддерживают это, но и PyCharm тоже.

Плагин Live Edit

В более старых версиях IDE JetBrains, поддерживающих разработку HTML, вам требовался плагин Live Edit, чтобы получить эту функцию автоматической перезагрузки. Вам это больше не нужно, поскольку эта функция интегрирована в IDE.

Использование предварительного просмотра HTML в PyCharm

Если вы предпочитаете хранить 100 % своей работы в PyCharm, а не использовать внешний браузер, есть новая функция предварительного просмотра HTML, как показано на рис. 7.7, которая запускает внутреннюю версию браузера *Chromium*:

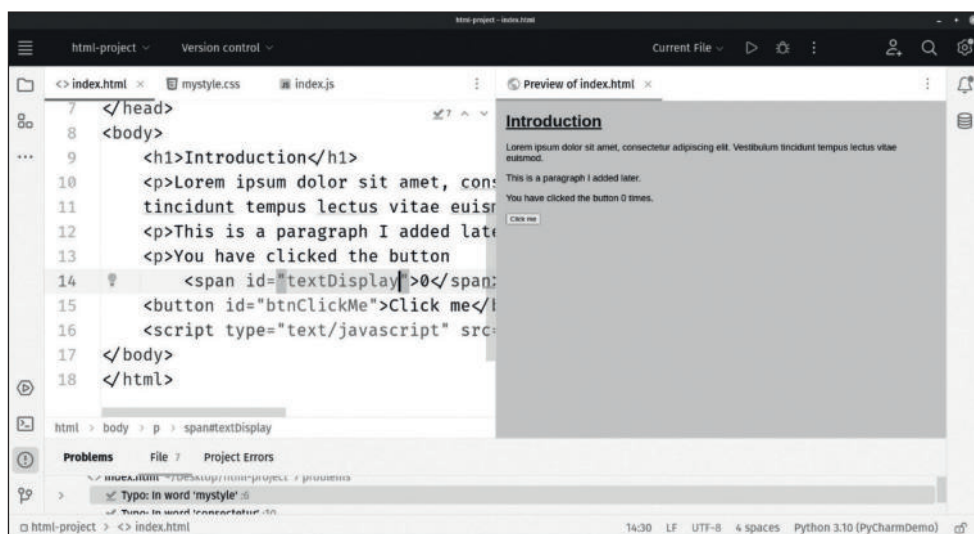


Рис. 7.7. Окно предварительного просмотра PyCharm позволяет вам просматривать свою работу в окне браузера Chromium, реализованном в виде вкладки пользовательского интерфейса PyCharm

Если вы не знакомы, Chromium – это версия Chrome с открытым исходным кодом. Это важно отметить, поскольку за прошедшие годы я видел множество продуктов для веб-разработки с включенными в них глупыми продуктами предварительного просмотра, которые не имеют большого отношения к тому, как ваша работа будет отображаться в реальном браузере.

Настройка доступных браузеров

Значки браузера, которые отображаются в виде браузеров предварительного просмотра, можно настроить. Вы можете добавить или удалить любой браузер на своем компьютере в конфигурацию или из нее. Вы найдете это, перейдя в **Settings** в разделе **Tools | Web Browsers**, а затем **Preview**, как показано на рис. 7.8:

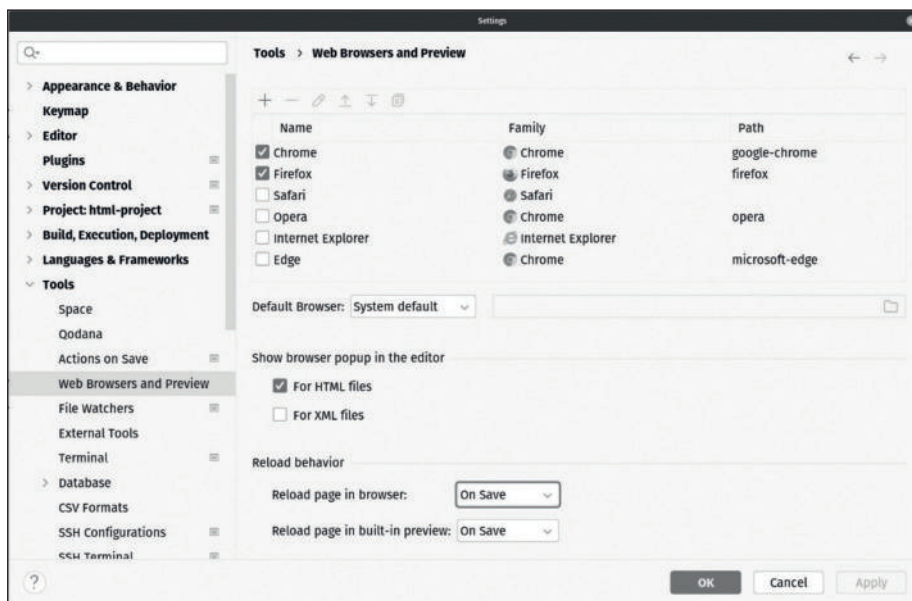


Рис. 7.8. Вы можете добавлять и удалять браузеры для предварительного просмотра и настраивать параметры перезагрузки

Я думаю, вам будет так же просто добавлять новые браузеры, как кликнуть значок + и найти исполняемый файл браузера. Если вы хотите удалить мешающее присутствие Internet Explorer, просто кликните его и значок –.

Обратите внимание на раскрывающийся список, который позволяет вам установить системный браузер запуска по умолчанию, который является первым в списке, или настраиваемый путь. Вы также можете настроить, будут ли отображаться значки запуска для файлов HTML и или XML.

Параметры перезагрузки можно настроить в нижней части экрана. Вы можете настроить перезагрузку при сохранении, при изменении или отключить ее. На мой взгляд, слишком часто срабатывает настройка **On Change**. Я оставляю значение **On Save**.

Навигация по коду структуры с помощью окна структуры

Окно структуры позволяет вам просмотреть структуру вашего кода. Большую часть времени в окне структуры отображается просто список функций и глобальных переменных. Если вы откроете файл класса, то увидите свойства и методы класса. На рис. 7.09 показано, что происходит, когда вы используете окно структуры с документом HTML:

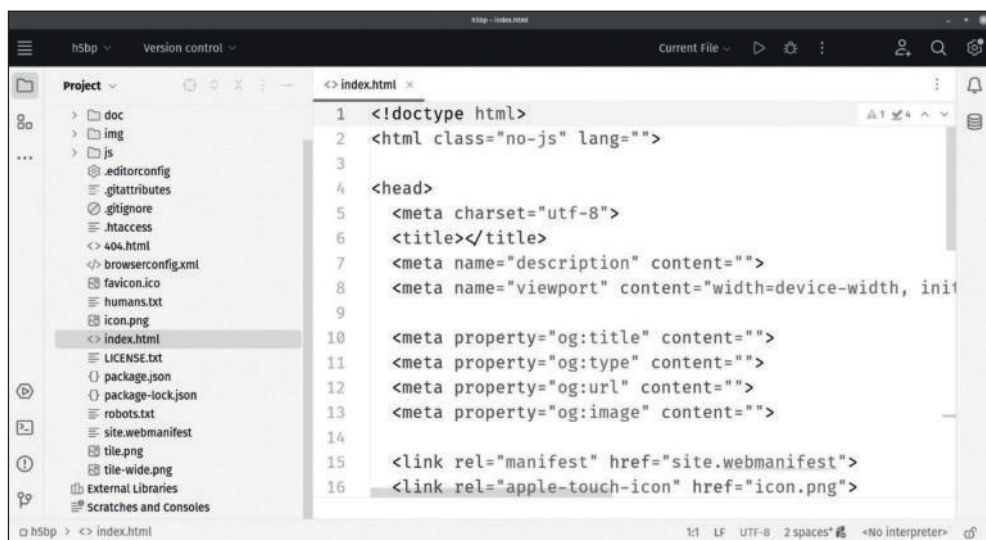


Рис. 7.9. Окно структуры позволяет вам видеть и перемещаться по DOM в окне кода

Когда вы открываете представление структуры на странице HTML, вы можете видеть и легко перемещаться по всей структуре DOM HTML-документа.

ДОБАВЛЕНИЕ CSS

Кликните правой кнопкой мыши папку проекта и добавьте файл таблицы стилей с именем `mystyle.css`, как показано на рис. 7.10. Вам будет предложено выбрать тип таблицы стилей. Мы придерживаемся файлов CSS. Остальные параметры являются причудливыми, но все они в конечном итоге переносятся, обычно через библиотеку **Babel** и скрипт сборки **Webpack**, в обычный CSS. Если вы новичок в работе с интерфейсом, я рекомендую изучить простой CSS, прежде чем переходить к экзотическим темам, таким как **Less** или **Sass**.

Обязательно назовите файл `mystyle.css` так, чтобы это соответствовало тому, что есть в теге `<link>` в нашем HTML-файле. Он также должен находиться в той же папке, что и `index.html`. Добавьте этот код:

```
body {
    background-color: lightblue;
    margin: 20px;
    font-family: Arial, Helvetica, sans-serif;
    font-size: 18px;
}
h1 {
    font-size: 32px;
    color: navy;
    text-decoration: underline;
}
```

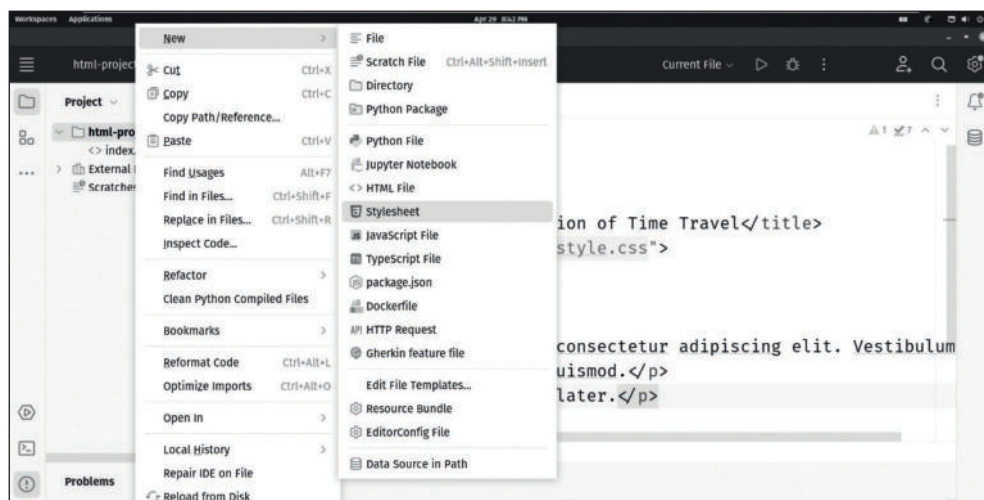


Рис. 7.10. Создайте новый файл CSS, используя File | New | Stylesheet

Сохраните файл и вернитесь в браузер. Не ожидайте, что он обновится; функция перезагрузки PyCharm отслеживает файл `index.html`, а не вновь созданный файл `mystyle.css`. Нажмите кнопку перезагрузки в браузере. Я не буду заморачиваться с черно-белым скриншотом цветной HTML-страницы. Изменение должно быть сразу заметно. Теперь, когда браузер загрузил CSS, можете редактировать CSS в PyCharm. При каждом сохранении страница будет обновляться так же, как это было при редактировании HTML на странице `index.html`.

Вы, несомненно, заметили превосходное завершение кода при вводе кода CSS. Все цветовые коды отображались по имени, а также по шестнадцатеричному значению, как показано на рис. 7.11:

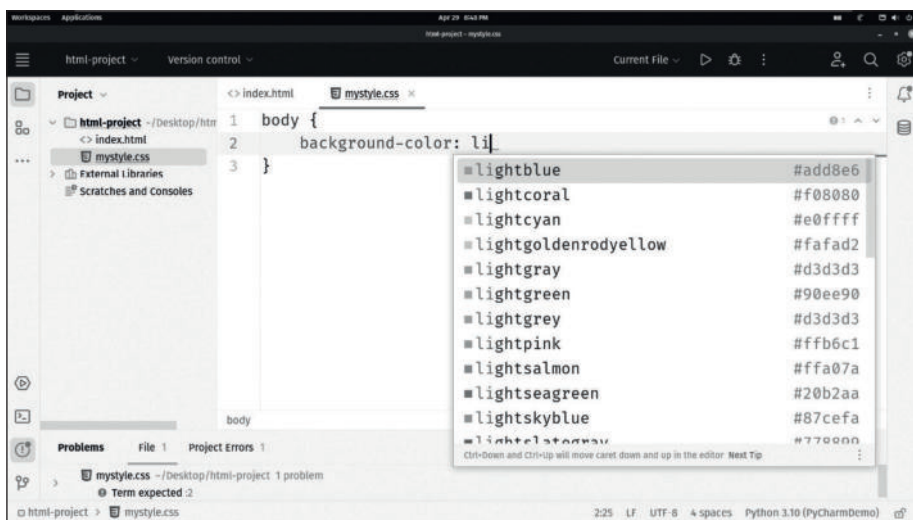


Рис. 7.11. При вводе названий цветов CSS вы увидите имя, шестнадцатеричное значение и предварительный просмотр цвета

Рядом с именем цвета CSS также есть предварительный просмотр цвета, чтобы можно было видеть, какой цвет вы устанавливаете.

Использование селекторов цвета

Вы можете задать цвет, используя имя или шестнадцатеричное значение, либо отредактировать существующий цвет с помощью образца, как показано на рис. 7.12. Здесь я кликнул образец цвета в желобе в строке 10 файла `mystyle.css`-файл:



Рис. 7.12. Вы можете изменить цвет, кликнув образец в поле и выбрав новый цвет

Здесь я могу выбрать цвет, используя палитру RGBA, шестнадцатеричное значение, или просто перетащить курсор по окну цвета. Есть ползунок оттенка, а также ползунок альфа-канала. Это контролирует непрозрачность, которую пользовательский интерфейс отображает в процентах, а не в традиционном целочисленном значении от 0 до 255.

Вы также заметите, что есть инструмент *eye dropper* (пипетка). Можно использовать пипетку, чтобы выбрать любой цвет, видимый на экране. Это позволяет вам, например, сопоставить цвет, который будет использоваться в вашем шрифте, с цветом изображения, видимого на вашем экране.

ДОБАВЛЕНИЕ JAVASCRIPT

Пришло время дополнить наш опыт работы с HTML, CSS и JavaScript, добавив на нашу страницу немного интерактивности. Кликните правой кнопкой мыши папку проекта и добавьте файл JavaScript с именем `index.js`.

Мы собираемся добавить кнопку в наш HTML-файл, чтобы эта кнопка реагировала, когда мы нажимаем на нее с помощью JavaScript. Сначала мы напишем код JavaScript, затем вернемся и добавим кнопку.

Добавляем немного кода JavaScript

Введите этот код в свой файл `index.js`:

```
const btn = document.getElementById("btnClickMe");
const textDisplay = document.getElementById("textDisplay");
let clickCount = 0;
```

Через минуту мы вставим в HTML-файл некоторые элементы, на которые есть ссылки в этом JavaScript. Сначала добавим кнопку с идентификатором `btnClickMe`. Затем применим тег `span` внутри абзаца с идентификатором `textDisplay`. Наконец, мы создадим переменную с именем `clickCount`. Вы, вероятно, понимаете, к чему это ведет. Когда вы загружаете страницу, значение `clickCount` будет равно 0. Каждый раз, когда вы нажимаете кнопку, мы увеличиваем переменную `clickCount`, а затем обновляем HTML-код внутри тега `span`, чтобы отразить новое значение.

Чтобы это работало, нам нужен хендлер `onclick` для кнопки. Добавьте этот код:

```
btn.onclick = function() {
  clickCount++;
  textDisplay.textContent = String(clickCount);
}
```

Кнопка `onclick` назначена анонимной функции, которая увеличивает значение `clickCount` и обновляет дисплей. Я даже приложил дополнительные усилия и приписал к строке `clickCount`.

Имея готовый JavaScript, давайте обновим HTML.

Добавление элементов в HTML-файл

Теперь вернемся к файлу `index.html` и добавим два новых элемента. Сначала мы добавим текстовую область:

```
<p>You have clicked the button
<span id="textDisplay">0</span>
times. </p>
```

Далее добавим кнопку:

```
<button id="btnClickMe">Click me</button>
```

Кнопка имеет атрибут ID `btnClickMe`, который я использовал в JavaScript для события `onclick` в браузере. При каждом нажатии кнопки **ClickMe** это событие срабатывает, и мы запускаем анонимную функцию, которая обновляет значение `clickCount`. Затем функция меняет текст, который отображается в теге `span`, на идентификатор `textArea`.

Мы почти закончили! Нам просто нужно добавить тег скрипта в нижнюю часть HTML-файла. Добавьте эту строку чуть выше закрывающего тега `body`:

```
<script type="text/javascript" src="index.js"></script>
```

Эта строка загрузит ваш файл JavaScript после того, как весь HTML загрузится в окно браузера. Обновите браузер вручную. Вы можете нажать кнопку **ClickMe** и увидеть приращение счетчика. Мой вариант – на рис. 7.13:

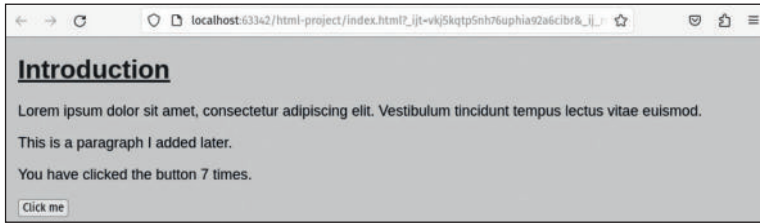


Рис. 7.13. Наша веб-страница теперь интерактивна!

Во время работы вы, несомненно, воспользовались преимуществами обычного автодополнения кода, о котором мы говорили. Это очень приятно, правда? Но будет еще лучше! Я собираюсь показать вам, как выполнять отладку в вашей IDE. Обычно вам придется полагаться исключительно на инструменты отладки в вашем браузере, что означает переключение между отображением кода в браузере и вашей IDE. По крайней мере один или два раза я забыл, в каком инструменте работал, и обнаружил, что пытаюсь редактировать напрямую в инструменте отладки браузера. Это работает, за исключением того, что ваш файл не сохраняется обратно в фактический файл кода. По общему признанию, это происходит только тогда, когда я действительно устал, но с двумя дочерьми среднего школьного возраста, постоянной работой в команде разработчиков и написанием книг я почти всегда «очень устал». Ранее я упоминал, что отладчик – моя любимая функция, поэтому без лишних слов я предлагаю вам отладку JavaScript на стороне клиента (client-side)!

Отладка JavaScript на стороне клиента

Вы можете отладить JavaScript, который работает в вашем браузере, настроив Google Chrome в качестве браузера по умолчанию. В Chrome встроен очень хороший сервер удаленной отладки. PyCharm может подключиться к нему и отобразить отладочную информацию непосредственно в пользовательском интерфейсе PyCharm, так же как она выглядела бы при отладке кода Python в главе 6.

Откройте файл `index.js` и установите точку останова в нашей функции хендлера кликов в строке 7, как показано на рис. 7.14:

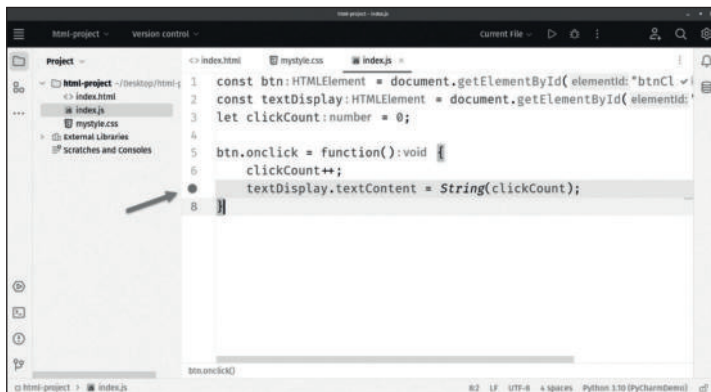


Рис. 7.14. Установив точку останова в клиентском коде, вы можете отлаживать JavaScript со стороны клиента так же легко, как и код Python

Чтобы использовать отладчик JavaScript непосредственно в PyCharm, необходимо создать конфигурацию запуска. Мы подробно рассмотрели создание конфигураций запуска в главе 6. Чтобы создать конфигурацию запуска, способную работать с файлом JavaScript, загруженным в HTML, создайте конфигурацию запуска, используя шаблон отладки JavaScript, как показано на рис. 7.15:

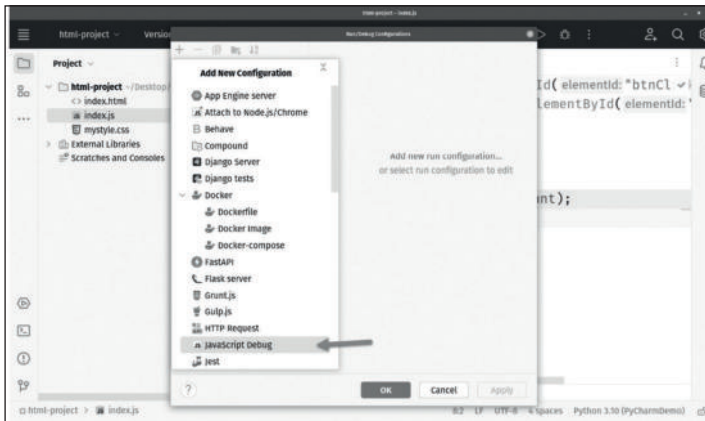


Рис. 7.15. Используйте шаблон отладки JavaScript для настройки конфигурации запуска, которая позволит вам отлаживать код JavaScript непосредственно в PyCharm

Выбрав шаблон, вы увидите экран настроек, аналогичный показанному на рис. 7.16:

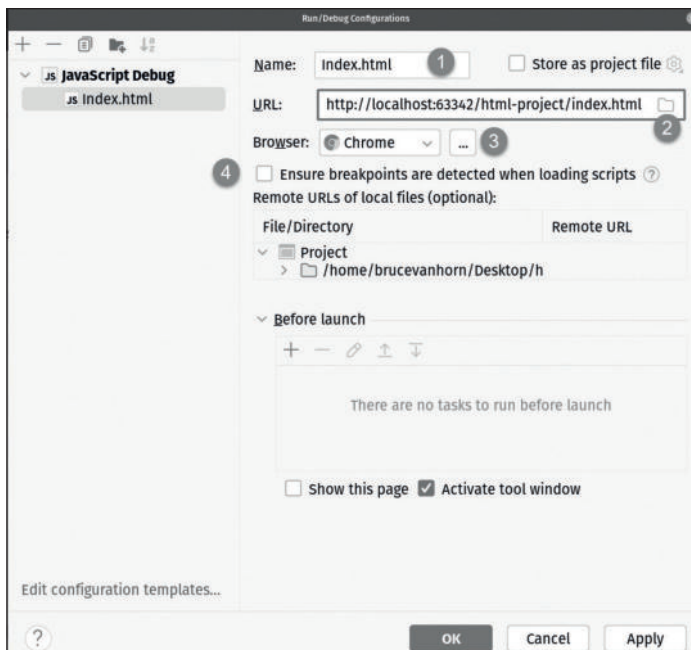


Рис. 7.16. Заполните настройки для запуска HTML-страницы, которая загружает ваш JavaScript для отладки

Поля основных настроек пронумерованы на рис. 7.16.

В позиции 1 вы можете ввести имя конфигурации запуска. Это может быть что угодно, поскольку имя не влияет на работу конфигурации запуска. Для получения URL-адреса вам нужно кликнуть значок папки под номером 2 и просто перейти к файлу `index.html`. Когда вы это сделаете, PyCharm заполнит URL-адрес локального хоста, а также номер порта, используемый встроенным веб-сервером PyCharm. Выбор 3 определяет, какой веб-браузер будет запущен. PyCharm поддерживает отладку только в браузерах на базе Chromium. Например, Chrome, Chromium, Edge или Brave будут работать нормально, а Safari или Firefox – нет.

Параметр 4 важен, поскольку он загружает точки останова JavaScript до того, как сработает событие `onPageLoad`. Если вы используете JavaScript, который загружается на основе **выражения немедленно вызванной функции (IIFE)** или вызове `$(function() {jQuery, вам следует установить этот флажок. Наш скрипт очень прост, поэтому я могу оставить этот флажок без отметки. Обратите внимание, что установка этого флажка замедлит время загрузки вашего отладчика.`

Чтобы начать сеанс отладки, нажмите обычную кнопку отладки. Если вы не помните, где это находится, вернитесь к главе 6. Chrome запустится вместе с вашей веб-страницей. После нажатия кнопки **Click Me**, которую мы сделали ранее, отладчик в PyCharm должен перехватить это событие. Как и в случае с отладчиком, рассмотренным в главе 6, вы можете использовать все те же функции для проверки и пошагового выполнения вашего кода JavaScript.

Работа с шаблонами Emmet

Emmet – мощный и широко используемый плагин в PyCharm, который обеспечивает эффективный и упрощенный способ написания кодов HTML, **JavaScript XML (JSX)** и CSS. Он предлагает ряд функций шаблонов, которые значительно повышают производительность разработчиков и ускоряют процесс кодирования. Используя сокращения и фрагменты кода Emmet, разработчики могут писать фрагменты кода с сокращенным синтаксисом и расширять их до полных структур HTML или CSS всего несколькими нажатиями клавиш. Единственным недостатком является то, что вам нужно выучить и запомнить соответствующие аббревиатуры. У меня нет места в этой книге, чтобы превратить ее в учебник по Emmet. Шпаргалка доступна по адресу <https://docs.emmet.io/cheat-sheet/>.

Чтобы использовать Emmet в PyCharm, вам нужно всего лишь ввести сокращение Emmet в окно редактора и нажать **Tab**. Например, если набрать `ul>li.item$*5` и нажать клавишу **Tab**, Emmet может создать неупорядоченный список с пятью элементами списка, где символ `$` автоматически увеличивается для каждого элемента. Эта функция особенно полезна при работе с повторяющимися структурами HTML и устраняет необходимость вручную вводить повторяющиеся блоки кода.

Еще одна мощная функция Emmet в PyCharm заключается в том, что вы можете эффективно перемещаться и редактировать код HTML и CSS. С помощью функции **Go to Edit Point** разработчики могут перемещаться между определенными точками редактирования в расширенном сокращении, что позволяет быстро вносить изменения и корректировки. Кроме того, функция

автоматического закрытия тегов Emmet гарантирует автоматическое закрытие HTML-тегов, что снижает вероятность синтаксических ошибок и экономит время в процессе кодирования. Функция **Go to Edit Point** эффективно позволяет размещать заполнители в шаблоне кода. Если вы опробовали предыдущий пример с неупорядоченным списком, вы могли заметить, что код был сгенерирован, но ваш курсор находился внутри тега первого элемента списка, как показано на рис. 7.17:

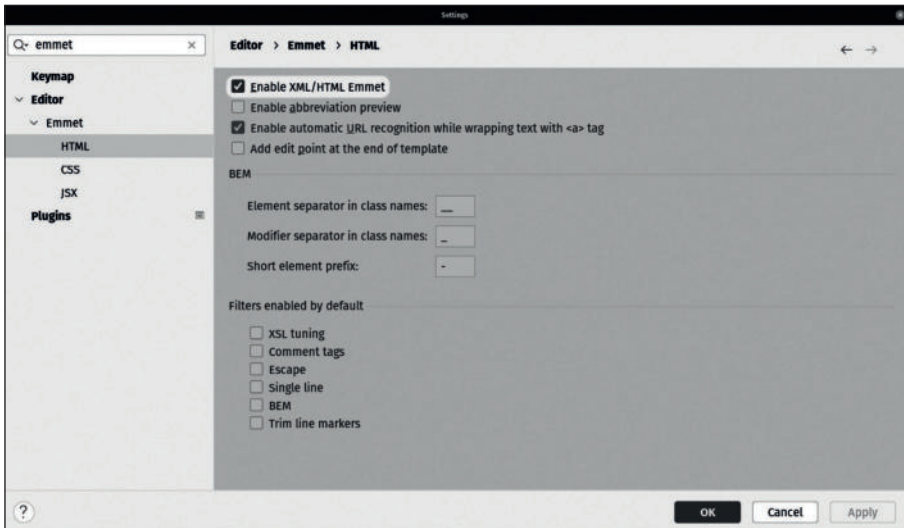


Рис. 7.17. Emmet создал неупорядоченный список благодаря точке редактирования, определенной в шаблоне

Emmet также предоставляет интеллектуальные сокращения CSS, которые упрощают процесс написания и расширения свойств CSS. Используя такие сочетания клавиш, как «bg» для «фона» (background) или «p» для «отступа» (padding), разработчики могут быстро создавать фрагменты кода CSS без необходимости запоминать полные имена свойств. Эта функция ускоряет процесс разработки CSS и повышает читаемость кода.

Типы HTML-проектов в PyCharm Professional

Ранее я упоминал, что хочу начать с простого пустого проекта с простым кодом HTML, CSS и JavaScript, который мы создали с нуля. PyCharm Professional предлагает новые дополнительные типы проектов на чистом HTML. Под чистым я подразумеваю, что они не используют современные JavaScript-фреймворки, такие как React или Angular. Эти современные фреймворки представляют собой сдвиг парадигмы в разработке внешнего интерфейса. Чистый HTML-проект будет продолжать использовать очень традиционные манипуляции с DOM, как мы это делали ранее в нашем примере с JavaScript. Чтобы изменить содержимое диапазона счетчика кликов, мы использовали этот код:

```
Document.getElementById
```

Поскольку клиентский JavaScript, подобный этому, запускается в браузере, код имеет доступ к объекту документа, который фактически является окном браузера. JavaScript может манипулировать загруженным в данный момент документом или даже обновлять содержимое самого окна браузера.

Современные фреймворки JavaScript больше не манипулируют DOM. Вместо этого они полагаются на механизм изменения состояния в сочетании с идеей теневого DOM. В случае React ваша программа, состоящая в основном из разметки JavaScript, CSS и HTML, созданной JSX, поддерживает конечный автомат¹, управляемый компонентами. Изменения состояния вызывают события в среде React, которые, в свою очередь, вызывают повторную отрисовку затронутых областей пользовательского интерфейса.

Эту парадигму довольно сложно объяснить непосвященным, и, поскольку эта книга не посвящена современному JavaScript, я отнесу идею более глубокого объяснения к другим книгам в разделе «Дальнейшее чтение» в конце этой главы.

Мы можем рассмотреть два типа проектов, основанных на DOM. Давайте проверим их.

Шаблон HTML 5

Проект **HTML 5 Boilerplate (H5BP)** можно найти по адресу <https://html5boilerplate.com/>. Этот проект существует с 2011 года и представляет собой инструмент для создания сайтов, воплощающий в себе лучшие практики разработки HTML. Если бы вы могли создать на 100 % индивидуальный веб-сайт в качестве отправной точки, а затем изменить его, чтобы сделать что угодно, у вас был бы H5BP.

Вот некоторые из его ключевых особенностей:

- хорошо продуманный и структурированный файл `index.html`, наполненный тегами метаданных и всеми необходимыми оптимизациями. Просто добавьте контент;
- `Normalize.css` и `Main.css` позаботятся о сбросе CSS и предоставляют некоторые базовые стили для помощников, медиа-запросы для адаптивной разработки и даже параметры для печати;
- встроенный сервис Google Analytics;
- библиотека Modernizr включена, чтобы вы могли определить, в каком браузере выполняется ваш код, и отреагировать соответствующим образом;
- файлы настроек сервера, позволяющие обеспечить производительность и безопасность;
- значки-заполнители для мобильных устройств (Favicon) и прогрессивные веб-приложения.

¹ Конечный автомат (state machine) – это некоторая математическая абстрактная модель дискретного устройства, имеющего один вход, один выход и в каждый момент времени находящегося в одном состоянии из конечного множества возможных. Используется для представления и управления потоком выполнения каких-либо команд. Конечный автомат идеально подходит для реализации искусственного интеллекта в играх, получая правильное решение без написания громоздкого и сложного кода. – Прим. ред.

Как я уже сказал, это идеальный стартовый набор HTML 5 для любого веб-сайта, который не ограничивает вас во многом. Старые версии используют jQuery, но, как и многие другие платформы, более поздние сборки удаляют его как определенную зависимость, если вы не хотите его использовать.

Давайте создадим проект H5BP в PyCharm. Нажмите **File | New Project** и выберите тип проекта **HTML 5 Boilerplate**. Вас спросят, какую версию вы хотите использовать. Просто выберите последнюю. Для меня это версия 8.0.0. Затем просто укажите место, где вы хотите создать проект, и как его назвать. Я называю свой h5bp и просто помещаю его в папку PyCharm Projects на своем компьютере по умолчанию.

За своими кулисами PyCharm сгенерирует большой проект, используя H5BP через npm. На рис. 7.18 показан мой сгенерированный проект:

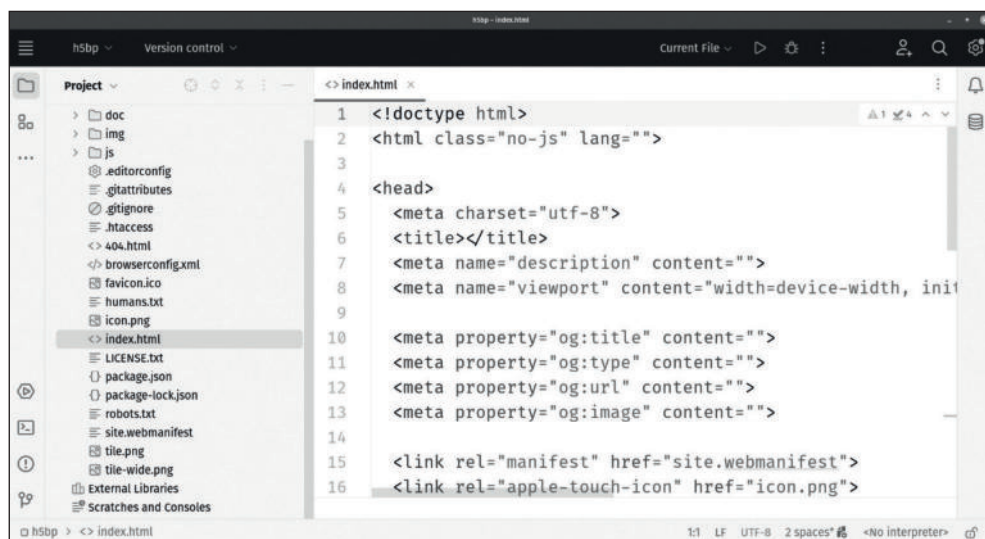


Рис. 7.18. Мой шаблонный проект HTML 5 с открытым index.html

Ух ты! Просто посмотрите на весь этот код, который вам не нужно было вводить, искать или запоминать! Теперь, когда загружен более обширный проект, давайте рассмотрим несколько полезных функций, о которых я еще не упомянул.

Предварительный просмотр и редактирование графики с помощью внешних инструментов

Вы можете просмотреть графику в PyCharm, просто дважды кликнув файл. Я открою файл `tile.png`, как показано на рис. 7.19:

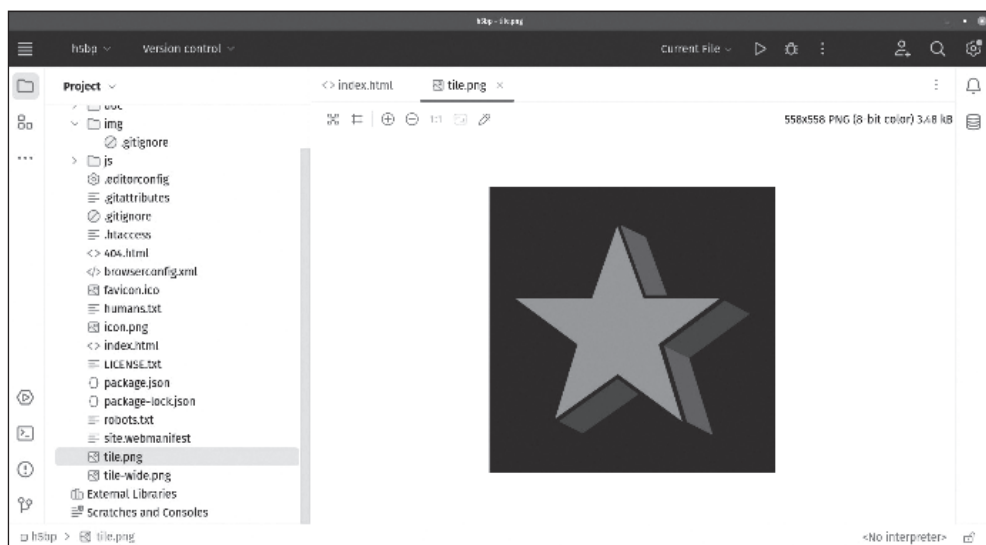


Рис. 7.19. Вы можете просмотреть графические элементы в PyCharm, просто открыв их

Возможно, я немного подразнил вас, намекнув, что вы можете редактировать изображения в PyCharm. Вы не можете. Но можно настроить внешний редактор, который будет запускаться при открытии графики.

Запустите окно настроек и найдите **Tools | External Tools**. Нажмите значок + на верхней панели инструментов. На рис. 7.20 я настраиваю **Gnu Image Manipulation Program (GIMP)**, которая является альтернативой программе **Adobe Photoshop** с открытым исходным кодом:

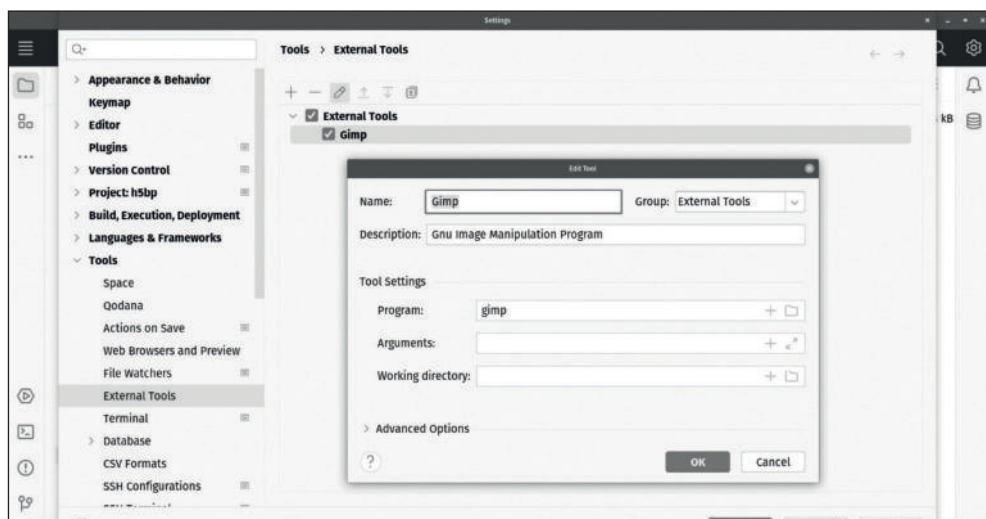


Рис. 7.20. Вы можете настроить GIMP как внешний редактор для открытия изображений в вашем проекте

Поймите меня правильно: я предпочитаю Photoshop. Если он у вас есть, используйте его. Он не работает в Linux (для меня Wine не в счет), поэтому GIMP – следующий лучший вариант. Любой тип файла, который PyCharm не может редактировать, можно использовать во внешнем редакторе. Еще одно хорошее дополнение, которое я часто делаю, – это **Inkscape** или **Adobe Illustrator** для работы с файлами **масштабируемой векторной графики (SVG)**. Они полезны, поскольку не зависят от разрешения. Аналогичным образом, если вы работаете с файлами PDF, добавление **Adobe Acrobat** или какой-либо другой программы для управления PDF-файлами является хорошим вариантом для добавления в качестве внешнего инструмента.

Чтобы открыть файл во внешнем редакторе, просто кликните файл правой кнопкой мыши, выберите пункт меню **External Tools** и выберите соответствующий внешний инструмент.

Загрузка вашего сайта на сервер

Многие крупные сайты и программные проекты используют систему непрерывного развертывания для развертывания кода на серверах. Если ваш проект еще не так уж далеко продвинулся или, может быть, вам просто нужно что-то простое, в PyCharm есть механизм публикации, который поможет вам легко опубликовать веб-сайт или приложение. В этом отношении его можно использовать для публикации любого программного обеспечения, но чаще всего я вижу, что он используется для развертывания веб-проектов на веб-серверах.

Вы найдете конфигурацию и инструменты развертывания в меню **Tools**, как показано на рис. 7.2:

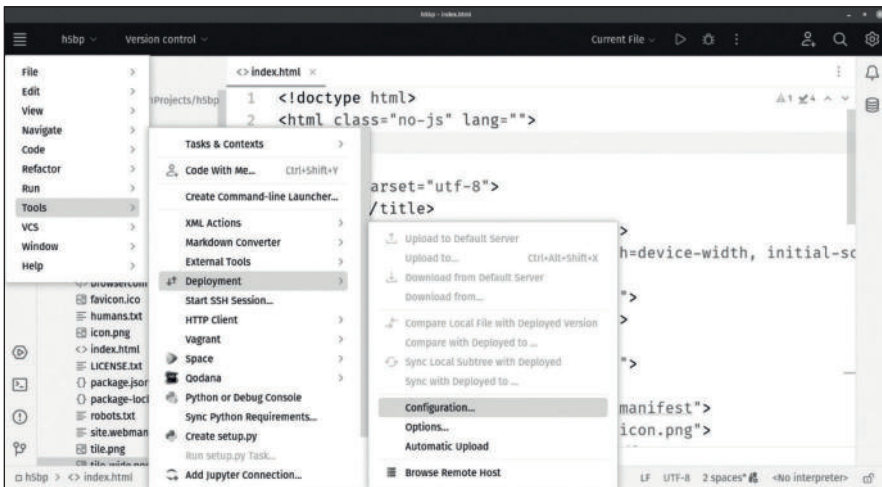


Рис. 7.21. Настройки и инструменты развертывания можно найти, перейдя в Tools | Deployment

Процесс настройки развертывания предполагает настройку одного или нескольких удаленных серверов. Давайте посмотрим, как это делается.

Настройка удаленного сервера

Нажмите пункт меню **Configuration...**, показанный на рис. 7.21. Вас встретит простое серое окно с надписью **Please add a web server to configure**. Нажмите

значок + в верхнем левом углу окна и выберите тип соединения, которое собираетесь использовать. Вы можете увидеть варианты выбора на рис. 7.22:

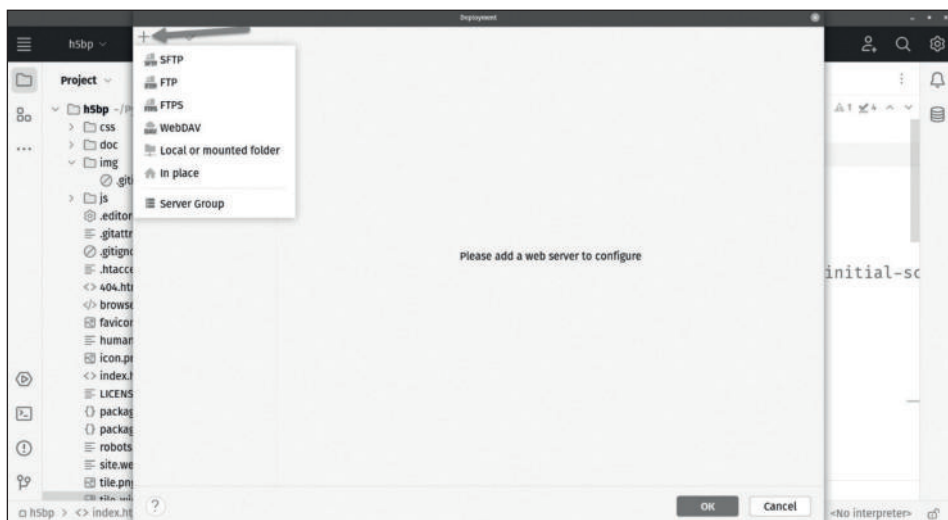


Рис. 7.22. Выберите тип подключения к серверу, который вы хотите использовать

У вас есть несколько протоколов подключения на выбор.

- **Протокол безопасной передачи файлов (SFTP).** Это должно быть вашим любимым занятием. Из вариантов в списке он является наиболее безопасным, поскольку основан на **протоколе безопасной оболочки (Secure Shell, SSH)**.
- **Протокол передачи файлов (FTP).** В реальном мире вам никогда не следует использовать эту опцию. FTP сам по себе отправляет все свои данные аутентификации в незашифрованном текстовом потоке. Используйте его только в том случае, если хотите, чтобы ваш сайт взломали.
- **Защищенный протокол передачи файлов (FTPS)** – это FTP с шифрованием **уровня защищенных сокетов (Secure Sockets Layer, SSL)**. Те же сертификаты шифрования, которые вы используете для защиты своего веб-сайта с помощью HTTPS, можно использовать с FTP, чтобы стать FTPS.
- **Система распределенной разработки и управления версиями в интернете (WebDAV)** была создана еще в те времена, когда такие компании, как Microsoft и Macromedia (которая была куплена Adobe), продавали инструменты веб-разработки для веб-дизайнеров – т. е. людей, которые писали HTML, но делали это слишком уверенно. Большинство из них погибло во время великого восстания разработчиков в 1998 году. WebDAV позволил инструменту разработки беспрепятственно подключаться к серверу с минимальными усилиями. Однако, как и у FTP, у него есть свои проблемы с безопасностью. Я не рекомендую его использовать.
- Опция **Local or mounted folder** используется, когда у вас есть прямой сетевой доступ к папке, используемой для обслуживания вашего сайта или приложения.

- Алгоритм **In Place**¹ позволяет копировать файлы проекта в подпапку текущего проекта. Это напоминает мне многие инструменты сборки, которые создают папку с именем `build`, а затем копируют в нее необходимые файлы. Я считаю, что опция **Local or mounted folder** более полезна.

Я собираюсь рассказать вам о подключении к серверу с помощью SFTP. Когда вы выберете опцию **SFTP** в меню, показанном на рис. 7.22, вам будет предложено назвать сервер. Это понятное имя, которое поможет запомнить, с каким сервером вы работаете, а не DNS-имя или IP-адрес. Я собираюсь вызвать свой Web Server, как показано на рис. 7.23:



Рис. 7.23. Дайте вашему серверу описательное имя

Далее вам потребуются учетные данные. Обычно вы получаете их от дружелюбного местного системного администратора или службы хостинга. Они вводятся в конфигурацию SSH, которая отличается от конфигурации вашего сервера развертывания. Пользовательский интерфейс здесь становится немного странным. Чтобы настроить конфигурацию SSH, нажмите кнопку с многоточием, показанную на рис. 7.24:

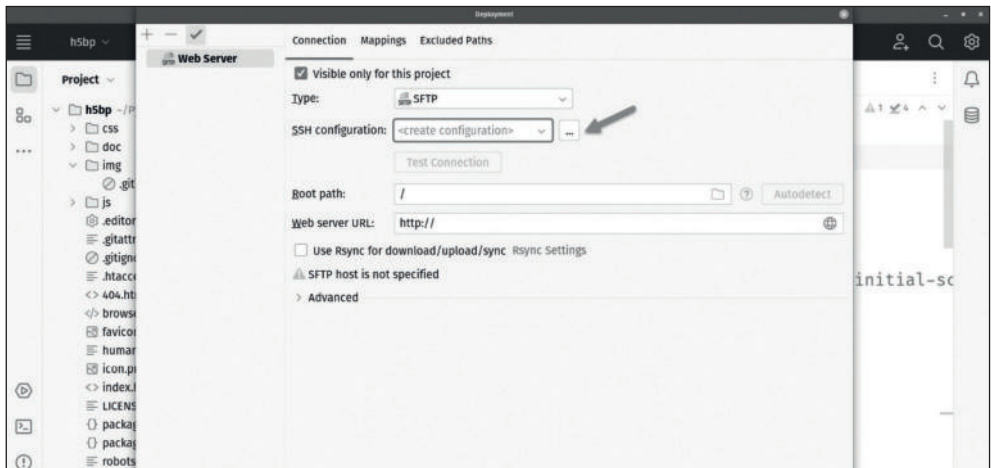


Рис. 7.24. Для создания развертывания вам также необходимо создать конфигурацию SSH, кликнув показанное здесь многоточие

¹ Алгоритм «на месте» – это алгоритм, который не требует дополнительного пространства и создает выходные данные в той же памяти, что и данные, путем преобразования входных данных «на месте». Однако допускается использование небольшого постоянного дополнительного пространства для переменных. – Прим. ред.

Откроется еще одно пустое серое окно. Кликните знак + в верхнем углу этого окна, и вы увидите экран, подобный моему, как показано ниже на рис. 7.25.

Заполните данные вашего подключения, т. е. ваш хост, порт, имя пользователя и пароль. Помимо пароля, вы также можете использовать ключи SSH или подключение к агенту конфигурации и аутентификации OpenSSH. Я буду проще и буду придерживаться пароля. Нажатие кнопки **Test Connection** сообщает мне, работают ли мои учетные данные.

Далее нам нужен корневой путь. Это должен быть корень папки документов вашего веб-сервера, если вы хотите развернуть веб-сайт. Там есть кнопка автоопределения, которая найдет домашнюю папку пользователя SSH. Это нормально, если вы работаете из этой локации. Это обычно – работать из таких локаций, как `/var/www/html` на удаленном сервере. Если это папка, которую использует ваш сервер, убедитесь, что ваш системный администратор предоставил вам учетные данные, необходимые для доступа к этой локации.

Ниже вы увидите запись для URL-адреса веб-сервера. Корневой путь на сервере должен соответствовать URL-адресу веб-сервера. Мои настройки вы можете увидеть на рис. 7.25:

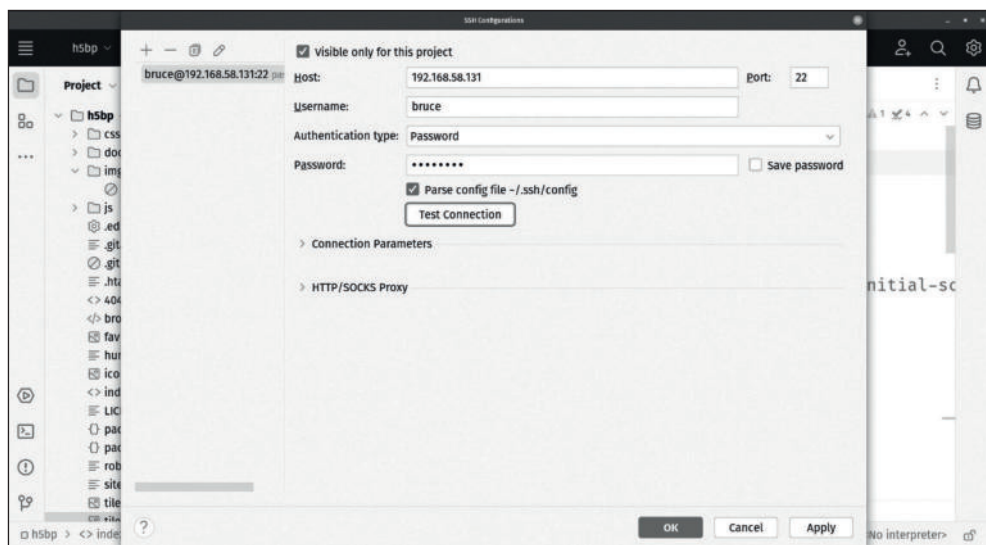


Рис. 7.25. Мои настройки развертывания (на этот момент)

Ваша следующая остановка – вкладка **Mappings**. На рис. 7.26 показана вкладка **Mappings** с указанным маппингом.

Для веб-проекта вам обычно достаточно одного маппинга между папкой, содержащей ваш код на вашем компьютере, и корнем документа веб-сервера. Затем переключитесь на вкладку **Excluded Paths**, как показано на рис. 7.27:

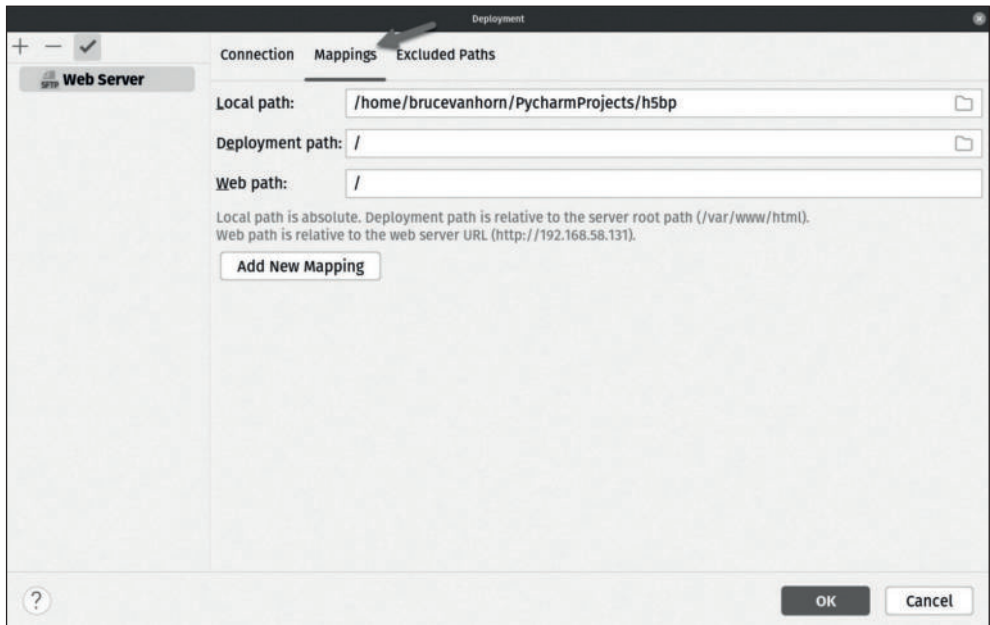


Рис. 7.26. На вкладке Mappings задаются сопоставления (маппинги) между папками на вашем компьютере и папками на сервере

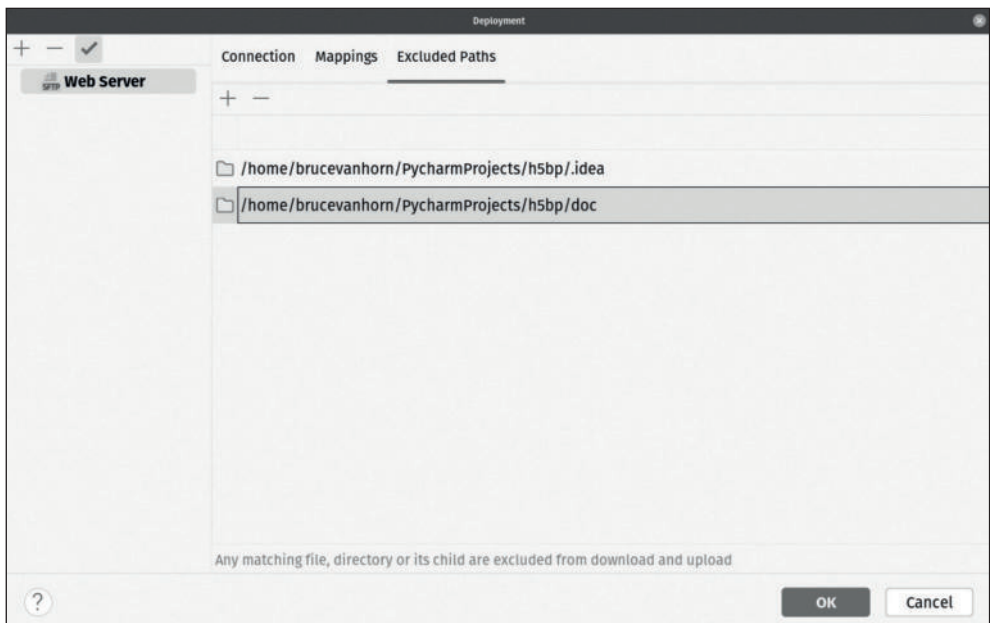


Рис. 7.27. Исключите пути, которые не нужно копировать на сервер

Исключите все файлы, копирования которых вы не хотите. Можете указать локальные файлы, которые не следует копировать на сервер, и удаленные файлы, которые не следует копировать на ваш компьютер. Я добавил содержимое папки `doc`, содержащей документацию для разработчика, и содержимое папки `.idea`, которая используется PyCharm. Нажмите **ОК**, и ваше развертывание будет сконфигурировано.

Загрузка на сервер

Теперь вы готовы загрузить свой проект. Вернитесь в меню **Tools | Deployment** и на этот раз нажмите **Upload to Web Server**, как показано на рис. 7.28:

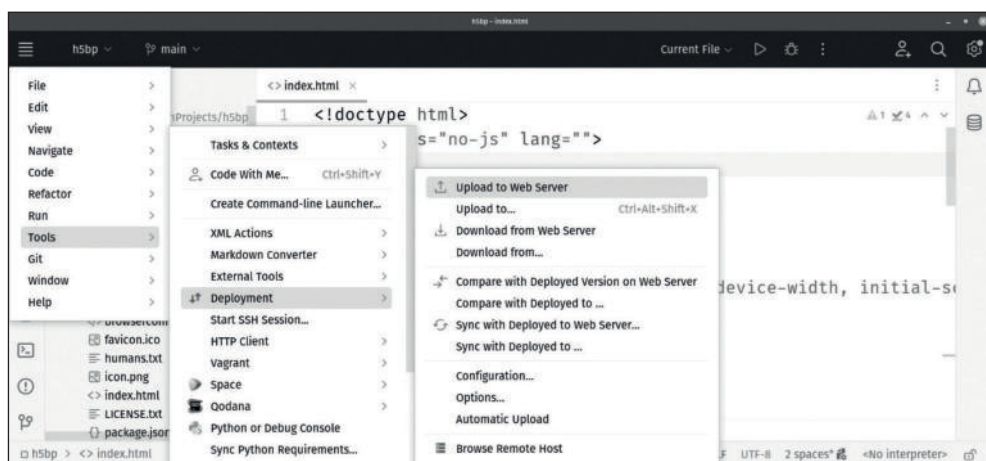


Рис. 7.28. Элементы управления загрузкой находятся в меню развертывания

Если вы назвали свой удаленный сервер как-нибудь по-другому, кроме **Web Server**, именно это имя появится в меню. Вам будет предложено подтвердить операцию загрузки. PyCharm отобразит сообщение о том, что загрузка прошла успешно, как показано на рис. 7.29:

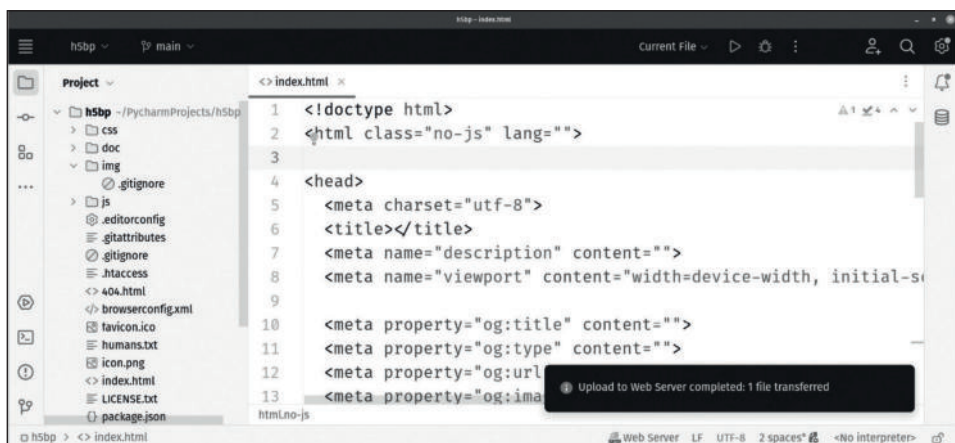


Рис. 7.29. Файл `index.html` был успешно передан

Операция загрузки скопировала только открытый файл `index.html`. Мы можем проверить это, нажав **Tools | Deployment | Browse Remote Host**. Это открывает вид удаленного хоста, позволяя вам видеть файлы на хосте. Можете увидеть мои на рис. 7.30:

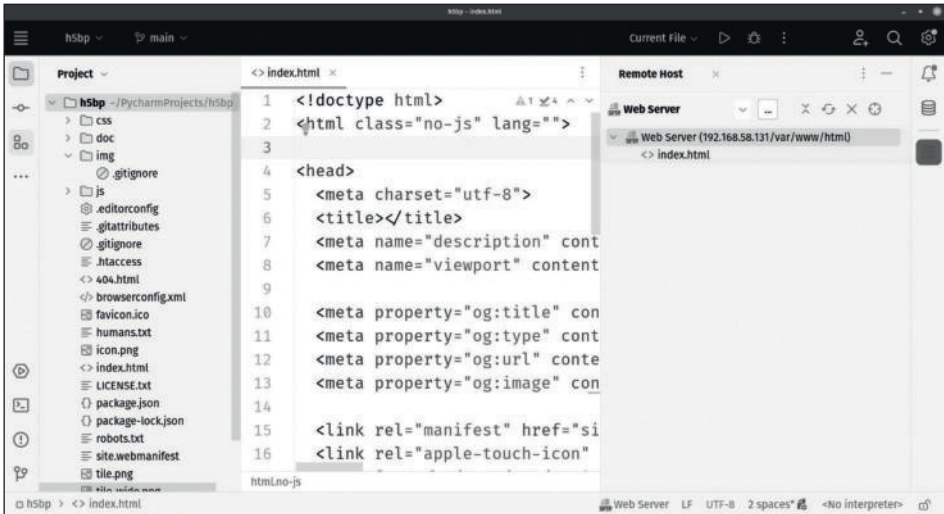


Рис. 7.30. Вы можете просмотреть мапированную корневую папку проекта на сервере

Вероятно, вы захотите перенести весь сайт, а не только открытый файл. Кликните правой кнопкой мыши внутри окна удаленного хоста и выберите **Upload here**, как показано на рис. 7.31:

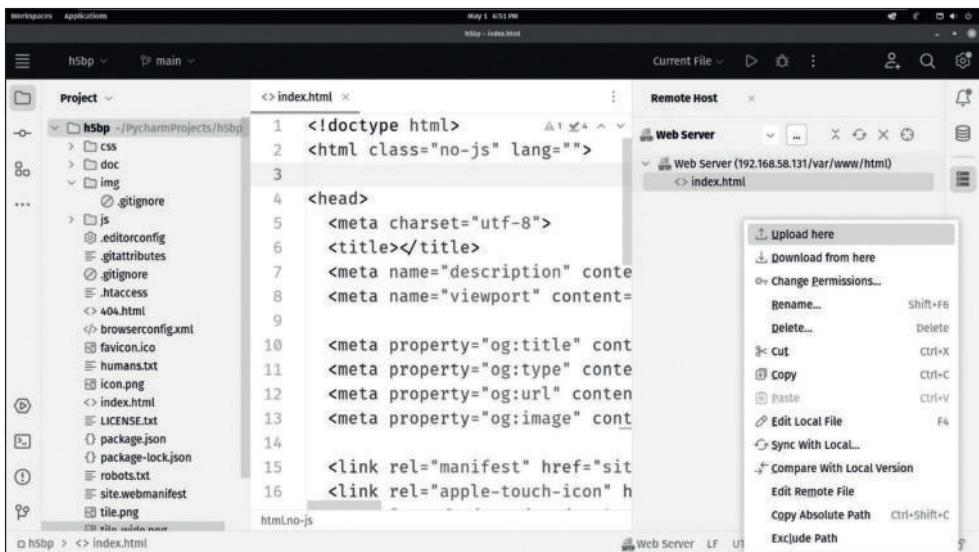


Рис. 7.31. Кликните правой кнопкой мыши и выберите «Загрузить сюда», чтобы загрузить весь сайт

Как видно из меню, у вас есть полный графический контроль над файлами на сервере, включая все операции с файлами, такие как переименование, копирование или удаление. Вы можете сравнивать операции, которые будут отличать локальный файл от удаленного, и, конечно же, файловые операции являются двусторонними, поэтому можете синхронизировать работу, выполняемую непосредственно на сервере, с вашим компьютером.

В целом этот инструмент более эффективен, чем отдельная программа транспортировки файлов, такая как FileZilla. Он интегрирован непосредственно в IDE, и есть даже настройка автоматической синхронизации, которая загружает локальную копию на удаленный компьютер при каждом сохранении файла. В PyCharm есть все необходимое для работы с HTML-интерфейсом веб-сайта или приложения, включая возможность публикации прямо из PyCharm.

Создание проекта Bootstrap

Bootstrap – это еще один тип проекта HTML 5, ориентированный на DOM, который вы можете использовать в PyCharm Professional. Этот вариант более полезен, если вы создаете приложение, а не веб-сайт. Bootstrap – это библиотека стилизованных HTML-компонентов, разработанная Twitter и помогающая пользователям быстро разрабатывать приложения. По сути, вы можете создать красивое приложение, просто сгенерировав этот проект, а затем скопировав и вставив фрагменты Bootstrap для таких элементов, как кнопки, макеты карточек, ползунки, переключатели и сетка, ориентированная на мобильные устройства.

Вы можете создать проект Bootstrap, кликнув **File | New Project**, затем выбрав **Bootstrap** из шаблонов справа, как показано на рис. 7.32:

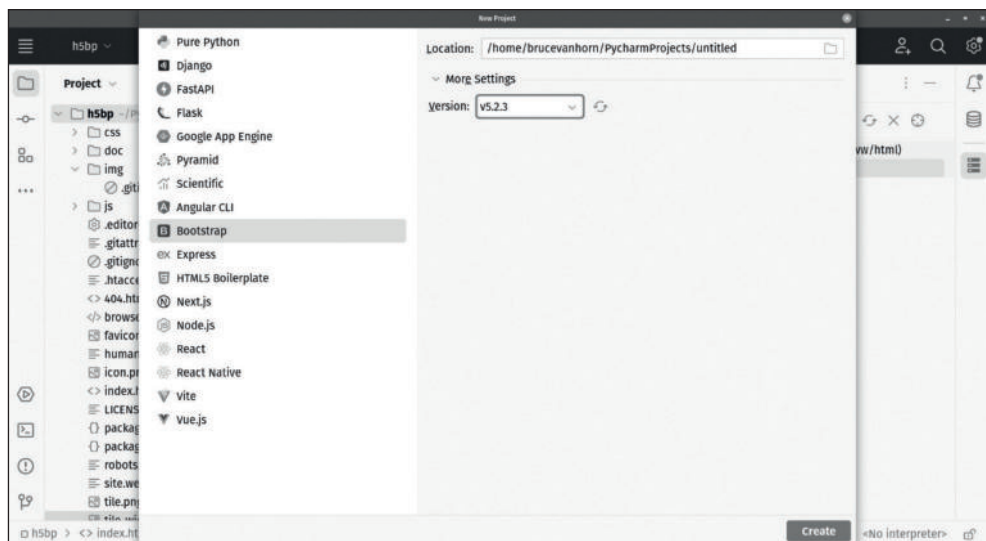


Рис. 7.32. Создание проекта Bootstrap

Все, что делает этот шаблон, – это создает папку `js` и папку `css`. Затем PyCharm загружает Bootstrap и помещает файлы библиотеки в соответствующие папки.

Вот и все. Он даже не генерирует `index.html`- файл для вас. Это просто быстрый способ настроить проект с помощью Bootstrap.

РАБОТА С СОВРЕМЕННЫМИ JAVASCRIPT И NODEJS

В PyCharm есть все функции и шаблоны, доступные в продукте Web Storm, которые предназначены для работы с современными проектами JavaScript. Поскольку я предполагаю, что вас больше всего интересуют проекты Python, я не буду тратить много времени на детали разработки серверного JavaScript, поскольку мы рассмотрим варианты Python, такие как Flask, FastAPI, Pyramid и Django, в следующих нескольких главах.

Однако вы должны знать, что, если у вас есть проекты JavaScript и проекты Python, нет необходимости покупать два отдельных продукта.

Создание проекта NodeJS

Чтобы создать новый проект Node JS, просто нажмите **File | New Project** и выберите шаблон проекта NodeJS. Это эквивалент создания проекта с помощью команды `npm init -y`. Все, что вы получаете, – это общий файл `package.json`. Это довольно простой способ, но он избавляет вас от необходимости запускать терминал и использовать команду `init`.

В дополнение к базовому проекту Node вы также можете создать проект **Next.js** или проект **Express**. Express – это ответ JavaScript на Flask, о котором мы поговорим в следующей главе. Он используется для разработки серверной части вашего проекта. Next.js, с другой стороны, представляет собой сочетание фронтенд- и бэкенд-разработки¹ с простой в использовании службой хостинга. Подробнее об этом см. <https://nextjs.org>.

Создание проекта React

React – один из самых популярных интерфейсных фреймворков, доступных сегодня. Он представляет собой сдвиг парадигмы в разработке внешнего интерфейса, поскольку не использует DOM для управления внешним видом веб-интерфейса. Вместо использования DOM для отображения и скрытия компонентов в **одностраничном приложении (SPA)** по мере необходимости вы манипулируете объектом состояния², что заставляет React обновлять страницу.

Пользовательские интерфейсы React разрабатываются как набор компонентов, которые работают вместе с методами жизненного цикла или перехватами для обработки стандартных событий по умолчанию или даже собственных событий. Здесь я показываю немного больше работы с React, потому что это то, что я использую в своей повседневной работе, и потому что в следующей главе я расскажу, как использовать FastAPI с интерфейсом React.

¹ Бэкенд (англ. *backend*) – все, что работает на сервере, то есть не в браузере или на компьютере, подсоединенном к сети. – Прим. ред.

² В React состояние – это объект, представляющий собой части приложения, которые могут изменяться. У каждого компонента может быть свое состояние, которое находится в свойстве объекта `this.state`. – Прим. ред.

Создать проект React так же просто, как и другие, которые мы видели до сих пор. Просто нажмите **File | New Project**, выберите **React** из шаблонов и укажите локацию PyCharm. Как вы можете видеть на рис. 7.33, PyCharm использует `create-react-app`, который обычно запускается из командной строки:

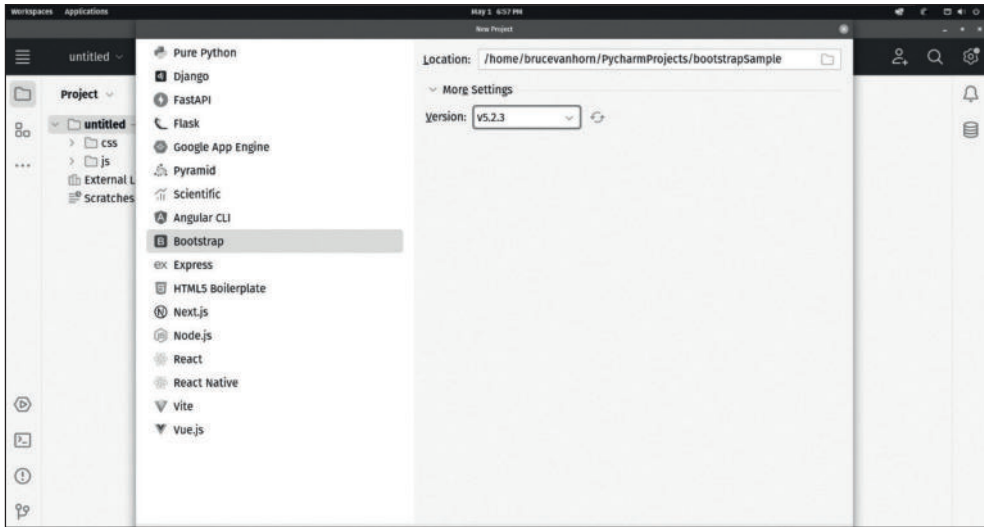


Рис. 7.33. Приложение Create React можно запустить из графического интерфейса PyCharm

PyCharm также предоставляет вам простой способ выбора между JavaScript, который используется по умолчанию, и TypeScript, который предпочитают многие разработчики. TypeScript – это вариант JavaScript, впервые разработанный Microsoft после выхода книги Дугласа Крокфорда «JavaScript, the Good Parts». TypeScript стремится исправить многие недостатки, такие как отсутствие строгой системы типов.

Другие фреймворки для фронтенда

PyCharm поддерживает несколько других современных интерфейсных фреймворков.

- **Angular** похож на React в том, что он использует компоненты, управляемые конечным автоматом. Самым большим отличием, помимо печально известного отсутствия обратной совместимости основных выпусков, является двунаправленная связь между компонентами. В React свойства детализируются сверху вниз. В Angular связь между компонентами идет в обоих направлениях. Это может затруднить отладку ваших приложений.
- **React Native** – это ответвление React, предназначенное для создания собственного пользовательского интерфейса для мобильных и настольных приложений. Последняя редакция .NET Framework от Microsoft включает вариант React Native для создания пользовательских интерфейсов Windows для настольных приложений.

- **Vite** – это современный фреймворк, целью которого является решение проблем с производительностью, связанных с раздуванием¹. Для проекта в любой другой среде легко потребовать десятки или даже сотни импортов модулей JavaScript, что замедляет производительность разработки и самого приложения. Vite использует передовые инструменты объединения для оптимизации процесса разработки интерфейса.
- **Vue** – еще один интерфейсный фреймворк, предоставляющий декларативную модель программирования на основе компонентов. Хотя React в значительной степени опирается на JSX, Vue использует стандартные HTML, CSS и JavaScript и может быть настроен для включения таких элементов, как маршрутизация, рендеринг на стороне сервера и многое другое.

КРАТКОЕ СОДЕРЖАНИЕ

В этой главе было рассмотрено использование PyCharm в качестве инструмента разработки интерфейса для веб-страниц и приложений. Мы обнаружили, что PyCharm обладает очень богатым набором возможностей в этой области, поскольку продукт JetBrains Web Storm интегрирован непосредственно в PyCharm Professional в виде предустановленного плагина.

Это обеспечивает все функциональные возможности надежного инструмента, специализирующегося на разработке HTML, JavaScript и CSS. Все те же возможности отладки, доступные нам в Python, также предоставляются в коде JavaScript, будь то на стороне клиента или сервера. Хотя мы не говорили о серверном JavaScript, возможность работать с проектами NodeJS имеется.

PyCharm Professional предоставляет нам множество шаблонов проектов как в традиционных стратегиях разработки на основе DOM, таких как HTML 5 Boilerplate и Bootstrap, так и в современных системах на основе конечных автоматов, таких как Angular и React.

Мы узнали, как использовать инструменты развертывания PyCharm для загрузки сайтов и приложений на удаленные серверы. Мы также узнали, как настроить внешние инструменты, такие как редакторы изображений, чтобы они запускались непосредственно из PyCharm.

Рассмотрев инструменты внешнего интерфейса, в следующей главе мы узнаем, как разработать полнофункциональное веб-приложение с использованием инфраструктуры Flask.

Вопросы

Ответьте на следующие вопросы, чтобы проверить свои знания по этой главе.

1. Какова цель HTML-кода? Как структурирован HTML-файл?
2. Какова цель CSS-кода? Как структурирован CSS-файл?

¹ Раздутая программа – это программа, имеющая слишком много дополнительных функций, на работу которых уходит непропорционально много ресурсов системы, в особенности если эти функции не нужны или малополезны для работы программы. – Прим. ред.

3. Какова цель кода JavaScript? В общем, что делает его одним из самых популярных языков веб-программирования?
4. Как включить таблицу стилей CSS или скрипт JavaScript в HTML-файл в PyCharm?
5. Что такое Emmet? Как он поддерживается PyCharm?
6. Какие варианты доступны при отладке JavaScript в PyCharm?
7. Какой самый безопасный способ развертывания веб-приложения из PyCharm?

Создание динамических сетевых приложений с Flask

Мне посчастливилось работать в сфере веб-разработки с момента ее зарождения. Я был инженером-программистом до того, как появилась Всемирная паутина, по крайней мере, та, которой пользовалась широкая публика. Я помню, как меня впервые попросили создать веб-приложение. Пришлось спросить, что это такое. Парень рассказал мне, и я помню, как подумал: «Ну, это глупо! Почему бы людям просто не использовать для этого CompuServe или **America Online (AOL)**?» Я думал, что интернет в лучшем случае станет причудой. Он был сложным, наполненным жаргоном, пользовательский интерфейс был ужасен по сравнению с онлайн-сервисами того времени, и все это казалось каким-то неуклюжим. По всей видимости, я ошибался.

Как только я это понял, я выучил HTML и JavaScript. CSS еще даже не существовало. Вскоре я столкнулся с проблемой возможностей HTML. Как вы хорошо знаете, HTML не является языком программирования. Это язык разметки текста, который управляет представлением статического контента. Самая ранняя версия JavaScript была не очень полезна. Вы можете проверять формы. Вот и все. Генерация динамического контента с помощью JavaScript не была доступна до появления HTML 3.

Как я уже сказал, я уперся о стену. Мне нужно было получить данные о взаимодействии пользователя из браузера и использовать их для взаимодействия с базой данных, создания файлов и многого другого. С HTML и JavaScript это было просто невозможно. Мне нужен был серверный язык. Изначально это был язык C. Даже это имело ограничения. Вам приходилось писать модули на C, которые могли бы взаимодействовать с веб-сервером Apache, используя интерфейс, называемый **интерфейсом общего шлюза (CGI)**.

Так было первые несколько лет. Написание динамических веб-приложений было трудной задачей и не имело ничего общего с возможностями сегодняшнего дня. Появились новые языки и парадигмы, которые сделали практику веб-разработки более доступной. Мой первый хороший опыт был с продуктом под названием **Cold Fusion**. Эта программа состояла из структуры, которая эффек-

тивно заменила требования к разработке CGI и позволяла создавать скрипты на специальном языке, называемом **языком разметки Cold Fusion (CFML)**. Это была смесь HTML-разметки и специализированных тегов. Веб-сервер распознает файлы с расширением `.cfml` и обрабатывает их иначе, чем обычные файлы HTML. Я смог очень легко получить доступ к базе данных Oracle и, в сочетании с множеством долгих ночей и творческим потенциалом моей юности, я создал программное обеспечение для графического конвейера, которое принесло моему работодателю патент на программное обеспечение.

CFML был частью растущей тенденции. Та же самая технология использовалась многими другими зарождающимися компаниями и стеками, в том числе следующими:

- Microsoft создала **Активные серверные страницы (ASP Classic)**,
- Sun Microsystems представила **Серверные страницы Java (JSP)**,
- **Препроцессор гипертекста (PHP)**,
- **Национальный центр суперкомпьютерных приложений (NCSA)** создал **Серверные включения (SSI)**, которые не были такими многофункциональными, как другие в этом списке, но тем не менее существовали как способ генерации динамического контента.

В этой главе мы сделаем квантовый скачок вперед и рассмотрим более современную структуру для создания динамического контента, который генерируется на стороне сервера, а не на стороне клиента в браузере. В частности, рассмотрим фреймворк под названием **Flask** – популярное и нестандартное решение для создания веб-приложений на Python. К концу этой главы вы поймете следующее:

- основы веб-разработки, такие как **архитектура клиент–сервер и модель запроса–ответа без сохранения состояния**, используемая в интернете;
- что такое Flask и чем он отличается от других фреймворков Python для веб-разработки;
- как создать приложение Flask в PyCharm;
- как работать с шаблонами **Jinja2** в PyCharm, которые используются для обслуживания динамического контента, смешанного с обычной HTML-разметкой, стилями CSS и интерактивностью JavaScript;
- как создать конечную точку RESTful API, которая возвращает данные в формате JSON, а не контент;
- как использовать функцию HTTP-запросов PyCharm для тестирования вашего API.

Имейте в виду, что эта глава не является руководством по Flask. Это пособие о том, как использовать PyCharm для работы с Flask. Если вы ищете полное руководство по Flask, посетите мой сайт <https://www.maddevskilz.com>. По Flask есть несколько расширенных руководств, в которых содержится глубокое погружение и влечет за собой создание целых проектов.

ТЕХНИЧЕСКИЕ ТРЕБОВАНИЯ

Чтобы пройти эту главу, как и остальную часть книги, вам понадобится следующее:

- установленный и работающий интерпретатор Python. Я буду использовать последнюю версию <https://python.org>;

- установленные копии `pip` и `virtualenv`. Вы получаете их автоматически при установке Python в Windows, а в macOS они включены в каждую операционную систему. Если вы используете Linux, необходимо отдельно установить менеджеры пакетов, такие как `pip`, и инструменты виртуальной среды, такие как `virtualenv`. В наших примерах будут использоваться `pip` и `virtualenv`;
- установленная и работающая копия PyCharm. Установка была описана в главе 2 на случай;
- пример исходного кода этой книги взят с GitHub. Мы рассмотрели клонирование кода в главе 2. Соответствующий код этой главы вы найдете на <https://github.com/PacktPublishing/Hands-On-Application-Development-with-PyCharm---Second-Edition/tree/main/chapter-08>.

ОСНОВЫ ВЕБ-ТЕХНОЛОГИЙ – АРХИТЕКТУРА КЛИЕНТ – СЕРВЕР

Когда я начал свою карьеру в сфере ИТ еще в 1991 году, я работал в компании **Electronic Data Systems (EDS)**. Это была другая эпоха. В то время любые серьезные вычисления выполнялись монолитными системами, называемыми **мейнфреймами**. Представьте себе мейнфрейм как невероятно мощный и масштабный компьютер, существовавший до эпохи персональных компьютеров и смартфонов. Это было похоже на суперкомпьютер, способный обрабатывать огромные объемы данных и выполнять сложные вычисления.

Мейнфреймы обычно размещались в специально спроектированных помещениях или центрах обработки данных, поскольку для правильной работы им требовалось много места и специальные системы питания и охлаждения. Типичный мейнфрейм обычно был размером с минивэн, а его отдельный **блок распределения питания (PDU)** был примерно в два раза больше обычной сушилки для одежды. Устройство **хранения данных с прямым доступом (DASD)** находилось в еще одном таком же большом прямоугольном металлическом ящике. Эти различные компоненты соединялись толстыми кабелями диаметром с рулон использованного бумажного полотенца. Модели серии IBM Z14 весили от 2500 до 4000 кг (от 5500 до 8800 фунтов).

В прошлом мейнфреймы обычно использовались крупными организациями, такими как банки, правительственные учреждения, университеты и крупные корпорации. Они отвечали за обработку и управление огромными объемами данных, запуск критически важных бизнес-приложений и поддержку работы целых предприятий. В моем случае я работал с мейнфреймами IBM, отвечающими среди прочего за управление сборочными линиями автомобилей для **General Motors (GM)**.

Мейнфреймы были известны своей надежностью, безопасностью и высокой производительностью. Они могли выполнять несколько задач одновременно и обеспечивать быстрое время отклика даже при работе с обширными рабочими нагрузками. Люди получали доступ к мейнфреймам через терминалы или другие подключенные устройства для выполнения задач или получения информации.

С развитием технологий и появлением персональных компьютеров роль мейнфреймов изменилась. Хотя они по-прежнему играют жизненно важную роль в некоторых отраслях, многие вычислительные задачи, которые когда-то

были эксклюзивными для мейнфреймов, теперь решаются распределенными системами, относительно дешевыми рек-системами на базе процессоров Intel и IBM Power, облачными вычислениями и устройствами меньшего размера, такими как ноутбуки и смартфоны.

Мейнфреймы были и, я полагаю, до сих пор остаются очень дорогими во владении и эксплуатации. Аппаратное обеспечение было не только дорогим, но и, как правило, для поддержания работоспособности системы требовалась команда опытных компьютерных операторов и специалистов по обслуживанию. Цена была недоступна для всех, кроме крупнейших корпораций и университетов¹.

Меньшим компаниям и даже меньшим странам приходилось тратить время на мейнфреймы других людей, чтобы получить доступ к вычислениям в больших масштабах. Это, собственно, и была услуга, оказанная EDS. У нас были площади в нескольких очень крупных центрах обработки данных, расположенных по всему миру, и мы продавали время и предоставляли услуги почти всем компаниям из списка Fortune 500. В конечном итоге затраты, а также последствия закона Мура привели к падению мейнфреймов.

Закон Мура – это наблюдение и прогноз, сделанный Гордоном Муром, соучредителем корпорации Intel, в 1965 году. Он гласит, что количество транзисторов на микрочипе удваивается примерно каждые два года, что приводит к значительному увеличению вычислительной мощности и производительности, одновременно и снижению стоимости электронных устройств.

Первоначально Мур отметил, что этот экспоненциальный рост плотности транзисторов начался со времени изобретения интегральных схем, и предсказал, что он будет продолжаться в обозримом будущем. На протяжении многих лет закон Мура оставался в высшей степени верным: достижения в технологии производства полупроводников позволили создавать все более миниатюрные по размеру транзисторы и более сложные интегральные схемы.

Удвоение плотности транзисторов на процессорах каждые два года имело глубокие последствия для области вычислений. Это позволило разработать более мощные и эффективные компьютеры с увеличенной скоростью обработки, объемом памяти и возможностями хранения данных. Поскольку на чипе можно разместить больше транзисторов, общая производительность электронных устройств улучшается, а их физический размер уменьшается.

Когда компьютеры стали меньше и появились такие архитектуры микросхем, как **вычисления с сокращенным набором команд (RISC)** и в конечном итоге архитектура Intel x86, появилась новая модель вычислений: **клиент-сервер**.

Мейнфреймы были централизованы, и доступ к ним осуществлялся с помощью «глупых» терминалов. Эти терминалы не имели ни вычислительных возможностей, ни памяти, а памяти хватало только на поддержание комму-

¹ В СССР научные сотрудники академических институтов использовали их бесплатно, по собственному опыту работы в одном из вычцентров АН СССР – по собственным нуждам, доступ был чисто формальным. Его использование контролировалось разве что протоколом задействованных ресурсов использующих мейнфрейм пользователей, вывешенном на стене вычислительного зала, который носил чисто показательный характер, чтобы отдельные программисты не «зарывались». – *Прим. ред.*

никационного буфера для отправки всего, что вы вводили на клавиатуре, на мейнфрейм для обработки.

В архитектуре клиент–сервер часть вычислений, хранилища и памяти перекладывалась на локальный клиент, которым обычно был ПК. Клиент был подключен через **локальную сеть (LAN)** к серверу, который, как правило, был более мощным, чем ПК, и мог выполнять вычислительные нагрузки корпоративного уровня. Обычно у вас было клиентское программное обеспечение, которое представляло собой пользовательский интерфейс настольного компьютера, работающий в операционной системе ПК. Клиентское программное обеспечение взаимодействует с централизованным программным обеспечением, работающим на сервере.

Помимо клиентского и серверного оборудования, примерно в то же время появился еще один последний компонент: стандартизированный сетевой протокол, известный как **протокол управления передачей / интернет-протокол (TCP/IP)**. Я учился в колледже до появления TCP/IP, и для того, чтобы взаимодействовать с мейнфреймом Университета Оклахомы, мне приходилось держать под рукой стопку дискет, на которых содержался странный, несовпадающий набор протоколов связи. Некоторые системы использовали протокол под названием **Ядро для эффективного, удаленного и множественного взаимодействия компьютеров (KERMIT)**¹. Еще у меня были диски для *XMODEM*, *YMODEM* и *ZMODEM*. В зависимости от типа компьютера, к которому я хотел получить доступ, мне приходилось использовать другой протокол. TCP/IP изменил все это благодаря стандартному набору протоколов, поддерживаемому всем: от мейнфреймов и ПК до современных смартфонов и тостеров, подключенных к интернету.

Если для вас это звучит как интернет, вы будете правы, но с некоторыми оговорками:

- клиентский сервер работал очень медленно и часто даже не в полном дуплексном режиме, а это означало, что данные могли передаваться одно- временно только в одном направлении;
- большинство клиентских программ не имели настоящего графического интерфейса и не поддерживали взаимодействие с мышью. Они были известны как интерфейсы «зеленого экрана», поскольку отображались с использованием текстовых меню на монохромных экранах, которые часто были зелеными². Более поздние приложения «толстых клиентов» имели настоящие графические интерфейсы, обычно написанные на Java, Visual Basic, C++ или Delphi. Их называли толстыми, потому что размер программы был достаточно большим, и загрузка графического интерфейса могла занять много часов при обычном соединении. Это резко контрастирует с типичным веб-приложением, работающим в современном интернете;
- клиентское программное обеспечение всегда разрабатывалось с учетом специализированного набора вариантов использования. Напротив,

¹ Тоже юмор создателей. Лягушонок Кермит – центральный персонаж мегапопулярного телевизионного шоу Джима Хенсона «Маппет-шоу». – *Прим. ред.*

² То есть на них были зеленые символы на черном фоне; также были распространены дисплеи с оранжевым цветом символов на черном фоне и белым на черном. – *Прим. ред.*

сегодняшние веб-браузеры клиента могут использоваться для запуска любого типа программного обеспечения, от обычных приложений для обработки текста до конкретных бизнес-приложений.

Современный интернет – это результат естественной эволюции клиент-серверной архитектуры. «Легкое» клиентское программное обеспечение, веб-браузер, подключается к централизованному серверу, где основная часть реальной работы выполняется с использованием общепринятых сетевых протоколов.

ИЗУЧЕНИЕ МЕХАНИЗМА ЗАПРОСА–ОТВЕТА В HTTP – КАК ВЗАИМОДЕЙСТВУЮТ КЛИЕНТЫ И СЕРВЕРЫ

Одним из величайших достижений 1980-х годов, помимо лака для волос с сильной фиксацией, стала разработка универсального набора сетевых протоколов, известных как TCP/IP. Если вы новичок в этом деле, правильно произносить это буквами: *tea sea pea eye pea*. Слеш бесшумен, как ниндзя.

Потребовалось некоторое время, чтобы он был принят повсеместно, но в конечном итоге это произошло, и протоколы TCP/IP составляют основу современной сети. Хотя существует множество полезных протоколов, выполняющих множество функций, я хочу сосредоточить ваше внимание на **протоколе передачи гипертекста (HTTP)**. Вы можете пойти дальше и включить безопасный аналог *HTTPS*, где *S* означает *безопасный*. Фактически они работают одинаково, за исключением того, что HTTPS шифруется.

Механизм запроса–ответа включает в себя цепочку событий, описывающую диалог, происходящий между веб-браузером или клиентом и веб-сервером. Вы можете наблюдать, как разворачивается этот разговор, на рис. 8.1:

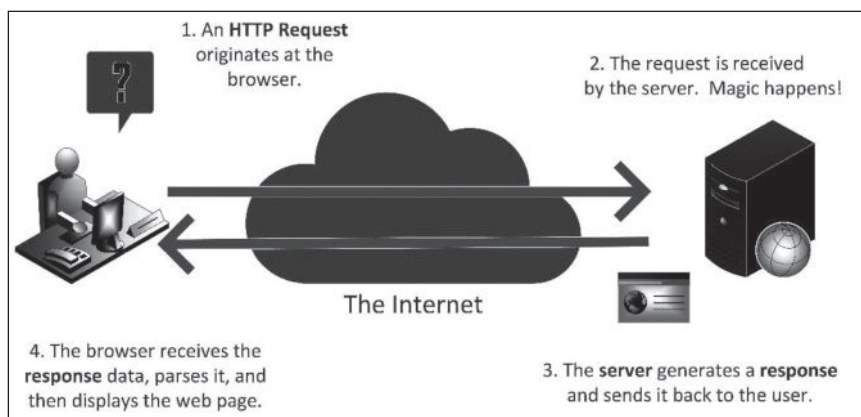


Рис. 8.1. Механизм запроса-ответа в HTTP передает запрос от браузера на сервер, который вычисляет ответ и отправляет его обратно браузеру

Механизм запроса–ответа, используемый HTTP, является фундаментальной моделью связи между клиентами, такими как веб-браузеры или мобильные приложения, и серверами во Всемирной паутине. Это работает следующим образом.

1. **HTTP-запрос исходит из браузера:** процесс начинается, когда клиент отправляет HTTP-запрос на сервер. Запрос состоит из определенного метода HTTP, такого как GET, POST, PUT или DELETE, указывающего желаемое действие, которое необходимо выполнить с ресурсами сервера, а также дополнительные заголовки и – в некоторых случаях – тело запроса, содержащее данные.
2. **Запрос получен сервером:** после получения запроса сервер обрабатывает информацию, предоставленную в запросе. Это может включать доступ к базам данных, выполнение вычислений или выполнение других операций на стороне сервера в зависимости от характера запроса. Самый простой запрос – это запрос HTML-документа или другого файла, который просто возвращается.
3. **Сервер генерирует ответ и отправляет его обратно пользователю:** после обработки запроса сервер генерирует ответ HTTP. Ответ содержит соответствующий код состояния, указывающий результат запроса. Например, успешный запрос имеет в заголовке ответа код состояния 200. Если вы запрашиваете ресурс на сервере, которого там нет, вы получите код 404, сигнализирующий, что ресурс не найден. Заголовок ответа также включает дополнительные поля, предоставляющие дополнительную информацию, и тело ответа, содержащее запрошенные данные или любую соответствующую информацию.
4. **Браузер получает данные ответа:** клиент получает ответ HTTP от сервера и обрабатывает содержащуюся в нем информацию. Это может включать в себя рендеринг HTML-контента, обработку данных или выполнение других действий на основе ответа.
5. **Цикл запрос–ответ завершен:** с получением ответа цикл запрос–ответ завершается. Клиент может решить отправить дополнительные запросы на сервер для выполнения дальнейших взаимодействий, или процесс может завершиться.

Этот механизм запроса–ответа формирует основу обмена информацией между клиентами и серверами через HTTP, что позволяет клиентам запрашивать ресурсы или выполнять действия на сервере, а сервер отвечает соответствующими результатами или необходимой информацией. Этот цикл обеспечивает динамичный и интерактивный характер веб-приложений и сервисов.

Следует помнить, что HTTP не имеет состояния. Это означает, что каждый цикл запрос–ответ является дискретным. В HTTP нет собственного способа обмена или сохранения данных между запросами.

Что такое Flask?

Flask – это беспристрастный фреймворк для работы с механизмом запрос–ответ, присутствующим в HTTP. Он делает одно и только одно: помогает получать запросы в простую объектную структуру Python, а затем создавать ответы с использованием кода Python.

Вернемся к слову *беспристрастный*. Под этим я подразумеваю, что Flask по своей конструкции обрабатывает только цикл запроса–ответа. Я понимаю, что

уже говорил это, но стоит повторить. Если вы сравните Flask с его виртуальной противоположностью – Django, – разница будет поразительна.

Django чрезвычайно своеволен в том, как создавать веб-приложения. Он диктует вам файловую структуру, шаблоны приложений и базу данных, которая будет использоваться. Он имеет собственный реляционный преобразователь объектов, собственный механизм ответа на запросы и собственный набор соглашений по кодированию. Короче говоря, Django будет изначально определять ваш стек и большинство архитектурных деталей вашего проекта.

Flask предлагает некоторые предложения, но они не высечены в камне, и вам не обязательно использовать их, если вы этого не хотите. Несколько лет назад я переписал флагманский программный продукт своей компании Visual Storage Intelligence (см. <https://www.visualstorageintelligence.com>) как приложение Flask исключительно потому, что он не является беспристрастным. Я считаю себя экспертом в выборе лучшего стека, учитывая мои знания, опыт и понимание бизнес-требований моей компании.

Например, я практически никогда не использую ORM. У меня глубокий опыт работы с SQL и системами реляционных баз данных. Я могу писать и настраивать запросы, хранимые процедуры и представления для создания быстрого и гибкого веб-приложения в различных коммерческих базах данных и базах данных с открытым исходным кодом. ORM предназначен для того, чтобы взять все это из ваших рук и предоставить вам уровень абстракции над базой данных, поэтому разработчику нужно иметь дело только с объектами.

ORM, по сути, является черным ящиком. Большинство разработчиков не знают, как он работает и как повысить производительность запросов, генерируемых ORM. Для меня это просто накладные расходы. Лично я бы предпочел построить эти колесики и заставить их крутиться самостоятельно. Кстати, если какой-либо из этих терминов, связанных с базами данных, вызывает у вас недоумение, следите за обновлениями. Подробнее о базах данных я расскажу в главе 11.

Дело в том, что Flask не волнует, как я взаимодействую со своей базой данных. Его не волнует, как я структурирую свое приложение, и ему не важно, как выглядит мой стек. Он делает только две вещи, и одна из них является опционной.

ОБРАБОТКА И МАРШРУТИЗАЦИЯ ЗАПРОСОВ И ОТВЕТОВ С ПОМОЩЬЮ WERKZEUG

Первое, что Flask делает для нас, – это упрощает работу с механизмом запрос-ответ в HTTP. Строго говоря, входящий запрос является двоичным по своей природе. Но, как разработчики, мы предпочитаем иметь дело с текстоподобными объектными абстракциями бинарных структур. Работа с двоичными файлами напрямую – это 1939 год. К счастью, существует библиотека Python, которая справляется с этим, под названием Werkzeug.

Название библиотеки Werkzeug происходит от немецкого языка и переводится как «инструмент» или «орудие». Платформа Flask, включая лежащую в ее основе служебную библиотеку Werkzeug, изначально была разработана Армином Ронахером, немецким разработчиком программного обеспечения.

Название Werkzeug было выбрано, чтобы отразить характер библиотеки как универсального и мощного набора инструментов для создания веб-приложений. Точно так же, как мастер полагается на набор инструментов для создания и формирования своей работы, разработчики могут использовать Werkzeug для управления различными аспектами веб-разработки, такими как маршрутизация, обработка запросов и утилиты HTTP. Flask построен на основе Werkzeug; он добавляет дополнительные абстракции и функции, обеспечивая легкий и удобный веб-фреймворк.

Flask использует функции Werkzeug для обработки низкоуровневых деталей HTTP-запросов и ответов, что позволяет разработчикам сосредоточиться на быстром и эффективном создании веб-приложений. Результатом является возможность создавать веб-приложения так же просто, как и любое другое приложение Python: вы создаете функции для обработки входящего HTTP-запроса, который возвращает правильно отформатированный HTTP-ответ. Если вы можете написать функцию на Python – вы можете создать веб-приложение.

Реализация во Flask выглядит так:

```
@app.route('/hello')
def hello_world():
    return 'Hello, World!'
```

Как видите, это простая функция Python, которая не принимает аргументов и возвращает строку. Единственное, что здесь странное, – это украшение над определением функции. Мы узнаем об этом гораздо больше позже. А пока поймите, что это украшение соответствует маршруту в URL-адресе вашего приложения. Естественно, вы научитесь делать больше, чем просто обрабатывать такой простой запрос, но моя точка зрения остается неизменной: если вы можете создать функцию на Python, вы можете написать и веб-приложение!

Использование этой функции Flask не подлежит обсуждению. Если вы не хотите использовать эту абстракцию Werkzeug, не нужно использовать Flask. Flask опять же не настойчив, а это означает, что это действительно единственное, что его «заботит». Мы увидим контраст в главе 12. Django очень настойчив. Он хочет, чтобы вы использовали заранее определенный им стек. Фреймворку Flask все равно. Тем не менее он содержит одно сильное предложение, которое вы можете спокойно игнорировать, если хотите.

Создание шаблонов с помощью Jinja2

Flask по умолчанию предлагает библиотеку шаблонов под названием Jinja2. Когда вы устанавливаете Flask, то получаете Jinja2 в качестве зависимости. Системы шаблонов предназначены для того, чтобы вы могли вставлять контент в документ разметки. Вот пример скрипта Jinja2, который генерирует некоторые элементы в неупорядоченном списке на HTML-странице:

```
from jinja2 import Template

# Define the template
template_str = ''
```

```
<ul>
{% for item in items %}
<li>{{ item }}</li>
{% endfor %}
</ul>
'''
```

Шаблон представляет собой просто строку, содержащую некоторые специальные символы, указывающие, где должен выполняться код, подобный Python. Так работают все языки шаблонов, от классического ASP до JSP и PHP. Процесс Flask получает запрос от веб-сервера. Он анализирует запрос, и здесь мы отображаем ответ с помощью нашего шаблона. Сам шаблон представляет собой фрагмент разметки. Создается переменная `template_str`, которой присваивается пустая строка. Помните, что это не код Python, это код, подобный Python, поскольку он существует в контексте разметки. Одинарные кавычки экранируются одинарными кавычками, что означает, что `` интерпретируется в коде как одинарная кавычка.

Как видите, у нас есть цикл `for`, перебирающий некоторый список с именем `items`, который будет передан в шаблон в качестве данных. Код в шаблоне обозначен `{%}` и `%}`. Каждый `` содержит выражение, заключенное в двойные фигурные скобки. В этом случае выражение – это просто элемент итерируемой переменной, определенный с помощью цикла.

В моем примере строка шаблона определена в коде, но в реальной жизни шаблон обычно находится в файле, поскольку его гораздо проще поддерживать. Далее мы создаем объект `template` и передаем строку `template` в конструктор шаблона:

```
# Create the Jinja2 template object
template = Template(template_str)
```

Теперь мы вернулись к простому коду Python. Мы создадим список под названием `items` и заполним его:

```
# Generate the list of items
items = ['Item 1', 'Item 2', 'Item 3', 'Item 4', 'Item 5', 'Item 6', 'Item 7',
'Item 8', 'Item 9', 'Item 10']
```

Теперь визуализируем `template`, который создает новую строку, присвоенную переменной с именем `output`:

```
# Render the template with the items
output = template.render(items=items)
```

Здесь мы печатаем выходные данные, но в контексте приложения Flask выходные данные будут возвращены в качестве ответа:

```
# Print the rendered output
print(output)
```


Jinja2 имеет мощный набор языковых функций, общих для большинства хороших систем шаблонов. Однако помните, что Flask позиционирует себя как *ненастойчивый*. Вам не обязательно использовать Jinja2. Существуют и другие библиотеки шаблонов, или вы можете полностью отказаться от использования системы шаблонов. В продукте моей компании *Visual Storage Intelligence* я использовал Jinja2 для версии 4 продукта. В то время я не очень хорошо знал Python и привык к шаблону Microsoft MVC для веб-приложений C#. Я ограничил свою кривую обучения, чтобы уложиться в срок. Jinja2 совсем несложно освоить. В основном это HTML с некоторым дополнительным синтаксисом. В версии 5 я вернулся и заменил уровень пользовательского интерфейса приложения интерфейсом React. Это не потребовало никаких изменений в приложении Flask, кроме удаления шаблонов, которые я больше не использовал.

Напоминание: это не книга о Flask

Я потратил некоторое время на основы высокого уровня Flask на случай, если вы столкнетесь с этим как новичок в веб-разработке. Хотя может показаться, что я собираюсь научить вас Flask, я намерен лишь охватить особенности разработки Flask, связанные с PyCharm. Пока мы будем создавать проект, я не буду глубоко вникать в то, как и почему код такой, какой он есть.

Таким образом, если вы действительно новичок и заинтригованы намеренно немногословным освещением Flask, которое я предоставляю здесь, и вам нужен правильный учебник, я включу более подробные ресурсы в раздел «Дальнейшее чтение» в конце этой главы.

ПРИМЕЧАНИЕ О НАИМЕНОВАНИИ ФАЙЛОВ И ПАПЕК

Прежде чем мы перейдем к созданию следующего проекта, я хочу дать вам несколько советов, основанных на принципе, который я всегда указываю своим ученикам. **Интернет не терпит пробелов.** Пробелы в URL-адресах не допускаются, поскольку они являются зарезервированными символами. Когда вы включаете пробел в URL-адрес, он должен быть закодирован как %20, чтобы считаться действительным. Веб-серверы автоматически обрабатывают эту кодировку, но кодирование пробелов вручную может привести к проблемам с читаемостью URL-адресов, а также может вызвать проблемы при попытке запуска ваших проектов на локальном компьютере.

В веб-проекте большинство путей вашего проекта в какой-то момент станут URL-адресами. Это может произойти, даже если ваш проект не предназначен для интернета. Проблема усугубляется тем, что разные операционные системы по-разному обрабатывают символы в своих файловых системах. Имена файлов Windows нечувствительны к регистру. Если вы назовете свой файл или папку MyProject, а затем попытаетесь создать папку с именем myproject, вы получите коллизию, поскольку папка уже существует, несмотря на разницу в прописных и строчных буквах в имени. В Linux и macOS имена файлов и папок чувстви-

тельны к регистру. Совершенно нормально хранить `MyProject` и `myproject` в одной папке.

Я предлагаю выбрать стандарт, который вам нравится, и использовать его. Для проектов Python наиболее распространенным соглашением является присвоение файлам имен в **snake_case** (змеином регистре), названном так из-за его использования в Python. Змеиный регистр предполагает использование всех строчных букв и замену пробела на символ подчеркивания. Вы не сможете создать проект `мурproject` в папке `MyProject`. Вам придется назвать его `му_project`.

С учетом вышесказанного большинство моих проектов в этой книге названы с использованием значений по умолчанию PyCharm, которые часто отображаются в регистрах `Camel` или `Pascal`. В этих стандартах, обычно используемых разработчиками Java и C#, пробелы опускаются, а граница между словами обозначается заглавными буквами: `myProject` и `MyProject` могут быть примером регистров `Camel` и `Pascal` соответственно. Я подозреваю, что использование верблюжьего регистра в именах проектов по умолчанию, вероятно, связано с тем фактом, что PyCharm является родоначальником Java IDE. Существуют и другие стратегии устранения пробелов в файлах кода. Регистр `Kebab` («шашлык») распространен в разработке на JavaScript и использует тире вместо подчеркиваний. `MyProject` превращается в `my-project`.

Я искренне рекомендую отказаться от привычки использовать пробелы в именах папок и файлов в вашей практике программирования. В эту рекомендацию входят все вышестоящие папки, например домашняя папка вашей операционной системы.

СОЗДАНИЕ ПРИЛОЖЕНИЯ FLASK В PYCHARM PROFESSIONAL

Инструменты для Flask – это функция, доступная только в профессиональной версии PyCharm. Естественно, вы можете создать приложение Flask в бесплатной версии PyCharm, но создавать файлы, настраивать профили запуска, специальную отладку и т. д. вы будете самостоятельно.

Чтобы создать приложение Flask в PyCharm, просто выберите **File | New Project** и шаблон **Flask**, как показано на рис. 8.2.

Я пронумеровал наиболее важные части на рис. 8.2.

1. В диалоговом окне **New Project** PyCharm Professional вы найдете шаблон для проектов **Flask**.
2. Эта часть ничем не отличается от любого проекта, который мы делали до сих пор. Заполните локацию проекта.
3. Создайте свою виртуальную среду. Как только все будет заполнено, PyCharm создает и активирует виртуальную среду и устанавливает Flask и его зависимости.
4. Этот раздел уникален для проектов Flask. Я упоминал, что Flask рекомендует использовать шаблонизатор Jinja2. Вы можете отказаться от этого, и PyCharm позаботится об этом за вас. Также есть настройка папки, которую вы хотите использовать для своих шаблонов Jinja2. Мы собираемся сохранить настройки по умолчанию, и я рекомендую, если вы собираетесь использовать шаблоны Jinja2, оставить настройки такими, какие они есть, потому что именно этого ожидает большинство разработчиков.

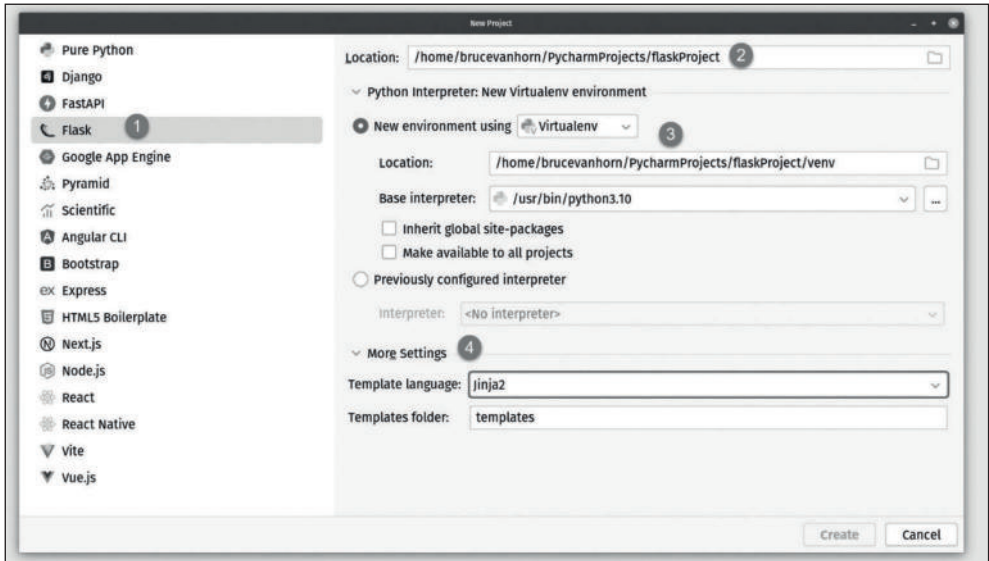


Рис. 8.2. Чтобы создать новый проект Flask в PyCharm Professional, выберите шаблон в диалоговом окне New Project

После того как вы все заполнили, нажмите кнопку **Create** в нижней части диалогового окна, и PyCharm настроит для вас ваш проект, включая создание некоторого стартового кода, как показано на рис. 8.3.

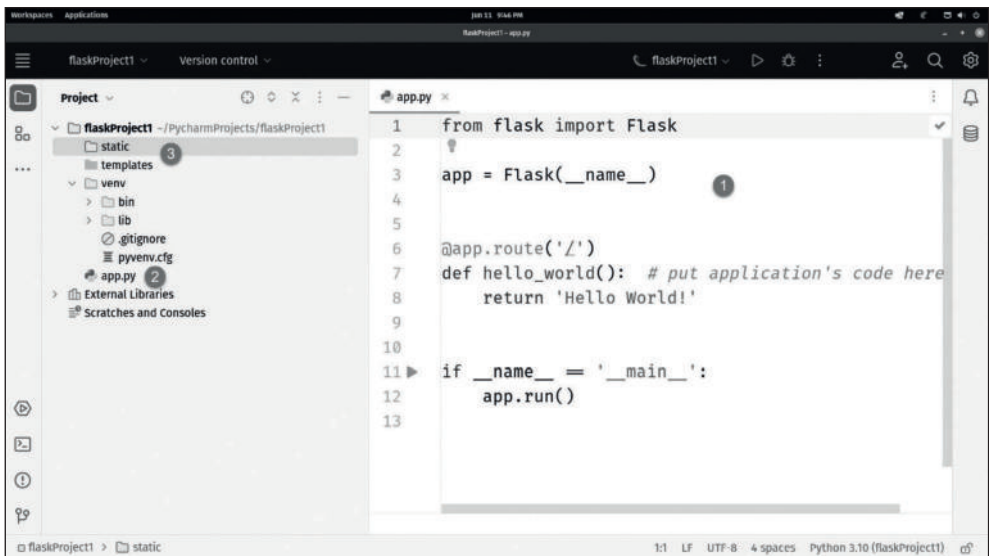


Рис. 8.3. PyCharm автоматически генерирует ваш проект Flask

В позиции (1) мы видим сгенерированный код PyCharm. По сути, это идиома «Hello, World», описанная ранее, за исключением того, что она отвечает на кор-

новой маршрут вашего веб-приложения. Вверху мы импортируем Flask, затем создаем экземпляр Flask в качестве переменной `app`. Теперь, когда вы это знаете, оформление маршрута в строке 6 имеет больше смысла; `app.route` просто исходит из экземпляра `app Flask`.

Весь этот код содержится в позиции `app.py` (2) на рис. 8.3. Имя не является обязательным, и вы можете изменить его, если хотите. В позиции (3) вы также можете видеть, что PyCharm сгенерировал две папки для шаблонов Jinja2 и статических файлов, таких как ваши изображения, файлы CSS и JavaScript.

СОЗДАНИЕ ДИНАМИЧЕСКОГО ВЕБ-ПРИЛОЖЕНИЯ

Программа «Hello, World», созданная PyCharm, является хорошей отправной точкой для новых приложений. PyCharm предоставляет вам полезные познавательные подсказки на случай, если вы давно не создавали приложение Flask с нуля. Естественно, мы заменим «Hello, World» чем-то более полезным.

Давайте создадим очень простое приложение для каталогизации новых библиотек Python! Проекты такого типа обычно выполняются с использованием базы данных, но это не обязательно. Наше приложение будет содержать список библиотек Python вместе с их описанием и оценкой от 1 до 5, насколько они, по нашему мнению, полезны. Чтобы выполнить часть базы данных, мы собираемся просто использовать массив списков в памяти. Использование этого подхода избавляет от необходимости глубоко погружаться в возможности базы данных PyCharm, чего не будет до главы 11. Мы также воспользуемся всеми возможностями, чтобы охватить некоторые функции, рассмотренные ранее, в практической обстановке, например используя функции HTML из последней главы.

НАСТРОЙКА СТАТИЧЕСКИХ ЧАСТЕЙ

Самый простой способ начать создание веб-приложения – заставить работать статические части. Я говорю о нединамических частях приложения, таких как страница `index.html` с ее базовой структурой, любым CSS, изображениями и JavaScript, которые могут нам понадобиться.

Начните с щелчка правой кнопкой мыши по папке `Templates` и создайте новый файл с именем `index.html`. Это будет шаблон Jinja2, поэтому он будет находиться в папке `Templates`, а не в папке `static`.

Обратите внимание, что мы создали простой HTML-файл. Для файла Jinja2 не существует специального типа файла. Файл создается с использованием системы шаблонов Emmet, которую мы рассмотрели в главе 7. Как вы, надеюсь, помните, **Emmet** – это язык шаблонов для генерации кода, в частности HTML. Он предоставляет шаблон времени разработки, который обеспечивает подход «заполните пробелы». Когда вы добавили файл HTML, то обнаружили, что код был сгенерирован, но IDE перенаправила вас прямо к атрибуту `title` в файле HTML, что позволило вам заполнить это пустое место в шаблоне HTML. Вы можете увидеть мой вариант на рис. 8.5.

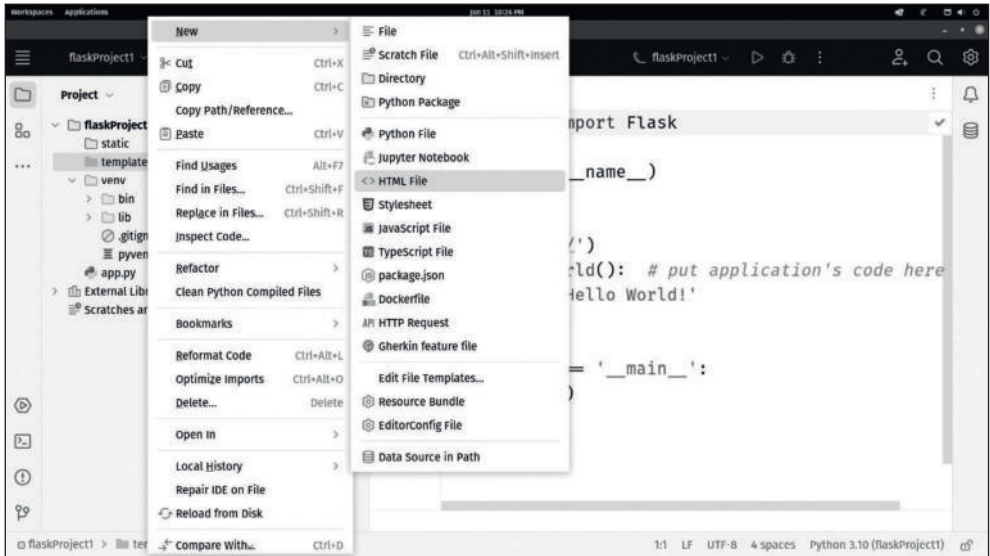


Рис. 8.4. Кликните правой кнопкой мыши папку Templates и создайте новый файл HTML

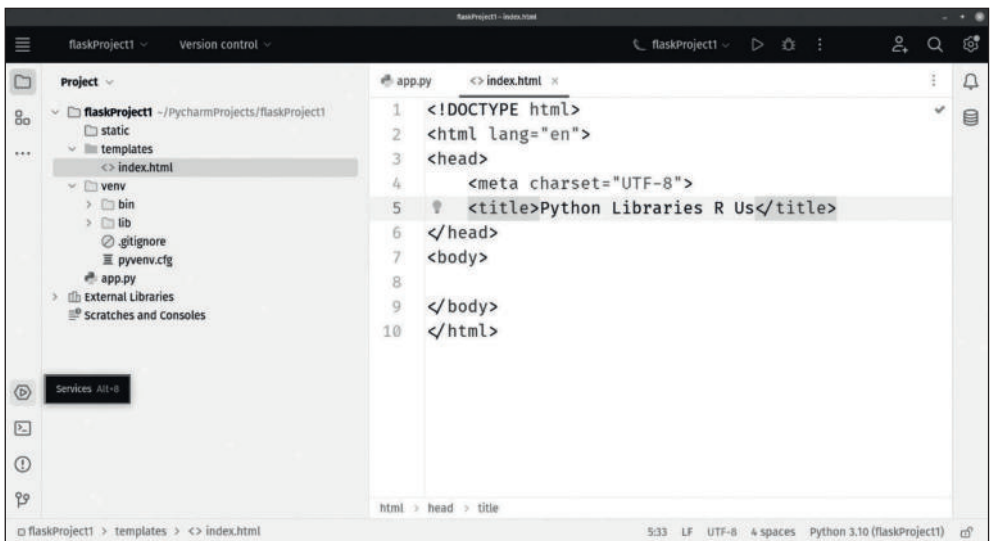


Рис. 8.5. Шаблон создания HTML помещает курсор в тег заголовка, чтобы вы могли заполнить эту часть страницы

Далее давайте изменим содержимое тега `<head>` следующим образом:

```
<head>
  <meta charset="UTF-8">
  <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3
    .0/dist/css/bootstrap.min.css" rel="stylesheet"
    integrity="sha384-9ndCyUaIbzAi2FUVXJi0CjmCapSm07SnpJef048
    6qhLnuZ2cdeRh002iuK6FUUVM"
    crossorigin="anonymous">
  <title>Python Libraries R Us</title>
</head>
```

Это приводит к использованию загрузочных библиотек CSS и JavaScript из **content delivery network (сети доставки контента, CDN)**. Если вы не хотите вводить все это, можете скопировать код из кода репозитория книги или перейти на <https://getbootstrap.com> и найти последние новости в разделе **Getting Started** на их сайте, который обычно включает копируемые ссылки на их CDN.

Мы используем CDN, поэтому нам не нужно хранить эти общие файлы в нашем проекте. Кроме того, CDN обычно обслуживает такие файлы быстрее, чем ваш собственный веб-сервер, когда вы запускаете свое замечательное приложение в производство.

Далее измените содержимое тега `<body>` на следующее:

```
<body>
  <h1>Python Libraries R Us</h1>
  <h2>All the libraries that are fit to use!</h2>
</body>
```

Далее мы собираемся использовать функцию Emmet, о которой узнали в главе 7, для создания заголовка таблицы HTML. Введите этот код Emmet в редактор в новой строке чуть ниже тега `<h2>`:

```
table>thead>tr>th*4
```

Это сокращение создаст таблицу, за которой следует тег `thead`, с одной строкой таблицы (`tr`) и четырьмя полями заголовка (`th*4`). Нажмите *Tab*, чтобы развернуть сокращение. Плагин Emmet PyCharm сгенерирует ваш код и перенаправит вас прямо к содержимому тегов `th`, как показано на рис. 8.6.

Как только Emmet расширит ваш код, вы увидите подсказки внутри тегов `th`, позволяющие редактировать содержимое, как показано на рис. 8.7.

Обычно для перемещения между полями используется **Tab**, но в PyCharm клавиша *Tab* привязана к действию, которое расширяет Emmet, как мы только что видели. Вы не можете использовать *Tab* для навигации, как и следовало ожидать, если вы переходите из другого редактора с поддержкой Emmet. Вместо этого вам нужно будет выяснить, какое клавиатурное сокращение используется в вашей системе, поскольку это будет зависеть от того, какую раскладку сочетаний клавиш вы настроили при установке PyCharm. Я выбрал раскладку Windows, поэтому перемещение между полями у меня осуществляется с помощью *Alt+Shift+J*.

Внутри первого тега `th` введите `Library Name`. Затем используйте *Alt + Shift + J*, чтобы перейти к следующему полю, и измените его на `Description`. Третье поле будет называться `Rating`, а четвертое — `URL`.

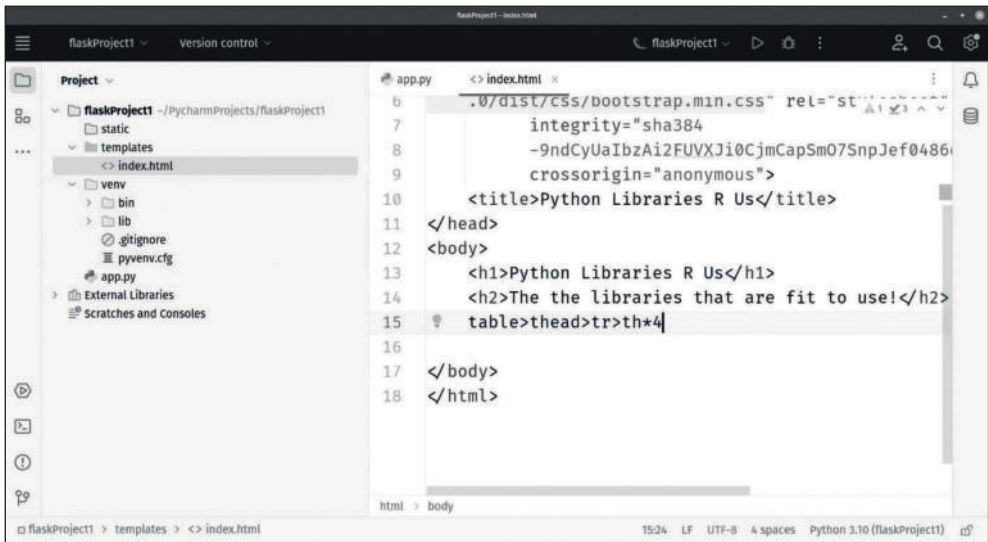


Рис. 8.6. Emmet расширяет код до полного кода, необходимого для заголовка таблицы

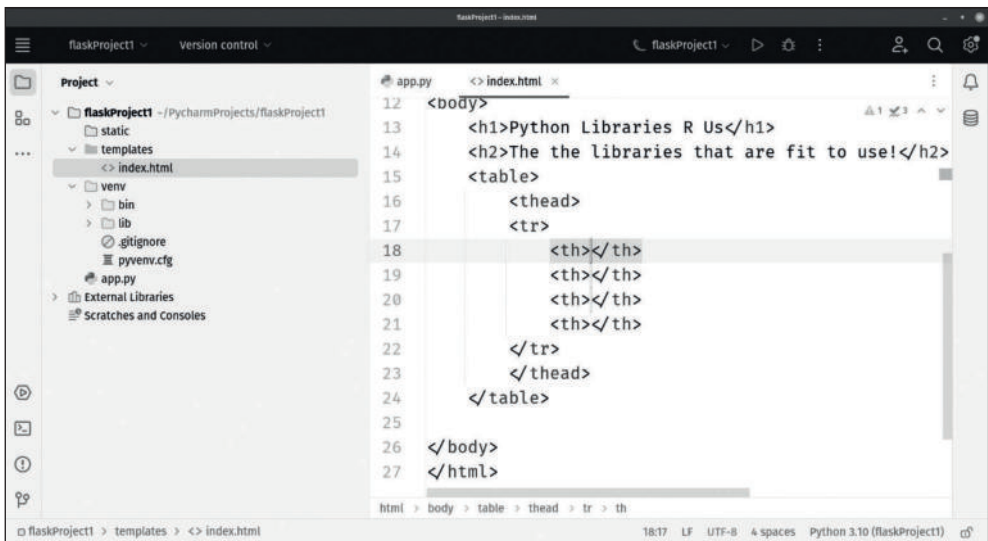


Рис. 8.7. Emmet расширяет сокращение в строке 15 на рис. 8.6 до того, что мы видим здесь

Если **Alt + Shift + J** вам не подходит, давайте выясним, что поможет. Зайдите в настройки PyCharm и найдите настройки **Keymap**, как показано на рис. 8.8.

Я ввел Emmet в поле поиска и вижу, что мои настройки **Navigate > Next / Previous Emmet Edit Point** – это **Alt + Shift +]** и **Alt + Shift + [** соответственно. Если у вас стоят другие, вы увидите, какие они здесь, и, как мы узнали, вы можете изменить их на все, что захотите, при условии, что это изменение не противоречит чему-то еще.

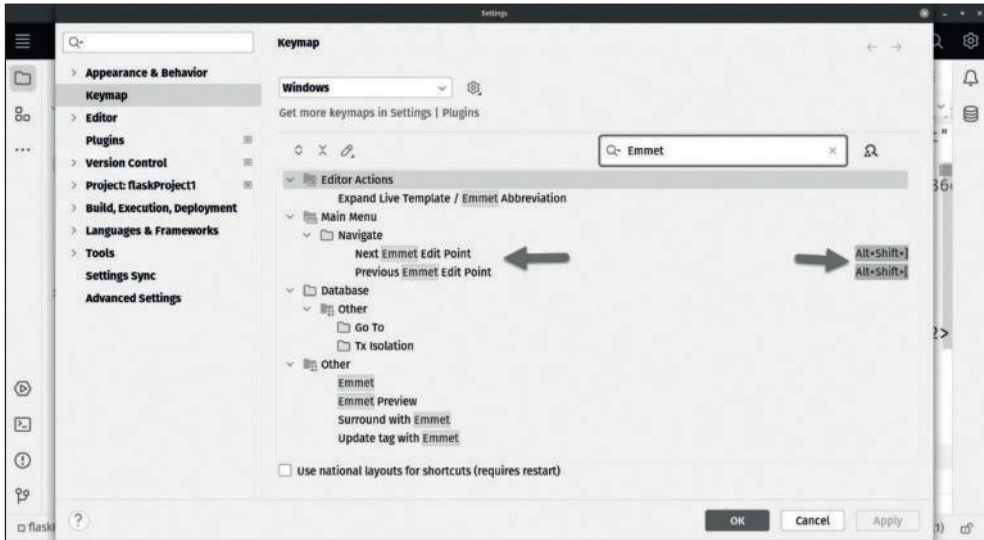


Рис. 8.8. Ваши настройки навигации Emmet находятся и устанавливаются здесь, в Settings

На этом этапе HTML-код вашей table должен выглядеть следующим образом:

```
<table>
  <thead>
    <tr>
      <th>Library Name</th>
      <th>Description</th>
      <th>Rating</th>
      <th>URL</th>
    </tr>
  </thead>
</table>
```

Добавим еще кое-что в код table. Поместите это ниже закрытия </thead>, но перед завершением </table>:

```
<tbody>
  <tr>
    <td colspan="4">No libraries to display</td>
  </tr>
</tbody>
```

Если ваш код стал беспорядочным, воспользуйтесь функцией переформатирования кода PyCharm, чтобы навести порядок. Обычно это *Ctrl + Alt + L* или *Cmd + Opt + L* на Mac.

Теперь у нас есть базовая веб-страница. Прежде чем сделать ее динамической, давайте создадим конфигурацию запуска приложения, чтобы мы могли просмотреть нашу ручную работу.

Запуск приложения Flask

Когда мы создавали наш проект, PyCharm создал для нас конфигурацию запуска Flask. Давайте посмотрим на это, чтобы понять, как PyCharm будет запускать приложение. Кликните раскрывающийся список **Run configuration** и выберите **Edit Configurations...**, как показано на рис. 8.9.

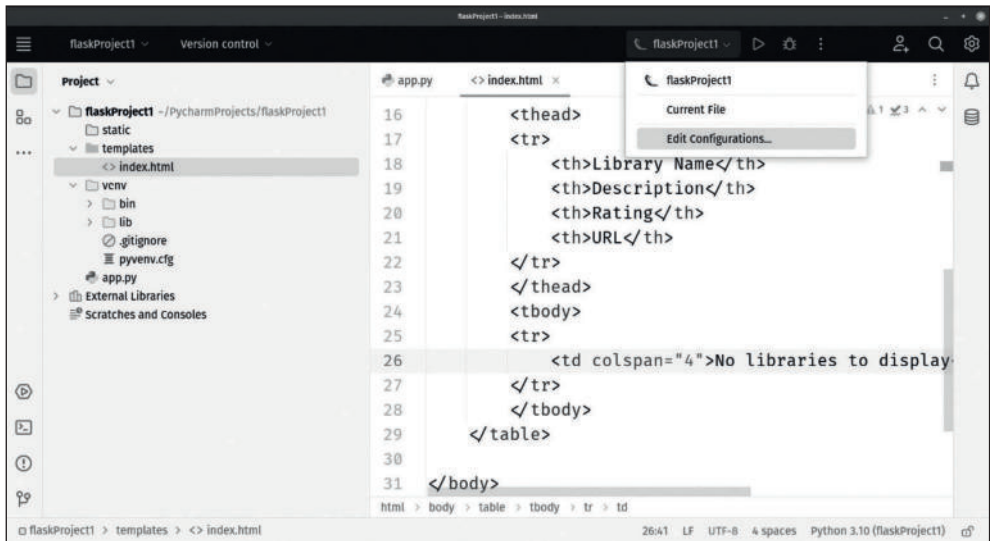


Рис. 8.9. Редактирование конфигураций запуска, чтобы мы могли видеть, из чего они состоят

Настройки выглядят так, как показано на рис. 8.10.

Теперь большинство этих настроек вам знакомы, поэтому отмечены лишь некоторые.

Есть несколько способов запустить приложение Flask. Самый простой – просто запустить файл `app.py`, который является опцией **Script path** в (1). Поскольку PyCharm сгенерировал строку `dunder-main`, запускающую приложение, это работает нормально. Вспомните код внизу `app.py`:

```
if __name__ == '__main__':
    app.run()
```

Вы также можете запустить его, используя имя модуля, но давайте пока остановимся на том, что сгенерировал PyCharm.

В позиции (2) переменная среды `FLASK_DEBUG` получает собственный флажок. Если вы установите этот флажок, выделенный сервер разработки Flask перейдет в режим отладки, что дает несколько преимуществ. Самым большим из них является то, что сервер приложений будет перезапускаться всякий раз, когда вы меняете код. Это избавит вас от необходимости останавливать и перезапускать сервер каждый раз, когда вы вносите изменения. Вообще говоря, нужно, чтобы этот флажок был установлен. Над проверкой находится часто используемая переменная среды `FLASK_ENV`, которая также передается работающему приложению. По умолчанию это `development`. Вы можете использовать эту пере-

менную среды для включения и отключения определенных действий в вашем приложении, включая настройку уровней детализации журнала.

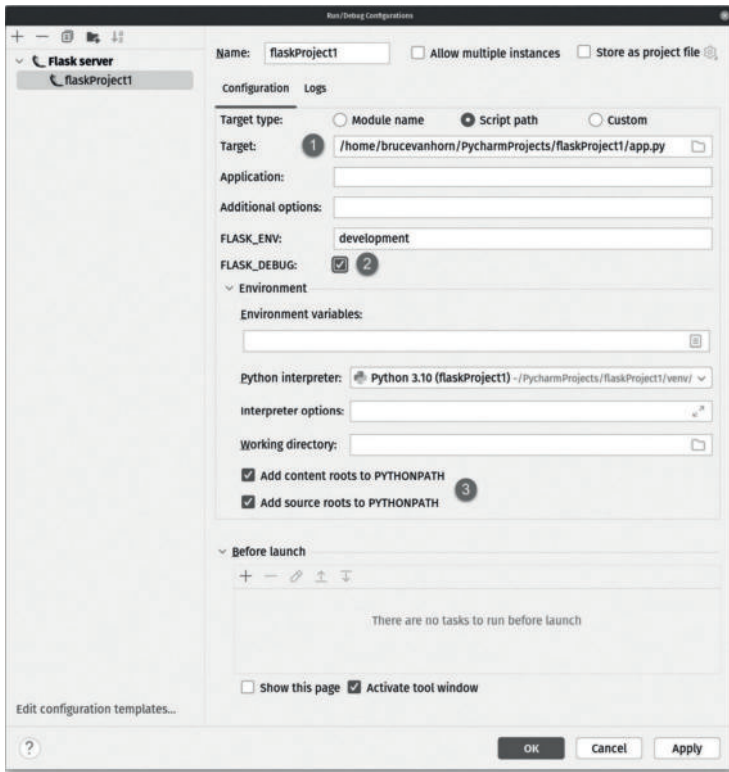


Рис. 8.10. Конфигурация запуска Flask, созданная PyCharm

Два флажка PYTHONPATH в позиции (3) добавляют папки вашего проекта в папку PYTHONPATH, что предотвращает появление ошибки о том, что Python не может найти ваше приложение. Вам нужно это проверить.

Обратите внимание, что со всеми этими флажками PyCharm дает вам возможность устанавливать общие переменные среды, которые передаются в работающее приложение. Это хорошо, потому что установка их на уровне ОС – это дополнительная работа, о которой мы часто забываем, а изменение переменных среды часто не работает так, как должно, в зависимости от вашей операционной системы. Когда вы используете переменные среды на уровне ОС, вам обычно необходимо либо перезагрузиться, либо выйти из системы и снова войти, чтобы убедиться, что новые значения активны. PyCharm вводит переменные непосредственно в работающий сервер разработки, что экономит много времени и избавляет от разочарований.

Нажмите кнопку **ОК**, чтобы закрыть диалоговое окно настройки запуска.

Прежде чем мы запустим это, нужно внести изменения в код `app.py`. Нам нужно изменить код «Hello, World», чтобы загрузить и отобразить наш шаблон. Измененный код выглядит так:

```

from flask import Flask, render_template

app = Flask(__name__)

@app.route('/', methods=["GET"])
def root():
    return render_template("index.html")

if __name__ == '__main__':
    app.run()

```

Во-первых, я добавил импорт метода `render_template` из `Flask`. Мы будем использовать его, как вы уже догадались, для рендеринга нашего HTML-шаблона `Jinja2`.

Далее я добавил код в декоратор¹ маршрута. Раньше у нас был `@app.route('/')`, определяющий маршрут `root` для нашего сайта, который будет обрабатываться нашей функцией `root`. Я добавил второй аргумент, `methods=["GET"]`. `Flask` позволяет вам заблокировать маршрут к одному или нескольким методам HTTP-запросов, которые я описал ранее в этой главе. Привязка конечных точек приложения `Flask` к определенному методу HTTP, например `GET` или `POST`, является фундаментальной практикой в веб-разработке по нескольким важным причинам.

- **Безопасность.** Различные методы HTTP имеют разные цели, и применение их несет разные последствия для безопасности. Например, запросы `GET` обычно используются для получения данных и не должны иметь никаких побочных эффектов на сервере. Напротив, запросы `POST` используются для отправки данных на сервер и могут иметь побочные эффекты, такие как создание, обновление или удаление ресурсов. Ограничив конечные точки определенными методами HTTP, вы можете предотвратить непреднамеренные или вредоносные действия. Это известно как «контроль доступа на основе методов».
- **Предсказуемость.** Привязка конечных точек к определенным методам HTTP делает ваш API или веб-приложение более предсказуемым и самодокументируемым. Другие разработчики (возможно, вы сами в будущем) будут иметь более четкое представление о том, как работать с вашим приложением. Например, если конечная точка предназначена для запросов `GET`, понятно, что она предназначена для получения данных.
- **Согласованность.** Согласованность в разработке вашего API может улучшить взаимодействие с пользователем и уменьшить путаницу. Когда пользователи или клиенты знают, что для конечной точки ожидается определенный метод HTTP, они с меньшей вероятностью будут отправлять неверные запросы.
- **Предотвращение катастроф.** Случайное неправильное использование конечной точки может привести к непредвиденным последствиям. Ограничивая разрешенные методы HTTP, вы снижаете вероятность того, что разработчики пользовательского интерфейса допустят ошибки, например попытаются удалить данные с помощью запроса `GET`.

¹ Декоратор – это функция, которая позволяет обернуть другую функцию для расширения ее функциональности без непосредственного изменения ее кода. – *Прим. ред.*

- **Поддержка фреймворка.** Flask, как и многие другие веб-фреймворки, предоставляет встроенную поддержку маршрутизации запросов на основе методов HTTP. Это упрощает реализацию контроля доступа на основе методов, поскольку вы можете определить отдельные маршруты и хендлеры для каждого метода HTTP.

Если вы присмотритесь, аргумент, который мы передаем, представляет собой массив. Можете передать один или несколько методов, позволяя одному маршруту обрабатывать один или несколько методов по-разному. Здесь мы привязываем функцию маршрута `root` к методу HTTP GET, который выдает ваш браузер при посещении сайта.

Добавим `render_template` Flask в наш импорт. Я изменил имя функции с `hello_world` на `root`, затем изменил возвращаемый результат строки 'Hello, World' на результат функции `render_template`, которая принимает имя файла шаблона. Flask знает, что `index.html` нужно искать в папке `Templates`.

Теперь мы готовы это попробовать. Убедитесь, что ваше приложение Flask выбрано в раскрывающемся списке конфигурации запуска, и нажмите зеленую кнопку **Run**. Вкладка **Run** появится в нижней части экрана PyCharm. Мой выглядит так, как показано на рис. 8.11.

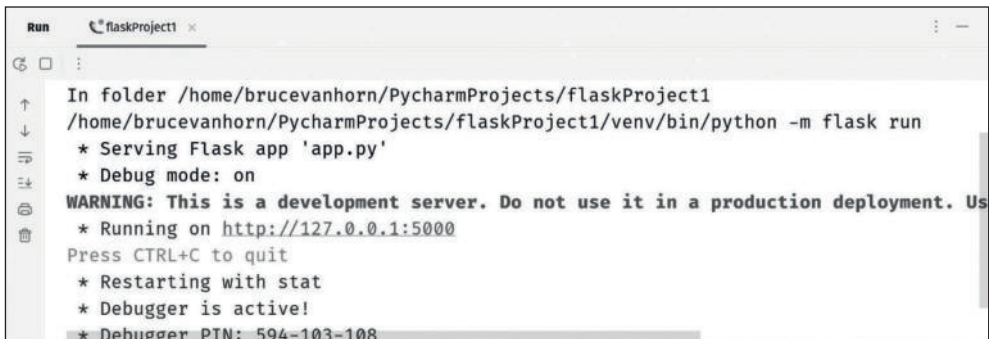


Рис. 8.11. Окно Run для нашего приложения Flask

Следует отметить несколько примечательных элементов. Первое – это большое красное предупреждающее сообщение. Я знаю, для вас это серый цвет, но вы увидите это, когда попробуете в первый раз. Мы запускаем наше приложение, используя встроенный веб-сервер разработки Flask. Пожалуйста, не используйте это в производстве. Вместо этого вам необходимо использовать сервер приложений производственного качества, такой как Green Unicorn. Это выходит за рамки разработки с помощью PyCharm, но развертывание вашего приложения с использованием встроенного сервера является настолько огромной ошибкой, что я счел необходимым указать и обосновать это предупреждение, набранное большими красными буквами.

Во-вторых, это сообщает вам, что приложение работает по адресу <http://127.0.0.1:5000>. Адрес указан в виде гиперссылки, по которой можно кликнуть, чтобы открыть браузер. Моя открыта на рис. 8.12.

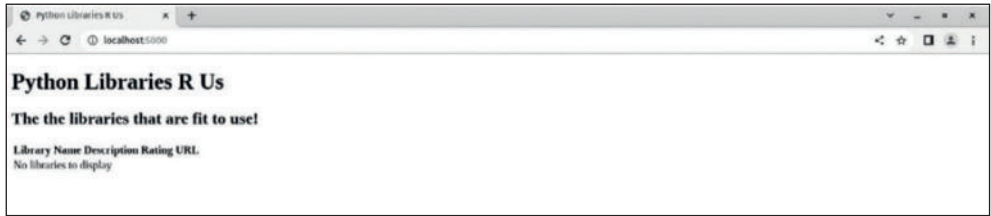


Рис. 8.12. Это сработало, но 1991 год только что позвонил по телефону, и они хотят вернуть свой веб-дизайн

Это ужасно некрасиво, не так ли? Мы позаботились о добавлении Bootstrap, и минимум, что мы можем сделать, – это использовать его.

Давайте сделаем, чтобы это выглядело немного лучше

На странице мало что происходит с точки зрения разработки, и, хотя мы не будем создавать какой-то потрясающий пользовательский интерфейс, который мог бы дать вам работу дизайнера в Apple, мы можем, по крайней мере, сделать ее немного более презентабельной, добавив несколько классов Bootstrap. Мы будем делать это в теге `body` внутри нашего шаблона Jinja2, `index.html`.

Сначала давайте настроим сетку макета Bootstrap. Это позволяет нам создавать приложения, которые корректно адаптируются к экрану любого размера: от самого маленького браузера телефона до самого большого дисплея с разрешением 8K.

Добавьте этот код внутри тега `body`. Когда вы закончите, он должен столкнуться с кодом `table`, который мы создали ранее:

```
<div class="container-fluid h-100">
  <div class="row">
    <div class="col-12">
      <h1>Python Libraries
      <span class="flipped-letter">R</span> Us</h1>
      <h2>All the libraries that are fit to use!</h2>
      <hr/>
    </div>
  </div>
  <div class="row">
    <div class="col-8">
      <h3>Here are the libraries:</h3>
```

Здесь мы добавили элемент `div`, отмеченный классом `container-fluid`. Это даст нам столь необходимый макет и отступы, чтобы наш контент не упирался в края окна браузера. Я также установил это значение так, чтобы оно занимало всю доступную высоту в окне браузера, ожидая, что в нашей таблице будет много полезных библиотек. Я сделал это с классом `h-100` Bootstrap.

После этого я добавил элемент `div`, который будет действовать как строка, а затем еще один, действующий как столбец. Я установил для столбца класс `col-12`, что в Bootstrap означает, что он должен занимать всю ширину окна браузера с соответствующими полями и отступами, определенными классом `container-fluid` в родительском теге `div`, который мы добавили ранее.

Следующие три тега – это просто дополнения к контенту – некоторая маркировка, которая объясняет, что видит пользователь, с помощью тегов H1 и H2, за которыми следует горизонтальная линия¹. Я добавил класс CSS, чтобы перевернуть букву R, чтобы она напоминала популярный, но в настоящее время обанкротившийся магазин игрушек в США. Ушла в небытие еще одна икона из моего детства². Имейте в виду, что мы еще не создали этот класс CSS. Мы сделаем это всего за минуту.

Давайте добавим в таблицу класс Bootstrap `.table`. Измените код `table`, добавив `class="table"`:

```
<h5>Here are the libraries:</h5>
  <table class="table">
    <thead>
      <tr>
        <th>Library Name</th>
        <th>Description</th>
        <th>Rating</th>
        <th>URL</th>
      </tr>
```

Это позволит исправить расстояние внутри и вокруг стола, чтобы избежать тесноты.

Чтобы закрыть все это, нам нужно добавить все необходимые закрывающие теги после закрытия нашей таблицы. Это всего лишь три вложенных замыкания `div`:

```
</div>
</div>
</div>
```

Первый закрывает столбец, второй – строку, а третий – контейнер. Последний шаг – добавить немного CSS.

Добавляем немного CSS

Кликните правой кнопкой мыши папку `static` и создайте новый файл CSS с именем `index.css`. Добавьте в файл следующий код:

```
body {
  margin: 92px;
  height: 100%;
}

.flipped-letter {
  display: inline-block;
  transform: scaleX(-1);
}
```

¹ Горизонтальная линия (англ. *horizontal rule*) – это графический элемент, который часто используется в HTML-документах для разделения разных разделов документа. Горизонтальная линия создается с помощью HTML тега `<hr>`. – *Прим. ред.*

² «Toys Я Us». – *Прим. ред.*

```
.gold-star {
  color: gold;
}
```

В этом CSS-файле мы добавляем дополнительное поле вокруг страницы и устанавливаем высоту на 100 %, чтобы не получить коротенькую страницу, которая полностью зависит от количества контента в нашей таблице.

Я добавил класс `flipped-letter` в `index.css`, чтобы перевернуть букву R в названии нашего воображаемого сайта. Мы собираемся отображать рейтинг для каждой библиотеки. Я подумал, что вместо скучного старого номера мы добавим туда несколько звезд. Не просто звезды – золотые звезды! Итак, для этого есть класс, который я назвал `.gold-star`. Если вы новичок в CSS, ведущая точка имеет значение. Остальные, отмеченные `gold-star`, помечаются как пользовательский класс. В CSS класс не имеет ничего общего с объектно ориентированной разработкой, поэтому, если вы изучили Java или другой язык, основанный на классах, слово *класс* не связано с концепциями этих языков. Через минуту вы увидите, что, когда класс будет использоваться в HTML, точка будет отсутствовать. Это не ошибка.

Теперь, когда у нас есть файл CSS в папке `static`, нужно сослаться на него в файле HTML. Поскольку он находится в папке `static`, которая имеет особое значение в приложении Flask, наш код немного отличается от простой ссылки CSS, подобной тем, которые мы видели ранее с простым HTML. Добавьте эту строку кода в тег `head`:

```
<link href="{{ url_for('static', filename='index.css') }}"
      rel="stylesheet"
      type="text/css">
```

Обратите внимание на выражение Jinja2, которое определяет расположение папки `static`.

Я только что упомянул, что хочу использовать звезды для отображения своих рейтингов. Я действительно не хочу использовать графику для этого. Вместо этого я бы предпочел использовать шрифт, в частности **Font Awesome**. Font Awesome – это, по сути, гигантский веб-шрифт, который вместо буквенных символов содержит сотни графических значков, полезных для создания современного пользовательского интерфейса и разработки веб-сайтов. Для получения дополнительной информации о Font Awesome посетите <https://www.fontawesome.com>.

Вместо того чтобы включать Font Awesome в наш проект (что, конечно, возможно), я собираюсь дать ссылку на его версию, размещенную в **сети поставки контента (CDN)**. CDN являются желательным способом размещения контента, поскольку они предназначены для обслуживания статического контента с большой скоростью. Они делают это не только за счет обычной оптимизации серверов, но и за счет стратегического позиционирования серверов по всему миру. Когда ваша страница загружает статический контент из CDN, запрос этого контента направляется на ближайший сервер. Ваши пользователи в Индии получают доступ к серверу CDN в Индии, а пользователи в Канзасе (центр США) будут получать контент с сервера, расположенного гораздо ближе.

Font Awesome размещает ссылки на CDN на своем веб-сайте. Я собираюсь использовать ссылку, скопированную с веб-сайта Font Awesome, и добавить ее в тег head нашей страницы. Поскольку это присходит из CDN, а не из статической папки, мне не нужна магия вуду Jinja2, чтобы решить эту проблему:

```
<link rel="stylesheet"
      href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/5.15.3/
      css/all.min.css">
```

Все должно выглядеть намного лучше! Я не буду показывать скриншот, пока мы не добавим динамический контент. Не хочу испортить большой сюрприз! Не забывайте, что вы можете проверить окончательный код в папке репозитория главы, клонированной с GitHub.

Делаем страницу динамичной

Мы создали приложение Flask и заработали шаблон. Добавим динамичности странице с помощью Flask и Jinja2. Мы собираемся смоделировать базу данных, используя список dicts с полями, соответствующими полям, которые мы поместили в нашу таблицу.

Переключитесь на файл app.py и найдите эту строку, которая должна находиться рядом со строкой номер 3:

```
app = Flask(__name__)
```

Эта строка, сгенерированная PyCharm при создании проекта, создает экземпляр Flask и присваивает этот экземпляр переменной с именем app. Мы начнем добавлять новый код ниже этой строки. Начнем с создания глобальной переменной для хранения наших данных:

```
library_data = list()
library_data.append({"python_library": "Flask",
                    "description": "An unopinionated web framework",
                    "rating": 5,
                    "url": "https://pypi.org/project/Flask"})
library_data.append({"python_library": "Jinja2",
                    "description": "Templating library",
                    "rating": 3,
                    "url": "https://pypi.org/project/Jinja2"})
```

В общем, следует избегать глобальных переменных, особенно в больших программах. В данном случае мы используем глобальную переменную для имитации подключения к базе данных, что обычно является исключением из правил.

Здесь мы добавляем несколько записей в нашу фейковую базу данных. Если вы поклонник Jinja2, пожалуйста, не надо меня ненавидеть. Я поставил ему 3 балла по содержанию просто для того, чтобы увидеть некоторую визуальную разницу между двумя образцами записей.

Есть еще одно изменение. Добавьте второй параметр к вызову render_template:

```
return render_template("index.html", library_data=library_data)
```

Метод `render_template` – это **вариативная функция**. Вы можете передать в него столько параметров, сколько захотите. Jinja2 сможет отображать переданные данные в шаблоне. Здесь мы просто добавляем одну переменную данных.

Мы закончили с `app.py`! Окончательный код выглядит так:

```
from flask import Flask, render_template

app = Flask(__name__)
library_data = list()

library_data.append({"python_library": "Flask",
                    "description": "An unopinionated web framework",
                    "rating": 5,
                    "url": "https://pypi.org/project/Flask"})
library_data.append({"python_library": "Jinja2",
                    "description": "Templating library",
                    "rating": 3,
                    "url": "https://pypi.org/project/Jinja2"})

@app.route('/', methods=['GET'])
def root(): # put application's code here
    return render_template("index.html", library_data=
        library_data)

if __name__ == '__main__':
    app.run()
```

Теперь, когда мы передаем некоторые данные в шаблон, нужно вернуться и изменить шаблон `index.html` для отображения данных. Вам нужно изменить содержимое тега `tbody` на следующее:

```
<tbody>
    {% if library_data|length > 0 %}
```

Маркеры `{%}` и `{{}}` в шаблоне отмечают места, где реализуется логика и контент. Здесь мы проверяем длину массива. Если она больше нуля, мы отображаем строки таблицы, используя содержимое массива. Далее есть `else`, что будет отображать то, что у нас есть сейчас, а именно одну строку, сообщающую об отсутствии данных:

```
    {% for data in library_data %}
        <tr>
            <td>{{ data.python_library }} </td>
            <td>{{ data.description }}</td>
            <td>
                {% for _ in range(data.rating) %}
                    <i class="fas fa-star gold-star"></i>
                {% endfor %}
            </td>
            <td><a href="{{ data.url }}"
                target="_blank">View on pypi.org</a></td>
        </tr>
    {% endfor %}
```

В предыдущем коде мы просматриваем список `library_data` и генерируем строку таблицы (`tr`). Затем строка таблицы снабжается столбцами. Заполнители `{{ }}` обозначают, где должно быть размещено содержимое словаря `dict` текущей итерации списка. В первом столбце показано содержимое `data.python_library`. Второй показывает описание.

В третьем добавим небольшой сюрприз! Мы добавляем блок кода, который циклически использует диапазон для создания звездочек в этом столбце. Если рейтинг равен 3, цикл выполняется 3 раза, и мы получаем 3 звезды!

Для URL-адреса я использовал значение `data.url` в качестве атрибута `href` гиперссылки.

Остается только оператор `else`, о котором я вам говорил:

```
{% else %}
    <tr>
        <td colspan="4">No libraries to display</td>
    </tr>
{% endif %}
</tbody>
```

Подводя итог, если мы передадим массив с нулевыми элементами, Jinja2 отобразит строку, которая у нас была раньше, где указано, что данных нет. Если в массиве есть данные, Jinja2 перебирает массив и генерирует строку таблицы для каждой строки данных.

Запустите проект и направьте ваш браузер на `http://localhost:5000`. Вы увидите визуализированную таблицу с двумя записями, как показано на рис. 8.13.

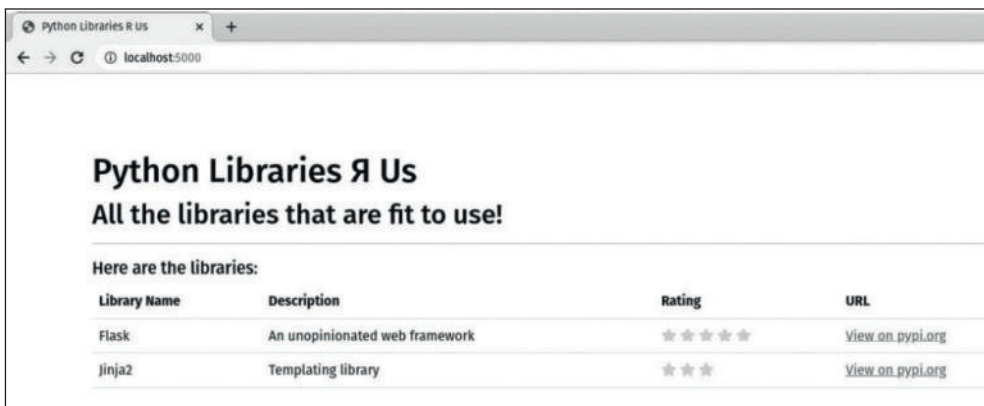


Рис. 8.13. Большое открытие! Наша страница теперь динамична

Давайте остановимся на минутку и подведем итоги того, как PyCharm нам помог.

Улучшения редактора для работы с Flask и Jinja2

Вы получили большую помощь во время этого упражнения. Если вы мне не верите, попробуйте повторить это упражнение с помощью Vim или Notepad! К настоящему времени вы уже привыкли к тому, как много PyCharm снимает

с вас с точки зрения облегчения когнитивной нагрузки и физической обработки большого количества текста.

Возможно, вы даже не заметили следующее:

- функции редактирования HTML, такие как подсветка синтаксиса, автоматическое закрытие тегов и индикаторы с цветовой кодировкой для вложенных открывающих и закрывающих тегов в разметке;
- мы использовали Emmet для создания некоторых наших разметок;
- PyCharm понимает синтаксис Jinja2, и проверка не сходит с ума, когда вы начинаете повсюду расставлять фигурные скобки;
- фактически он автоматически закрывает фигурные скобки как для `{{ expressions }}`, так и для `{% code blocks %}`;
- если вы допустили какие-либо ошибки на этом пути, то могли заметить, что проверки и предложения PyCharm четко объясняют, как использовать Flask. Это не просто интроспективное автозаполнение. PyCharm окажет вам конкретную помощь, адаптированную к разработке Flask.

Давайте рассмотрим несколько особенностей, которые не столь очевидны.

- Переключитесь на `app.py`, поместите курсор внутрь имени файла шаблона (`index.html`), затем нажмите `Ctrl / Cmd + B`. Это активирует ярлык **Go to declaration**, который приведет вас непосредственно к файлу шаблона. Это также работает и наоборот. Если вы поместите курсор на переменную `library_data` в шаблоне и нажмете `Ctrl / Cmd + B`, вы перейдете к вызову шаблона рендеринга (визуализации), который передает эту переменную в Jinja2.
- Вас не должно удивлять, что отладчик работает в `app.py`. Он также работает в шаблоне Jinja2 внутри блоков кода. Поместите точку останова в цикл `for` шаблона Jinja2, `index.html`. Запустите приложение с помощью `debug`, и отладчик остановится в цикле. Вы можете использовать те же возможности пошагового выполнения, что и для всего остального, что вы отлаживаете. Можете проверять переменные шаблона так же, как и переменные в обычном скрипте Python. Учитывая, что Jinja2, по сути, является метаязыком, полностью отличным от Python и специализирующимся на том, что он может делать, это весьма примечательно.
- Если вы допустили какие-либо ошибки, у Werkzeug есть отличная страница ошибок со ссылками на трассировку стека, которая включает трассировку вашего шаблона. PyCharm собирает эту информацию и отображает ее вместе с гиперссылками в окне **Run**. Гиперссылки ведут к коду в PyCharm, а не просто отображают его, как это делает Werkzeug, в окне браузера.
- Вы увидите тот же уровень поддержки популярных плагинов Flask, что и для Jinja2. Flask спроектирован так, чтобы его можно было расширять. Существует множество плагинов, которые облегчат вашу жизнь во многих областях веб-разработки: от разработки REST API до ORM баз данных, обработки сеансов и аутентификации. Вы найдете PyCharm, он поможет вам в разработке любого скрипта разработки Flask, в котором вы находитесь.

Краткое содержание

Я уже несколько раз отмечал, что эта глава представляет собой весьма ограниченное руководство по Flask. Мы лишь слегка коснулись того, что может Flask,

но рассмотрели все способы, которыми PyCharm может помочь вам в разработке Flask, и это одна из немногих IDE, которая обеспечивает тот уровень помощи и инструментов, которые мы видели, когда разрабатывали этот простой проект.

Во-первых, Flask предоставляет простой способ создания структуры проекта и стартового кода для проекта Flask. Как и любой другой проект, PyCharm настраивает вашу виртуальную среду и предоставляет вам стартовый код. Когда мы используем шаблон Flask в PyCharm Professional, PyCharm также устанавливает для вас зависимости вашего проекта и настраивает специальную конфигурацию запуска для вашего проекта.

Как только вы приступите к редактированию своего проекта, вы обнаружите, что все функции, которые мы рассмотрели в предыдущих главах, собраны воедино. Функции, связанные с HTML, CSS и JavaScript, работают не только с обычными HTML-проектами, но и с языком шаблонов Jinja2, родным для Flask. Мы получаем специальные проверки Flask, подсказки по коду и документацию не только для Flask, но и для расширенной экосистемы Flask.

Мы даже обнаружили, что можем отлаживать шаблоны Jinja2, как если бы они на самом деле были кодом Python! Объедините это с некоторыми очень приятными улучшениями навигации, которые помогут вам переключаться между логикой представления и внутренней логикой, и вы, как разработчик Flask, получите непревзойденное сочетание мощности и функциональности.

Flask не единственная игра в городе. За последние несколько лет стали популярны новые модели разработки: в частности, **одностраничные приложения (SPA)** в сочетании с чистыми API-интерфейсами RESTful на серверной части. Следующая глава посвящена быстрому и современному подходу к созданию RESTful API с использованием платформы FastAPI.

Как вы увидите, FastAPI чем-то похож на Flask, но с некоторыми важными отличиями. Flask использует рабочую модель для обслуживания контента или данных, в то время как FastAPI немного больше похож на NodeJS, который использует модель асинхронного программирования. FastAPI имеет тенденцию фокусироваться исключительно на создании RESTful API, и в нем отсутствуют утилиты шаблонов, имеющиеся в Flask.

ДАЛЬНЕЙШЕЕ ЧТЕНИЕ

Следующие ресурсы будут полезны:

- Gaspar, D., & Stouffer, J. (2018). *Mastering Flask Web Development: Build Enterprise-grade, Scalable Python Web Applications*. Packt Publishing Ltd.;
- Van Horn II, B. (2019). *Building RESTful APIs with Flask*. LinkedIn Learning: <https://www.linkedin.com/learning/building-restful-apis-with-flask/restful-apis-with-python-3-and-flask-4>;
- Van Horn II, B. (2021). *MongoDB for Python Developers*. MadDevSkilz.com. <https://www.maddevskilz.com/courses/mongodb-for-python-developers>. Примечание: несмотря на название, это краткое руководство по созданию приложения Flask, использующего MongoDB. Оно очень похоже на мой курс на LinkedIn и новее.

Создание RESTful API с помощью FastAPI

В последней главе мы узнали о фреймворке Flask. Flask представляет Python в среде традиционных сред веб-разработки, предназначенных для генерации контента на сервере и отправки его обратно в браузер. Именно так мы разрабатывали веб-приложения на протяжении десятилетий. 2010-е годы принесли смену парадигмы, но это не произошло в одночасье.

В 2004 году Джесси Джеймсом Гарреттом в статье под названием «Аjax: новый подход к веб-приложениям» был придуман термин **AJAX**, аббревиатура от **Asynchronous JavaScript** и **XML**. Эта статья помогла популяризировать концепцию и методы асинхронных веб-приложений. К 2005 году все основные браузеры поддерживали новый вызов веб-API **XMLHttpRequest (XHR)**. Эта функция позволяла разработчику запрашивать чистые данные вместо сгенерированной HTML-страницы с данными, интегрированными с разметкой.

Рост **одностраничных приложений (SPA)** в 2010-х годах был тесно связан с развитием таких фреймворков JavaScript, как AngularJS (теперь Angular), React и Vue.js. Это предоставило разработчикам инструменты и возможности для создания динамических интерактивных веб-приложений, отличных от традиционных приложений, которые одновременно запрашивают HTML и данные. SPA загружают всю свою разметку, CSS и JavaScript в одном запросе. После этого приложение использует XHR для запроса данных, и в ответ на полученные данные и действия пользователя разработчик использует JavaScript для отображения и скрытия различных элементов в пользовательском интерфейсе, а не перерисовывает всю страницу каждый раз, когда возникает запрос на изменение данных или взаимодействие с пользователем.

AngularJS, выпущенный Google в 2010 году, сыграл значительную роль в популяризации концепции SPA. Он представил декларативный подход к созданию веб-приложений, позволяющий разработчикам создавать насыщенные и отзывчивые пользовательские интерфейсы без необходимости полной перезагрузки страницы. AngularJS обеспечил прочную основу для создания SPA,

а его успех вдохновил на разработку других фреймворков JavaScript, которые еще больше усовершенствовали и улучшили опыт разработки SPA.

React, разработанный Facebook и выпущенный в 2013 году, также способствовал популярности SPA. React представил архитектуру на основе компонентов, которая упростила управление состоянием и компонентами пользовательского интерфейса приложения. Алгоритм сравнения виртуальной **DOM (объектной модели документа)** React и эффективный механизм рендеринга сделали его хорошо подходящим для создания быстрых и масштабируемых SPA.

Vue.js, созданный Эваном Ю и выпущенный в 2014 году, завоевал популярность как легкий и доступный фреймворк для создания SPA. Он предлагал щадящее обучение и предоставлял гибкий и интуитивно понятный способ создания пользовательских интерфейсов.

В целом сочетание этих фреймворков JavaScript, а также достижений в области браузерных технологий и API привело к росту SPA в начале 2010-х годов. Сегодня одностраничные приложения продолжают обеспечивать более плавный и отзывчивый пользовательский интерфейс за счет динамического обновления контента на одной веб-странице, устранения необходимости полной перезагрузки страницы и создания веб-приложений, более похожих на настоящие приложения.

Мы, конечно, могли бы создать серверную часть SPA с помощью Flask, но это, может быть, не лучший выбор. Это особенно верно, если вы ожидаете, что база пользователей вашего приложения будет большой. Flask критикуют за то, что он работает несколько медленно при большом объеме запросов к серверу. К счастью, в игре есть и другие игроки, и в этой главе мы сосредоточимся на FastAPI.

Самое интересное в FastAPI – то, как он обрабатывает веб-запросы. Большинство продуктов, включая Flask, используют модель воркеров (worker model)¹. Группа воркеров отвечает за обслуживание нескольких входящих запросов к отдельным процессам, управляемым операционной системой. FastAPI использует модель **асинхронного программирования**. В области асинхронного программирования функции вызываются, но немедленный возврат результатов не гарантируется. Давайте рассмотрим следующий иллюстративный код Python:

```
async def add_two(a: int, b: int) -> int:
    return a + b
```

При вызове `add_two` в контексте синхронного программирования поток выполнения останавливается до тех пор, пока функция не завершится и не будет получен результат. Однако при использовании асинхронного программирования механика отличается. Асинхронный вызов не останавливает вызывающий поток. Вместо этого вызывающий поток продолжает выполнение после получения сопрограммы, связанной с событием. Проще говоря, сопрограмма означает, что определенная задача будет выполнена позже, после некоторого запускающего события. В нашем скрипте мы обязуемся предоставить сумму `a + b` после выполнения этого вычисления. Важно понимать, что это может произойти

¹ Worker («рабочий») – это процесс Python, обычно работающий в фоновом режиме и существующий исключительно как рабочая лошадка для выполнения длительных или блокирующих задач, которые вы не хотите выполнять внутри веб-процессов. – *Прим. ред.*

не мгновенно, но во время вычислений другие операции остаются разблокированными. Естественно, сложение целых чисел, скорее всего, завершится быстро.

Рассмотрим альтернативный скрипт, в котором делается запрос к сетевому ресурсу, например вызов веб-страницы. Скорость ответа больше не зависит исключительно от скорости выполнения процессора. Скрипт включает в себя такие факторы, как производительность компьютера, задержка сети и различные другие элементы, которые могут привести к задержкам от секунд до даже минут. Это особенно актуально для тех, кто не проживает в городских районах с хорошим транспортным сообщением и высокоскоростным доступом к интернету. Поскольку асинхронное программное обеспечение не блокируется, оно может последовательно обслуживать множество запросов, поскольку ни один из них не ожидает завершения предыдущего вызова функции или, в данном случае, веб-запроса. Вы можете думать об этом как о ресторане с одним официантом. Вы отправляете свой заказ последовательно с другими посетителями, но повар возвращает вашу еду только после того, как она будет приготовлена. При таком наборе ожиданий повар может готовить столько блюд одновременно, сколько позволяет его плита, и существует неявное обещание, что, как только ваша еда будет готова, официант принесет ее вам. Тот, кто заказывает стакан сока, может получить свой заказ немедленно, тогда как приготовление десерта «Запеченная Аляска» займет немного больше времени.

В синхронном ресторане, напротив, будет работать, скажем, восемь поваров. Одновременно может быть получено восемь заказов, но каждый повар полностью посвящает себя обработке этого заказа до тех пор, пока он не будет выполнен. Как оказалось, пропускная способность, по крайней мере для компьютеров, часто выше при использовании асинхронной модели для типичных рабочих нагрузок веб-запросов.

В Python 3.4 появился новый модуль под названием `asyncio`, который привнес в наш любимый язык функции асинхронного программирования. Три года спустя мы нашли первый коммит на GitHub библиотечного проекта под названием **Starlette**. Starlette – это асинхронный веб-фреймворк для создания высокопроизводительных приложений на Python. Он обеспечивает основные функции для эффективной обработки HTTP-запросов и ответов. Starlette известен своей простотой, скоростью и поддержкой современных функций Python, что делает его идеальной основой для создания веб-приложений. Однако, как и Werkzeug, который питает Flask, Starlette должна была стать лишь основой. В этой главе рассматривается **FastAPI**, платформа, построенная на основе Starlette, которая представляет собой полную реализацию веб-инфраструктуры, специализирующейся на разработке **интерфейсов прикладного программирования RESTful (REST API)**. Там, где Flask использовал модель воркеров, FastAPI использует асинхронную модель. Flask был разработан для создания традиционной генерации контента на основе шаблонов, а FastAPI использует SPA для обработки обслуживаемых данных в форме **JavaScript Object Notation (JSON)**.

Я понимаю, что мы только что ввели много жаргонных терминов, что может быть новым для разработчиков Python, которые обычно не являются веб-разработчиками. Мы объясним нашу терминологию в контексте, когда будем рассматривать создание простого проекта FastAPI с использованием PyCharm.

К концу этой главы вы сможете следующее:

- объяснить разницу между традиционной системой генерации контента на основе шаблонов, такой как Flask, и системой, которая обслуживает данные непосредственно;
- описать природу HTTP (и HTTPS) без сохранения состояния, а также то, как **передача репрезентативного состояния (REST)** используется для устранения отсутствия состояния в HTTP;
- создать проект FastAPI, используя встроенный шаблон, предоставленный в PyCharm Professional;
- выполнить тесты вашего проекта FastAPI с помощью HTTP REST-клиента PyCharm;
- создать интерфейсное приложение React в отдельном, но подключенном (прикрепленном) проекте в PyCharm, что позволит вам разрабатывать полнофункциональное приложение без смешивания внешнего кода JavaScript с внутренним кодом Python;
- управлять конфигурациями с несколькими запусками и отлаживать весь конвейер запросов и ответов в отладчике PyCharm.

Имейте в виду, что эта глава не является полным руководством по FastAPI или React. Основная цель этой книги – обучить вас PyCharm в контексте создания приложений. Наше описание FastAPI может быть частичным, тогда как наше описание PyCharm как инструмента для создания приложений с помощью FastAPI будет очень полным.

ТЕХНИЧЕСКИЕ ТРЕБОВАНИЯ

Чтобы пройти эту главу, как и остальную часть книги, вам понадобится следующее:

- установленный и работающий интерпретатор Python. Я буду использовать последнюю версию <https://python.org>;
- установленные копии `pip` и `virtualenv`. Вы получаете их автоматически при установке Python в Windows, а в macOS они включены в систему. Если вы используете Linux, вам необходимо отдельно установить менеджеры пакетов, такие как `pip`, и инструменты виртуальной среды, такие как `virtualenv`. В наших примерах будут использоваться `pip` и `virtualenv`;
- установленная и рабочая копия PyCharm. Установка была описана в главе 2;
- Пример исходного кода этой книги взят с GitHub. Мы рассмотрели клонирование кода в главе 2. Соответствующий код этой главы вы найдете на <https://github.com/PacktPublishing/Hands-On-Application-Development-with-PyCharm---Second-Edition/tree/main/chapter-09>.

В ЖЕСТОКОМ МИРЕ СТЕЙТЛЕСС НЕТ REST¹

Я слышал, что компьютеры – самые тупые существа на планете. Они делают только то, что им говорят, и воспринимают ваши инструкции предельно

¹ Стейтлесс (Stateless protocol) – «протокол без сохранения состояния», протокол передачи данных, относящий каждый запрос к независимой транзакции, которая не связана с предыдущим запросом, то есть общение с сервером состоит из независимых пар запрос–ответ. Игра слов автора. В оригинале это можно прочесть как «В жестоком мире стейтлесса нет покоя». – *Прим. ред.*

буквально. Вот что делает программирование компьютера таким трудным. Вы должны выбрать правильный синтаксис и точно и лаконично структурировать свои идеи, потому что любая двусмысленность приведет к ошибке.

Есть только один способ усложнить себе жизнь – это строить нашу работу и карьеру на системе с вниманием плодовой мушки¹. Я говорю, конечно, о веб-серверах. Термин «*веб-сервер*» может означать две разные вещи: он может относиться к аппаратному или программному обеспечению. Аппаратное обеспечение – это любая компьютерная система, на которой установлено программное обеспечение. Я видел, как люди собирали оборудование для веб-серверов, которое помещалось в спичечный коробок, и видел, как оборудование для веб-серверов заполняло обширные системы, монтируемые в стойку, в центрах обработки данных со специальным охлаждением. По правде говоря, по крайней мере для нас, аппаратное обеспечение – самая скучная часть.

Программное обеспечение веб-сервера немного более интересно тем из нас, тех, кто пишет код. Программное обеспечение веб-серверов, такое как **Apache**, **Nginx** (произносится как «engine-ex»), **LightHTTPD** и др., разработано как простая, но надежная реализация общей спецификации протокола HTTP. Эти характеристики согласованы на международном уровне **Консорциумом Всемирной паутины (W3C)**. Сам протокол очень прост. Мы уже упоминали кое-что об этом в предыдущих главах, например следующее:

- **модель клиент–сервер:** HTTP следует модели клиент–сервер, где клиент (обычно веб-браузер) отправляет запрос на сервер, а сервер отвечает данными, которые были запрошены;
- **парадигма запрос–ответ:** HTTP-запросы выполняются клиентами для получения или отправки данных. Запрос обычно включает метод (например, GET, POST, PUT или DELETE), **универсальный идентификатор ресурса (URI)**, который идентифицирует ресурс, к которому осуществляется доступ, и заголовки, предоставляющие дополнительную информацию. Сервер отвечает кодом состояния, указывающим на успех или неудачу запроса, и включает запрошенные данные в тело ответа. Если вы это пропустили, полное обсуждение парадигмы запрос–ответ (с иллюстрациями) приведено в главе 8;
- **текстовый протокол:** HTTP – это текстовый протокол, что означает, что как запросы, так и ответные сообщения удобочитаемы. Сообщения имеют определенный формат, называемый форматом сообщений HTTP, который состоит из заголовков и необязательного тела сообщения;
- **безопасный вариант:** HTTP может быть дополнен функциями шифрования и безопасности с использованием **HTTPS (HTTP Secure)**. HTTPS добавляет уровень шифрования **Transport Layer Security (TLS)** или **Secure Sockets Layer (SSL)** для защиты конфиденциальности и целостности данных, передаваемых между клиентом и сервером.

В ходе нашего непосредственного обсуждения я хочу указать на важный момент, который мы до сих пор умалчивали: HTTP является протоколом без сохранения состояния. Это означает, что каждый запрос клиента обрабатывается независимо, без каких-либо сведений о предыдущих запросах. Сервер

¹ Это означает невозможность сосредоточиться даже на короткое время. – *Прим. ред.*

не сохраняет никакой информации о предыдущих взаимодействиях клиента. В США есть известная фраза про Лас-Вегас, в которой говорится: «Все, что происходит в Вегасе, остается в Вегасе». Аналогично то, что происходит внутри механизма запроса–ответа, остается в жизненном цикле запроса–ответа. Как только ответ получен и подтвержден веб-браузером, в журнале не остается ничего, кроме записи о том, что запрос вообще имел место.

По сути, мы имеем здесь тупую машину, выполняющую запросы, о которых у нее нет памяти. Это может несколько усложнить нашу работу. Мы хотим, чтобы наши пользователи постоянно взаимодействовали с нашим приложением, но сервер не поможет нам сделать это без некоторого рода уговоров.

За прошедшие годы был изобретен ряд механизмов, помогающих управлять состоянием в нашем приложении. Если вы не понимаете, что я имею в виду под состоянием, я предпочитаю описывать его как сохранение в вашей любимой компьютерной игре.

Представьте, что вы играете в приключенческую игру. Вы играли довольно долго и нашли вход в темный замок, который находится на третьем уровне. Вы ответили на загадку привратника и получили поражающий меч. Вы накопили 32 768 золотых монет, и ваш персонаж полностью здоров. Потом мама просит вынести мусор. Как мы все знаем, это «мамин код», означающий «бросить играть в видеоигры и заняться чем-нибудь продуктивным». Она никогда не поймет вас, не так ли? Естественно, вам захочется продолжить с того места, где вы остановились, поэтому вы сохраняете игру. Вы можете выключить компьютер и снова включить его завтра, после чего вы сможете загрузить игру, и она будет вести себя так, как будто вы и не прерывались. Это связано с тем, что ваша сохраненная игра представляет собой сохраненное состояние программы. Это снимок всех объектов, переменных и данных, используемых вашей игрой на момент ее сохранения. Это и есть «состояние». Это то, чего веб-приложениям не хватает, по крайней мере, на уровне протокола.

Состоянием можно управлять с помощью нескольких методов. Серверные решения, такие как Flask, позволяют реализовывать закрепленные сеансы, в которых HTTP-запросы генерируют токен. Детали запроса могут храниться между сеансами со ссылкой на токен. Это обрабатывается не вашим веб-сервером, а вашим сервером приложений, который использует веб-сервер для коммуникационной части: механизма запроса–ответа. Сеансы обычно нежелательны, поскольку в большинстве случаев они не масштабируются. Если у вас большой веб-трафик, обычной практикой является балансировка нагрузки трафика между несколькими серверами. Если ваш сеанс сохраняется на сервере А при первом запросе, а последующий запрос поступает на сервер В, вашего сеанса на нем не будет. Естественно, мы и для этого придумали решения, но давайте не будем углубляться в подробности.

Вы можете использовать файлы cookie, которые сохраняют данные на стороне клиента. К сожалению, файлы cookie имеют плохую репутацию, поскольку на протяжении многих лет ими активно злоупотребляли. Основные браузеры предоставляют файлы cookie, которые вы сохраняете, и ваши потенциально параноидальные пользователи могут отказаться принимать ваш запрос на сохранение файлов cookie.

Лучший ответ – хранить состояние вашего приложения в самом приложении. Идея здесь называется репрезентативной передачей состояния, или сокращенно REST. В скрипте RESTful мы сохраняем состояние программы в памяти на клиенте. Мы передаем любую часть состояния или даже (но не обычно) все состояние, используя механизм запроса–ответа. По сути, программа отправляет части состояния, необходимые серверу для выполнения запроса. Сервер делает все, что должен, а затем отправляет обратно измененное состояние в ответе. Если вы знакомы с шаблонами проектирования программного обеспечения, REST напоминает мне шаблон команды: запрос инкапсулирует все, что необходимо серверу для выполнения запроса.

Теперь, когда все готово, давайте вспомним, что SPA теперь отвечают за поддержание своего состояния, что вся разметка HTML, CSS и JavaScript загружается в браузер и что все последующие запросы содержат только данные о состоянии, которые отправляются на сервер, где изменяются и возвращаются.

Формат передачи данных может быть любой текстовой формой. Чаще всего это JSON. В былые времена мы использовали XML, но оставили это, потому что обработка XML в браузере происходит смехотворно медленно. JSON работает быстрее, поскольку браузер уже изначально понимает JavaScript, поэтому синтаксический анализ текста не требуется. На случай, если это ваше первое родео, давайте сравним два формата. Во-первых, вот немного XML:

```
<person>
  <firstName>Bruce</firstName>
  <lastName>Van Horn</lastName>
  <dateOfBirth trueDate="Heck No">12/19/1987</dateOfBirth>
</person>
```

XML – это разметка на основе тегов, такая же, как HTML, но вы можете определять свои собственные теги, используя схему XML. Этот формат изначально использовался в браузерах с вызовом API **XMLHttpRequest (XHR)**. X на самом деле означает *XML*. XHR все еще используется, но уже почти никто (несмотря на команды API Microsoft Azure) не использует XML. Вместо этого, по уже упомянутым причинам производительности, я даю вам то же самое в JSON:

```
{
  "person": {
    "firstName": "Bruce",
    "lastName": "Van Horn",
    "dateOfBirth": {
      "date": "12/19/1987",
      "trueDate": false
    }
  }
}
```

Представлены одни и те же данные, как и структура этих данных. Как разработчик Python, вы, несомненно, распознали это как dict. Вместо тегов с содержимым и атрибутами у нас есть пары ключ–значение, хранящиеся в фигурных скобках. Правило JSON заключается в том, что ключи и текстовые значения заключаются в двойные кавычки. Будьте осторожны здесь. И JavaScript, и Python

считают одинарные и двойные кавычки взаимозаменяемыми, а JSON – нет. Допускаются только двойные кавычки! К счастью, в стандартной библиотеке Python есть библиотека `json`, которая преобразует ваши структуры в JSON и обратно без каких-либо проблем:

```
import json

# Convert dictionary to JSON
data_dict = {"name": "John", "age": 30, "city": "New York"}
json_data = json.dumps(data_dict)
# Print the JSON data
print("JSON data:", json_data)
```

Эти первые несколько строк импортируют библиотеку `json`, а затем преобразуют ее в объект JSON с помощью метода `json.dumps`. Просто помните, что мы выгружаем строку, следовательно, и `dumps` (*s* означает *строку*). Теперь давайте преобразуем другое направление:

```
# Convert JSON to dictionary
parsed_dict = json.loads(json_data)

# Access the dictionary
print("Name:", parsed_dict["name"])
print("Age:", parsed_dict["age"])
print("City:", parsed_dict["city"])
```

Мы использовали `json.loads` для преобразования JSON обратно в `dict`. Просто запомните это как «Мы загружаем строку JSON», следовательно, `loads` (*s* означает *строку*).

Теперь вы понимаете основную механику того, что мы собираемся делать с FastAPI. Запросы будут поступать так же, как и в случае с Flask, но, как правило, вместо простых запросов GET, которые перехватываются и обрабатываются с помощью шаблонизатора Jinja2, запросы будут содержать полезные данные JSON, которые мы будем обрабатывать. Результаты обработки будут возвращены в формате JSON. Механизм запроса–ответа будет обрабатываться с использованием асинхронных функций, поэтому код будет выглядеть несколько иначе, чем в Flask.

Давайте немного запачкаем руки, чтобы вы поняли, что я имею в виду!

СОЗДАНИЕ ПРОЕКТА FASTAPI В PYCHARM PROFESSIONAL

К настоящему моменту мы создали много проектов в PyCharm Professional, и это не сильно отличается. Напомню, что этот набор функций доступен только в профессиональной версии PyCharm. Если вам нужно использовать версию Community, можете это сделать, но вы сами будете настраивать проект, поскольку у вас не будет доступа к инструментам, которые мы собираемся использовать.

Создайте новый проект в PyCharm, кликнув **File | New Project**. Затем найдите **FastAPI** в списке шаблонов. Вы можете увидеть мой на рис. 9.1:

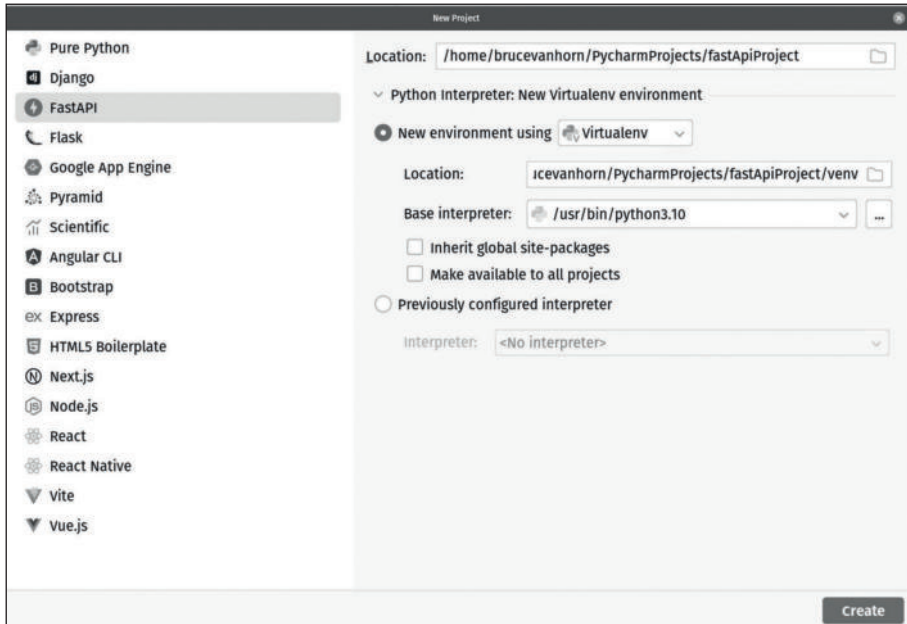


Рис. 9.1. Меню проекта PyCharm содержит шаблон для проекта FastAPI

Как и в случае с Flask, шаблон проекта FastAPI генерирует для нас некоторый стартовый код и конфигурацию запуска, как показано на рис. 9.2:

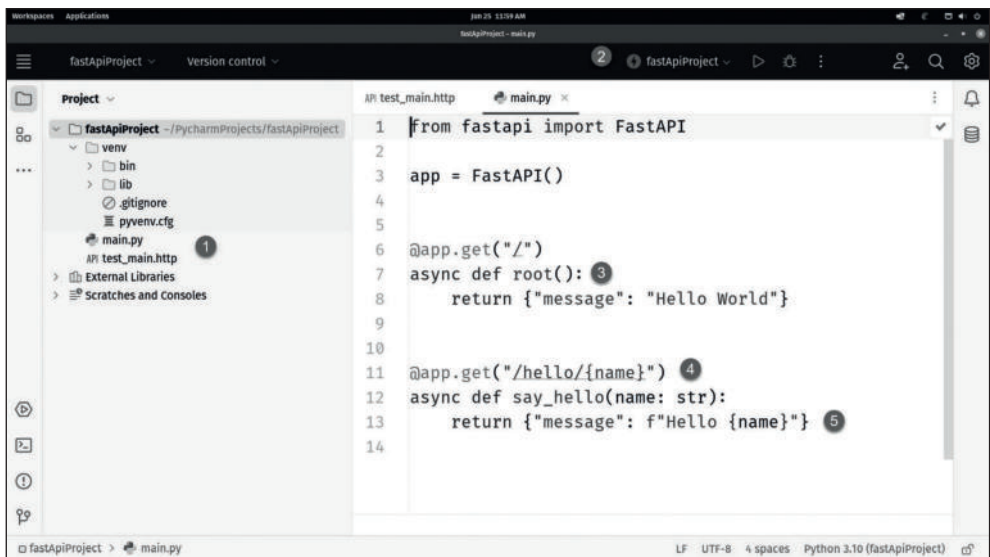


Рис. 9.2. Шаблон генерирует наш стартовый код FastAPI и конфигурацию запуска

По поводу рис. 9.2 есть о чем поговорить, поэтому я пронумеровал диаграмму для справки.

PyCharm создал виртуальную среду вместе с двумя файлами, как показано в (1): `main.py` и `test_main.http`. Мы вернемся к файлам через минуту. PyCharm сгенерировал для нас конфигурацию запуска. Вы можете это сказать, потому что это текущая выбранная конфигурация запуска в (2), где в этом меню обычно указывается **Current File**, если в PyCharm не настроена конфигурация запуска.

В позиции (3) мы видим пример конечной точки «Hello, World» для нашего проекта FastAPI. Сила этих шаблонов стартового кода заключается в психологических подсказках, которые они обеспечивают. Если вы использовали какую-либо систему для работы с веб-маршрутизацией и конечными точками, независимо от языка или фреймворка, вы можете прочитать этот код и рассказать, что происходит.

Строки 1 и 3 показывают нам типичный конструктор импортированного экземпляра библиотеки FastAPI. Строка 6 показывает нам, как FastAPI украшает конечные точки. `@app` возникает из создания экземпляра в строке 3, и здесь мы вызываем метод HTTP GET, чтобы запросы, поступающие с использованием этого метода, были получены и обработаны. Если клиент делает запрос, используя другой HTTP-verb, например PUT или POST, он получит ошибку, поскольку в настоящее время не существует кода для обработки этого HTTP-метода.

Позиция (3) показывает нам наш корневой маршрут, как указано в определении конечной точки в строке 6. Как и было обещано, без всяких каламбуров, у нас есть функция `asunc` под декоратором в строке 7, и мы видим, что возвращаем что-то похожее на dict, который также выглядит как JSON.

Позиция (4) показывает нам еще одну конечную точку GET, определенную как `/hello/{name}`. Смещение переменной `name` в фигурных скобках относится к части URL-адреса, которая может меняться, и называется параметром пути или переменной пути. Вы можете видеть, что оно дублируется в определении функции `asunc` в строке 12. Содержимое ветвей (`name`) в строке 11 должно совпадать с именем параметра функции в строке 12.

Позиция (5) в строке 13 показывает нам переменную пути, используемую в возвращаемом JSON, где строковое выражение Python `f` заполняет имя в данные. Не существует специального механизма шаблонов, кроме обычной строки `f Python`.

ЗАПУСК ПРОЕКТА FASTAPI

Естественно, мы можем запустить проект FastAPI, нажав кнопку запуска с зеленой стрелкой рядом с позицией (2) на рис. 9.2. Если вы пропустили главу о настройке и использовании конфигураций запуска и отладки, возможно, вам следует просмотреть главу 6 для получения подробной информации о том, как эта функция работает в PyCharm. Вы можете увидеть мою программу, работающую на рис. 9.3.

Это похоже на запуск в Flask, но предупреждение о сервере разработки отсутствует. Это связано с тем, что FastAPI работает в приложении под названием `uvicorn`, которое является вариантом Green Unicorn (`gunicorn`). `Uvicorn`

готов к производству, поэтому предупреждений нет. Можете разрабатывать приложение, используя то же программное обеспечение сервера приложений, которое вы будете использовать при развертывании приложения для своих клиентов. Разница между `uvicorn` и `gunicorn`, который чаще всего используется в качестве рабочего сервера для приложений Flask, заключается в том, что `uvicorn` обрабатывает модель асинхронного программирования, тогда как `gunicorn` использует традиционный процесс воркеров, как описано ранее в этой главе.

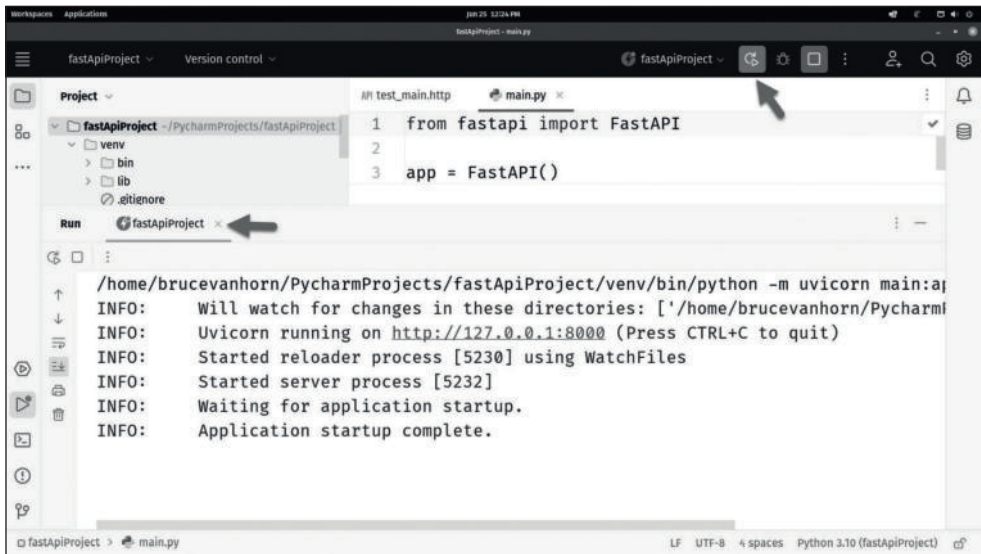


Рис. 9.3. Я нажал зеленую кнопку Run, в результате чего в окне запуска моего проекта в PyCharm появилась вкладка

Здесь мы оказались в затруднительном положении. Когда мы делали это с помощью Flask, приложение генерировало HTML-код, который мы могли просмотреть. Здесь этого нет. Если хотите, вы можете просмотреть конечную точку в браузере, как показано на рис. 9.4:



Рис. 9.4. Посещение корневого маршрута не очень увлекательно, но работает

Я также могу посетить другой URL-адрес, указав параметр пути как часть URL-адреса, как показано на рис. 9.5:

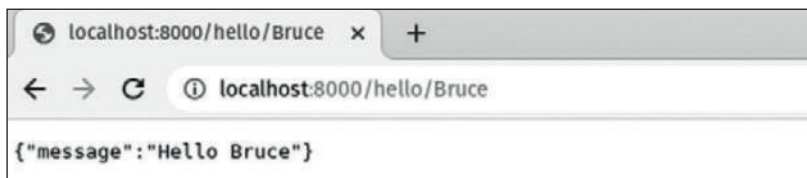


Рис. 9.5. Указание параметра пути изменяет данные в ответе

На этот раз я добавил параметр пути в конец URL-адреса, и, как мы видели в коде ранее, мы получаем обратно сгенерированные данные JSON.

Возникла проблема с использованием браузера. Браузер позволяет отправлять только HTTP-запросы GET. Существует еще немало методов HTTP, которые обычно используются в проекте REST API. Фактически четыре наиболее популярных метода, иногда называемых *verb*¹, соответствуют типичным операциям CRUD, используемым в приложениях баз данных. **CRUD** – это аббревиатура от **Create, Read, Update, Delete**. Карта глаголов показана на рис. 9.6. Ваш запрос определяется этими операциями. Вам не обязательно использовать эти методы, но рекомендуется. Я видел большие, дорогие коммерческие приложения, использующие только GET и POST.

В дополнение к стандартам для запросов ваши ответы могут быть стандартизированы с помощью лучших практик с использованием правильных кодов состояния HTTP. Они документированы в спецификациях HTTP, предоставленных W3C. Чтение спецификации – верное лекарство от бессонницы, поэтому я направлю вас на отличную страницу **Mozilla Developer Network (MDN)**, где вы найдете коды состояния. Это удобный адрес для добавления в закладки вашего браузера: <https://developer.mozilla.org/en-US/docs/Web/HTTP/Status>.

HTTP Verb/Метод	Операция CRUD	Статус HTTP для успешного ответа	Объяснение
GET	READ	200 (OK)	Используется для получения информации с сервера или базы данных. Например, для получения списка пользователей в вашем приложении
POST	CREATE	201 (Created)	Используется для создания новой записи в базе данных. Используйте это в любое время, когда создаете что-либо на сервере. Например, регистрацию нового пользователя
PUT	UPDATE	204 (No Content) or 200 (OK)	Используйте этот метод при отправке обновления уже существующего объекта, такого как запись в базе данных. Например, обновление пароля
DELETE	DELETE	204 (No Content) or 200 (OK)	Используйте это, если хотите что-то удалить

Рис. 9.6. Методы HTTP сопоставляются с операциями CRUD 1:1 и имеют стандартные коды, обозначающие успешные ответы

¹ Verb – методы статических глаголов. – Прим. ред.

Учитывая, что ваш браузер ограничен HTTP-запросами GET, и со 100%-ной вероятностью того, что мы захотим использовать хотя бы один из других методов в нашем проекте, нам нужно что-то лучшее, чем просто браузер для тестирования нашего API. У нас есть несколько вариантов.

- Существуют плагины для браузера, которые позволяют отправлять различные виды запросов. Вы можете найти их в магазине вашего любимого браузера.
- Инструменты командной строки, такие как cURL, позволяют создавать HTTP-запросы, используя любой из методов HTTP.
- Специальные инструменты тестирования API, такие как Insomnia (<https://www.insomnia.rest>) или Postman (<https://getpostman.com>), предоставляют вам графический инструмент для работы с запросами API. Эти инструменты могут быть очень раздутыми, поскольку они предназначены для гораздо большего, чем просто выполнение различных типов запросов. С учетом сказанного я использую оба из-за их широкого распространения. Хотите верить, хотите нет, но в моей команде есть разработчики, которые не используют PyCharm.
- Встроенная функция HTTP-запросов PyCharm Requests.

РАБОТА С HTTP-ЗАПРОСАМИ PYCHARM

Когда мы создали наш проект, PyCharm создал два файла. Он создал `main.py`, который мы уже рассмотрели. Он также создал файл с именем `test_main.http`. Этот файл уникален для PyCharm. Давайте рассмотрим файл, показанный на рис. 9.7:

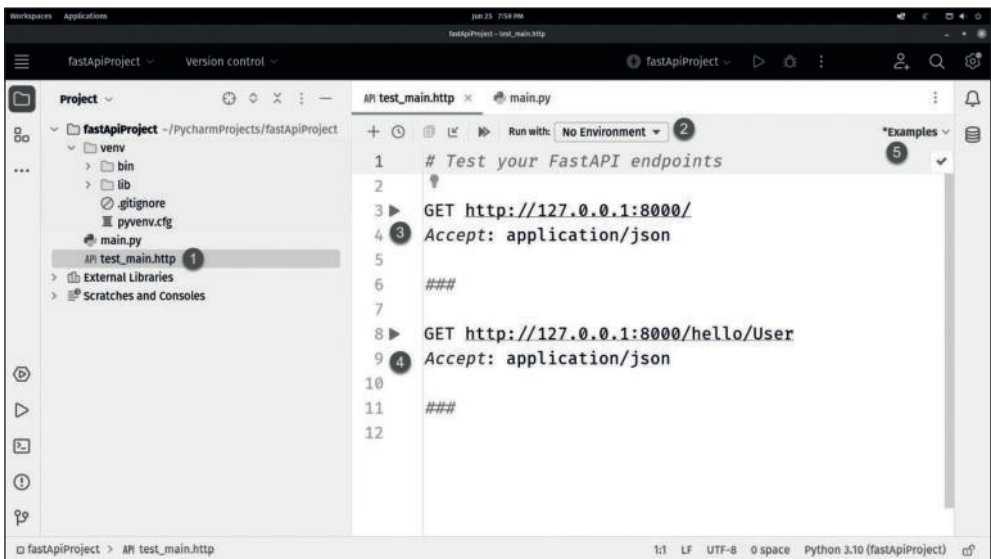


Рис. 9.7. Тестовый файл HTTP, созданный PyCharm как часть проекта FastAPI

Сам файл вы найдете прямо рядом с `main.py` (1). Мы упомянули об этом как об одном из популярных соглашений о размещении тестовых файлов – рядом с файлом, который мы тестируем. Это называется `test-main.http`, он позволяет

нам точно знать, что тестируется. Файл `main.py` будет содержать конечные точки, а файл `test_main.http` – тесты для всех конечных точек.

Этот тестовый файл `http` не является кодом, как мы видели в главе 6. Это спецификация для HTTP-запросов. Позиции (3) и (4) показывают по одному тесту на каждую конечную точку в `main.py`, что является хорошей отправной точкой. Эти тесты очень просты, и, как и модульные тесты, их можно запускать индивидуально, используя зеленые стрелки. Если вы хотите запустить все тесты в файле, можете использовать панель инструментов (2), где есть кнопка с двумя зелеными стрелками. Существует также селектор среды, к которому мы скоро вернемся. Позиция (5) показывает ссылку, дающую вам набор примеров, которые можно вставить в ваши тестовые файлы. Мы к этому тоже вернемся. Сначала давайте проведем наш тест и посмотрим, что он делает. Я нажму кнопку **Run** с двойной стрелкой, показанную верхней стрелкой на рис. 9.8:

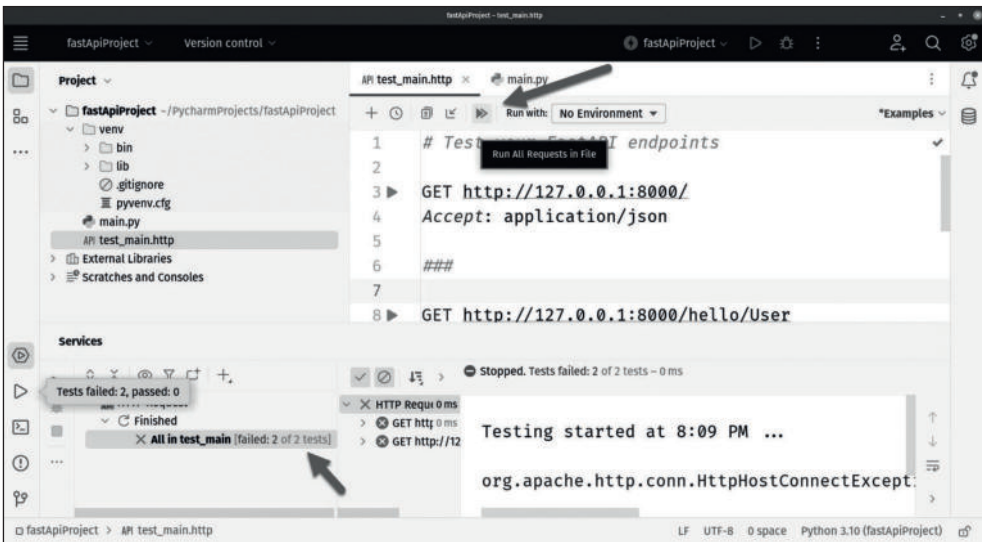


Рис. 9.8. Нажмите двойную стрелку, чтобы запустить все тесты, и вы обнаружите, что оба они провалились! О нет!

Это нехорошо. Оба теста с треском провалились. Я говорю с *треском*, потому что PyCharm не оставляет двусмысленности. Я вижу красный баннер, сообщающий, что мой тест не пройден. Рядом с нижней стрелкой я вижу красный текст, сообщающий, что два теста не пройдены. Я вижу красные крестики рядом со списком тестов. Я вижу сообщение **Stopped. Tests failed 2 of 2**. Если этого недостаточно, есть область журнала, которая также сообщит вам, что ваш код не попал в цель. Короче говоря, на этом экране больше красных отметок, чем в рукописи этой главы после первой правки редактора! Поверьте, это очень много! Что в мире может быть не так? Зачем PyCharm генерировать неудачный тестовый код, подобный приведенному в примере «Hello, World»?

В коде нет ничего плохого! В главе 6 мы узнали о модульном тестировании. Модульные тесты включают код, который проверяет тестируемый код с помощью ассертов (утверждений, assertions). HTTP-файл не содержит кода, и это не модульное тестирование, это интеграционное тестирование. Для работы

этих тестов требуется работающий сервер. Давайте попробуем это еще раз. Изучите рис. 9.9 и, если хотите, следуйте инструкциям:

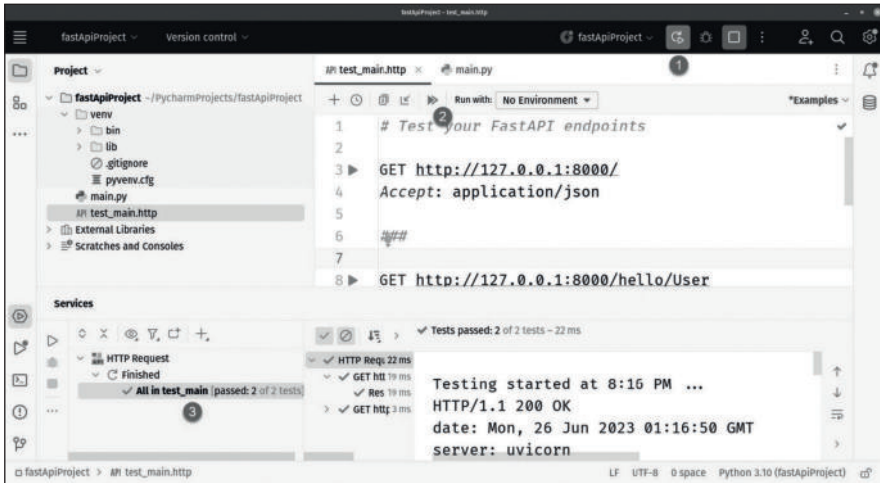


Рис. 9.9. Перед запуском HTTP-тестов необходимо запустить сервер разработки

Нажмите кнопку запуска проекта API (1). Затем – кнопку запуска тестов (2). При работающем сервере все тесты пройдут (3). Если этого не произойдет, позвоните в JetBrains и попросите вернуть вам деньги. Скорее всего, вы ничего не получите, и это нормально, поскольку ваши тесты должны пройти успешно.

Рассмотрение деталей возврата

Недостаточно знать, что мы прошли проверку, поскольку на самом деле это означает лишь то, что оба запроса были отправлены на локальный сервер разработки и оба вернулись с кодом состояния 200. В большинстве случаев вам действительно нужно иметь возможность видеть данные JSON, полученные в ответе. Давайте найдем это. Найдите выходные данные теста и прокрутите вниз, пока не увидите упоминание о файле JSON, подобное показанному на рис. 9.10:

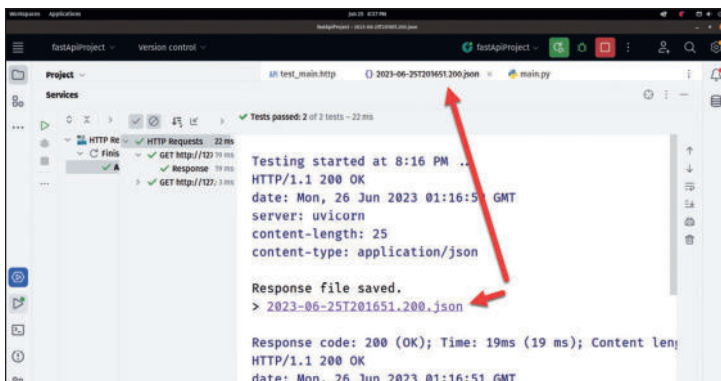


Рис. 9.10. Прокрутите журнал и найдите ссылки на файлы JSON. Нажмите на них, чтобы открыть данные в области вкладок в редакторе

вете, сделанном в ходе теста. Мы можем увидеть следующее:

- дата запроса,

гиперссылку в журнал:

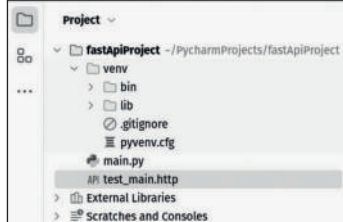


Рис. 9.11. Возвращенные данные отображаются на отдельной вкладке при нажатии гиперссылки

вкладку правой кнопкой мыши, как показано на рис. 9.12.

файловым менеджером по умолчанию в **Gnome 42**. Windows будет содержать ссылку на Explorer, а macOS должна иметь ссылку на Finder. Нажав на эту опцию, вы увидите локацию в файловом менеджере вашей ОС. Вы можете увидеть мою на рис. 9.13:

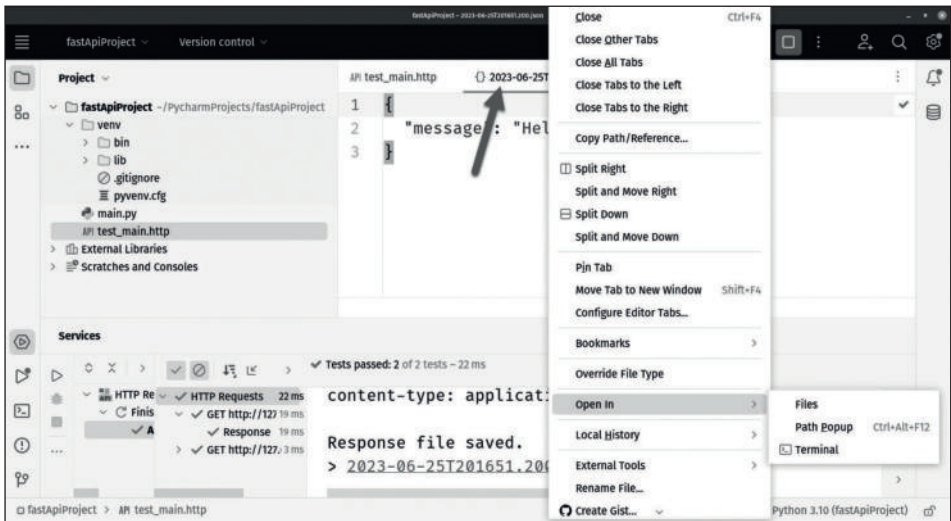


Рис. 9.12. Кликните вкладку правой кнопкой мыши и используйте меню Open In, чтобы открыть локацию JSON-файла результатов теста

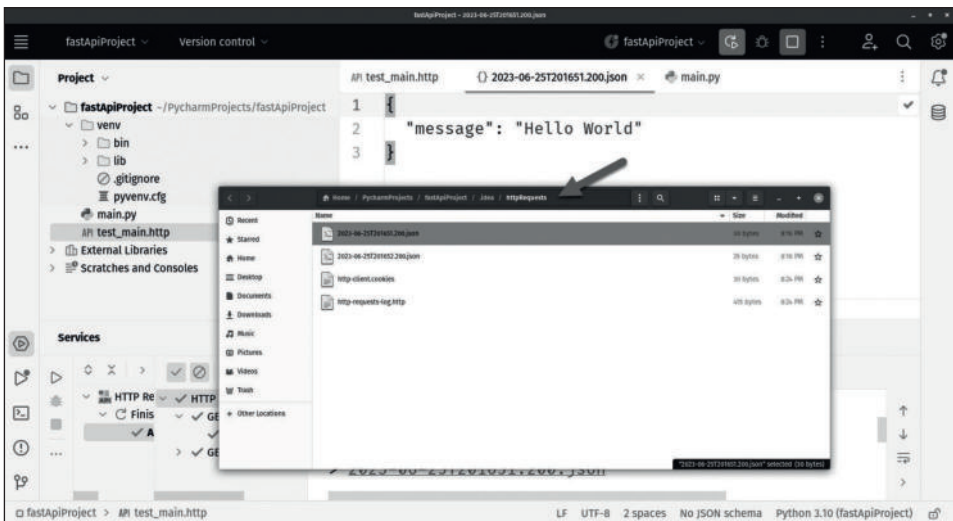


Рис. 9.13. Мой файловый менеджер показывает расположение файлов ответов HTTP, которые находятся в папке .idea проекта

Как видите, результаты теста сохраняются в папке проекта PyCharm – скрытой папке `.idea`, которую IDE создает, когда вы создаете новый проект или открываете папку в PyCharm. Помните, что любая папка, имя которой начинается с точки, скрыта на Mac или Linux, но вы можете увидеть их в Windows, потому что Билл любит все делать не так, как все. Если вы часто тестируете, как и должно быть, эти файлы могут начать накапливаться. Лично я предпочитаю исключить их из своих репозиторий.

Мы только что создали новую конфигурацию запуска

Побочным эффектом запуска всех тестов из окна тестирования является то, что PyCharm создает новую конфигурацию запуска, которую вы найдете в раскрывающемся списке конфигурации, показанном на рис. 9.14:

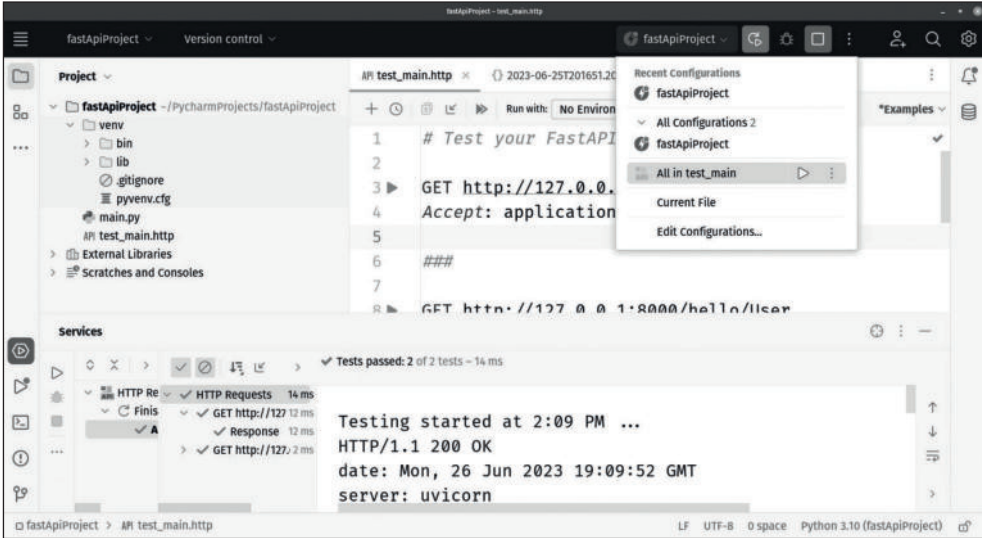


Рис. 9.14. PyCharm автоматически создал новую конфигурацию запуска после первого запуска HTTP-тестов

Если отредактировать конфигурацию, то можно увидеть более подробную информацию, как показано на рис. 9.15:

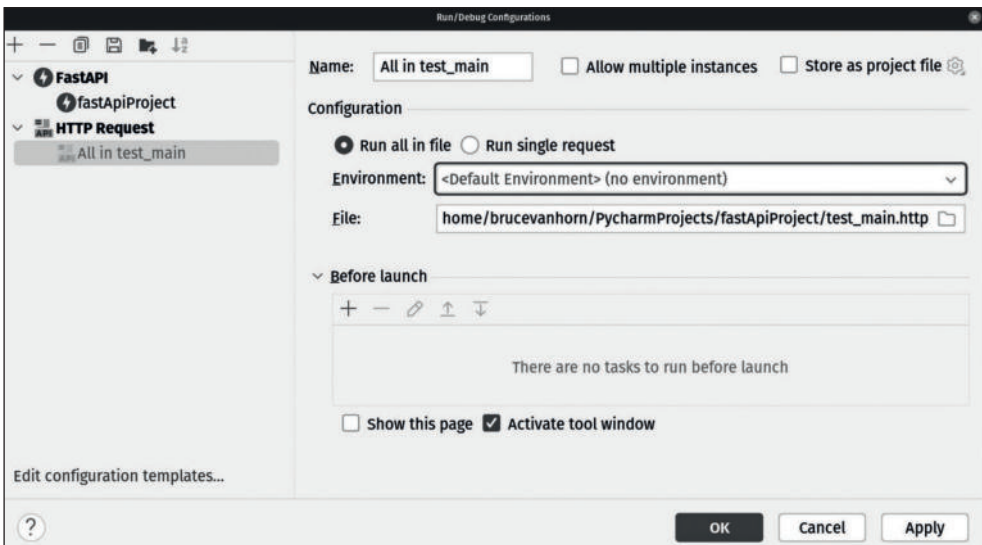


Рис. 9.15. Параметры выполнения тестов HTTP-запроса можно найти в редакторе конфигурации запуска

Как видите, вы можете выполнить отдельные запросы или все запросы в файле. Выполнение одиночных запросов удобно, если вы сосредоточены на коде для одной конечной точки и вам нужно запускать его снова и снова. Если в тесте было много конечных точек, выполнение тестов может занять некоторое время.

Использование Before в конфигурациях запуска

Мы видели, что при каждом запуске наших HTTP-тестов создается новый файл JSON, содержащий ответ. Это может довольно быстро привести к заполнению папки сотнями файлов. Поскольку PyCharm предоставляет нам конфигурацию запуска, у нас есть возможность настроить автоматизацию удаления старых файлов. В этом разделе мы рассмотрим некоторые возможности автоматизации, доступные в настройках конфигурации запуска PyCharm.

Для этого вам понадобится скрипт для удаления файлов JSON по вашему усмотрению. Возможно, вы хотите удалить все, что старше недели, или просто сохранить последние 25 запусков. Эта часть зависит от вас.

Чтобы автоматизировать это, вам необходимо создать скрипт оболочки для вашей операционной системы. Если вы не гурู скриптов оболочки, ничего страшного, потому что современный искусственный интеллект может прийти вам на помощь. На рис. 9.16 показано, как я прошу ChatGPT создать скрипт оболочки для оболочки Bash, который сохраняет только последние 25 файлов JSON, созданных в ходе наших тестовых запусков.

Я понимаю, что вы не можете увидеть на рисунке весь скрипт для Bash, поэтому я обязательно включу его в исходный код проекта для этой главы.

Предупреждение – никогда не запускайте сгенерированные скрипты, которые вы не понимаете!

Никогда не следует слепо запускать скрипт, сгенерированный ChatGPT или кем-либо (или кем-либо) еще, включая мой или даже особенно мой, не понимая до конца, как он работает! ChatGPT, вероятно, не даст вам того же результата, что и я, поэтому будьте осторожны при запуске любого скрипта, который он вам дает. Будьте особенно осторожны, если скрипт включает в себя что-то вроде переключателей -Force, как в скрипте PowerShell на рисунке. Если вы не знаете, что делает скрипт, ни в коем случае не запускайте его на своем компьютере!

Чтобы использовать сгенерированный код, можете просто добавить новый файл в свой проект, как и любой другой. Я собираюсь кликнуть правой кнопкой мыши свой проект в окне Project и создать новый файл с именем `delete-oldhttp-test-results.sh`. Естественно, если вы работаете в Windows, вам понадобится PowerShell, который обычно имеет расширение `.ps1`, поэтому это будет `delete-old-http-test-results.ps1`. Скопируйте скрипт, созданный ИИ, как только вы полностью поймете последствия запуска указанного скрипта, и сохраните файл.

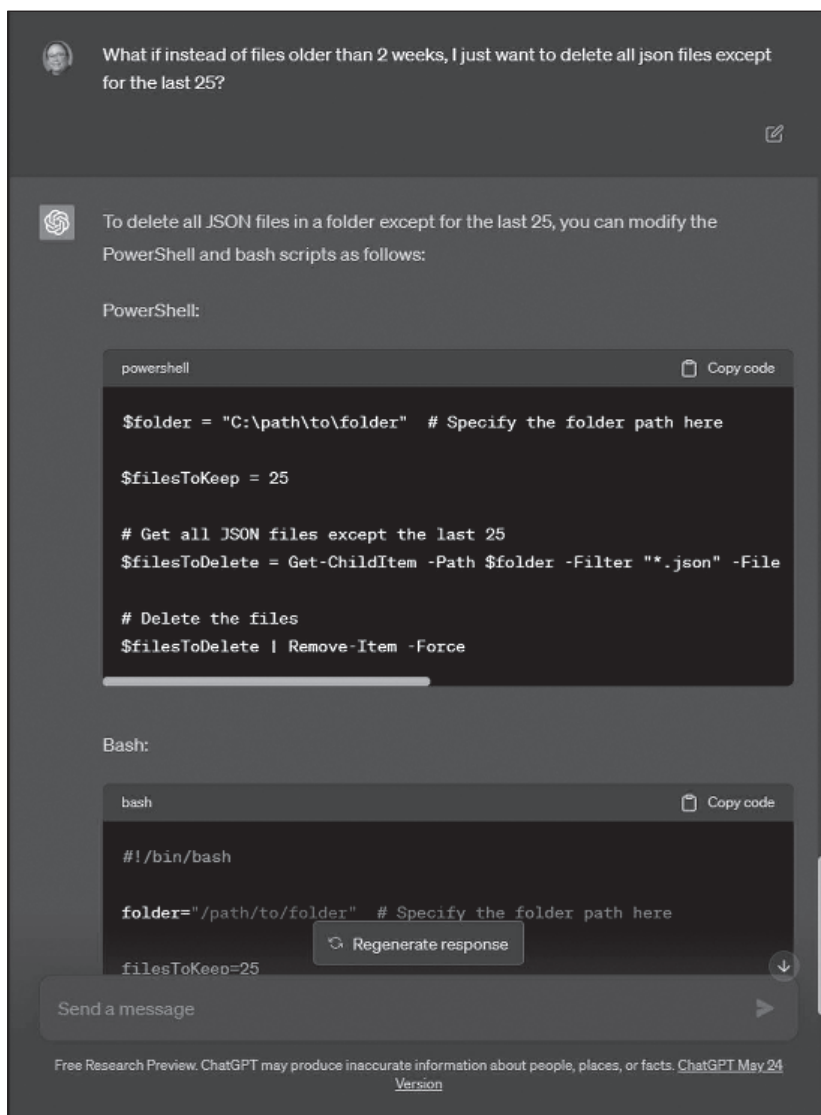


Рис. 9.16. Я спросил ChatGPT о нескольких различных способах удаления старых файлов, и он сгенерировал для меня скрипт

Убедитесь, что у скрипта есть разрешения на запуск

Убедитесь, что созданный вами файл скрипта имеет разрешение на запуск на вашем компьютере. Большинство выпусков Windows строго ограничивают запуск любого скрипта PowerShell, и ваша возможность делать это может быть ограничена политикой безопасности даже вашего работодателя.

На Mac и Linux вам может потребоваться запустить это в терминале, прежде чем PyCharm выполнит скрипт:

```
chmod +x delete-old-http-test-results.sh
```

Если вы не можете запустить скрипт вручную, он, вероятно, не будет работать и в PyCharm.

Далее нам нужно создать конфигурацию запуска, которая выполняет скрипт. Кликните раскрывающийся список конфигурации запуска и выберите **Edit Configurations**. Если вы не помните, как это сделать, просмотрите главу 3. Добавьте новую конфигурацию запуска, используя шаблон Shell Script. Вы можете увидеть мою на рис. 9.17:

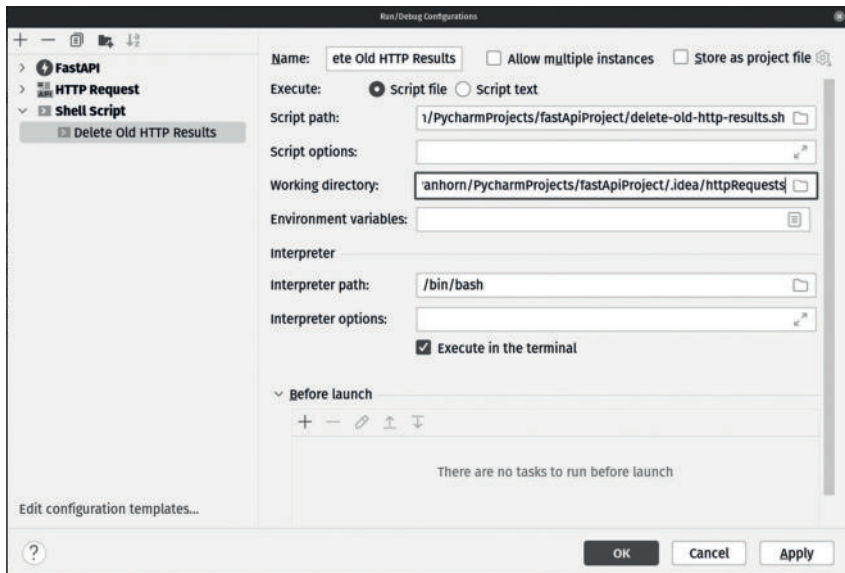


Рис. 9.17. Создайте конфигурацию запуска для только что созданного скрипта оболочки

Можете перейти к скрипту оболочки, используя кнопку папки в текстовом поле **Script path**. Обязательно установите **Working directory** для скрипта; в противном случае вы можете удалить файлы в своем проекте! Ранее мы видели, что эти файлы находятся в папке `httpRequests` внутри папки `.idea` проекта. Кликните значок папки в текстовом поле **Working directory**, чтобы просмотреть эту папку. Если вы не в Windows, это будет скрытая папка, поэтому обязательно включите просмотр скрытых папок в диалоге выбора, как я это сделал на рис. 9.18:

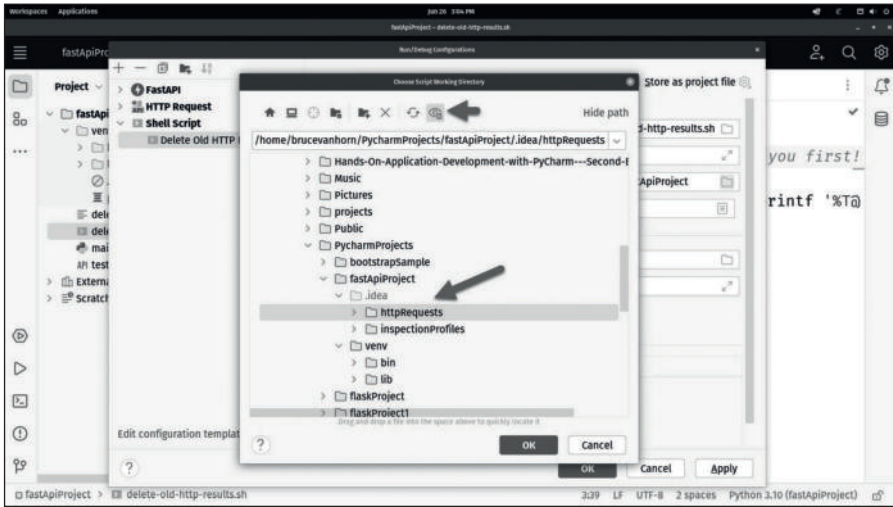


Рис. 9.18. Чтобы найти папку `httpRequests` внутри папки `.idea`, вам нужно включить скрытые папки, кликнув глазик (о) на верхней панели инструментов

Теперь у вас есть конфигурация запуска, которая выполняет ваш скрипт `delete`. Следует это проверить. Запустите тесты еще 26 раз и убедитесь, что в папке `httpRequests` есть только последние 25 результатов. Я покажу вам свои результаты на следующих 25 полностраничных цветных скриншотах. Это я шучу. Иногда забавно напугать своих редакторов.

Если это работает и вы довольны, то есть еще кое-что, что вы можете сделать, чтобы стало еще более круто. Вы можете связать конфигурацию запуска скрипта удаления с конфигурацией тестового запуска. Отредактируйте конфигурацию запуска тестов и добавьте условие **Before build**. Нажмите кнопку **+**, чтобы добавить новое условие. Нажмите **Run another run configuration**. Нажмите **Delete Old HTTP Results run configuration**. Вы должны увидеть что-то вроде рис. 9.19, на котором показано, что скрипт удаления будет запускаться перед каждым запуском нового теста.

Ради книги я изменил свой коэффициент удержания на последние пять файлов JSON и опробовал его. После каждого запуска я могу заглянуть в свой файловый браузер, показанный на рис. 9.20, и проверить, работает ли он.

В вашей папке будет не более 25 результатов или столько, сколько вы указали в своем скрипте. Однако не забывайте, что сначала запускается скрипт удаления, и PyCharm сгенерирует один новый файл JSON для каждой тестируемой конечной точки. Если я настрою свой скрипт удаления на сохранение пяти файлов JSON и запущу сгенерированный тестовый скрипт, после запуска у меня будет семь файлов, поскольку тестируются две конечные точки.

Используя этот метод, вы можете реализовать множество скриптов автоматизации для своего кода. В Python, как и во многих других языках, обычно нет скрипта сборки, поэтому приятно знать, что такой уровень автоматизации доступен в самой IDE. Помимо запуска другой конфигурации запуска, было несколько вариантов. Рекомендую изучить все возможности!

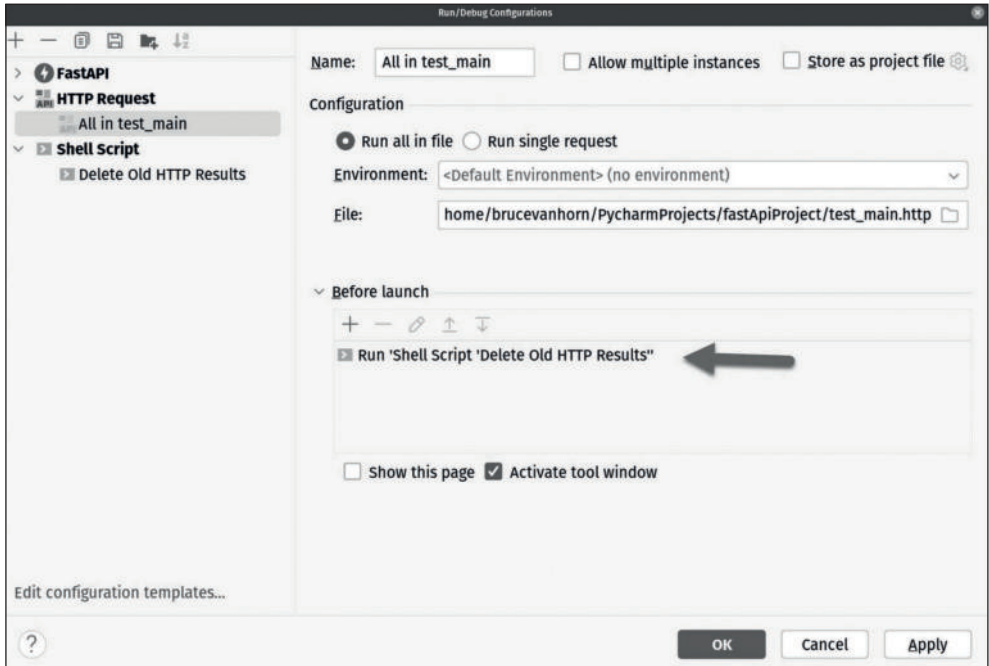


Рис. 9.19. Конфигурация запуска удаления теперь будет запускаться перед каждым запуском моих тестов

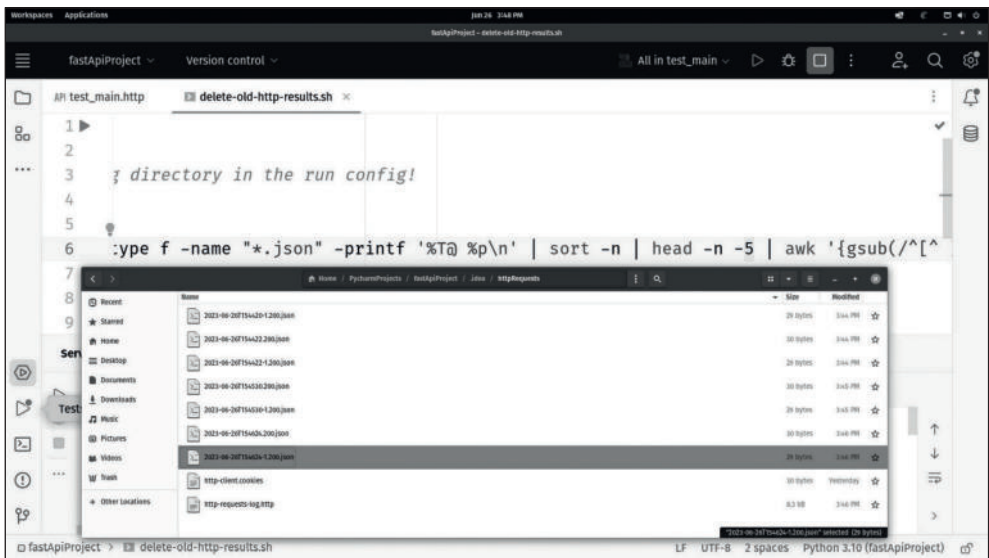


Рис. 9.20. После моего запуска имеется семь файлов JSON, поскольку скрипт удаления перед запуском уменьшает количество файлов до пяти, и тест дает результаты для двух конечных точек

Работа со средами HTTP-запросов Request

Большинство веб-проектов начинают свою жизнь на компьютере разработчика. Когда вы тестируете, то запускаете свое приложение локально на своем ноутбуке или ПК, и все ваши тестовые запросы обычно передаются на localhost, который представляет собой адрес обратной связи, назначенный каждому компьютеру с сетевой картой. Проект начинается с этого, но, если приложение будет иметь хоть какой-то успех, оно на этом не остановится.

Передовая практика требует, чтобы приложения, предназначенные для публикации, использовали своего рода **среду непрерывной интеграции (CI)**, в которой можно запускать автоматические тесты. JetBrains выпускает CI-продукт под названием Team City. Я использовал Team City много лет и могу подтвердить, что это отличная система CI, которую легко настроить и она бесплатна для небольших команд разработчиков. В наши дни существует даже облачная версия, поэтому вам не нужно настраивать собственные серверы, если это вас не интересует. Однако у Team City есть тот же плагин, который мы использовали для создания тестов для наших конечных точек HTTP в нашем проекте FastAPI. Это позволяет вам автоматически тестировать ваш проект каждый раз, когда кто-то фиксирует код в вашем исходном репозитории.

Как только ваш код проходит проверку в системе CI, его обычно развертывают на тестовом сервере. Это должен быть сервер, максимально похожий на производственный, которым вы можете управлять. Некоторые называют это промежуточным сервером, другие – **пользовательским приемочным тестом (UAT)**. Неважно, как вы это называете, оно представляет собой среду разработки. Разработка тоже является средой. Среда – это просто среда конфигурации, в которой вы можете запускать свой код. Для ясности: до сих пор мы упомянули четыре таких среды:

- ваш локальный компьютер (localhost),
- среду CI, которая в наши дни, вероятно, представляет собой контейнер Docker,
- UAT/промежуточную среду, в которой вы можете протестировать свое приложение перед его выпуском в производство,
- среду разработки, в которой ваши реальные клиенты используют ваше приложение.

Каждая из этих сред может иметь разные атрибуты. Например, доступ к вашему приложению FastAPI, работающему на вашем ноутбуке, можно получить по IP-адресу 127.0.0.1. По умолчанию приложение работает на порту 8000, а также по умолчанию используется протокол HTTP, а не HTTPS, поскольку немногие разработчики тратят время на настройку сертификата SSL/TLS на своих ноутбуках.

Однако когда вы тестируете свое приложение в промежуточном режиме, все эти параметры будут другими. Вероятно, у вас будет сертификат SSL, поэтому в качестве протокола вы будете использовать HTTPS. У вас обязательно будет другой IP-адрес. У вас может даже быть сервер доменных имен, разрешающий хорошее тестовое доменное имя для вашего приложения. Вероятно, вы не будете использовать порт 8000, поскольку это будет не очень похоже на производственную среду. Вместо этого вы будете использовать порт 443 или 80, и в этом случае вам вообще не обязательно указывать порт.

Разработка тоже снова будет иметь другие атрибуты.

В наших тестах API мы можем настроить набор переменных, присваиваемых имени среды, которые будут использоваться в нашем тестовом скрипте. Убедитесь, что у вас есть файл `test_main.http` или любой файл `.http`. Кликните раскрывающийся список среды, показанный на рис. 9.21:

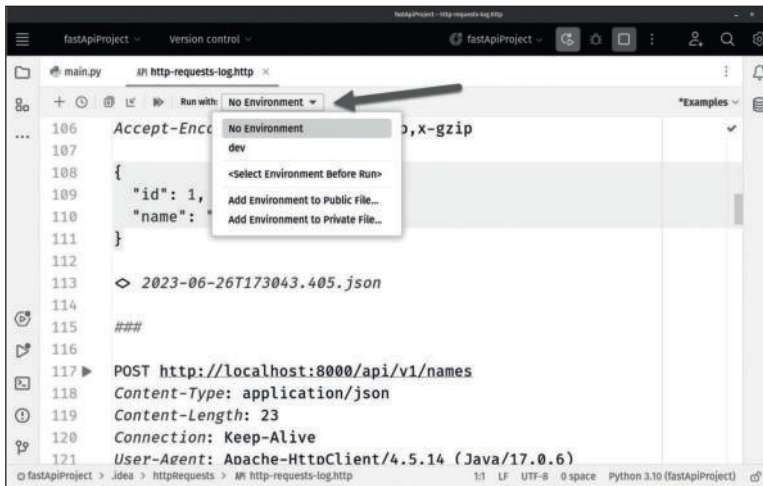


Рис. 9.21. Вы можете выбрать среду для запуска тестов, каждая из которых имеет свои собственные переменные конфигурации

Я буду честен. Я уже поработал и установил ее, но сделаем вид, что я этого не делал. Прямо сейчас ваш единственный вариант – **No Environment**. Как скучно! Проверьте два варианта внизу и нажмите **Add Environment to Public File**. Вы обнаружите, что PyCharm добавляет в ваш проект новый файл с именем `http-client.env.json`. Вы можете увидеть мой на рис. 9.22:

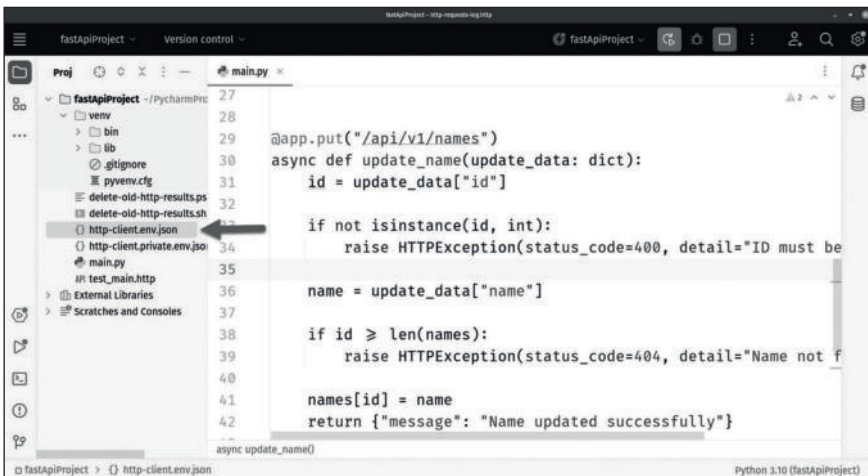


Рис. 9.22. PyCharm создал `http-client.env.json`, который позволит мне настроить различные среды для тестирования

Если бы вы выбрали опцию **Private file**, вы бы получили `http-client-private.env.json`. Назначение и разница между этими двумя файлами не описаны в документации на момент написания этой статьи, поэтому нам придется использовать воображение. Лично я помещаю среды, которыми хочу поделиться со своей командой разработчиков, в общедоступный файл. Если я хочу создать какую-то частную среду, например виртуальную машину, подсистему Windows для Linux, специальные эксперименты в Docker или, может быть, кластер Kubernetes в лаборатории, я могу использовать частный файл, который, скорее всего, поместил бы в свой файл `gitignore`. Поскольку содержимое файлов одинаковое, сосредоточусь на общедоступном файле.

Я собираюсь добавить определение для моего локального компьютера. Измените содержимое `http-client.env.json` на это:

```
{
  "dev": {
    "protocol": "http://",
    "base_url": "localhost",
    "port": 8000,
    "api_version": "v1"
  }
}
```

Теперь у нас есть среда разработки с некоторыми переменными среды. Давайте используем их в нашем файле `test_main.http`. Ваше первое определение конечной точки выглядит следующим образом:

```
GET http://127.0.0.1:8000/
```

Чтобы использовать переменные среды, замените части URL-адреса текстом в формате шаблонов усов. Формат усов предполагает помещение переменных, которые вы хотите разрешить, в двойные фигурные скобки, например `{{ ... }}`:

```
GET {{protocol}}{{base_url}}:{{port}}
```

В совокупности URL-адрес будет преобразован в исходный, т. е. `http://localhost:8000` или, если вы предпочитаете, `http://127.0.0.1:8000`. Разница в том, что теперь вы можете создавать другие среды таким же образом. Вы можете переключить среду и запустить тесты без изменений, а переменные среды решают ваши URL-адреса за вас.

Если вы просмотрите мой код в примере кода этой главы, то обнаружите, что я таким образом схитрил во всем файле.

ОПЕРАЦИИ CRUD

На данный момент единственное, что есть в нашем приложении, – это две заранее сгенерированные конечные точки. Давайте добавим еще, чтобы сделать это немного интереснее. Мы собираемся создать приложение поддельного списка, как мы это делали в главе 8.

Откройте `main.py` и давайте добавим по одной конечной точке для каждой операции CRUD и, в свою очередь, для каждого из четырех основных методов,

используемых при создании RESTful API. Мы собираемся внести некоторые радикальные изменения, поэтому я просто представлю их с начала файла.

В первой строке, где у нас есть импорт, измените его на следующее:

```
from fastapi import FastAPI, status, HTTPException

app = FastAPI ()
```

Экземпляр приложения не изменился. Однако ниже добавьте этот список удивительных людей:

```
names = ["Bruce", "Karina", "Kitty", "Phoebe"]
```

Этот список будет служить фальшивой базой данных, что сэкономит нам время на настройку серверов и бла-бла-бла, что серьезно раздует книгу и не будет способствовать нашему будущему мастерству PyCharm на уровне чемпионов. Мы собираемся оставить первые две сгенерированные конечные точки в покое, поэтому просто начните редактирование под возвратом функции `say_hello`. Наша первая конечная точка CRUD предоставит нам список имен, как будто из запроса к базе данных:

```
@app.get("/api/v1/names")
async def get_names():
    return [{"id": idx, "name": name} for idx, name in enumerate(names)]
```

Рекомендуется начинать конечные точки вашего проекта, что должны возвращать данные JSON, а не разметку, с префикса API, за которым следует обозначение версии. Поверьте мне, когда я говорю, что вам нужно это сделать. По мере развития вашего кода API можете предлагать обновленные конечные точки, которые не обязательно будут обратно совместимыми, и этот метод позволяет вам поддерживать обратную совместимость и бесперебойность вашего API с вашими старыми клиентами, одновременно вводя улучшенные функциональные возможности для новых клиентов. По мере дальнейшего развития кода можете удалить конечные точки v1 в любое время по вашему выбору:

```
@app.post("/api/v1/names", status_code=status.HTTP_201_CREATED)
async def create_name(name: dict):
    new_name = name["name"]
    names.append(new_name)
    return {"message": "Name added successfully"}
```

В предыдущей конечной точке POST рекомендуется при успешном вызове возвращать код состояния HTTP 201, указывающий, что API создал новые данные. Код FastAPI работает совсем не так, как Flask. Вместо отдельных объектов запроса и ответа все неявно. Если вы ожидаете, что JSON будет отправлен в качестве полезной нагрузки в ваш API, вам нужно указать это только с помощью типа аргумента `dict`. В этом случае я ожидаю, что данные будут опубликованы в следующем формате:

```
{"name": "Igor"}
```

Настоящее приложение будет иметь более богатую структуру. Мы сохраняем простоту. Когда этот JSON поступает в качестве полезных данных для POST, мы извлекаем имя и добавляем его в наш список. Для простоты я не выполняю никакой проверки этой конечной точки. Вы всегда должны проверять входящие данные, чтобы защитить себя от токсичных данных и атак путем внедрения. Это тема другой книги. С учетом вышесказанного я не буду полным лентяем. Я немного сделаю это в конечной точке PUT, чтобы вы могли увидеть, как это будет выглядеть. Помните, что вызов PUT – это операция UPDATE. Мы берем значение `id` и новое значение `name` и соответствующим образом изменим существующее значение. Добавьте эти строки под возвратом функции `create_name`:

```
@app.put("/api/v1/names")
async def update_name(update_data: dict):
    id = update_data["id"]
```

Теперь вернемся к коду. Давайте удостоверимся, что отправленный атрибут ID является числом, а именно целым числом:

```
if not isinstance(id, int):
    raise HTTPException(status_code=400, detail="ID must be an integer")
```

Что тут происходит? Если вы отправите атрибут ID, который не преобразуется в число, я отправлю вам обратно код состояния ошибки HTTP 400, что означает, что ваш запрос имеет неверный формат. Просто для удобства я добавлю небольшое сообщение о том, что вы сделали не так. Например, предположим, что вы используете PUT для этих данных следующим образом:

```
{ "id": "2", "name": "Igor" }
```

Все должно работать. Но предположим, что вы используете PUT для следующего:

```
{ "id": "this isn't a number", "name": "Igor" }
```

Ваш запрос не будет выполнен. Повышение расширения HTTP приведет к немедленной отправке ответа, а остальная часть этого кода не будет выполнена. Итак, продолжим. Если вы предоставили хорошие данные, мы можем двигаться дальше и получить новое имя:

```
name = update_data["name"]
```

Затем определяем, не выходит ли введенный вами числовой идентификатор за допустимые пределы. Наш список начинается с четырех имен. Если вы попытаетесь отправить обновление на `id 600`, это не сработает, если вы не добавите в список хотя бы столько же имен через конечную точку POST. В PUT, если вы указываете неверное значение `id`, обычно возвращается ошибка 404, сооб-

щающая, что вы не смогли найти этот идентификатор в своей базе данных (или в списке в нашем случае):

```
if id >= len(names):
    raise HTTPException(status_code=404, detail="Name not found")

names[id] = name
return {"message": "Name updated successfully"}
```

Это три игрока, и остался только один. Для конечной точки DELETE мне нужен только id. Отправлять полезную нагрузку JSON для чего-то такого простого было бы глупо, поэтому я просто возьму идентификатор в качестве параметра URL-адреса конечной точки. Мне все еще нужно проверить, находится ли длина в пределах длины списка names:

```
@app.delete("/api/v1/names/{id}")
async def delete_name(id: int):
    if id >= len(names):
        raise HTTPException(status_code=404, detail="Name not found")
```

Если да, то я просто вытащу его из списка и верну великолепное сообщение:

```
deleted_name = names.pop(id)
return {"message": f"Deleted name: {deleted_name}"}
```

Запустите приложение и убедитесь, что оно запускается. Внесите необходимые корректировки. Помните, что окончательный рабочий код находится в примере кода главы из репозитория, который мы проверили в главе 2.

Больше тестирования

Теперь, когда у нас есть больше разнообразных конечных точек, давайте узнаем больше об их тестировании с помощью HTTP-клиентов. Как оказалось, этот инструмент чрезвычайно богат, столь же богат, как и специализированные инструменты, такие как Insomnia или Postman, только без всего этого пользовательского интерфейса.

Откройте файл test_main.http и позвольте нам добавить тесты. Не только простые, давайте создадим несколько реальных и почувствуем, как рабочий процесс PyCharm помогает вам создавать RESTful API.

Эти функции тестирования PyCharm работают с любой платформой

Я должен отметить, что, хотя мы используем FastAPI, эти функции не уникальны и не привязаны к этой платформе. Эти инструменты будут работать так же, если вы разработали свой API в Flask или даже в какой-либо другой среде.

На данный момент у вас есть только два теста, которые были сгенерированы при создании проекта:

```
# Test your FastAPI endpoints

GET {{protocol}}{{base_url}}{{port}}
Accept: application/json

###

GET {{protocol}}{{base_url}}{{port}}/hello/User
Accept: application/json
```

Я не говорил вам конкретно модифицировать вторую конечную точку теста с помощью переменных среды, которые мы создали ранее, но я пошел дальше и сделал это, поскольку это кажется очевидным улучшением. При добавлении кода в тестовый файл стоит обратить внимание на несколько вещей.

- Запросы могут храниться в файлах с расширениями `.http` или `.rest`. До сих пор мы работали с файлом `.http`, созданным в рамках проекта. В реальной жизни вы можете увидеть любое расширение, и между ними нет никакой принципиальной разницы, кроме значков, отображаемых в IDE. Это чисто косметический эффект.
- Подсветка кода и завершение синтаксиса активны. Синтаксис этих файлов специализирован, но здесь вы можете рассчитывать на тот же уровень инструментов, что и где-либо еще.
- Для кода в этом файле также появится встроенная документация.
- Запросы должны быть разделены тремя хештегами/решетками: `###`. Если вы не добавите их, IDE сообщит вам об этом морем красных волнистых линий.
- Можете вставить команду `cURL` командной строки в этот файл, и она автоматически преобразует ее в синтаксис, используемый PyCharm.
- Поддерживается шаблонизация. Мы уже видели работу замены переменных среды. Оставайтесь с нами, скоро будет еще интереснее!

Редактор тестов имеет небольшой пользовательский интерфейс, но, честно говоря, как только вы изучите синтаксис, вы, вероятно, не будете его использовать. Позвольте мне показать – см. рис. 9.23:



Рис. 9.23. Редактор тестов имеет минимальный UI

Давайте посмотрим на цифры.

Значок + в (1) позволяет добавить еще один тест. Честно говоря, это проще сделать прямо в редакторе, но если вам нравится кликать мышью, то вперед.

Значок часов (2) на самом деле очень полезен. Откроется журнал истории запросов, который содержит последние 50 сделанных вами запросов. Он открывается как обычный файл проекта, и вы найдете его в том месте, которое мы посетили ранее, в папке `httpRequests` внутри папки проекта `.idea`. Поскольку PyCharm автоматически перезагружает файлы при их изменении, оставляя этот файл открытым, вы можете видеть все, что происходит, без необходимости просматривать отдельные файлы ответов. Ранее мы добавили автоматизацию для удаления многих из этих файлов ответов, поскольку их часто бывает очень много. Это не влияет на файл журнала, поскольку информация запроса добавляется из окна журнала, а не из самого файла ответов. Фактически в журнале также отображаются данные ответа.

Совет для профессионалов: откройте журнал в отдельном окне, чтобы вы всегда могли его видеть

Если кликнуть правой кнопкой мыши вкладку, содержащую файл журнала, и выбрать один из вариантов разделения, например **Split Right**, вкладка откроется в отдельной группе вкладок. Если у вас хороший широкий монитор 4K, у вас достаточно места, чтобы хранить журнал открытым вместе с кодом и тестовыми файлами, чтобы вы могли видеть все это вместе.

Значок документа (3) предназначен для того, чтобы порадовать знатоков командной строки. Если вы выберете запрос и нажмете эту кнопку, то сможете сгенерировать команду `cURL`, которую затем можно будет вставить в окно терминала. Значок импорта (4) предоставляет вам пользовательский интерфейс, в который можно вставить команду `cURL`, после чего она будет преобразована в формат запроса, используемый PyCharm. Это не так уж и полезно, поскольку вы можете вставить команду `cURL` непосредственно в сам редактор, и преобразование произойдет автоматически.

Мы уже видели кнопку **Run All Tests** (5), а также раскрывающийся список выбора среды. Вероятно, самый полезный элемент на этой панели инструментов, помимо журнала истории, – это ссылка на примеры по адресу (6). Это не причуда, это полезно. При нажатии на эту ссылку откроется файл, содержащий множество примеров, которые вы можете копировать, вставлять и изменять. По сути, это то же самое, что происходит, когда вы нажимаете кнопку пользовательского интерфейса в позиции (1). На мой взгляд, открывать образцы быстрее и проще, потому что вы можете увидеть их все в одном месте. Чтобы использовать его, просто нажмите ссылку ***Examples**, как показано на рис. 9.24:

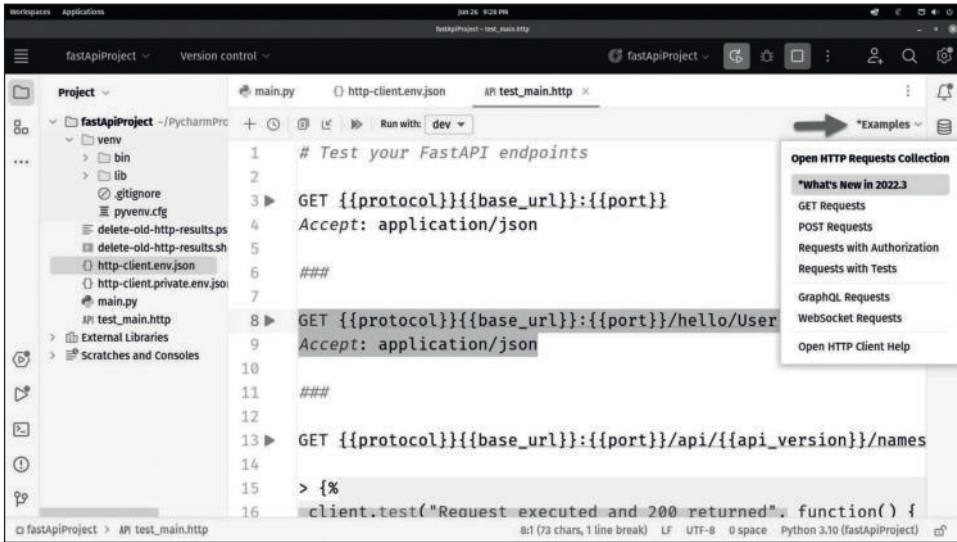


Рис. 9.24. Тестовые примеры разбиты по категориям

Мне нравится использовать открытые сбоку образцы POST-запросов, как показано на рис. 9.25.

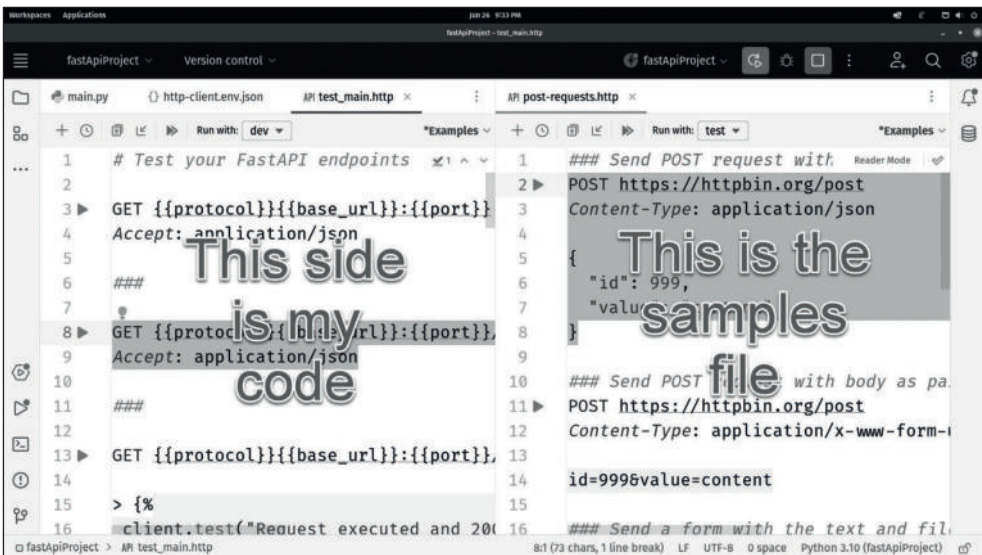


Рис. 9.25. Вы можете открыть образцы сбоку для удобства вставки

Чтобы разделить файлы таким образом, просто кликните вкладку правой кнопкой мыши и выберите **Split Right** или влево, вверх или вниз в соответствии с вашими предпочтениями.

Создание тестов

Давайте создадим эти тесты! Убедитесь, что вы работаете в `test_main.http`. Добавьте эти строки под последним тестом:

```
###
GET {{protocol}}{{base_url}}:{{port}}/api/{{api_version}}/names
```

Три хештега являются разделителем между тестами. Остальные – это переменные, взятые из переменных среды, которые мы создали ранее. Это будет преобразовано в `http://localhost:8000/api/v1/names`. Убедитесь, что ваше приложение запущено, и запустите тест. Вы должны увидеть результаты, подобные показанным на рис. 9.26:

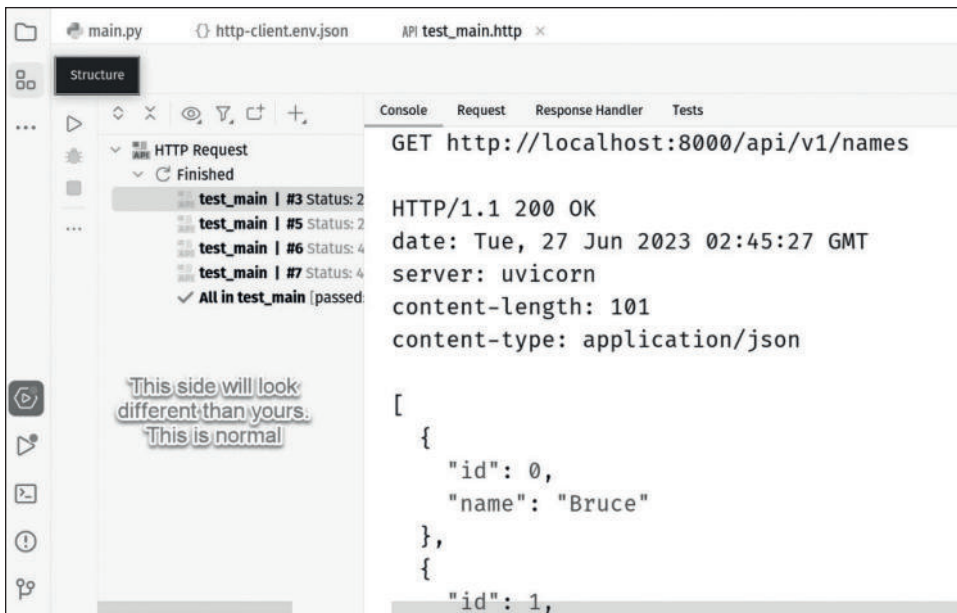


Рис. 9.26. При запуске теста вы можете увидеть данные ответа в журнале. Левая часть моих результатов обычно будет отличаться от вашей

Это приятно, правда? Вы можете видеть, что получили правильный код состояния (200), и можете просмотреть свои данные. Что, если бы вы могли превратить это в полноценный тест? Вы можете! Вернитесь к файлу `test_main.http`. Каждому великому моменту американской, а возможно, и мировой истории предшествовала фраза, которую я произнесу дальше: «Подержи мое пиво и смотри, как надо».

Прямо под тестом добавьте этот код:

```
> {%
  client.test("Request executed and 200 returned", function() {
    client.assert(response.status === 200, `Response status is
    ${response.status} but should be 200`);
  });
```

Первый персонаж имеет решающее значение. Вам нужно ввести `>`, а затем `{%}`. Если вы этого не сделаете, PyCharm сильно рассердится! Просто взгляните на волшебство мгновения. Примите все это во внимание. Перед нами испытание. НАПИСАНО НА JAVASCRIPT! Мне жаль. Я должен был предупредить, чтобы вы сели или что-то в этом роде. Однако я упомянул, что здесь замешано пиво. Здесь есть какая-то магия. В этом тестовом окне есть объект `client`, мало чем отличающийся от объекта `window`, который всегда присутствует в окне браузера. Мы вызываем метод `.test` для этого клиентского объекта и передаем ему два аргумента. Первый – это строка, описывающая тест. Это может быть все, что вы захотите. Содержимое никак не влияет на тест. Второй аргумент – это анонимная функция, которая фактически выполняет тест. Если вы не знакомы с использованием лямбда-функций в JavaScript¹, вам просто придется освоить их и скопировать эти тесты из примеров. Я полагаю, вы также могли бы изучить JavaScript, но это может занять некоторое время. К счастью, часть `client.assert` похожа на тесты, которые мы написали в главе 6, так что давайте продолжим. Утверждение принимает выражение, которое будет иметь значение `true` или `false`. JavaScript использует тройной знак равенства для проверки равенства без принуждения. Если вы еще не слышали об этом, просто знайте, что вам всегда следует использовать тройной знак равенства, потому что использование двойного означает что-то другое, а вам этого не нужно. Двойной знак равенства, который является нормой в Python, может привести к ложному сбою в вашем тестовом коде JavaScript, поскольку JavaScript попытается принудительно совместить любые типы данных, с которыми он сталкивается. Тройное равенство избегает этого.

После условного оператора у нас есть литерал шаблона JavaScript. Это эквивалент f-строки Python. Для цитат используются гравюрные метки, которые представляют собой сдвинутую тильду (~) на клавиатуре США и выглядят как толстые одинарные кавычки: ```. Заключение строки в гравюрные метки указывает, что строка является литералом шаблона. Вы можете заменять выражения или значения переменных, используя `${whatever}` в фигурных скобках. Итак, в Python, учитывая переменную `foo = bar`, f-строка будет выглядеть следующим образом:

```
f"I'd rather be at the {foo}."
```

В JavaScript, учитывая `const foo = "bar"`, это будет выглядеть так:

```
`I'd rather be at the ${foo}`
```

Здесь я использовал шаблонный литерал, чтобы предоставить немного больше информации, поскольку эта вторая строка буквально предоставляет сообщение, которое появляется в случае неудачного теста.

¹ Лямбда-функции JavaScript – это краткие анонимные функции, которые могут быть использованы для определения функции в месте их вызова. Они могут быть использованы для достижения простоты и элегантности в коде, а также для улучшения читаемости кода. Также они могут быть использованы в различных ситуациях, например в качестве аргумента для другой функции или для создания замыкания. – *Прим. ред.*

Этот конкретный тест проверяет, что код ответа HTTP для запроса был 200, что указывает на то, что он прошел успешно. Если вы не примените такой тест, ваши результаты всегда будут зелеными, даже если ваш код конечной точки расплавил сервер и вызвал великого Ктулху из его глубокого сна в самом сердце космоса. Это тест, который следует применять каждый раз, хотя ожидаемый код может меняться в зависимости от того, что вы делаете.

Дополнительные тесты, которые вы всегда должны добавлять

Проверка кода состояния – это минимум. Я утверждаю (видите, что я там сделал?), что вам следует проверить как можно больше вещей. Здесь я добавлю тест, чтобы убедиться, что возвращаемые данные имеют тип `mime application/json`, что важно для того, как клиенты будут использовать эти данные. Добавьте это под наш предыдущий тест:

```
client.test("Response should be json", function() {
    const type = response.contentType.mimeType;

    client.assert(type === "application/json", `Expected application/json
but got ${type}`);
});
%}
```

Помните, это JavaScript! Итак, я добавил константу с именем `type` и извлек значение из другого магического объекта, доступного в тестовом окне. Именно это дает вам `response.contentType.mimeType`. Мы делаем проверку, чтобы убедиться, что это `"application/json"`. Лично я считаю, что внимание к деталям важно, и считаю API, созданные профессионалами, очень непрофессиональными, если тип контента установлен неправильно. Так уж получилось, что FastAPI делает это за нас, но не каждый фреймворк сможет это сделать.

Теперь, когда вы освоили основы, я приглашаю вас изучить мой готовый файл `test_main.http` в примере кода этой главы. Там есть тесты для всех условий, которые мы помещаем в конечную точку PUT, позволяющие вам убедиться, что статус 400 возвращается, если ваш пользователь отправляет нечисловой идентификатор. Существует также тест, позволяющий убедиться, что вы получите ошибку 404, если ваш идентификатор выходит за пределы допустимого диапазона для списка `names`.

РЕДАКТИРОВАНИЕ И ОТЛАДКА ПОЛНОФУНКЦИОНАЛЬНОГО ПРИЛОЖЕНИЯ ПУТЕМ СОЕДИНЕНИЯ ПРОЕКТОВ

PyCharm имеет возможность объединять несколько проектов, что позволяет вам работать над полнофункциональными приложениями в одном экземпляре IDE. Когда вы запускаете несколько соединенных проектов в PyCharm IDE, можете легко отладить их вместе! Одной этой возможности, вероятно, достаточно, чтобы прекратить использовать vim или VS Code и никогда не оглядываться назад! Хотя это можно сделать и в других IDE или даже в VS Code, PyCharm делает это настолько простым, что вам, скорее всего, не захочется их использовать.

Создание приложения React в отдельном проекте

Чтобы создать интерфейс для нашего бэкенда FastAPI, мы собираемся использовать React. Как обычно, у меня здесь не будет места, чтобы много рассказывать вам о React. Я позабочусь о том, чтобы в разделе «Дальнейшее чтение» в конце этой главы была справочная информация.

Чтобы создать приложение React в PyCharm Professional, просто используйте обычную операцию **File | Project**, к которой вы, надеюсь, уже привыкли. Существует шаблон проекта, использующий продукт с открытым исходным кодом под названием **create-react-app (CRA)**. Это инструмент, широко используемый разработчиками React, потому что, по правде говоря, настройка полноценного приложения React с нуля – это утомительно и отнимает много времени.

Чтобы заставить скрипт create-react-app работать, вам необходимо установить Node.js. Если вы этого не сделаете, PyCharm попытается установить его за вас, как вы можете видеть на рис. 9.27:

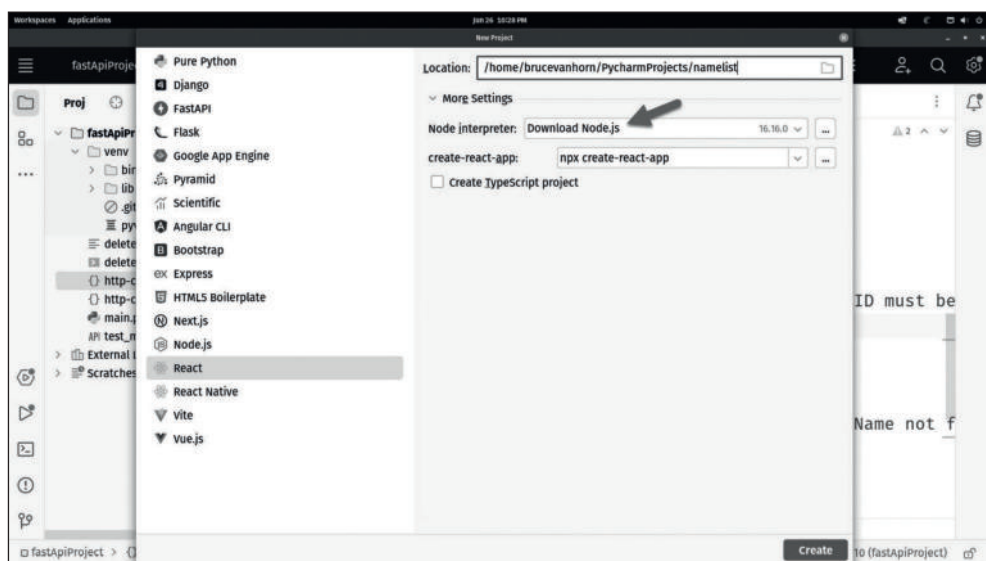


Рис. 9.27. При создании проекта JavaScript, например проекта React, вам необходимо установить Node.js, иначе PyCharm попытается установить его за вас

По моему опыту, это не всегда срабатывает. Такая же функция существует в Visual Studio, и я обычно советую своим студентам устанавливать Node.js вручную, а не использовать IDE. В общем, IDE часто не указывают на последнюю версию Node.js. На рис. 9.27 я вижу, что PyCharm намерен установить версию 16.16.0, которая не является последней. Кроме того, при выборе правильной версии я также рекомендую вам использовать последнюю версию Node.js с **долгосрочной поддержкой (LTS)**, а не версию с наибольшим номером версии. Продукт LTS гарантированно стабилен, тогда как самая новая версия – нет.

Еще одна вещь, которую я рекомендую для Node.js и Python, если на то пошло, – это использовать менеджер пакетов, такой как **Chocolately** (<https://chocolatey.org/>) или **Homebrew** (<https://brew.sh/>). Менеджеры пакетов являются стандартными

в Linux, и каждый дистрибутив использует свои собственные, поэтому, если вы используете Linux, я предполагаю, что вы знаете, какой из них вам нужен. Причина, по которой я рекомендую использовать менеджеры пакетов для установки Node.js, заключается в том, что менеджер пакетов позволяет очень легко обновить или даже полностью удалить ваше программное обеспечение, если это вам когда-нибудь понадобится. Если хотите попробовать Homebrew для Mac, на моем веб-сайте есть короткий бесплатный курс, который научит вас настраивать компьютер Apple для разработки Node.js. Вы можете найти его по адресу <https://www.maddevskilz.com/courses/setting-up-a-nodejs-development-workstation-in-macos>. Я подумая о том, чтобы сделать больше такого после того, как закончу писать эту книгу, поэтому загляните на сайт, если хотите увидеть версию для Windows.

Установив Node.js вручную, вернемся к созданию нашего проекта React. Я назвал мой проект `namelist`. React требователен к именам, поэтому я выбрал то, которое, как я знаю, подойдет. Оставляю более красивые названия для книг о React. Теперь нажмите кнопку **Create**. CRA занимает некоторое время, но, как только это закончится, у вас должна быть готова к работе папка проекта. Теперь мы готовы к работе на 100 %, в полном объеме! Далее я собираюсь показать вам мою любимую технику в PyCharm, позволяющую работать с обоими проектами так, как если бы они были одним, сохраняя при этом полное отделение кода пользовательского интерфейса от серверной части.

Привязка проекта к проекту FastAPI, который мы создали ранее

Откройте папку `fastAPIProject`, которую мы создали в начале этой главы. Теперь используйте меню **File | Open**, чтобы открыть папку `namelist`, содержащую проект React. Обычно, делая это, мы говорим PyCharm открыть проект либо в новом окне, либо в текущем окне. На этот раз я хочу, чтобы вы сказали ему прикрепить проект к текущему, как показано на рис. 9.28:

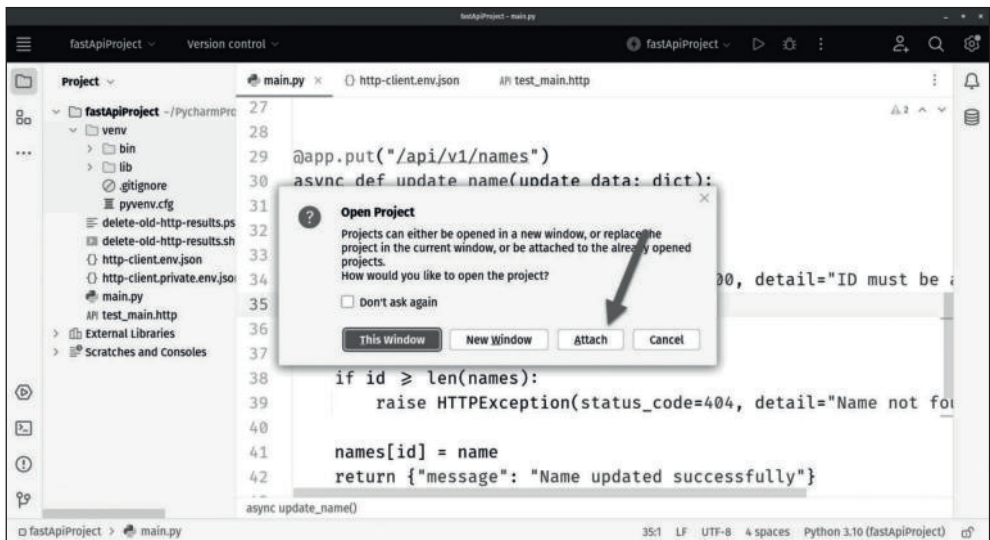


Рис. 9.28. Выберите опцию Attach при открытии проекта React

Оба проекта будут открыты одновременно в IDE. Это очевидно в окне проекта, показанном на рис. 9.29:

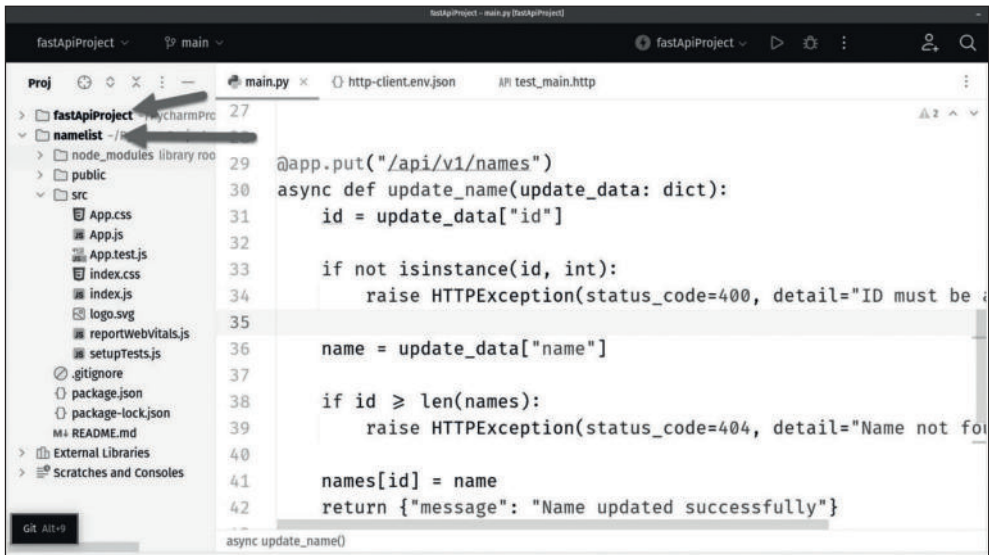


Рис. 9.29. Обе половины полнофункционального приложения могут быть открыты одновременно, что позволяет запускать и отлаживать их, как если бы они были одним проектом

Работа с соединенными проектами доставляет массу удовольствия при разработке полного стека. Рассмотрим преимущества.

- Технически это отдельные проекты в разных репозиториях.
- PyCharm позволяет создавать конфигурации запуска и отладки для всех соединенных проектов. Например, в вашем приложении React можно установить точку останова для проверки некоторой формы данных непосредственно перед их отправкой. Затем в своем проекте FastAPI вы можете установить точку останова в функции конечной точки для проверки полученных данных, что упрощает поиск ошибок в форматировании данных между двумя проектами.
- Другие люди, которые не способны работать в полном стеке, могут по-прежнему работать над отдельными проектами.

В главе 11 мы узнаем, что вы можете работать с внешним интерфейсом, серверной частью и базами данных – и все это в одном окне PyCharm. Для полнофункциональной разработки PyCharm трудно превзойти! Но сначала в следующей главе давайте рассмотрим третью среду веб-разработки, поддерживаемую PyCharm: Pyramid.

КРАТКОЕ СОДЕРЖАНИЕ

В этой главе мы рассмотрели все, что вам нужно знать, чтобы приступить к созданию проекта RESTful API с использованием FastAPI в PyCharm.

FastAPI отличается от Flask и многих других шаблонно ориентированных платформ веб-разработки тем, что он специально разработан для создания только RESTful API. RESTful API – это серверная часть (backend), которая отделена от любой разметки внешнего интерфейса (frontend), макета (layout), интерактивности или логики отображения (display logic). Вместо этого API фокусируется исключительно на получении запросов, несущих данные о взаимодействии с пользователем, и возврате обработанных данных, например полученных или обработанных в реляционной базе данных.

Поскольку эти типы приложений ориентированы на данные, мы узнали, что SPA обычно используется в качестве внешнего уровня представления. Ряд современных фреймворков поддерживают эту парадигму, включая React, Angular и Vue. Задача внешнего приложения – контролировать состояние приложения, поскольку HTTP – это протокол без сохранения состояния, и любой серверный интерфейс плохо подходит для этой задачи.

Создать проект FastAPI было легко с помощью шаблона, встроенного в PyCharm Professional. Он генерировал стартовый код, а также специальный тип тестового файла, уникальный для PyCharm. HTTP-файл содержит спецификацию запроса и среду тестирования, основанную на JavaScript, которая позволяет нам проверять ответ с помощью той же логики утверждений, которую мы видели в нашем предыдущем знакомстве с модульным тестированием.

Наконец, мы создали внешний интерфейс React в отдельном, но подключенном (прикрепленном) проекте в PyCharm, что позволяет вам разрабатывать полнофункциональное приложение без смешивания кода JavaScript внешнего интерфейса с внутренним кодом Python. В следующей главе мы обсудим Django и Pyramid.

Вопросы

1. Какой фреймворк лежит в основе FastAPI и чем он отличается от Flask и Werkzeug?
2. Что подразумевается под «передачей репрезентативного состояния» и какую проблему она решает?
3. Где хранится состояние приложения в проекте RESTful API, использующем SPA в качестве внешнего интерфейса?
4. Каковы четыре наиболее широко используемых метода HTTP?
5. Что такое методы CRUD и как методы HTTP сопоставляются с методами CRUD?
6. Каковы преимущества разделения интерфейсных и серверных проектов и как PyCharm упрощает работу с такими полнофункциональными проектами?

ДАЛЬНЕЙШЕЕ ЧТЕНИЕ

- Pandey, R. (2023) Build Full Stack Projects with FARM Stack [Video]. Packtpub.com <https://www.packtpub.com/product/build-full-stack-projects-with-farm-stack-video/9781803236667>.

- Van Horn, B. (2021) Setting Up a Python Development Workstation in Windows 10.
- maddevskilz.com <https://www.maddevskilz.com/courses/setting-up-a-python-development-workstation-in-windows>.
- Van Horn, B. (2021) Setting Up a NodeJS Development Workstation in MacOS.
- maddevskilz.com <https://www.maddevskilz.com/courses/setting-up-a-nodejs-development-workstation-in-macos>.

Полнофункциональные фреймворки – Django и Pyramid

Веб-фреймворки, которые мы рассмотрели до сих пор, были образцовыми в плане своей работы. Flask – это беспристрастный микрофреймворк. Под этим мы подразумеваем, что Flask обрабатывает только маршрутизацию конечных точек и (при необходимости) обслуживание шаблонного содержимого страницы. FastAPI имеет структуру, специально предназначенную для создания RESTful API, а не для обслуживания контента, помимо данных. Он также имеет модель асинхронного программирования, напоминающую современные фреймворки JavaScript, работающие в NodeJS.

Есть еще две платформы, которые вы найдете в меню **New Project** в PyCharm Professional, и мы собираемся рассказать о них здесь. **Django** – очень популярный фреймворк, который с философской точки зрения является диаметральной противоположностью Flask. Django – это очень пристрастный фреймворк, который пытается сделать за вас выбор любой платформы и фреймворка.

Последняя структура, о которой мы поговорим, – это **Pyramid**. Pyramid стремится найти золотую середину между Flask и Django, предлагая больше встроенных функций, чем Flask, но большую гибкость, чем Django.

В этой главе вы узнаете следующее:

- как создать проект Django, используя новый шаблон проекта PyCharm Professional,
- как определить основные файлы и папки, присутствующие в шаблонном проекте, созданном PyCharm,
- как использовать инструменты Django в PyCharm для задач `manage.py`,
- как создать проект Pyramid в PyCharm Professional.

В предыдущих главах мы уже рассмотрели много теории о том, как работают веб-приложения, поэтому давайте сразу приступим к разработке этих двух платформ с использованием PyCharm. Как и в других средах веб-разработки, эти функции доступны только в профессиональной версии PyCharm.

ТЕХНИЧЕСКИЕ ТРЕБОВАНИЯ

Чтобы пройти эту главу, как и остальную часть книги, вам понадобится следующее:

- установленный и работающий интерпретатор Python. Я буду использовать последнюю версию <https://python.org>;
- установленные копии `pip` и `virtualenv`. Вы получаете их автоматически при установке Python в Windows, а в macOS они включены в каждую систему. Если вы используете Linux, необходимо отдельно установить менеджеры пакетов, такие как `pip`, и инструменты виртуальной среды, такие как `virtualenv`. В наших примерах будут использоваться `pip` и `virtualenv`;
- установленная и рабочая копия PyCharm. Установка была описана в главе 2. Я использую сборку PyCharm Professional 2023.1 (сборка № PY-231.8109.197) с включенным новым пользовательским интерфейсом;
- пример исходного кода этой книги с GitHub. Мы рассмотрели клонирование кода в главе 2. Соответствующий код этой главы вы найдете на <https://github.com/PacktPublishing/Hands-On-Application-Development-with-PyCharm---Second-Edition/tree/main/chapter-10>.

ЧТО ЗА СУМАТОХА ВОКРУГ ДЖАНГО?

Если вы спросите большинство разработчиков Python, какой фреймворк они предпочитают, я готов поспорить, что между Flask и Django предпочтения распределятся примерно поровну. Я также готов поспорить, что поклонники Django – это восторженные фанаты, в то время как Flask больше похож на вещь, которую вы используете для выполнения задач в коллективе. Это похоже на восторг от коктейля «отвертка». Django гораздо более своеволен и, как следствие, сильнее «бьет по мозгам». Вы едва замечаете Flask, потому что он всего лишь как кусочек пазла. Django – это все части пазла в одной коробке, там же и клей для склеивания его в картину, плюс дорогая рамка для этой картины, плюс пляжный домик, чтобы было где ее повесить. На подушках в этом пляжном домике могут даже валяться мятные конфетки, но это уже не наверняка.

Django, таким образом, обещает стать веб-фреймворком, который справится с тяжелыми и повторяющимися аспектами разработки веб-приложений. Веб-разработчики могут сосредоточиться на конкретной логике своих приложений. Веб-фреймворки обычно включают в свою структуру общие шаблоны проектирования и передовой опыт, так что веб-приложение, разработанное с помощью фреймворка, по умолчанию будет соответствовать общим стандартам, и его разработчику не придется вручную интегрировать эти стандарты в приложение.

Django стремится иметь все необходимое для разработки веб-приложения, встроенное в инфраструктуру. Поскольку это правда, вам вообще не нужно думать о своем стеке. Django становится больше, чем просто фреймворком; он становится этосом. Вы становитесь поклонником пути Django. Вы можете даже сказать: «Таков путь!», и Django ответит вам этой же фразой¹. Вот некоторые вещи, в которых Django хорош.

¹ «Таков путь!» – фраза главного героя сериала «Мандалорец», части вселенной «Звездных войн» Джорджа Лукаса, одним из персонажей которой является **Джанго** Фетт. – Прим. ред.

- **Скорость.** Как и сам Python, Django подчеркивает простоту разработки и воплощения идей в реальный код. Благодаря простым, но обширным API-интерфейсам Django стремится обеспечить широкий спектр веб-приложений и функций.
- **Безопасность.** Веб-разработка – одна из тем программирования, в которой безопасность является наивысшим приоритетом. Платформа Django предлагает функции, которые помогают веб-разработчикам, новичкам и экспертам избегать недостатков безопасности в их приложениях.
- **Масштабируемость.** По мере того как веб-сайт привлекает все больше клиентов, масштабируемость становится все более важной. Масштабируемость в Django может быть достигнута гибкими и интуитивно понятными способами; фактически некоторые из крупнейших сайтов в интернете (Bitbucket, Instagram, Pinterest и т. д.) созданы с использованием Django именно по этой причине.

Ни один из этих вопросов не рассматривается во Flask конкретно. На самом деле Flask обычно не используется для больших веб-приложений, поскольку он относительно медленный. Лично я перенес весь серверный код своего продукта из Flask в FastAPI, и производительность заметно увеличилась.

Пока что Django звучит намного лучше, чем все, о чем мы говорили до сих пор. Несмотря на то что Django является популярным и мощным веб-фреймворком, у него, как и у любой технологии, есть свои недоброжелатели. Некоторые из распространенных критических замечаний и опасений, высказанных в отношении Django, приведены ниже.

- **Крутая кривая обучения.** Некоторые разработчики считают кривую обучения Django относительно крутой, особенно для новичков с ограниченным опытом в Python или веб-разработке. Его обширный набор функций и всеобъемлющий характер могут сбить с толку новичков.
- **Магия и абстракция.** Акцент Django на «батареях в комплекте» и абстракции может оказаться палкой о двух концах. Хотя это экономит время разработки, некоторые разработчики утверждают, что это может скрыть основные механизмы, усложняя понимание и устранение сложных проблем.
- **Накладные расходы для небольших проектов.** Некоторые разработчики считают, что для небольших проектов или простых веб-сайтов богатство функций и структура Django могут быть излишними, добавляя ненужные накладные расходы и сложность.
- **Монолитная природа.** Критики утверждают, что Django – это монолитная среда, и она, возможно, не лучшим образом подходит для архитектур микросервисов или узкоспециализированных приложений, где облегченная структура может быть более подходящей.
- **Гибкость против диктата.** Django следует определенной философии разработки и применяет определенные шаблоны, которые некоторые разработчики считают чрезмерно навязанными. Это может привести к дебатам о «способе Джанго» в сравнении с альтернативными подходами.
- **Производительность.** Хотя Django достаточно производителен, некоторые разработчики утверждают, что он может быть не таким быстрым, как некоторые микрофреймворки или специализированные инструменты.

Проекты, критичные к производительности, могут потребовать дополнительных усилий по оптимизации.

- **Ограничения ORM.** Хотя ORM в Django является мощным и простым в использовании, он может не охватывать все крайние случаи и не обеспечивать тот же уровень контроля, что и написание необработанных SQL-запросов. В определенных скриптах разработчики могут предпочесть использовать другие ORM или построители запросов.
- **Обновления версий.** Обновление между основными версиями Django иногда может быть сложной задачей, особенно для старых проектов, сильно зависящих от устаревших функций. Это может привести к проблемам с обслуживанием и дополнительным усилиям по разработке.
- **Сложность настройки.** Хотя Django предлагает некую гибкость, некоторым разработчикам сложно настроить определенные встроенные компоненты, такие как интерфейс администратора, в соответствии с конкретными требованиями разработки.
- **Сообщество и экосистема.** Хотя у Django большое и активное сообщество, у него может не быть такой обширной экосистемы или стольких сторонних пакетов, как у некоторых других веб-фреймворков.

Важно отметить, что, хотя эта критика имеет определенные основания, Django имеет обширную базу пользователей, и многие разработчики ценят его производительность, стабильность и обширный набор функций. В конечном итоге выбор веб-фреймворка зависит от конкретных потребностей и предпочтений проекта и команды разработчиков.

Компоненты фреймворка Django

Django включает в себя полный набор компонентов, которые делают его комплексной и многофункциональной средой для веб-разработки. Вот некоторые из ключевых компонентов.

- **Диспетчеризация URL-адресов (маршрутизация).** Django использует диспетчер URL-адресов для маршрутизации входящих HTTP-запросов к соответствующим функциям представления или представлениям на основе классов. Это обеспечивает чистые и логичные шаблоны URL-адресов для вашего веб-приложения.
- **Функции представления и представления на основе классов.** Представления (views) в Django отвечают за обработку пользовательских запросов и возврат HTTP-ответов. Вы можете использовать простые функции в качестве представлений или использовать представления Django на основе классов для более организованного и многократного использования кода.
- **Шаблоны.** Система шаблонов Django позволяет вам определять структуру и макет ваших веб-страниц, применяя шаблоны HTML с заполнителями для динамического контента. Такое разделение задач (логики и представления) упрощает поддержку и масштабирование вашего веб-приложения.
- **Архитектура Модель-Представление-Шаблон (MVT).** Подобно шаблону Модель-Представление-Контроллер (MVC), Django следует шаблону MVT. Модели представляют структуры данных и схему базы

данных, представления управляют логикой и обработкой, а шаблоны обеспечивают рендеринг выходных данных.

- **Объектно-реляционный маппинг (ORM).** ORM Django – одна из его определяющих особенностей. Оно обеспечивает высокоуровневый Python-способ взаимодействия с базами данных без необходимости писать необработанные SQL-запросы. Он позволяет вам определять модели как классы Python, а ORM обрабатывает маппинг этих моделей с таблицами базы данных.
- **Формы.** Django включает систему обработки форм, которая упрощает создание форм, проверку данных и обработку пользовательского ввода. Он помогает обрабатывать HTML-формы и преобразовывать предоставленные пользователем данные в типы данных Python.
- **Интерфейс администратора.** Интерфейс администратора Django – это автоматический интерфейс, который можно использовать для управления моделями данных вашего приложения. Он предоставляет готовое решение для управления данными модели и настраивается в соответствии с вашими конкретными потребностями.
- **Промежуточное программное обеспечение.** Компоненты промежуточного программного обеспечения в Django представляют собой перехватчики, которые позволяют вам обрабатывать запросы и ответы глобально до того, как они достигнут представления, или после того, как покинут представление. Он включает такие функции, как аутентификация, проверка безопасности и изменение запросов-ответов.
- **Статические файлы.** Django имеет встроенную поддержку для управления статическими файлами, такими как CSS, JavaScript и изображения. Это упрощает процесс обслуживания статического контента во время разработки и развертывания.
- **Аутентификация и авторизация.** Django предоставляет надежную систему аутентификации для управления учетными записями пользователей, разрешениями и группами. Это упрощает добавление в ваше приложение функций регистрации пользователей, входа в систему и управления паролями.
- **Интернационализация и локализация.** Django поддерживает интернационализацию и локализацию, позволяя создавать приложения, которые можно перевести на несколько языков и адаптировать к различным регионам.
- **Платформа тестирования.** Django поставляется с платформой тестирования, которая облегчает модульное и интеграционное тестирование вашего приложения. Вы можете написать тестовые примеры, чтобы убедиться, что ваш код работает должным образом, и избежать регрессий.

Несмотря на все эти накладные расходы, хорошо, что у нас есть такой мощный инструмент в PyCharm. Давайте углубимся в создание приложения Django.

СОЗДАНИЕ ПРОЕКТА DJANGO

Этот процесс ничем не отличается от большинства других проектов. Нажмите **File | New Project**, затем выберите опцию **Django**, как показано на рис. 10.1.

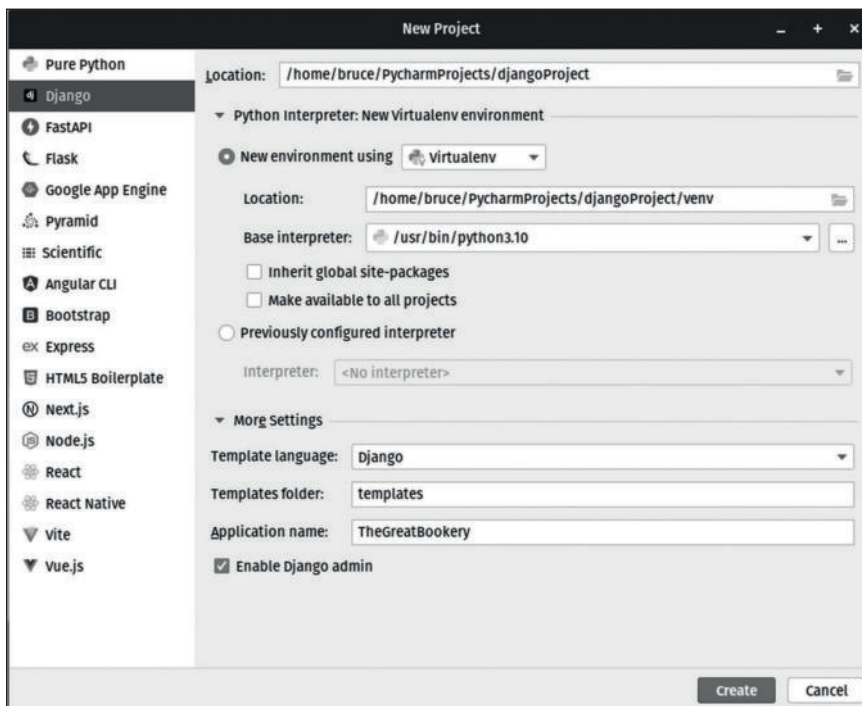


Рис. 10.1. Создание нового проекта Django очень похоже на создание любого другого проекта

Обязательно прокрутите вниз раздел **More Settings**. Здесь вы можете установить язык шаблона. Можете использовать либо **Django templating language**, либо **Jinja2**, который мы использовали в главе 9. Поскольку мы уже рассмотрели Jinja2, давайте остановимся на Django. Вы можете указать имя папки, которую будете использовать для шаблонов, и обязательно следует установить значение для поля **Application name**, как показано в нижней части рис. 10.1.

Не пропускайте название приложения

Если вы пропустите настройку **Application name** в нижней части рис. 10.1, PyCharm предположит, что вам нужен проект Django без каких-либо приложений. Конечно, можно добавить дополнительные приложения позже, используя инструмент `manage.py`, который мы рассмотрим далее, но это, вероятно, не то, что вам нужно в большинстве случаев, когда вы создаете новый проект. Более того, не следует называть приложение так же, как проект PyCharm, поскольку это может вызвать путаницу при индексации PyCharm, не говоря уже о людях, работающих над проектом.

Проекты Django часто содержат несколько приложений, и параметр **Application name** здесь используется для создания и присвоения имени пер-

вому приложению в проекте. В нашем проекте мы собираемся создать одно библиотечное приложение. Поскольку термин «библиотека» (*library*) в книгах по программированию неоднозначен, я уточню название приложения, чтобы мы понимали, что *library* – это место, где обслуживаются книги, а не библиотека программирования.

Получившийся в результате вновь созданный проект показан на рис. 10.2.

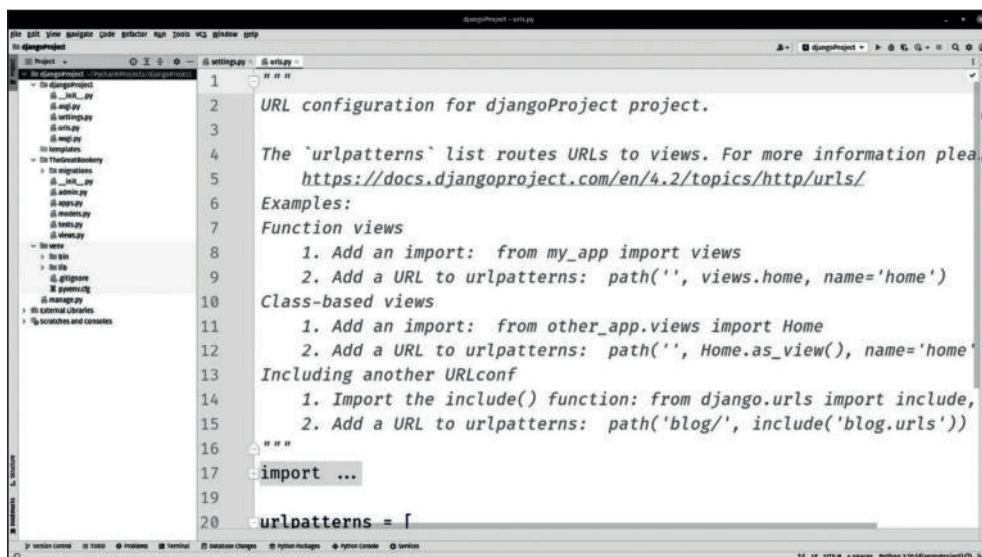


Рис. 10.2. Проект Django создан, и PyCharm открывает файлы `urls.py` и `settings.py` в качестве отправной точки

Учитывая очень длинный список компонентов, немного удивительно, как мало файлов было создано. PyCharm автоматически открыл два из них в качестве отправной точки: `urls.py` и `settings.py`. Давайте разберемся, для чего нужны эти и остальные файлы и папки в нашем только что созданном приложении.

СТРУКТУРА ПРОЕКТА DJANGO

Первое, что вы можете заметить, – это то, что проект, который я только что назвал `djangoProject`, имеет собственный набор файлов, отличный от приложения, который находится в папке `TheGreatBookery`. Все файлы внутри папки `djangoProject` используются при разворачивании вашего приложения в рабочей среде, а также для запуска встроенного сервера разработки.

Внутри папки `TheGreatBookery` у нас есть папка для миграции баз данных. Это характерно для приложений, использующих ORM. Вам нужен способ внесения изменений в производственную базу данных при выпуске новых выпусков в производство. Вы не можете просто удалить базу данных и перестроить ее, поскольку это приведет к удалению всех данных вашего приложения. Должна существовать система для переноса изменений схемы в существующую базу данных при сохранении существующих производственных данных. Система миграции Django использует скрипты миграции, хранящиеся в этой папке.

В папке шаблонов, которая технически находится за пределами папки приложения, мы храним наши HTML-шаблоны.

Возможно, вы сможете догадаться о функциях остальных файлов по их именам.

- `admin.py` используется для регистрации модулей, моделей данных и, что наиболее очевидно из названия, для управления пользовательским интерфейсом администратора.
- `apps.py` предоставляет центральное место для настройки поведения вашего приложения в проекте Django. Он позволяет вам указать различные настройки и метаданные для вашего приложения, такие как удобочитаемое имя, конфигурация приложения по умолчанию и сигналы.
- `models.py` будет содержать вашу модель данных, на которую влияет ORM.
- В файле `tests.py` будут храниться модульные тесты вашего приложения, которые запускаются с использованием собственной среды тестирования Django.
- `views.py` будет содержать функции просмотра. Это функции Python, которые принимают HTTP-запрос в качестве аргумента и возвращают HTTP-ответ. В рамках этих функций вы можете обрабатывать запрос, обрабатывать данные из моделей или других источников и генерировать соответствующий HTTP-ответ, часто путем визуализации шаблона с динамическими данными.

Начальная конфигурация

Хотя это книга не о Django, было бы упущением, если бы я не указал на некоторые моменты в файле `settings.py`. Как и Flask, Django имеет встроенный веб-сервер разработки, который не предназначен для использования в рабочей среде. Файл `settings.py` содержит некоторые опасные настройки в виде жестко запрограммированных значений, которые подходят только для локальной разработки. Откройте `settings.py` и найдите эти строки:

```
# SECURITY WARNING: keep the secret key used in production secret!
SECRET_KEY = 'django-insecure-39u&w+cgs2t4*jwe3nuz4y4j^s!s65^xb7eqtb_
a3bl!a_s%tn'

# SECURITY WARNING: don't run with debug turned on in production!
DEBUG = True
```

Что вам следует сделать заранее – так это обязательно нарисовать эти значения вне вашего кода. Никогда не следует жестко программировать секрет в коде вашего приложения, и никогда не следует передавать секреты в систему контроля версий! Вы можете экспортировать эти значения, используя переменные среды, файлы `.ini` или файлы `.env`, используя библиотеки из PyPi.org. Оставлять настройку `DEBUG` жестко запрограммированной может быть опасно, поскольку возникающие ошибки могут быть хорошо видны вместе с трассировкой стека и другими деталями, которыми могут воспользоваться злоумышленники.

Еще один параметр, который вы, возможно, захотите просмотреть, – это ядро базы данных. Вы найдете эти строки в нижней части файла `settings.py`:

```
# Database
# https://docs.djangoproject.com/en/4.2/ref/settings/#databases

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': BASE_DIR / 'db.sqlite3',
    }
}
```

PyCharm по умолчанию установил для нас SQLite3, который не совсем подходит для производственного приложения. SQLite3 – это файловая база данных, и ее использование для локальной разработки выгодно, поскольку не требует установки сервера. Вам не следует обслуживать производственное приложение с помощью SQLite3, и, по моему мнению, следует разрабатывать его с использованием той же базы данных, которую вы будете использовать в производстве.

Тем не менее мы не будем запускать это приложение в производство, поэтому я оставлю настройки такими, какие они есть. Ваша база данных появится в каталоге проекта в виде файла с именем db.sqlite3. Мы узнаем о возможностях базы данных PyCharm в главе 11. Знайте, что PyCharm имеет богатый набор инструментов для просмотра баз данных и работы с ними, включая SQLite3, который может помочь во время разработки.

Запуск проекта Django

Когда мы создавали проект, PyCharm создал для нас конфигурацию запуска. Мы подробно рассмотрели конфигурации запуска в главе 6. Мы просто хотим запустить тот, который сгенерирован PyCharm, поэтому просто нажмите зеленую кнопку **Run**, показанную на рис. 10.3.

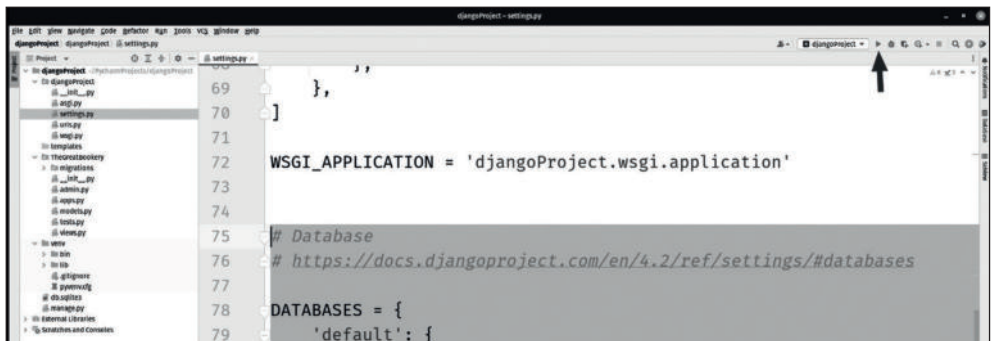


Рис. 10.3. Кнопка Run запускает сервер разработки

Запуск проекта откроет панель **Run** в нижней части окна IDE, как показано на рис. 10.4.

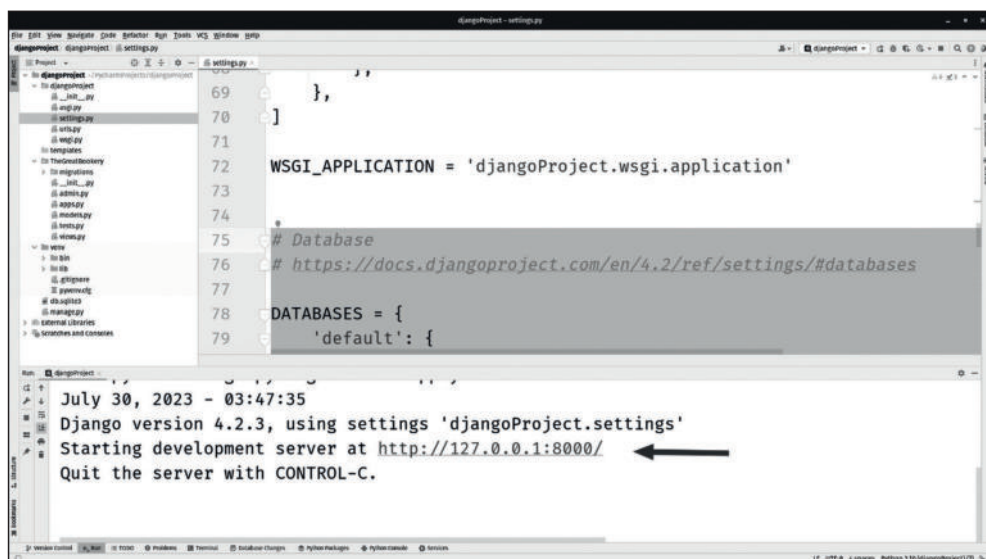


Рис. 10.4. Наш проект запущен и обслуживается через локальный порт 8000

Мы видим, что наш проект работает на порту 8000, и есть удобная ссылка, которая откроет наш браузер по умолчанию для работающего приложения, как показано на рис. 10.5.



Рис. 10.5. Щелчок по ссылке открывает браузер и отображает приятное сообщение, сообщающее, что все работает

PyCharm фактически создал для нас минимальное работающее приложение. В реальных проектах я обычно рассматриваю развертывание посредством **непрерывной интеграции (CI)** и **непрерывной доставки (CD)**. Один совет, который я всегда даю новым веб-разработчикам, – как можно раньше внедрить производственную механику. Если вы все сделаете правильно, не придется снова с ними возиться долгое время.

У JetBrains есть очень хороший сервер CI/CD под названием TeamCity. В течение последних 10 лет я использовал исключительно его и только в прошлом году неохотно перешел на Microsoft Azure DevOps. TeamCity – гораздо лучшая система, и она определенно стоит вашего времени и усилий, если вы ищете

сервер CI/CD. JetBrains даже предлагает плагины для PyCharm, которые позволяют вам контролировать и отслеживать сборки прямо из IDE.

Однако это глава о Django, поэтому давайте продолжим разговор и перейдем к работе с моделями данных.

СОЗДАНИЕ МОДЕЛЕЙ DJANGO

В приложении Django модели играют центральную и решающую роль. Они являются основой структуры данных приложения и отвечают за определение схемы базы данных и обработку операций с данными. Модели позволяют разработчикам взаимодействовать с основной базой данных объектно ориентированным способом, что упрощает управление данными и манипулирование ими.

Модели служат интерфейсом между приложением и базой данных. Они абстрагируют сложность SQL-запросов и предоставляют высокоуровневый API для выполнения операций **создания, чтения, обновления и удаления (CRUD)**. Используя методы модели и наборы запросов, разработчики могут взаимодействовать с базой данных без необходимости писать необработанный код SQL. Все, что связано с базой данных, управляющей приложением (включая то, что многие разработчики обычно упускают из виду, например проверку и ограничения), можно найти в модели Django.

Распространенным антишаблоном, который иногда возникает в проектах, в которых не используются ORM, является использование базы данных для инкапсуляции бизнес-логики. Обычно это оправдывается тем, что изменение базы данных происходит быстро, легко и не требует новой компиляции или нового развертывания. Это глупая экономика, и в современную эпоху, которая требует управления и документированного контроля за изменениями, она предается анафеме. Старший разработчик баз данных в моей команде, который также является техническим рецензентом главы 11, описывает эту практику как замену шин на полуприцепе, когда он движется по шоссе со скоростью 100 миль в час. Однако модели Django, поскольку технически они являются объектами, способны эффективно и без дополнительных табу обрабатывать бизнес-логику. Это связано с тем, что в базу данных переводится только структура модели данных, а логика остается внутри объекта.

При использовании ORM, который мы делаем, когда создаем модели и манипулируем ими, вам нужно помнить, что мы работаем с абстракцией нашей базы данных. Манипулируя экземплярами наших моделей, мы изменяем данные в базе данных. Аналогично, когда мы меняем структуру наших классов, мы изменяем структуру базы данных. Существуют различия между тем, как работают типы данных на вашем языке программирования, и тем, как они работают в базе данных. Мы увидим это в действии позже, а сейчас я хочу, чтобы вы подумали о том факте, что Python – это динамический язык, использующий утиную типизацию¹. В Python я могу изменить любую

¹ Утиная типизация (англ. *Duck typing*) – это концепция, характерная для объектно ориентированных языков программирования с динамической типизацией, согласно которой конкретный тип или класс объекта не важен, а важны лишь свойства и методы, которыми этот объект обладает. Другими словами, при работе с объектом его тип не проверяется, вместо этого проверяются свойства и методы этого объекта. – Прим. ред.

структуру или тип любым удобным для меня способом. С другой стороны, базы данных являются статическими и строго типизированными. ORM Django требует много работы, учитывая различия в парадигме между тем, как Python работает с классами, типами и переменными, и тем, как база данных работает с таблицами.

Давайте рассмотрим создание простой модели. Найдите файл `models.py`. Вы можете найти его на рис. 10.6 или воспользоваться функциями навигации в PyCharm, нажав `Ctrl + Shift + N` (**Navigate | File** в меню) и введя имя файла `models.py`.

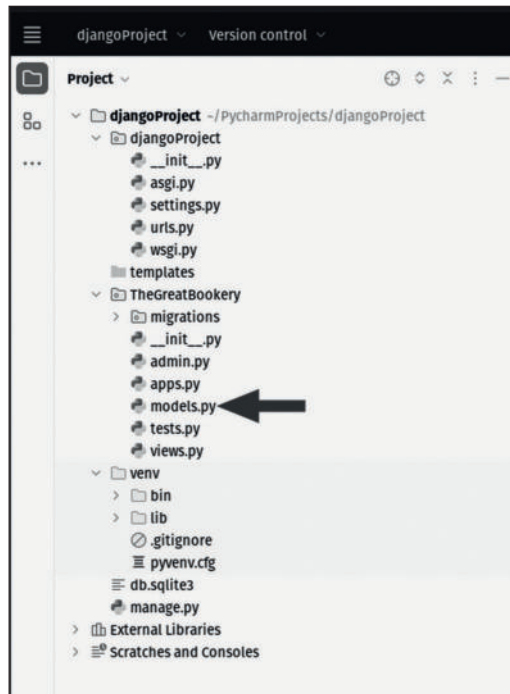


Рис. 10.6. Откройте файл `models.py`, чтобы добавить модель

Сгенерированный код будет выглядеть следующим образом:

```
from django.db import models

# Create your models here.
```

Мы собираемся создать две модели для нашего приложения Bookery: `author` и `book`. Для этого изменим код в `models.py` следующим образом:

```
from django.db import models

import datetime
from django.utils import timezone
```

Здесь мы добавили два импорта, связанных со временем и датами. Мы собираемся использовать эти библиотеки в некоторой бизнес-логике, которую при-

крепим к модели книги. Однако не будем забегать вперед. Давайте создадим простую модель для представления авторов книг:

```
# Create your models here.
class Author(models.Model):
    first_name = models.CharField(max_length=100)
    last_name = models.CharField(max_length=100)

    def __str__(self):
        return f'{self.last_name}, {self.first_name}'
```

Надеюсь, урок достаточно простой. Класс `Author` наследует класс `model.Model` Django. Есть два поля для имени и фамилии автора; используйте тип `models.CharField` для представления того, что будет `varchar`s в базе данных. Реляционные базы данных требуют объявления строгого типа, а в случае `varchar` (символьного поля переменной длины) мы должны указать максимальную длину. Метод `dunder string` (двойное подчеркивание) просто дает нам удобный формат, если мы запросим содержимое модели в виде строки. Здесь мы решили отформатировать имя автора как фамилию, запятую, пробел и имя.

Каждый ORM, независимо от платформы, работает одинаково. Его цель – позволить разработчикам работать только с объектами на оригинальном, родном языке. С учетом вышесказанного им обычно нужна собственная система типов, чтобы ORM мог осуществлять преобразование между системой типов языка (Python) и базой данных. В главе 11 вы узнаете, что типы данных базы данных отличаются от типов данных в языках программирования. Как минимум очевидны такие различия, как вызов строк `varchar`, а типы, включая `int` и `float`, существуют в обоих контекстах. Иногда они даже различаются на разных платформах баз данных. В `SQLite3` есть типы данных, которых нет в `MySQL`, и наоборот.

ORM должен работать везде, поэтому наличие собственной системы типов для моделей позволяет разработчику использовать единую систему типов, которую можно адаптировать к любой платформе базы данных. Это классическая реализация шаблона **адаптера** «Банда четырех»¹, о котором я рассказываю в своей книге «Практическая реализация шаблонов проектирования C#», изданной Packt.

Класс `author` не очень затейлив. Давайте напишем еще один интересный код в коде модели `Book`:

```
class Book(models.Model):
    title = models.CharField(max_length=200)
    author = models.ForeignKey(Author, on_delete=models.CASCADE,)
    pub_date = models.DateTimeField('date published')
```

Пока что у нас есть класс `Book`, наследуемый от `models.Model`, как и раньше. У нас есть поле заголовка длиной не более 200 символов. Поле `author` есть, но

¹ «Банда четырех» – это авторы книги «Образцы проектирования: элементы объектно ориентированного программного обеспечения, обеспечивающие многократное использование». – *Прим. ред.*

вместо того, чтобы делать его строкой, мы использовали класс `Author` для настройки **foreign key (внешнего ключа)**. На языке базы данных это означает, что данные об авторах представлены в таблице `authors`, которая управляется классом `Author`, созданным минуту назад. Отношения внешнего ключа указывают, что любая `Book`, которую мы добавляем в базу данных, должна содержать базу данных, связанную (это реляционная база данных) с существующим автором. С практической точки зрения это означает, что мы должны убедиться, что автор существует в базе данных, прежде чем мы сможем добавить книги автора в базу данных.

Ограничения внешнего ключа служат формой встроенной проверки данных, но они также помогают поддерживать целостность данных в базе данных путем соблюдения правил. Если бы я добавил автора в базу данных, а затем добавил несколько книг, написанных этим автором, между двумя таблицами была бы хорошая связь. Если бы я затем удалил автора из таблицы `authors`, у нас все равно осталась бы куча записей книг без связанного автора. Это будут сиротские записи. Сиротские записи плохи тем, что занимают место в базе данных и нарушают целостность взаимоотношений между авторами и книгами.

В этом классе, когда мы определяем поле `author`, мы делаем это с помощью ограничения внешнего ключа с фактическим правилом, охватывающим то, что происходит, когда вы удаляете автора. Код, который говорит `on_delete=models.CASCADE`, сообщает ORM, что при удалении автора удаляются и записи написанных им книг. Это называется каскадным удалением, поскольку удаление каскадируется от автора к записям книг, которые, в свою очередь, могут каскадно передаваться на другие таблицы, возможно, имеющие аналогичные отношения с книгами. Хорошо построенная база данных может сохранять чистоту и отсутствие бесхозных записей независимо от того, насколько сложной может быть структура базы данных.

Наконец, у нас есть дата, содержащая дату публикации:

```
pub_date = models.DateTimeField('date published')
```

Если поле `foreign key` не добавило вам достаточно остроты, давайте добавим еще одну вещь. Поскольку мы имеем дело с объектом и поскольку объекты могут содержать как данные, так и функциональную логику, давайте добавим функцию в нашу модель `Book`:

```
def was_published_recently(self):
    now = timezone.now()
    return now - datetime.timedelta(days=1) <= self.pub_date <= now
```

Это добавляет поле в нашу модель, но вместо того, чтобы эти данные сохранялись в базе данных, они вычисляются во время вызова метода. Мы берем статическое поле для даты публикации и вычисляем разницу (дельту) времени с текущей датой. Мы учитываем часовой пояс и представляем его как количество дней с момента публикации книги. Это позволяет нам показывать нашим пользователям возраст книги в днях, что может быть использовано для того, чтобы побудить авторов выпускать новые и обновленные издания своих книг.

МИГРАЦИЯ С ПОМОЩЬЮ MANAGE.PY

Одним из файлов, созданных для вашего проекта Django, является `manage.py`. Вы можете увидеть это на рис. 10.7.

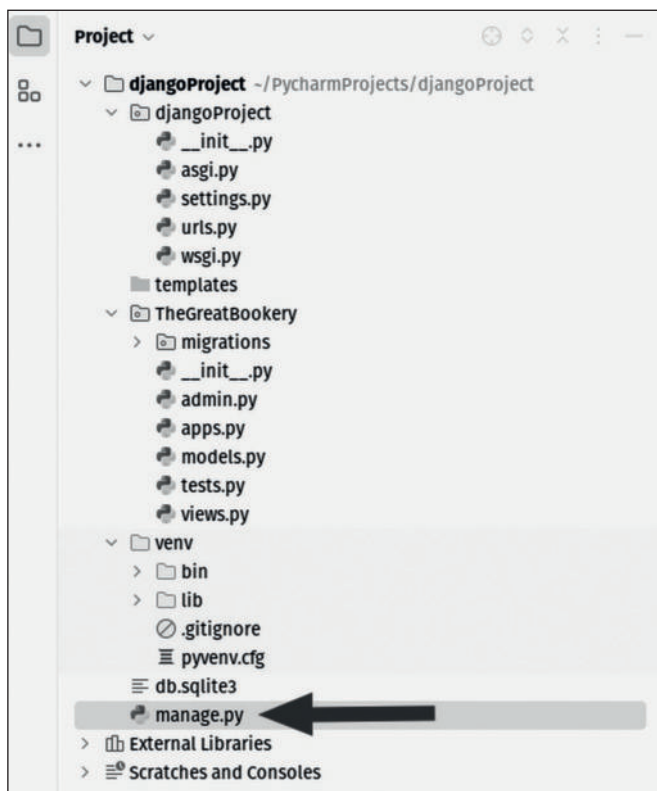


Рис. 10.7. Файл `manage.py` предоставляет утилиты для работы с вашим проектом Django, поэтому вам не нужно вводить длинные команды или запоминать полные пути

Цель задачи `manage.py` – позволить вам запускать команды, специфичные для Django, из командной строки без необходимости запоминать полный путь к вашему проекту Django или вручную настраивать среду Python. Используя `manage.py`, вы имеете гарантию, что команды выполняются в правильном контексте проекта Django.

Некоторые распространенные задачи, которые вы можете выполнять с помощью `manage.py`, приведены ниже.

- **Запуск сервера разработки.** Вы можете запустить сервер разработки Django с помощью `manage.py runserver`. Это позволяет тестировать приложение локально во время разработки. Ранее мы запускали проект Django, используя сгенерированную конфигурацию запуска. Мы могли бы также использовать для этого `manage.py`.
- **Создание и применение миграций базы данных.** С помощью `manage.py makemigrations` и `manage.py migrate` вы можете создавать и приме-

нять миграции базы данных соответственно. Это поможет управлять изменениями в ваших моделях и поддерживать актуальность схемы базы данных.

- **Создание суперпользователя.** Вы можете создать суперпользователя для интерфейса администратора Django, используя `manage.py createsuperuser`.
- **Запуск тестов.** Вы можете выполнить свой набор тестов с помощью `manage.py test`, чтобы убедиться, что ваше приложение работает должным образом.
- **Управление статическими файлами.** Вы можете собирать статические файлы и управлять ими с помощью `manage.py collectstatic`.
- **Управление переводами.** Для интернационализации и локализации вы можете использовать `manage.py makemessages` и `manage.py compilemessages`.

Это лишь краткий список. В `manage.py` есть еще много утилит. Файл `manage.py` служит точкой входа для команд управления Django, упрощая выполнение административных задач, не выходя из среды разработки. Перед переходом к кнопке терминала в PyCharm есть более простой способ работы с `manage.py`. Нажмите **Click Tools | Run manage.py task...**, как показано на рис. 10.8.

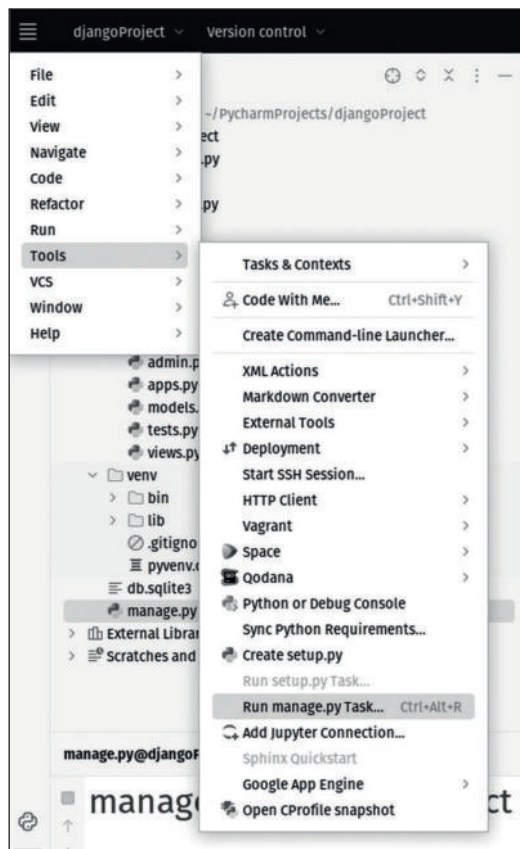


Рис. 10.8. PyCharm имеет специальную панель инструментов для работы с файлом `manage.py`

Это даст вам новую панель, как показано на рис. 10.9.

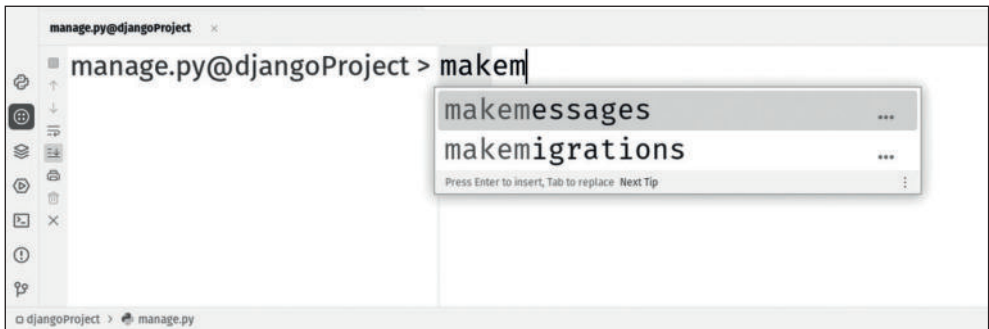


Рис. 10.9. Панель Run manage.py task – это специальный интерфейс командной строки, который позволяет легко вызывать команды, специфичные для файла manage.py Django

Панель manage.py открывается в нижней части IDE, как и многие другие. На первый взгляд это похоже на окно терминала, но оно специфично для Django и файла manage.py. Как видите, на панели предусмотрено автозаполнение, помогающее с командами.

Чтобы создать наши миграции, вам следует ввести в панель следующую команду:

```
makemigrations TheGreatBookery
```

Результат будет напоминать рис. 10.10.

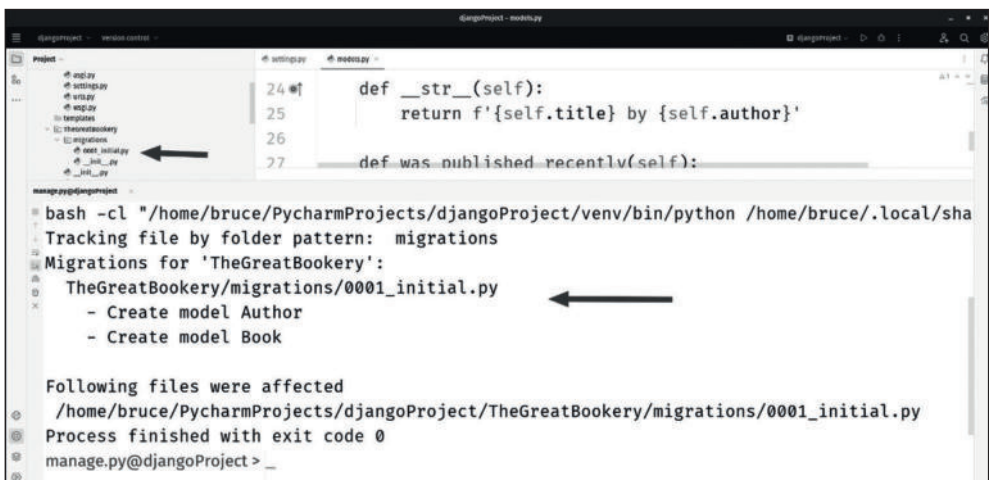


Рис. 10.10. Результат выполнения команды makemigrations

Вы найдете несколько сообщений на панели manage.py, а также новый файл в папке migration, который называется примерно 0001_initial.py. Команда makemigrations сгенерировала код, необходимый для миграции, но еще не выполнила ее. Выполнение миграции влияет на изменения в базе данных. Чтобы запустить миграцию, введите это в панель задач manage.py:

```
sqlmigrate TheGreatBookery 0001
```

Вывод команды, показанный на рис. 10.11, показывает, что структура базы данных обновлена в соответствии с моделями.

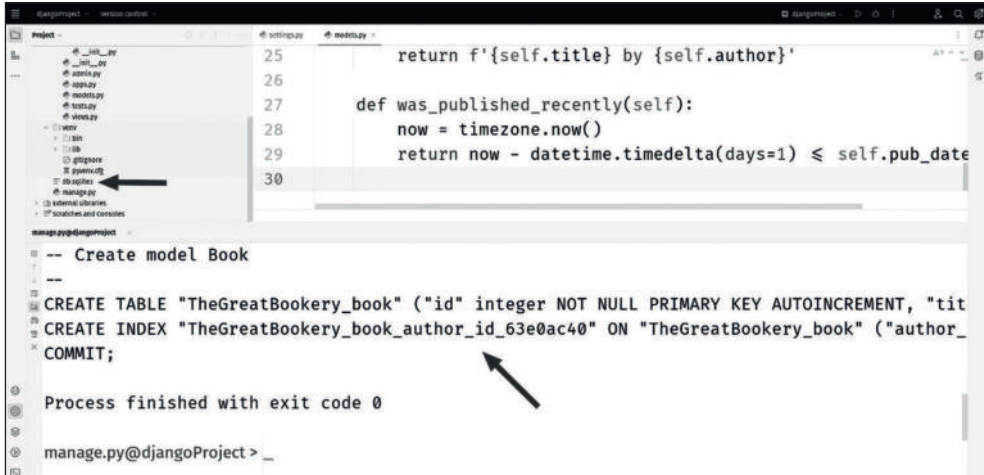


Рис. 10.11. Результат миграции – набор сообщений и внешний вид нашего файла базы данных

Мы видим набор операторов SQL, которые были созданы для приведения структуры базы данных в соответствие со структурой моделей. Мы также видим новый файл для нашей базы данных. Если вы помните, ранее мы видели, что имя и путь к этому файлу были заданы в файле settings.py. Поскольку SQLite3 – это файловая база данных, это и есть сама база данных. Это то, что вы, возможно, захотите исключить из контроля версий в реальном проекте.

ИНТЕРФЕЙС АДМИНИСТРАТОРА DJANGO

Это одно из убийственных преимуществ Django: он создает для вас веб-панель администрирования. Это означает, что вам не нужно создавать интерфейс для работы с пользователями, обработки входов в систему или создания экранов для простого ввода данных в структуру вашей модели. Первое, что нужно каждому приложению, – это администратор или суперпользователь. Приложения обычно имеют доступ пользователей на основе ролей, а суперпользователь – это пользователь, который может делать все, включая добавление новых пользователей. Чтобы приступить к работе над подобными вещами, разработчику может потребоваться два или три дня, но в Django это уже сделано.

Создание суперпользователя и вход в систему

Прежде чем пользователи, логины или любые другие преимущества, о которых мы здесь говорим, заработают, вам необходимо применить миграцию, которая была создана при создании проекта.

Давайте вернемся к нашей панели задач manage.py и введем следующую команду:


```
migrate
```

Вот так просто. При этом будут добавлены все базовые таблицы для приложения. Вероятно, нам следовало сделать это в первую очередь, но я не хотел прерывать ход того, над чем мы работали. Ладно, я забыл, но так звучит лучше.

Благодаря этому запуску миграции у нас теперь есть все таблицы и структуры в базе данных для поддержки функций входа в систему и управления пользователями Django. Далее давайте создадим суперпользователя. На панели введите следующее:

```
createsuperuser
```

Вам зададут ряд вопросов, предназначенных для создания суперпользователя для вашего приложения, как показано на рис. 10.12.

```
manage.py@djangoProject > createsuperuser
bash -cl "/home/bruce/PycharmProjects/djangoProject/venv/bin/python /home/bruce/.local/sha
Tracking file by folder pattern: migrations
Username (leave blank to use 'bruce'):
```

admin

```
Email address: admin@myproject.com
Warning: Password input may be echoed.
Password: P@ssw0rd
Warning: Password input may be echoed.
Password (again): P@ssw0rd
This password is too common.
Bypass password validation and create user anyway? [y/N]: y
Superuser created successfully.
manage.py@djangoProject > _
```

Рис. 10.12. Взаимодействие с панелью manage.py для создания суперпользователя

Давайте опробуем интерфейс управления! Хотя вы можете просто использовать конфигурацию запуска, давайте посмотрим, что значит запустить ее из панели задач manage.py. Введите следующее:

```
runserver
```

Затем кликните ссылку в сообщении панели, показанном на рис. 10.13.

```
manage.py@djangoProject > runserver
bash -cl "/home/bruce/PycharmProjects/djangoProject/venv/bin/python /home/bruce/.local/sha
Tracking file by folder pattern: migrations
Watching for file changes with StatReloader
Performing system checks ...

System check identified no issues (0 silenced).
July 30, 2023 - 23:04:06
Django version 4.2.3, using settings 'djangoProject.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
```

Рис. 10.13. На этот раз давайте запустим с панели!

По правде говоря, это то же самое, что вы получили бы, нажав зеленую кнопку **Run**, как мы это сделали при первом запуске приложения. После нажатия на ссылку ваш браузер откроется на той же странице, которую мы видели раньше. Измените URL-адрес в браузере на `http://127.0.0.1:8000/admin`. Вы попадете на страницу административного входа, показанную на рис. 10.14.

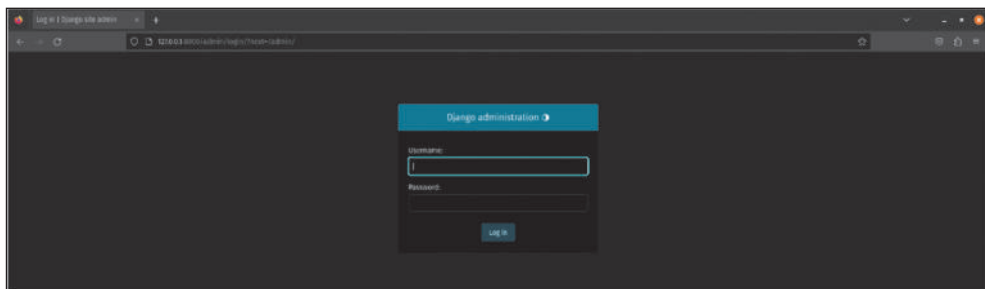


Рис. 10.14. Лучший способ входа в систему – тот, который вам не нужно создавать самостоятельно!

Введите имя пользователя и пароль суперпользователя, которые вы указали на панели задач. На рис. 10.15 показан экран администратора, который должен появиться.

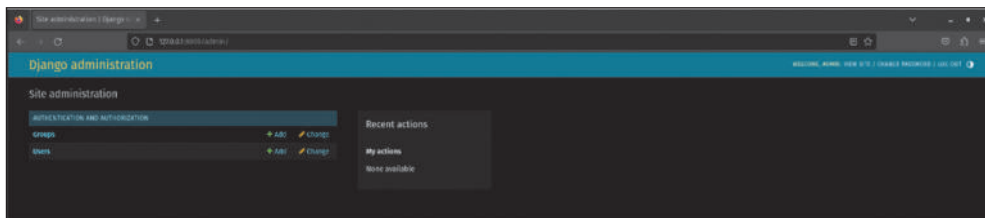


Рис. 10.15. Без написания кода мы можем добавлять пользователей и группы в приложение

Кликните мышью, чтобы увидеть все, что сгенерировал для нас Django. Вы можете добавлять пользователей и группы в приложение, даже если для этого не написали никакого кода – Django сделал это за нас. Недостатком здесь является то, что он был создан на основе идей проекта Django о том, как должна выглядеть панель администратора и как она должна работать. Хотя некоторые из этих функций можно настроить, помните, что я постоянно говорю, что Django *упрям*. В нем заложены предположения. Менее самоуверенные фреймворки потребуют от вас создания этого или использования плагина, но вы будете полностью контролировать, как эта функция будет реализована.

Добавление моделей Author и Book в интерфейс администратора

Хотя вы получаете пользователей и группы бесплатно, придется проделать небольшую работу, чтобы авторы и книги отображались в интерфейсе администратора. Начнем с модели Author.

Откройте файл `admin.py` в папке `TheGreatBookery` и найдите комментарий `# Register your models here.` Добавьте следующий код:

```
# Register your models here.
from .models import Author
admin.site.register(Author)
```

Если ваше приложение остановилось или произошел сбой, перезапустите его, а затем перейдите на страницу администратора, как и раньше. Теперь вы должны увидеть **Authors** на странице администратора, как показано на рис. 10.16.

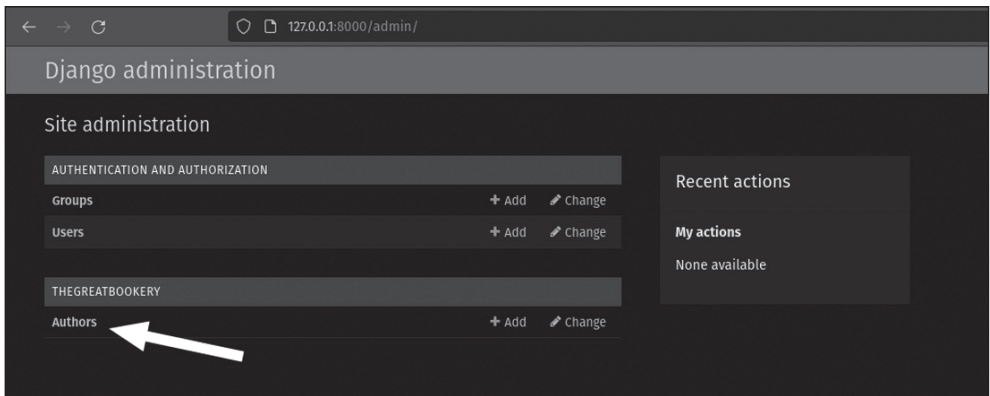


Рис. 10.16. Теперь мы можем без особых усилий добавлять, редактировать и удалять авторов

У нас есть модель `Author`; давайте добавим `Book`. Этот немного причудливее. Помните, что между Книгой и Автором существует связь. Каждый Автор может быть связан со многими Книгами, или, наоборот, многие Книги связаны с одним Автором.

Поскольку это так, нам нужно, чтобы интерфейс администратора не только отражал эти отношения, но и обеспечивал их соблюдение на уровне пользовательского интерфейса. Вы можете просто импортировать модель книги и ожидать, что она будет работать идеально.

Вернитесь к файлу `admin.py` и добавьте этот код:

```
from django.contrib import admin

# Register your models here.
from .models import Author, Book

class BookInline(admin.TabularInline):
    model = Book
    extra = 1

    fieldsets = [
        (None, {'fields': ['title']}),
        ('Date information', {'fields': ['pub_date']})
    ]
```

```
class AuthorAdmin(admin.ModelAdmin):
    inlines = [BookInline]

admin.site.register(Author, AuthorAdmin)
```

Этот код создает дополнение к пользовательскому интерфейсу Author, которое позволяет вам добавлять книги встроенным образом, т. е., когда вы добавляете автора, у вас будет возможность добавлять книги. Вы не можете иметь книги без предварительного добавления авторов. Поскольку мы определили отношения и пользовательский интерфейс таким образом, возможность добавлять и редактировать книги является частью администрирования автора. У вас даже не будет возможности ошибиться, добавив книги перед добавлением авторов, или каким-то образом отделить книгу от автора.

Перезагрузите сервер и обновите браузер. Ничего не изменилось! Подождите, все в порядке. Мы еще не пробовали добавить автора. Нажмите **Authors**, а затем – **Add**, и вы увидите интерфейс, показанный на рис. 10.17.

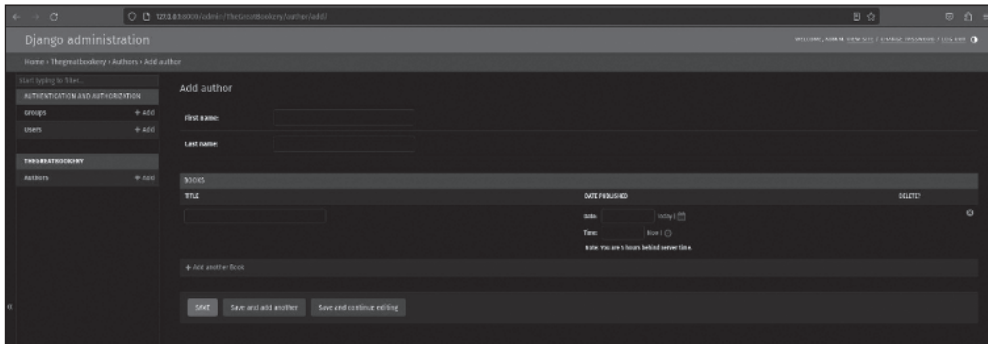


Рис. 10.17. Наш код добавил встроенные книги в пользовательский интерфейс для управления авторами

Как видите, форма добавления книг теперь встроена в форму работы с авторами.

СОЗДАНИЕ ПРЕДСТАВЛЕНИЙ DJANGO

Пользовательский интерфейс администратора просто волшебный, но на самом деле он предназначен только для создания простых экранов управления данными в базе данных. Рано или поздно вам придется создать настоящие экраны для самого приложения.

Здесь нам нужно быть осторожными с терминологией, потому что слово «представления» (views) не означает того, что вы могли бы ожидать, если вы разбегаетесь в более традиционных шаблонах проектирования веб-разработки. Преобладающий в отрасли шаблон называется **Модель-Представление-Контроллер** (MVC). Шаблон, используемый Django, называется **Модель-Представление-Шаблон** (MVT). *Представление* не означает одно и то же для двух шаблонов. Давайте сравним эти два случая.

MVC	МВТ
Модель относится к структуре данных, обычно из базы данных	Модель означает то же самое, что и в MVC
Представление относится к визуальным элементам, которыми обычно является пользовательский интерфейс	Представление относится к уровню контроллера, который принимает входящие запросы, выполняет логику и возвращает контент или данные
Контроллер относится к уровню, который принимает входящие запросы, выполняет логику и возвращает контент или данные	Шаблон относится к представлению в MVC, которое представляет собой элементы HTML или пользовательского интерфейса в приложении

Подводя итог, можно сказать, что представление Django на самом деле является контроллером, а шаблон Django будет представлением в приложении MVC. Итак, собираясь создать представление, мы говорим о той части приложения, которая получает запросы, что-то делает, а затем возвращает ответ.

Откройте файл `views.py` в папке `TheGreatBookery`. Измените код на этот:

```
from django.shortcuts import render
from .models import Book

# Create your views here.
def index(request):
    latest_books = Book.objects.order_by('-pub_date')[:5]
    context = {'latest_books': latest_books}
    return render(request, 'TheGreatBookery/index.html', context)
```

Здесь мы импортировали модель `Book`, затем определили метод, предназначенный для приема запроса, извлечения всех книг в базе данных и упорядочивания их по полю `pub_date` в порядке убывания, обозначенном знаком минус рядом с `pub_date`. Мы собираемся отображать только первые пять найденных книг, обозначенных срезом `[:5]`.

Далее создаем переменную с именем `context`, становящуюся словарем, который будет использоваться для передачи данных в шаблон во время рендеринга. В этом случае создается пара «ключ–значение», где ключом является `latest_books`, а значением – `QuerySet` последних книг, полученных на предыдущем шаге.

Возвращаемая строка вызывает метод `render`, импортированный в начало файла. Он передает объект `request`, путь к шаблону (который нам еще предстоит создать) и переменную словаря `context`, которую мы только что создали.

Эта функция просмотра не будет выполнять всю работу сама по себе. Чтобы получить возможность ее использовать, необходимо зарегистрировать ее в файле `urls.py`, чтобы у нас был маршрут для использования этой функции.

Откройте `urls.py` и измените код на следующий:

```
from django.contrib import admin
from django.urls import path
from TheGreatBookery.views import index
```

```
urlpatterns = [
    path('admin/', admin.site.urls),
    path('', index, name='index')
]
```

Мы добавили импорт в только что созданную функцию `index` в файле `views.py`. Затем добавили в качестве функции `path` пустую строку, которая будет отображать корневой маршрут приложения, ссылку на функцию `index`, которая будет выполняться в ответ на запросы корневого маршрута, и понятное имя.

Мы почти закончили. Есть еще одна вещь. Вернитесь к файлу `view.py`, и вы увидите проблемную желтую подсветку, см. рис. 10.18.

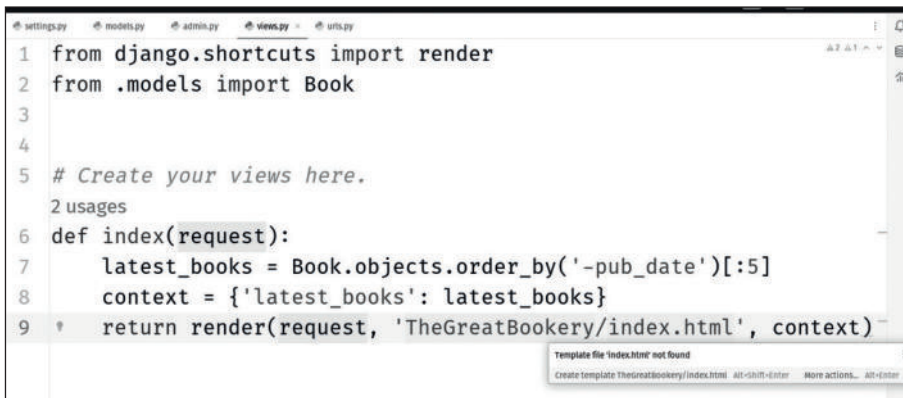


Рис. 10.18. У нас есть представление и сопоставленный URL-адрес, но мы еще не создали шаблон

Наведите указатель мыши на `index.html` и обратите внимание, что во всплывающей подсказке есть действие, позволяющее создать для нас шаблон, либо кликните синий текст ссылки в окне подсказки, или используйте `Alt + Shift + Enter`, как указано. На рис. 10.19 показано следующее диалоговое окно.

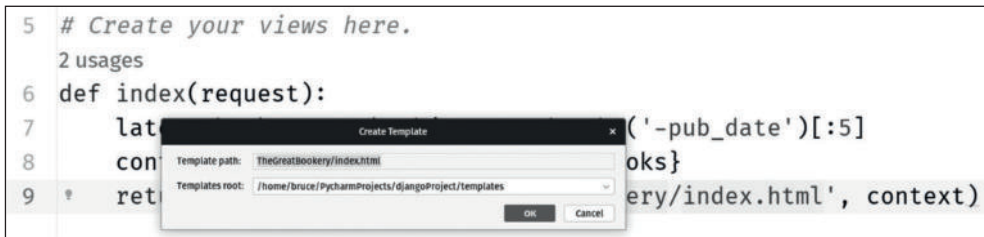


Рис. 10.19. Диалоговое окно, которое появляется при использовании встроенного действия¹ для создания отсутствующего шаблона, автоматически заполняется на основе некоторых очевидных предположений

Когда вы нажмете **ОК**, вам будет предложено создать в `Templates` папку под названием `TheGreatBookery`. В вашем проекте Django можно иметь несколько

¹ Встроенное действие – это действие, заданное для определенного анализа или информационной панели, но не сохраняемое с уникальным именем в каталоге. – Прим. ред.

приложений. Поскольку это так, можете разделить свои шаблоны. Позвольте PyCharm создать папку за вас. После этого вы получите пустой файл с именем `index.html`.

Это не книга по дизайну веб-страниц, поэтому мы даже не будем пытаться сделать ее красивой. Введите этот код на свою страницу `index.html`:

```
{% if latest_books %}
<ul>
    {% for book in latest_books %}
        <li>{{ book.title }} by {{ book.author }}</li>
    {% endfor %}
</ul>
{% else %}
    <p>No books available.</p>
{% endif %}
```

Это выглядит как код Jinja2, который мы видели в главе 8 с Flask. Очень похоже. Все, что делает этот код, – это проверяет, существует ли ключ словаря с именем `late_books` в переданном контексте данных. Если вы вернетесь и посмотрите на функцию `gender`, которую мы вызвали в представлении, она передает это.

Если ключ есть, мы отображаем содержимое, используя шаблоны синтаксиса цикла. Это не самый лучший пользовательский интерфейс. Он просто отображает неупорядоченный список, в котором каждая книга будет элементом списка. Если ключа там не было, мы визуализируем абзац с надписью **No books available**.

Что за странный значок Python в желобе шаблона?

Возможно, вы заметили большой, сочный значок Python в конце файла `index.html`. Посмотрите на рис. 18.20.

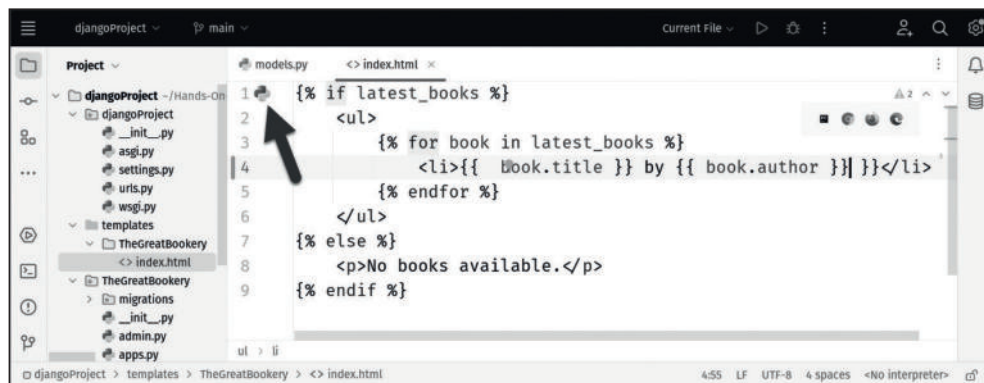


Рис. 10.20. Посмотрите на этот значок наверху, который так и просится, чтобы на него нажали!

Этот значок Python на самом деле называется *Pythicon*.

Хорошо, нет, это не так, я просто это выдумал. Тем не менее вам действительно хочется нажать на него, не так ли? Вперед, кликайте! Вы перейдете к коду Python в `views.py`, где мы его пробуждаем. Путешествие работает в обе

стороны. В `views.py` обратите внимание на более тонкий значок в желобе, где вызывается шаблон (см. рис. 18.21).

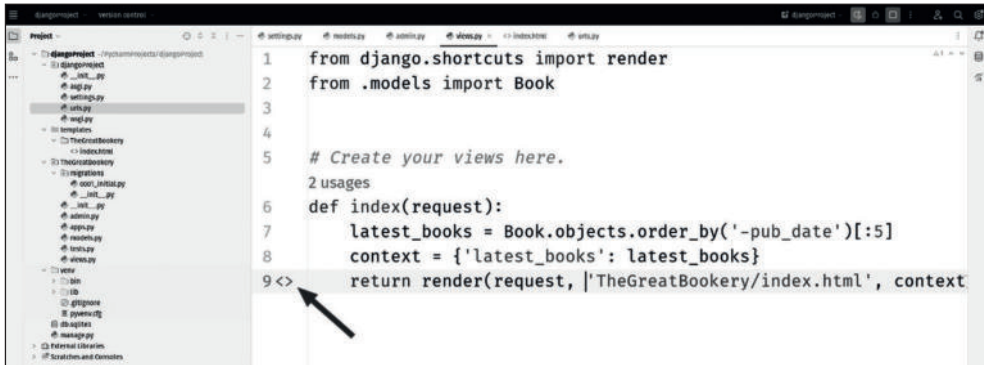


Рис. 10.21. Кликните значок HTML, чтобы перейти к шаблону

Опять же, просто кликните значок `html`, я имею в виду значок HTML, чтобы перейти к шаблону. Я бы не стал винить вас за то, что вы переключаетесь между ними и говорите: «*Получайте, vi u emacs!*»¹.

Запустим его!

Запустите или перезапустите сервер и проверьте результаты, как показано на рис. 18.22.

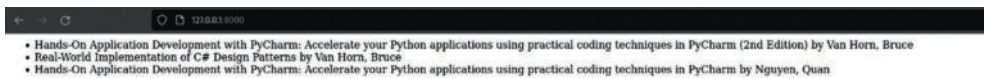


Рис. 10.22. Возможно, это лучший веб-сайт на свете!

Если вы направите свой браузер на `http://localhost:8000/`, вы должны увидеть либо несколько книг, либо сообщение о том, что книг нет. Я добавил несколько очевидных вариантов для чтения, и мой запуск выглядит просто великолепно!

Очевидно, что вы могли бы использовать инструменты HTML, которые мы изучили в главе 7, чтобы сделать это лучше, но нам еще есть над чем поработать, поэтому я оставляю это на ваше усмотрение.

Создание приложений PYRAMID с помощью PYCHARM

На данный момент мы рассмотрели три популярных веб-фреймворка. Остался один, который указан в меню **New Project** PyCharm: **Pyramid**. Подробную информацию о Pyramid можно найти на <https://trypyramid.com/>.

Организация, стоящая за Pyramid, позиционирует ее как структуру, которая позволяет вам начинать с малого, заканчивать большим и останавливаться на достигнутом. На мой взгляд, это может быть своего рода критикой его основ-

¹ Напоминание о «войне редакторов» – соперничестве между пользователями текстовых редакторов `vi` и `Emacs` (сейчас `vi` – это `Vim`, а в последнее время – `Neovim`). – *Прим. ред.*

ных конкурентов. Django критикуют за чрезмерную самоуверенность и излишество при разработке небольших проектов. Это соответствует идее «начни с малого» в Pyramid. У вас больше свободы в выборе компонентов для вашего приложения, и вы можете исключить то, что вам не нужно. Например, можно выбрать одну из нескольких библиотек ORM или просто отказаться от использования той или другой. Django же хочет, чтобы вы использовали их все.

Flask, напротив, не поставляется с ORM или чем-то еще, кроме Jinja2 для создания шаблонов. Flask критикуют за низкую производительность в больших приложениях. Концепция «играть по-крупному», кажется, отвечает на эту критику. Я не уверен, что именно критикует директива «останавливаться на достигнутом». Я думаю, что это нечто, чего мы все хотим.

После знакомства с первоклассными инструментами Django переход на Pyramid может показаться неутешительным. Django – хороший кандидат на использование специальных инструментов, поскольку он сложен и принимает за вас множество технических решений. В качестве примера можно рассмотреть панель задач `manage.py`. В `manage.py` есть множество команд, поэтому есть смысл иметь собственную панель. Помогает также и то, что Django пользуется бешеной популярностью. Pyramid – нет. Итак, с позиции PyCharm это гораздо более простое предложение с точки зрения инструментов.

Создание проекта Pyramid

Создание проекта Pyramid ничем не отличается от любого другого. Просто используйте **File | New Project** и выберите шаблон **Pyramid**, как показано на рис. 10.23.

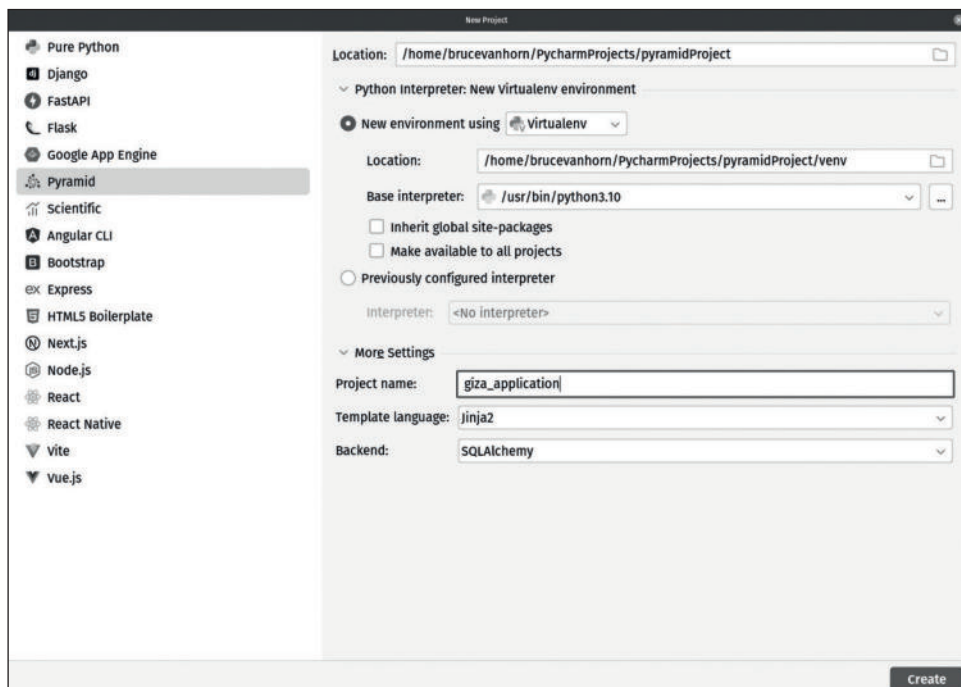


Рис. 10.23. PyCharm имеет шаблон проекта Pyramid

Как и в случае с Django, вам следует покрутить дополнительные настройки, как показано на рис. 10.23. Вы можете установить имя проекта, как это было в Django. Будьте осторожны с этой настройкой!

Название проекта Pyramid

Убедитесь, что имя проекта отличается от имени проекта PyCharm, введенного в поле **Location**.

При работе с Pyramid рекомендуется иметь имя проекта PyCharm, отличное от имени проекта, которое вы определяете в Pyramid. Такое разделение имен рекомендуется во избежание потенциальных конфликтов и путаницы между двумя системами с пространствами имен: PyCharm, IDE, и Pyramid, веб-платформой Python.

Вот почему желательно, чтобы они были разными:

- **конфликты пространств имен.** Если имя вашего проекта PyCharm точно совпадает с именем проекта Pyramid, вы можете столкнуться с конфликтами имен или конфликтами пространств имен. Эти конфликты могут затруднить PyCharm различение между настройками, конфигурациями и файлами, специфичными для проекта, и файлами и конфигурациями, специфичными для платформы Pyramid. Это может привести к путанице и потенциальным ошибкам в процессе разработки;
- **настройки проекта PyCharm.** PyCharm использует имя проекта для своих внутренних настроек и конфигураций, включая виртуальные среды, анализ кода и настройки, специфичные для проекта. Если имя вашего проекта PyCharm совпадает с именем проекта Pyramid, PyCharm может перезаписать его или помешать конфигурациям и настройкам, специфичным для Pyramid;
- **ясность и удобство сопровождения.** Сохранение разных названий для вашего проекта PyCharm и проекта Pyramid помогает поддерживать ясность и организованность в вашей среде разработки. Это облегчает понимание того, какие аспекты вашей разработки контролируются PyCharm, а какие специфичны для вашего веб-приложения Pyramid;
- **гибкость.** Наличие разных имен обеспечивает гибкость, особенно если вы работаете с несколькими проектами с помощью Pyramid или используете PyCharm для других проектов Python. Это предотвращает потенциальные конфликты при переключении между проектами.

Я собираюсь назвать свой воображаемый проект `giza_application`, намекая на место расположения Великих пирамид Египта.

Задав имя проекта, вы можете выбрать один из двух шаблонизаторов: **Jinja2** и **Chameleon**. Наконец, у вас есть настройки для ORM, который здесь указан как **Backend**. Есть два варианта: **SQLAlchemy**, который очень популярен, и **ZODB**, который несколько экзотичен.

SQLAlchemy – это простой ORM, используемый для работы с реляционными базами данных. Вы создаете модели, как мы это делали с ORM Django,

и в процессе миграции можете применить изменения схемы базы данных к новой или существующей базе данных.

ZODB относится к базе данных Zope. Вообще говоря, Zope – это сервер веб-приложений с открытым исходным кодом, написанный на Python. Проще говоря, он используется как система управления контентом, аналогичная WordPress. Компонент базы данных ZODB имеет некоторые интересные особенности, отличающие его от типичных реляционных баз данных. Это мощный инструмент, который можно использовать в сочетании с веб-платформой Pyramid для управления данными более характерным для Pythonic способом. Вот как ZODB интегрируется в проект Pyramid и некоторые его особенности:

- **оригинальный Python.** ZODB позволяет хранить объекты Python без необходимости их сериализации в другой формат. Вы можете работать со сложными структурами данных напрямую, без необходимости маппинга их со схемой реляционной базы данных;
- **ACID-транзакции.** ZODB поддерживает транзакции **атомарности, согласованности, изоляции и долговечности (ACID)**. Это гарантирует, что ваши данные останутся согласованными, даже если во время транзакции что-то пойдет не так;
- **интеграция с Pyramid.** ZODB можно легко интегрировать с Pyramid с помощью пакета `Pyramid_zodbconn`. Это обеспечивает простой способ получения соединения ZODB при обработке запросов Pyramid;
- **иерархическое хранилище.** ZODB позволяет организовать данные в древовидную структуру. Это может быть особенно полезно в приложении Pyramid, использующем обход, где структура URL-адресов часто отражает иерархическую структуру данных;
- **масштабируемость.** ZODB можно масштабировать на несколько компьютеров, что позволяет создавать более обширные и сложные приложения;
- **управление версиями.** ZODB поддерживает управление версиями объектов, что позволяет отслеживать изменения объектов с течением времени. Это может быть полезно для реализации таких функций, как отмена или повтор действия;
- **запросы.** Хотя ZODB не предоставляет язык запросов, такой как SQL, он предлагает способы индексации и поиска объектов с использованием инструментов, включая модуль `BTrees` или сторонние пакеты, такие как `repoze.catalog`;
- **схема не требуется.** В отличие от реляционных баз данных ZODB не требует фиксированной схемы, что обеспечивает гибкость моделирования данных. Вы можете изменить классы Python, используемые для хранения, без необходимости переноса данных;
- **поддержка Blob.** ZODB поддерживает хранение больших двоичных объектов (BLOB), таких как изображения или видео, наряду с обычными объектами;
- **персистентность.** ZODB предоставляет простую модель персистентности. Любые изменения в постоянных объектах внутри транзакции автоматически сохраняются в базе данных при фиксации транзакции;
- **совместимость.** ZODB хорошо работает с различными серверами WSGI и может быть интегрирован в приложение Pyramid, работающее на разных платформах.

ZODB может стать убедительным выбором для разработчиков Pyramid, которым нужна база данных, близко соответствующая объектно ориентированной парадигме Python. Его интеграция в проект Pyramid позволяет интуитивно управлять данными без необходимости сложных SQL-запросов или маппинга ORM, что делает его привлекательным вариантом для определенных типов приложений. Если иерархическая объектно ориентированная природа ZODB соответствует модели данных вашего приложения Pyramid, это может быть отличным выбором. Однако с точки зрения PyCharm для этой базы данных не существует специального инструмента.

Нажатие кнопки **ОК** создаст структуру проекта и конфигурацию запуска.

Мне бы очень хотелось, чтобы здесь было что сказать, но, по правде говоря, кроме Django, любая другая структура, которая могла бы быть рассмотрена последней, имела бы ту же проблему. Мы уже рассмотрели инструменты PyCharm для разработки веб-приложений в простой среде. Мы видели поддержку шаблонов Jinja2, работу с ORM и, очевидно, общую работу над проектами Python.

КРАТКОЕ СОДЕРЖАНИЕ

В этой главе мы рассмотрели различные функции PyCharm, касающиеся поддержки и автоматизации задач в процессе веб-разработки с помощью Django. Хотя этот список функций никоим образом не является исчерпывающим, я надеюсь, что он может стать надежной отправной точкой для дальнейшего открытия других мощных функций для вашего процесса веб-разработки.

Во-первых, мы видим, что если указать тип проекта PyCharm как Django, будет создан обширный скелет проекта с уже заполненным удобным шаблонным кодом. Благодаря реализации панели `manage.py` внутри окна проекта, а также ее конфигурации запуска/отладки PyCharm дополнительно обеспечивает более высокий уровень разработки с различными задачами, традиционно выполняемыми через командную строку, такими как запуск сервера или выполнение миграции. Наконец, признавая интегрированные представления и шаблоны в Django, PyCharm максимально упрощает для разработчиков работу с ними в редакторе – будь то создание отсутствующего шаблона, завершение кода даже в HTML и Jinja или даже динамическое переключение между представлениями и шаблонами.

Мы закончили кратким обзором Pyramid. Pyramid – это фреймворк, который стремится быть более гибким, чем Django, но содержит больше функций, чем Flask. Это золотая середина между ними. К несчастью для Pyramid, мы рассмотрели его в последнюю очередь. PyCharm не имеет каких-либо специальных инструментов для Pyramid, кроме шаблона проекта, который создает структуру проекта. Честно говоря, если бы Flask был последним, его постигла бы та же участь, потому что большинство инструментов веб-разработки PyCharm полезны независимо от того, какую платформу вы выберете. Pyramid обладает некоторыми замечательными функциями и достоин рассмотрения для применения в любом проекте.

В следующей главе мы изучим последний важный компонент любого веб-приложения: базу данных. Пристегните ремни, потому что инструменты для работы с базами данных в PyCharm обширны!

Вопросы

1. Каковы основные характеристики Django и чем они отличают Django от другого популярного веб-фреймворка Python – Flask?
2. Какова роль панели PyCharm `manage.py` в проекте Django и как ее открыть и использовать?
3. Какова роль интерфейса администратора Django? Как создать экземпляр модели (т. е. новую запись в таблице базы данных) в этом интерфейсе? Как изменится процесс, если модель ссылается на другую модель?
4. Какова цель конфигурации запуска и отладки в PyCharm в контексте запуска сервера Django?
5. Применяется ли логика завершения кода PyCharm только к коду Python в проектах Django?
6. Каково значение возможности переключения между представлениями Django и соответствующими шаблонами в PyCharm?
7. Описать инструменты PyCharm, доступные для платформы Pyramid.

ДАЛЬНЕЙШЕЕ ЧТЕНИЕ

Чтобы узнать больше о том, что было рассмотрено в этой главе, посетите следующие ресурсы:

- *Web Development with Django, Second Edition*;
- *Hands-On RESTful Python Web Services, Second Edition*.

Понимание управления базами данных в PyCharm

Что общего между мечехвостами, целакантами, крокодилами и реляционными базами данных? Я подожду, пока вы будете гуглить *целаканта*. Все четыре из них существуют уже миллионы лет, но эволюционировали очень мало. Хорошо, баз данных не существует уже миллионы лет, но они существуют уже миллионы лет в интернете. Все знают, что интернет-годы очень коротки. Разработчики JavaScript часто шутят, что во время вашего обеда во множестве точек мира были изобретены десятки новых фреймворков, которые приобрели известность, впали в немилость, а затем были забыты, прежде чем вы успели доесть свою лапшу.

В начале 1970-х годов исследователь по имени Э. Ф. Кодд работал в Исследовательской лаборатории IBM в Сан-Хосе в Калифорнии. Он разработал революционную концепцию, названную **реляционной моделью данных**. В своей основополагающей статье под названием «Реляционная модель данных для больших общих банков данных», опубликованной в 1970 году, Кодд изложил принципы и основы этого нового подхода к организации и извлечению данных.

Реляционная модель Кодда предложила способ представления данных в виде набора таблиц, известных как отношения, где каждая таблица состоит из строк и столбцов. Он представил концепцию **реляционной алгебры** – математической основы для манипулирования и запроса данных в этих таблицах. В модели особое внимание уделялось использованию **теории множеств** и математической логики для определения связей и выполнения операций с данными.

Идеи Кодда бросили вызов преобладавшим в то время иерархическим и сетевым моделям баз данных, которые были более сложными и менее гибкими. Реляционная модель предлагала более простой и интуитивно понятный способ управления данными, обеспечивая основу для **языка структурированных запросов (SQL)** и других инструментов, используемых в реляционных базах данных.

В 1974 году IBM выпустила первую коммерчески доступную **систему управления реляционными базами данных (RDBMS)**, названную *System R*, основанную на работе Кодда. Система R реализовала многие концепции, изложенные в исследованиях Кодда, и стала влиятельным предшественником

последующих систем реляционных баз данных, но этот продукт не был включен в кампанию агрессивного маркетинга, поскольку руководители IBM были обеспокоены снижением продаж других систем баз данных, продаваемых компанией. В это время родился еще один стартап по производству баз данных: **Oracle**. Первоначальную версию Oracle разработали Ларри Эллисон, Боб Майнер и Эд Оутс. Выпущенная в 1983 году, система Oracle считается первым коммерческим успехом, хотя у IBM и было серьезное преимущество на старте.

Технология реляционных баз данных развивалась на протяжении 80-х и 90-х годов и со временем стала основой корпоративных ИТ во всех отраслях во всех уголках земного шара. В 1986 году язык SQL стал стандартизирован. Стандарт развивался с течением времени, но, честно говоря, спустя 49 лет после изобретения SQL большая часть разработки ведется с использованием самого простого и старого набора языковых операторов.

Минуту назад я заявил, что идеи Кодда бросили вызов преобладавшим в то время иерархическим и сетевым моделям баз данных, которые были более сложными и менее гибкими. Интересно отметить, что в начале 2000-х годов произошел переход от технологии реляционных баз данных к тому, что мы называем **базами данных NoSQL**. Мир информационных технологий часто действует как маятник, постоянно раскачиваясь назад и вперед. Раньше у нас были относительно маломощные ПК, способные воспроизводить видеофайлы со скоростью 12 кадров в секунду в окне размером с почтовую марку. Технологии улучшились до такой степени, что мы можем смотреть на них видео высокой четкости с разрешением 4K на полной скорости, но затем мы изобрели небольшие портативные устройства, такие как iPod, и вернулись к низкоуровневой обработке и прерывистому, зернистому видео. В конце концов они превратились в iPhone, который может воспроизводить видео с высоким разрешением и высокой частотой кадров. Некоторые iPhone даже с большим разрешением, чем телевизор, который был у меня в колледже.

Аналогично и с базами данных: когда-то мир использовал иерархические и сетевые базы данных. На следующий день все перешло к новым, называемым *реляционными*. Сегодня мы видим поворот в другом направлении. Сегодня все больше и больше проектов отдают предпочтение нереляционным базам данных, многие из которых поддерживают иерархические данные. Независимо от того, какую технологию вы предпочитаете, можно с уверенностью сказать, что практически любой проект, который вы создаете, особенно в мире корпоративных ИТ, будет в значительной степени взаимодействовать с какой-либо базой данных.

В 2015 году JetBrains создала новый продукт, призванный стать популярной IDE для разработчиков баз данных. **DataGrip** был создан, чтобы предоставить унифицированный интерфейс и надежный набор инструментов для работы с различными базами данных. Он предлагает такие функции, как интеллектуальное завершение кода, расширенные возможности редактирования SQL, навигацию по схеме, анализ данных и интеграцию с системами контроля версий. Как и продукт WebStorm от JetBrains, продукт DataGrip интегрирован в профессиональную версию PyCharm через систему плагинов. Вложите 99 долл. США в PyCharm Professional, и вы получите IDE Python, IDE JavaScript, IDE для работы с веб-интерфейсом в HTML и CSS, а теперь и полноценную IDE для работы с базами данных, и вы поймете, что ваша сделка была удачной!

В этой главе мы рассмотрим функции PyCharm, связанные с базами данных. К концу главы вы узнаете следующее:

- немного истории баз данных и немного основ, просто чтобы убедиться, что мы все находимся на одной волне с точки зрения терминологии. Если вы какое-то время занимаетесь разработкой программного обеспечения, это будет просто обзор. Если вы новичок, я постараюсь предоставить вам наилучшее введение в технологию баз данных;
- как перейти к инструментам базы данных в PyCharm, спрятанным на вкладке в правой части интерфейса. Я встречал многих разработчиков, которые использовали PyCharm в течение многих лет, но даже не знали, что эти инструменты вообще существуют;
- как подключаться к разным базам данных, в том числе как добавлять необходимые драйверы подключения в PyCharm. JetBrains сделал это очень просто!
- как настроить диалекты SQL для отдельных проектов, а также глобально;
- как использовать шаблоны генерации SQL для более быстрого написания SQL-запросов;
- как создать **entity relationship diagram (ERD)**;
- как использовать графические дизайнеры для простого создания таблиц;
- как использовать консоли для создания и выполнения специальных запросов к любой базе данных.

От себя лично: помимо обучения программированию на BASIC, когда мне было 12 лет, реляционные технологии были первым навыком, который я освоил, когда пришел в сферу IT более 30 лет назад. Этот набор навыков всегда был востребован и, вероятно, будет востребован еще долгие годы. У меня большой опыт в этой теме, и я рад поделиться этим опытом с вами в этой главе. Итак, приступим!

ТЕХНИЧЕСКИЕ ТРЕБОВАНИЯ

Чтобы продолжить работу с этой главой, как и с остальной частью книги, вам понадобится следующее:

- установленный и работающий интерпретатор Python. Я буду использовать последнюю версию с <https://python.org>;
- установленная и рабочая копия PyCharm. Установка была описана в главе 2;
- в этой главе полезно иметь сервер базы данных, чтобы вы могли попрактиковаться на реальной базе данных. На выбор доступны десятки популярных реляционных баз данных, поэтому охватить их все нецелесообразно. Я буду использовать **MySQL** с помощью **Docker Desktop**. Если вы планируете следовать инструкциям, потребуется установить Docker Desktop на ваш компьютер. Инструкции по установке можно найти по адресу <https://www.docker.com/products/docker-desktop/>;
- пример исходного кода этой книги взят с GitHub. Мы рассмотрели клонирование кода в главе 2. Соответствующий код этой главы вы найдете по адресу <https://github.com/PacktPublishing/Hands-On-Application-Development-with-PyCharm---Second-Edition/tree/main/chapter-11>.

Оболочки реляционных баз данных

Идея реляционных данных, предложенная Э. Ф. Коддом, основана на нескольких простых принципах. Во-первых, данные могут быть представлены в наборах, называемых *таблицами*. Таблица состоит из **строк** и **столбцов**. Например, если бы мы хотели создать базу данных, которая будет использоваться книжным интернет-магазином, первая таблица, о которой мы могли бы подумать, – это таблица под названием `books`. Таблица `books` будет содержать столбцы, определяющие данные в этих столбцах, – возможно, что-то вроде этого:

Заголовок	ISBN	Количество страниц	Автор	Цена
Искусство войны	1599869772	68	Сунь Цзы	4,99
Книга пяти колец	8387743849	43	Миямото Мусаси	4,50

В этой таблице пять столбцов, предназначенных для структурирования данных о книгах. У нас есть две книги, которые хранятся в виде строк в таблице.

Язык структурированных запросов

Статья Э. Ф. Кодда, определяющая реляционную алгебру, послужила основой для SQL. SQL не похож ни на один другой язык программирования, который вы когда-либо использовали, поскольку это один из немногих языков, использующих декларативную парадигму. Большинство языков, которые вы используете, включая Python, используют императивную парадигму. Короче говоря, язык служит синтаксической основой, дающей компьютеру инструкции о том, что вы от него хотите. По сути, вы микроменеджер. Вы указываете каждый ввод, каждый вывод и каждый шаг, который программа предпримет во время обработки, преобразуя ввод в вывод. Вы должны быть дотошны, так как компьютер поймет вас буквально. Если вы упустите хотя бы одну деталь, все развалится.

Декларативное программирование предполагает простое указание желаемого результата из неявного ввода. У вас практически нет контроля над операциями, выполняемыми для получения выходных данных из входных. Рассмотрим следующий оператор SQL, предназначенный для получения некоторых строк из базы данных вашего книжного магазина:

```
SELECT title AS Title, isbn AS ISBN, author AS Author, price AS Price FROM
books ORDER BY price DESC
```

Этот запрос выдаст некоторый результат, состоящий из таблицы, содержащей несколько столбцов из исходной базы данных, которая служит неявным входом. Вы получите следующие столбцы:

```
Title
ISBN
Author
Price
```

Но вы не получите количество страниц, потому что этого не было в запросе. Источник входных данных подразумевается в разделе запроса FROM book. В конце у нас есть запрос на упорядочивание результатов по цене в порядке убывания (DESC), благодаря чему наш результирующий табличный список книг будет выглядеть от самых дорогих к самым дешевым.

Две половины SQL

Сам SQL разделен на два отдельных набора синтаксиса: **язык определения данных (DDL)** и **язык манипулирования данными (DML)**. DDL используется для определения структуры базы данных. DML используется для запроса базы данных и выполнения четырех основных операций **создания, чтения, обновления и удаления (CRUD)**, о которых мы упоминали в предыдущих главах. Представленный ранее оператор SELECT является примером операции чтения. В SQL есть ключевые слова для операций INSERT, UPDATE и DELETE для записей, которые просты, поэтому я хочу сосредоточиться на DDL, поскольку с ним будет связана большая часть вашей работы в PyCharm.

В нашем предыдущем примере таблицы, которая будет использоваться в приложении книжного магазина, была одна таблица под названием books. Эта таблица содержала набор столбцов, которые мы определили с помощью DDL следующим образом:

```
CREATE TABLE books (  
    title VARCHAR(255),  
    isbn VARCHAR(255) UNIQUE,  
    price DECIMAL(10, 2),  
    author VARCHAR(255),  
    page_count INT  
);
```

Если вы новичок в SQL, то могли заметить, что ключевые слова пишутся с заглавной буквы. На самом деле я готов поспорить, вы правильно предположили, что слова, написанные с заглавной буквы, являются ключевыми. Хотя использование ключевых слов с заглавной буквы не является обязательным, это хорошая практика, особенно если руководитель базы данных вашего проекта – крупный мужчина в отставке из Корпуса морской пехоты США, который очень внимательно относится к деталям синтаксиса в *своей* базе данных! В этом случае передовая практика может стать вопросом выживания.

Большую часть того, что вы читаете в DDL-таблицах, легко понять. Мы создаем таблицу под названием books. Мы уже знаем об именах столбцов. SQL использует строгую систему типов. Это означает, что вам необходимо определить тип данных, которые будут помещены в столбец, и нарушение этого может повлечь за собой последствия. Я уже много раз говорил, что задача разработчика – защитить программу от перехода в недопустимое состояние. Аналогичным образом задача разработчика базы данных, а также **администратора базы данных (DBA)** – ограничить возможность программы вводить недопустимые данные в базу данных. Первая линия защиты – это система типов. Что смущает или даже пугает разработчиков – так это разные названия типов. Глядя на наш предыдущий список, вы можете догадаться, что VARCHAR относится к набору символов пе-

ременной длины. Программисты называют это *строкой*. Число (255) после типа относится к максимальной длине VARCHAR. Все поля VARCHAR (строковые) в базе данных имеют длину не более 255 символов. Число 255 очень распространено, потому что многие из нас, старожилов, выросли в мире 8-битных вычислений. Число 255, являющееся максимальным значением 8-битного целого числа без знака, было общим максимумом длины поля. Мы устанавливали это значение, когда не были уверены, какой длины могут быть данные.

Естественно, сегодня мы обычно работаем в 32- и даже 64-битных архитектурах, поэтому максимум может быть гораздо выше. Ограничиваться старым максимумом – это нормально, потому что 255 символов обычно достаточно для большинства ситуаций. Указание разумных максимальных значений помогает поддерживать разумные требования к хранилищу базы данных и памяти.

Посмотрите на поле price. Оно указано как тип DECIMAL, который, очевидно, похож на число с плавающей запятой. Системы баз данных имеют разные имена для своих типов, поэтому вам всегда следует обращаться к документации вашей системы баз данных, чтобы получить точное имя. DECIMAL действительно является числом с плавающей запятой, но мы также указали уровень точности с помощью (10,2). Это означает, что у нас может быть 10-значное число с двумя десятичными знаками. Обычно это используется для указания цен. Некоторые системы баз данных имеют типы, специально предназначенные для валюты. SQL Server является примером этого со своими типами MONEY и SMALLMONEY. Еще раз проверьте документацию вашей системы баз данных на наличие соответствующих типов, поскольку эти специализированные типы не являются частью стандартного SQL.

Взаимосвязи

Учтите, что наш книжный магазин естественным образом будет расти в размерах и сложности. Допустим, мы добавили несколько разных полей и удалили некоторые, которые не использовали. Наша новая структура таблицы выглядит следующим образом:

Заголовок	Имя автора	Электронная почта автора	Страницы	Цена
Язык программирования C, 2-е издание	Брайан Керниган и Деннис Ритчи	bkernigan@notrealaddress.com	272	53,60
Практическая разработка шаблонов проектирования C#	Брюс Ван Хорн	bvanhorn@notrealaddress.com	442	44,99
Практическая разработка приложений с помощью PyCharm	Куан Нгуен	qnguyen@notrealaddress.com	785	35,45
Практическая разработка приложений с помощью PyCharm, 2-е издание	Брюс Ван Хорн	bvanhorn@notrealaddress.com	840	44,99

Помимо сложности, связанной с большим количеством записей, мы добавили некоторую сложность с точки зрения разработки. В какой-то момент нам стало важно отслеживать больше информации об авторах той или иной книги. Мы добавили поля для имени и адреса электронной почты, что, поначалу казалось, решило наши проблемы, но в конечном итоге привело к новым проблемам.

У книги по языку С два автора, но у нас только одно поле. Хотя, возможно, было бы неплохо сохранить обоих авторов, как мы это сделали, но есть также поле для адреса электронной почты, и оно может содержать только один адрес. Это не лучшая ситуация, поскольку наша цель – отправка авторам отчетов о гонорах или отчетов о продажах. Один адрес – это проблема.

Адреса электронной почты будут дублироваться, если мы храним две книги одного и того же автора. Если автор меняет адреса электронной почты, все записи необходимо обновить. Это эквивалент жестко закодированных значений, хранящихся в нескольких местах программы. Вы должны не забыть изменить его во многих местах. Проблема усугубляется, если данные адреса электронной почты дублируются в нескольких разных таблицах.

Решение этой проблемы предписывается SQL и идеями реляционной алгебры. Вам нужен единый источник достоверных данных автора вместо смешения их с другими таблицами. Итак, создаем новую таблицу для авторов:

```
CREATE TABLE authors (  
    id SERIAL PRIMARY KEY,  
    first_name VARCHAR(255),  
    last_name VARCHAR(255),  
    email_address VARCHAR(255)  
);
```

В этой таблице есть поля для имени и фамилии, а также адреса электронной почты. В ней также есть поле `id`, передающееся на сервер в качестве первичного ключа. Первичный ключ – это поле, которое может однозначно идентифицировать информацию в этой строке. Идея здесь в том, что каждый автор получает только одну строку для хранения своей информации. Нам нужен некоторый фрагмент данных, чтобы однозначно идентифицировать эту строку. Рассмотрим записи обо мне, например. Мои имя и фамилия не позволяют однозначно определить мою запись в базе данных. Я знаю как минимум еще трех человек по имени Брюс Ван Хорн. Одним из них был мой отец, публиковавшийся в области медицины. Один из них – мотивационный оратор, публикующий книги в этой литературной области. Я нашел еще одного Брюса Ван Хорна в LinkedIn, и – хотите верить, хотите нет – он тоже разработчик программного обеспечения! Таким образом, имя – плохой выбор в качестве уникального идентификатора.

Адреса электронной почты могут подойти, за исключением того, что мы уже выяснили, что адреса электронной почты могут меняться. У меня не менее пяти адресов электронной почты, и по крайней мере один из них настолько завален спамом, что я его даже не проверяю. Электронная почта не решение проблемы.

Лучше всего использовать некоторый фрагмент данных, который является уникальным, но произвольным и не имеет никакого отношения к остальным данным в записи. Есть два способа сделать это. Наиболее распространенным является использование **последовательности** базы данных. Последователь-

ность (sequence) – это автоматически увеличивающийся источник целых чисел. Каждый раз, когда запись вставляется в таблицу, секвенсор генерирует порядковый номер, который гарантированно уникален.

Другой подход – использовать **универсально уникальные идентификаторы (UUID)**. В этом подходе используется алгоритм, который представляет комбинацию данных, составляющих уникальный идентификатор. Например, вы можете взять серийный номер процессора пользователя, объединить его с текущей отметкой даты на компьютере и, возможно, добавить некоторую информацию о серийном номере BIOS компьютера или, допустим, IP-адресе или MAC-адресе пользователя при подключении к сети. Если принять во внимание множество факторов, один из которых связан со временем, вероятность того, что два компьютера сгенерируют один и тот же UUID, практически равна нулю. С UUID работать немного сложнее, чем с простыми целыми числами. UUID может выглядеть так: 6f35e0e7-d99a-4437b894-f73ff35bd3ad по сравнению с записью со значением id 16. Что бы вы предпочли ввести в свои запросы?

Теперь, когда у нас есть поле, однозначно идентифицирующее запись автора, мы можем настроить структуру таблицы книг следующим образом:

Заголовок	Страницы	Цена	id_автора
Практическая разработка шаблонов проектирования C#	442	44,99	2
Практическая разработка приложений с помощью PyCharm	785	35,45	1
Практическая разработка приложений с помощью PyCharm, 2-е издание	840	44,99	2

Наша таблица authors выглядит так:

id_автора	Имя	Фамилия	Электронная почта
1	Цюань	Нгуен	qnugyen@notrealaddress.com
2	Брюс	Ван Хорн	bvanhorn@notrealaddress.com

Идентификатор автора в таблице authors используется как связанный столбец в таблице books с использованием связи «один ко многим». У каждого автора будет одна запись, которая связана со многими записями в таблице books.

Больше реляционных структур

Мы решили большую проблему, используя нашу структуру таблицы «один ко многим». У нас больше нет повторяющихся данных в нескольких записях наших авторов. Однако мы не решили всех наших проблем. Например, у книги по языку C несколько авторов. Как мы можем хранить авторов таким образом, чтобы каждая книга могла поддерживать нескольких авторов?

Боюсь, мы дошли до того момента, когда я скажу вам, что это не книга по SQL и не по реляционной теории. Если я вас зацепил, то могу порекомендовать несколько отличных книг, которые я использовал, когда изучал эти вещи.

Мы представили достаточно словарного запаса реляционных баз данных, чтобы помочь вам понять, что вы увидите в инструментах PyCharm, и это была моя настоящая цель.

Поскольку я не полный дегенерат, несмотря на то что вы можете прочитать обо мне на Stack Overflow, я дам вам решение быстро и без многих страниц объяснений.

Можете решить эту проблему, используя то, что мы называем таблицей маппинга. Ваша структура потребует добавления таблицы `book_authors_map`:

Id	id_книги	id_книги
1	1	1
2	1	2

Ваши таблицы `books` будет выглядеть так:

id_книги	Заголовок	Страницы	Цена
1	Язык программирования Си	442	44,99

Мы убрали поле `author_id` и добавили поле `book_id`, которое действительно должно было быть там с самого начала. Таблица `authors` будет выглядеть так же:

id_автора	Имя	Фамилия	Электронная почта
1	Брайан	Керниган	bkernigan@notrealaddress.com
2	Деннис	Ричи	dritche@noterealaddress.com

Таблица маппинга может отображать **взаимосвязи «многие ко многим»** между `books` и `books`! Если один из авторов выйдет в одиночку и опубликует книгу без своего соавтора, это сработает. Запись таблицы `books` будет иметь одну связанную запись на карте, которая имеет одну связанную запись в таблице `books`.

Если у книги 10 авторов, у нее будет одна запись книги, 10 записей авторов и 10 записей маппинга. Реляционная алгебра – это круто! Это недооцененный навык в современном мире **объектно-реляционных преобразователей (object relational mappers, ORM)**, которые абстрагируют все это в обычные объектные структуры. Сегодня разработчики склонны забывать этот навык. Если вы его лишитесь, то много потеряете. Ведь вы можете внести изменения в свой DDL, которые приведут к огромному увеличению производительности вашего приложения.

Говоря об этом, давайте узнаем об инструментах PyCharm для разработки баз данных.

В терминологии базы данных используются простые английские формы множественного числа

Когда мы пройдемся по терминологии, используемой при разговоре о базах данных, я чувствую себя обязанным указать на то, что беспокоило меня очень дол-

гое время. Многие термины, используемые при этом, происходят от латинских основ. Например, **индекс** – это дополнение к таблице, которое может ускорить поиск данных за счет скорости вставки новых данных. Это латинское слово, обозначающее *указатель, индикатор* или *отметка*. **Схема** – это способ разделения таблиц и других структур в базе данных. Слово «схема» произошло от греческого слова *σχῆμα* (*схема*) и на обоих языках означает *форму, фигуру* или *контур*.

Говоря о формах множественного числа, слова могут быть на самом деле не такими, как вы подумали бы, если бы получили классическое образование, как я. Если вы приехали из страны, где ваш язык основан на латыни, вы также смогли бы увидеть эту проблему.

Я знаю, что множественное число слова *schema* – *schemata*, а *index* во множественном числе будет *indices*. Если вы тоже думаете так, то будете разочарованы. В отрасли стандартизированы простые формы множественного числа, такие как *schemas* и *indexes*.

ИНСТРУМЕНТАРИЙ БАЗ ДАННЫХ В PYCHARM

Инструментарий базы данных в PyCharm является полным, но слишком общим. Под этим я имею в виду, что PyCharm пытается поддерживать каждую существующую базу данных и поэтому обычно поддерживает общие для всех функции. Иногда имеет смысл использовать более конкретные инструменты, такие как **SQL Server Management Studio (SSMS)** для SQL Server. Однако для общей разработки инструментов PyCharm более чем достаточно. Отправной точкой для работы с базами данных в PyCharm является открытие инструментов базы данных и создание соединения. Для этого вам также понадобится база данных. PyCharm поддерживает десятки самых популярных серверов баз данных. Я не могу предсказать, какой из них вы предпочтете, поэтому собираюсь использовать тот, который хорошо знаю: MySQL. Независимо от ваших предпочтений инструменты PyCharm являются универсальными, поэтому, пока вы выбираете реляционную базу данных, соответствующую стандартам, процессы будут одинаковыми.

НАСТРОЙКА СЕРВЕРА БАЗЫ ДАННЫХ MySQL с помощью Docker

Самый простой способ опробовать любую систему баз данных или вообще любую серверную технологию – использовать Docker. Далее в книге у меня есть дополнительные планы относительно Docker, и я буду использовать Docker Desktop вместе с командной строкой. Мне нравится графический интерфейс настольного компьютера, позволяющий видеть, что происходит в графическом виде, но вам следует освоить навыки работы с командной строкой Docker, чтобы оставаться конкурентоспособными. Если у вас не установлен Docker Desktop, инструкции по установке можно найти по адресу <https://www.docker.com/products/docker-desktop/>. Естественно, есть и другие варианты, например установка сервера базы данных на ваш компьютер. Лично я против этого, потому что серверы баз данных очень сложны. Уста-

новка чего-либо, например SQL Server или Oracle, приведет к изменениям в вашей ОС на таком уровне, что эти программные пакеты будет трудно удалить. Раньше существовало эмпирическое правило: если вы допустили какие-либо ошибки при установке сервера базы данных, самым разумным вариантом было бы снести ОС и установить ее заново. Я почти уверен, что это уже не так, но я по-прежнему отношусь к серверам баз данных с уважением, поскольку из всех движущихся частей вашего решения эта, пожалуй, самая сложная. Последнее, что вам нужно, – это упавший сервер базы данных на вашем ноутбуке в процессе разработки эпического проекта. Итак, я рекомендую Docker. Если что-то пойдет не так, вы удалите контейнер и создадите новый.

Аналогичным образом можно создать **виртуальную машину (VM)** с помощью таких продуктов, как **VMware Workstation** или **Oracle VirtualBox**. Это еще один прекрасный способ работы, хотя он требует больше места и ресурсов, чем Docker, и вы должны не забывать поддерживать свои виртуальные машины в состоянии, соответствующем последним версиям.

Еще один хороший вариант – разместить выбранный сервер базы данных в вашем любимом облаке. Я рекомендую для этого **DigitalOcean**, поскольку его цена невелика и настройка чрезвычайно проста. Я использую этот сервис для размещения сопутствующего веб-сайта этой книги. Если ваш компьютер не предназначен для запуска сервера базы данных VMware или Docker, лучшим вариантом будет использование облачного провайдера.

Все эти варианты хороши, но мне нужно выбрать один, поэтому я буду работать с Docker. Помните, это не книга о Docker. Мое освещение будет не очень пространным. Ваша цель – запустить сервер базы данных, чтобы вы могли попрактиковаться. Если сможете сделать это с помощью чего-то помимо Docker, переходите к следующему разделу.

Установка и запуск контейнера MySQL

Я предполагаю, что на вашем компьютере работает Docker и что ваши команды Docker доступны в вашем PATH. Мы можем подтвердить это, открыв наш терминал и введя следующую команду:

```
docker ps -a
```

Эта команда выведет список всех контейнеров, работающих в данный момент или остановленных. Если вы только что установили Docker, то должны увидеть пустой список, т. е. вообще ничего. На самом деле проверка предназначена для того, чтобы убедиться, что команда выполняется и не выдает никаких ошибок. Если это не так, вы готовы получить MySQL с помощью этой команды:

```
docker pull mysql
```

Вы увидите красиво анимированное изображение установки, как показано на рис. 11.1.

```

root@photon-bd5886da020 [ ~ ]# docker pull mysql
Using default tag: latest
latest: Pulling from library/mysql
e2c03c89dcad: Pull complete
68eb43837bf8: Pull complete
796892cdf5ac: Pull complete
6bca45eb31e1: Pull complete
ebb53bc0dcca: Pull complete
2e2c6bdc7a40: Pull complete
6f27b5c76970: Pull complete
438533a24810: Pull complete
e5bdf19985e0: Pull complete
667fa148337b: Pull complete
5baa702110e4: Pull complete
Digest: sha256:232936eb036d444045da2b87a90d48241c60b68b376caf509051cbbcffea6fdc
Status: Downloaded newer image for mysql:latest
docker.io/library/mysql:latest
root@photon-bd5886da020 [ ~ ]#

```

Рис. 11.1. Команда Docker, используемая для получения изображений, необходимых для MySQL

Эта команда собрала все требования, необходимые для запуска одного или нескольких контейнеров MySQL. Далее нам нужно создать и запустить контейнер с помощью команды:

```
docker run --name pycharm-mysql -e MYSQL_ROOT_PASSWORD=P@ssw0rd -p 3306:3306 -d mysql
```

Эта команда создаст и запустит контейнер с именем `pycharm-mysql`. Он устанавливает корневой пароль для базы данных MySQL `P@ssw0rd`. Флаг `-p` мапирует порт 3306, который является стандартным портом MySQL, с одним и тем же значением между вашим контейнером и хостом. Это создаст впечатление, будто вы используете сервер MySQL прямо на своем компьютере. Флаг `-d` сообщает Docker запускать MySQL в качестве фонового процесса, а не ждать его завершения. Это обычное явление для серверного программного обеспечения. Надеюсь, мне не нужно напоминать вам, что *это еще не готово к производству*. Если вы находитесь в небольшой группе, где разработчики отвечают за поддержание производственной среды, не дублируйте свою среду разработки на сервере, открытом в интернете. Вам следует, по крайней мере, сопоставить свою базу данных с более надежным постоянным хранилищем, а также усилить защиту MySQL, использовать непривилегированные учетные записи для своего приложения и использовать нестандартный корневой пароль.

Теперь у нас запущен сервер MySQL. Когда вы запускаете команду Docker, ваш результат будет несколько загадочным и неудовлетворительным. Мой вариант – на рис. 11.2.

```

bruce@bruce-workstation: $ docker run --name pycharm-mysql -e MYSQL_ROOT_PASSWORD=P@ssw0rd -p 3306:3306 -d mysql
8d3076aa35f8e0c60bf57425e72f08a39bbccada3bb2d0fe61f8136bed358552
bruce@bruce-workstation: $

```

Рис. 11.2. Длинная строка, казалось бы, случайных букв и цифр – это способ Docker сказать «Я люблю тебя» или, по крайней мере, что ваш контейнер работает

Есть старая поговорка: «Скажи человеку, что в галактике триллион звезд, и он тебе поверит. Сообщи ему, что его контейнер успешно работает в Docker, и он запустит `docker ps` -а на всякий случай, чтобы удостовериться, что это так».

По правде говоря, возможно, я это только что придумал. Тем не менее давайте удостоверимся:

```
docker ps -a
```

Вы должны увидеть доказательство, подобное моему, на рис. 11.3. Значение CONTAINER ID будет разным для каждого запуска, поэтому не ждите, что ваше будет совпадать с моим.

```
bruce@bruce-workstation: $ docker run --name pycharm-mysql -e MYSQL_ROOT_PASSWORD=P0ssw0rd -p 3306:3306 -d mysql
8d3076aa35fbc0c60bf57425e72f08a39bbccada3bb2d0fe61f8136bed358552
bruce@bruce-workstation: $ docker ps -a
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS
8d3076aa35fb	mysql	"docker-entrypoint.s..."	11 minutes ago	Up 11 minutes	0.0.0.0:3306->3306/tcp, 33060/tcp

```
pycharm-mysql
bruce@bruce-workstation: $
```

Рис. 11.3. Я вижу, что мой контейнер с именем pycharm-mysql запущен и имеет открытый порт 3306

Остановка и запуск контейнера

Когда вы будете готовы уйти с работы домой, можете остановить контейнер:

```
docker stop pycharm-mysql
```

Это останавливает контейнер. Вы можете проверить это с помощью той же команды `docker ps -a`, которую мы использовали для подтверждения изменения статуса с *Up* на *Stopped*. Завтра утром, когда вернетесь, напишите следующее:

```
docker start pycharm-mysql
```

Это запустит процесс, и вы сможете продолжить с того места, на котором остановились.

Подключение к источникам данных с помощью PyCharm

Откройте PyCharm Professional и создайте новый проект Python под названием `data_fun`. Теперь найдите инструменты базы данных. Их можно найти на правой панели инструментов через значок базы данных, который выглядит как трехслойный торт (ням!). Кроме того, можете найти его через меню-гамбургер (ням!), кликнув **View | Tool Windows | Database**. Оба варианта показаны на рис. 11.4.

Открыв инструменты базы данных, вам необходимо создать новый **источник данных**. Обратите внимание на общую терминологию. PyCharm поддерживает как реляционные, так и нереляционные базы данных, поэтому термин «источник данных» – это всего лишь общий способ указать на это. Кликните значок +, показанный на рис. 11.5, затем наведите указатель мыши на **Data Source**. Вы увидите длинный список поддерживаемых источников данных. Найдите **MySQL** и кликните его.

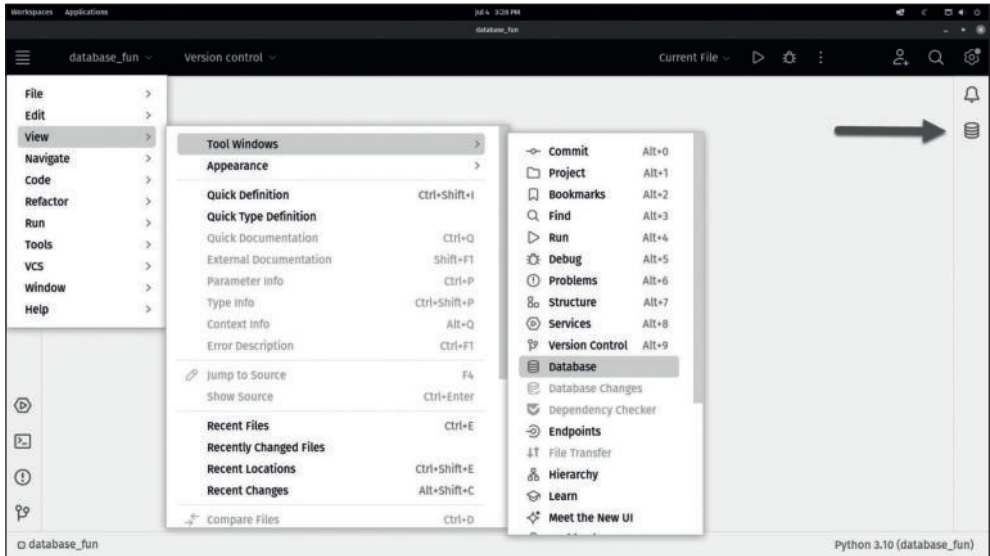


Рис. 11.4. Два варианта открытия инструментов базы данных – один из меню, а второй – путем щелчка по значку инструментов базы данных

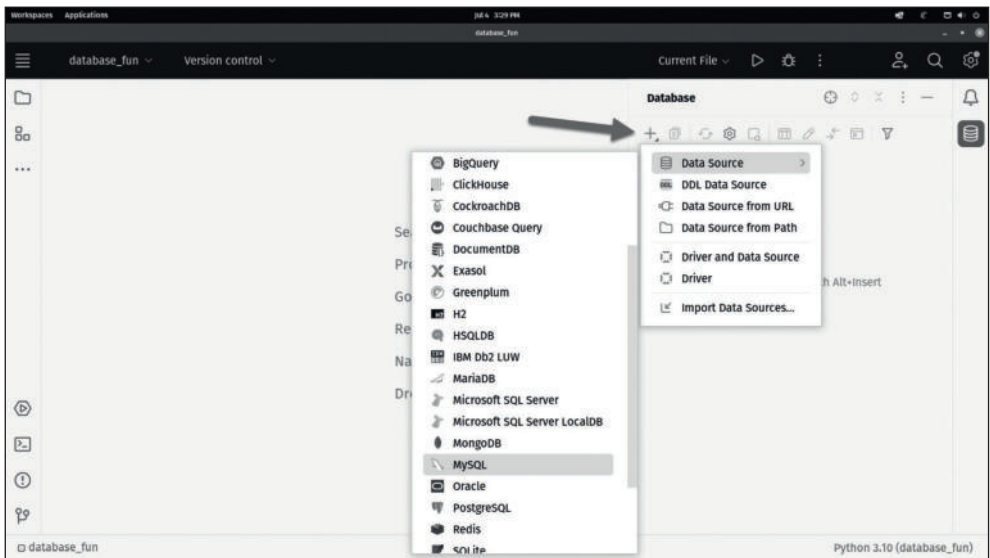


Рис. 11.5. Поддерживаемые источники данных в PyCharm

Вы увидите окно конфигурации, подобное моему на рис. 11.6.

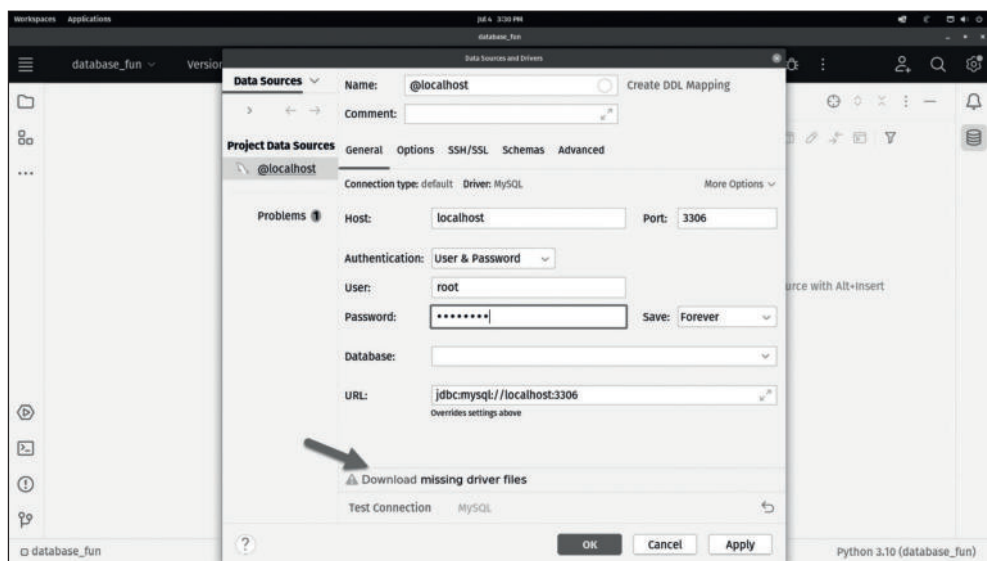


Рис. 11.6. Окно конфигурации базы данных MySQL

Каждый сервер базы данных может иметь несколько разные настройки, но, по сути, они сводятся к IP-адресу (или DNS-имени), порту, учетным данным безопасности и, очень часто, базе данных по умолчанию, которая может еще не существовать. Мы определили корневой пароль для нашего сервера MySQL как `P@ssw0rd` и знаем, что, поскольку мы работаем в Docker, наш IP-адрес будет просто `localhost`. Возможно, вы также помните, что порт, отображаемый в выводе команды `docker ps -a`, которую мы запускали ранее, – `3306`. Стрелка на предыдущем рисунке указывает на важный аспект инструментов базы данных в PyCharm.

PyCharm, как и большинство IDE JetBrains, написан на Java. Таким образом, для работы PyCharm использует **драйверы подключения к базе данных Java (JDBC)**. Большинство драйверов JDBC написаны той же компанией или группой, которая публикует базу данных, а это означает, что JetBrains обычно не имеет права объединять эти драйверы с PyCharm без привлечения юристов. Никто этого не хочет! JetBrains сделала следующую отличную вещь. Среда IDE может загрузить и установить драйвер автоматически, но вам необходимо инициировать это, кликнув ссылку **Download missing driver files** на этом экране. Это необходимо сделать только при первом использовании драйвера базы данных. После установки драйверов опция, показанная стрелкой на рис. 11.6, больше не отображается. Вы можете проверить свое соединение, нажав ссылку **Test Connection**. Если все работает, вы получите подтверждающее сообщение о том, что подключение к базе данных прошло успешно. Нажмите **OK**, чтобы закрыть диалоговое окно подключения.

После закрытия диалогового окна подключения вы увидите список подключений к источникам данных для вашего проекта на панели **Database**. В верхней части панели базы данных видна небольшая панель инструментов, как показано на рис. 11.7.

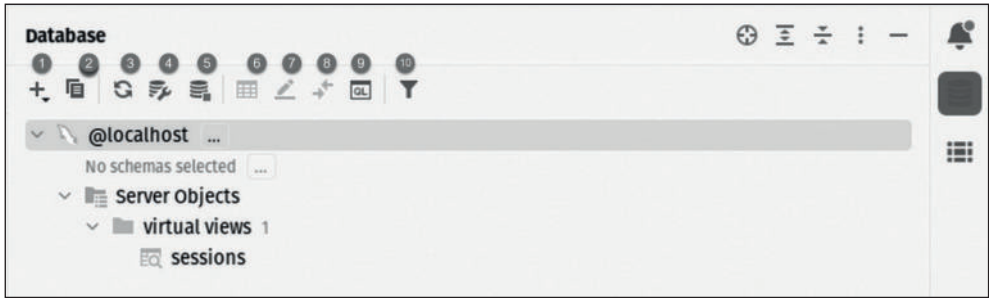


Рис. 11.7. Панель базы данных имеет небольшую строку меню сверху

Как обычно, я пронумеровал варианты. Давайте рассмотрим их.

1. Кнопка **Add Data Source**, которую мы уже видели, позволяет добавлять в проект новые источники данных.
2. Кнопка **Duplicate** позволяет быстро сделать копию, предположительно с небольшими изменениями. Многие из моих проектов включают в себя несколько баз данных на одном сервере. Все, что мне нужно сделать, – это настроить первый, затем продублировать соединение и изменить имя базы данных. Эта опция делает процесс быстрым и простым.
3. Кнопка **Refresh** перезагружает метаданные для ваших подключений. Запомните это. PyCharm не отслеживает автоматически все изменения в вашей базе данных, особенно если они сделаны вне PyCharm. Вам нужно будет периодически нажимать кнопку **Refresh**, чтобы убедиться, что вы просматриваете самую последнюю информацию о своих источниках данных.
4. Кнопка **Data source properties** отобразит диалог, позволяющий изменить параметры конфигурации источника данных.
5. Кнопка **Disconnect** отключит вас от сервера базы данных.
6. Кнопка **Edit data** позволяет напрямую редактировать данные в таблицах базы данных с помощью графического пользовательского интерфейса, похожего на электронную таблицу. Это удобно для быстрого добавления или изменения тестовых данных.
7. Кнопка **Go To DDL** приведет вас к определению SQL того, что вы в данный момент выбрали. Для правильной работы вам необходимо иметь маппинг DDL.
8. **Compare Structure** позволяет сравнить две структуры базы данных. Обычно это используется для облегчения миграции одной структуры базы данных в другую после внесения изменений в ходе обычного процесса разработки. Технология миграции в PyCharm 2023 реализована лишь частично, поэтому к моменту чтения этой книги эта функция могла измениться.
9. **Jump to Query Console** – это ваш инструмент для взаимодействия с базой данных из командной строки. Консоль запросов открывалась автоматически при первом подключении к базе данных, но, если вы закрыли ее, эта кнопка откроет ее и переведет в фокус.

10. Кнопка **Filter** позволяет фильтровать то, что вы видите в определенных вами источниках данных. По умолчанию все включено, что может оказаться «слишком» для большинства разработчиков, которые обычно не привыкли видеть все внутренности базы данных столь явно отображаемыми.

Помимо этих инструментов, есть и другие, которые позволяют настроить просмотр источников данных и работу с ними. Это будет немного проще объяснить, когда у нас будет база данных, с которой можно будет поиграть. Давайте потратим некоторое время на ее создание.

Создание новой базы данных

Прежде чем мы сделаем что-нибудь еще, нам нужна новая база данных. Большинство серверов баз данных называют это просто «новой базой данных». MySQL немного отличается, здесь называют новую базу данных новой **schema**. Чтобы создать новую базу данных или схему, кликните правой кнопкой мыши сервер (@localhost) и выберите **New | Schema**, как показано на рис. 11.8.

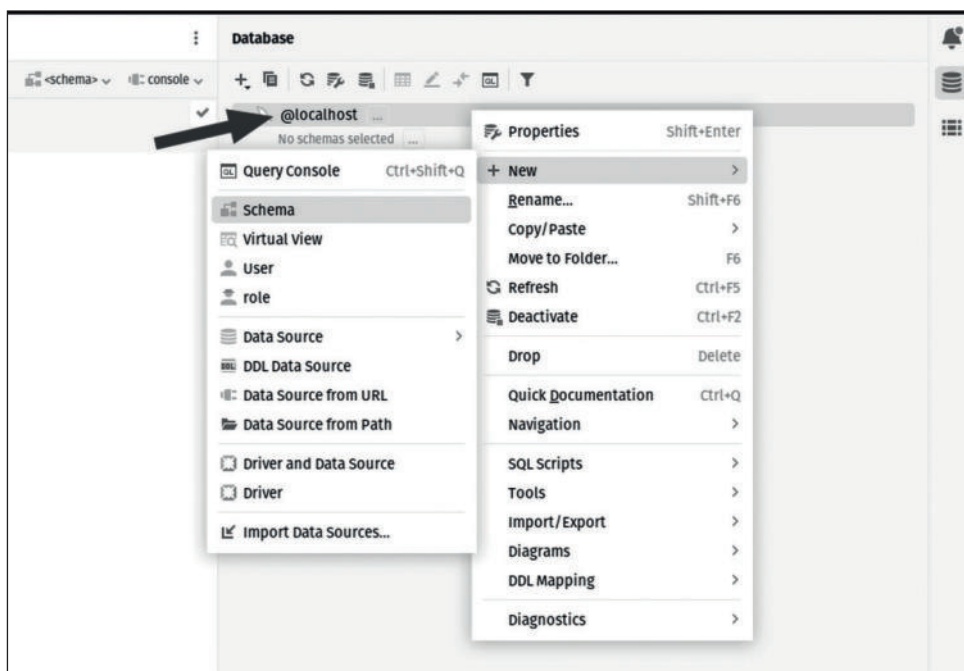


Рис. 11.8. Создайте новую базу данных, кликнув правой кнопкой мыши сервер (указан стрелкой), затем **New | Schema**

При этом вам будет предложено назвать схему, как показано на рис. 11.9. Я назову свою схему в соответствии с моим проектом в PyCharm. Я назову это `Database_fun`.

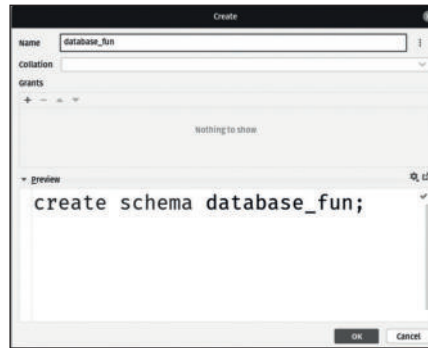


Рис. 11.9. Назовите свою схему с помощью этого диалогового окна

Если заглянуть под капот, PyCharm просто генерирует и выполняет инструкции DDL. Именно поэтому он может быть настолько независимым в отношении множества параметров базы данных. Предварительный просмотр команды, которую он запустит, можно увидеть на рис. 11.9. Нажмите **ОК**, чтобы выполнить команду, и окно базы данных обновится, чтобы отобразить новую схему.

Вы только что создали новую базу данных! Так же легко использовать любое другое поддерживаемое программное обеспечение сервера баз данных от PyCharm.

Прежде чем мы приступим к работе над структурой нашей базы данных, необходимо рассмотреть еще несколько вариантов настройки.

Установка диалекта SQL (это важно)

Поскольку PyCharm поддерживает десятки различных баз данных, каждая из которых имеет свой собственный диалект SQL, вполне понятно, что может потребоваться указать PyCharm, какой диалект SQL вы собираетесь использовать. Вам не составит труда запомнить это, потому что PyCharm будет беспокоить вас, пока вы не заполните эту настройку.

В окне проекта кликните проект правой кнопкой мыши и создайте новый файл с именем `test.sql`. Это тот же процесс, который мы использовали для файлов Python, за исключением того, что в списке нет шаблона. Кликните проект правой кнопкой мыши, затем выберите **New File**, как показано на рис. 11.10.

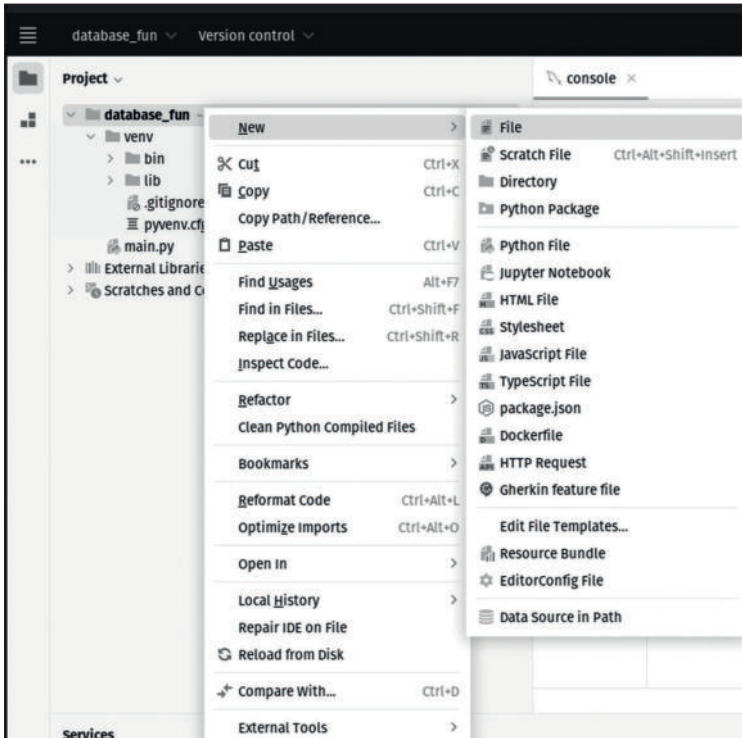


Рис. 11.10. Специального списка для файлов SQL нет, поэтому просто кликните правой кнопкой мыши и выберите New File

Появится небольшой диалог. Введите имя файла `test.sql`. В тот момент, когда вы это сделаете, и начнется нытье. Вы увидите сообщение о том, что диалект SQL не настроен, как показано на рис. 11.11.

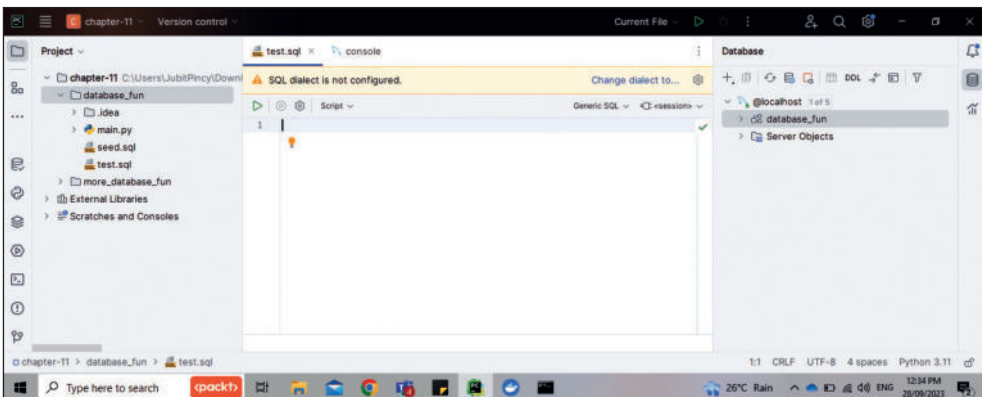


Рис. 11.11. PyCharm будет надоедать вам, пока вы не настроите диалект SQL

Если бы PyCharm был мной, а пользователем PyCharm была бы моя 13-летняя дочь, было бы много закатываний глаз, раздраженного ворчания, за которым

последовало бы: «Хорошо! Я установлю диалект SQL! Но никто из моих друзей этого не делает!» Тогда я бы сказал: «Если бы все твои подруги установили на свои компьютеры Windows 7, ты бы последовала их примеру?» Она опустила бы голову и сказала: «Нет, конечно, нет».

Никто не хочет, чтобы этот диалог разыгрался, поэтому нам лучше порадовать нашего повелителя IDE. Кликните ссылку на рис. 11.11, чтобы установить диалект. Вам нужно будет установить его локально и глобально. Конфигурация показана на рис. 11.12.

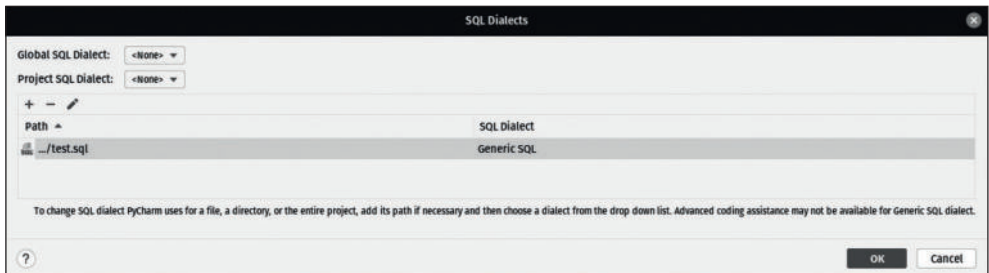


Рис. 11.12. Настройка диалектов SQL глобально и локально

Глобальная настройка распространяется на все проекты, поэтому, если вы, как и я, используете только один тип сервера базы данных, можете установить его здесь глобально, и он будет установлен для всех ваших проектов. Диалект проекта немного более тонкий. Если не хотите свихнуться, вам действительно нужно создать папку для файлов SQL, а затем установить диалект для этой папки. Здесь я просто показываю вам, где находятся настройки. Продолжайте и выберите **MySQL** в качестве глобального и локального диалекта, а обо всей идее папок мы поговорим чуть позже.

Группирование и цветовое кодирование источников данных

Моя собственная работа заключается в создании SaaS-продукта, который продает мой работодатель. Это означает, что в течение первых семи лет проекта мне приходилось иметь дело только с одной базой данных. По мере расширения возможностей продукта мы добавили в проект еще две базы данных SQL, базу данных MongoDB и несколько кешей данных Redis. Мой проект все еще довольно скучен по сравнению с некоторыми, над которыми я работал. Если у вас много баз данных, PyCharm позволяет организовать их несколькими различными способами.

Группирование по папкам

Вы можете упорядочить источники данных, сгруппировав их по папкам. У нас есть только одна, но мы все равно пройдем через этот процесс. Кликните источник данных, указанный стрелкой на рис. 11.8. В прошлый раз мы кликнули правой кнопкой мыши; на этот раз просто кликните источник данных и нажмите **F6** на клавиатуре. При этом появится диалоговое окно, как показано на рис. 11.13.

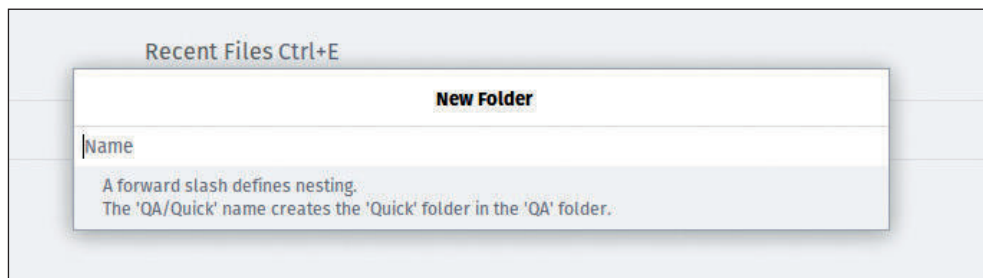


Рис. 11.13. Создайте новую папку для вашего источника данных

Организация по папкам может быть полезна, если в вашем проекте много баз данных или, возможно, у вас есть отдельные наборы подключений для разработки, **пользовательского приемочного тестирования (UAT)** и производства.

Цветовое кодирования баз данных

Мне нравится эта функция, поскольку у меня есть четыре среды.

- Локальная база данных, такая же, как у нас сейчас.
- Центральная база данных тестирования разработки, где разработчики проверяют свои проекты, прежде чем команда разработчиков сможет увидеть наши изменения.
- Промежуточная база данных, подключенная к промежуточной версии нашего приложения для UAT.
- Производственная база данных.

Я всегда все раскрашиваю! Вы можете установить цвет источника данных, открыв окно **Data Source Properties**. Для этого нажмите кнопку **Data Source Properties** на панели инструментов. Я показал вам это ранее на рис. 11.7 (4). При нажатии этой кнопки открывается диалоговое окно, похожее на то, которое вы использовали ранее для создания источника данных. Посмотрите на мое на рис. 11.14.

Вероятно, вы сможете понять это здесь. Как только вы закроете диалоговое окно, элементы вашей базы данных в IDE окрасятся в приятные пастельные тона. Лично я установил для своей локальной среды разработки фиолетовый цвет, среды разработки – зеленый, промежуточной – желтый, а для рабочей – красный. Не розовый из списка, а яркий, неприятный, горящий красный цвет пожарной машины, выбранный в пользовательском диалоговом окне внизу списка. Как руководителю разработки проекта, бывают случаи, когда мне нужно посмотреть, как выглядит проект в производстве. Я хочу убедиться, что помню, где нахожусь! Цветовое кодирование помогает!

Совместное использование источника данных различными проектами

Если у вас много проектов, использующих один и тот же источник данных, было бы утомительно снова и снова задавать одно и то же для каждого проекта. К счастью, нам это не нужно. PyCharm позволяет сделать источник данных глобальным, т. е. он доступен всем вашим проектам в PyCharm. В окне свойств источника данных, показанном на рис. 11.15, найдите кнопку, указанную стрелкой.

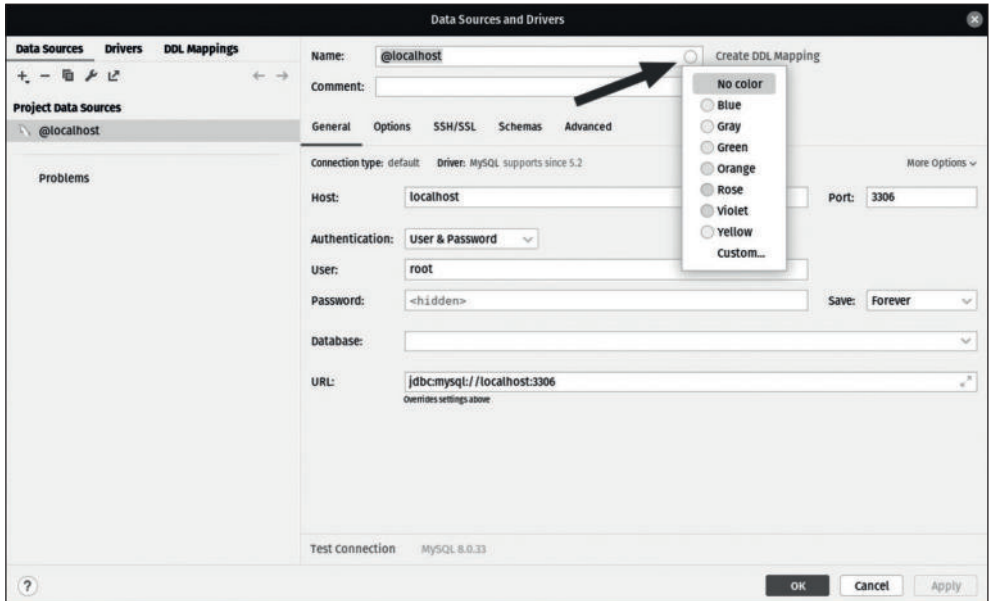


Рис. 11.14. Кликните безобидную точку рядом с именем источника данных в окне свойств, чтобы установить цвет для вашего источника данных

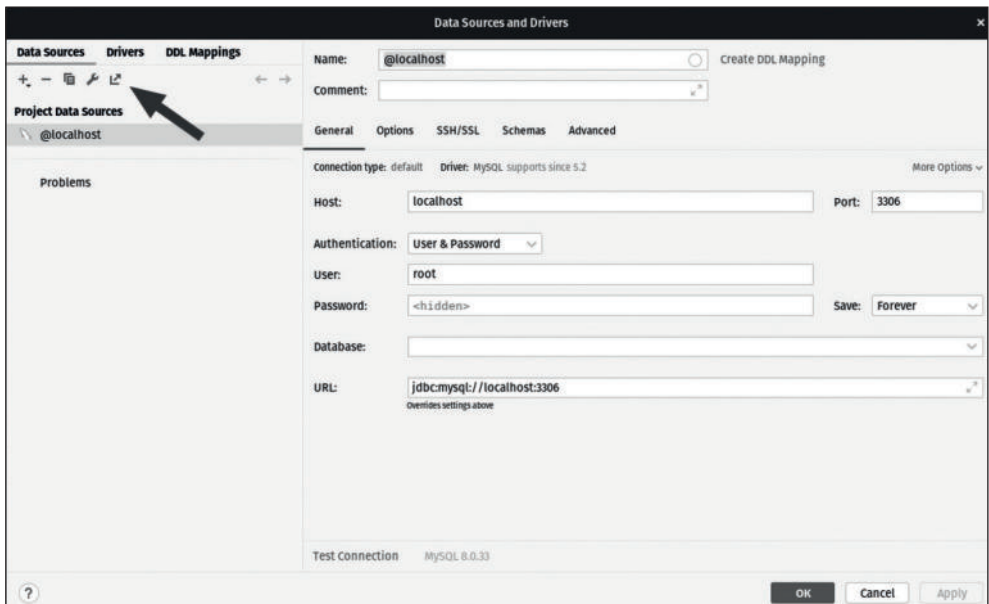


Рис. 11.15. Вы можете сделать конфигурацию источника данных глобальной, чтобы она была доступна всем вашим проектам

Если кликнуть по ней, диалоговое окно слегка изменится, показывая, что источник данных теперь является глобальным. Если вы хотите проверить, про-

сто создайте новый проект, как мы делали ранее. Откройте инструменты базы данных, и вы обнаружите, что источник данных уже там. Я создал проект под названием `morg_database_fun`, и, как вы можете видеть на рис. 11.16, источник данных действительно был перенесен.

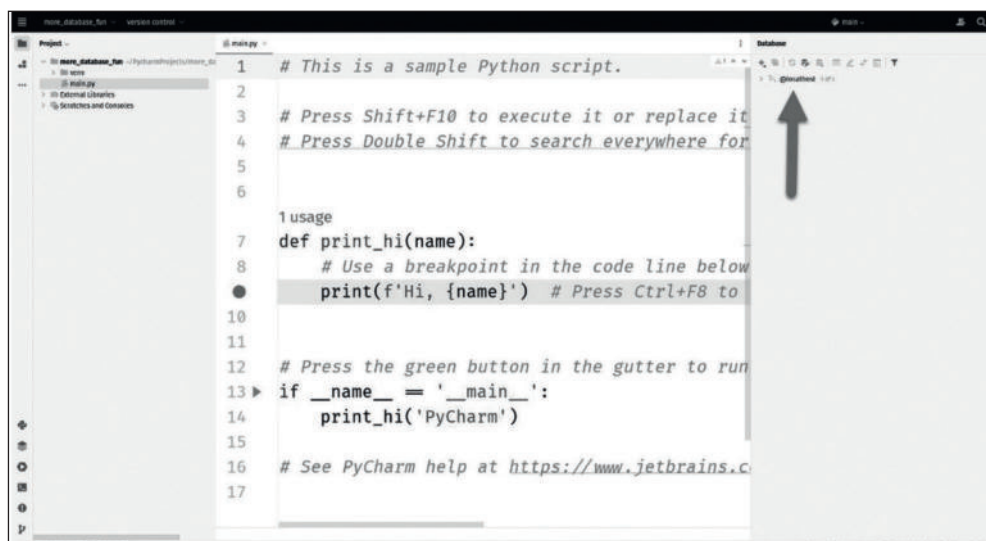


Рис. 11.16. Я создал новый проект и открыл инструменты базы данных. Глобальный источник данных здесь

Поскольку во втором проекте нет кода, я не включил его в исходный код главы.

Еще одна интересная функция – глобальные источники данных экспортируются при экспорте настроек IDE. Вернитесь к главе 2, если вам нужно освежить знания об импорте и экспорте настроек, но я продолжу и укажу на настройки экспорта, о которых говорю, на рис. 11.17.

Если вы являетесь руководителем разработки, можете рассмотреть возможность экспорта настроек вашей IDE, который не включает такие вещи, как цвет и размер шрифта. Вероятно, это сугубо личные предпочтения, и они имеют смысл, если вы хотите скопировать настройки между ноутбуком и рабочей станцией. Но имеет смысл экспортировать более специфичные для работы настройки, например источники данных, чтобы ваша команда могла легко их импортировать. Если вы сделаете это, помните, что вы также потенциально копируете учетные данные! Источники данных разработки следует предоставлять только в виде глобального экспорта, чтобы файл `settings.zip` не попал в чужие руки. Это особенно важно в эпоху, когда все больше и больше из нас используют облачные серверы баз данных, а не локально установленные серверы.

Если вам нужен жесткий контроль над учетными данными, PyCharm можно интегрировать с продуктом под названием *KeePass*. Поскольку это довольно специальная тема, я не буду ее здесь освещать, а оставлю ссылку с более подробной информацией в разделе «Дальнейшее чтение» этой главы.

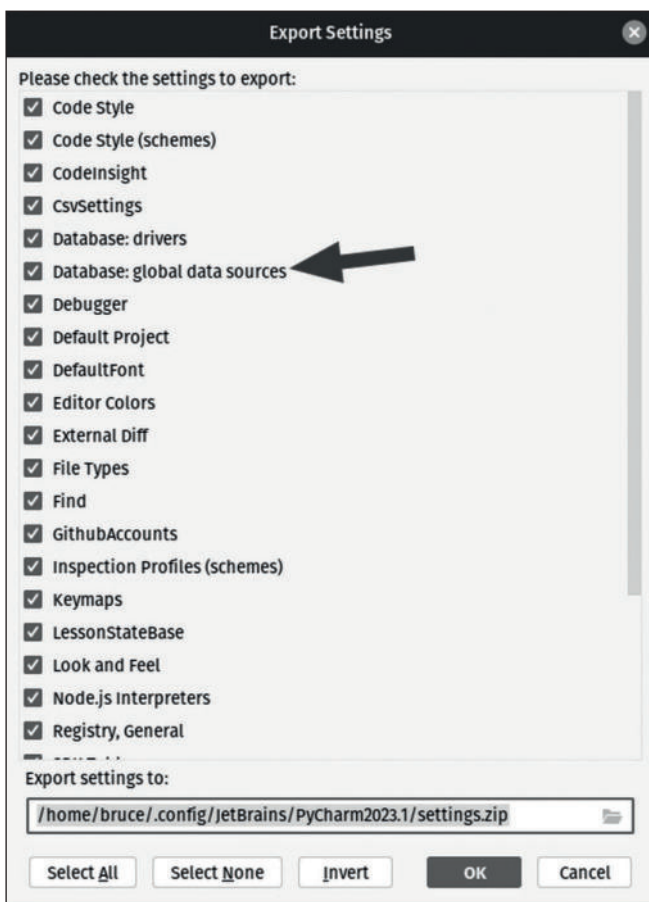


Рис. 11.17. Вы можете экспортировать глобальные источники данных в ZIP-файл экспорта настроек, позволяя другим просто импортировать соответствующие настройки

Подключение с помощью опций SSL и SSH

Для еще большей безопасности, особенно в отношении облачных источников данных, PyCharm поддерживает параметры конфигурации SSL и SSH. Смотрите рис. 11.18, где показаны варианты настройки безопасности для ваших источников данных.

Я не буду углубляться в это, но вам нужно знать, где находятся эти настройки в IDE, поскольку они понадобятся вам для провайдеров облачных услуг, таких как Microsoft Azure.

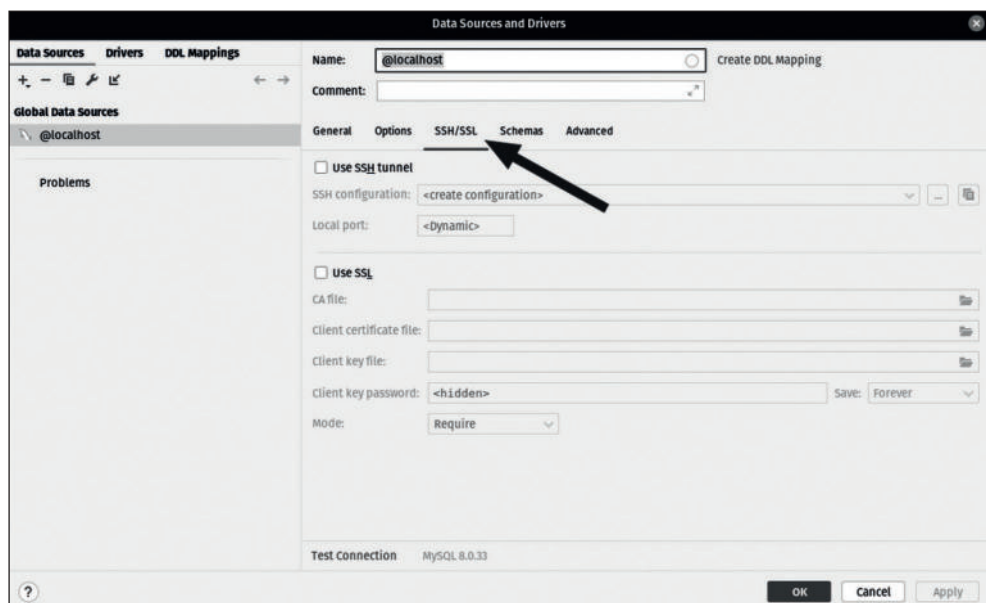


Рис. 11.18. Параметры безопасности SSH и SSL можно найти в настройках источника данных на вкладке SSH/SSL

СОЗДАНИЕ БАЗЫ ДАННЫХ И МАНИПУЛЯЦИИ С НЕЙ

Давайте перейдем к той части, которую вы, вероятно, ждали: той части, где мы собираемся создать базу данных. Схему мы уже создали, но на данный момент таблиц у нас нет. Давайте это исправим!

Создание таблицы

Кликните правой кнопкой мыши схему, **Database_fun**, которую мы создали ранее, и выберите **New | Table**, как показано на рис. 11.19.

Вы получите новое окно, как показано на рис. 11.20.

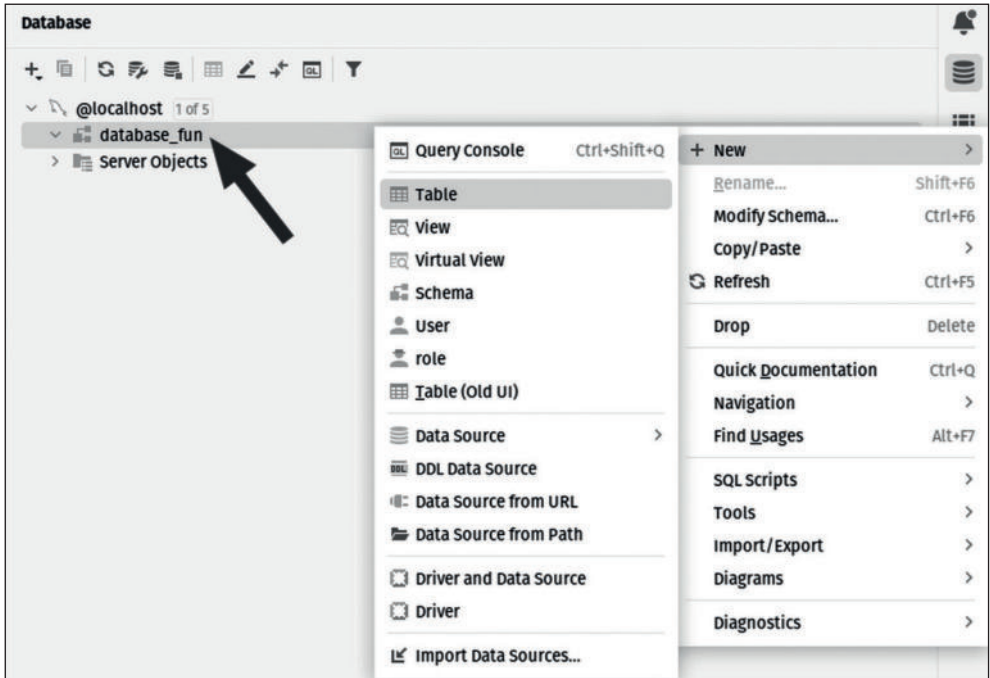


Рис. 11.19. Кликните правой кнопкой мыши схему, указанную стрелкой, затем выберите New | Table

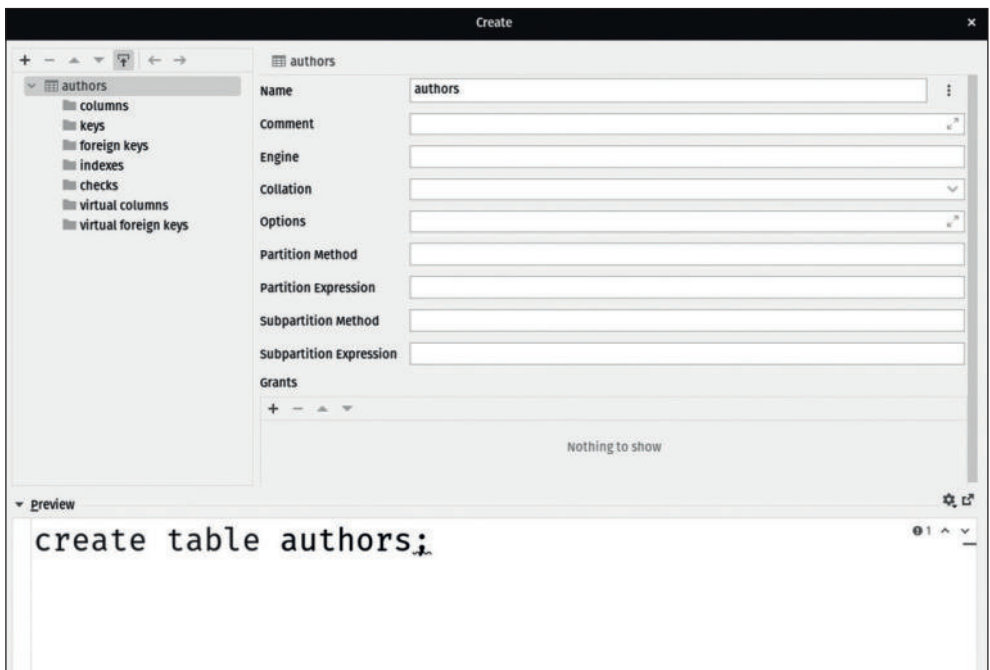


Рис. 11.20. Новое окно

Мне пришлось немного растянуть свое, чтобы увидеть все, как показано выше. В следующих нескольких шагах мы построим нашу таблицу. Я иду дальше и ввожу имя таблицы в поле **Name** сверху. Вы можете увидеть, что здесь происходит. Как и раньше, при создании DDL схема PyCharm создает скрипт DDL в предварительном просмотре в нижней части окна. Прямо сейчас у нас есть красная волнистая линия под точкой с запятой, потому что нам еще предстоит добавить какие-либо поля, поэтому этот DDL недействителен.

В левом верхнем углу окна находится кнопка со значком +. Нажмите на нее, чтобы увидеть список элементов, которые вы можете добавить в таблицу, как показано на рис. 11.21.

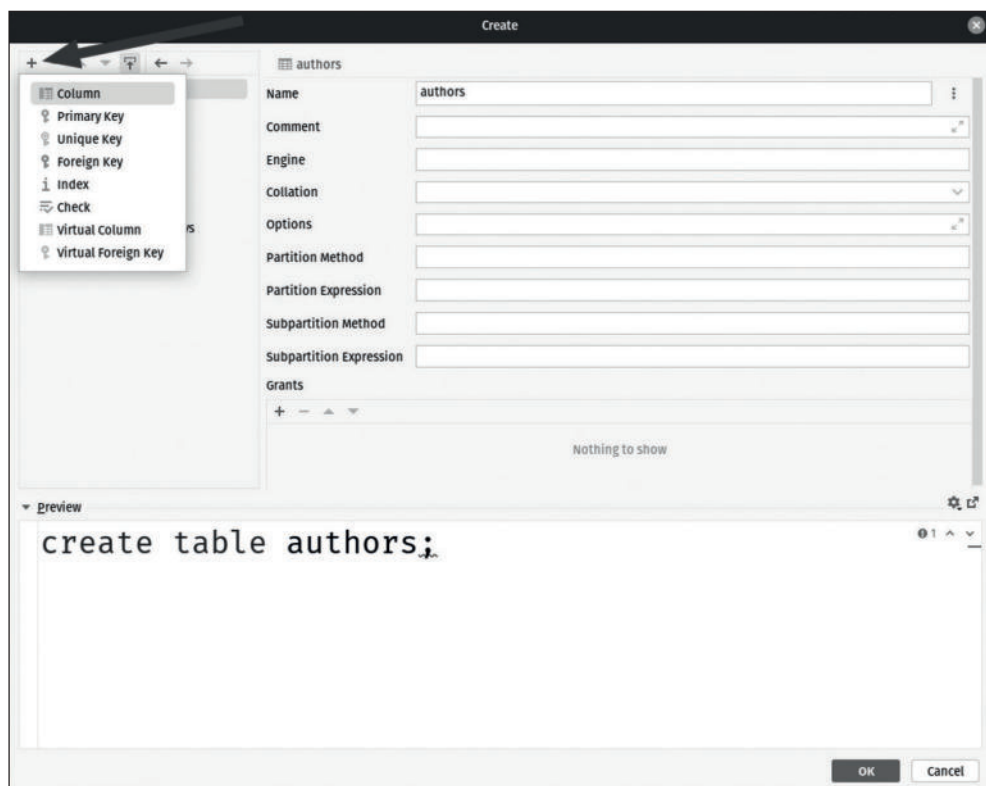


Рис. 11.21. Вы можете добавить в таблицу, нажав кнопку +

С этого момента мы можем выбрать все, что нужно добавить. Сначала добавим поле первичного ключа. Если вы новичок в проектировании реляционных баз данных и вам нужен хороший учебник, обязательно прочтите «Проектирование баз данных для простых смертных» Майкла Эрнандеса. Я использую его в качестве учебника на своих занятиях более 20 лет! У Эрнандеса есть не менее впечатляющая книга по SQL-запросам, как и у другого автора по имени Бен Форта¹. Подробности я оставляю в разделе «Дальнейшее чтение» этой главы.

¹ Бен Форта – директор департамента разработки в компании Adobe Systems. – Прим. ред.

Короче говоря, каждая создаваемая нами таблица должна иметь поле, которое может однозначно идентифицировать отдельные записи. Оно не должно содержать данные, относящиеся к области данных, содержащихся в таблице. Под этим я подразумеваю, что мы создаем таблицу для хранения информации об авторах книг. Мы не должны использовать какое-либо поле, связанное с автором, в качестве первичного ключа, уникального идентификатора записи. Вместо этого мы должны использовать что-то не связанное с этим. В MySQL нормой является использование целочисленного поля с автоматическим приращением.

Целочисленное поле с автоматическим приращением – это поле, которое автоматически заполняет число из последовательности, начиная с числа 1, и автоматически увеличивается при каждой вставке записи. Увеличение – это то, что вам в коде делать не следует. Это особенность самой базы данных. Это важно. Реляционные базы данных спроектированы как атомарные, т. е. все транзакции изолированы. Это важно, поскольку означает, что база данных никогда не будет генерировать одно и то же значение для автоинкрементного ключа¹, даже если две вставки записей происходят с разницей в микросекунды. Это то, что вы не можете гарантировать своим собственным кодом. Для этого вам нужно полагаться на сервер.

Используя меню, показанное на рис. 11.21, добавьте столбец и настройте его, как показано на рис. 11.22.

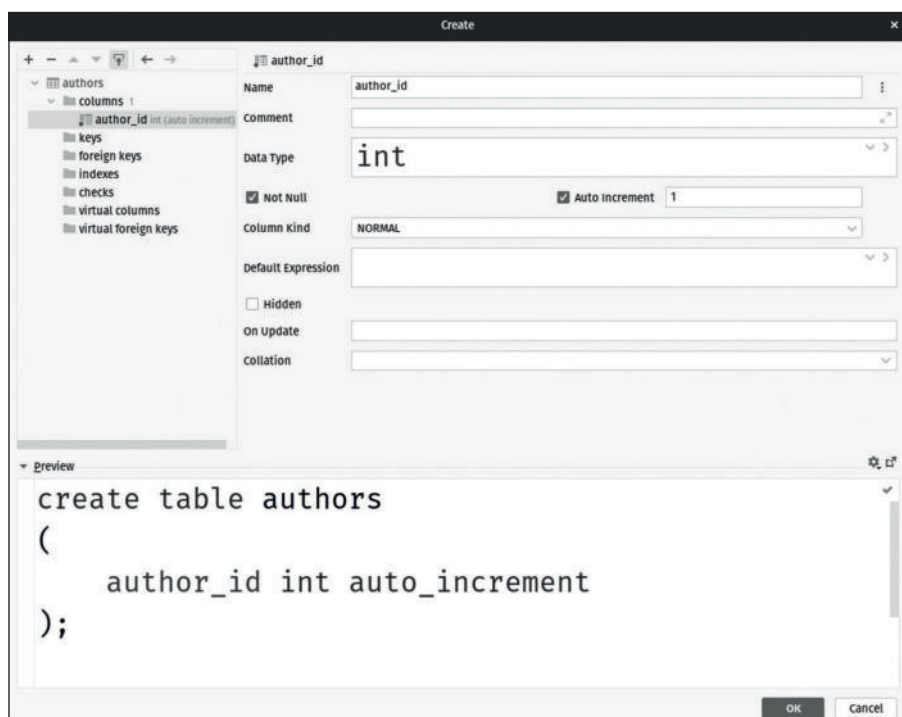


Рис. 11.22. Добавьте первый столбец с показанными здесь настройками

¹ Автоинкрементный ключ – это специальный тип полей в базах данных, который автоматически генерирует уникальное числовое значение для каждой новой записи. – *Прим. ред.*

Здесь мы создали целочисленное поле с автоматическим приращением под названием `author_id`. Я использую змеиный регистр, но это не обязательно. Используйте любые соглашения об именах, которые требуются вашему проекту. Обратите внимание: я еще официально не сделал его первичным ключом. Это будет чуть позже. Во-первых, давайте закончим с нашими оставшимися столбцами.

Это можно сделать быстро, если оставить поле `author_id` выбранным, а затем трижды нажать кнопку «плюс». Когда у вас выбран столбец или даже если в окне выбрана папка столбцов, PyCharm предполагает, что при нажатии кнопки + вам нужен столбец. Аналогичным образом вы можете выбрать любую другую папку для таких вещей, как первичные ключи, внешние ключи и индексы. PyCharm просто создаст это, не заставляя вас использовать меню, которое мы видели ранее.

Настройте три поля следующим образом:

- `first_name: varchar(30);`
- `last_name: varchar(30);`
- `email: varchar(255) not null.`

На рис. 11.23 показана последняя конфигурация поля. Я задал ей ограничение `not null`, которое предотвратит завершение вставки записи, если поле электронной почты останется пустым. Это делается путем установки флажка **Not null** как показано на рис. 11.22.

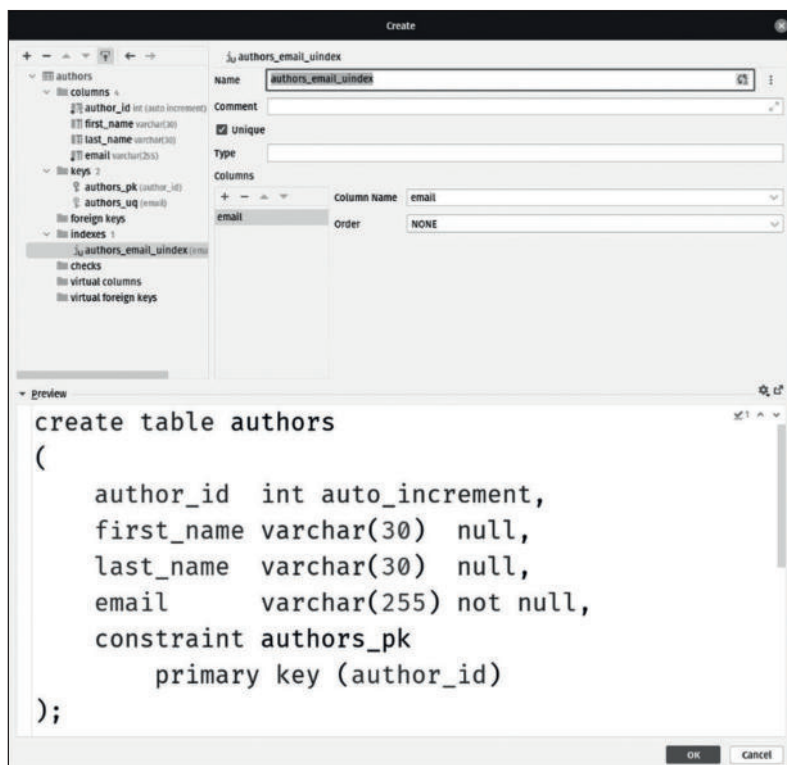


Рис. 11.23. Настройте поле электронной почты, как показано

Это начинает напоминать таблицу, не так ли? Нам понадобится еще пара вещей. Давайте продолжим и настроим поле `author_id` как правильный первичный ключ.

Установка первичного ключа

Нажмите на папку с `keys`, затем – кнопку `+`. Вас спросят, хотите ли вы создать первичный или уникальный ключ, как показано на рис. 11.24.

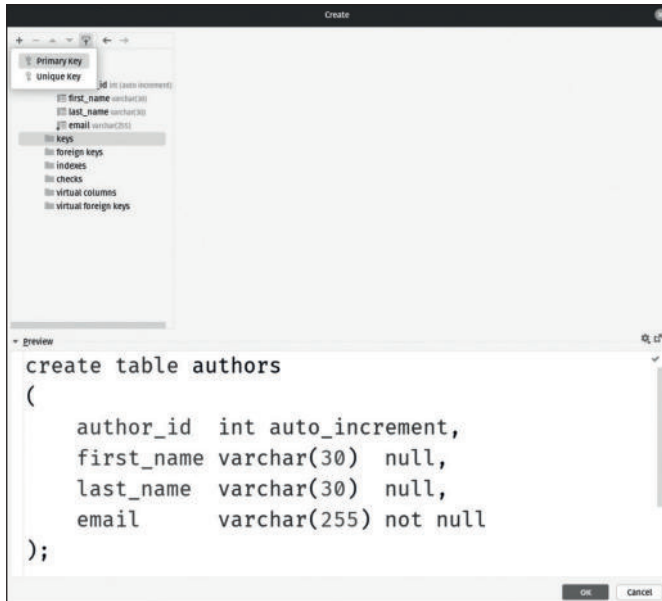


Рис. 11.24. Убедитесь, что папка ключей выбрана, как показано на этом рисунке, затем кликните значок `+` и выберите `Primary Key`

Откроется другое окно, как показано на рис. 11.25.

Чтобы добавить ключ, нажмите кнопку `+`, показанную стрелкой, затем кликните раскрывающийся список, чтобы открыть поля. Кликните поле `author_id`, чтобы добавить его в качестве первичного ключа. Реляционная теория допускает создание составных ключей, поэтому потенциально вы можете добавить более одного поля. В общей практике он не используется, поскольку если вы придерживаетесь автоинкрементного целого числа для своего ключа, создание составного первичного ключа не требуется. Я считаю, что код с составным ключом издает некий запах, не будем уточнять.

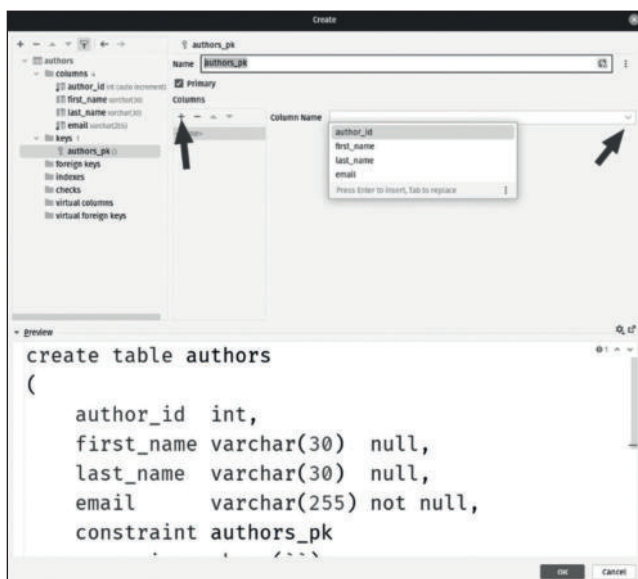


Рис. 11.25. Для добавления первичного ключа необходимо кликнуть значок + и затем выбрать имя поля из раскрывающегося списка

Добавление ограничения уникального ключа

Далее давайте ограничим поле email таким образом, чтобы значение любой вставленной записи было уникальным. Это предотвращает проблемы токсичности данных, возникающие из-за беспорядочных вставок. Для каждого адреса электронной почты должна быть только одна запись. Давайте формально реализуем это с помощью уникального ключа.

Процесс такой же, как и с первичным ключом, за исключением того, что на этот раз мы выберем уникальный ключ вместо первичного ключа и укажем поле email. Однако есть хитрость. На этот раз вам нужно будет кликнуть правой кнопкой мыши папку keys, чтобы выбрать **Unique Key**. Если вы просто кликните значок +, будет добавлено еще одно поле первичного ключа, а это не то, что нам нужно. На рис. 11.26 показано, что делать.

Кликните правой кнопкой мыши папку keys, обозначенную стрелкой на предыдущем рисунке, затем выберите **New | Unique Key** для создания ограничения уникального ключа. Затем, как и раньше, кликните раскрывающийся список, далее кликните поле email. Я бы также изменил имя сгенерированного значения authors_pk2 на authors_uq. Это сразу скажет вам в DDL, что это уникальное ограничение поля.

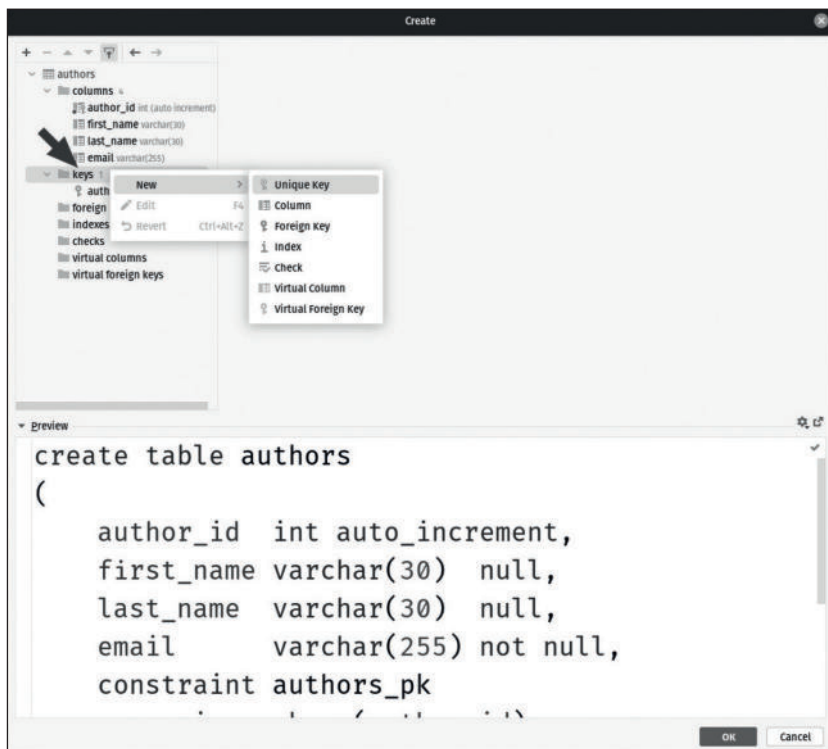


Рис. 11.26. Кликните правой кнопкой мыши по клавишам, затем выберите New | Unique Key, чтобы избежать создания второго первичного ключа

Добавление индекса

Добавление индекса¹ в таблицу повысит производительность фильтрованных операций чтения. Было бы разумно ожидать, что приложение позволит пользователям искать автора по адресу электронной почты. Добавление индекса в поле email ускорит этот поиск за счет замедления вставки новых записей. Снижение производительности при вставке невелико, но, если бы вы добавили индекс к каждому отдельному полю, это стало бы заметно, поэтому тщательно выбирайте поля, которые хотите индексировать.

Процесс создания ключей такой же (см. рис. 11.27).

¹ В Python индекс – это, по сути, отступ от начала списка. Первый элемент имеет индекс 0 (нет никакого отступа, элемент в самом начале списка). Второй элемент имеет индекс 1 и т. д. В общем, если в списке n элементов, последний из них будет иметь индекс $(n-1)$. – *Прим. ред.*

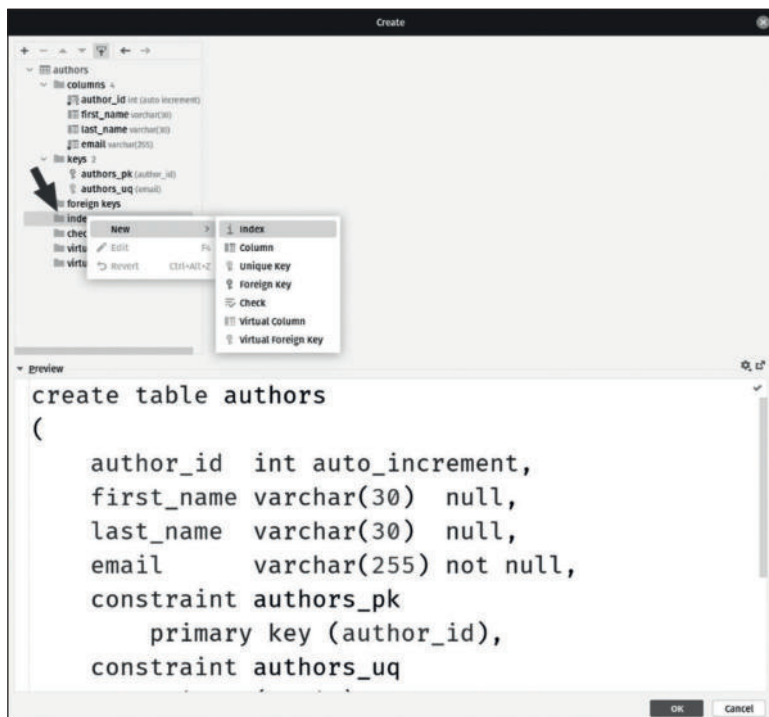


Рис. 11.27. Кликните правой кнопкой мыши папку индексов и добавьте индекс, как показано

На этот раз мы кликаем правой кнопкой мыши папку `indexes` и выбираем **New | Index**. Отсюда знакомый диалог, как показано на рис. 11.28.

На этом этапе код нашей таблицы не помещается на скриншоте, поэтому вот что у нас есть:

```
create table authors
(
  author_id int auto_increment,
  first_name varchar(30) null,
  last_name varchar(30) null,
  email varchar(255) not null,
  constraint authors_pk
    primary key (author_id)
);

create unique index authors_email_uindex
on authors (email);

alter table authors
add constraint authors_uq
unique (email);
```

Когда вы будете довольны структурой таблицы, нажмите **ОК**, и PyCharm применит сгенерированный код DDL к базе данных. Результаты можно увидеть в источнике данных. Посмотрите на рис. 11.29.

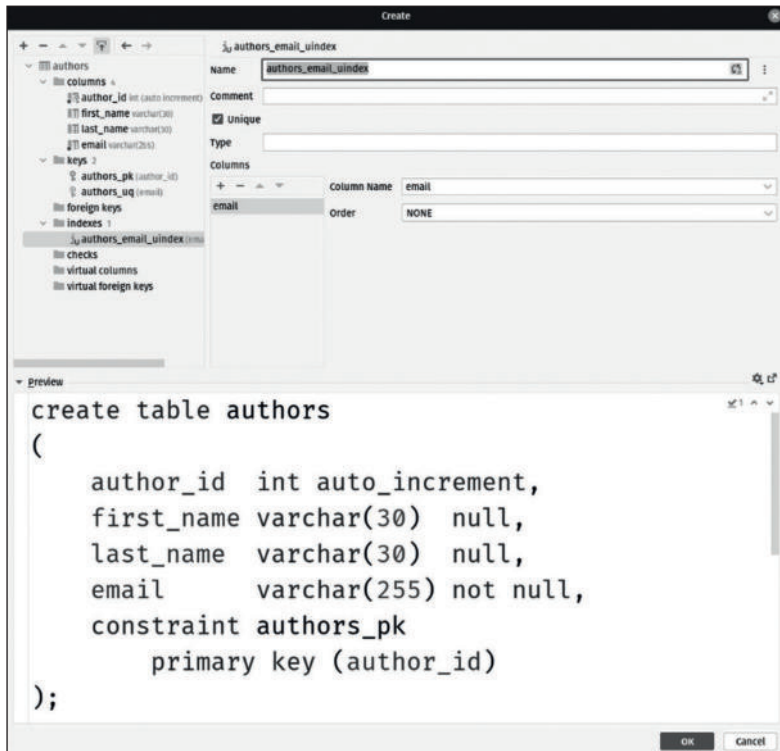


Рис. 11.28. Мы добавили индекс в поле электронной почты

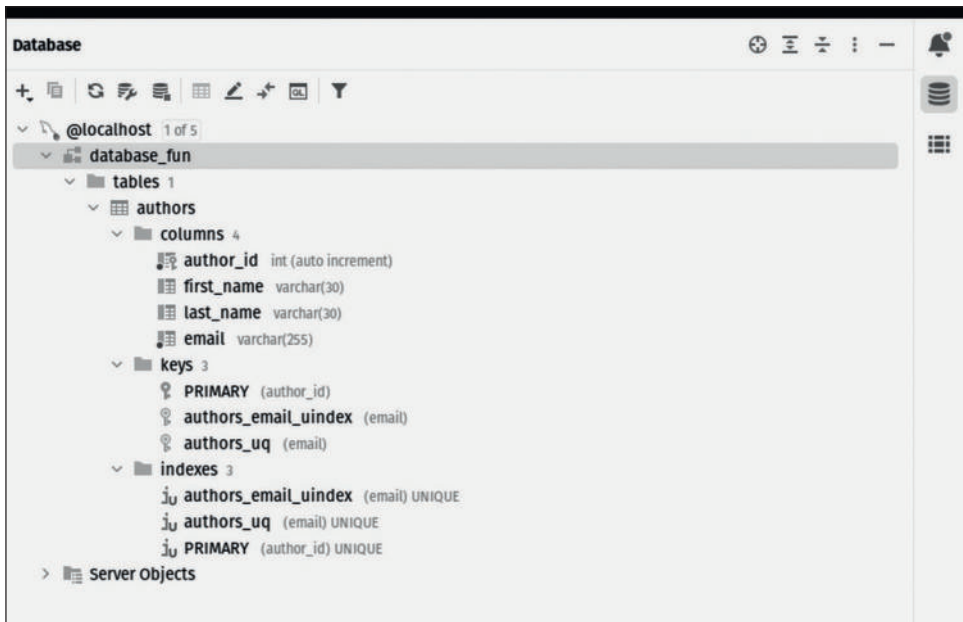


Рис. 11.29. Результаты нашей кропотливой работы показаны на панели источника данных

Изменение существующих структур

За прошедшие годы я видел множество инструментов, позволяющих графически создавать структуру базы данных, но, когда дело доходит до изменения существующих структур, они терпят неудачу. На ум приходит Microsoft SSMS! Он позволит вам весело проектировать только для того, чтобы сообщить вам, когда вы попытаетесь зафиксировать изменения, что он не может этого сделать.

PyCharm не такой. PyCharm обрабатывает изменения так, как этого требует администратор базы данных вашей компании. Если вы новичок в этой области, за базу данных отвечает администратор базы данных. Он босс. Если вам посчастливилось получить *право* создавать что-либо в их базе данных, вы будете следовать их правилам. И они хотят, чтобы вы изменяли структуры с помощью операторов SQL `alter`.

Кликните правой кнопкой мыши таблицу `authors` и выберите **Modify Table...**, как показано на рис. 11.30.

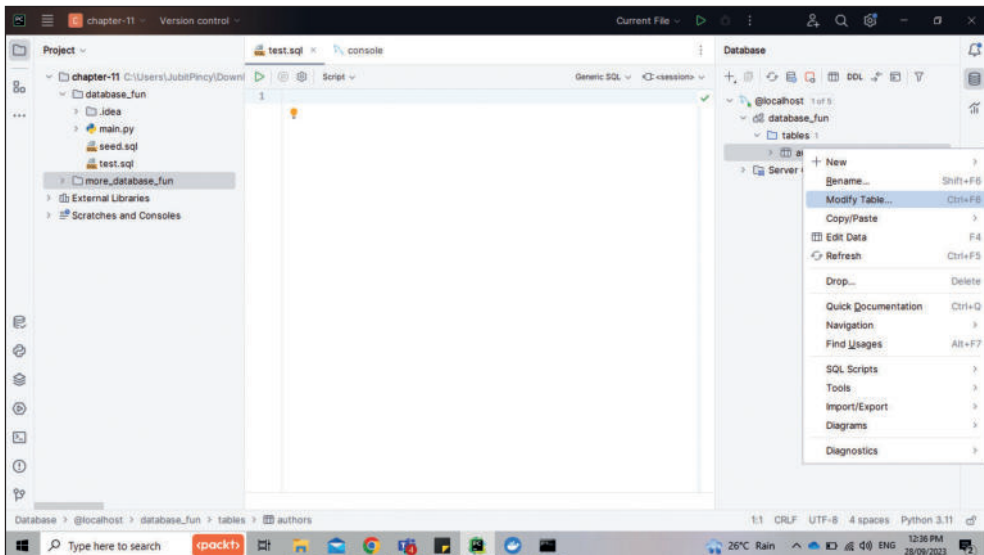


Рис. 11.30. Вы можете изменить существующие таблицы, кликнув таблицу правой кнопкой мыши

При этом вы получите окно, подобное тому, которое показано на рис. 11.31.

На рис. 11.31 я добавил столбец `date_of_birthd` с типом даты. Как вы можете видеть в окне предварительного просмотра, PyCharm генерирует оператор `alter`, а не пытается удалить и воссоздать таблицу, как это делают многие редакторы баз данных. Если вы хотели удалить таблицу, в меню, показанном на рис. 11.30, была опция для этого.

Генерация скриптов

PyCharm имеет очень мощную утилиту для создания всех видов SQL-скриптов. Поскольку мы говорим здесь о DDL, имеет смысл научиться генерировать полный скрипт SQL, который создаст таблицу, над которой мы работаем.

Кликните правой кнопкой мыши базу данных, как показано на рис. 11.32, и выберите пункт меню **SQL Scripts | SQL Generator...**

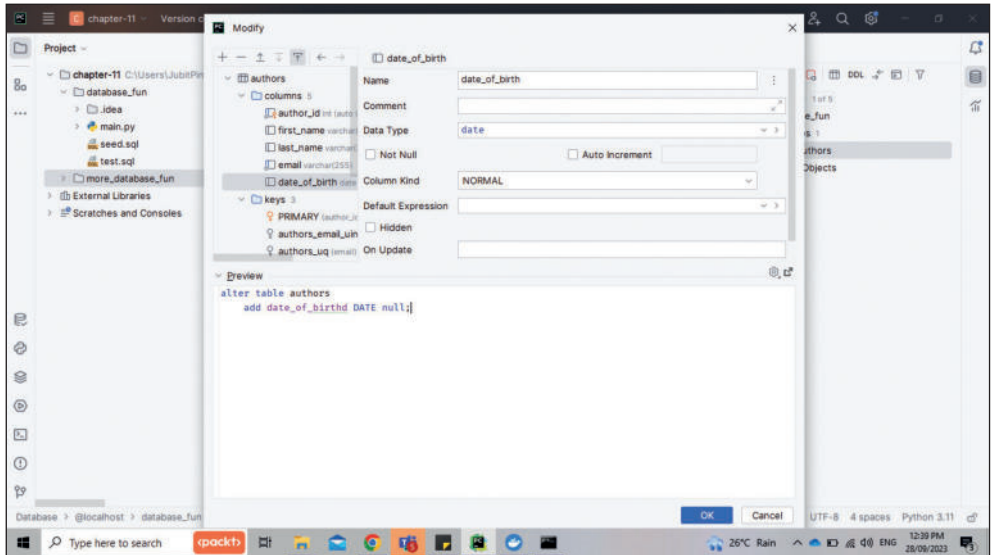


Рис. 11.31. Изменения, которые вы вносите здесь, генерируются как оператор `alter`, что является лучшей практикой при разработке баз данных

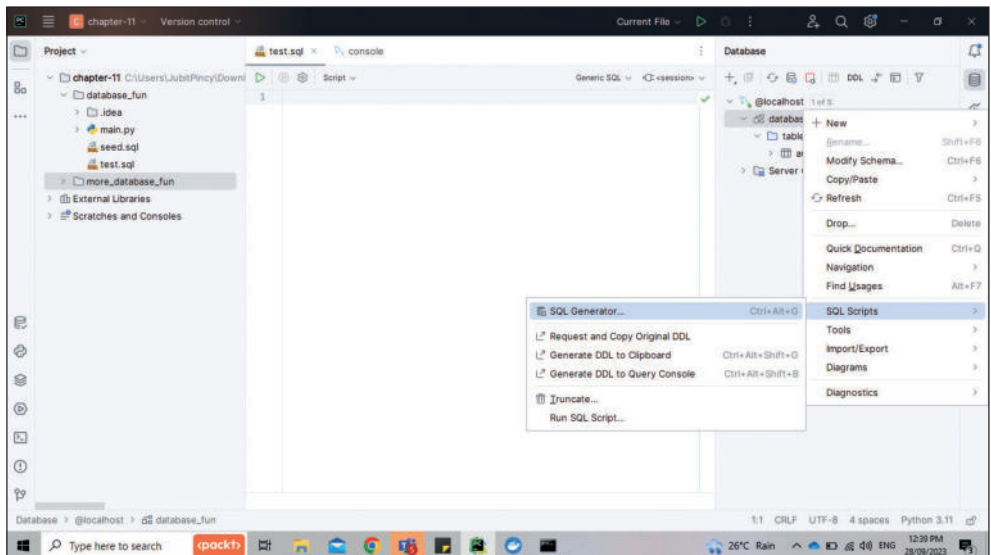


Рис. 11.32. SQL-генератор создает скрипт для нашей базы данных

Панель слева позволяет установить несколько интересных опций. Когда вы нажмете на них, сгенерированный SQL изменится, если это сработает. Я говорю «если это сработает», потому что не каждая платформа базы данных поддерживает все параметры синтаксиса в списке. Мы используем MySQL, ко-

торый поддерживает синтаксис `USE CREATE IF NOT EXISTS`, но не поддерживает синтаксис `CREATE OR REPLACE`, который вы встретите в Postgres или Oracle. Щелчок по синтаксису, что не поддерживается, просто не меняет синтаксис в окне предварительного просмотра.

Если скрипт DDL соответствует вашим потребностям, можете использовать кнопки справа (обведены на рис. 11.32), чтобы скопировать, сохранить или запустить скрипт в окне запроса.

Создание диаграммы базы данных

Традиционные методы работы с базами данных обычно предполагают наличие проектного документа, такого как диаграмма схемы базы данных. В старые добрые времена мы использовали специальные инструменты для построения диаграмм. PyCharm имеет встроенный инструмент для построения диаграмм, и самое приятное то, что диаграмма создается на основе структуры базы данных, а не рисуется с нуля.

Чтобы создать диаграмму, просто кликните правой кнопкой мыши свою базу данных, найдите пункт меню **Diagrams** и выберите любой вариант, показанный на рис. 11.33.

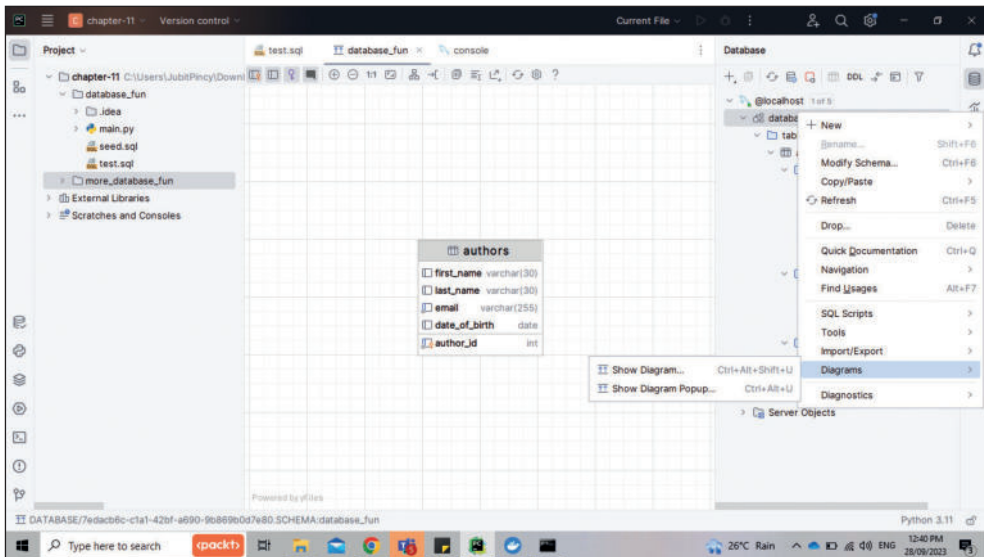


Рис. 11.33. Пункты меню **Diagrams** показаны вместе со сгенерированной диаграммой

Единственное различие между этими двумя вариантами заключается в том, что один рисует диаграмму непосредственно в области содержимого IDE, а другой создает ее во всплывающем окне. Всплывающее окно полезно для просмотра диаграммы на другом мониторе. Вы можете перемещаться по диаграмме с помощью инструментов масштабирования, а также кликая правой кнопкой мыши по диаграмме во время перетаскивания.

На верхней панели инструментов вы также найдете параметры экспорта для экспорта диаграммы в файлы изображений, а также файлы данных, которые можно импортировать в специальные инструменты построения диаграмм.

Запрос к источнику данных с помощью SQL

Запросы к базе данных, вероятно, являются второй наиболее полезной функцией встроенных в IDE инструментов для работы с базами данных. Как и многие другие функции PyCharm, эта гарантирует, что вам никогда не придется покидать PyCharm, пока вы не закончите свою работу.

Есть несколько мест, где можно выполнять запросы. Вы можете запускать специальные запросы в консоли запросов или напрямую из файлов `.sql`.

Специальные запросы

Специальные запросы (ad hoc) – это просто запросы для непосредственной цели. Специальный запрос имеет ряд характеристик.

- **Незапланированность.** Специальные запросы не являются частью предопределенного набора запросов. Они пишутся на месте для удовлетворения конкретной потребности.
- **Временность.** Эти запросы используются для получения данных для конкретной задачи или ситуации и не сохраняются для использования в будущем. Они не являются частью вашего приложения, хотя после некоторых экспериментов и настроек вы можете формализовать их в своем коде.
- **Отсутствие оптимизации.** Специальные запросы могут быть неоптимизированы с точки зрения эффективности, поскольку они быстро собираются при отсутствии времени для точной настройки.
- **Вариативность.** Синтаксис и структура специальных запросов могут различаться в зависимости от знаний и опыта пользователя.
- **Они не сохраняются.** В отличие от хранимых процедур или представлений специальные запросы не сохраняются в базе данных, как, скажем, представление или хранимая процедура. Это означает, что их нельзя будет вызвать или выполнить снова позже, если вы не сохраните их в файл.

Специальные запросы выполняются в консоли базы данных. Окно консоли появляется автоматически, когда вы завершаете подключение к источнику данных, но в интерфейсе есть много мест, где вы можете создать новое представление консоли. Посмотрите на рис. 11.34, и вы увидите окно источника данных. Кнопка с надписью **QL (Язык запросов)** обведена кружком. Здесь не говорится «SQL», потому что, помните, PyCharm также поддерживает базы данных NoSQL, такие как MongoDB, Apache Cassandra и Redis.

Кнопка **QL** чувствительна к базе данных, выбранной в окне источника данных, поэтому, если у вас более одного источника, можете открыть источник запроса к любому из них, выбрав их в окне источника данных и затем нажав кнопку **QL**. Я фактически закрыл консоль, которая автоматически запускалась при создании источника данных. Чтобы создать новую консоль, я нажму кнопку **QL** и выберу **New Console**.

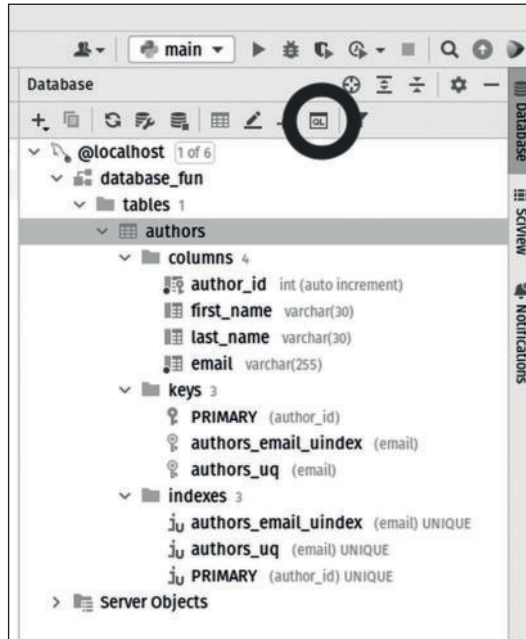


Рис. 11.34. В любом месте, где есть значок QL, подобный этому, вы можете запустить окно консоли и запросить свой источник данных, используя специальный запрос

Первая консоль, которую вы откроете, станет **default console**. PyCharm отслеживает ваши консоли в папке **Scratches and Consoles**, как показано на рис. 11.35.

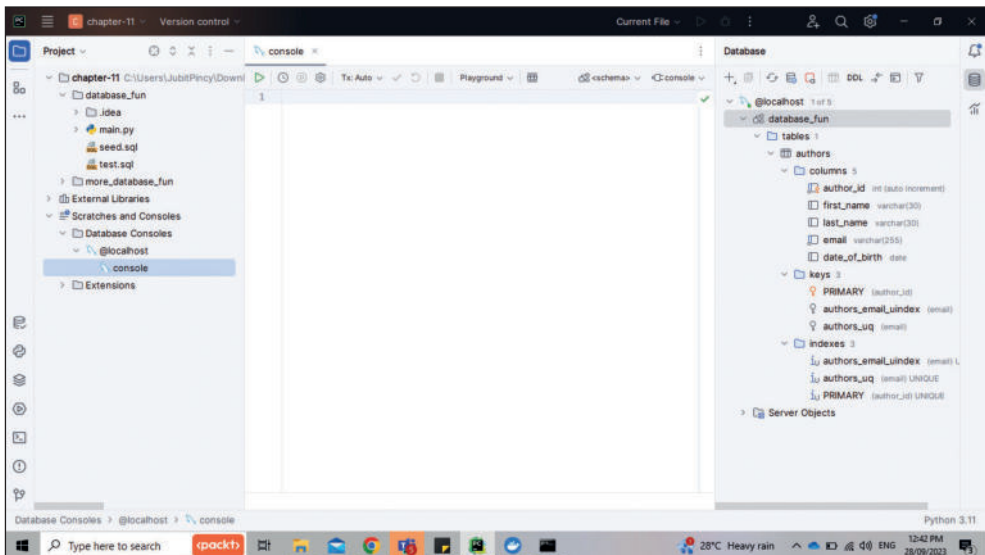


Рис. 11.35. PyCharm отслеживает консоли вместе со временными файлами в специальной папке в IDE

Если вы ввели какой-либо SQL-код в консоли, PyCharm сохранит содержимое даже между сеансами. Пустые консоли исчезнут при выходе из PyCharm. Это удобно для отслеживания вашей специальной работы.

Генерация операторов SQL

Нам нужно создать несколько исходных записей, которые в конечном итоге будут сохранены как исходный скрипт (seed script). Исходный скрипт – это скрипт, который можно использовать для тестирования приложения путем заполнения базы данных некоторыми начальными тестовыми данными. Мы отходим от операторов строго DDL, которые использовали до сих пор. Мы собираемся использовать больше DML, т. е. операторов, используемых для работы с данными, а не со структурой базы данных. Начальные скрипты также полезны для заполнения таблиц данных статическими данными, которые редко меняются.

Мы собираемся заполнить нашу таблицу `authors` несколькими записями, но PyCharm будет генерировать за нас большую часть SQL. Начните с создания консоли, если она у вас еще не открыта. Кликните правой кнопкой мыши таблицу `authors`, наведите указатель мыши на **SQL Scripts** и найдите **Insert rows into a table**. Посмотрите на рис. 11.36, чтобы увидеть это в действии.

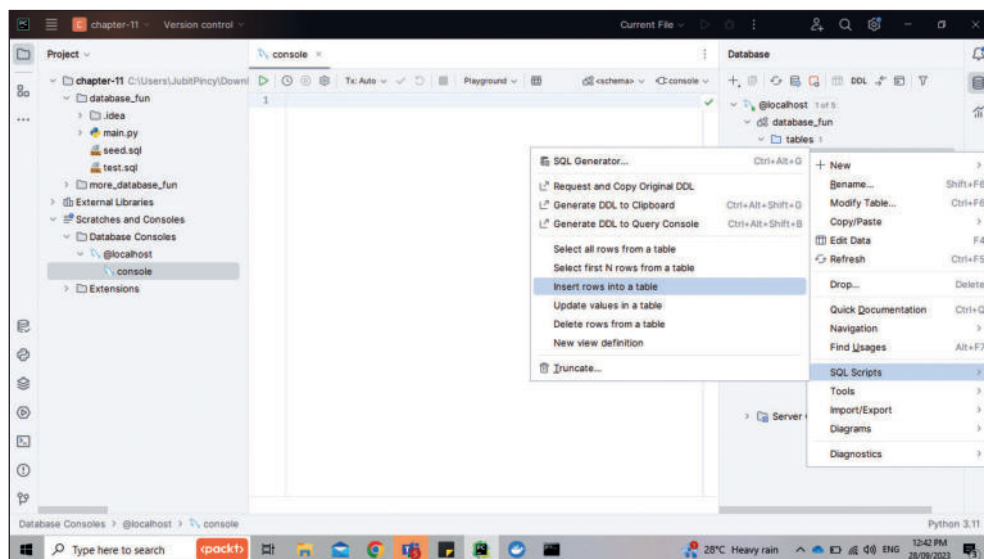


Рис. 11.36. PyCharm автоматически сгенерирует для вас запросы DML!

Выбрав эту опцию, вы обнаружите, что оператор `insert` был вставлен (о да!) прямо в консоль. Можете увидеть мой вариант на рис. 11.37.

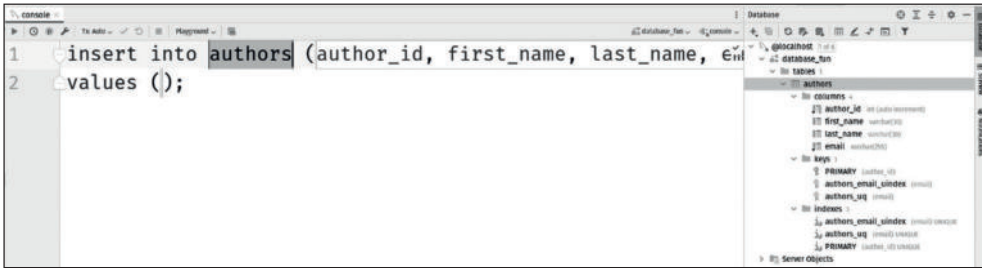


Рис. 11.37. PyCharm сгенерировал оператор insert для выбранной нами таблицы

Обратите внимание на ограничивающую рамку вокруг текста. Мы видели это раньше. Вы смотрите на шаблон! Это означает, что мы можем переходить с места на место в шаблонном тексте, поэтому вам не следует возиться с текстом, как это сделал я.

Предупреждение о copy-paste!

Если у вас есть электронная копия этой книги, позволяющая копировать и вставлять ее из самой электронной книги, будьте осторожны с любым кодом, который использует одинарные кавычки, как это делает следующий оператор SQL. В процессе редактирования книги они часто превращаются в специальные символы, которые не будут работать, если вставить их в консоль. Убедитесь, что одинарные кавычки остались одинарными кавычками!

Операторы insert можно разделить на две половины:

```
insert into authors (first_name, last_name, email)
values ('Bruce', 'Van Horn', 'bruce@test.com');
```

Половинки, о которых я говорю, – это те части утверждения, которые не являются шаблонными, а именно части, заключенные в два набора круглых скобок. Первый набор определяет, какие поля вы будете заполнять, а второй набор – это значения. Необходимо согласовать порядок и тип полей в первой половине при вводе значений для каждого поля. Нажав клавишу *Tab*, вы будете перемещаться между двумя половинами оператора insert, что позволит вам изменить его по своему усмотрению.

Нетрадиционный сгенерированный SQL

PyCharm по умолчанию не генерирует ключевые слова SQL, написанные заглавными буквами, что является обычным соглашением. Вы могли заметить, что все операторы, которые я печатаю, написаны заглавными буквами, а сгенерированные – нет. Мне пришлось побороть искушение исправить это ради книги, но мне также нужно объяснить, почему в этой игре присутствуют несоответствия.

PyCharm создал заполнитель для первичного ключа – поля, которое вы обычно не заполняете, поскольку база данных заполняет его автоматически. Идите дальше и удалите это первое поле из набора полей, заданного в первых круглых скобках, как я сделал в предыдущем коде.

Далее перейдите в раздел `values` и начните заполнять значения. Помните, что в SQL значения `varchar` (string) должны быть заключены в одинарные кавычки. Python позволяет использовать как одинарные, так и двойные кавычки, а SQL – нет. Когда вы вводите значения, PyCharm будет подсказывать, какое поле соответствует значению. Это делается с помощью всплывающей подсказки над текстом в редакторе, а также цветовой маркировкой текущего поля в верхней половине оператора (см. рис. 11.38, чтобы понять, что я имею в виду).



Рис. 11.38. PyCharm предоставляет всплывающие подсказки для поля и типа, которые вы сейчас заполняете, во второй половине запроса

Прекрасно! Сколько раз вы писали длинный оператор `insert` только для того, чтобы обнаружить, что у вас больше значений, чем полей? Или, что еще хуже, вы обнаружили, что переставили несколько полей и по ошибке ввели кучу мусорных данных? PyCharm предоставляет вам визуальный индикатор, позволяющий убедиться, что вы точно знаете, какое поле вы вводите, показывая всплывающую подсказку с полем и типом, который вы заполняете. Он также затемняет поле в верхней половине оператора SQL.

Запуск запроса

Вероятно, вам не нужно это говорить, потому что на данный момент вы, скорее всего, уже видите зеленые стрелки запуска там, где уместно что-то запустить. Можете запустить содержимое консоли, кликнув зеленую стрелку вверх вкладки консоли в IDE. Вы также можете выделить часть содержимого консоли и выполнить часть запроса.

Измените содержимое вашей консоли, чтобы оно соответствовало следующему коду:

```
insert into authors (first_name, last_name, email)
values ('Bruce', 'Van Horn', 'bruce@test.com');
insert into authors (first_name, last_name, email)
values ('Kinnari', 'Chohan', 'kinnari@test.com');
insert into authors (first_name, last_name, email)
values ('Prajakta', 'Naik', 'prajakta@test.com');
insert into authors (first_name, last_name, email)
values ('Jubit', 'Pincy', 'jubit@test.com');
```


Все, что я сделал, – это скопировал один и тот же SQL четыре раза, затем изменил имена для небольшого разнообразия и чтобы поблагодарить мою команду Packt (привет!). Далее выделите первый оператор и запустите запрос. Откроется панель **Services**, в которой будут показаны результаты запроса. Как видно на рис. 11.39, была вставлена только одна запись.

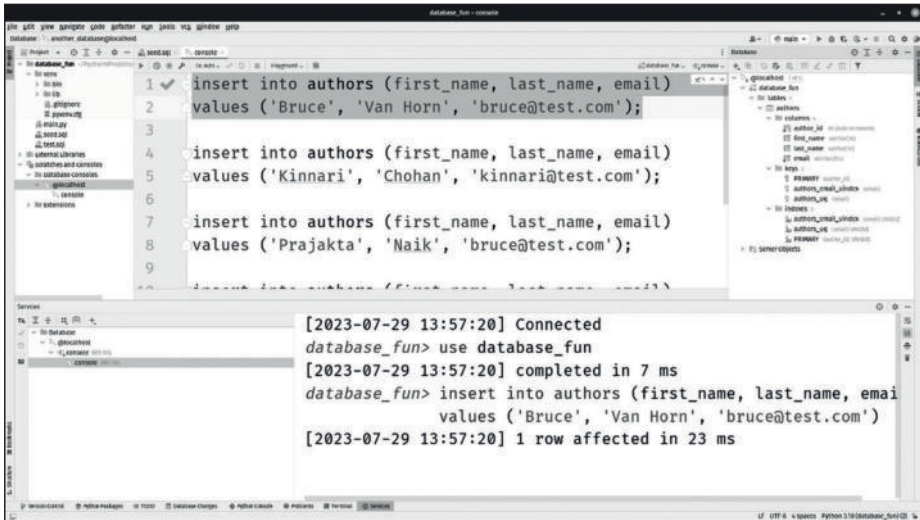


Рис. 11.39. Вы можете выполнить отдельные инструкции, выбрав их и нажав кнопку Run

Если вы выберете остаток и запустите еще раз, то обнаружите три вставки.

Теперь, когда у нас есть некоторые данные, давайте выполним запрос с помощью оператора SELECT. Добавьте этот код в консоль:

```
SELECT * FROM authors;
```

Я мог бы сгенерировать select, но этот запрос короткий. Я хотел показать вам, как PyCharm отображает результаты. Взгляните на рис. 11.40.

Это плотная панель! Здесь можно многое сделать. По умолчанию вы видите табличное представление данных (1). Можете дважды кликнуть ячейки и редактировать данные на месте. Вы также можете добавлять и удалять строки, используя интерфейс (2). После того как вы отредактировали данные по своему вкусу, можете отправить свои изменения с помощью маленькой зеленой стрелки вверх на панели инструментов (2) на рис. 11.40.

Если вы предпочитаете изменить представление вывода с табличного на необработанный текст или древовидное представление (что имеет больше смысла для запроса иерархических данных), можете кликнуть значок глаза на панели инструментов (2) на рис. 11.41.

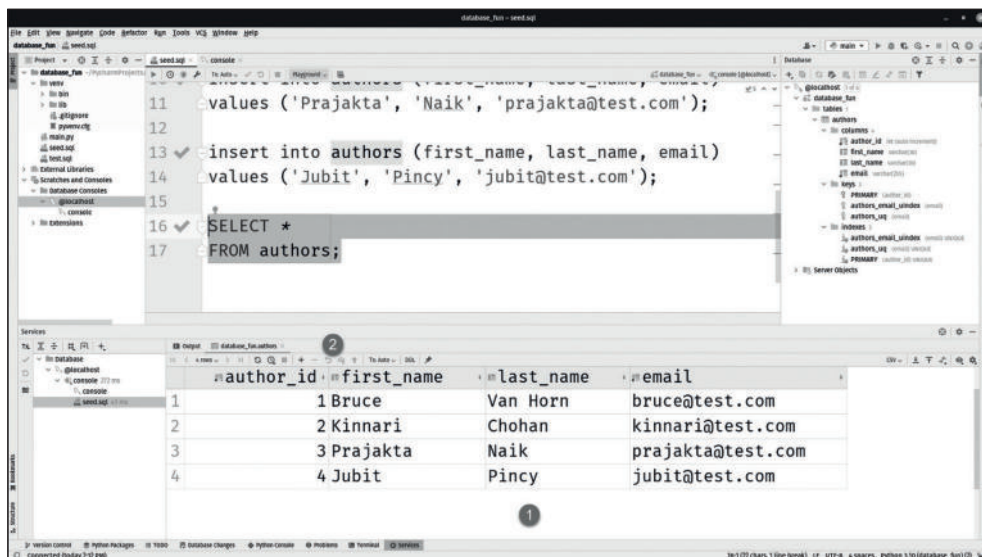


Рис. 11.40. По умолчанию операторы select выдают красивый табличный результат

Экспорт результатов запроса

Можно экспортировать результаты операторов select, используя кнопку экспорта (рис. 11.41). На рис. 11.41 показано, как это выглядит.

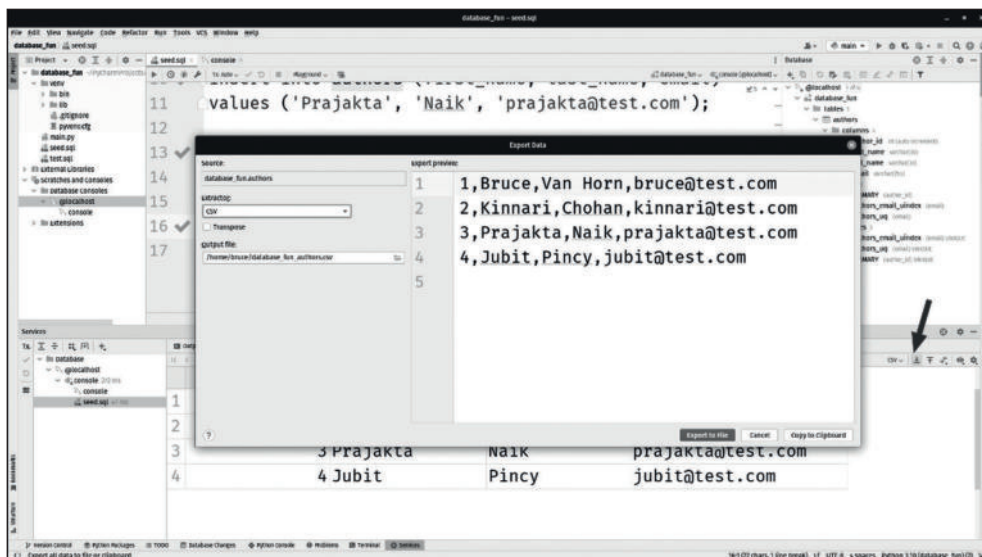


Рис. 11.41. Экспортировать данные в PyCharm легко

Формат экспорта по умолчанию – **значения, разделенные запятыми (CSV)**, что является распространенным форматом для обмена табличными данными. PyCharm поддерживает десятки других полезных форматов, которые вы найдете в раскрывающемся списке **Extractor**. Измените выходной файл на то, что вам нравится, затем нажмите кнопку **Export to file**.

Работа с файлами SQL

PyCharm поддерживает файлы с расширением `.sql` или `.ddl`. Как и другие признанные типы файлов, вы получите автозаполнение, подсветку синтаксиса и т. д. Также можете запускать инструкции непосредственно в файле SQL, как мы это делаем в консоли.

Давайте увековечим нашу консольную работу в виде полноценного скрипта. Создайте новый файл в PyCharm под названием `seed.sql`. Затем добавьте эту строку в начало файла скрипта:

```
USE database_fun;  
TRUNCATE TABLE authors;
```

Оператор `USE` специфичен для MySQL и гарантирует, что перед выполнением операторов SQL выбрана база данных `Database_fun`. Оператор `TRUNCATE` удалит все записи в таблице `authors` и сбросит последовательность автоинкрементатора в таблице обратно к ее исходному значению. Это полезно для тестирования разработки, поскольку теперь исходный скрипт может сбрасываться до чистого набора тестовых значений. Только будьте осторожны и никогда не запускайте исходный скрипт в рабочей среде!

Затем вырежьте и вставьте содержимое консоли под оператором `TRUNCATE`. Обратите внимание, я сказал «вырезать». Вы хотите, чтобы консоль была пустой. Скрипт должен выглядеть так:

```
USE database_fun;  
TRUNCATE TABLE authors;  
insert into authors (first_name, last_name, email)  
values ('Bruce', 'Van Horn', 'bruce@test.com');  
  
insert into authors (first_name, last_name, email)  
values ('Kinnari', 'Chohan', 'kinnari@test.com');  
  
insert into authors (first_name, last_name, email)  
values ('Prajakta', 'Naik', 'prajakta@test.com');  
  
insert into authors (first_name, last_name, email)  
values ('Jubit', 'Pincy', 'jubit@test.com');  
  
SELECT * FROM authors;
```

Сохраните исходный скрипт и запустите его. Он должен выполниться успешно.

КРАТКОЕ СОДЕРЖАНИЕ

У меня ужасное ощущение, что я не все описал. Это потому что так и есть. Я мог бы написать целую книгу только о функциях баз данных PyCharm, но, если бы я это сделал, я мог бы назвать ее как-нибудь вроде «Практическое программирование баз данных с помощью DataGrip». Не забывайте, что мы видим совершенно другую IDE, запихнутую в щели PyCharm, так же, как мы это делали с WebStorm в главе 7. Набор функций поистине ошеломляет, и я не рассмотрел некоторые действительно интересные функции либо из-за нехватки места в этой книге, либо потому, что некоторые из действительно интересных функций все еще находятся в стадии разработки.

Например, вы можете установить код в качестве источника данных и синхронизировать его со структурой базы данных. На данный момент он не завершен на 100 %, поэтому я его не включил. Как разработчик баз данных, я ожидаю, что смогу получить двусторонний опыт, при котором изменения в коде отражаются в базе данных, а изменения в базе данных копируются обратно в код. Я думаю, мы уже в пути.

Однако мы сделали довольно полный охват как для полнофункциональной веб-разработки, так и для специалистов по обработке данных, которым необходимо использовать несколько платформ баз данных и несколько парадигм хранения данных.

Мы начали с небольшого экскурса в историю. Мне доставило удовольствие осветить историю в этой и предыдущих главах, потому что за плечами преимущество долгой карьеры в этих областях. Большую часть того, что я описываю, я делал своими руками, и у меня было место в первом ряду. Я учился в первом классе, когда Э. Ф. Кодд формализовал реляционную алгебру. Что-то во мне хочет похвастаться тем, что я изучал его ранние работы, но, по правде говоря, я был большим поклонником субботнего утреннего телешоу «Лесси и спасатели» в своей футбольной пижаме, чем операторов SELECT. Но, когда я стал старше, мне пришлось работать на мейнфреймах IBM, так что в конце концов я наверстал упущенное.

Переходя к более практическим вопросам, мы узнали, что PyCharm, используя отдельно установленные драйверы JDBC, может подключаться к десяткам различных платформ баз данных. Мы узнали, что он поддерживает не только реляционные базы данных, но и NoSQL. После того как мы узнали, как открыть инструменты базы данных и подключить наш источник данных, мы научились проектировать базу данных, используя множество инструментов, которые генерируют для нас DDL.

Затем мы перешли к использованию PyCharm для создания DML, который будет способствовать работе, как только наши таблицы будут заполнены данными. Наконец, после работы в консоли над созданием первоначальной работы мы сохранили наш SQL в файл `.sql`.

Расположение этой главы в книге не случайно. Я воздерживался от большого количества кода в полностекowych фреймворках, которые мы рассмотрели в последних нескольких главах. В реальной жизни к таким проектам наверняка будут подключены базы данных. Хотя многие разработчики полагаются на ORM для выполнения работы с базой данных, никогда не помешает иметь

хотя бы возможность проверять базу данных и ее содержимое напрямую с помощью консоли.

Если вы, как и я, полностью отказались от ORM, теперь у вас есть недостающий фрагмент головоломки. Можете создать код базы данных, используя замечательный набор инструментов, не выходя из PyCharm.

В этой главе также предложено логическое разделение, поскольку в следующих главах начинает рассматриваться набор функций PyCharm Professional для обработки данных. Имеет смысл только предварять этот набор глав описанием этих наиболее распространенных механизмов хранения и поиска данных.

В следующей главе мы включим *научный режим*! Заберите свой лабораторный халат из химчистки, возьмите с собой большую чашку $C_8H_{10}N_4O_2$ ¹, и увидимся в следующей главе!

ДАЛЬНЕЙШЕЕ ЧТЕНИЕ

- Инструкции по установке Docker Desktop: <https://www.docker.com/products/docker-desktop/>.
- Использование KeePass в PyCharm : <https://www.jetbrains.com/help/pycharm/reference-ide-settings-password-safe.html>.
- Codd, E. F. (1983). A relational model of data for large shared data banks. *Communications of the ACM*, 26(1), 64-69.
- Forta, B. (2013). *Sams teach yourself SQL in 10 minutes*. Pearson Education.
- Hernandez, M. J. (2013). *Database Design for Mere Mortals: A Hands-On Guide to Relational Database Design*. Pearson Education.
- Pettit, T. and Cossetino, S. (2022). *The MySQL Workshop*. Packt Publishing.
- Poulton, N. (2023). *Docker Deep Dive – 2nd Edition*. Packt Publishing.
- Viescas, J. L. and Hernandez, M. J. (2014). *SQL Queries for Mere Mortals: A Hands-On Guide to Data Manipulation in SQL*. Pearson Education.

¹ Кофеин. – Прим. ред.

Часть IV

Обработка данных с помощью PyCharm

Как и предыдущая часть, эта часть книги посвящена конкретному применению программирования на Python, на этот раз анализу данных и науке о данных. Читатели смогут использовать PyCharm и его функции для эффективной работы над своими проектами по науке о данных и научным вычислениям.

Эта часть состоит из следующих глав:

- глава 12 «Включаем научный режим»,
- глава 13 «Динамический просмотр данных с помощью SciView и Jupyter»,
- глава 14 «Создание конвейера данных в PyCharm».

Включаем научный режим

Добро пожаловать в четвертый раздел книги. Одна из моих любимых франшиз видеоигр – «Цивилизация» Сиды Мейера. На прохождение игры, в которой вы должны стать главой цивилизации на заре истории, уходит много дней. Вы продвигаетесь через доисторические эпохи, через Средневековье, эпохи Возрождения и возникновения промышленности. С каждой новой эпохой возникают уникальные проблемы, поскольку игроки соперничают за различные аспекты глобального господства. Игра реально ускоряется, когда игроки достигают информационного века. Я просто слышу, как мой редактор говорит: «Отложи видеоигру и пиши быстрее!» Хорошо, я послушаюсь. Говоря о книге, я чувствую, что с этой главой мы достигли аналогичной вехи. Мы прошли долгий путь с момента нашего первого сообщения *hello world* в файле `main.py` в первых главах этой книги.

PyCharm нацелен на разработчиков Python, но PyCharm Professional на самом деле ориентирован на две группы: веб-разработчиков и специалистов по обработке данных. Последнюю группу, похоже, выделить немного сложнее, поскольку у JetBrains есть множество продуктов для анализа данных. Помимо PyCharm Professional, JetBrains также создает отдельную IDE под названием **DataSpell**, которая позиционируется как *IDE для профессиональных специалистов по данным*. Что это дает?

Я задавался вопросом о том же. Согласно часто задаваемым вопросам по DataSpell, PyCharm должен поддерживать разработчиков программного обеспечения, которые занимаются изучением данных в рамках своей работы в качестве разработчиков программного обеспечения. DataSpell – для профессионалов, которые не являются разработчиками, а просто занимаются аналитикой. В плане функций есть некоторое пересечение, но, в отличие от того, что мы видели в WebStorm и DataGrip, которые полностью встроены в PyCharm, у DataSpell есть функции, которых нет в PyCharm, и нет планов это менять. Они нацелены на две разные группы профессионалов.

Поддержка научных и аналитических рабочих процессов предполагает включение научного режима в PyCharm. Это открывает новые макеты пользовательского интерфейса и панели инструментов, которые обычно не открываются при отключенном научном режиме. В этой главе мы рассмотрим следующие темы:

- запуск научного проекта в PyCharm,
- расширенные возможности научных проектов PyCharm.

К концу этой главы вы поймете, как эти функции могут повысить производительность в проектах научных вычислений. Эта глава послужит общим обсуждением различных инструментов, предлагаемых PyCharm, и поможет вам понять научные вычисления и то, как эти инструменты интегрируются и работают друг с другом.

ТЕХНИЧЕСКИЕ ТРЕБОВАНИЯ

Чтобы пройти эту главу, как и остальную часть книги, вам понадобится следующее:

- поскольку мы переходим к теме науки о данных, я переключу свой любимый дистрибутив Python на Anaconda, который представляет собой дистрибутив Python, адаптированный для рабочих нагрузок науки о данных. Вы можете найти его вместе с инструкциями по установке для вашей операционной системы по адресу <https://anaconda.com>;
- аналогично вместо обычного `pip` я буду использовать `conda` – менеджер пакетов Anaconda. Он установлен рядом с Анакондой;
- установленная и рабочая копия PyCharm. Установка была описана в главе 2.

Пример исходного кода этой книги с GitHub. Мы рассмотрели клонирование кода в главе 2. Соответствующий код этой главы вы найдете по адресу <https://github.com/PacktPublishing/Hands-On-Application-Development-with-PyCharm---Second-Edition/tree/main/chapter-12>.

СОЗДАНИЕ НАУЧНОГО ПРОЕКТА В PYCHARM

К настоящему моменту вы хорошо знакомы с диалоговым окном **New Project** в PyCharm. Давайте создадим научный проект! Нажмите **File | New Project**, и вы найдете шаблон под названием **Scientific**, как показано на рис. 12.1.

Ранее я упоминал, что в этом разделе книги я буду использовать Anaconda, потому что именно ее использует большинство профессионалов в области науки о данных. Единственная интересная настройка, помимо изменения в интерпретаторе, – это настройка **Data folder** в разделе **More Settings**. Она устанавливает папку в нашем проекте, который скоро будет создан и который будет хранить, как вы уже догадались, данные. Подробнее об этом после создания проекта. Идите вперед и нажмите **Create**.

После завершения процесса создания вы увидите настройку, подобную моей на рис. 12.2:

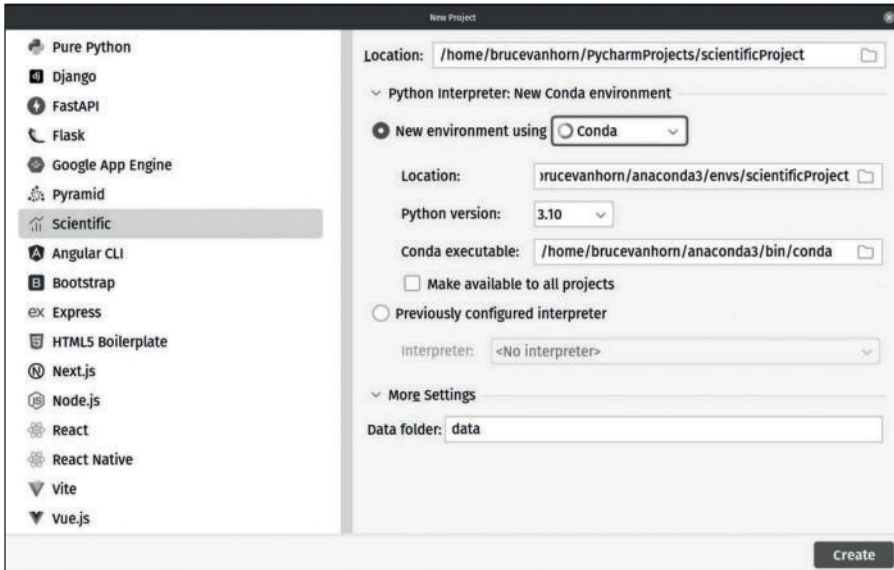


Рис. 12.1. Отойдите! Мы собираемся заняться НАУКОЙ!

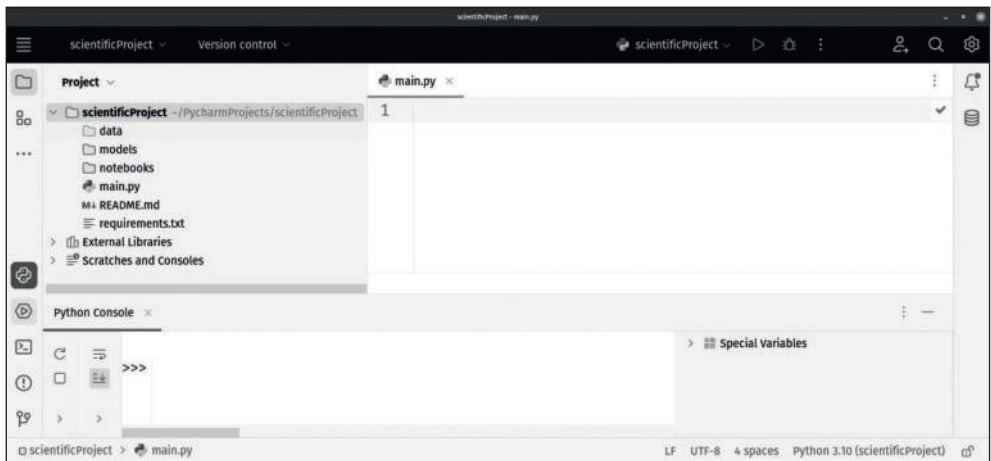


Рис. 12.2. Вот! Мы создали научный проект в PyCharm!

Давайте поговорим о том, что только что произошло. Мы сгенерировали проект, как обычно. У нас есть папка `data`, которая должна быть особенной, потому что ее цвет отличается от остальных. У нас появилось больше пустых папок для тетрадей и моделей. У нас есть файл `main.py`, файл `requirements.txt` и файл Markdown с именем `README.md`, все они пусты. Обычно мы получаем шаблонный код, но не в этот раз.

Каждый файл и папка служат определенной цели. Как я уже упоминал, папка данных должна быть особенной, поскольку она другого цвета. Моя – выделена желтым цветом. Эта папка важна, поскольку она предназначена для хранения файлов данных для вашего научного анализа. Поскольку это не код и ваши

файлы данных могут быть очень большими, PyCharm помечает эту папку как специальную для проекта. Возможно, вы заметили, что папка шаблонов в приложениях Flask и Django также окрашена иначе, чем остальные. Во многих типах проектов существуют специальные папки.

Некоторые подсказки по этому поводу мы можем получить в свойствах проекта. У меня есть окно свойств проекта, что показано на рис. 12.3:

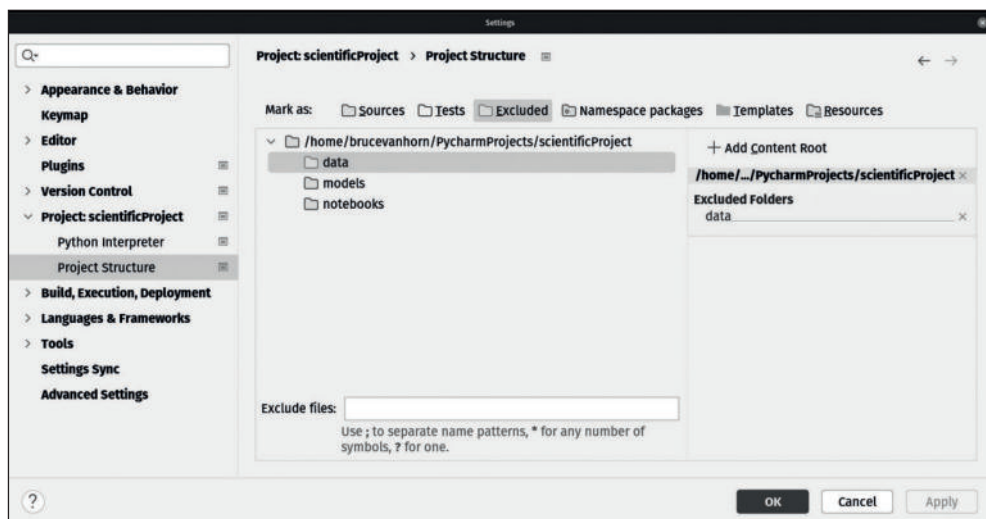


Рис. 12.3. Вы можете установить несколько типов специальных папок во многих типах проектов в PyCharm, включая научные проекты

В этом случае папка данных исключается из остальной части проекта. PyCharm не будет каким-либо образом индексировать или сканировать эту папку. Нам это нужно для папки данных, поскольку файлы данных могут быть очень большими. В этой папке нет кода, поэтому на самом деле нет смысла сканировать ее с целью предоставления обычной помощи и информации, которую вы ожидаете от файлов кода. Можете даже подумать о том, чтобы ваша система контроля версий также игнорировала эту папку, поскольку может быть нецелесообразно возвращать ваши данные в репозиторий по ряду причин, начиная от проблем конфиденциальности, в зависимости от характера данных, а также размера файла. У Git есть ограничения на размер файлов, которые он может обрабатывать, а GitHub ограничит размер вашего репозитория. Папка данных – очевидное раздувание, которое, вероятно, не обязательно должно быть в репозитории.

Разгадав эту загадку, давайте посмотрим на файл `requirements.txt`. Мы видели это раньше. Этот файл используется проектом Python независимо от того, какую IDE вы используете, для отслеживания требований библиотеки для проекта. Каждый раз, когда вы используете стороннюю библиотеку из PyPi, необходимо включить ее в `requirements.txt`. Этот файл следует вернуть в ваш репозиторий, чтобы другие разработчики могли скопировать вашу виртуальную среду. Файл пуст, а инструменты, связанные с файлом `requirements.txt`, мы видели в главе 3.

Здесь есть нечто, чего мы раньше не видели: файл README.md. Расширение .md указывает, что это файл Markdown. Markdown – альтернатива HTML, языку разметки. Видите, что они там сделали? Markdown позволяет создать форматированный текстовый файл без каких-либо накладных расходов на HTML.

Избавившись от накладных расходов, вы также потеряете много возможностей. Markdown используется только для создания файлов документации. Если вы когда-либо просматривали репозиторий на GitHub и у проекта была привлекательная целевая страница, вы на самом деле смотрите на файл README.md, преобразованный в HTML с помощью GitHub и вашего браузера. На рис. 12.4 показан пример красивой страницы README.md для одного из моих репозиториях:



Рис. 12.4. Файл README.md отображается как целевая страница на GitHub и используется для документирования содержимого репозитория

Для этого проекта PyCharm создал пустой файл README.md. Файл main.py – это обычный старый файл Python. На самом деле это очень минимальный шаблон проекта.

ДОПОЛНИТЕЛЬНАЯ КОНФИГУРАЦИЯ для НАУЧНЫХ ПРОЕКТОВ в PYCHARM

Готовая конфигурация PyCharm обычно очень полная. Однако для научных проектов вы можете добавить в нее несколько плагинов. Далее в книге будет целая глава, посвященная замечательным плагинам, но они вам понадобятся прямо сейчас, так что давайте посмотрим на них.

Плагины устанавливаются с помощью PyCharm Marketplace. Вы можете попасть в Marketplace, используя значок шестеренки, который обычно используется для доступа к настройкам. Вы узнаете это на рис. 12.5:

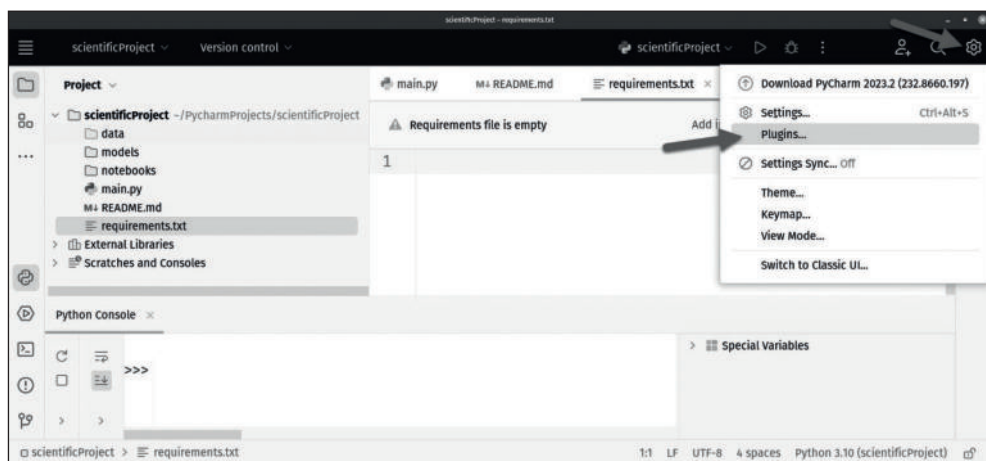


Рис. 12.5. Пункт меню Plugins приведет вас к окну плагинов, включающему PyCharm Marketplace

Откройте окно **Plugins**. Вы можете увидеть мое на рис. 12.6:



Рис. 12.6. Рынок Marketplace позволяет найти всевозможные плагины для PyCharm и других IDE JetBrains

Обратите внимание, что вы попадете прямо на вкладку **Marketplace**. Это напоминает мне о неизбежном походе в сувенирный магазин в конце каждой экскурсии по музею, в которой я когда-либо участвовал. Справа есть вкладка, позволяющая увидеть, что уже установлено. Хотя это скучно, да? Более заманчиво просмотреть то, чего у нас еще нет. Многие плагины здесь бесплатны, а некоторые нет. Аналогичным образом многие из представленных здесь плагинов хорошо документированы и поддерживаются разработчиками, которые тщательно разрабатывают свои предложения на торговой площадке, а некоторые представляют собой лишь краткие описания с заголовком. Итак, пройдемся по магазинам!

Плагины Markdown

Наша первая остановка – хороший плагин Markdown¹. Markdown – это не совсем ракетостроение, хотя мы и находим его в научном проекте. Плагин Markdown сделает разметку вашего кода, и вы сможете увидеть, как он будет выглядеть в форматированном виде.

Чтобы найти плагин Markdown, вам необходимо выполнить поиск, используя поле **Search**, указанное на рис. 12.7:

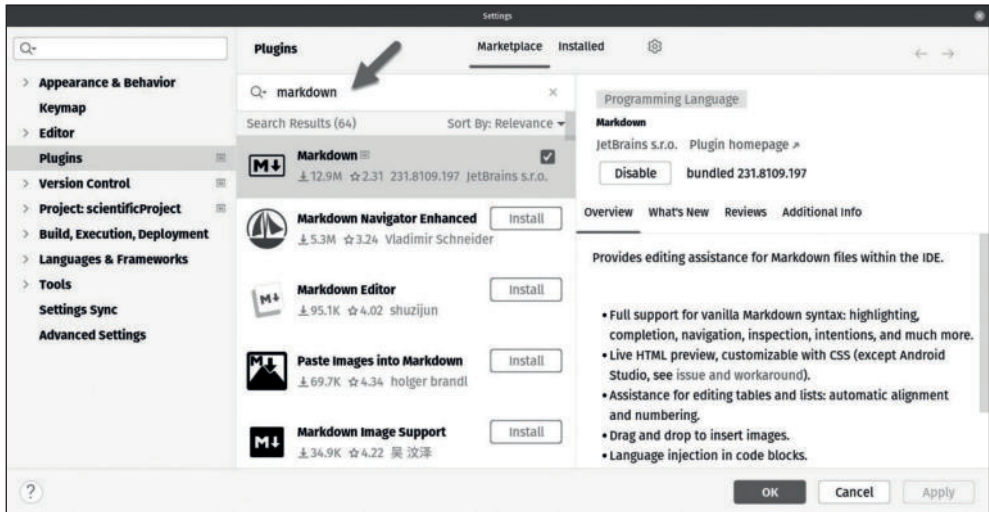


Рис. 12.7. Ищите плагины здесь и избегайте соблазна поиска в окне, которое помогает вам найти настройки

Так получилось, что JetBrains теперь включает плагин Markdown в стандартную установку. Они не всегда это делали. Вы можете видеть, что есть альтернативы для изучения. Я буду использовать только тот, что в комплекте. Если у вас более старая версия PyCharm, возможно, вам придется установить ее. В этом случае будет кнопка **Install**, вы можете видеть ее наряду с другими.

Давайте попробуем. Выйдите из окна настроек и откройте файл README.md. Добавьте следующий код в файл README.md:

```
# The Science Project
Welcome to my science project!
```

Вы должны увидеть свой код Markdown слева и визуализированный предварительный просмотр справа, как показано на рис. 12.8:

¹ Markdown – это язык разметки, который позволяет преобразовать обычный текст в форматированный. – *Прим. ред.*

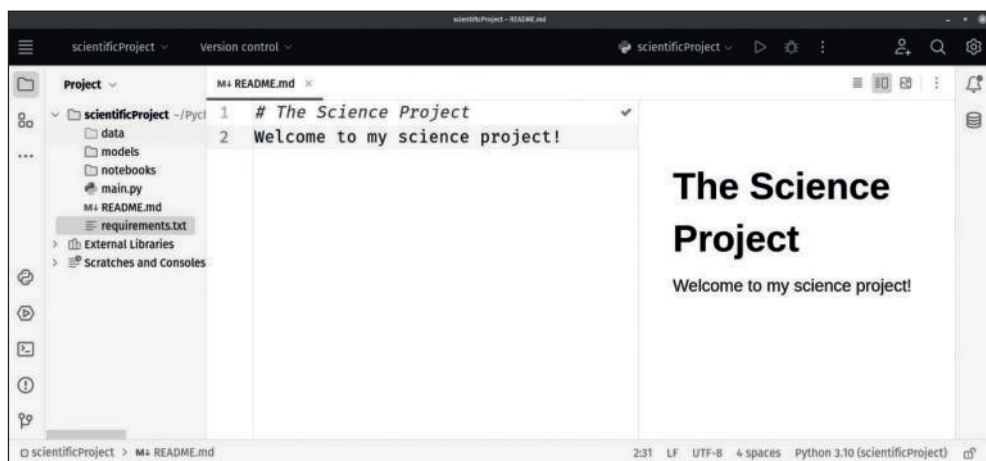


Рис. 12.8. Код Markdown слева, визуализированный результат справа

Вы можете изменить макет предварительного просмотра или вообще скрыть его, используя кнопки в правом верхнем углу редактора Markdown, как показано на рис. 12.9:

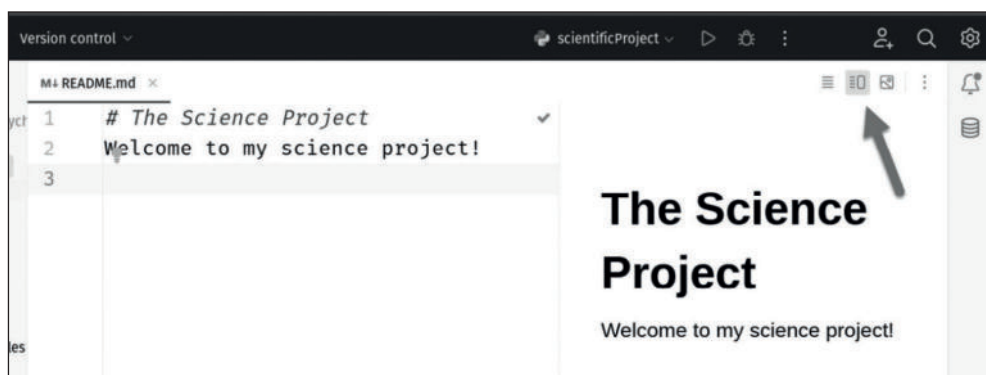


Рис. 12.9. Вы можете настроить расположение окна предварительного просмотра

Нажатие левой кнопки скрывает предварительный просмотр. Нажатие правой кнопки скрывает код. Вы можете один раз нажать кнопку посередине, чтобы переключить предварительный просмотр рядом с кодом. Нажмите на нее еще раз – код и предварительный просмотр будут наложены друг на друга.

Добавление изображений

Необходимая характеристика хорошего редактора Markdown – это возможность легко добавлять изображения. В PyCharm, если вам нужны изображения в Markdown, начните с создания папки в вашем проекте под названием `assets`. Скопируйте файлы изображений в эту папку. Затем просто перетащите изображение и поместите его в редактор. На рис. 12.10 я перетаскиваю изображение из папки ресурсов, а затем помещаю его в свой редактор Markdown:

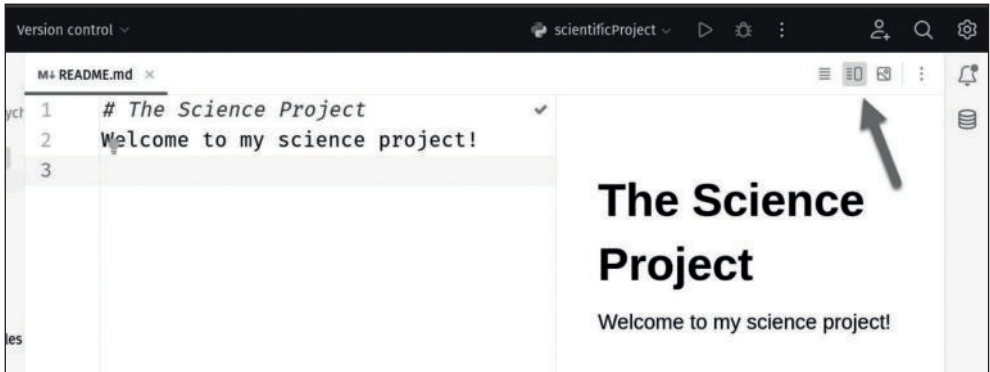


Рис. 12.10. Перетащите изображения из-за пределов PyCharm и поместите их в код, чтобы включить их в свой Markdown

Результат можно увидеть на рис. 12.11:

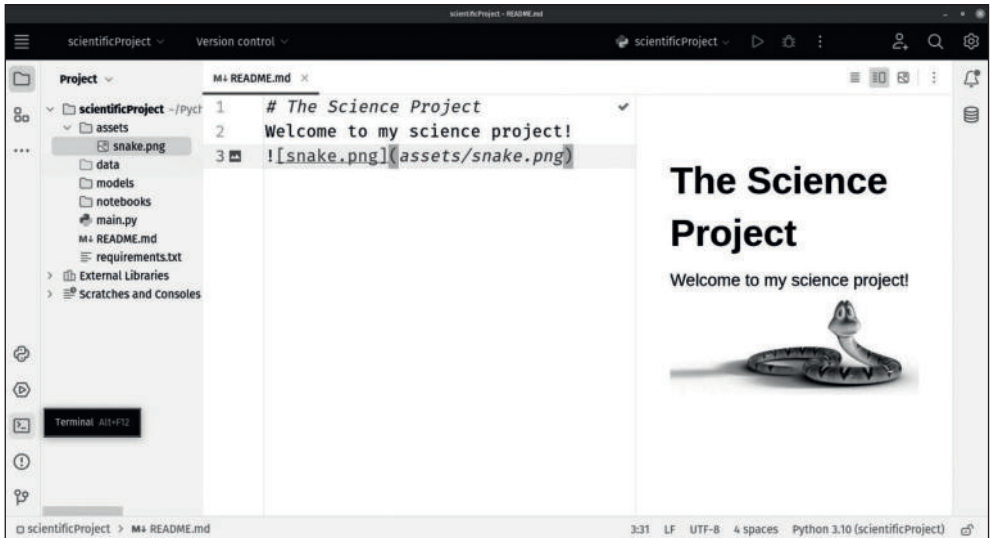


Рис. 12.11. Перетащенное изображение появляется в вашем коде и в окне предварительного просмотра

В целом этот плагин с настройками по умолчанию довольно хорош. Некоторые из других плагинов имеют более продвинутые функции, но на самом деле единственное, что мне нужно, – это конвертировать Markdown в другие форматы. Если вам нужно больше, другие представленные на рынке плагины помогут вам это сделать.

Установка плагина CSV

Распространенным форматом хранения данных является формат **значений, разделенных запятыми (CSV)**. Этот формат поддерживается всеми когда-либо разработанными программами для работы с электронными таблицами.

На торговой площадке есть функция CSV, которая позволяет открывать CSV и аналогичные файлы, такие как **значения, разделенные табуляцией (TSV)**, в интерфейсе, похожем на электронную таблицу. Вы можете увидеть страницу Marketplace для него на рис. 12.12:

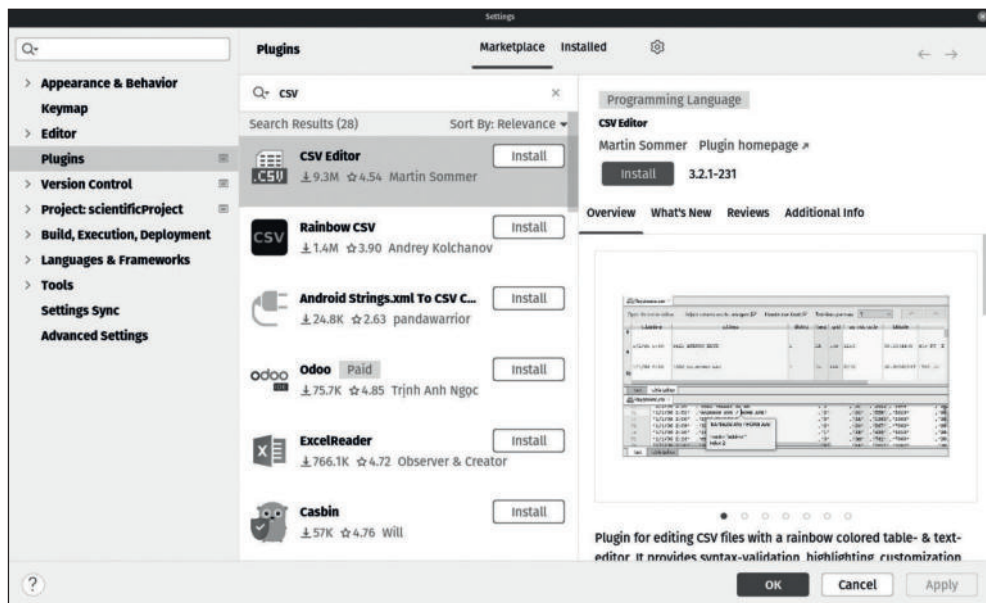


Рис. 12.12. Вероятно, в какой-то момент вы будете использовать файлы CSV. К счастью, для этого есть плагин

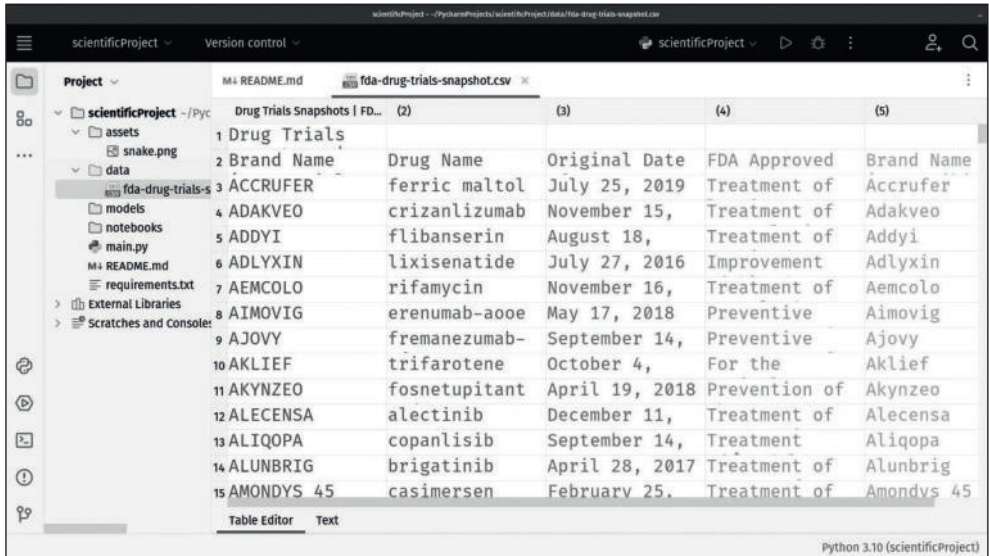
После того как вы нажмете кнопку установки плагина CSV Editor, вам будет предложено перезапустить IDE. Это очень распространенное явление. После перезапуска можете опробовать его с любым из ваших собственных файлов CSV или использовать мой образец данных исследования FDA в папке данных примера кода. Плагин CSV отображает мои данные в полном цвете, как показано на рис. 12.13.

Естественно, вы можете редактировать данные, но это не электронная таблица. Не ждите, что сможете добавлять в ячейки формулы и выражения. Внизу вы увидите, что можете переключаться между представлением **Table Editor** и **Text view**.

Есть еще один, который вам наверняка понадобится, – это плагин режима ячеек.

Установка плагина режима ячеек

Я пока не буду вдаваться в подробности того, что делает плагин режима ячеек (cell mode). Просто поверьте мне, когда я говорю, что это чрезвычайно удобно, когда вы работаете с ячейками кода PyCharm, о которых мы вскоре расскажем. Поскольку мы уже находимся на торговой площадке, давайте возьмем его сейчас. На рис. 12.14 показана страница Marketplace для плагина режима ячеек.



The screenshot shows the PyCharm IDE interface with the 'scientificProject' workspace. The 'fda-drug-trials-snapshot.csv' file is open in the 'Table Editor' view. The table has 5 columns: Brand Name, Drug Name, Original Date, FDA Approved, and Brand Name. The data is displayed with alternating row colors for readability.

Brand Name	Drug Name	Original Date	FDA Approved	Brand Name
ACCURUFER	ferric maltol	July 25, 2019	Treatment of	Accrufer
ADAKVEO	crizanlizumab	November 15,	Treatment of	Adakveo
ADDYI	flibanserin	August 18,	Treatment of	Addyi
ADLYXIN	lixisenatide	July 27, 2016	Improvement	Adlyxin
AEMCOLO	rifamycin	November 16,	Treatment of	Aemcolo
AIMOVIG	erenumab-aooe	May 17, 2018	Preventive	Aimovig
AJOVY	fremanezumab-	September 14,	Preventive	Ajovy
AKLIEF	trifarotene	October 4,	For the	Aklief
AKYNZEO	fosnetupitant	April 19, 2018	Prevention of	Akynzeo
ALECENSA	alectinib	December 11,	Treatment of	Alecensa
ALIQUOPA	copanlisib	September 14,	Treatment	Aliqopa
ALUNBRIG	brigatinib	April 28, 2017	Treatment of	Alunbrig
AMONDYS 45	casimersen	February 25,	Treatment of	Amondys 45

Рис. 12.13. Плагин CSV Editor отображает данные в столбцах разными цветами. Это похоже на красивую электронную таблицу, встроенную в вашу IDE

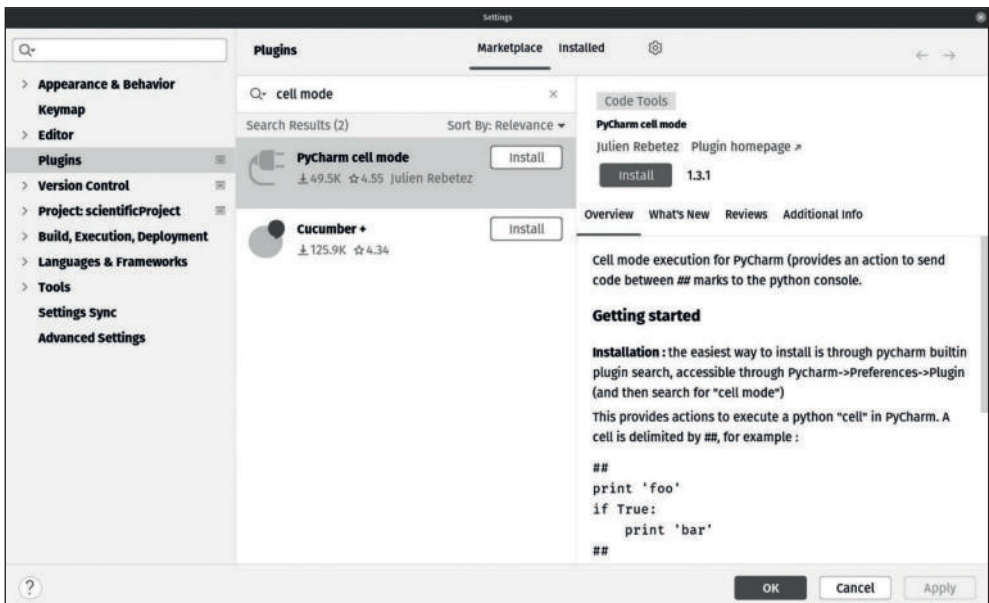


Рис. 12.14. Плагин режима ячеек упростит работу с ячейками кода чуть позже в этой главе

Установите его, нажав кнопку **Install**. Теперь у вас есть PyCharm, достаточно подготовленный для работы с данными.

УСТАНОВКА ПАКЕТОВ

Мы готовимся заняться наукой, но сначала нам нужно установить к нашему проекту некоторые необходимые пакеты. Мы рассмотрели это в предыдущих главах – быстро рассмотрим это и здесь.

Самый новый способ добавить пакеты в любой проект Python – использовать панель **Python Packages**, показанную на рис. 12.15:

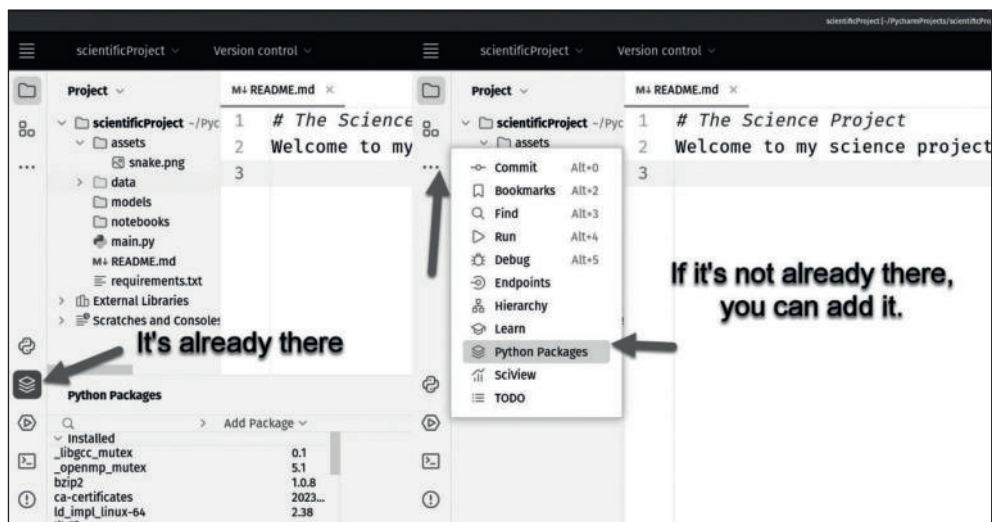


Рис. 12.15. Используйте панель Python Packages для управления пакетами Python. Если ее нет на правой панели инструментов, вы можете ее добавить

Если вы уже использовали эту панель раньше, она будет находиться на правой панели инструментов. Если нет – можете добавить ее, кликнув три точки и выбрав опцию **Python Packages**. Найдя кнопку открытия панели, вы увидите поле **Search**, показанное на рис. 12.16:



Рис. 12.16. Вы можете легко искать пакеты из PyPI на этой панели и добавлять их с помощью conda

Мы собираемся добавить в наш проект пакеты `numpy` и `matplotlib`. Введите названия каждого из них в поле **Search**, указанное на рис. 12.17, затем установите пакет с помощью кнопки **Install with conda**. Если вы решили использовать ванильный Python¹, на кнопке будет написано **Install with pip**.

Заполните файл `requirements.txt`

Теперь, когда у вас установлены два пакета, откройте файл `requirements.txt`. PyCharm сообщит вам, что список внешних зависимостей, установленных в среде `conda`, не соответствуют пустому файлу. На рис. 12.17 показана возможность добавить только что установленные пакеты в файл `requirements.txt` автоматически.

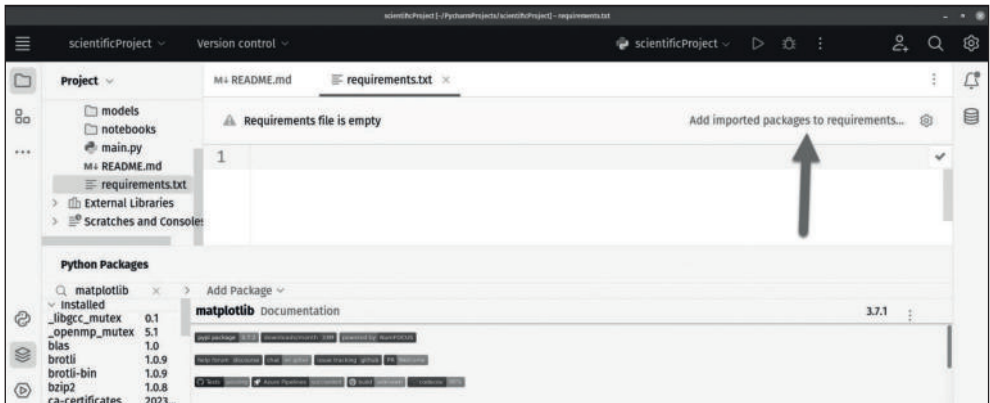


Рис. 12.17. PyCharm помогает поддерживать файл `requirements.txt` в актуальном состоянии

Нажмите кнопку добавления, чтобы обновить файл требований.

ДОБАВЛЯЕМ НАУЧНЫЙ КОД

Мой редактор продолжает говорить, что «научный» – это не то слово. Один мудрый человек однажды сказал: «Наука не о том, почему. Она о том, почему бы и нет?!» Я собираюсь продолжать использовать это слово, и, если вы продолжаете читать, значит, мне это сошло с рук.

Мы настроили IDE и установили необходимые пакеты. Давайте откроем `main.py` и добавим немного кода, чтобы мы могли видеть, как работает PyCharm! В `main.py` добавьте следующий код:

```
import numpy as np
import matplotlib.pyplot as plt
```

¹ Ванильный Python – это основной, стандартный и исходный вариант языка программирования Python без каких-либо дополнительных библиотек или фреймворков. Он предоставляет только базовые возможности и синтаксис Python, позволяя программисту создавать и запускать простые программы без использования сторонних расширений. – *Прим. ред.*

Эти первые два импорта просто добавляют `numpy` и `matplotlib` с псевдонимами. Оказывается, ученые ненавидят печатать больше, чем обычные разработчики:

```
N = 100
x = np.random.normal(0, 1, N)
y = np.random.normal(2, 3, N)

### plot data in histograms
plt.hist(x, alpha=0.5, label='x')
plt.hist(y, alpha=0.5, label='y')
plt.legend
```

Используя NumPy, мы просто создаем два образца набора данных по 100 элементов из нормального распределения. Значения `x` создаются из распределения со средним значением 0 и стандартным отклонением 1, а `y` – из распределения со средним значением 2 и стандартным отклонением 3. Затем мы рисуем соответствующие гистограммы с помощью Matplotlib. Запустите программу как обычно и обратите внимание на результаты, показанные на рис. 12.18:

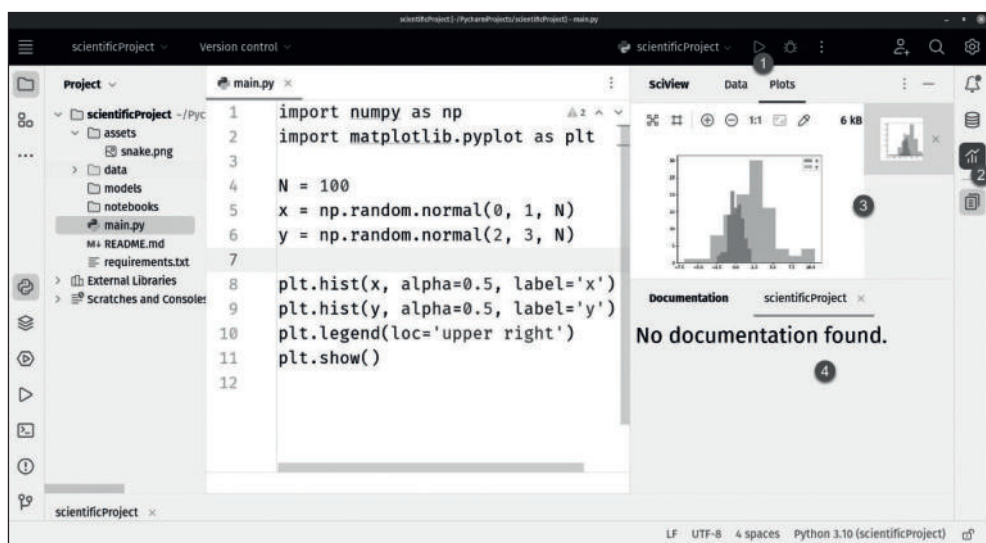


Рис. 12.18. Нажмите кнопку Run (1), и диаграмма matplotlib отобразится в SciView

Наш скрипт Python запустился, но при этом включился SciView. SciView является центральным элементом набора функций анализа данных PyCharm. Он позволяет просматривать графики и диаграммы вместе со сложной документацией по API, общей для таких библиотек, как NumPy, pandas и PyTorch.

Я выделил кнопку запуска (1) на рис. 12.18 на тот случай, если вы присоединились к нам поздно и пропустили предыдущие главы, посвященные

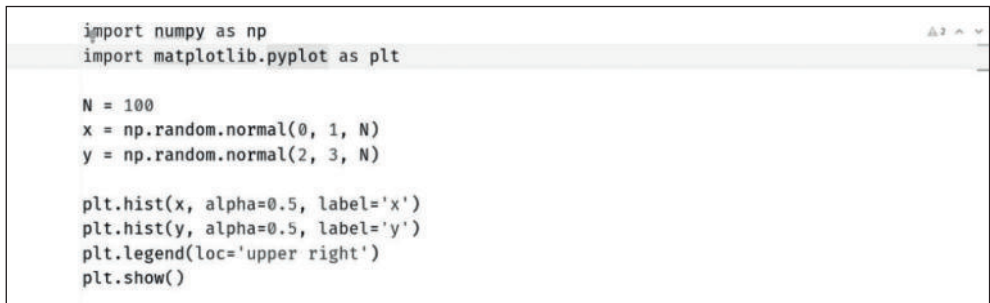
запуску вашего кода в IDE. Значок (2) указывает на то, что SciView включен. Можете включать и выключать его, как и любую другую панель. При включении SciView вы получаете панель визуализаций (3) и панель документации (4). Мы разберемся с этими инструментами подробнее в следующих нескольких главах.

ПЕРЕКЛЮЧЕНИЕ НАУЧНОГО РЕЖИМА

Я уже пару раз упоминал термин «*научный режим*»; теперь мы поймем значение этого режима в проектах PyCharm.

Scientific mode состоит из множества компонентов, которые мы будем изучать в этой и последующих главах, наиболее заметными из которых являются панели **SciView** и **Documentation**. Важно отметить, что этот специальный режим в PyCharm не эквивалентен научному проекту. Более того, это скорее настройка конфигурации, позволяющая упростить доступ и использование различных функций PyCharm, поддерживающих научные вычисления.

Существует множество программ, которые имеют разные конфигурации пользовательского интерфейса в зависимости от того, как вы собираетесь использовать программное обеспечение. Adobe Photoshop имеет разные наборы представлений для фотографов и веб-дизайнеров. Многие браузеры имеют опции или режимы без Chrome, оптимизированные для чтения без отвлечения внимания. Сам PyCharm имеет *режим Zen* («Дзен»), который удаляет весь пользовательский интерфейс, кроме редактора. Вы можете увидеть это на рис. 12.19:



```
import numpy as np
import matplotlib.pyplot as plt

N = 100
x = np.random.normal(0, 1, N)
y = np.random.normal(2, 3, N)

plt.hist(x, alpha=0.5, label='x')
plt.hist(y, alpha=0.5, label='y')
plt.legend(loc='upper right')
plt.show()
```

Рис. 12.19. В объятиях режима «Дзен». Код PyCharm танцует спокойно. Четкость в линиях

Если вас интересует режим Zen, вы найдете его, нажав **View | Appearance | Enter Zen** в главном меню. Это хорошо и все такое, но мы здесь для научного режима, и я хотел сказать, что этот режим – это скорее конфигурация внешнего вида PyCharm, а не специальные панели инструментов, уникальные для научного проекта. Вы можете включить научный режим в любом типе проекта. Чтобы сделать это, нажмите **View | Scientific Mode**, как показано на рис. 12.20:

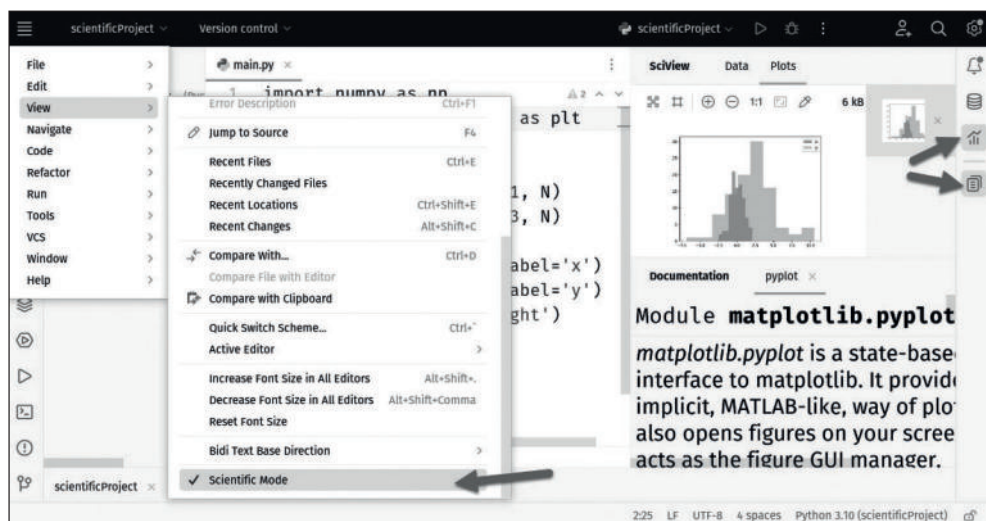


Рис. 12.20. Вы можете включить научный режим в меню просмотра.

Когда научный режим включен, можете переключать две панели с помощью кнопок на панели инструментов у правого края экрана

PyCharm немного любопытен. Если он увидит, что вы используете NumPy, он предложит вам включить научный режим с небольшим тостом в правом нижнем углу IDE (см. рис. 12.21).

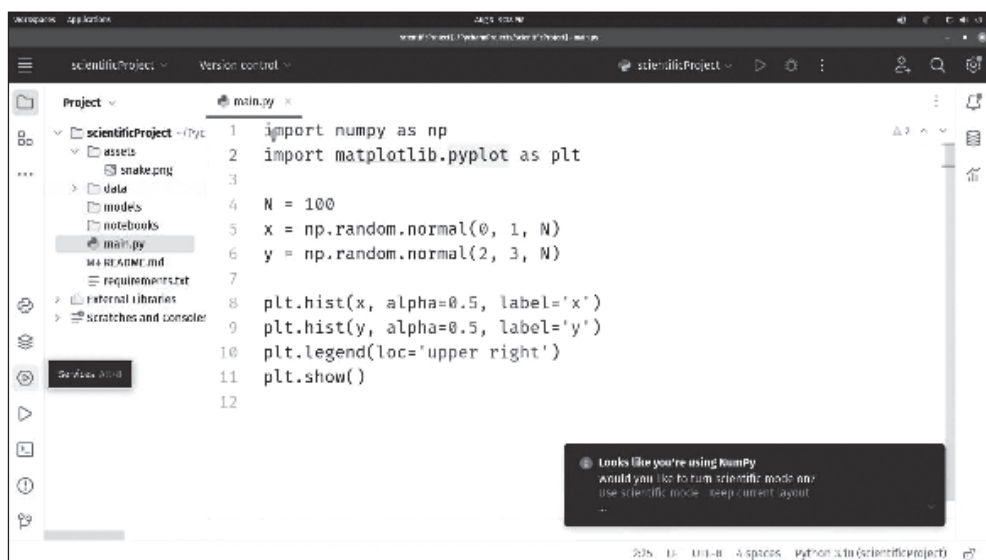


Рис. 12.21. Ваш Большой брат Pythonic наблюдает за вами, но, по крайней мере, пытается быть полезным

В целом научный режим предлагает интуитивно понятный интерфейс, который может повысить производительность вашего труда в проектах научных

вычислений. В следующем разделе мы более подробно рассмотрим другие расширенные функции научного проекта, а именно панель **Documentation** и ячейки кода PyCharm.

ПОНИМАНИЕ РАСШИРЕННЫХ ВОЗМОЖНОСТЕЙ НАУЧНЫХ ПРОЕКТОВ PYCHARM

Благодаря функциям, которые мы обсуждали в предыдущем разделе, вы можете эффективно и продуктивно перемещаться по научным проектам PyCharm и работать с ними. Однако есть и другие тонкие функции PyCharm, которые могут оказаться полезными в этом контексте. Сначала мы рассмотрим панель **Documentation** и ее использование.

Окно просмотра документации

Документация является важной частью программирования и разработки программного обеспечения, и PyCharm предлагает самые мощные и простые функции для поддержки задач работы с документацией на Python. Впервые мы увидели это в действии в главе 4.

В научном проекте панель **Documentation**, как мы видели, закреплена как одна из основных панелей окна проекта. Эта программа просмотра документации отображает данные документации в реальном времени в динамическом режиме. В частности, когда вы перемещаете курсор на определенный метод или вызов функции в редакторе, на панели **Documentation** отображается официальная документация, соответствующая этому методу или функции.

Вернитесь к коду, который мы написали ранее, и поместите курсор в строку 10. Поместите курсор на слово `legend`¹, и в окне **Documentation** мгновенно отобразится документация для этого метода, который является частью библиотеки Matplotlib. Вы можете увидеть это на рис. 12.22.

Особенность этих научных библиотек, популярных в науке о данных, заключается в том, что они могут иметь сложные структуры данных, используемые в качестве аргументов и возвращаемых типов. Наличие справочных материалов по образцам чрезвычайно полезно. Я бы порекомендовал экран большего размера, чтобы по-настоящему воспользоваться его преимуществами. Мой снимок экрана на рис. 12.22 немного перегружен только потому, что он показан на меньшем экране, шрифты которого подобраны таким образом, чтобы они были понятны для печатной книги.

¹ Легенда – это область диаграммы, описывающая каждую часть графика. График может быть максимально простым. Но добавление заголовка, меток X, Y и легенды будет более понятным. По названиям мы можем легко догадаться, что представляет собой график и какой тип данных он представляет. – *Прим. ред.*

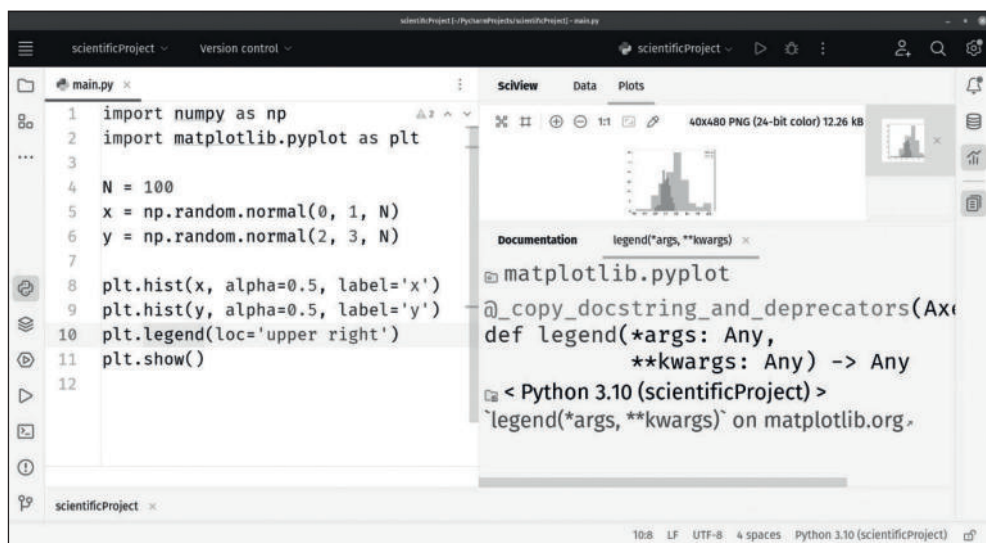


Рис. 12.22. Нажмите практически на что угодно, и на панели Documentation отобразится документация по любому свойству или методу

Documentation не только отображает документацию по тому месту, куда вы нажимаете, но и обновляется по мере ввода кода. Вы не только получаете обычное автозаполнение, но и одновременно видите полную документацию. Панель **Documentation** имеет несколько параметров конфигурации, доступных путем щелчка по трем вертикальным точкам в верхней части панели, как показано на рис. 12.23:

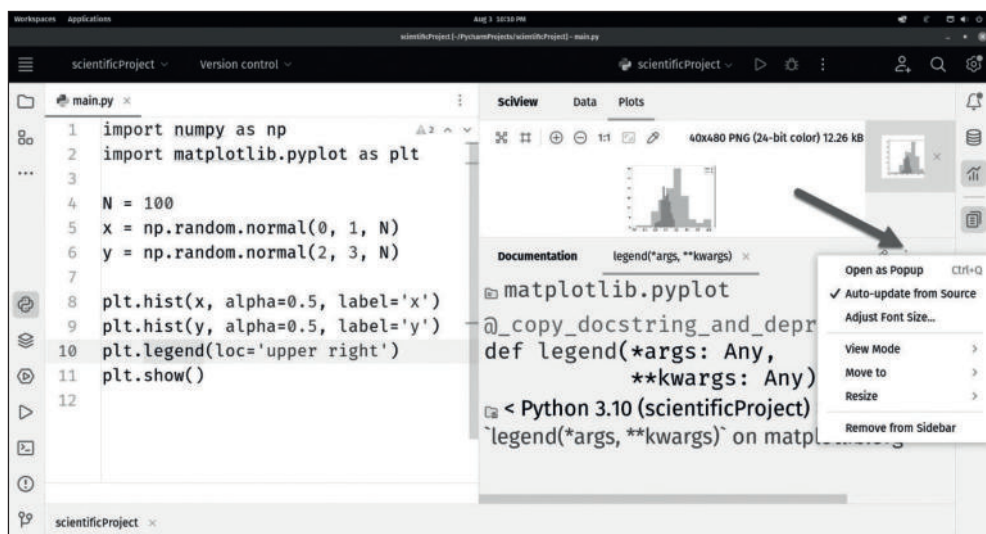


Рис. 12.23. Существует несколько параметров конфигурации панели Documentation, расположенных под меню в верхней части панели

Далее мы рассмотрим уникальную функцию PyCharm, когда дело доходит до выполнения кода Python, т. е. реализации ячеек кода.

Использование ячеек кода в PyCharm

Ячейки кода PyCharm – это способ разделения и последовательного выполнения различных частей большой программы Python. Вы должны узнать эту функцию, если когда-либо использовали **Jupyter Notebook**. Ячейки кода в PyCharm – это, по сути, урезанная версия этого инструмента. PyCharm также имеет прямую поддержку блокнотов Jupyter, но я оставляю это для следующей главы.

Потребности эксперта по аналитическим данным (Data Scientist) отличаются от потребностей обычного разработчика программного обеспечения. Специалист по данным не беспокоится о написании готового к отправке кода с теми же архитектурными и масштабируемыми ограничениями, что и разработчик программного обеспечения. Они также не имеют дело с типичными переменными, которые в основном являются примитивами или небольшими объектами с несколькими свойствами и методами. Они имеют дело с объемами, а программы, которые они пишут, алгоритмичны. Не приходите в ужас от слова **алгоритм**. Алгоритм – это просто набор шагов, составляющих процесс. Конечно, алгоритм PageRank, лежащий в основе поискового бизнеса Google, является удивительным достижением запатентованной разработки. Это алгоритм, как и рецепт старого доброго сэндвича с арахисовым маслом и джемом. Сложность не является обязательным требованием, на самом деле лучшие алгоритмы обычно просты.

Дата-сайентисты изучают проблемы, собирая, фильтруя и обрабатывая большие объемы данных, используя ряд совокупных шагов. Каждый шаг дискретен, имеет вход и выход. Ячейки кода упрощают создание и работу с этими отдельными шагами вместо необходимости иметь дело со сложными структурами кода, как это сделал бы обычный разработчик программного обеспечения.

Использование ячеек кода PyCharm

Ячейки кода в PyCharm определяются строками кода, которые начинаются со следующих символов: `#%%`. Эти строки рассматриваются как стандартные комментарии при низкоуровневом выполнении Python, но PyCharm распознает их как разделители ячеек кода в своем редакторе. Давайте посмотрим эту функцию в действии.

Вернемся к нашему демонстрационному коду, представленному ранее, и внесем некоторые изменения:

```
import numpy as np
import matplotlib.pyplot as plt

#%% generate random data
N = 100
x = np.random.normal(0, 1, N)
y = np.random.normal(2, 3, N)
```

```

### plot data in histograms
plt.hist(x, alpha=0.5, label='x')
plt.hist(y, alpha=0.5, label='y')
plt.legend(loc='upper right')
plt.show()

```

Обратите внимание на добавление двух строк, начинающихся с `###`. Эти строки определяют ячейки нашего кода. Более того, они также служат обычными комментариями, объясняющими, что делает блок кода. Теперь посмотрите, как изменился пользовательский интерфейс PyCharm. Теперь вы должны увидеть несколько кнопок запуска, как показано на рис. 12.24:

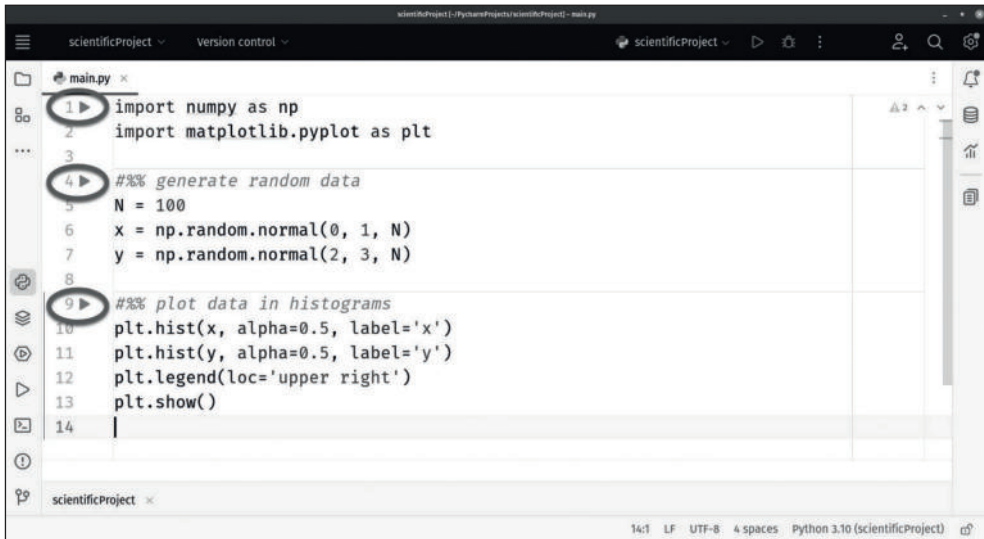


Рис. 12.24. Кнопки Run появляются в поле в ответ на специальный формат комментариев

Теперь мы можем запустить каждую ячейку и просмотреть результат, прежде чем запускать следующую. Если нам нужно внести корректировку в среднюю ячейку, можно повторно запустить эту ячейку и последующие. Когда вы нажимаете кнопки запуска, код выполняется в консоли PyCharm. Если ваша консоль не видна, можете перейти к ней, нажав **View | Tool Windows | Python Console**. Открыв панель **Python Console**, вы можете проверить код во время его выполнения, как показано на рис. 12.25.

По мере выполнения каждого шага вы можете видеть выполнение кода в окне консоли. Помните, что консоль обращается к Python REPL, а не к окну терминала. Мы обсуждали это в главе 3. В окне **Python Console** имеется панель проверки, позволяющая проверять состояние так же, как при отладке.

Поскольку ваш код выполняет консоль PyCharm, если вам нужно начать заново, можете просто закрыть консоль и снова выполнить действия. Это создаст новую консоль.



Рис. 12.25. Вы можете видеть, что код запускается в консоли. Существует проверочная панель, позволяющая увидеть, что произошло после выполнения ячейки

Плагин режима ячеек

Ранее, когда мы обсуждали несколько интересных плагинов, которые можно использовать для работы с данными, мы установили плагин под названием Cell Mode. Плагин режима ячеек расширяет возможности ячеек кода.

Для начала с плагином режима ячеек вам нужно использовать двойные знаки решеточки/хештега (##) вместо обычного %%. Когда вы это сделаете, плагин возьмет на себя управление и предоставит вам дополнительные возможности запуска. Взгляните на рис. 12.26:

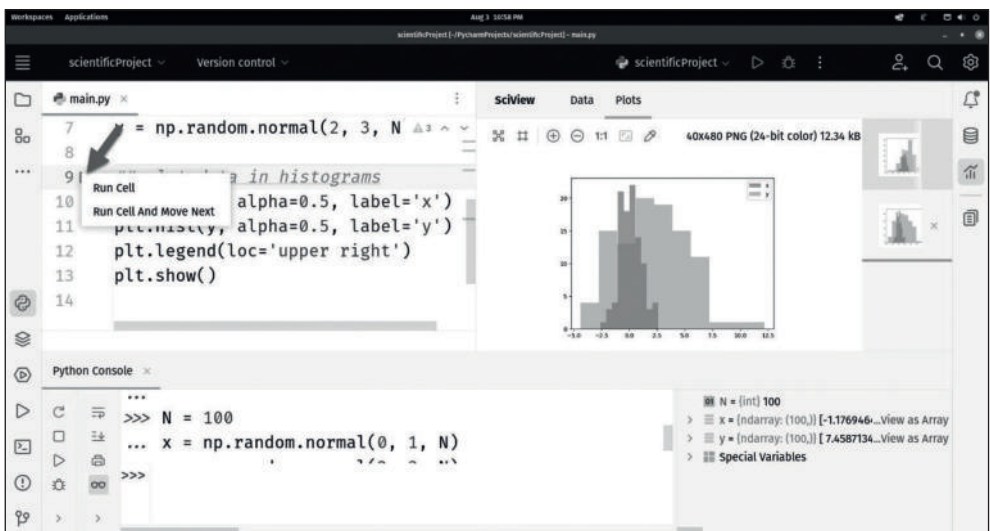


Рис. 12.26. На этот раз при нажатии вы получаете несколько дополнительных опций

Кнопки запуска теперь позволяют вам запустить ячейку и немедленно перейти к следующей, а не просто останавливаться в нижней части ячейки, которую вы только что выполнили. На рис. 12.27 показано, что у вас есть дополнительные элементы управления на уровне меню:

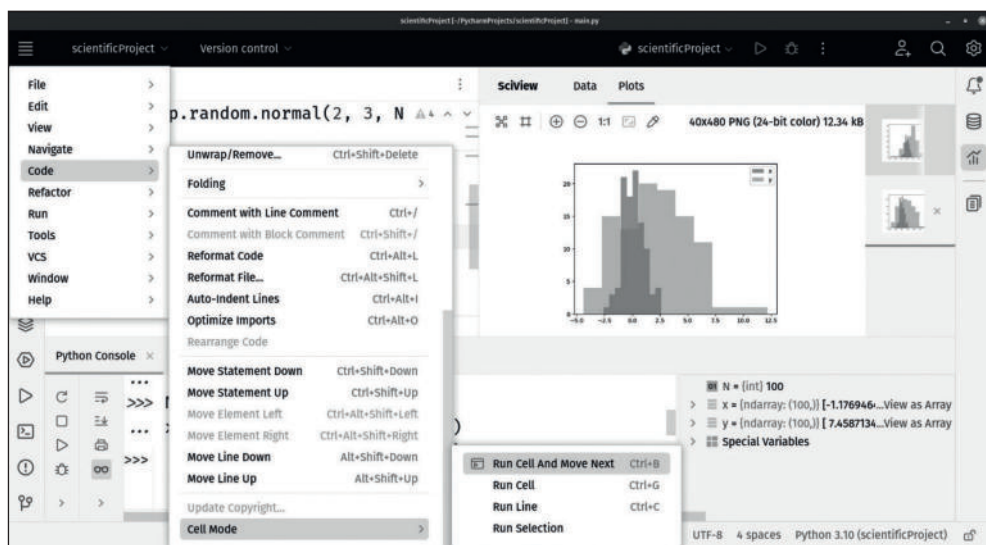


Рис. 12.27. Плагин режима ячеек добавляет дополнительные пункты в меню Code

Когда вы нажимаете **Code | Cell Mode**, то получаете еще более детальные параметры запуска ячеек кода. В целом мы видим, что этот плагин предлагает все преимущества использования блокнотов Jupyter без необходимости фактического переключения на приложения Jupyter.

КРАТКОЕ СОДЕРЖАНИЕ

Научный проект в PyCharm создается с общей структурой, которая является общей для проектов в реальной жизни, включая передовые методы, такие как папка данных, исключенная из контроля версий, файл README.md и файл `requirements.txt`. Как вы можете себе представить, необходимость вручную создавать эту настройку для каждого проекта может оказаться сложной и трудоемкой задачей. Эта функция помогает пользователям PyCharm сразу приступить к процессу разработки после создания проекта, чтобы им не приходилось беспокоиться о различных деталях. Это позволит нам работать быстрее и продуктивнее в процессе разработки.

Кроме того, научный режим PyCharm включает в себя различные функции, которые поддерживают процесс разработки научных вычислений или проектов по обработке данных, а именно панели Documentation и SciView. В сочетании с этим режимом вы также можете воспользоваться другими мощными функциями, такими как ячейки кода и плагин CSV, чтобы упростить выполнение различных задач и эффективно повысить производительность в проектах по науке о данных.

Однако эти функции знаменуют собой лишь начало того, что PyCharm может предложить нам при оказании помощи в проектах, связанных с данными. Опираясь на эти темы, в следующей главе мы рассмотрим использование панели SciView и блокнотов Jupyter, которые являются важной частью экосистемы обработки данных Python в PyCharm.

Вопросы

1. Что такое язык Markdown? Какой цели служит файл README.md в репозитории GitHub?
2. Почему папка данных в научном проекте в PyCharm исключена из контроля версий?
3. Как включить и выключить научный режим в PyCharm? Какой эффект это окажет на данное окно проекта?
4. Какие функции доступны на панели **Documentation** в PyCharm?
5. Что такое ячейки кода в PyCharm и как их реализовать?
6. Какие функции доступны в плагине CSV в PyCharm?

Динамический просмотр данных с помощью SciView и Jupyter

В этой главе мы продолжим наше научное путешествие, исследуя две жизненно важные функции PyCharm: **SciView** и интеграцию с **Jupyter notebooks**. Обе функции предоставляют нам интегрированные и удобные интерфейсы, позволяющие просматривать данные и переменные в наших научных проектах и работать с ними.

Мы начнем с обсуждения панели SciView, которая была неявно представлена в предыдущей главе. Здесь мы углубимся и сделаем это более непосредственно, работая с массивами NumPy и DataFrames библиотеки pandas.

После этого мы еще больше усовершенствуем наш рабочий процесс, включив в него работу с интерактивными вычислительными инструментами Python, такими как блокноты Jupyter, в контексте наших научных проектов в PyCharm.

К концу этой главы вы должны получить знания в следующих областях, таких как:

- просмотр данных и взаимодействие с ними на панели SciView в PyCharm,
- интеграция **Interactive Python (IPython)** в PyCharm,
- использование поддержки блокнотов Jupyter для интерактивного программирования в рамках научного проекта PyCharm.

Перевернув страницу на следующую главу, вы будете вооружены знаниями, необходимыми для использования двух самых важных видов оружия в войне за уменьшение объема ваших данных.

ТЕХНИЧЕСКИЕ ТРЕБОВАНИЯ

Поскольку мы перешли к теме науки о данных, я переключил свой любимый дистрибутив Python на Anaconda, который представляет собой дистрибутив

Python, адаптированный для рабочих нагрузок науки о данных. Вы можете найти его вместе с инструкциями по установке для вашей операционной системы на <https://anaconda.com>.

Аналогично вместо обычного `pip` я буду использовать `conda` – менеджер пакетов Anaconda. Он установлен рядом с Anaconda.

Вам понадобится установленная и рабочая копия PyCharm. Установка была описана в главе 2.

Вам также понадобится пример исходного кода этой книги с GitHub. Мы рассмотрели клонирование кода в главе 2. Код этой главы вы найдете на <https://github.com/PacktPublishing/Hands-On-Application-Development-with-PyCharm---Second-Edition/tree/main/chapter-13>.

ПРОСМОТР ДАННЫХ С ПОМОЩЬЮ ПАНЕЛИ SciVIEW PYCHARM – ЛЕГКО И ПРОСТО

В главе 12 мы рассмотрели панель SciView с высоты 10 000 футов, поэтому примерно представляем, с чем сейчас столкнемся. Мы знаем, что SciView – это панель, позволяющая нам графически визуализировать данные, наряду со второй панелью, которая обеспечивает нам простой и интегрированный доступ к документации для более сложных библиотек обработки данных, используемых в типичном научном проекте.

Чтобы увидеть это волшебство в действии, вернемся к коду, который мы написали еще в главе 12, и сделаем небольшие улучшения. Вместо того чтобы заставлять вас возвращаться к коду главы 12, я скопировал проект в папку главы 13 в репозитории, чтобы все было вместе. Вы можете найти его в проекте Chapter-13/sci_view_panel в файле `main.py`. Не забывайте, что вам нужно будет установить требования в файле `requirements.txt` в виртуальной среде, чтобы использовать пример проекта. Если вам нужно освежить знания о том, как это сделать, вернитесь к главе 3:

```
import numpy as np
import matplotlib.pyplot as plt

N = 100
for _ in range(5):
    x = np.random.normal(0, 1, N)
    y = np.random.normal(2, 3, N)

    plt.hist(x, alpha=0.5, label='x')
    plt.hist(y, alpha=0.5, label='y')
    plt.legend(loc='upper right')
plt.show()
```

Вы узнаете, что делает этот код, из последней главы. Код генерирует два выборочных набора данных, а затем отображает результаты в виде гистограммы. Разница в том, что мы добавили `for _ in range(5)`. Это тот же код, мы просто запускаем его пять раз подряд, что генерирует пять разных гистограмм.

ПРОСМОТР ДИАГРАММ И РАБОТА С НИМИ

В этом примере вам следует воспользоваться возможностью запуска кода в консоли. Вероятно, этот параметр уже отмечен галочкой, но вы можете еще раз проверить конфигурацию запуска по умолчанию, как показано на рис. 13.1.

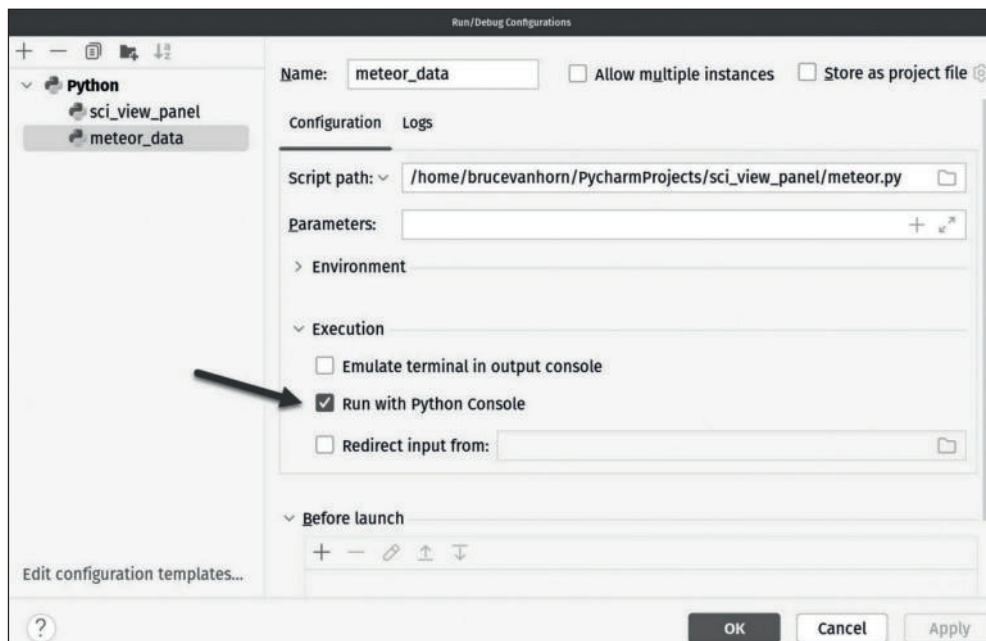


Рис. 13.1. Указанный параметр указывает, что этот код будет запускаться в окне консоли, которое часто используется в научных проектах

Вы видели эту диаграмму в предыдущей главе, но взгляните на результаты этого прогона на рис. 13.2.

Уже можно увидеть разницу: на этот раз диаграмм – пять. Нажатие на каждую из них изменит гистограмму, которая изменится в просмотрном окне. Это улучшение по сравнению с обычными запусками Python, поскольку показ диаграммы обычно блокирует выполнение. Обычно можно просматривать только одну диаграмму за раз, а следующая генерируется только тогда, когда вы нажимаете клавишу **Q**, чтобы выйти из текущей.

Вместо функций по умолчанию PyCharm захватывает все ваши графики и диаграммы, чтобы избавить вас от необходимости экспортировать каждое изображение вручную. Это огромная экономия времени! У вас также есть возможность увеличивать и уменьшать масштаб с помощью панели инструментов в верхней части панели, удалять диаграмму из поля зрения с помощью кнопки **X** справа от ее значка, сохранять диаграмму в файл изображения или удалять все графики и диаграммы из окна просмотра, кликнув правой кнопкой мыши по соответствующему значку, как показано на рис. 13.3.

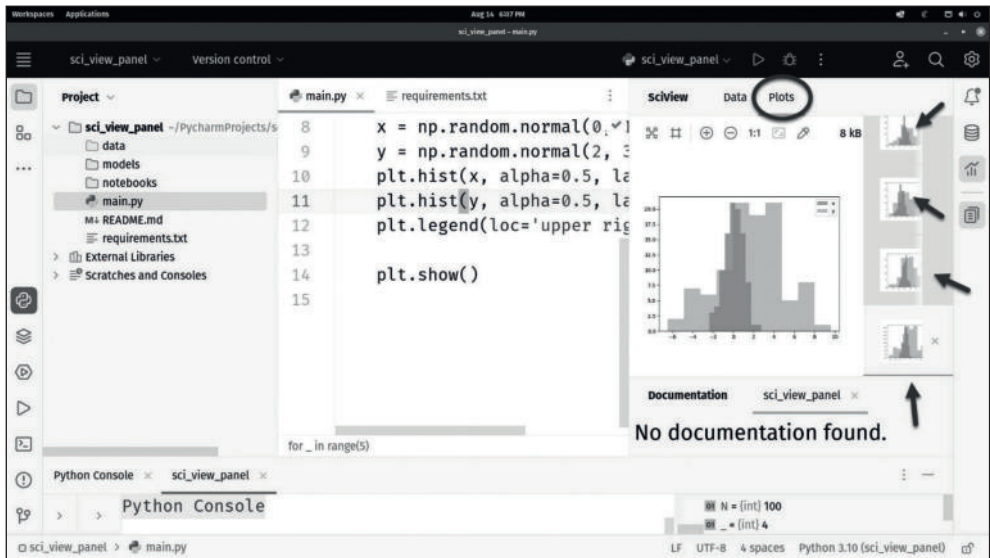


Рис. 13.2. Окно SciView показывает нам последнюю диаграмму, но также позволяет нам выбирать их из других запусков

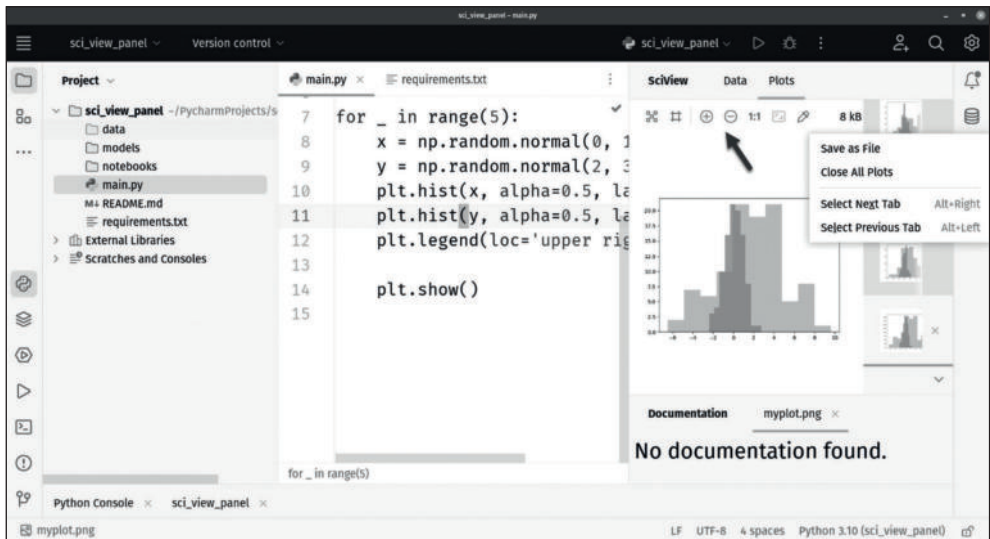


Рис. 13.3. PyCharm предоставляет множество возможностей для работы с графиками и диаграммами

Помимо возможности сохранять интересные вас диаграммы, ваша программа просмотра дает возможность настраивать некоторые параметры, например увеличение масштаба, что указано стрелкой на предыдущем рисунке.

Тепловые карты и корреляционные данные

PyCharm может раскрашивать тепловые карты¹, чтобы помочь вам легко обнаружить корреляции в корреляционной матрице. Чтобы увидеть это, давайте сгенерируем некоторые данные, в которых некоторые точки данных будут коррелированы, а некоторые нет. В исходном коде главы вы найдете файл `correlation_heatmap.py`. Код таков:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

### generate sample data
# x and z are randomly generated
# y is loosely two times of x
x = np.random.rand(50,)
y = x * 2 + np.random.normal(0, 0.3, 50)
z = np.random.rand(50,)

df = pd.DataFrame({
    'x': x,
    'y': y,
    'z': z
})

### compute the correlation matrix
corr_mat = df.corr()

### plot the heatmap
plt.matshow(corr_mat)
plt.show()
```

Обратите внимание, что код содержит разделители ячеек, разделенные символами `###`. Мы рассказали об особом значении этих символов для PyCharm в главе 12.

В этой программе мы создаем **pandas DataFrame** с тремя разными атрибутами (x, y и z), которые генерируются случайным образом. Мы корректируем эти случайные числа между x, y и z так, чтобы между ними был некоторый уровень корреляции. Затем вычисляем корреляционную матрицу этого набора данных с помощью метода `corr()`. Наконец, мы отображаем эту корреляционную матрицу в виде тепловой карты, используя метод `matshow()` из Matplotlib.

Теоретически матрица корреляции показывает, насколько атрибут в каждом наборе данных коррелирует с другим. Более высокое значение означает более высокую корреляцию между парой атрибутов. Как правило, знание того, какие атрибуты тесно связаны друг с другом, даст ценную информацию о наборе данных проекта.

Чтобы продемонстрировать это, мы генерируем атрибут y примерно в два раза больше атрибута x, создавая корреляцию между этими двумя атрибутами. Атрибут z, с другой стороны, генерируется случайным образом и независимо

¹ Тепловая карта – это тип диаграммы, в которой для представления значений данных применяются разные оттенки цвета. – *Прим. ред.*

от x и y , поэтому не должно быть высокой корреляции между z и любым из других атрибутов.

Проект должен иметь конфигурацию запуска, называемую *correlation*. Настройки конфигурации запуска показаны на рис. 13.4.

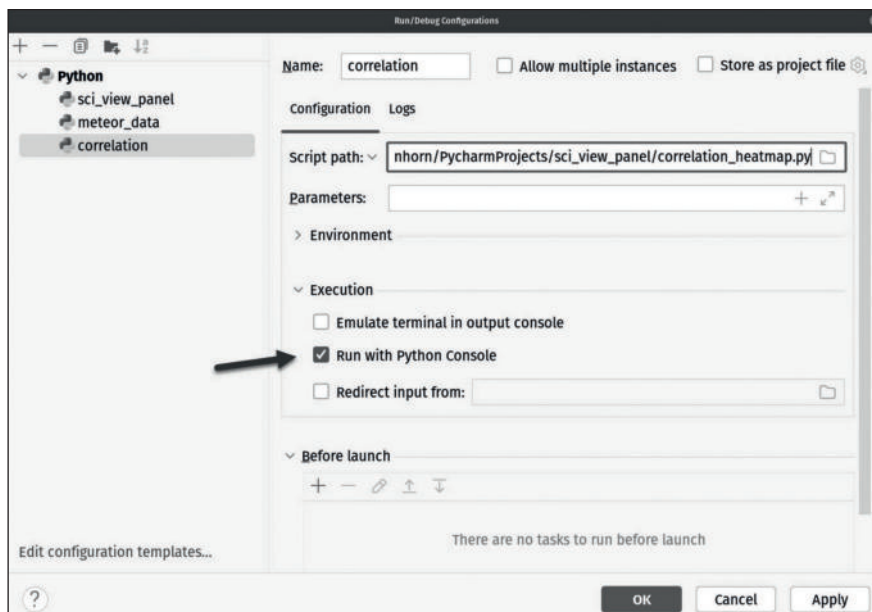


Рис. 13.4. Это конфигурация запуска файла `correlation_heatmap.py`

Помните, что важно установить флажок **Run with Python Console**. Запустите файл и просмотрите результат. Мой показан на рис. 13.5. Поскольку данные случайны, ваши могут выглядеть иначе, чем мои.

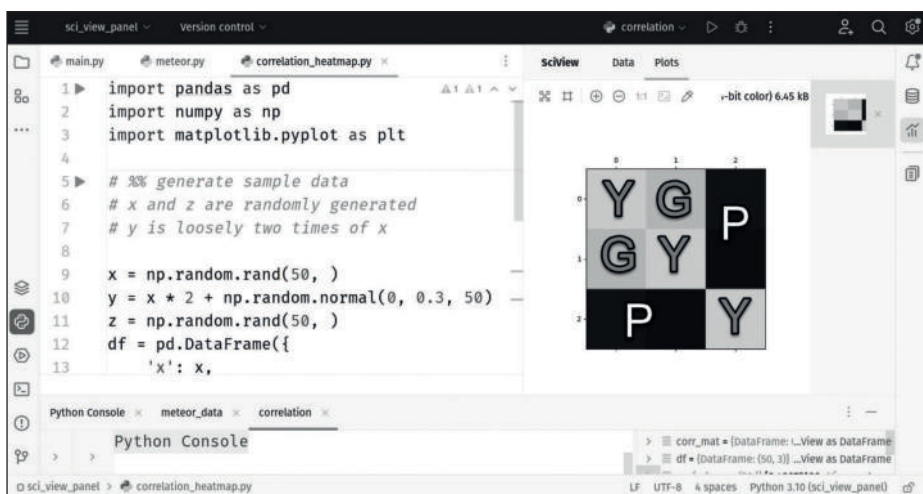


Рис. 13.5. Корреляционные тепловые карты отображаются с использованием цветов, обозначающих уровень корреляции в корреляционной матрице

Поскольку эта книга напечатана в черно-белом режиме, я постарался сделать ее более убедительной для вас, добавив буквы для обозначения цветов. Это не особенность диаграммы. Клетки, помеченные буквой **Y**, имеют желтый цвет, **G** – зеленый, **P** – фиолетовый. Первый и второй атрибуты сильно коррелируют, поэтому цвет в соответствующих ячейках корреляционной матрицы (строка 1, столбец 2 и строка 2, столбец 1) яркий (желтый или зеленый, каждый из которых имеет одинаковую яркость). Корреляция между третьим атрибутом и двумя другими низкая, на что указывает темно-фиолетовый цвет. Естественно, каждый признак прекрасно коррелирует сам с собой, отсюда и ярко-желтый или зеленый цвет в диагональных ячейках.

Визуальные разветвления здесь очень полезны. Работая с корреляционными матрицами, мы можем мгновенно увидеть взаимосвязи между нашими переменными.

Вкладка **Plots** в SciView предоставляет нам множество отличных инструментов, которые мы можем использовать для просмотра диаграмм и управления ими в наших проектах науки о данных. Также обратите внимание, что в верхней части панели SciView есть две вкладки, показанные на рис. 13.2 (обведены кружком). Далее мы поговорим о вкладке **Data**.

ПРОСМОТР ДАННЫХ И РАБОТА С НИМИ

Нажав на вкладку **Data**, вы можете оказаться немного разочарованы. Как видно на рис. 13.6, реальные действия происходят ниже в окне отладчика.

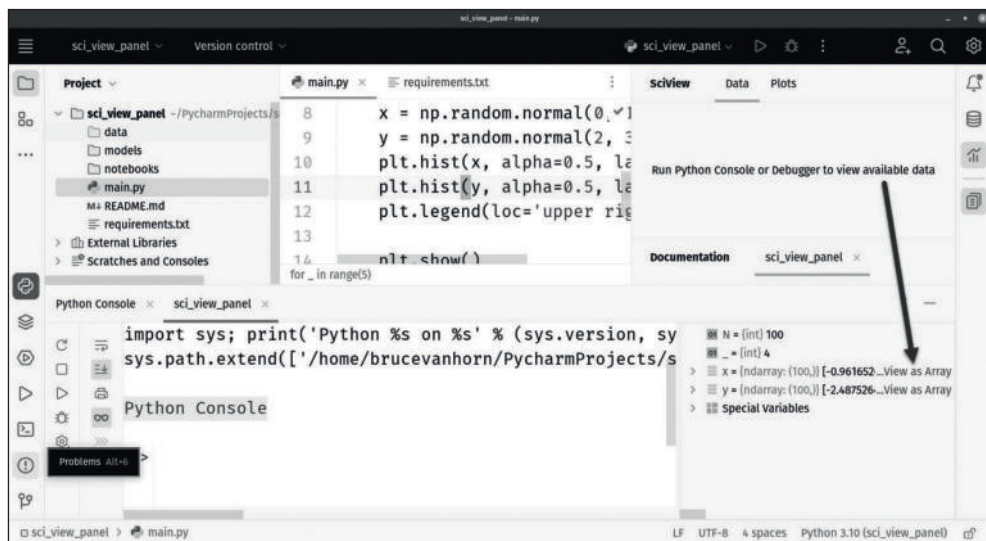


Рис. 13.6. На вкладке **Data** ничего не отображается, пока вы не выберете один из наборов образцов для просмотра в виде массива

Переключите свое внимание на панель **Python Console**, которая появилась в нижней части окна проекта, когда мы запускали нашу программу в консоли. В правой части панели, обозначенной стрелкой, мы видим раздел, в котором

перечислены все переменные в нашей программе и их соответствующие значения. Для одиноких переменных, таких как `N`, этой панели достаточно. Однако если вы, как и мы, используете массивы NumPy, просмотр массива в контексте обычного окна отладчика, подобного этому, проблематичен.

Если вы кликнете ссылку **View as Array**, которую также можно активировать, кликнув правой кнопкой мыши переменную, то увидите таблицу в виде электронной таблицы на панели **Data**. Посмотрите на рис. 13.7.

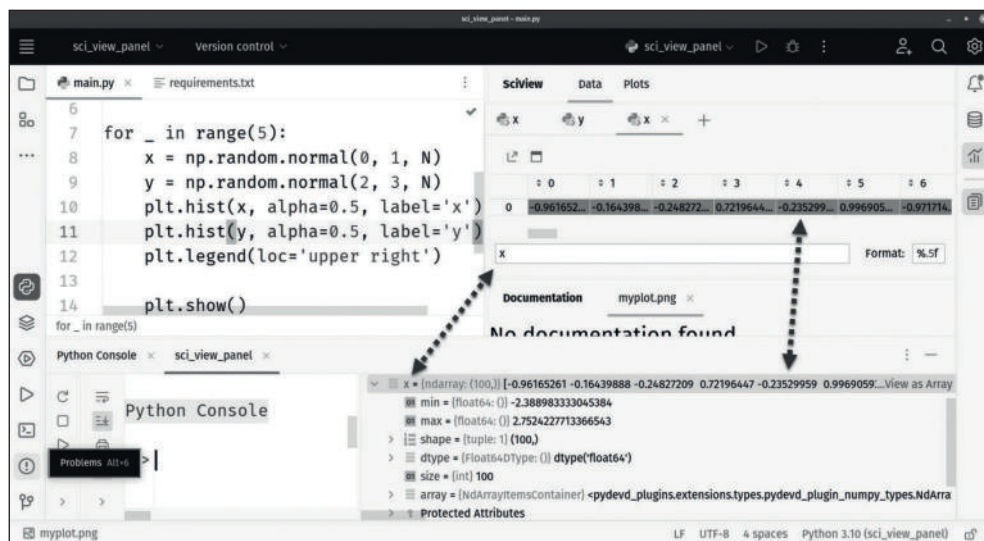


Рис. 13.7. После выбора переменной `X` на нижней панели на вкладке **Data** теперь отображаются элементы массива NumPy внутри `X`

Как видите, панель **Data** заполнена подробностями того, что вы выбрали в нижней консоли. Я нажал **View as Array** для `x` и увидел, что `x` отображается в виде длинной строки цветных данных. Я могу прокрутить вправо и проверить все 100 элементов. Ячейки с высокими значениями заполняются более теплыми цветами, а ячейки с низкими значениями – более холодными. Это обеспечивает контраст, поэтому вы можете визуальнo обнаруживать различия между значениями, не щурясь на десятичный знак.

Крайняя левая стрелка на рис. 13.7 показывает, что `x` в текстовом поле выше соответствует `x` в окне ниже. Вы можете изменить значение в текстовом поле на `y` или любое другое допустимое имя переменной, присутствующее в нижней консоли, и просмотреть данные, представленные более графически убедительнее выше. Ключевым моментом здесь является слово *valid*. На вкладке **Data** отображаются только массивы NumPy и pandas DataFrames. Нажатие на переменную `N` отобразит ошибку на вкладке **Data**.

Справа от текстового поля находится формater. Это позволяет вам указать, как будут выглядеть числа на вкладке **Data**. Строка форматирования использует тот же формат, что и обычное форматирование строк Python.

Когда вы выбираете несколько переменных в представлении консоли, каждая переменная будет отображаться на отдельной вкладке, как показано на рис. 13.8.

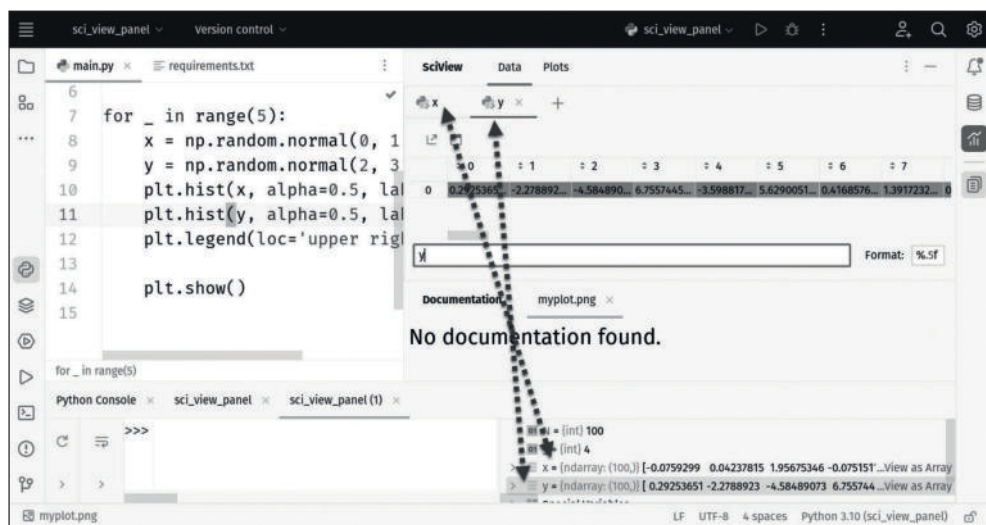


Рис. 13.8. При выборе нескольких переменных в консоли откроется вкладка для каждой переменной на вкладке Data

Просто помните, что это работает только для массивов NumPy и pandas DataFrames. Эти данные немного скучны, поскольку здесь всего одна строка. Давайте попробуем что-нибудь более острое, чтобы увидеть больше функций на вкладке **Data**.

В коде главы вы найдете файл с именем `meteor.py`, в котором всего две строки:

```
import pandas as pd
meteor_data = pd.read_csv("../data/Meteorite_Landings.csv")
```

Такие данные предоставило **Национальное управление по аэронавтике и исследованию космического пространства (NASA)** США. Вы найдете его вместе с тысячами других свободно доступных наборов данных по адресу <https://data.gov>, все любезно предоставлено лично мной, так как я плачу много налогов. Конечно, другие помогали, но хотя бы раз в год кажется, что все зависит от меня.

Файл данных содержит данные НАСА о приземлениях метеоритов. Подробности можно найти по адресу <https://catalog.data.gov/dataset/meteorite-landings>.

В проекте из репозитория должна быть конфигурация запуска `meteor_data.py`, но на всякий случай мои настройки есть на рис. 13.9. Если вам нужно освежить знания о конфигурациях запуска, мы рассмотрели эту тему в главе 3.

Как и раньше, главное – убедиться, что файл запускается в **Python Console**, чтобы вы получили представление панели, необходимое для SciView. Идите дальше и запустите файл. Результат моего запуска вы можете увидеть на рис. 13.10.

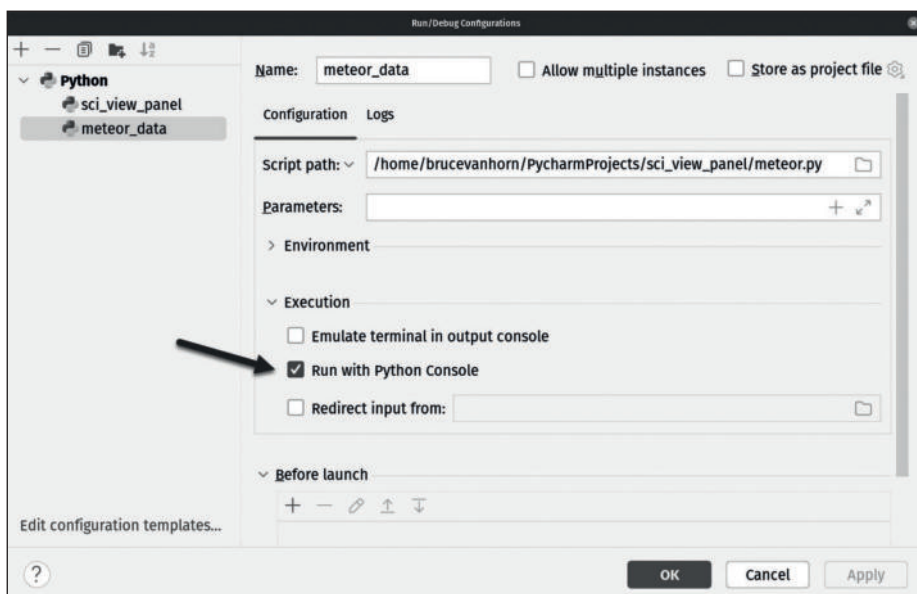


Рис. 13.9. Это диалоговое окно настроек запуска файла meteor.py. Убедитесь, что установлен флажок Run with Python Console

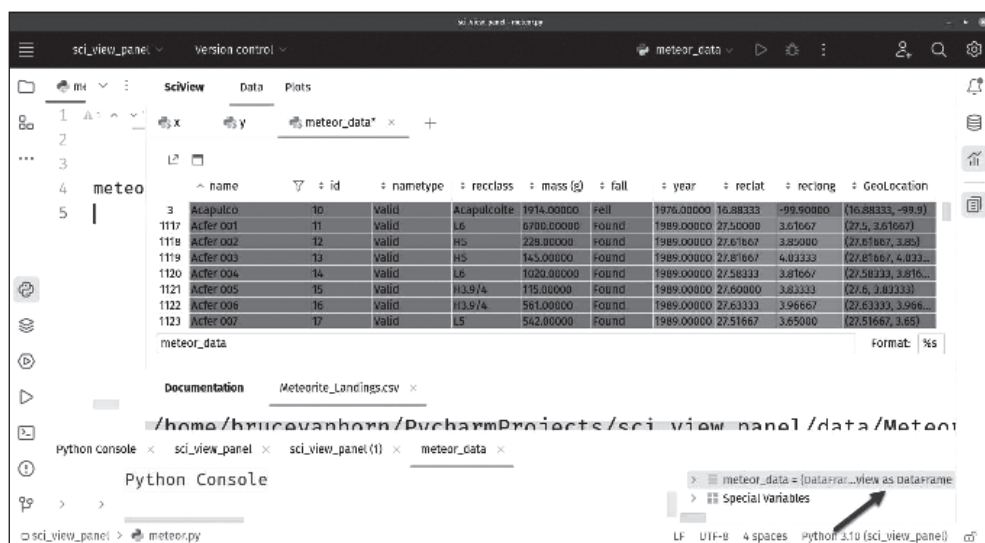


Рис. 13.10. Моя серия данных о метеоритах должна быть масштабирована так, чтобы вы могли видеть все

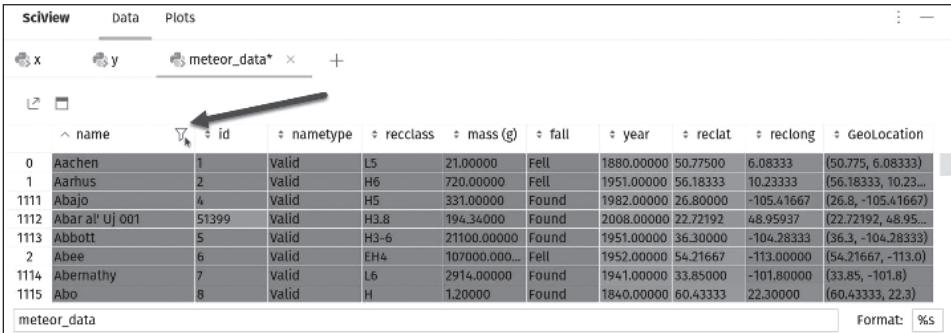
Мне пришлось серьезно потаскать панели, чтобы все это поместилось на экране. Обратите внимание, что на этот раз кнопка, указанная стрелкой, говорит **View as DataFrame**, поскольку на этот раз мы использовали pandas вместо NumPy. Я кликнул по нему, и на вкладке **Data** была загружена DataFrame для просмотра.

Фильтрация на вкладке Data

Набор данных о метеоритах немного более реалистичен. Он содержит много значений, и, возможно, имеет смысл отфильтровать некоторые поля. PyCharm дает вам возможность фильтровать данные, отображаемые на вкладке данных, используя подстановочный знак¹ или выражение².

Фильтрация с помощью подстановочного знака

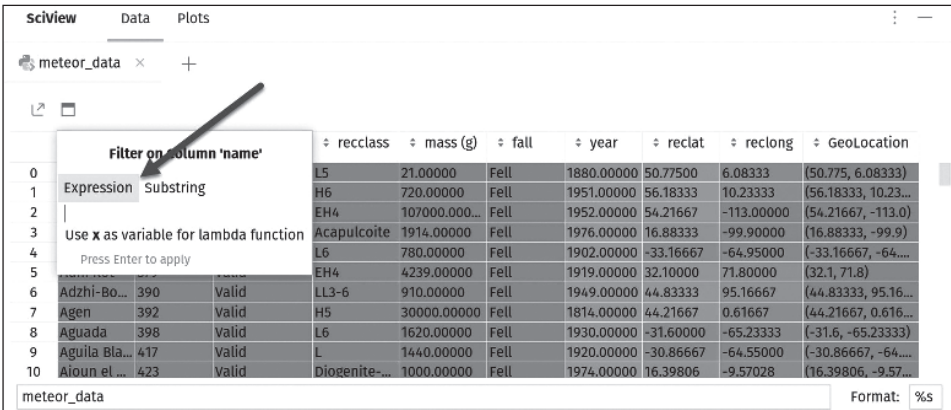
На рис. 13.11 вы можете видеть, что я навожу курсор над столбцом `name` на вкладке **Data**.



	name	id	nametype	recclass	mass (g)	fall	year	reclat	reclong	GeoLocation
0	Aachen	1	Valid	L5	21.00000	Fell	1880.00000	50.77500	6.08333	(50.775, 6.08333)
1	Aarhus	2	Valid	H6	720.00000	Fell	1951.00000	56.18333	10.23333	(56.18333, 10.23...
1111	Abajo	4	Valid	H5	331.00000	Found	1982.00000	26.80000	-105.41667	(26.8, -105.41667)
1112	Abaral' Uj 001	51399	Valid	H3.8	194.34000	Found	2008.00000	22.72192	48.95937	(22.72192, 48.95...
1113	Abbott	5	Valid	H3-6	21100.00000	Found	1951.00000	36.30000	-104.28333	(36.3, -104.28333)
2	Abee	6	Valid	EH4	107000.000...	Fell	1952.00000	54.21667	-113.00000	(54.21667, -113.0)
1114	Abernathy	7	Valid	L6	2914.00000	Found	1941.00000	33.85000	-101.80000	(33.85, -101.8)
1115	Abo	8	Valid	H	1.20000	Found	1840.00000	60.43333	22.30000	(60.43333, 22.3)

Рис. 13.11. При наведении курсора на столбец отображается значок фильтра, напоминающий воронку

Когда я это сделаю, появится значок фильтра. Кликните этот значок, чтобы открыть параметры фильтрации, как показано на рис. 13.12.



	name	id	nametype	recclass	mass (g)	fall	year	reclat	reclong	GeoLocation
0	Aachen	1	Valid	L5	21.00000	Fell	1880.00000	50.77500	6.08333	(50.775, 6.08333)
1	Aarhus	2	Valid	H6	720.00000	Fell	1951.00000	56.18333	10.23333	(56.18333, 10.23...
2	Abajo	4	Valid	EH4	107000.000...	Fell	1952.00000	54.21667	-113.00000	(54.21667, -113.0)
4	Acapulcoite	1914.00000	Valid	L6	780.00000	Fell	1902.00000	-33.16667	-64.95000	(-33.16667, -64...
5	Adzhi-Bo...	390	Valid	EH4	4239.00000	Fell	1919.00000	32.10000	71.80000	(32.1, 71.8)
7	Agan	392	Valid	LL3-6	910.00000	Fell	1949.00000	44.83333	95.16667	(44.83333, 95.16...
8	Agua	398	Valid	L6	1620.00000	Fell	1930.00000	-31.60000	-65.23333	(-31.6, -65.23333)
9	Agula Bla...	417	Valid	L	1440.00000	Fell	1920.00000	-30.86667	-64.55000	(-30.86667, -64...
10	Aloun el ...	423	Valid	Diojenite...	1000.00000	Fell	1974.00000	16.39806	-9.57028	(16.39806, -9.57...

Рис. 13.12. Вы можете фильтровать по выражению или по подстроке (подстановочный знак)

¹ Подстановочный знак – это символ, используемый для замены или представления одного или нескольких символов. – *Прим. ред.*)

² Выражение (от англ. expression) – это любая единица исходного кода Python, которая может быть вычислена интерпретатором для получения значения. При этом литералы и переменные считаются выражениями сами по себе, а вызовы функций и некоторые другие конструкции вроде генераторов следует относить к выражениям из-за того, что они явно или неявно, но дают результат. – *Прим. ред.*

Перейдите на вкладку **Substring** и отфильтруйте по Bal. Вы можете увидеть мой вариант на рис. 13.13.

The screenshot shows the SciView application with the 'Data' tab selected. A data table is displayed with columns: name, id, nametype, recclass, mass (g), fall, year, reclat, reclang, and GeoLocation. A filter dialog box is open over the table, titled 'Filter on column 'name''. The 'Expression' field contains 'Substring' and the 'Filter' field contains 'Bal'. The table shows rows 67 to 26194. The 'name' column contains various meteorite names, and the 'mass (g)' column shows values ranging from 1000.00000 to 13773.00000. The 'fall' column shows 'Fell' or 'Found'. The 'year' column shows years from 1929 to 1976. The 'reclat' and 'reclang' columns show coordinates. The 'GeoLocation' column shows latitude and longitude coordinates. The filter dialog box has a 'Press Enter to apply' button.

Рис. 13.13. Мы фильтруем столбец имени, чтобы отображать все имена, в которых где-то есть символы Bal

Обратите внимание, что значок мусорной корзины позволяет удалить фильтр. Хотя фильтр подстроки может быть полезен, фильтр выражений гораздо более мощный и точный.

Фильтрация с помощью выражений

Используя выражения, вы можете быть немного более избирательными. Метеориты – это круто, но крошечный камешек никого не впечатлит. Например, давайте отфильтруем метеориты весом более 10 кг, как показано на рис. 13.14.

The screenshot shows the SciView application with the 'Data' tab selected. A data table is displayed with columns: name, id, nametype, recclass, mass (g), fall, year, reclat, reclang, and GeoLocation. A filter dialog box is open over the table, titled 'Filter on column 'mass (g)''. The 'Expression' field contains 'x > 10000'. The table shows rows 2 to 59. The 'name' column contains meteorite names, and the 'mass (g)' column shows values ranging from 424 to 50000.00000. The 'fall' column shows 'Fell' or 'Found'. The 'year' column shows years from 1929 to 1976. The 'reclat' and 'reclang' columns show coordinates. The 'GeoLocation' column shows latitude and longitude coordinates. The filter dialog box has a 'Press Enter to apply' button.

Рис. 13.14. Фильтрация по столбцу масс, где x (значение в столбце) больше 10 000

Их довольно много! Теперь давайте отфильтруем более свежие падения, отфильтровав только те, которые произошли с 2010 года. Моя таблица показана на рис. 13.15.

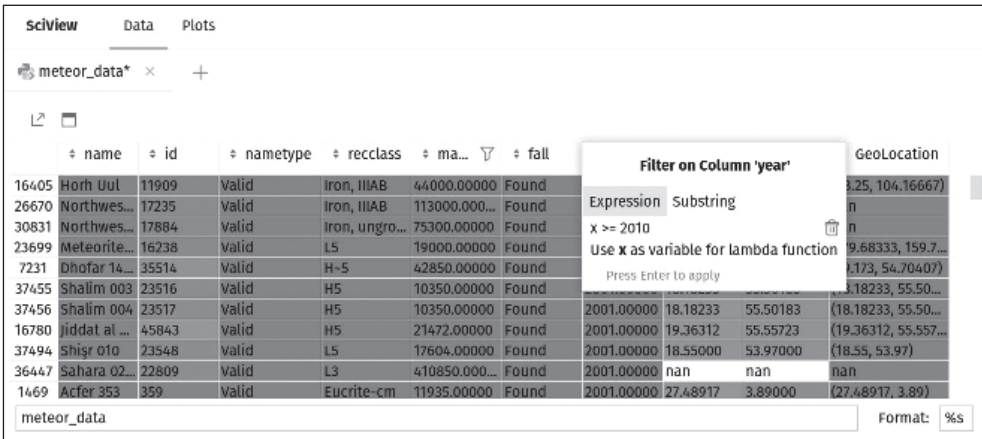


Рис. 13.15. Ограничьте данные только падениями метеоритов в течение и после 2010 года

К счастью, их не так много, и данные заканчиваются только одним падением в 2013 году. По совпадению это последний год, когда я вовремя заплатил подоходный налог. Я же говорил вам, что это все я!

Уважаемая налоговая служба!

Эту последнюю строчку мы в писательском бизнесе называем «шуткой». Я знаю, что для вас всех это чуждое понятие, но поверьте мне, все остальные засмеялись.

Они точно следят за мной.

В любом случае еще одна вещь, которую мы можем увидеть в ходе нашего эксперимента по фильтрации, – это отсутствие данных в полях Reclat, Reclong и GeoLocation. Без сомнения, они были зафиксированы в Area 51, где данные геолокации являются незаконными. Тем не менее это может быть быстрый способ проверить ваши данные на наличие аномалий в рамках создания конвейера данных, который мы рассмотрим в следующей главе.

Экспорт в файлы или новые вкладки

Взгляните на верхнюю часть вкладки **Data** на рис. 13.16.

Вкладка **Data** позволяет экспортировать данные в файл .csv или .tsv. Это довольно просто, поэтому я думаю, что без скриншота можно обойтись. Вы также можете перенести содержимое вкладки данных в SciView на обычную вкладку в представлении редактора. Я сделал это на рис. 13.17.

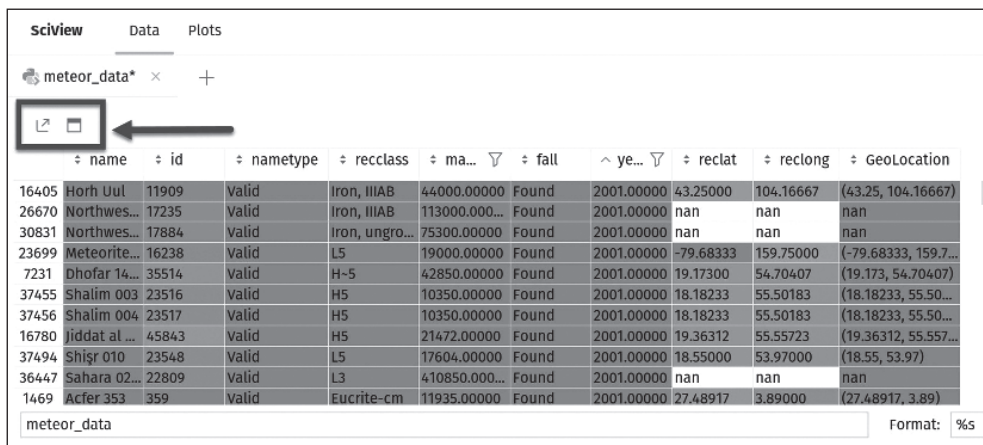


Рис. 13.16. Вы можете экспортировать свои данные или вывести их на новую вкладку в окне редактора

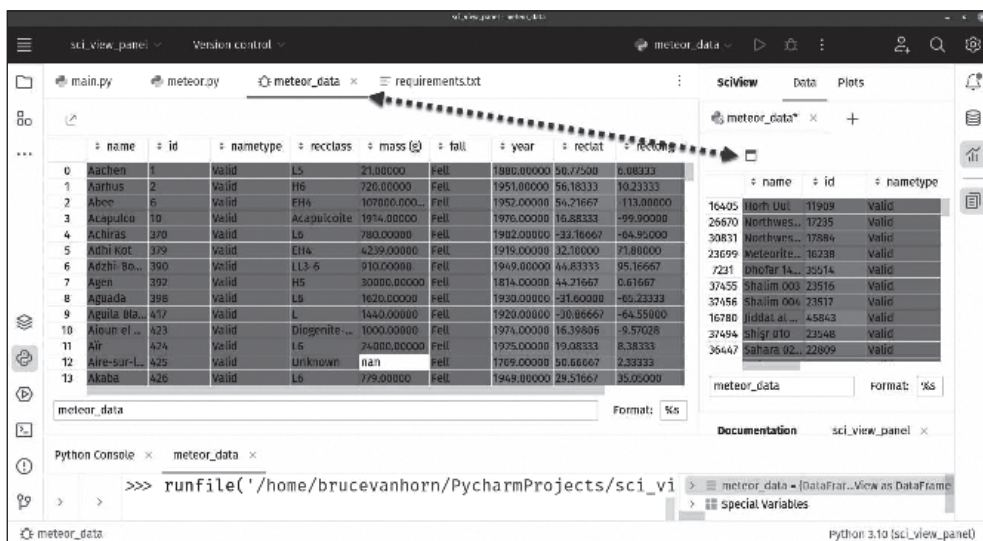


Рис. 13.17. Я поместил содержимое вкладки SciView Data на вкладку в главном окне редактора

В целом вкладка **Data** в SciView содержит множество полезных функций для выполнения исследований ваших данных, будь то необработанные, как в случае данных о метеорах, или сгенерированные посредством вычислений, как в случае с нашим предыдущим случайным набором. Когда вы объедините этот графический способ просмотра входных и выходных данных с режимом ячеек PyCharm, описанным в последней главе, вы начнете видеть мощь инструментов обработки данных в PyCharm. Это всего лишь несколько простых панелей, но с ними можно сделать так много!

ПОНИМАНИЕ IPYTHON И МАГИЧЕСКИХ КОМАНД

IPython – это расширенная интерактивная оболочка для языка программирования Python. Он обеспечивает более многофункциональную и удобную для пользователя среду по сравнению с интерактивным интерпретатором Python по умолчанию. IPython был разработан, чтобы сделать интерактивные вычисления и задачи анализа данных более удобными и эффективными.

Работая с IPython за пределами PyCharm, вы обнаружите некоторые функции, которых нет ни в одной стандартной консоли или среде REPL. Интерфейс оболочки значительно улучшен и теперь поддерживает функции, ожидаемые от PyCharm, такие как завершение табуляции, подсветка синтаксиса и навигация по истории.

Этот инструмент предоставляет множество богатых возможностей отображения. Представьте себе интерфейс командной строки, который включает возможность работы с изображениями, видео-, аудио- и интерактивными виджетами непосредственно в интерактивной среде. Очевидно, что вы можете прогонять код Python в виде фрагментов, выражений или целых скриптов, и все это делать в интерактивном режиме.

Python также упрощает работу в распределенной вычислительной среде. Как вы, возможно, знаете, потоковые вычисления в большинстве основных интерпретаторов Python из-за **глобальной блокировки интерпретатора (GIL)** фактически отключены. GIL – это механизм интерпретатора CPython, наиболее широко используемой реализации Python. GIL – это мьютекс (или блокировка), который позволяет одновременно выполняться в интерпретаторе только одному потоку, даже в многоядерных системах. Это означает, что в многопоточной программе Python только один поток может выполнять байт-код Python в любой момент времени независимо от того, сколько ядер CPU доступно.

GIL был введен для упрощения управления памятью и предотвращения потенциальных конфликтов, которые могут возникнуть из-за одновременного доступа нескольких потоков к объектам Python и их изменения. Хотя GIL гарантирует, что управление памятью Python остается более простым и безопасным, он также имеет последствия для многопоточных программ, поскольку программам Python, как правило, сложно фактически использовать все вычислительные ресурсы, доступные в системе, поддерживающей несколько потоков.

Хотя у IPython нет какого-либо секретного решения, позволяющего обойти ограничение GIL, он предоставляет модуль под названием `ipython.parallel`, который упрощает работу с ресурсами параллельных вычислений. Вы можете создавать кластеры механизмов IPython, которые могут параллельно запускать код на нескольких процессах или даже на нескольких машинах.

Кажется, что IPython нас должен очень заинтересовать, учитывая, что это, очевидно, мощный инструмент, который более полно обеспечивает режим обогащенного пошагового взаимодействия с данными, который является образцом работы современных рабочих процессов обработки данных. Тогда возникает постоянный вопрос: как мы можем использовать IPython в PyCharm?

Установка и настройка IPython

Установка IPython выполняется так же, как и любая другая библиотека Python. Вы можете использовать `pip install ipython` из терминала PyCharm или использовать экран управления пакетами, с которым мы работаем начиная с главы 3. На рис. 13.18 показано, как я собираюсь выполнить установку с помощью панели **Python Packages** в PyCharm.

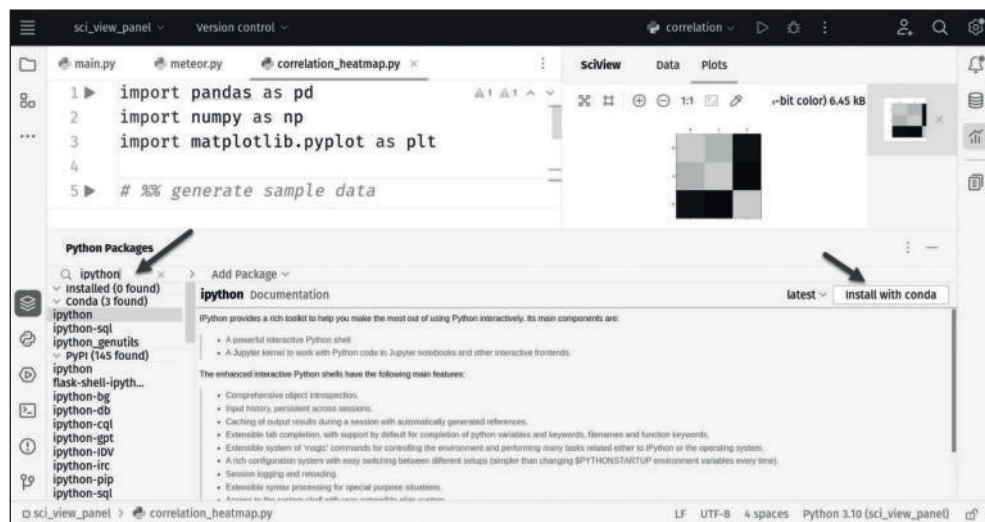


Рис. 13.18. Я нашел `ipython` на панели **Python Packages** и собираюсь установить его в свой проект с помощью `conda`

Если вы планируете часто работать с IPython, можете установить его глобально, чтобы он был доступен для всех ваших проектов. Вы можете сделать это, просто установив IPython за пределами виртуальной среды, используя `pip`, `conda` или любой другой инструмент, который вы предпочитаете для управления пакетами.

Как только IPython станет доступен, PyCharm будет использовать его согласно настройкам по умолчанию в настройках **Build, Execution, Deployment** среды IDE, как показано на рис. 13.19.

Мы подробно рассмотрели настройки в главе 2, поэтому, если вы ее пропустили и не знаете, как добраться до настроек, просмотрите предыдущую главу. Как только IPython станет доступен в вашем проекте, необходимо закрыть или переключить все консоли, которые могут быть открыты. На данный момент мы запустили три программы, которые намеренно настроили для запуска в консоли, поэтому вы не увидите никаких изменений в них, поскольку они работали до того, как мы установили IPython. На рис. 13.20 показано, как выглядит приглашение IPython.

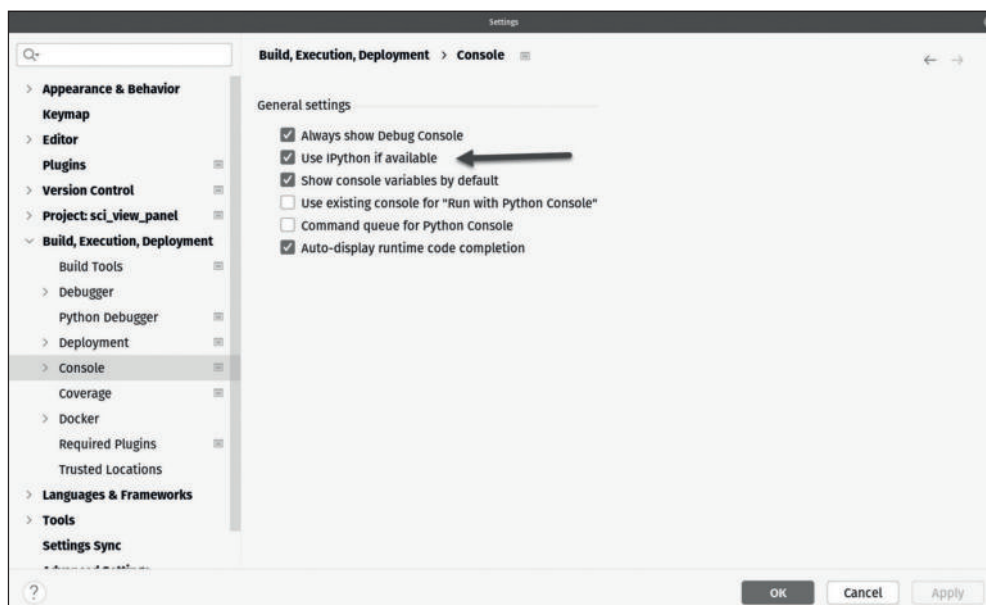


Рис. 13.19. По умолчанию PyCharm будет использовать IPython для своей консоли, если она доступна

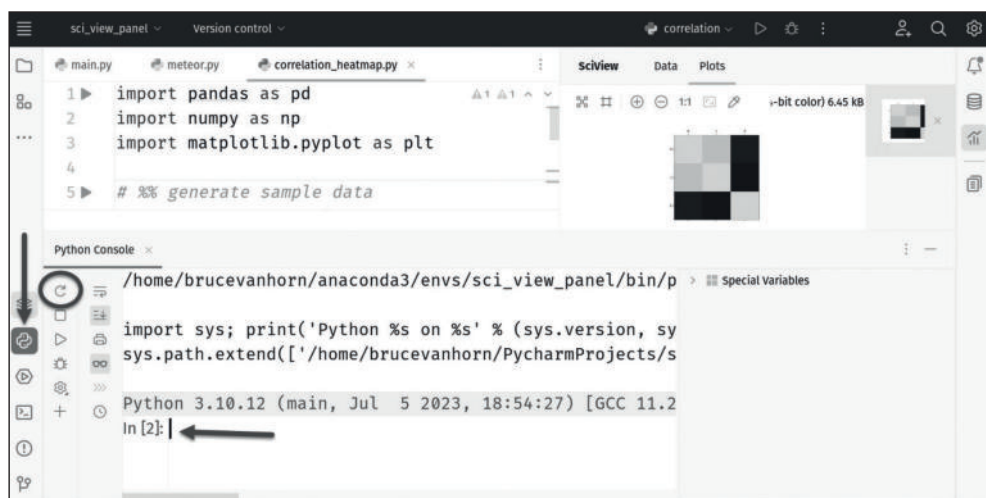


Рис. 13.20. Приглашение IPython появляется после закрытия или перезапуска существующей консоли

На рис. 13.20 я нарисовал стрелку, чтобы напомнить вам, как открыть консоль, если она еще не открыта. Если вы не видите этот значок, попробуйте кликнуть значок с тремя точками над стрелкой. Если вы все еще не можете ее найти, просмотрите главу 2, где мы впервые рассказываем о работе с консолью.

Как я уже говорил, если у вас уже открыты консоли, их необходимо перезапустить. Я обвел эту кнопку на рис. 13.20. При перезапуске консоли вы должны

увидеть другое приглашение, начинающееся со слова `In`, в отличие от обычного приглашения, в котором отображаются символы `>>>`. После слова `In` вы увидите несколько квадратных скобок, содержащих число. На рис. 13.20 показано 2. Это число указывает порядок введенных на данный момент команд. Если вы посмотрите на верхнюю часть новой консоли IPython, то увидите, что несколько команд были введены автоматически – они импортируют библиотеку `sys`, распечатывают версию Python и расширяют наш путь. Это три команды, но помните, что мы начинаем с 0. Следующая команда, которую вы введете, будет четвертой и будет помечена как 3.

Если бы PyCharm был видеоигрой, установка IPython была бы подобна открытию суперпушки, меча или магической силы, которую вы можете использовать, чтобы победить своих ничего не подозревающих и совершенно неподготовленных врагов. Если вы играете впервые, вам, скорее всего, не терпится взорвать зомби данных, поэтому давайте взглянем на магические команды IPython. Никаких шуток, *магические команды* буквально так и называются!

Знакомство с IPython и магическими командами

В IPython **магическая команда** – это специальная команда, которая начинается с символа `%` (для магических операций в строке) или `%%` (для магических операций в ячейках) и используется в сеансе IPython или в ячейке блокнота Jupyter. Эти волшебные команды предоставляют ярлыки для выполнения различных задач: от запуска кода с определенными параметрами до взаимодействия со средой или управления поведением IPython. Магические команды – это удобный способ выполнять стандартные операции без написания обширного кода.

Существует два типа магических команд.

- **Магия строки.** Магия строки начинается с одного символа `%` и используется в одной строке. Обычно она использует аргументы и варианты, чтобы изменить свое поведение. Например, `%run script.py` можно использовать для запуска внешнего скрипта в сеансе IPython.
- **Магия ячейки.** Магия ячейки имеет префикс `%%` и используется в начале ячейки. Она позволяет влиять на содержимое всей ячейки. Например, `%%time` можно использовать для измерения времени выполнения всего кода ячейки.

Вот несколько примеров часто применяющихся магических команд, которые вы будете использовать ежедневно:

- `%run script.py`: запускает внешний скрипт Python в сеансе IPython,
- `%timeit`: измеряет время выполнения оператора или выражения Python,
- `%matplotlib inline`: настраивает `matplotlib` для отображения графиков непосредственно в блокноте,
- `%writefile filename.txt`: записывает содержимое ячейки блокнота в файл с именем `filename.txt`,
- `%load_ext Extension_name`: загружает определенное расширение IPython,
- `%%html`: отображает содержимое ячейки в формате HTML,
- `%reset`: сбрасывает пространство имен, удаляя все имена, определенные пользователем,

- `object_name?`. Не поймите меня буквально. Вы не вводите `object_name?`. Вместо этого вы вводите имя объекта, экземпляра которого вы создали. Может, нам стоит сделать это через минуту, чтобы вы поняли.

К настоящему моменту вы начинаете чувствовать запах того, что мы здесь готовим. Мы видели режим ячеек в главе 12, но теперь благодаря IPython мы можем сделать с ячейкой гораздо больше, используя эти магические команды. Как и любое хорошее магическое заклинание из фильмов, здесь достаточно загадочного, но короткого синтаксиса, который нужно выучить, что добавляет загадочности и заставляет вас чувствовать себя немного более продвинутым, чем ваши коллеги, не использующие магию.

Давайте попробуем, хорошо?

В консоли IPython введите `a = 1`. Теперь давайте попробуем заклинание `object_name?`. Вы не вводите *имя_объекта?*; вместо этого вы вводите `?`. Имя объекта – `a`, после которого ставите вопросительный знак, как если бы вы были канадцем или австралийцем¹. Какое бы имя объекта вы ни использовали, если оно допустимо, вы получите информацию об этом объекте. Мой результат можете увидеть на рис. 13.21.

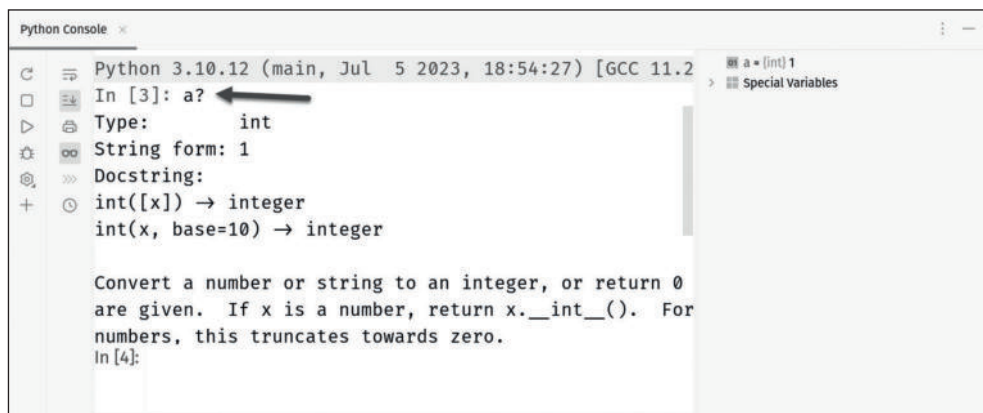


Рис. 13.21. Как дела, `a`?

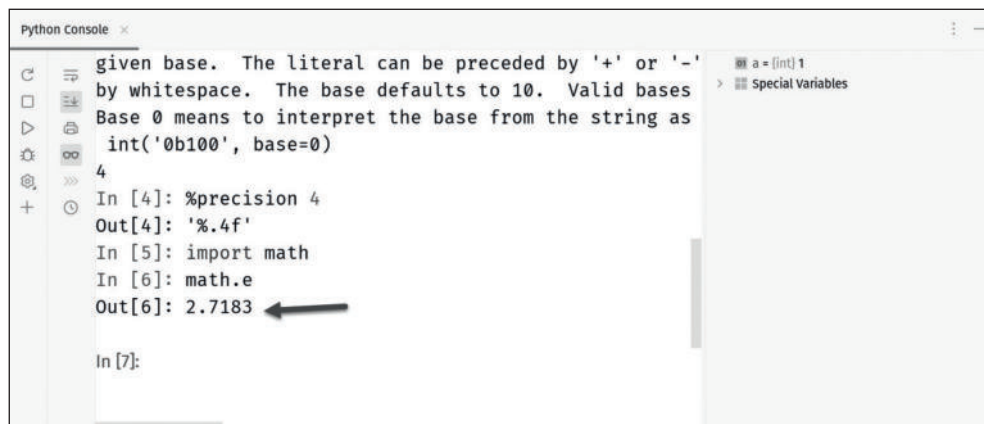
Мы много узнали об `a`, `a?`. Хорошо, давайте попробуем еще одну. Введите `%precision 4`. Результатом должно быть `'%.4f'`. Это говорит нам о том, что мы установили для нашего сеанса точность до четырех десятичных знаков. Давайте проверим это, введя следующий код:

```
import math
math.e
```

Строчная `e` – это ссылка на **константу Эйлера**, которая является основой натурального логарифма. Она похожа на число «пи» тем, что представляет со-

¹ Когда канадцы или австралийцы говорят «а?», это может означать, что они подтверждают, слушает другой человек или нет. Говоря «а?», они вселяют уверенность в получении ответа от слушателя. – Прим. ред.

бой бесконечную десятичную дробь без периода¹. Константа e с точностью до 10 знаков равна 2,7182818285. Однако мы настроены на точность до 4 знаков после запятой, поэтому результат IPython должен показывать 2,7183, как показано на рис. 13.22².



```

Python Console x
given base. The literal can be preceded by '+' or '-'
by whitespace. The base defaults to 10. Valid bases
Base 0 means to interpret the base from the string as
int('0b100', base=0)
4
In [4]: %precision 4
Out[4]: '%.4f'
In [5]: import math
In [6]: math.e
Out[6]: 2.7183
In [7]:
  
```

Рис. 13.22. Установив точность до четырех знаков после запятой, мы видим, что константа e отображается правильно

Давайте попробуем еще одну магическую команду. Магия `%timeit` очень полезна при создании алгоритмов. Алгоритмы – это просто повторяемые наборы инструкций; однако, чтобы быть полезным, алгоритм должен быть эффективным. Эффективность оценивается по двум параметрам: времени, необходимом для его завершения, и пространством, необходимым для его работы. Например, **Национальная система здравоохранения (NHS)** в Англии разработала алгоритм, который сопоставляет доноров органов с реципиентами на основе ряда сложных факторов. Его цель – подобрать орган к донору таким образом, чтобы среди прочих факторов свести к минимуму риск отторжения ткани. Как вы понимаете, это сложный алгоритм, который должен работать с набором данных, содержащим тысячи зарегистрированных доноров и тысячи зарегистрированных получателей. Когда орган становится доступным, его срок годности ограничен. Если алгоритму сопоставления (*matching algorithm*) требуется слишком много времени для создания совпадения, орган становится непригодным для трансплантации, и алгоритм фактически бесполезен. Мы рассмотрели профилирование в главе 6, оно предназначено для процессов разработки программного обеспечения, чтобы помочь найти узкие места в исполнении.

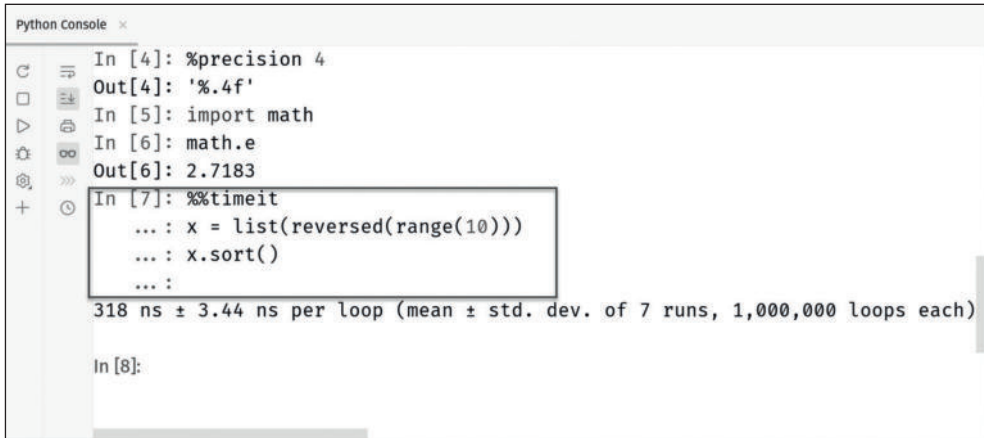
В науке о данных мы работаем за пределами типичного программного проекта, поэтому имеет смысл рассчитать время для отдельных шагов, чтобы получить хотя бы некоторое представление о том, как может масштабироваться шаг наших вычислений. Даже если нашей целью не является создание алго-

¹ То есть является иррациональным и, более того, трансцендентным, как и число «пи». – Прим. ред.

² Обратите внимание, что сделано округление. – Прим. ред.

ритма, нам все равно придется иметь дело с практическими ограничениями по времени и вычислительному пространству (например, оперативной и постоянной памяти) в нашей обычной исследовательской работе.

Магия `%%timeit` становится неотъемлемой частью нашей работы. Теперь, когда я потратил все это время на создание огромного примера, давайте попробуем его на чем-то смехотворно тривиальном, показанном на рис. 13.23.



```

Python Console x
In [4]: %precision 4
Out[4]: '%.4f'
In [5]: import math
In [6]: math.e
Out[6]: 2.7183
In [7]: %%timeit
... : x = list(reversed(range(10)))
... : x.sort()
... :
318 ns ± 3.44 ns per loop (mean ± std. dev. of 7 runs, 1,000,000 loops each)
In [8]:
  
```

Рис. 13.23. Мы пробуем `%%timeit` с простой сортировкой, и определяем, что операция завершается примерно за 318 нс

Это действительно была тривиальная задача. Начните с ввода `%%timeit` в консоли IPython. Нажимая *Enter*, вы переходите на следующую строку. Во второй строке мы создаем список с числами от 0 до 9 в обратном порядке. В последней строке сортируем их обратно в нормальный порядок. Это слишком скучно! Нажмите *Enter* в последней пустой строке, чтобы запустить код. Посмотрите, что `%%timeit` сделал! Он мог бы просто запустить часы, запустить скрипт, затем остановить часы и сообщить о разнице. Вместо этого `%%timeit` запускал этот код снова и снова, чтобы получить размер выборки, при котором можно вычислить среднее и стандартное отклонение. Это дает нам лучшее и более точное представление о том, сколько времени займет выполнение этого кода. Естественно, разные компьютеры будут работать по-разному. Для меня это заняло 318 нс плюс-минус 3,44 нс.

Здесь мы рассмотрели три наиболее распространенные магические команды в IPython. Конечно, есть много других полезных команд, которыми вы можете воспользоваться. Их можно найти в официальной документации IPython: <https://ipython.readthedocs.io/en/stable/index.html>.

Основная цель IPython – это не просто дать вам возможность использовать удобные API-интерфейсы для выполнения конкретных задач, таких как проверка переменных, форматирование или профилирование, – IPython фактически использует эти функции для реализации своих основных интерактивных характеристик. В контексте проектов науки о данных IPython при использовании в PyCharm предлагает нам отличный способ проверять и тестировать небольшие блоки кода перед их использованием в большой программе.

На этом давайте перейдем к следующему разделу, где мы рассмотрим еще одну заметную поддержку, которую PyCharm предлагает для научных вычислений, – блокноты Jupyter.

ИСПОЛЬЗОВАНИЕ БЛОКНОТОВ JUPYTER

Jupyter notebooks – пожалуй, наиболее часто используемый инструмент в научных вычислениях Python и проектах по обработке данных. В этом разделе мы кратко обсудим основы блокнотов Jupyter, а также причины, по которым они являются отличным инструментом для анализа данных. Затем мы рассмотрим, как PyCharm поддерживает использование этих блокнотов.

Мы будем работать с проектом `jupyter_notebooks` в исходном коде главы. Не забывайте, что вам нужно будет установить требования в файле `requirements.txt` в виртуальной среде, чтобы использовать пример проекта. Если нужно узнать, как это сделать, вернитесь к главе 3.

Несмотря на то что мы будем писать код в блокнотах Jupyter, полезно сначала рассмотреть простую программу в традиционном скрипте Python, чтобы мы могли в полной мере оценить преимущества использования блокнота позже. Давайте посмотрим на файл `main.py` и как с ним можно работать. Мы видим, что этот файл содержит ту же программу из предыдущего раздела, где мы случайным образом генерируем набор данных из трех атрибутов (x , y и z) и рассматриваем их корреляционную матрицу:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# Generate sample data
x = np.random.rand(50,)
y = x * 2 + np.random.normal(0, 0.3, 50)
z = np.random.rand(50,)

df = pd.DataFrame({
    'x': x,
    'y': y,
    'z': z
})

# Compute and show correlation matrix
corr_mat = df.corr()

plt.matshow(corr_mat)
plt.show()
```

Это примерно тот же код, который мы использовали ранее в этой главе, когда хотели показать функции тепловой карты в PyCharm при просмотре корреляционной матрицы. Здесь мы добавили две последние строки, которые отличаются. Вместо тепловой карты на этот раз мы рисуем диаграмму рассеяния. Ранее я объяснял, что мы намеренно и искусственно ввели корреляцию в наши случайно сгенерированные данные. Посмотрите, где мы установили y , и вы уви-

дите, что мы умножили матрицу x на 2, а затем добавили несколько небольших чисел из другой генерации случайной выборки. В результате у будет выглядеть примерно (но не точно) коррелирующим с x , что позволит нам увидеть правдоподобную, хотя и надуманную корреляционную матрицу. Это подтверждается, когда мы запускаем файл. Мой результат показан на рис. 13.24. Не забывайте, что наши данные случайны, поэтому ваши не будут точно совпадать с моими.



Рис. 13.24. Это диаграмма рассеяния, созданная нашим кодом

Как я уже сказал, здесь нет ничего нового или удивительного, кроме перехода от тепловой карты к диаграмме разброса. Мы использовали это для сравнения при обсуждении блокнотов Jupyter.

ПОНИМАНИЕ ОСНОВ JUPYTER

Блокноты Jupyter построены на идее итеративной разработки. В любых усилиях по разработке, независимо от того, что вы создаете, разбиение большого проекта на более мелкие части всегда приносит плоды. Никто в Ford Motor Company не занимается производством автомобилей. Они делают миллионы деталей, которые позже собираются в автомобиль. Каждая деталь может быть спроектирована, изготовлена, протестирована и проверена как отдельная деталь.

Аналогичным образом, разделив программу на отдельные части, которые можно писать и запускать независимо друг от друга, программисты в целом и специалисты по обработке данных в частности могут поэтапно работать над логикой своих программ.

ИДЕЯ ИТЕРАТИВНОЙ РАЗРАБОТКИ

Большинство практиков в мире разработки программного обеспечения привыкли к идеалам, лежащим в основе гибких методологий. Существуют десят-

ки гибких фреймворков, призванных помочь вам управлять программным обеспечением или любым проектом с целью создания какого-либо полезного продукта. Единственное, что их постоянно объединяет, – это идея итеративной разработки. Усилия по разработке разбиваются на более мелкие и простые задачи, называемые итерацией. В конце каждой итерации у нас должен быть какой-то полезный продукт. Это важно!

Вы можете подумать о создании автомобиля, но подумайте по-настоящему об исследованиях и разработках, которые были проведены при создании первых автомобилей. Нашей главной заботой в то время было создание транспортного средства, которое доставило бы нас из точки А в какую-то отдаленную точку Б таким образом, чтобы это было бы быстрее и эффективнее, чем использование гужевого транспорта.

Мы уже знаем о колесе, поэтому давайте на несколько комичном уровне рассмотрим процесс, который мог бы использовать средний разработчик программного обеспечения, если бы он собирался создать первый в мире автомобиль. Помните, что каждая итерация должна создавать какие-то удобные средства перемещения с места на место. Наша команда разработчиков могла бы начать со скейтборда. Это что-то простое, что мы могли бы сделать за короткую итерацию. Следующая итерация может привести к появлению самоката, следующая – велосипеда, следующая – трехколесного велосипеда с электроприводом и, наконец, простейшего автомобиля (см. рис. 13.25).

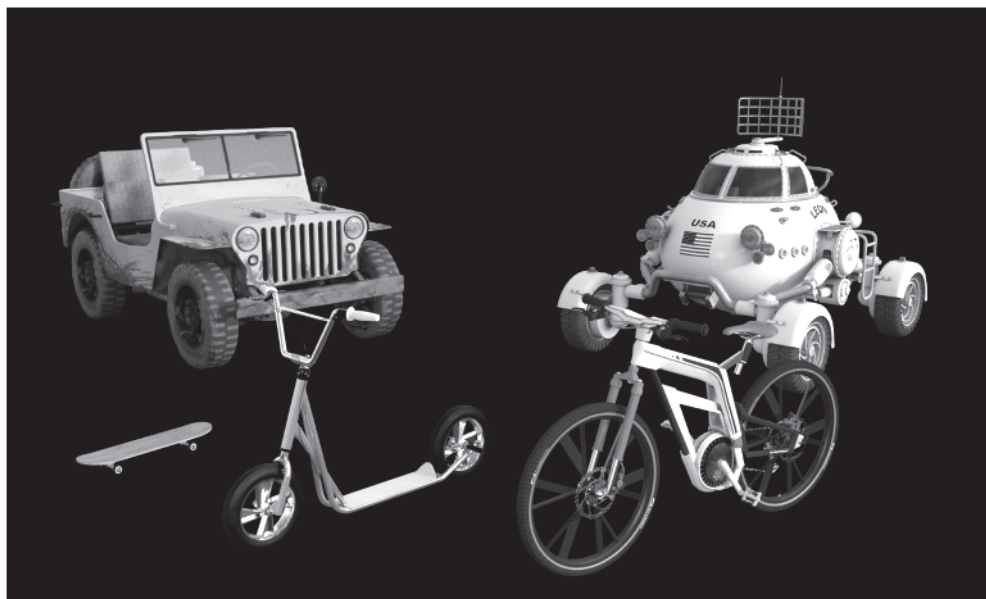


Рис. 13.25. Итерационный процесс реализует выполнение поставленной задачи шаг за шагом

Каждая итерация дает полезный результат, и мы можем назвать это непрерывным прогрессом. В конце каждой итерации вам следует задуматься о следующей, потому что с каждой итерацией ваше понимание конечной цели

становится яснее. Вы многому учитесь на каждой итерации, и, если ваши итерации будут небольшими, будет возможность изменить направление, когда вы обнаружите, что пошли не туда.

Это происходит постоянно при традиционной разработке программного обеспечения и в работе с данными. Вы можете поставить перед собой конкретную цель исследования, но в конечном итоге придется идти туда, куда вас ведут данные, а не по собственному произволу. Итеративные процессы делают это возможным.

Мы можем применить это к идеям блокнотов Jupyter, поскольку они предполагают построение вашей работы по одной ячейке или итерации за раз. Во время каждой итерации вы добавляете или вносите соответствующие изменения в ячейку кода, которая считывает набор данных, и перезапускаете последующие ячейки, а не весь код перед ней. В знак уважения к пользователям блокноты Jupyter были названы в честь трех наиболее распространенных языков научного программирования: **Julia**, **Python** и **R**.

Еще одной неотъемлемой частью блокнотов Jupyter является поддержка языка Markdown. Как мы упоминали ранее в начале предыдущей главы, Markdown – это язык разметки, который обычно используется в файлах README.md на GitHub. Кроме того, из-за способности работать с набором макрорасширений LaTeX (который обычно используется для написания математических уравнений и научных статей в целом) Markdown пользуется большой популярностью в сообществе специалистов по обработке и анализу данных.

Теперь давайте посмотрим, как мы можем использовать блокнот Jupyter в обычном проекте PyCharm.

Блокноты JUPYTER В PYCHARM

Для этой задачи мы переведем программу, которая есть в файле main.py, в блокнот Jupyter, чтобы можно было увидеть интерфейс, который предлагает Jupyter, в отличие от традиционного скрипта Python. Я буду использовать свой существующий проект jupyter_notebooks, с которого мы начали. Вы можете найти его в примере кода главы. Если у вас нет этого репозитория, мы рассмотрим его клонирование в главе 2. Если вы хотите начать с нуля – можете просто создать новый научный проект, как описано в главе 12.

Создаем блокнот и добавляем наш код

Чтобы добавить новый блокнот Jupyter в проект PyCharm, создайте его, как если бы это был просто файл. Нажмите **File | New | Jupyter Notebook**, как показано на рис. 13.26.

Вам сразу же будет предложено дать имя блокноту. Я назвал свой Basic.ipynb. Файл был создан в корневой папке моего проекта, но я перетащил его в папку notebooks. Вы можете увидеть мою отправную точку на рис. 13.27.

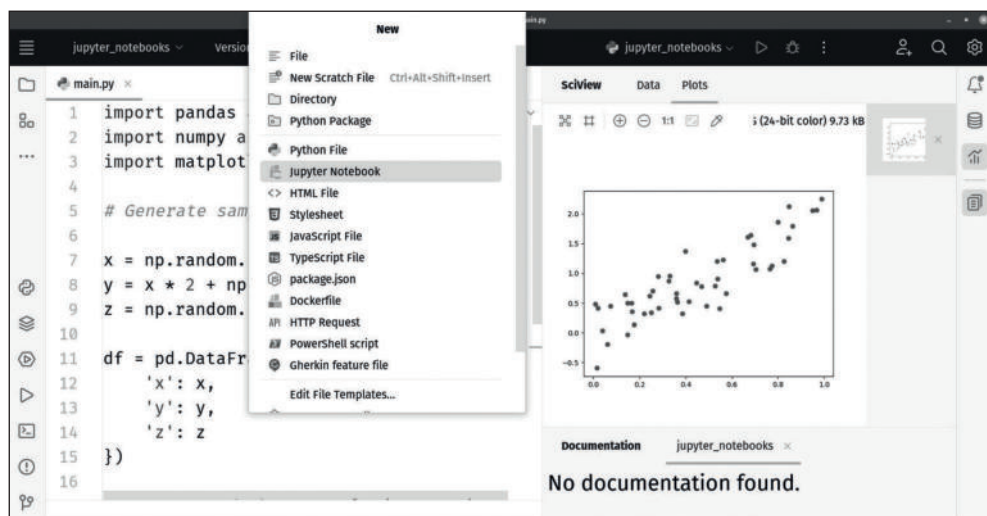


Рис. 13.26. Создайте новый блокнот Jupyter, используя File | New | Jupyter Notebook

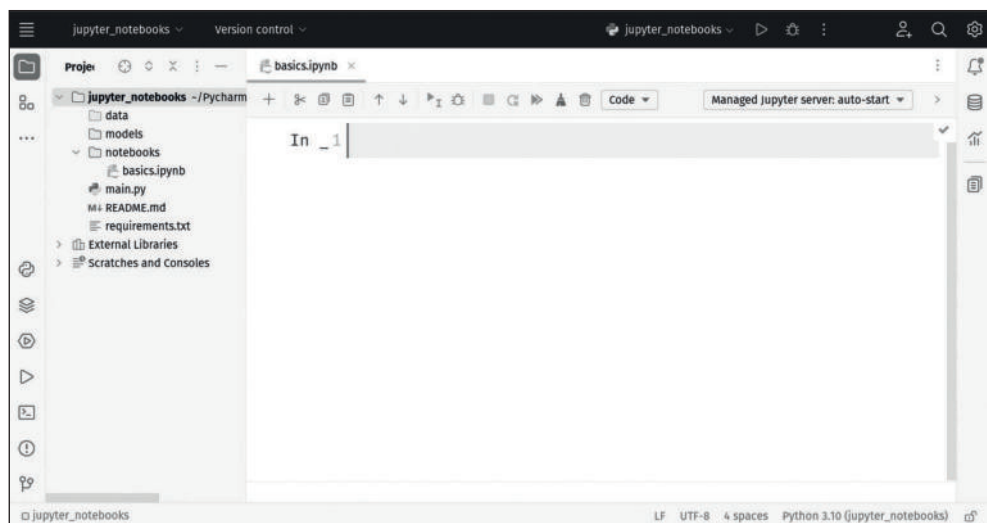


Рис. 13.27. Мой новый блокнот готов!

Давайте начнем с ввода некоторой документации вместе с некоторыми известными импортами. В командной строке введите этот код:

```

### Importing libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
###
  
```

Когда вы введете последние символы `%%`, то обнаружите, что PyCharm создаст для вас новую ячейку. Во вторую ячейку давайте введем часть кода, который был ранее в нашем иллюстративном скрипте Python:

```
x = np.random.rand(50,)
y = x * 2 + np.random.normal(0, 0.3, 50)
z = np.random.rand(50,)

df = pd.DataFrame({
    'x': x,
    'y': y,
    'z': z
})
%%
```

Я объяснил данный код в этой главе ранее, поэтому больше не буду этого делать. Как и раньше, последние `%%` создадут новую ячейку. Смотрите рис. 13.28, чтобы увидеть, что у меня есть на данный момент.

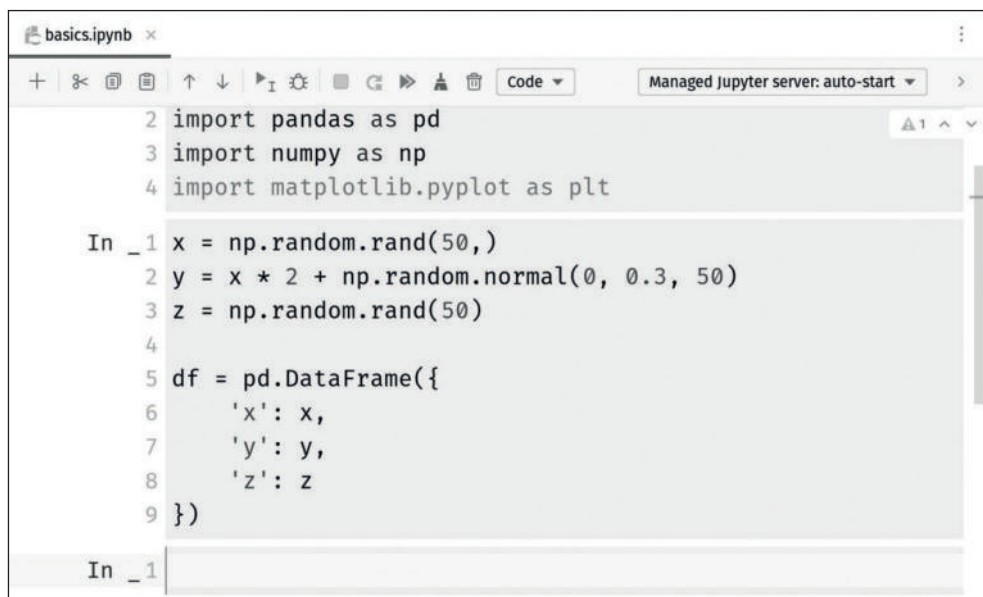


Рис. 13.28. Теперь у меня есть две ячейки, содержащие код из нашей предыдущей программы

Все идет нормально! У меня есть импорт и ячейка, которая генерирует набор данных, как это было в моем скрипте Python. Я разбил свою программу на небольшие итеративные шаги. Сначала мой импорт, затем набор данных. Далее я могу посмотреть, как что-нибудь сделать с моими данными. Из предыдущих обсуждений мы знаем, что это будет корреляционная матрица. Что, если это университетский проект, и наш профессор хочет, чтобы мы задокументировали формулы, которые используем? Вероятно, это хорошая идея, даже если вы не ходите на лекции в университет. Давайте на минутку рассмотрим интересную функцию документации, которую мы можем использовать. Мы знаем, что

у нас могут быть ячейки кода. У нас также могут быть ячейки документации, которые используют не только Markdown, но и LaTeX.

Документирование с помощью Markdown и LaTeX

Markdown – это то, что мы уже видели. Это простой язык разметки, который позволяет создавать HTML-подобные документы, но вместо богатого набора тегов для обозначения документации вы используете символы. Markdown охватывает только базовые элементы, такие как заголовки, списки и простые изображения. Мы рассмотрели использование встроенного в PyCharm плагина Markdown в главе 12.

Работа с Markdown в наших блокнотах позволяет нам использовать простое форматирование, чтобы улучшить внешний вид и читабельность наших блокнотов.

LaTeX, который произносится как «LAY-tech» или «LAN-tech», в зависимости от того, где учился ваш профессор математики, представляет собой соглашение о наборе текста, используемое в области науки и математики. Подобные вещи необходимы в академических журналах, потому что вы не можете легко набирать сложные формулы на клавиатуре, поскольку очень немногие профессора математики умеют использовать Adobe Illustrator и создавать свои собственные иллюстрации. Вместо графического инструмента они изобрели краткое, трудное для понимания и невозможное для запоминания соглашение о разметке своих формул в журнальных статьях. Затем они назвали это общепринятым английским словом, но настояли на том, чтобы мы все произносили его неправильно.

Давайте попробуем! Во-первых, я не совсем откровенно рассказал вам, как создавать ячейки. Конечно, можете просто начать вводить данные, и использование `#%%` для разделения ячеек работает нормально. Вы также можете навести указатель мыши на пространство между ячейками или на то место, где будет разделение, если вы находитесь в верхней или нижней части блокнота. Посмотрите на рис. 13.29.

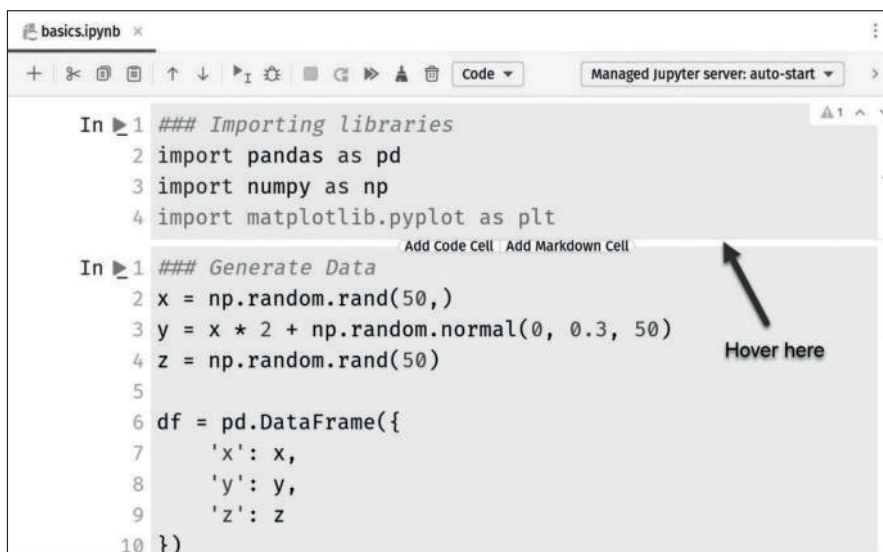


Рис. 13.29. Вы можете навести указатель мыши на пространство между ячейками, чтобы в графическом интерфейсе получить помощь при добавлении новых ячеек

Когда вы нажимаете эти кнопки между ячейками, это запускает процесс деления клеток, называемый митозом. Подождите, нет, это не биология. Это действие разделяет ячейки, добавляя новую между двумя, которые у нас уже есть. Я собираюсь добавить ячейку ниже последней, наведя курсор чуть ниже последней ячейки и нажав **Add Markdown Cell** (см. рис. 13.30).

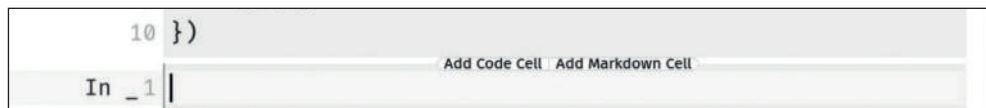


Рис. 13.30. Наведите указатель мыши на последнюю пустую ячейку и нажмите Add Markdown Cell

Это создаст голубую ячейку без запроса IPython. Поскольку это ячейка Markdown, PyCharm не ожидает кода, поэтому запрос не требуется. Теперь внутри ячейки введите эту абсолютную ерунду:

```
### Pearson's correlation
$r_{XY}
= \frac{\sum_{i=1}^n \{(X_i - \bar{X})(Y_i - \bar{Y})\}}{\sqrt{\sum_{i=1}^n \{(X_i - \bar{X})^2\}} \sqrt{\sum_{i=1}^n \{(Y_i - \bar{Y})^2\}}}
```

Это синтаксис разметки LaTeX, который передает формулу корреляции Пирсона. Поверьте мне, через мгновение, когда мы запустим наш ноутбук, это будет весьма впечатляюще. Давайте продолжим и добавим две диаграммы, созданные ранее.

Добавляем наши диаграммы

Добавьте новую ячейку кода ниже последней и добавьте этот код для нашей тепловой карты:

```
# Compute and show correlation matrix
corr_mat = df.corr()

plt.matshow(corr_mat)
plt.show()
```

Затем добавьте еще одну ячейку кода для нашей диаграммы рассеяния:

```
# Scatterplot
plt.scatter(df['x'], df['y'])
plt.show()
```

Реализация нашего кода в виде блокнота Jupyter завершена.

Запуск ячеек

Вы можете запустить блокнот, нажав кнопку **Run** в верхней части блокнота, как показано на рис. 13.31.

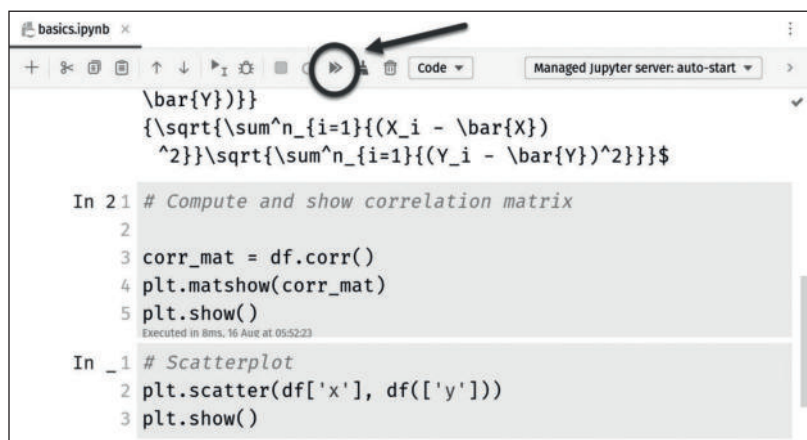


Рис. 13.31. Запустите все ячейки блокнота с помощью кнопки с двойной зеленой стрелкой

Это вызывает чудесную трансформацию в PyCharm. Рисунок 13.32 – хорошая отправная точка.



Рис. 13.32. Мы запустили блокнот, что вызвало множество изменений в пользовательском интерфейсе PyCharm

Во-первых, обратите внимание, что на левой боковой панели Jupyter появился новый инструмент. Мы видим, что запустили сервер Jupyter, работающий на порту 8888. Если бы вы использовали Jupyter независимо от PyCharm, это был бы обычный режим работы. Вы бы запустили сервер Jupyter из командной строки и перешли к блокноту в браузере. PyCharm заменяет этот опыт в IDE, но нам все равно нужно запустить сервер, чтобы получить результаты.

Если вы прокрутите вверх, то увидите, что разметка LaTeX была отображена в нашей ячейке Markdown, показанной на рис. 13.33.

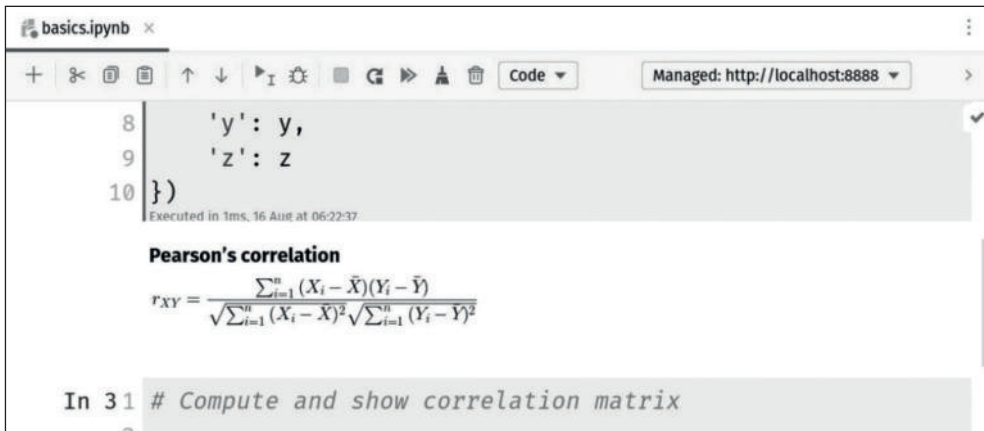


Рис. 13.33. Эта чушь теперь выглядит просто потрясающе!

Прокрутив вниз, чтобы просмотреть диаграмму рассеяния, мы увидим... Однако! Есть ошибка! Смотрите рис. 13.34.

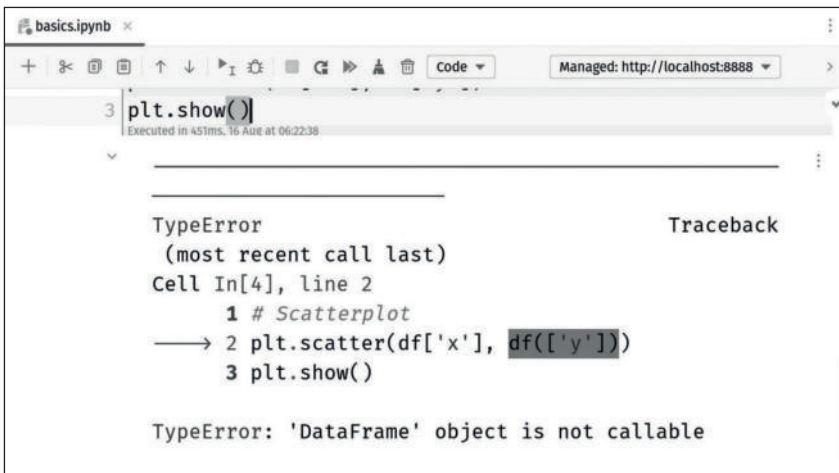


Рис. 13.34. Сообщение об ошибке показывает нам, что именно не так

Я все испортил. Я поставил круглые скобки рядом с объектом DataFrame, df. Нужно их снять, чтобы было похоже, что это соседи:

```
# Scatterplot
plt.scatter(df['x'], df['y'])
plt.show()
```

Теперь я мог бы перезапустить весь блокнот, но мне это не нужно. Проблема была в последней ячейке, поэтому я и мог просто поместить курсор в последнюю ячейку и кликнуть одну зеленую стрелку запуска (см. рис. 13.35).

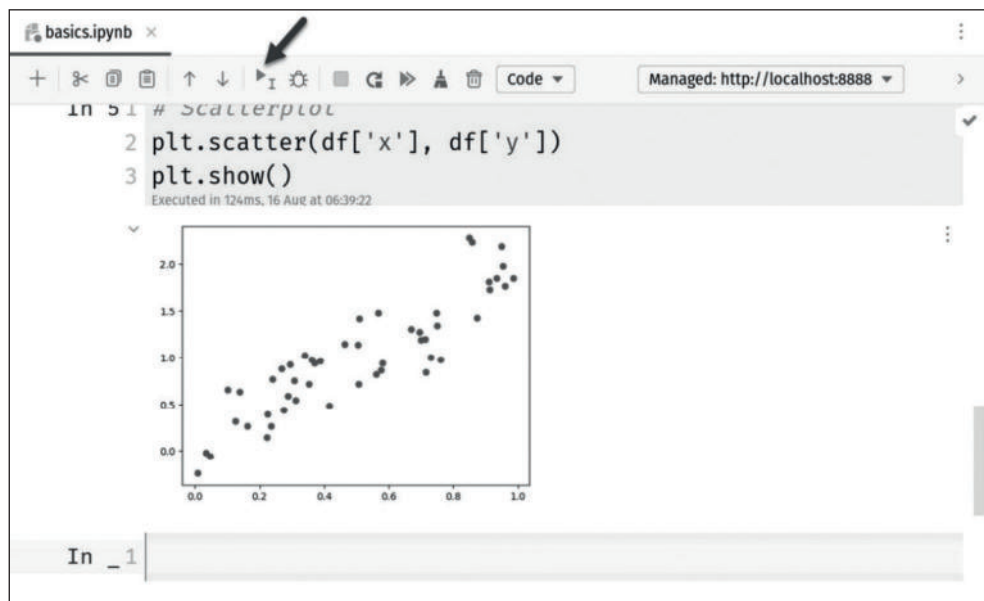


Рис. 13.35. Да, так намного лучше!

Давайте завершим наш обзор Jupyter несколькими мелочами.

Нечетности и завершения

В настоящее время у нас работает сервер Jupyter. Мгновение назад мы видели, что в левом боковом меню появился новый значок инструмента. Давайте нажмем на это и посмотрим, что происходит (см. рис. 13.36).

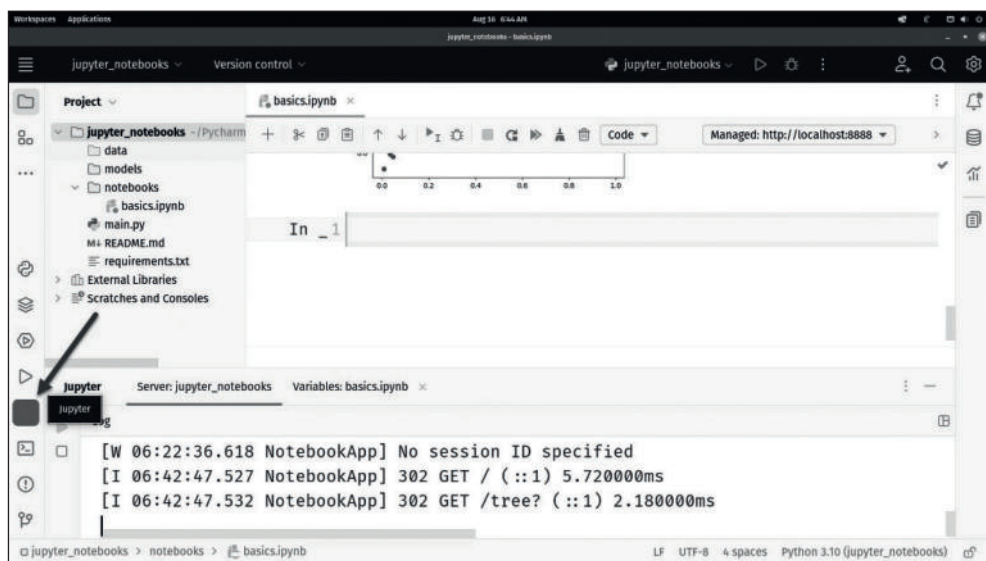


Рис. 13.36. Откроется панель Jupyter, и мы увидим выходные данные работающего сервера на вкладке Server

Прокрутив это окно до верха, вы увидите локальный URL-адрес сервера Jupyter, если хотите работать со своим блокнотом в браузере. Но зачем вам это делать, если PyCharm намного лучше?

Обратите внимание на кнопку остановки, которая позволяет остановить работающий сервер Jupyter. Рядом с вкладкой **Server** находится вкладка **Variables**, показанная на рис. 13.37.

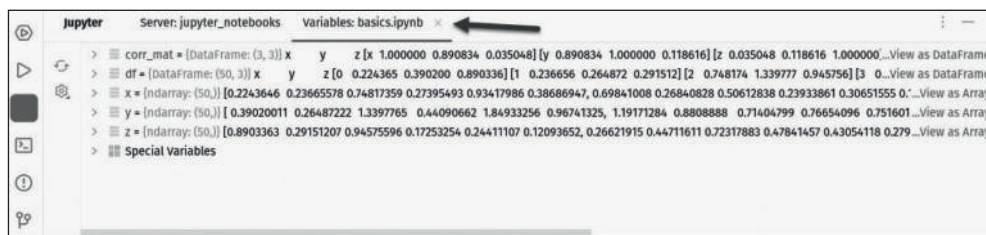


Рис. 13.37. Вкладка Variables на панели Jupyter позволяет глубоко изучить переменные в вашем блокноте

На вкладке **Variables** вы получаете проверки, которые позволяют детализировать любую переменную в блокноте. Также обратите внимание, что существуют параметры **View as Array** и **View as DataFrame**, поэтому вы также можете использовать панель SciView для просмотра и фильтрации содержимого массива.

Мы особо не говорили о панели инструментов в верхней части вкладки блокнота, которая показана на рис. 13.38.



Рис. 13.38. Элементы управления ячейками в блокноте

Имеется набор инструментов для вырезания, копирования и вставки ячеек (1). Стрелки (2) позволяют изменять порядок ячеек в блокноте, сдвигая их вверх или вниз. В (3) представлен широкий набор инструментов запуска и отладки, которые вы, несомненно, узнаете как общую тему.

Значок метлы очистит выходные данные ячейки, что удобно, если вы работаете над прогнозами заработной платы на основе повышения, о котором собираетесь попросить, когда ваш босс подходит к вам сзади. Корзина полностью удаляет ячейку. Здесь также есть раскрывающийся список с пометкой **Code**, который позволяет вам изменить тип ячейки, например с ячейки кода на ячейку разметки или наоборот.

Мы рассмотрели основные функции PyCharm в контексте блокнотов Jupyter. В целом одним из самых больших недостатков использования традиционных блокнотов Jupyter является отсутствие автодополнения кода при написании кода в отдельных ячейках кода. В частности, когда мы пишем код в блокнотах Jupyter в нашем браузере, этот процесс очень похож на написание кода в простом текстовом редакторе с ограниченной поддержкой.

Однако, работая с блокнотами Jupyter непосредственно внутри редактора PyCharm, мы увидим, что все функции поддержки написания кода, доступные для обычных скриптов Python, также доступны и здесь. Другими словами, используя PyCharm для написания блокнотов Jupyter, мы получаем лучшее от обоих миров – мощную интеллектуальную поддержку от PyCharm и итеративный стиль разработки от Jupyter.

КРАТКОЕ СОДЕРЖАНИЕ

Программист Python обычно работает над проектом науки о данных двумя способами: пишет традиционный скрипт Python или использует блокнот Jupyter, оба из которых активно поддерживаются PyCharm. В частности, панель SciView в PyCharm – это комплексный и динамичный способ просмотра, управления и проверки данных в рамках проекта науки о данных. Он предлагает нам отличный способ отображать визуализации, созданные скриптами Python, а также проверять значения в массивах Pandas DataFrames и NumPy.

С другой стороны, блокноты Jupyter – отличный инструмент для упрощения итеративной разработки на Python, позволяющий пользователям делать поэтапные шаги к анализу и извлечению информации из своих наборов дан-

ных. Блокноты Jupyter также хорошо поддерживаются PyCharm, поскольку их можно редактировать непосредственно в редакторе PyCharm. Это позволяет нам пропустить средний этап использования веб-браузера для запуска наших блокнотов Jupyter, одновременно имея возможность использовать мощные функции поддержки написания кода, которые предоставляет PyCharm.

Углубляясь в то, что помогает PyCharm в процессе просмотра и работы с данными, либо через панель SciView, либо с помощью блокнотов Jupyter, мы узнали, как использовать PyCharm для облегчения различных задач по обработке данных в Python. Благодаря этому мы вооружились достаточным количеством знаний и инструментов для реализации реальных проектов с использованием PyCharm.

В следующей главе мы объединим все полученные на данный момент знания по теме науки о данных и научных вычислений и рассмотрим процесс создания конвейера обработки данных в PyCharm.

Вопросы

1. Какие две основные функции содержит панель SciView?
2. В чем преимущество использования средства просмотра диаграмм на панели SciView, когда программа Python генерирует несколько визуализаций?
3. Какие структуры данных поддерживает окно просмотра данных на панели SciView?
4. В чем идея итеративной разработки и как блокноты Jupyter ее поддерживают?
5. Что такое Markdown и LaTeX? Почему выгодно иметь их поддержку в блокнотах Jupyter?
6. Как ячейка кода Jupyter представлена в редакторе PyCharm?
7. Каковы преимущества написания блокнотов Jupyter в редакторе PyCharm?

Создание конвейера данных в PyCharm

Термин «конвейер данных» обычно обозначает поэтапную процедуру, которая включает в себя сбор, обработку и анализ данных. Этот термин широко используется в отрасли для обозначения необходимости надежного рабочего процесса, который берет необработанные данные и преобразует их в ценную информацию. Некоторые конвейеры данных работают в огромных масштабах, например компания, занимающаяся **маркетинговыми технологиями (MarTech)**, принимает миллионы точек данных из потоков Kafka, сохраняет их в больших хранилищах данных, таких как **Hadoop** или **Clickhouse**, а затем очищает, обогащает и визуализирует эти данные. В других случаях данных меньше, но они гораздо более эффективны, как, например, в проекте, над которым мы будем работать в этой главе.

В этой главе мы изучим следующие темы:

- как работать с наборами данных и поддерживать их,
- как очистить и предварительно обработать данные,
- как визуализировать данные,
- как использовать **машинное обучение (ML)**.

В этой главе вы сможете применить все, что вы уже узнали по теме научных вычислений, к реальному проекту с PyCharm. Это служит практическим обсуждением, завершающим тему работы с научными вычислениями и проектами науки о данных.

Хочу особо отметить, что я активно использую текст, код и данные из первого издания, написанного другим автором, Куан Нгуеном. Во втором издании моей основной задачей было обновить существующий контент. Отношение Куана к этой главе было превосходным, поэтому большая часть того, что я сделал для обновления этой главы, заключалась в использовании более новой версии PyCharm, обновлении используемых библиотек до последних версий, а затем переписывании этой главы своими словами, чтобы стиль соответствует остальной части этой книги. Я бы ни за что не справился с этой задачей без оригинальной работы Куана, и я хочу снять шляпу перед оригиналом, мастером кунг-фу в области науки о данных Python.

ТЕХНИЧЕСКИЕ ТРЕБОВАНИЯ

Чтобы продолжить работу с этой главой, вам понадобится следующее:

- Anaconda – дистрибутив Python, адаптированный для рабочих нагрузок по обработке и анализу данных. Вы можете найти его вместе с инструкциями по установке для вашей ОС на <https://anaconda.com>;
- аналогично вместо обычного pip я буду использовать conda – менеджер пакетов Anaconda. Он установлен рядом с Anaconda;
- установленная и рабочая копия PyCharm. Его установка была описана в главе 2;
- пример исходного кода этой книги с GitHub. Мы рассмотрели клонирование кода в главе 2. Вы найдете код этой главы на <https://github.com/PacktPublishing/Hands-On-Application-Development-with-PyCharm---Second-Edition/tree/main/chapter-14>.

РАБОТА С НАБОРАМИ ДАННЫХ

Наборы данных являются основой любого проекта науки о данных. Имея хороший, хорошо структурированный набор данных, у нас есть возможность исследовать, выдвигать идеи и делать важные выводы из данных. Термины «хороший» и «хорошо структурированный» являются ключевыми. В реальном мире это редко бывает случайно. Я ведущий разработчик проекта, который занимается наукой о данных каждый день. Мы получаем данные диагностики, использования и производительности с различных аппаратных платформ, таких как массивы хранения, коммутаторы, узлы виртуализации (например, VMware), устройства резервного копирования и многое другое. Собираем по всему предприятию; каждое устройство в каждом центре обработки данных. Затем наше программное обеспечение преобразует эти необработанные данные в визуализации, которые предоставляют ценную информацию, позволяя организациям эффективно управлять своим ИТ-ресурсом посредством консолидации мониторинга работоспособности, отчетов об использовании и производительности, а также планирования мощности.

Я занимаюсь этим уже 10 лет, и мы всегда стремимся поддерживать новые устройства и системы. Однако наша задача – получить необходимые данные. Когда я начинал 10 лет назад, получение данных из массива хранения данных NetApp было очень трудным, поскольку его диагностические данные свалены в кучу хлама в виде неструктурированного текста. Сравните это с более современными массивами, которые сваливают данные в формате XML или JSON или, что еще лучше, имеют собственные SDK для взаимодействия с оборудованием и извлечения необходимых нам данных.

Много усилий уходит на сбор данных из различных источников и работу над преобразованием необработанных данных во что-то полезное. Иногда это легко, а иногда очень сложно. Плохо отформатированные данные могут привести к ошибочным заключениям и ложным выводам.

Замечательная предостерегающая история произошла с крупным производителем обуви. Около 20 лет назад я работал в компании, которая продавала программное обеспечение, предназначенное для управления заводским про-

изводством. Мы проконсультировались с обувной компанией и рассказали им, как моделировать данные для достижения наилучших результатов. Они проигнорировали нас и пошли другим путем. Мы сказали им, что это не сработает. Они поблагодарили нас за наш вклад. Их прогнозы оказались чудовищно ошибочными, поэтому они сделали то, что сделала бы любая крупная компания с советами директоров и акционерами, – они обвинили программное обеспечение. Наш генеральный директор ездил по бизнес-презентациям и объяснял, что к чему, но ущерб уже был нанесен. Акции нашей компании резко упали, и в том году многие люди потеряли работу, включая меня. По сей день я не ношу их обувь. Неверные данные могут стоить средств к существованию, репутации и, помимо обуви, даже жизней. В нашем распоряжении должны быть инструменты и процессы, которые помогут нам сделать все правильно.

Давайте рассмотрим несколько этапов этого процесса.

НАЧНЕМ С ВОПРОСА

Все в науке начинается с вопроса. Для наших целей мы рассмотрим два возможных сценария:

- у нас уже есть конкретный вопрос, и нам нужно собрать и проанализировать соответствующие данные, чтобы на него ответить;
- данные у нас уже есть, и в ходе их изучения возникает вопрос.

В нашем случае мы собираемся воссоздать этап анализа данных потенциально важного прорыва в области медицинской диагностики. Я представлю пример из Kaggle, взятый из работы под названием «Высокоточное обнаружение ранней стадии болезни Паркинсона с использованием некоторых характеристик движения пальцев при наборе текста», которую провел Уорвик Адамс в 2017 году. Вы найдете полный текст исследования и ссылки на наборы данных в разделе «Дальнейшее чтение» этой главы.

Примечание

Kaggle – это онлайн-сообщество по данным, созданное для специалистов по данным и инженеров машинного обучения. На сайте представлены конкурсы, наборы данных, игровые площадки и другие образовательные мероприятия, способствующие развитию науки о данных как в академических кругах, так и в промышленности. Более подробную информацию о веб-сайте можно найти на его домашней странице: <https://www.kaggle.com/>.

Болезнь Паркинсона (БП) – это заболевание, которое поражает мозг и вызывает проблемы с движением. Это прогрессирующее заболевание, что означает, что со временем ситуация ухудшается. Более 6 млн человек во всем мире страдают этим заболеванием. При БП особый тип клеток головного мозга, вырабатывающий химическое вещество под названием *дофамин*, начинает отмирать. Это приводит к множеству симптомов, включая трудности с движением и другие проблемы, не связанные с движением.

На момент написания у врачей не было определенного теста для диагностики БП, особенно на ранних стадиях, когда симптомы могут быть не очень очевидными. Это приводит к ошибкам в диагностике заболевания: до 25 % случаев диагноз ошибочно ставят врачи, не являющиеся специалистами в области БП. Некоторые люди могут страдать болезнью Паркинсона в течение многих лет, прежде чем им будет поставлен правильный диагноз.

Это приводит нас к вопросу...

Как мы можем эффективно и точно диагностировать БП, используя некоторые тестовые, метрические или диагностические данные, без специальной клинической подготовки?

Адамс предложил тест, в котором используются данные компьютерного набора текста, собранные в определенный промежуток времени. Поскольку набор текста включает в себя мелкую моторику и поскольку именно нарушении этой мелкой моторики является первым симптомом при раннем начале болезни Паркинсона, Адамс надеялся, что можно будет использовать обыденную задачу набора текста в качестве диагностического инструмента. Исследователи протестировали этот метод на 103 людях: у 32 из них была легкая форма БП, а у остальных, контрольной группы, БП не было. Компьютерный анализ их моделей набора текста позволил определить разницу между людьми с ранней стадией болезни Паркинсона и людьми без нее. Этот метод правильно идентифицировал БП с точностью 96 % при выявлении тех, у кого она была, и с точностью 97 % при правильном выявлении тех, у кого ее не было. Это говорит о том, что этот метод может быть хорош для различения этих двух групп. Давайте посмотрим, сможем ли мы сделать такой же вывод, учитывая данные их исследования.

Архивированные пользовательские данные

В исходном коде этой главы вы найдете проект по обработке и анализу данных под названием `pipeline`. Проект содержит папку данных, содержащую наши наборы данных в двух папках: `Archivedusers` и `Tappy Data`.

Данные в папке `Archived users` имеют формат текстового файла и выглядят следующим образом:

```
BirthYear: 1952
Gender: Female
Parkinsons: True
Tremors: True
DiagnosisYear: 2000
Sided: Left
UPDRS: Don't know
Impact: Severe
Levodopa: True
DA: True
MAOB: False
Other: False
```

Ради погружения давайте немного проясним это. Вот поля, которые есть в каждой записи:

- BirthYear: 1952. Этот человек родился в 1952 году;
- Gender: Female. Этот человек идентифицирует себя как женщина;
- Parkinsons: True. Человеку поставили диагноз БП;
- Tremors: True. У этого человека присутствует тремор, представляющий собой непроизвольные дрожащие движения. Тремор является частым симптомом БП;
- DiagnosisYear: 2000. В 2000 году человеку был поставлен диагноз БП;
- Sided: Left. Термин «*стороннее*» в этом контексте, вероятно, относится к той стороне тела, где симптомы более выражены. В этом случае симптомы более заметны на левой стороне тела;
- UPDRS: Don't know. **Единая шкала оценки болезни Паркинсона (UPDRS)** – это инструмент, используемый для оценки тяжести болезни Паркинсона. В этом случае неизвестно, каков конкретный балл UPDRS для этого человека;
- Impact: Severe. Влияние БП на жизнь этого человека считается тяжелым, что указывает на то, что симптомы оказывают значительное влияние на его повседневную деятельность и качество жизни;
- Levodopa: True. Леводопа – распространенное лекарство, используемое для лечения симптомов БП. Этот человек принимает Леводопу в рамках лечения;
- DA: True. **Агонисты дофамина (DA)** – это еще один тип лекарств, используемых для лечения симптомов Паркинсона. Этот человек принимает агонисты дофамина в рамках лечения;
- MAOB: False. **Ингибиторы моноаминоксидазы В (МОАВ)** – это лекарства, которые могут помочь справиться с симптомами Паркинсона за счет повышения уровня дофамина в мозге. В этом случае человек не принимает МАОВ;
- Other: False. Если бы я воссоздавал это исследование по-настоящему, я бы, скорее всего, связался с первоначальным исследователем, если бы эти данные не были объяснены непосредственно в публикации. Поскольку это не так, я предполагаю, что на наш проект это не повлияет. Вероятно, это относится к другим конкретным лекарствам или методам лечения БП, что указывает на то, что человек не проходит никакого другого специализированного лечения, кроме леводопы и DA.

Таким образом, на момент исследования этим человеком была 65-летняя женщина, у которой в 2000 году был диагностирован БП. Она испытывает тремор, особенно на левой стороне тела. Влияние болезни на ее жизнь очень серьезное. Она проходит лечение леводопой и DA для облегчения симптомов, но не использует МАОВ или какие-либо другие специализированные методы лечения. Конкретная тяжесть ее симптомов, измеренная UPDRS, в данной информации не указана.

Имена файлов в папке важны. Но публиковать **личную идентификационную информацию (PII)** неэтично. Во многих странах это явно незаконно. Итак, каждому субъекту исследования присвоен идентификационный номер, который отражен в имени файла. Этот образец данных взят из файла User_0EA27ICBLF.txt.

Тарру-данные

В методологии исследования используется приложение под названием Тарру, которое работает под управлением Windows и записывает время нажатия клавиш каждым испытуемым, а также данные о положении каждой клавиши. Если вы помните из нашего предыдущего обсуждения пользовательских данных, односторонность является важным фактором. Моторная кора – это область мозга, которая отвечает за планирование, контроль и выполнение произвольных движений. Она расположена в коре головного мозга, которая является самым внешним слоем мозга.

Моторная кора, как и большая часть остального мозга, разделена на два полушария: левое и правое. Каждое полушарие контролирует произвольные движения противоположной стороны тела. Другими словами, моторная кора левого полушария контролирует движения правой стороны тела, а правое полушарие контролирует движения левой стороны тела. Поскольку это правда, знание того, с какой стороны клавиатуры поступают данные о нажатии клавиш, потенциально имеет диагностическое значение.

Давайте откроем набор данных Тарру и посмотрим, что внутри:

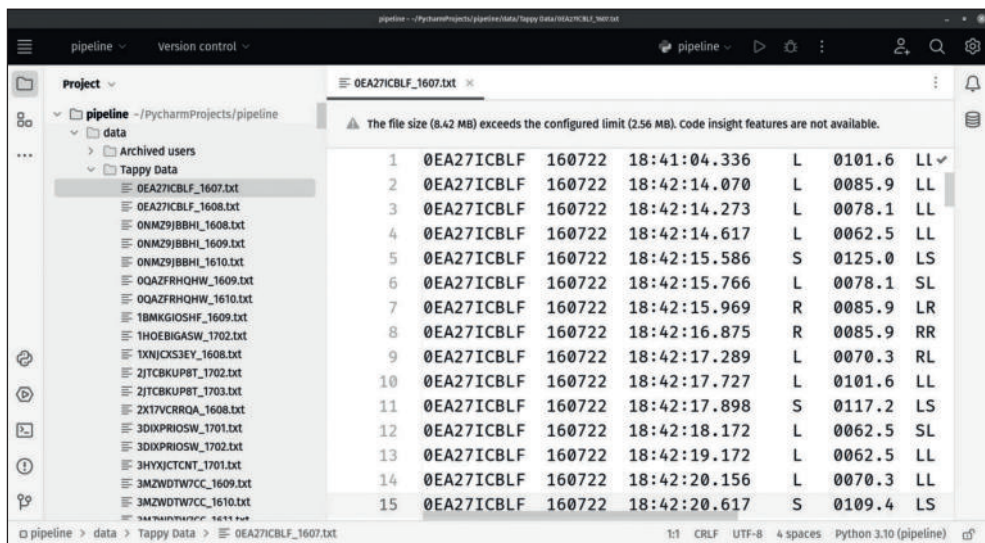


Рис. 14.1. Я открыл первый файл в папке данных Тарру и вижу, что это данные, разделенные табуляцией

Вверху я вижу предупреждение о том, что файл большой по стандартам редактора кода и что анализ кода недоступен. Это ложное уведомление, поскольку папка данных в научном проекте в PyCharm в любом случае исключается из индексации и анализа кода. Вы можете смело игнорировать это предупреждение.

Я также вижу, что файл разделен табуляцией, что будет хорошо воспроизводиться в конвейере данных. Всегда приятно видеть, что ваши данные поступают к вам в легко анализируемом формате. Это эффективно структурированные данные, которые можно импортировать в электронную таблицу или таблицу базы данных. Это не обязательно то, что мы будем делать с этими

данными, но если мы сможем выполнять такой импорт с данным файлом данных, то сможем делать с данными практически все, что угодно.

Как и прежде, имена файлов имеют большое значение. Первая часть файла, отмеченная подчеркиванием, представляет собой идентификатор субъекта из папки `Archived users`. Мы сможем связать данные об успеваемости каждого субъекта, найденные в папке `Tappy Data`, с его демографическими данными, найденными в папке `Archived users`.

Поля из файла данных Tappy следующие:

- Patient ID (идентификатор пациента),
- дата сбора данных,
- временная метка каждого нажатия клавиши,
- какая рука нажимала клавишу (*L* – левая, *R* – правая),
- время удержания (время между нажатием и отпусканием, в миллисекундах),
- переход от последнего нажатия клавиши,
- время задержки (время от нажатия предыдущей клавиши, в миллисекундах),
- время полета (время от отпускания предыдущей клавиши, в миллисекундах).

Мы установили, что имеем необработанные данные в удобоприменимом формате. Честно говоря, я бы назвал этот день хорошим. Это не совсем идеально; нам еще нужно будет немного покопаться, но это очень хорошая отправная точка.

Предупреждение о жаргоне – мунгинг

Мунгинг (munging) – это разговорный термин, используемый в компьютерном программировании и обработке данных для описания процесса манипулирования, очистки или преобразования данных из одного формата в другой. Часто это предполагает изменение структуры или содержания данных, чтобы сделать их более подходящими для определенной цели, например анализа, хранения или представления. Мунгинг может включать в себя следующие виды деятельности:

- **очистку данных:** удаление ошибок, несоответствий или нерелевантной информации из наборов данных,
- **преобразование данных:** изменение формата, структуры или представления данных в соответствии с конкретными требованиями,
- **анализ данных:** извлечение определенных фрагментов информации из более крупного набора данных,
- **агрегацию данных:** объединение нескольких наборов данных в один,
- **фильтрацию данных:** выбор или исключение данных на основе определенных критериев,
- **форматирование данных:** изменение способа представления или кодирования данных для совместимости с определенной системой или программным обеспечением.

Термин «*мунгинг*» является неформальным и происходит от смеси слов «*качать*» и «*модифицировать*». Он часто используется в контексте, когда данные необходимо подготовить или скорректировать для анализа, интеграции или какой-либо другой задачи, связанной с данными.

У нас хорошее начало проекта, но остался вопрос: можем ли мы обнаружить раннее начало БП с помощью теста на печать? У нас есть необработанные данные исследования, в котором был реализован такой тест. Мы готовы засучить рукава и приступить к делу!

СБОР ДАННЫХ

Вам повезло. Я уже нашел наши данные и представил их на ваше рассмотрение. В реальном мире нам пришлось бы выполнить обычный этап сбора данных. Хотя этой теме посвящены целые тома (большинство четырехлетних программ научного университетского образования в значительной степени посвящены этой теме), я не планирую глубоко погружаться в нее. Тем не менее я дам вам общий обзор, если вы новичок в том, что мы пытаемся изучить.

Загрузка из внешнего источника

Это относится к нашему примеру набора данных, поскольку я скачал его с Kaggle. При использовании набора данных, загруженного из интернета, мы всегда должны проверять лицензию на его авторские права. В большинстве случаев, если он находится в свободном доступе, мы можем свободно использовать и распространять его, не беспокоясь. Пример набора данных, который мы используем, является тому образцом. С другой стороны, если набор данных защищен авторским правом, мы все равно сможем использовать его, запросив разрешение у автора, точнее владельца прав на набор данных. Я обнаружил, что после обращения к ним по электронной почте и подробного объяснения того, как их наборы данных будут использоваться, владельцы наборов данных часто готовы поделиться своими данными с другими.

Ручной сбор данных и веб-скрейпинг

Если нужные нам данные доступны в интернете, но не отформатированы в таблицах или файлах CSV, в большинстве случаев необходимо собрать их и поместить в набор данных вручную. В лучшем случае мы можем написать веб-скрейпер, который сможет отправлять запросы на веб-сайты, содержащие целевые данные, и анализировать возвращаемый текст HTML. Когда вам приходится собирать данные таким образом, также важно убедиться, что вы не делаете это незаконно. Например, противозаконно использование программы для сбора данных с некоторых веб-сайтов; иногда вам может потребоваться спроектировать скрейпер так, чтобы в определенный момент выполнялось только определенное количество запросов. Примером этого может служить ситуация, когда LinkedIn в 2016 году подала иск против многих людей, которые

анонимно собирали данные. По этой причине всегда полезно узнать условия использования данных, которые вы пытаетесь собрать таким образом.

Сбор данных через третьих лиц

Студенты и исследователи, которые обнаруживают, что данные, которые они ищут в своем исследовании, невозможно собрать онлайн, часто полагаются на сторонние сервисы для сбора этих данных (например, с помощью краудсорсинга). Amazon **Mechanical Turk (MTurk)** – один из таких сервисов: вы можете ввести любой тип вопроса, чтобы провести опрос, и MTurk представит этот опрос своим пользователям. Участники получают деньги за участие в опросе, которые оплачивает владелец опроса. Этот вариант опять же особенно применим, когда вам нужен репрезентативный набор данных, которого нигде нет в интернете.

Экспорт баз данных

Скорее всего, вам приходится это делать, если вы работаете с данными своей компании или организации. К счастью, PyCharm предлагает множество полезных функций для работы с базами данных и их источниками данных. Этот процесс обсуждался в главе 11, и я настоятельно рекомендую вам ознакомиться с ним, если вы еще этого не сделали.

КОНТРОЛЬ ВЕРСИЙ НАБОРОВ ДАННЫХ

Поскольку мы немного отвлеклись, чтобы обсудить сбор данных, я надеюсь, что вы будете снисходительны ко мне, пока мы будем говорить об использовании данных в системе контроля версий, такой как Git. Чуть раньше мы открыли файл данных, и PyCharm сразу пожаловался на размер файла. По современным меркам файл размером 8 Мб не так уж и велик. Однако учтите, что большинство файлов кода, смысл существования PyCharm, в среднем имеют размер менее 100 Кб. Если ваши файлы очень большие, это плохой запах, исходящий от вашего кода, и следует выяснить, что вы делаете неправильно.

Здесь мы представляем PyCharm файл, который примерно на 8000 % больше, чем тот, к которому он привык. Git также в основном используется для работы с небольшими файлами, выходящими из IDE. Я говорю об этом, потому что в сообществе специалистов науки о данных и научных вычислениях наблюдается своего рода кризис воспроизводимости. Это когда одна команда специалистов по данным может извлечь конкретную информацию из набора данных, а другие не могут, даже используя те же самые методы. Во многих случаях это связано с тем, что данные, используемые этими разными командами, несовместимы друг с другом. Некоторые могут использовать один и тот же, но устаревший набор данных, тогда как другие наборы данных могут быть собраны из другого источника.

Контроль версий наборов данных – важная тема, которую следует учитывать. Git обычно имеет жесткий предел в 100 Мб для любого файла, и по опыту я могу вам сказать, что и на GitHub существует верхний предел общего размера ваших проектов. Те же ограничения существуют и в других системах контроля версий.

Раньше я преподавал разработку игр с помощью инструмента под названием Unity 3D, и мы всегда боролись с этими ограничениями, поскольку видеоигры обычно содержат в проектах очень большие ресурсы, которые не обязательно представляют собой код, но которые могут выиграть от контроля версий.

Использование поддержки больших файлов Git

Поскольку проблема является повсеместной, Git (и др.) добавили возможность отслеживать более крупные ресурсы через **поддержку больших файлов Git (Git LFS)**. Когда мы добавляем файл с помощью Git LFS, система заменит этот файл указателем, который просто ссылается на него. Когда файл помещается под контроль версий, Git будет иметь ссылку только на реальный файл, который теперь хранится во внешней файловой системе, возможно, на другом сервере. Git LFS позволяет нам применять контроль версий к большим файлам (в данном случае наборам данных) с помощью Git, без фактического хранения файлов в Git.

Эта функция обычно устанавливается вместе с современными установщиками Git. На рис. 14.2 показана установка Git для Windows, где LFS является частью установки по умолчанию:

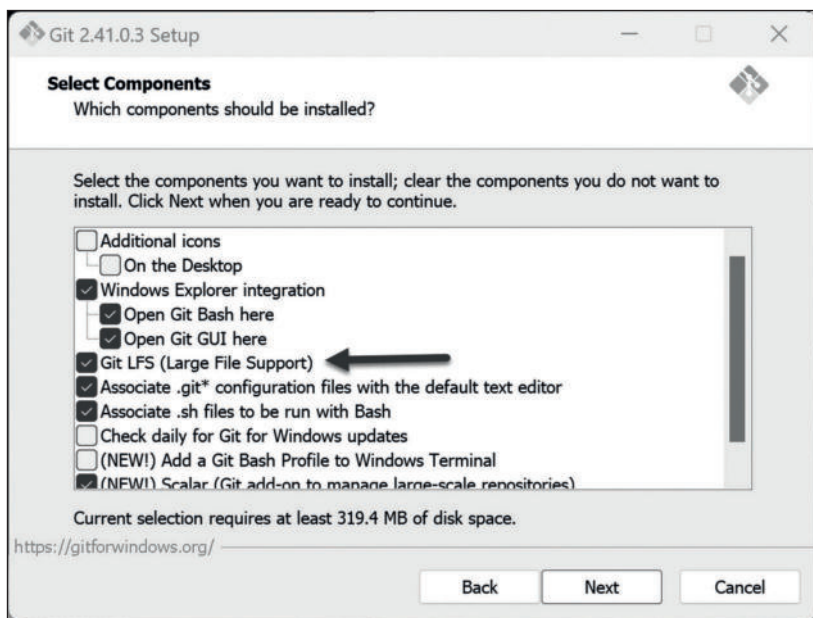


Рис. 14.2. LFS в Windows устанавливается по умолчанию

Вы можете проверить установку независимо от того, какую ОС вы используете, с помощью командной строки:

```
git lfs version
```

Мой результат выполнения этой команды в GitBash в Windows 11 показан на рис. 14.3:

```
AzureAD+BruceVanHorn@wintermute MINGW64 ~
$ git lfs version
git-lfs/3.3.0 (GitHub; windows amd64; go 1.19.3; git 77deabdf)
AzureAD+BruceVanHorn@wintermute MINGW64 ~
$
```

Рис. 14.3. Если LFS установлен, он должен сообщить вам номер версии

Единственная причина, по которой у меня стоит Windows (помимо Ghost Recon и Steam в целом), – это возможность использовать Microsoft Word для написания этой книги. Это была не моя идея. Я собирался написать все это в сыром LaTeX, используя vi. Не vim. Не neovim. Оригинальная gangsta vi, которую я, естественно, буду компилировать из исходников. Моя редакторша сказала – нет. Она такая супервежливая! Если бы мы поменялись ролями, кто знает, что бы сказали? В любом случае остальная часть моей реальной работы выполняется на **Pop_OS**, которая является вариантом Ubuntu Linux. Когда я отправляю команду в эту среду, я получаю менее гостеприимный ответ, как показано на рис. 14.4:

```
Terminal Local x + v
(pipeline) brucevanhorn@pop-os:~/PycharmProjects/pipeline$ git lfs version
git: 'lfs' is not a git command. See 'git --help'.

The most similar command is
log
(pipeline) brucevanhorn@pop-os:~/PycharmProjects/pipeline$
```

Рис. 14.4. Мой установщик недостаточно современен, чтобы иметь предустановленную LFS

У меня его нет! Мне нужно установить его с помощью этих команд:

```
sudo apt update
sudo apt install git-lfs
```

После этого я могу проверить еще раз:

```
Terminal Local x + v
Setting up git-lfs (3.0.2-1ubuntu0.2) ...#####]
Progress: [ 60%] [#####]
Processing triggers for man-db (2.10.2-1) ...#####]
(pipeline) brucevanhorn@pop-os:~/PycharmProjects/pipeline$ git lfs version
git-lfs/3.0.2 (GitHub; linux amd64; go 1.18.1)
(pipeline) brucevanhorn@pop-os:~/PycharmProjects/pipeline$
```

Рис. 14.5. Успех! Если вы используете какой-либо другой дистрибутив Linux, проверьте свою систему управления пакетами на наличие пакета git-lfs, если он отсутствует в вашей установке

Установщик **apt** уникален для Ubuntu и других вариантов Linux Debian. Если вы используете что-то вроде CentOS, Arch, SUSE или Photon (шучу), проверьте свой менеджер пакетов и найдите пакет `git-lfs`, специфичный для вашего дистрибутива Linux.

Использование Git LFS

Мы забегаем немного вперед. Если вы собираетесь продолжить это небольшое второстепенное упражнение, было бы лучше, если бы вы создали новую папку где-нибудь за пределами репозитория кода этой книги. Предположим, у вас есть что-то вроде этого в терминале вашей ОС:

```
cd ~/
mkdir git-lfs-test
cd git-lfs-test
git init
```

Эта серия команд будет работать в любой популярной ОС (Windows, macOS или Linux). Если вы используете Windows, эту серию команд можно запустить в PowerShell при условии, что у вас установлен клиент Git для Windows. Установщик доступен по адресу <https://git-scm.com/downloads>.

Первая команда приведет вас в папку `home`. Второй создает новую папку с именем `git-lfs-test`. Затем мы меняем каталог на только что созданную папку `git-lfs-test` и инициализируем новый репозиторий. Теперь мы готовы настроить поддержку Git LFS.

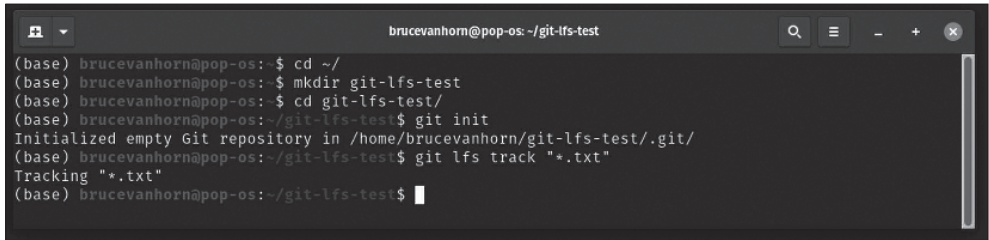
Не забывайте, что файлы глав уже находятся в репозитории Git

Если вы следите за исходным кодом этой главы, не забывайте, что файлы уже находятся в репозитории Git. Создание второго репозитория внутри существующего не сработает. Если вы хотите попрактиковаться, создайте отдельную папку вне репозитория этой книги и скопируйте файлы проекта в свою папку. При копировании нужно избежать копирования папки `.git` в вашу цель.

В нашем проекте мы собираемся использовать Git LFS для отслеживания файлов определенного расширения, в частности текстовых файлов с расширением `.txt`. Поскольку эти файлы по своей природе представляют собой обычный текст, можете проявить творческий подход к расширению, не влияя на то, как они используются, но мы будем придерживаться только `.txt`. Я запущу эту команду в окне терминала:

```
git lfs track "*.txt"
```

Вы можете увидеть мой тестовый запуск на рис. 14.6:



```

brucevanhorn@pop-os: ~/git-lfs-test
(base) brucevanhorn@pop-os: ~$ cd ~/
(base) brucevanhorn@pop-os: ~$ mkdir git-lfs-test
(base) brucevanhorn@pop-os: ~$ cd git-lfs-test/
(base) brucevanhorn@pop-os: ~/git-lfs-test$ git init
Initialized empty Git repository in /home/brucevanhorn/git-lfs-test/.git/
(base) brucevanhorn@pop-os: ~/git-lfs-test$ git lfs track "*.txt"
Tracking "*.txt"
(base) brucevanhorn@pop-os: ~/git-lfs-test$

```

Рис. 14.6. Git LFS теперь отслеживает все файлы с расширением .txt

Чтобы завершить тест LFS, я скопирую изученный ранее файл 0EA27ICBLF_1607.txt из папки `Tarpu Data` в папку `git-lfs-test`, которую мы используем для эксперимента. Для ясности: на рис. 14.7 показана моя папка. Мы не делаем этого ни в одной подпапке репозитория кода этой книги, поскольку создание репозитория внутри другого репозитория категорически запрещено:

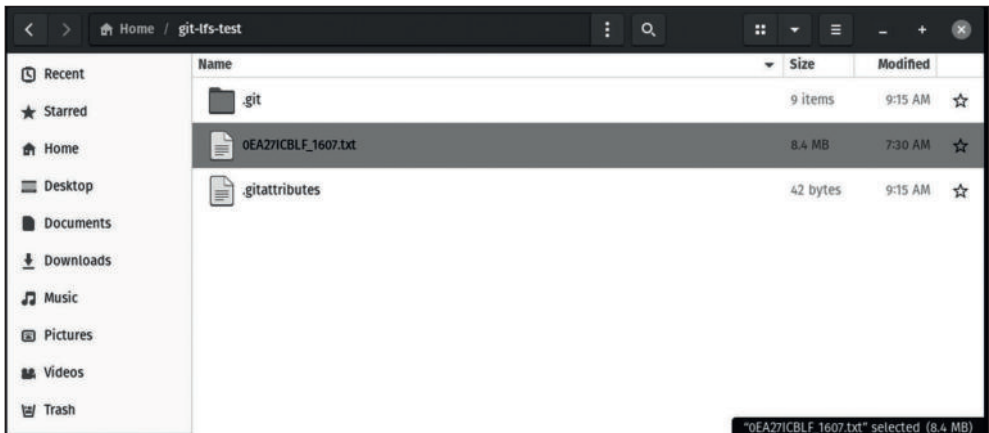


Рис. 14.7. Я скопировал 0EA27ICBLF_1607.txt в папку git-lfs-test

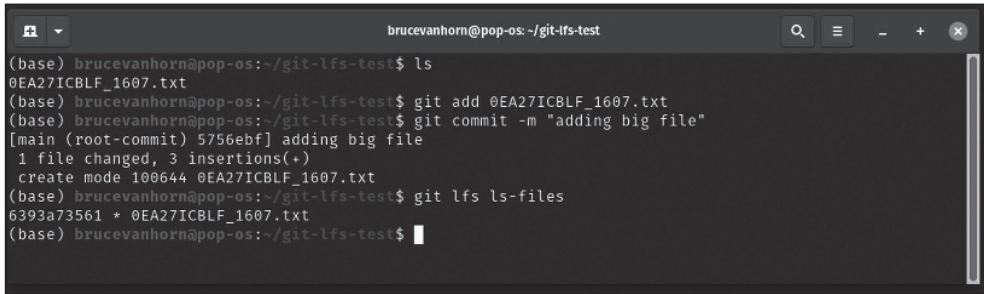
Теперь давайте добавим только что скопированный текстовый файл в репозиторий:

```

git add 0EA27ICBLF_1607.txt
git commit -m "adding big file"
git lfs ls-files

```

Мы подробно рассмотрели первые две команды Git в главе 5. Последняя команда выведет список всех файлов, отслеживаемых LFS в этом репозитории. Вы можете увидеть мои результаты на рис. 14.8:



```

brucevanhorn@pop-os: ~/git-lfs-test
(base) brucevanhorn@pop-os:~/git-lfs-test$ ls
0EA27ICBLF_1607.txt
(base) brucevanhorn@pop-os:~/git-lfs-test$ git add 0EA27ICBLF_1607.txt
(base) brucevanhorn@pop-os:~/git-lfs-test$ git commit -m "adding big file"
[main (root-commit) 5756ebf] adding big file
1 file changed, 3 insertions(+)
create mode 100644 0EA27ICBLF_1607.txt
(base) brucevanhorn@pop-os:~/git-lfs-test$ git lfs ls-files
6393a73561 * 0EA27ICBLF_1607.txt
(base) brucevanhorn@pop-os:~/git-lfs-test$

```

Рис. 14.8. Я вижу, что мой текстовый файл отслеживается LFS

Теперь вы понимаете, как использовать Git LFS для отслеживания больших файлов. Если бы это был настоящий репозиторий, который мы хотели бы сохранить, нам нужно было бы сделать еще одно. Когда мы дали команду Git отслеживать наши текстовые файлы, от нашего имени был создан специальный файл с именем `.gitattributes`. Мы должны добавить и зафиксировать этот файл:

```

Git add .gitattributes
Git commit -m "Added .gitattributes to repo"

```

Все готово! Давайте перейдем к следующему формальному шагу в процессе анализа данных, который влечет за собой очистку и предварительную обработку данных.

ОЧИСТКА И ПРЕДВАРИТЕЛЬНАЯ ОБРАБОТКА ДАННЫХ

Как я уже говорил ранее, нам очень повезло. Некоторые данные, с которыми работает моя команда, бывают очень грязными. Когда мы используем в отношении данных такие термины, как «*грязный*», «*мусорный*» и «*чистый*», мы говорим о формате данных, а также о пригодности данных для обработки. Данные можно использовать только тогда, когда они в формате, с которым мы можем работать. Структурированные данные – это то, что мы всегда предпочитаем.

Структурированные данные – это данные, которые разделены на идентифицируемые поля. Мы видели текст, разделенный запятыми и табуляцией. Другие примеры структурированных данных включают такие форматы, как XML, JSON, Parquet и HDF5. Первые два, XML и JSON, очень распространены и имеют то преимущество, что представляют собой текстовые форматы. Последние два, Parquet и HDF5, являются двоичными файлами и предназначены для хранения больших наборов данных, чем было бы удобно при работе с текстом. Как мы видели, большинство инструментов, включая PyCharm, дают сбой при попытке прочитать очень большие текстовые файлы. Вам нужны инструменты, специализированные для работы с большими файлами, если вы хотите просматривать или редактировать их на месте.

Когда я говорю о «грязных» и «чистых» данных, я имею в виду такие вещи, как отсутствие или неверные данные полей. Вспомните наш предыдущий образец данных:

```

BirthYear: 1952
Gender: Female
Parkinsons: True
Tremors: True
DiagnosisYear: 2000
Sided: Left
UPDRS: Don't know
Impact: Severe
Levadopa: True
DA: True
MAOB: False
Other: False

```

Поле UPDRS помечено как неизвестное. Это не лучший случай. Если поле включено, я бы хотел увидеть там значение. В этом случае нет возможности заполнить его, но в идеальном мире это могло бы стать кандидатом на очистку данных.

Пример токсичных данных с участием `ninja`

Самый интересный пример грязных – или, в данном случае, токсичных – данных, который я когда-либо встречал, пришел из корпоративного мира, а не из эксперимента науки о данных. Моя компания консультировала крупную авиационную компанию, которая также является подрядчиком Министерства обороны США. Я не буду называть здесь настоящие имена, потому что вообще не люблю, когда правительственные ниндзя ломаются в мою дверь в два часа ночи, или еще хуже, меня вызывают на налоговую проверку за то, что я здесь написал. Итак, мы оставим это более или менее теоретическим.

Авиационная компания вела дела со многими поставщиками, и когда вы ведете бизнес с крупными поставщиками, нередко можно увидеть скидки, применяемые к тому, что вы покупаете, в зависимости от объема. Если вы или я пойдем в Hammers R Us и купим молоток, мы можем заплатить за него 12,95 долл. Но если авиационная компания купит 5000 молотков по нескольким заказам за один квартал, она может получить скидку до 60 %. Задача авиационной компании – отслеживать, что и у кого они покупают, чтобы они могли заработать на любых сделках по оптовым закупкам, которые их компания заключила со своими поставщиками.

Когда приходит время запускать отчеты о скидках, бухгалтерский аналитик может запросить базу данных, заполненную данными, введенными сотнями или даже тысячами людей, работающих на местах от имени авиационной компании. Поскольку эти оперативники – люди, их способность вводить чистые, стандартизированные данные в плохо спроектированную систему без какой-либо проверки практически равна нулю. В этом случае программное обеспечение, используемое для ввода заказов, позволяло пользователям вводить название компании в текстовое поле, которое никогда не проверялось по какому-либо списку утвержденных поставщиков.

Один парень совершает покупку у продавца, указанного как «Hammers R Us». Другой вводит его как «HRUS» (естественно, это биржевой символ), а другой – как «H.R.U.S.». Кто-то пишет с ошибкой «Hammers Are Us», а третий – «Hammers-R-Us». Теперь у нас есть пять разных упоминаний об одной и той же компании,

что снижает нашу способность определить, какую скидку мы можем запросить. Если есть 5 вариантов написания и объемы закупок одинаковы для 5 заказов, каждый заказ будет только на 1000 молотков, и наша скидка составит только 20 % вместо 60 %. Наша проблема с токсичными данными обходится компании серьезными деньгами!

Авиационная компания наняла мою компанию для **очистки данных**. Нашей задачей было очистить все данные и стандартизировать все ссылки на Hammers R Us. Для нас проект оказался успешным, потому что все, что нам нужно было сделать, – это снять с клиента за нашу работу на несколько долларов меньше, чем он теряет на закупках, а разница была весьма существенной. Затем мы помогли им исправить программное обеспечение, чтобы сделать невозможным последующий ввод токсичных данных. Это была победа для всех! Я даже получил бесплатный молоток от Hammers R Us, по крайней мере, в моей версии истории, которая предполагает, что ко мне не нагрянули аудиторы или ниндзя.

Исследовательский анализ в PyCharm

Хотя очистка данных в проекте науки о данных обычно не приносит финансовой выгоды, это очень необходимый шаг. Когда вы впервые начнете изучать свои данные, вы часто будете слышать, как этот процесс называется **исследовательским анализом данных**, когда мы одновременно изучаем и анализируем данные. Однако мы подводим итоги, чтобы увидеть, что можно сделать с нашими данными. Было бы очень сложно выполнить табулирование, например вычислить суммы, средние значения и стандартные отклонения, не убедившись предварительно, что все необходимые данные имеются и находятся в удобном числовом формате. Мы также можем искать выбросы. Возможно, ордер на молоток был введен неправильно, и у нас есть заказ на миллион молотков, который был отменен посредством отдельной транзакции. Подобные выбросы, скорее всего, придется удалить, прежде чем мы начнем наш анализ всерьез.

Что касается наших данных, меня беспокоят несколько вещей.

- В исследовании говорится, что было обследовано 103 субъекта; однако в папке `Archived users` находится 277 пользовательских файлов. Я подозреваю, что не у каждого пользователя есть соответствующие собранные данные. Нам понадобится способ проверить, что у каждого пользователя в папке `Archived users` есть связанный набор данных в папке `Tappy Data`.
- Наши необработанные данные являются чисто текстовыми, а это означает, что, когда мы импортируем их в Python путем чтения файлов, данные будут выражены в виде строк. Это не идеально для анализа данных. Я бы хотел, чтобы числа были преобразованы в числовые типы, даты в типы дат, логические значения в логические значения и т. д.
- Столбец `Impact` должен быть полностью стандартизирован для учета отсутствующих значений в данных. Естественно, это относится к любому другому столбцу, в котором я вижу или подозреваю, что данные могут содержать пропущенные значения.
- Мы можем преобразовать некоторые поля в наборах пользовательских данных в двоичный формат, чтобы упростить анализ. Конкретные примеры включают `Parkinsons`, `Tremors`, `Levodopa`, `DA`, `MAOB` и `Other`.

- Мы можем использовать процесс, называемый горячим кодированием, чтобы упростить обработку полей с метками Sided, UPDRS и Impact. Я подробно расскажу о горячем кодировании, когда мы будем готовы выполнить этот процесс.

Это только то, что я вижу на первый взгляд. Могут появиться и другие возможности для очистки, когда мы начнем действовать.

Чтение данных из текстовых файлов

Давайте посмотрим, что нужно делать с предварительной обработкой наших данных. Если вы откроете файл `data_clean.py`, вы увидите наш скрипт очистки, который использует режим ячеек, описанный в главе 12. Наша первая ячейка обрабатывает импорт:

```
import pandas as pd
import numpy as np
import os
import gc
```

Если вы следуете коду этой главы, не забудьте создать виртуальную среду, используя файл `requirements.txt`. Здесь мы импортируем нескольких старых друзей. `numpy` и `pandas` – стандартные библиотеки анализа. Пакет `os` понадобится для работы с файловыми каталогами, а пакет `gc` позволяет нам управлять процессом **сборки мусора (GC)**. Если вы никогда об этом раньше не слышали, то это потому, что большинство языков программирования, включая Python, обрабатывают GC автоматически. Один из распространенных случаев GC происходит, когда переменная, которой будет выделена память для хранения ее значения, выходит из области определения и больше не нужна. В языке программирования C нужно будет самостоятельно выделить эту память, прежде чем вы сможете использовать переменную. Когда закончите работу с переменной, нужно будет освободить эту память «вручную». Если бы вы этого не сделали, вы бы использовали больше памяти, чем нужно, и именно из-за этого вас не пригласят на вечеринку с пиццей в честь «Дня Пи».

Большинство современных языков автоматически обрабатывают эту аллокацию и деаллокацию в процессе, называемом GC. Однако бывают случаи, особенно когда вы загружаете и манипулируете большими объемами данных, когда имеет смысл играть более активную роль при удалении мусора, что освобождает память для дальнейших подвигов.

Закончив с импортом, давайте прочитаем некоторые данные из следующей ячейки:

```
### Read in data

user_file_list = os.listdir('data/Archived users/')
user_set_v1 = set(map(lambda x: x[5: 15], user_file_list)) # [5: 15]
to return just the user IDs
```

Метод `os.listdir` берет нашу папку `data/Archived users/` и выдает нам повторяемый список файлов из этой папки. Это важно, поскольку нам нужен список идентификаторов каждого пользователя, который содержится в имени файла.

Мы создаем переменную с именем `user_set_v1` и создаем экземпляр набора. В Python `set` – это встроенный тип данных, представляющий неупорядоченную коллекцию уникальных элементов. Это означает, что набор не может содержать повторяющиеся значения, а порядок хранения элементов не обязательно будет таким же, как порядок их добавления.

Мы заполняем этот `set` данными с помощью оператора `map`, который перебирает наш список файлов в папке `Archived users`. Для каждой итерации карты мы используем лямбда-функцию для извлечения части каждого имени файла в `user_file_list`. В частности, она принимает подстроку с 5-го по 15-й символ каждого имени файла. Это предназначено для извлечения идентификаторов пользователей из имен файлов. Далее нам нужно будет сделать примерно то же самое с файлами `Tappy Data`:

```
tappy_file_list = os.listdir('data/Tappy Data/')
user_set_v2 = set(map(lambda x: x[: 10], tappy_file_list)) # [: 10] to return
just the user IDs
```

Теперь у нас есть два набора: один из пользовательских файлов и один из файлов данных Тарру. Нам нужно найти пересечение между множествами.

В **теории множеств** термин «пересечение» относится к операции, которая из двух множеств создает новое множество, содержащего только те элементы, которые являются общими для обоих исходных множеств. Пересечение двух множеств, часто обозначаемое символом \cap , представляет собой перекрытие или общие элементы между наборами.

Математически, если у вас есть два множества (или набора), A и B , пересечение A и B представляет собой новый набор, содержащий элементы, которые одновременно входят как в множество A , так и в множество B .

Я знаю, что все вы, фанаты математики, любите свои символы, и я также знаю, что ваш мозг настроен на поиск закономерностей, а не на дословное чтение, поэтому я вам помогу. Символически это представляется как $A \cap B = \{ x \mid x \in A \text{ и } x \in B \}$.

В контексте программирования на Python эту математическую операцию представляет метод `intersection()`. Из двух наборов он возвращает новый набор, содержащий только элементы, существующие в обоих наборах.

Например, предположим, что у вас есть следующие два набора данных:

- `set A = {1,2,3,4}`,
- `set B = {3,4,5,6}`.

Пересечение A и B будет $A \cap B = \{ 3, 4 \}$, поскольку 3 и 4 находятся в обоих множествах. В нашем случае важно получить пересечение, поскольку в тексте исследования указано, что в `Archived users` перечислено 227 субъектов. Я мог бы составить список, а затем просмотреть и визуально сравнить содержимое папки `Tappy Data`, чтобы убедиться, что все учтены, но это было бы скучно, отнимало много времени и чревато ошибками. Я просто попрошу Python сделать это за меня:

```
user_set = user_set_v1.intersection(user_set_v2)
```

Разве вам не нравятся строки Python? Конечно, были какие-то настройки, но потом мы повозились с методом `intersection`, и у нас появился новый набор, в котором все просто замечательно! Давайте посмотрим, что мы получили, распечатав длину:

```
print(len(user_set))
```

Я собираюсь запустить первые две ячейки, используя зеленые стрелки, указанные на рис. 14.9:



Рис. 14.9. Я запускаю первые две ячейки, которые мы рассмотрели, используя зеленые стрелки вверх каждой ячейки

Результат прогона показан на рис. 14.10.

У нас есть 217 пользователей с коррелирующими данными между двумя наборами, поэтому нам удалось исключить 60 пользовательских файлов, которые мы не собираемся использовать. Число не соответствует 103 испытуемым, указанным в тесте, но это нормально – время еще есть, и мы можем исключить еще больше позже. Даже если мы этого не сделаем, могут быть другие причины для последующего удаления правильно сопоставленных данных. Наш новый набор можно использовать для перебора данных в любой папке данных, поскольку имена файлов в обеих используют идентификатор как основную часть имени файла. Это будет очень полезно на следующем этапе процесса подготовки данных.

На рис. 14.10 я нажимаю ссылку просмотра, чтобы просмотреть свой список на панели **SciView**. Это не особенно интересно, поскольку это всего лишь список идентификаторов, но возможность легко делать проверку во время работы без выполнения дополнительных распечаток очень полезна.

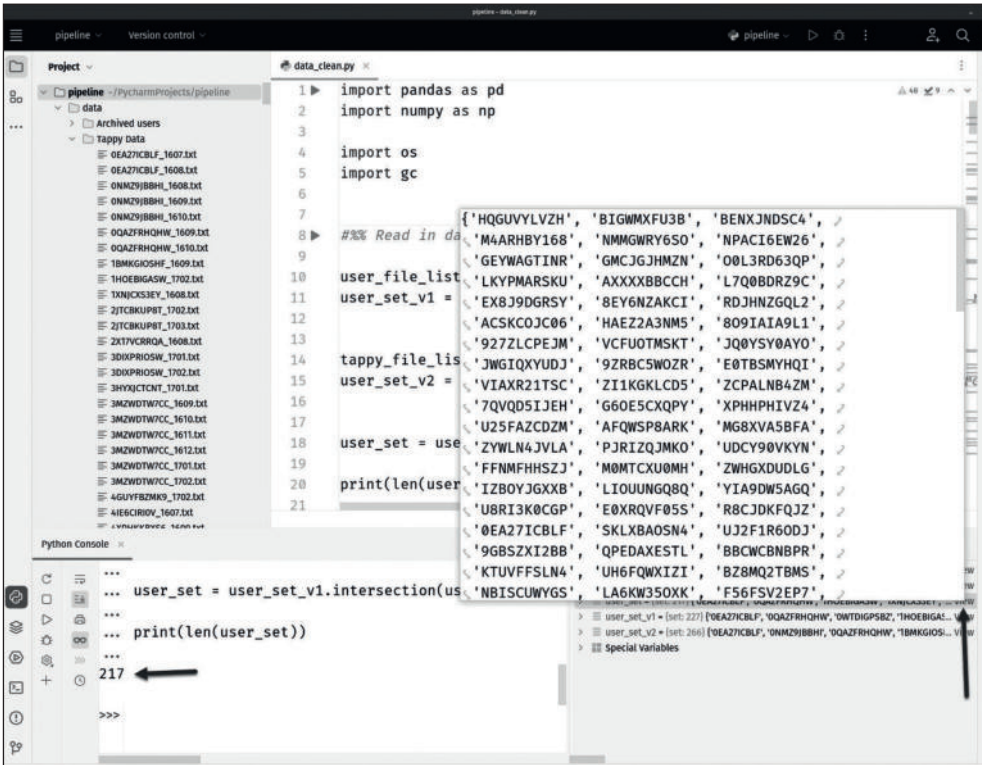


Рис. 14.10. После первых шагов по очистке данных у меня есть относительно чистый список пользователей

Перемещение наших данных в pandas DataFrame

Наша следующая ячейка содержит код, предназначенный для получения загруженного набора данных и переноса этих данных в pandas DataFrame. Библиотека pandas позволяет легко анализировать табличные данные и даже предоставляет множество очень полезных методов для загрузки данных непосредственно в DataFrame, который представляет собой табличную структуру внутри pandas. Объект DataFrame во многом похож на электронную таблицу в памяти без редактора. Вы можете выполнять все виды вычислений с минимальными усилиями.

Давайте рассмотрим код из следующей ячейки:

```
### Format into a Pandas dataframe
```

Не забывайте, что `###` – это специальный комментарий форматирования в PyCharm. Это не часть Python. Мы рассмотрели это еще в главе 13. Эти символы используются для разделения ячеек в нашем коде, что позволяет нам использовать один скрипт, но работать поэтапно от одной ячейки к другой. В конце концов, это все еще комментарий, поэтому нам следует включить некоторую документацию, объясняющую, что происходит в ячейке.

Далее мы создадим функцию, которая считывает данные из файлов в папке Archived users:

```
def read_user_file(file_name):
    f = open('data/Archived users/' + file_name)
    data = [line.split(': ')[1][-1] for line in f.readlines()]
    f.close()

    return data
```

Функция просто принимает имя файла в качестве аргумента и открывает файл. Затем она читает файл построчно. Для каждой строки мы используем функцию split, чтобы разбить ее на фрагменты в виде списка. Это позволяет брать только те части, которые нам нужны. Как вы помните, несколько строк данных для этих файлов выглядят следующим образом:

```
BirthYear: 1952
Gender: Female
Parkinsons: True
```

Имя поля и данные разделяются двоеточием и пробелом (:). Мы используем его в качестве разделителя, поэтому, если вы разделите "BirthYear: 1952".split(': '), вы получите список: ["BirthYear", "1952"]. Сейчас нас не волнует имя поля, нас волнует значение. Чтобы получить его, мы берем [1], это дает нам «1952», что является значением, но в конце каждой строки есть символ новой строки, и он был включен в наше разделение. Последнее, что мы делаем, прежде чем перейти к следующей итерации, – это удаляем символ новой строки с помощью оператора разделения Python [:-1], который фактически говорит «перейти к концу строки», о чем свидетельствует тот факт, что число стоит после двоеточия, «и отрезаем один символ с конца», что обозначается отрицательным числом. Вместо использования цикла мы использовали обработку списка, которая является альтернативным способом его перебора. Как правило, этот способ более производителен, чем обычный цикл for. Результатом обработки списка является новый список, содержащий только те данные, которые нам нужны.

Следующие несколько строк настраивают нас на заполнение DataFrame pandas. Сначала мы получаем список файлов в папке Archived users:

```
files = os.listdir('data/Archived users/')
```

Далее создаем список полей. Мы уже настроили функцию для извлечения данных из файлов без имени поля. Одновременное копирование имен может потребовать много времени, поскольку это одно и то же снова и снова; это просто более эффективно:

```
columns = [
    'BirthYear', 'Gender', 'Parkinsons', 'Tremors', 'DiagnosisYear', 'Sided',
    'UPDRS', 'Impact', 'Levodopa', 'DA', 'MAOB', 'Other'
]
```

Затем мы создаем пустой DataFrame в качестве отправной точки, используя список `columns`. Думайте об этом как о создании новой электронной таблицы и заполнении первой строки таблицы именами столбцов:

```
user_df = pd.DataFrame(columns=columns) # empty Data Frame for now
```

Далее давайте пройдемся по `user_set`, который мы создали в предыдущей ячейке. Помните, что это список идентификаторов пользователей, данные которых находятся в папке `Tappy Data`. Напомним, что структура имени этого файла представляет собой слово `User`, за ним следует знак подчеркивания, за которым следует идентификатор пользователя и к которому добавляется расширение файла `.txt`:

```
for user_id in user_set:
    temp_file_name = 'User_' + user_id + '.txt'
```

Далее убеждаемся, что файл существует. Так и должно быть, поскольку ранее мы выполнили операцию `set`, но стоит проверить. Если файла нет, наш набор анализа выйдет из строя. Это не имеет большого значения для нескольких сотен файлов, но может быть душераздирающим, если вы просматриваете десятки тысяч. Предполагая, что файл существует, мы считываем его в переменную `temp_data`, используя созданную ранее функцию. Помните, что эта функция возвращает список значений данных, которые выглядят так же, как ячейки в строке электронной таблицы. Затем вставляем эти данные в DataFrame, используя ID пользователя в качестве индекса строки:

```
if temp_file_name in files:
    temp_data = read_user_file(temp_file_name)
    user_df.loc[user_id] = temp_data
```

Естественно, нужно проверить, но нам не нужна каждая строка – мы просто хотим, чтобы первые несколько были отформатированы так, как мы ожидаем:

```
print(user_df.head())
```

Когда я запускаю эту ячейку, получаю следующий результат:

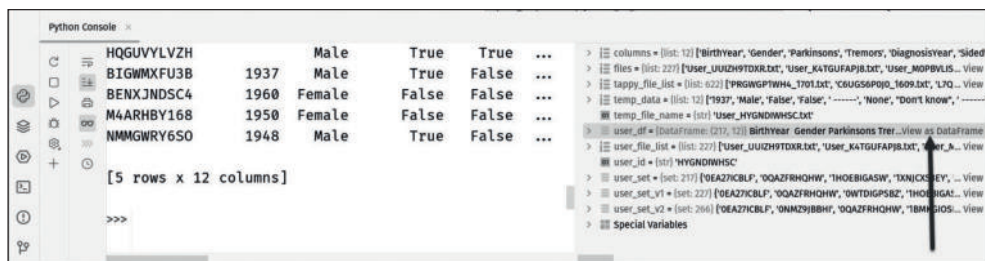


Рис. 14.11. Запуск нашей последней ячейки показывает, что у нас есть заполненный DataFrame pandas

Помните, что вы можете просмотреть DataFrame в **SciView**, нажав кнопку **View as DataFrame**, обозначенную стрелкой на рис. 14.11. Мой результат показан на рис. 14.12:

	BirthYear	Gender	Parkinsons	Tremors	DiagnosisYear	Sided	UPDRS	
HQG...		Male	True	True	2014	None	Don't know	Me
BIGW...	1937	Male	True	False	2016	Right	Don't know	Me
BENX...	1960	Female	False	False	-----	None	Don't know	---
M4A...	1950	Female	False	False	-----	Left	Don't know	---
NMM...	1948	Male	True	False	2016	Right	Don't know	Mil
NPAC...	1965	Female	True	True	2016	Left	Don't know	Me
GEY...	1929	Male	False	False	-----	None	Don't know	---
GMCJ...	1949	Female	True	False	1995	Right	Don't know	Me
OOL3...	1944	Female	True	True	2012	None	Don't know	Mil
LKYP...	1946	Female	True	True	2017	Right	Don't know	Mil
AXXX...	1949	Female	False	False	-----	None	Don't know	---
L7Q0...	1963	Male	True	False	2004	Right	Don't know	Me
EX8J...	1943	Male	True	False	2007	Left	Don't know	Mil
8EY6...	1950	Female	True	True	2012	Right	Don't know	Mil
RDJH...	1948	Male	False	False	-----	None	Don't know	---
ACSK...		Male	True	True	2016	Left	Don't know	Me

Рис. 14.12. Просмотр DataFrame, созданного на предыдущем шаге, в PyCharm прост и красочен

Из этого я вижу, что мне еще есть над чем работать. Год постановки диагноза спутан, как и некоторые другие области. Давайте продолжим от этого избавляться.

ОЧИСТКА ДАННЫХ

Теперь, когда мы можем видеть наши данные в табличном формате, есть несколько способов улучшить формат этих данных с конкретной целью выполнения числового анализа по любым измерениям, которые мы можем выбрать.

Преобразование числовых данных в реальные числа

Наша следующая ячейка содержит несколько строк кода, предназначенных для преобразования числовых значений в числовые типы. Помните, что все поступает в виде текста и обрабатывается как строка, пока вы не укажете pandas иное. Вот код ячейки:

```
##% Change numeric data into appropriate format

# force some columns to have numeric data type
user_df['BirthYear'] = pd.to_numeric(user_df['BirthYear'], errors='coerce')
user_df['DiagnosisYear'] = pd.to_numeric(user_df['DiagnosisYear'], errors='coerce')
```

У программиста приложения может возникнуть соблазн обрабатывать данные построчно и выполнять преобразования типов поле за полем. Особенность pandas заключается в том, что, поместив данные в DataFrame, вы можете работать с целыми строками и столбцами.

В этом коде мы делаем именно это. BirthYear и DiagnosisYear преобразуются в числа с помощью метода `pd.to_numeric`. Второй аргумент, `error='coerce'`, будет пытаться принудительно преобразовать данные в числовой тип. Если это невозможно, например при значении «-----» (последовательность тире), которое мы видели на панели SciView на предыдущем шаге, тире заменяются в DataFrame нижним значением условия NaN, или «нечислом»¹. Хотя NaN не имеет вычислительной ценности, оно, по крайней мере, стандартизирует все нечисловые значения только до этого, что облегчит игнорирование этих строк, если мы захотим.

Упоминание NaN также указывает на то, что пришло время испечь вкусный хлеб в мамином тандыре. Некоторые авторы используют Patreon, а я – хлеб, но это должен быть рецепт вашей мамы. Это означает, что вы должны позвонить ей и сказать, что любите ее. Сделайте это сейчас, даже если у нее нет тандыра и она не умеет печь хлеб! Я подожду.

Пока вы разговаривали по телефону, я запустил ячейку; мой результат показан на рис. 14.13:

	BirthYear	Gender	Parkinsons	Tremors	DiagnosisYear	Sided	UPDRS
HQG...	nan	Male	True	True	2014.00000	None	Don't know
BIGW...	1937.00000	Male	True	False	2016.00000	Right	Don't know
BENX...	1960.00000	Female	False	False	nan	None	Don't know
M4A...	1950.00000	Female	False	False	nan	Left	Don't know
NMM...	1948.00000	Male	True	False	2016.00000	Right	Don't know
NPAC...	1965.00000	Female	True	True	2016.00000	Left	Don't know
GEY...	1929.00000	Male	False	False	nan	None	Don't know
GMCJ...	1949.00000	Female	True	False	1995.00000	Right	Don't know
OOL3...	1944.00000	Female	True	True	2012.00000	None	Don't know
LKYP...	1946.00000	Female	True	True	2017.00000	Right	Don't know
AXXX...	1949.00000	Female	False	False	nan	None	Don't know
L7Q0...	1963.00000	Male	True	False	2004.00000	Right	Don't know
EX8J...	1943.00000	Male	True	False	2007.00000	Left	Don't know
8EY6...	1950.00000	Female	True	True	2012.00000	Right	Don't know
RDJH...	1948.00000	Male	False	False	nan	None	Don't know
ACSK...	nan	Male	True	True	2016.00000	Left	Don't know

Рис. 14.13. Поля года не являются фактическими числами. Везде, где были неверные данные, теперь мы видим стандартизированное значение NaN

¹ NaN (от англ. *Not-a-Number*) — одно из особых состояний числа с плавающей запятой. Используется во многих математических библиотеках и математических сопроцессорах. Данное состояние может возникнуть в различных случаях, например когда предыдущая математическая операция завершилась с неопределенным результатом или если в ячейку памяти попало не удовлетворяющее условиям число. – *Прим. ред.*

Бинаризация данных

Мы должны это делать в любом месте, где можем конвертировать данные, которые, по сути, являются двоичными. В наших данных пол указывается с двумя возможными значениями: мужской и женский, что дает возможность представить его в двоичном формате. Аналогично, многие поля представлены в виде двоичных файлов, как показано здесь:

```
Parkinsons: True
Tremors: True
Levodopa: True
DA: True
MAOB: False
Other: False
```

В этих случаях нам просто нужно стандартизировать значения в виде двоичных файлов, что может привести к переименованию или расширению списка имен полей. Давайте посмотрим на код ячейки:

```
### "Binarize" true-false data

user_df = user_df.rename(index=str, columns={'Gender': 'Female'})
user_df['Female'] = user_df['Female'] == 'Female'
user_df['Female'] = user_df['Female'].astype(int)
```

В предыдущем коде мы переименовали столбец в нашем DataFrame с Gender на Female. Вторая строка изменяет значение в каждой строке для недавно переименованного столбца на результат выражения, сравнивающего текущее значение со словом Female. Это либо Female, либо нет, поэтому мы возвращаем значение True или False. Третья строка преобразует логический тип в целое число, что делает его более удобным для анализа.

Далее обратим внимание на ранее перечисленные столбцы и выполним то же преобразование. На этот раз мы проверяем наличие в нашем выражении слова «True». Значение либо True, либо нет, что приводит к логическому значению:

```
str_to_binary_columns = ['Parkinsons', 'Tremors', 'Levodopa', 'DA', 'MAOB',
                          'Other'] # columns to be converted to binary data

for column in str_to_binary_columns:
    user_df[column] = user_df[column] == 'True'
    user_df[column] = user_df[column].astype(int)
```

Запуск этого кода приводит к изменениям в нашем DataFrame, как показано на рис. 14.14:

	BirthYear	Female	Parkinsons	Tremors	DiagnosisYear	Sided	UPDRS	
HQG...	nan	0	1	1	2014.00000	None	Don't know	Me
BIGW...	1937.00000	0	1	0	2016.00000	Right	Don't know	Me
BENX...	1960.00000	1	0	0	nan	None	Don't know	---
M4A...	1950.00000	1	0	0	nan	Left	Don't know	---
NMM...	1948.00000	0	1	0	2016.00000	Right	Don't know	Mil
NPAC...	1965.00000	1	1	1	2016.00000	Left	Don't know	Me
GEY...	1929.00000	0	0	0	nan	None	Don't know	---
GMCJ...	1949.00000	1	1	0	1995.00000	Right	Don't know	Me
OOL3...	1944.00000	1	1	1	2012.00000	None	Don't know	Mil
LKYP...	1946.00000	1	1	1	2017.00000	Right	Don't know	Mil
AXXX...	1949.00000	1	0	0	nan	None	Don't know	---
L7Q0...	1963.00000	0	1	0	2004.00000	Right	Don't know	Me
EX8J...	1943.00000	0	1	0	2007.00000	Left	Don't know	Mil
8EY6...	1950.00000	1	1	1	2012.00000	Right	Don't know	Mil
RDJH...	1948.00000	0	0	0	nan	None	Don't know	---
ACSK...	nan	0	1	1	2016.00000	Left	Don't know	Me

user_df Format: %s

Рис. 14.14. Мы успешно бинаризовали наши поля

Мы видим, что наши поля теперь представляют собой двоичные числа! Позже это облегчит задачу.

Давайте перейдем к следующей ячейке, поскольку первая часть кода выполняет дополнительную очистку, аналогичную той, что мы делали до сих пор. В первой части ячейки зачищаем поле Impact. Мы стандартизируем любое значение, кроме Mild, Medium или Severe, как None:

```
# prior processing for 'Impact' column
user_df.loc[
    (user_df['Impact'] != 'Medium') &
    (user_df['Impact'] != 'Mild') &
    (user_df['Impact'] != 'Severe'), 'Impact'] = 'None'
```

Далее, оставаясь в той же ячейке, мы собираемся изучить мощный и популярный метод, который используется алгоритмом ML¹ для подготовки данных к анализу.

Горячее кодирование

По какой-то причине, когда я впервые услышал термин **one-hot encoding** («горячее кодирование», или «быстрое кодирование»), я сразу подумал о хот-догах и о том, как бы мне хотелось закодировать один из них с горчицей и сладким

¹ Machine Learning – машинное обучение. – Прим. ред.

маринадом на вкусной булочке на пару, или, может быть, о том NaN, который вы все готовите, чтобы отправить мне.

Горячее кодирование – это метод, позволяющий нам брать данные, которые по своей сути не являются логическими, и делать их таковыми. Выбирая новый Jeep Wrangler, я рассматривал всего несколько цветов:

- Firecracker Red,
- Ocean Blue Metallic,
- Mojito!
- Hellayella.

Цветов больше, но все они представляют собой скучные варианты черного, белого или серого. Я не могу купить оранжевый джип, потому что люди подумают, что я учился в Университете штата Оклахома, а у нас это невозможно. Я могу игнорировать эти цвета, оставив себе список, который поместится на странице. Теперь давайте быстро закодируем этот список:

Color_Firecracker_Red	Color_Ocean Blue Metallic	Color_Mojito	Color_Hellayella
1	0	0	0
0	1	0	0
0	0	1	0
0	0	0	0
0	0	0	1

Вы можете легко увидеть, как работает горячее кодирование: оно поворачивает поля¹, а затем делает их двоичными. Если вы гуру реляционных баз данных, вы, вероятно, только что потеряли обед. Ученые, работающие с данными, поступают немного по-другому. В представлении с горячим кодированием каждое наблюдение получает «1» в столбце, соответствующем его категории, и «0» во всех остальных столбцах. Такое кодирование гарантирует, что категориальная информация сохраняется таким образом, чтобы алгоритмы ML могли ее понять и эффективно использовать. Для справки, я выбрал *Hellayella*², основываясь на идее, что, если мой джип застрянет где-нибудь в недоступном месте, например в пустынях национального парка Биг-Бенд или глубоко в районе Пайни-Вудс на востоке Техаса, спасательные вертолеты легко найдут мой труп³.

Горячее кодирование обычно используется для таких функций как категориальные переменные, которые нельзя напрямую использовать в качестве числовых входных данных во многих алгоритмах машинного обучения. Важ-

¹ Поворот (англ. *pivot*) в контексте обработки данных относится к операции изменения формы, которая преобразует данные из длинного формата в широкий или наоборот. Этот термин возник из идеи «поворота» или «поворота» данных для просмотра их под другим углом, подобно тому, как можно поворачивать физический объект, чтобы изучить его с разных точек зрения. – Прим. ред.

² Можно перевести как «адски желтый». – Прим. ред.

³ Эти места славятся как раз ярко-желтыми пейзажами. – Прим. ред.

ным шагом предварительной обработки данных является преобразование таких переменных в подходящий формат для обучения моделей.

Вернемся к нашему коду для текущей ячейки. Мы объяснили первые несколько строк, поэтому давайте перейдем к настройке горячего кодирования для нескольких полей:

```
to_dummy_column_indices = ['Sided', 'UPDRS', 'Impact'] # columns to be one-hot encoded
```

Мы собираемся закодировать эти три столбца. Одним из рассматриваемых столбцов является столбец `Impact`, который мы только что стандартизировали как вводный для этого шага. Здесь мы выполним горячее кодирование для всех трех столбцов:

```
for column in to_dummy_column_indices:
    user_df = pd.concat([
        user_df.iloc[:, : user_df.columns.get_loc(column)],
        pd.get_dummies(user_df[column], prefix=str(column)),
        user_df.iloc[:, user_df.columns.get_loc(column) + 1 :]
    ], axis=1)
print(user_df.head())
```

Внутри цикла код выполняет следующие шаги.

1. `user_df.iloc[:, : user_df.columns.get_loc(column)]`: выбирает столбцы слева от текущего обрабатываемого столбца. Это сохраняет столбцы перед тем, который был подвергнут горячему кодированию.
2. `pd.get_dummies(user_df[column], prefix=str(column))`: применяет горячее кодирование к текущему столбцу с помощью метода `pd.get_dummies()`. Он создает `DataFrame` с двоичными столбцами, представляющими различные категории в столбце. Параметр `prefix` добавляет префикс к именам столбцов, чтобы указать, из какого исходного столбца они были получены.
3. `user_df.iloc[:, user_df.columns.get_loc(column) + 1 :]`: выбирает столбцы справа от текущего обрабатываемого столбца. Это сохраняет столбцы после одного горячего кодирования.

При передаче в метод `pd.concat` эти шаги эффективно заменяют каждый из трех категориальных столбцов двоичными столбцами с горячим кодированием, сохраняя при этом остальную часть `DataFrame` нетронутой. Запустив ячейку, вы должны увидеть результаты, подобные моим, как показано на рис. 14.15.

При горячем кодировании в `DataFrame` будет добавлено много новых столбцов, поэтому вам может потребоваться прокрутить вправо, чтобы увидеть их все.

	Sided_Left	Sided_None	Sided_Right	UPDRS_1	UPDRS_2	UPDRS_3	UPDRS_4
HQG...	True	False	False	False	False	False	False
BIGW...	False	True	False	False	False	False	False
BENX...	True	False	False	False	False	False	False
M4A...	False	False	False	False	False	False	False
NMM...	False	True	False	False	False	False	False
NPAC...	False	False	False	False	False	False	False
GEY...	True	False	False	False	False	False	False
GMCJ...	False	True	False	False	False	False	False
OOL3...	True	False	False	False	False	False	False
LKYP...	False	True	False	False	False	False	False
AXXX...	True	False	False	False	False	False	False
L7QO...	False	True	False	False	False	False	False
EX8J...	False	False	False	False	False	False	False
8EY6...	False	True	False	False	False	False	False
RDJH...	True	False	False	False	False	False	False
ACSK...	False	False	False	False	False	False	False

Рис. 14.15. Я прокрутил вправо, чтобы вы могли увидеть недавно добавленные столбцы с горячим кодированием

Изучение второго набора данных

Поскольку наши пользовательские данные достаточно хорошо очищены и помещены в pandas DataFrame, мы теперь готовы заняться данными Tappu. Чтобы сохранить связь, я собираюсь произвольно выбрать один файл из набора Tappu Data. Давайте посмотрим на код в нашей следующей ячейке:

```
## Explore the second dataset
file_name = '0EA27ICBLF_1607.txt'
```

Как я уже сказал, я выбрал для изучения один произвольный файл. Ранее мы открыли один из этих файлов и заметили, что все они имеют формат, разделенный табуляцией. У pandas есть метод, который легко прочитает этот файл непосредственно в DataFrame. Несмотря на то что метод называется `read_csv`, вы можете указать разделитель, который не обязательно должен быть запятой. Метод будет читать любой файл с разделителями:

```
df = pd.read_csv(
    'data/Tappy Data/' + file_name,
    delimiter = '\t',
    index_col = False,
    names = ['UserKey', 'Date', 'Timestamp', 'Hand', 'Hold time',
'Direction', 'Latency time', 'Flight time']
)
```

Для наших целей нам не нужно поле UserKey:

```
df = df.drop('UserKey', axis=1)

print(df.head())
```

Запуская эту ячейку, мы создаем новый DataFrame с именем df. Обязательно выберите его на панели переменных консоли, показанной на рис. 14.16:

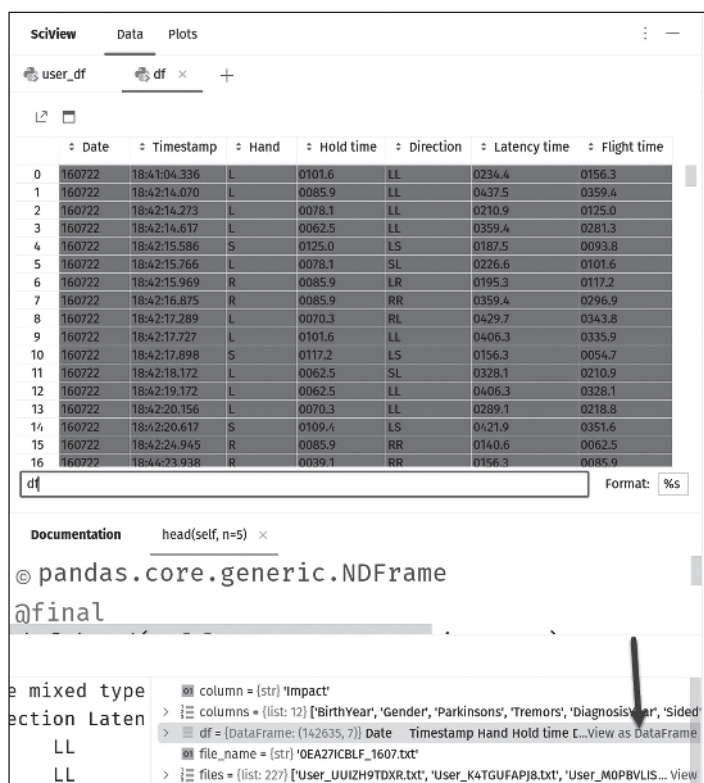


Рис. 14.16. Наш новый DataFrame можно просмотреть, нажав кнопку View as DataFrame

Форматирование данных даты и времени

Следующая ячейка фиксирует наши данные Datetime:

```
### Format datetime data
```

Эта первая строка пытается заставить значения в столбце Date быть датами. Если принуждение не сработает, мы увидим NaT (Not a Time, «невремя»), что разочаровывает, поскольку здесь нет никакой шутки о еде. Далее выполним еще несколько принудительных изменений в полях Hold time, Latency time и Flight time:

```
df['Date'] = pd.to_datetime(df['Date'], errors='coerce',
format='%y%M%d').dt.date
# converting time data to numeric
for column in ['Hold time', 'Latency time', 'Flight time']:
df[column] = pd.to_numeric(df[column], errors='coerce')
```

Любые наблюдения, в которых отсутствуют временные данные, должны быть исключены:

```
df = df.dropna(axis=0)
```

Распечатаем результат для проверки:

```
print(df.head())
```

Давайте запустим это! Результаты моего запуска ячейки показаны на рис. 14.17:

	Date	Timestamp	Hand	Hold time	Direction	Latency time	Flight time
0	2016-01-22	18:41:04.336	L	101.60000	LL	234.40000	156.30000
1	2016-01-22	18:42:14.070	L	85.90000	LL	437.50000	359.40000
2	2016-01-22	18:42:14.273	L	78.10000	LL	210.90000	125.00000
3	2016-01-22	18:42:14.617	L	62.50000	LL	359.40000	281.30000
4	2016-01-22	18:42:15.586	S	125.00000	LS	187.50000	93.80000
5	2016-01-22	18:42:15.766	L	78.10000	SL	226.60000	101.60000
6	2016-01-22	18:42:15.969	R	85.90000	LR	195.30000	117.20000
7	2016-01-22	18:42:16.875	R	85.90000	RR	359.40000	296.90000
8	2016-01-22	18:42:17.289	L	70.30000	RL	429.70000	343.80000
9	2016-01-22	18:42:17.727	L	101.60000	LL	406.30000	335.90000
10	2016-01-22	18:42:17.898	S	117.20000	LS	156.30000	54.70000
11	2016-01-22	18:42:18.172	L	62.50000	SL	328.10000	210.90000
12	2016-01-22	18:42:19.172	L	62.50000	LL	406.30000	328.10000
13	2016-01-22	18:42:20.156	L	70.30000	LL	289.10000	218.80000
14	2016-01-22	18:42:20.617	S	109.40000	LS	421.90000	351.60000
15	2016-01-22	18:42:24.945	R	85.90000	RR	140.60000	62.50000
16	2016-01-22	18:44:23.938	R	39.10000	RR	156.30000	85.90000

Рис. 14.17. Наши данные даты и времени теперь имеют числовой характер, и любые наблюдения с отсутствующими данными о времени, будучи бесполезными, были удалены

Моем руки и фиксируем направление

Следующая ячейка отмывает руки и столбцы направлений:

```
# cleaning data in Hand
df = df[
    (df['Hand'] == 'L') |
    (df['Hand'] == 'R') |
    (df['Hand'] == 'S')
]
```

Этот код использует логическое OR для фильтрации всего, что не имеет значения L, R или S. Поскольку оно представлено как OR, все, что выходит за рамки трех желаемых возможностей, вернется как false и будет исключено.

Давайте сделаем то же самое с направлением, у которого больше возможностей:

```
# cleaning data in Direction
df = df[
    (df['Direction'] == 'LL') |
    (df['Direction'] == 'LR') |
    (df['Direction'] == 'LS') |
    (df['Direction'] == 'RL') |
    (df['Direction'] == 'RR') |
    (df['Direction'] == 'RS') |
    (df['Direction'] == 'SL') |
    (df['Direction'] == 'SR') |
    (df['Direction'] == 'SS')
]
```

Конечно, мы напечатаем результат:

```
print(df.head())
```

Идите вперед и запустите ячейку. Все строки, содержащие недопустимые данные, были удалены. Этот результат не так нагляден, как большинство других, поэтому я не думаю, что нам нужен этот скриншот.

Обобщение данных

Наша следующая ячейка представляет собой пример того, как суммировать данные, над которыми мы так усердно работали, чтобы подготовить их для анализа. Мы готовы! Давайте попробуем что-нибудь простое. Как обычно, первая строка кода в ячейке просто отмечает начало ячейки:

```
### Group by direction (hand transition)
```

Напомним, что данные, с которыми мы до сих пор работали, – это данные о скорости набора текста для конкретного субъекта в данный момент времени. Субъект (User) – это просто одна точка данных в нашем первом наборе данных, и мы хотели бы каким-то образом объединить два набора данных, поэтому нам нужен способ агрегировать текущие данные в одну точку данных.

Поскольку мы работаем с числовыми данными (время набора текста), можно взять среднее значение данных времени по разным столбцам как способ суммировать данные конкретного пользователя. Мы можем добиться этого с помощью функции `groupby()` из `pandas`:

```
direction_grouped_df = df.groupby('Direction')[numeric_columns].mean()
```

Конечно, следует это напечатать:

```
print(direction_grouped_df)
```

Результат прогона показан на рис. 14.18. Код помещает результаты в новый `DataFrame` с именем `direction_group_df`, поэтому обязательно выберите его, как показано на рис.:

The screenshot displays the SciView interface with the 'Data' tab active. A table of grouped data is shown, with columns for 'Hold time', 'Latency time', and 'Flight time'. Below the table, the variable explorer shows 'direction_group_df' selected. The code editor at the bottom shows the following code:

```
column = {str} 'Flight time'
> columns = {list: 12} ['BirthYear', 'Gender', 'Parkinsons', 'Tremors', 'DiagnosisYear', 'Sided', 'UndRS', 'Impa... View
> df = {DataFrame: (142615, 7)} Date Timestamp Hand ... Direction Latency time Flight ti... View as DataFrame
> direction_grouped_df = {DataFrame: (9, 3)} Hold time Latency time Flight time [Direction ... View as DataFrame
file_name = {str} '0EA27ICBLF_1607.txt'
```

Рис. 14.18. Ура! У нас есть первый рассчитанный анализ!

Это впечатляет! У нас есть работающая механика, но теперь нужно сконцентрироваться на том, чтобы все это работало со многими файлами данных, а не с одним.

Рефакторинг для масштабирования

Наше исследование данных Тарру было сосредоточено на одном файле, чтобы было легко проверить, работает ли наш код. Мы определили, что это так, поэтому теперь нам следует провести рефакторинг нашего кода, чтобы мы могли обрабатывать тысячи файлов. Для этого следует объединить некоторые наши

ячейки в функцию. Код в следующей ячейке длинный, но знакомый, поскольку это всего лишь весь код, который мы написали до сих пор, объединенный в одну функцию. Если вы разработчик приложений и понимаете принцип проектирования, известный как **принцип единой ответственности (SRP)**, вы знаете, что это антипаттерн (антишаблон). Однако помните, что это не код приложения. Никто не запустит это, кроме как для выполнения анализа, поэтому строгость принципов SOLID, которые обычно применяются к разработке программного обеспечения, не соблюдается в работе по науке о данных.

Обработка данных Tappy с помощью одной функции

Вот наша функция:

```
### Combine into one function

def read_tappy(file_name):
```

Здесь мы читаем имя файла CSV, переданное в качестве аргумента нашей функции. Мы обогащаем данные жестко запрограммированными именами полей:

```
df = pd.read_csv(
    'data/Tappy Data/' + file_name,
    delimiter='\t',
    index_col=False,
    names=['UserKey', 'Date', 'Timestamp', 'Hand', 'Hold time',
          'Direction', 'Latency time', 'Flight time']
)
```

Удаляем ненужный столбец:

```
df = df.drop('UserKey', axis=1)
```

Фиксируем даты:

```
df['Date'] = pd.to_datetime(df['Date'], errors='coerce',
    format='%y%M%d').dt.date

# Convert time data to numeric
for column in ['Hold time', 'Latency time', 'Flight time']:
    df[column] = pd.to_numeric(df[column], errors='coerce')
df = df.dropna(axis=0)
```

Всегда мойте руки, избавляясь от недопустимых значений:

```
# Clean data in `Hand`
df = df[
    (df['Hand'] == 'L') |
    (df['Hand'] == 'R') |
    (df['Hand'] == 'S')
]
```


Сделайте то же самое со значениями данных направлений:

```
# Clean data in 'Direction'
df = df[
    (df['Direction'] == 'LL') |
    (df['Direction'] == 'LR') |
    (df['Direction'] == 'LS') |
    (df['Direction'] == 'RL') |
    (df['Direction'] == 'RR') |
    (df['Direction'] == 'RS') |
    (df['Direction'] == 'SL') |
    (df['Direction'] == 'SR') |
    (df['Direction'] == 'SS')
]
```

Мы занимаемся математикой! Здесь на помощь приходит ручной процесс GC. Хорошо, что мы вымыли руки, не так ли? В следующем коде мы выполняем наши вычисления. Результаты возвращаются в виде нового DataFrame, поэтому для экономии памяти удаляем старые DataFrame по ходу работы. Это освобождает память, поскольку такая работа требует большого объема памяти:

```
direction_group_df = df.groupby('Direction')[numeric_columns][numeric_columns].mean()
del df
gc.collect()
```

Получив новый результат, мы переиндексируем, а затем сортируем:

```
direction_group_df = direction_group_df.reindex(['LL', 'LR', 'LS', 'RL', 'RR', 'RS', 'SL', 'SR', 'SS'])
direction_group_df = direction_group_df.sort_index() # to ensure correct order of data
```

Эта строка возвращает одномерный массив NumPy, который содержит средние значения сгруппированных данных. Метод `.values.flatten()` преобразует DataFrame в двумерный массив NumPy, а затем сглаживает его в одномерный массив для простоты использования:

```
return direction_group_df.values.flatten()
```

Обработка пользователей с помощью функции

В той же ячейке находится вторая функция:

```
def process_user(user_id, filenames):
    running_user_data = np.array([])
```

Эта строка инициализирует пустой массив NumPy с именем `running_user_data`. Этот массив будет использоваться для накопления данных по мере того, как функция перебирает имена файлов, что и делает следующий блок:

```
for filename in filenames:
    if user_id in filename:
        running_user_data = np.append(running_user_data, read_tappy(filename))
```

Этот цикл перебирает список имен файлов. Если предоставленный идентификатор пользователя найден в имени файла, он вызывает функцию `read_tappy()` (которая возвращает сглаженный массив средних значений NumPy) и добавляет его содержимое в массив `running_user_data`.

После перебора имен файлов и добавления данных следующая строка преобразует массив `running_user_data` в двумерный массив, каждая строка которого содержит 27 столбцов. Такое сглаживание временных данных позволяет провести дальнейший анализ:

```
running_user_data = np.reshape(running_user_data, (-1, 27))
```

Последняя строка вычисляет средние значения по строкам (`axis=0`) массива `running_user_data` с помощью `np.nanmean()`. Функция `np.nanmean()` игнорирует значения NaN при вычислении среднего значения:

```
return np.nanmean(running_user_data, axis=0)
```

Подводя итог, можно сказать, что функция `process_user` обрабатывает данные для конкретного пользователя, перебирая соответствующие имена файлов, агрегируя данные с помощью функции `read_tappy`, изменяя форму данных и вычисляя средние значения, игнорируя значения NaN. Конечным результатом является массив средних значений для каждого столбца данных.

Обработка всех данных

Решающий момент! Следующая ячейка обрабатывает данные для всех доступных пользователей, агрегируя и вычисляя средние значения на основе данных Tappy. Для начала немного приберемся. Мы собираемся игнорировать любые предупреждения:

```
#%% Run through all available data

import warnings
warnings.filterwarnings("ignore")
```

Совершим еще одно путешествие по папке Tappy Data:

```
filenames = os.listdir('data/Tappy Data/')
```

Далее создадим имена столбцов для окончательного DataFrame:

```
column_names = [first_hand + second_hand + '_' + time
                 for first_hand in ['L', 'R', 'S']
                 for second_hand in ['L', 'R', 'S']
                 for time in ['Hold time', 'Latency time', 'Flight time']]

user_tappy_df = pd.DataFrame(columns=column_names)
```

Далее давайте пройдемся по пользовательским индексам и воспользуемся нашей функцией `process_user`:

```
for user_id in user_df.index:
    user_tappy_data = process_user(str(user_id), filenames)
    user_tappy_df.loc[user_id] = user_tappy_data
```

Следующие несколько строк выполняют небольшую промежуточную очистку, гарантируя, что все значения NaN заменяются нулями, а любые отрицательные числовые данные также нормализуются до нуля:

```
user_tappy_df = user_tappy_df.fillna(0)
user_tappy_df[user_tappy_df < 0] = 0
```

А потом мы печатаем так, как не делали этого никогда раньше! Хорошо, это неправда – мы делали это много раз:

```
print(user_tappy_df.head())
```

Сохранение обработанных данных

Последняя ячейка кода, вероятно, не нуждается в особых пояснениях:

```
#%% Save processed data
```

Сначала мы объединяем два DataFrame вместе:

```
combined_user_df = pd.concat([user_df, user_tappy_df], axis=1)
print(combined_user_df.head())
```

Наконец, сохраняем его в файл CSV:

```
combined_user_df.to_csv('data/combined_user.csv')
```

Обычно это хорошая практика для конкретного конвейера данных. Сохранение обработанной, очищенной версии набора данных может сэкономить специалистам по обработке данных массу усилий, если что-то пойдет не так. Это также обеспечивает гибкость, если и когда мы захотим изменить или расширить наш конвейер.

Я открою CSV-файл в PyCharm, чтобы еще раз просмотреть его, прежде чем мы приступим к настоящей аналитической работе. Вы можете увидеть мой на рис. 14.19:

(1)	BirthYear (2)	Female (3)	Parkinsons (4)	Tremors (5)	DiagnosisYear (6)	Sided_Left (7)
1	BirthYear	Female	Parkinson	Tremors	DiagnosisYear	Sided_Left
2	ARUGT4UL2R	1955.0	0	1	1	0
3	UUIZH9TDXR	1958.0	1	1	0	2015.0
4	JIRZDKEJQN	1985.0	0	1	1	2014.0
5	XWAX2IHF3O	1970.0	1	1	1	2007.0
6	2X17VCRRQA	1951.0	0	1	1	2003.0
7	EF9XEFXPBN	1951.0	1	1	0	2016.0
8	BIGWMXFU3B	1937.0	0	1	0	2016.0
9	WXNQ0QM0XD	1959.0	1	0	0	0
10	OMCPRWYBSQ	1952.0	0	0	0	0
11	9ZRBC5WOZR	1943.0	1	1	1	2016.0

Рис. 14.19. Наша тяжелая работа окупилась! Данные готовы к анализу

Теперь мы готовы начать изучение нашего набора данных и поиск информации.

Анализ данных и INSIGHT¹

Помните, что мы говорили о важности иметь определенный вопрос, начиная работать над проектом науки о данных? Это особенно актуально на этом этапе, когда мы изучаем наш набор данных и извлекаем информацию, которая должна вращаться вокруг первоначального вопроса – связи между скоростью набора текста и возможным наличием у пациента БП.

В этом разделе мы будем работать с файлом `EDA.ipynb`, расположенным в папке `notebooks` нашего текущего проекта. В следующих подразделах мы рассмотрим код, включенный в эту папку `notebooks`. Откройте этот блокнот Jupyter в редакторе PyCharm или, если вы следите за нашими обсуждениями и вводите свой собственный код, создайте новый блокнот Jupyter.

Запускаем блокнот и считываем обработанные данные

Помните, что когда вы открываете блокнот Jupyter на Python, то можете увидеть код, но Jupyter не запустится, пока не нажмете кнопку **Run**. Вы можете видеть, что PyCharm готов к этому, на рис. 14.20:

¹ Insight – библиотека Python для мониторинга, сравнения и извлечения информации из данных. – Прим. ред.

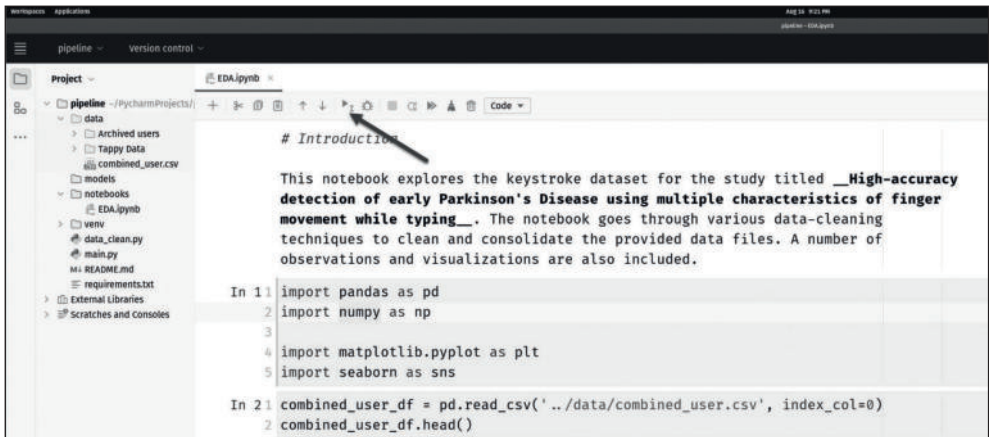


Рис. 14.20. Блокнот открыт, я кликнул первую ячейку (In 1) и теперь нажимаю кнопку Run, указанную стрелкой

Как только вы нажмете кнопку **Run**, сервер Jupyter запустится и запустит первую ячейку в блокноте, которая обрабатывает импорт и читает наш очищенный набор данных:

```
import pandas as pd
import numpy as np

import matplotlib.pyplot as plt
import seaborn as sns

combined_user_df = pd.read_csv('../data/combined_user.csv', index_col=0)
combined_user_df.head()
```

Поскольку в последней строке мы печатаем первые пять строк нашего вывода, вы увидите, что они появляются под кодом и рядом с маркером с надписью **Out 2**, как показано на рис. 14.21:

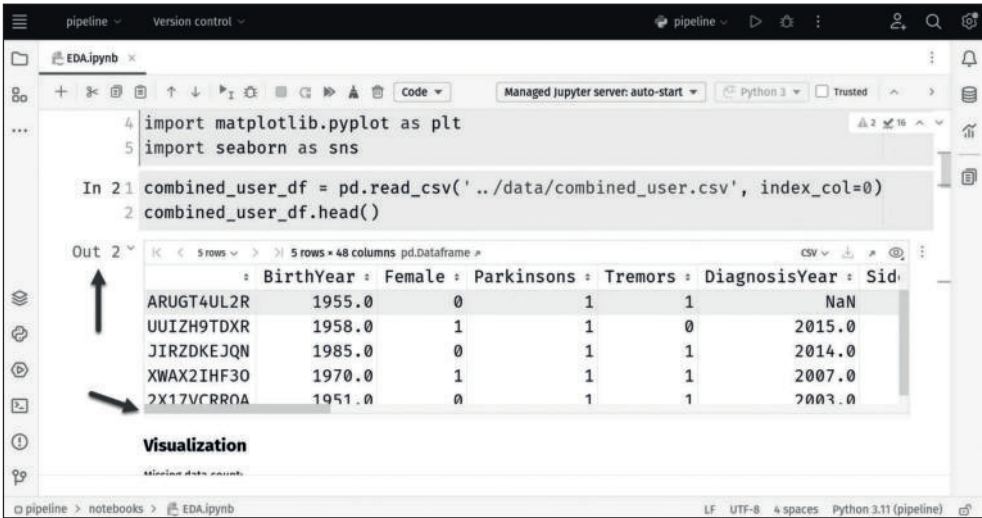


Рис. 14.21. Вывод оператора head в In 2 показан в Out 2 и доступен при горизонтальной прокрутке

Теперь, когда наши очищенные данные загружены, мы можем перейти к процессу анализа.

ИСПОЛЬЗОВАНИЕ ДИАГРАММ И ГРАФИКОВ

Конечной целью большей части моей работы обычно является визуализация, поэтому для меня это естественный следующий шаг. Я собираюсь начать с создания гистограммы, которая покажет мне распределение количества уникальных значений в данных. Я думаю, это может дать нам некоторое представление (insight) о том, какой фактор повлияет на зависимую переменную в этом исследовании, а именно есть ли у субъекта раннее начало болезни Паркинсона. Однако проблема все еще существует. Как показано на рис. 14.21, в данных все еще есть некоторые пробелы, которые мне нужно будет учесть, прежде чем я приступлю к серьезному анализу.

Сначала я собираюсь создать гистограмму, чтобы визуализировать недостающие данные. Следующая ячейка кода обрабатывает это:

```
###

missing_data = combined_user_df.isnull().sum()

g = sns.barplot(x=missing_data.index, y=missing_data)
g.set_xticklabels(labels=missing_data.index, rotation=90)

plt.show()
```

Запуск этого кода приводит к визуализации, показанной на рис. 14.22:

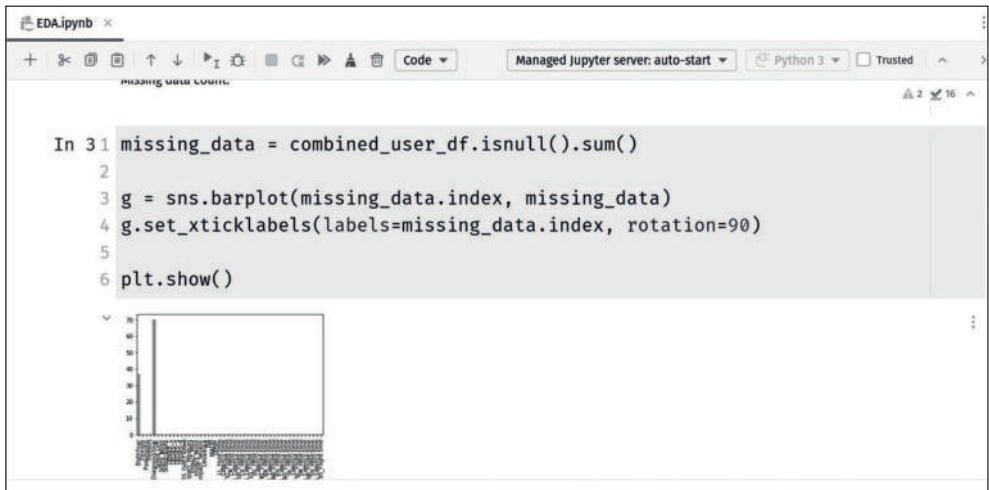


Рис. 14.22. Недостающие данные отображаются на гистограмме

К счастью, наша диаграмма очень скудна. Лишь небольшой объем данных отсутствует или является неполным. Отсутствуют некоторые значения для BirthYear и DiagnosisYear. Вы даже можете увидеть это в предварительном просмотре, показанном на рис. 14.21. Анализ пропущенных значений важен, и мы вернемся к процессу заполнения этих значений позже. А пока давайте продолжим процесс визуализации.

Замечательной функцией Matplotlib являются поддиаграммы, которые позволяют нам создавать несколько визуализаций одновременно. В следующей ячейке кода мы создаем несколько визуализаций с помощью этой функции, чтобы подчеркнуть потенциальные различия между пациентами с болезнью Паркинсона и без нее:

```
###

f, ax = plt.subplots(2, 2, figsize=(20, 10))

sns.distplot(
    combined_user_df.loc[combined_user_df['Parkinsons'] == 0,
    'BirthYear'].dropna(axis=0),
    kde_kws = {'label': "Without Parkinson's"},
    ax = ax[0][0]
)
sns.distplot(
    combined_user_df.loc[combined_user_df['Parkinsons'] == 1,
    'BirthYear'].dropna(axis=0),
    kde_kws = {'label': "With Parkinson's"},
    ax = ax[0][1]
)

sns.countplot(x='Female', hue='Parkinsons', data=combined_user_df,
ax=ax[1][0])
sns.countplot(x='Tremors', hue='Parkinsons', data=combined_user_df,
```



```
ax=ax[1][1])
plt.show()
```

После запуска этой ячейки кода будет сгенерирована визуализация, как показано на рис. 14.23:

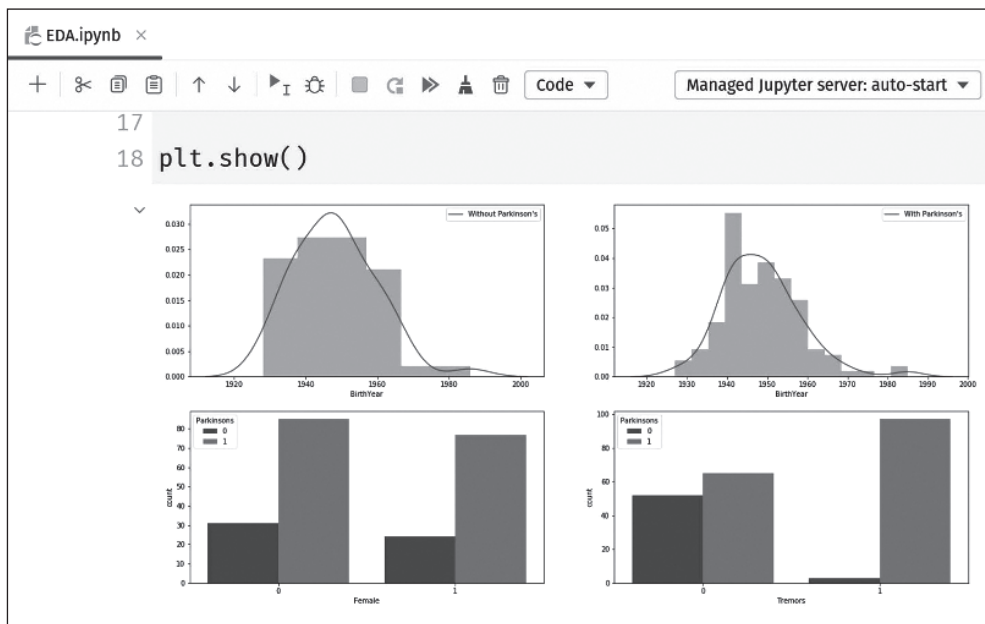


Рис. 14.23. Четыре диаграммы, составленные из предыдущей ячейки

Две верхние визуализации представляют распределение по годам рождения людей с болезнью Паркинсона (вверху справа) и без нее (вверху слева). Мы видим, что эти распределения примерно соответствуют нормальной колоколообразной кривой. Если вы столкнулись с искаженным распределением или странной формой, возможно, стоит углубиться в эти данные особо. Обратите внимание, что мы также можем применить эту же визуализацию для столбца `DiagnosisYear`.

В нижней левой визуализации у нас есть гистограмма, представляющая количество пациентов мужского пола (две полосы слева) и пациентов женского пола (две полосы справа). Пациенты с болезнью Паркинсона обозначаются оранжевыми столбиками, а пациенты без нее – синими столбцами. На этой визуализации мы видим, что, хотя пациентов с этим заболеванием больше, чем пациентов без него, гендерное распределение примерно одинаковое.

С другой стороны, визуализация в правом нижнем углу иллюстрирует распределение пациентов с тремором (две полосы справа) и пациентов без тремора (две полосы слева). Из этой визуализации мы видим, что тремор значительно чаще встречается у пациентов с болезнью Паркинсона, что вполне интуитивно понятно и может служить проверкой здравого смысла наших выводов.

Далее перейдем к коробчатым диаграммам¹. В частности, мы будем использовать коробчатые диаграммы для визуализации распределения различных временных данных (Hold time, Latency time, и Flight time) среди пациентов с болезнью Паркинсона и без нее. Еще раз мы будем использовать функцию поддиаграмм для одновременного создания нескольких визуализаций:

```
###

column_names = [first_hand + second_hand + '_' + time
for first_hand in ['L', 'R', 'S']]
for second_hand in ['L', 'R', 'S']
for time in ['Hold time', 'Latency time', 'Flight time']]

f, ax = plt.subplots(3, 3, figsize=(10, 5))

plt.subplots_adjust(
    right = 3,
    top = 3
)

for i in range(9):
    temp_columns = column_names[3 * i : 3 * i + 3]
    stacked_df = combined_user_df[temp_columns].stack().reset_index()

stacked_df = stacked_df.rename(
    columns={'level_0': 'index', 'level_1': 'Type', 0: 'Time'})
stacked_df = stacked_df.set_index('index')

for index in stacked_df.index:
    stacked_df.loc[index, 'Parkinsons'] = combined_user_df.loc[index, 'Parkinsons']

sns.boxplot(x='Type', y='Time',
    hue='Parkinsons',
    data=stacked_df,
    ax=ax[i // 3][i % 3]
).set_title(column_names[i * 3][2], fontsize=20)

plt.show()
```

В этой ячейке кода каждая поддиаграмма будет визуализировать данные определенного типа направления (LL, LR, LS и т. д.) и будет содержать различные разбиения, обозначающие пациентов с заболеванием и без него. Вы должны получить визуализацию, показанную на рис. 14.24:

¹ Коробчатая диаграмма используется для изучения одного или нескольких наборов данных в графическом виде. Данный тип диаграммы может использоваться для сравнения распределений между несколькими группами или наборами данных. Для каждой группы или набора данных вычисляются статистика центра (медиана, среднее) и статистики диапазона (квартили, стандартные отклонения) для различных моментов времени, и выбранные значения изображаются на диаграмме. – *Прим. ред.*

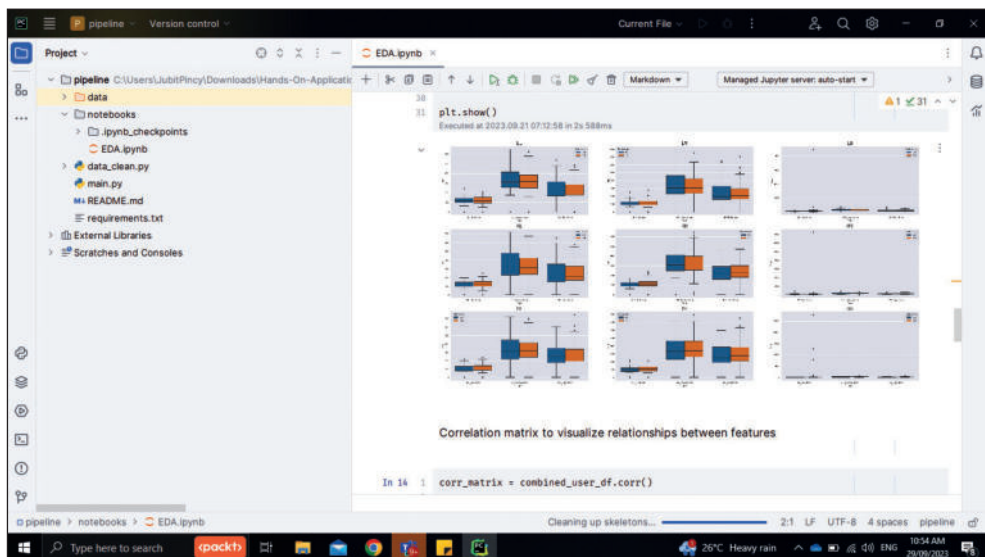


Рис. 14.24. Диаграммы из ячейки предыдущего запуска

Из этой визуализации мы можем понять, что, как ни удивительно, распределение скорости набора текста среди пациентов без болезни Паркинсона может иметь большую дисперсию, чем среди пациентов с болезнью Паркинсона, что может противоречить интуитивному мнению, которое некоторые могут иметь о том, что пациенты с болезнью Паркинсона всегда тратят больше времени на набор текста на клавиатуре.

В целом гистограммы, графики распределения и коробчатые диаграммы¹ являются одними из наиболее распространенных методов визуализации в задачах науки о данных, главным образом потому, что они просты для понимания и достаточно эффективны, чтобы выделить важные закономерности в наших наборах данных. В следующем и последнем подразделе, посвященном теме анализа данных, мы рассмотрим более продвинутые методы, а именно матрицу корреляции между атрибутами и использование моделей ML.

АНАЛИТИКА НА ОСНОВЕ МАШИННОГО ОБУЧЕНИЯ

В отличие от предыдущих методов анализа методы, обсуждаемые в этом подразделе и другие подобные, основаны на более сложных математических моделях и алгоритмах машинного обучения. Учитывая объем этой книги, мы не будем вдаваться в конкретные теоретические детали этих моделей, но все же стоит увидеть некоторые из них в действии, применив их к нашему набору данных.

Во-первых, давайте рассмотрим матрицу корреляции признаков для нашего набора данных. Как следует из названия, эта модель представляет собой матрицу (двумерную таблицу), содержащую корреляцию между каждой парой

¹ Их еще называют «ящики с усами». – Прим. ред.

числовых атрибутов (или признаков) в нашем наборе данных. Корреляция между двумя объектами – это действительное число от -1 до 1 , указывающее величину и направление корреляции. Чем выше значение, тем сильнее коррелируют эти две функции.

Чтобы получить матрицу корреляции функций из DataFrame pandas, мы должны вызвать метод `corr()`, как показано здесь:

```
corr_matrix = combined_user_df.corr()
```

Обычно мы визуализируем корреляционную матрицу с помощью тепловой карты, реализованной в той же ячейке кода:

```
f, ax = plt.subplots(1, 1, figsize=(15, 10))
sns.heatmap(corr_matrix)

plt.show()
```

Этот код создаст визуализацию, показанную на рис. 14.25:

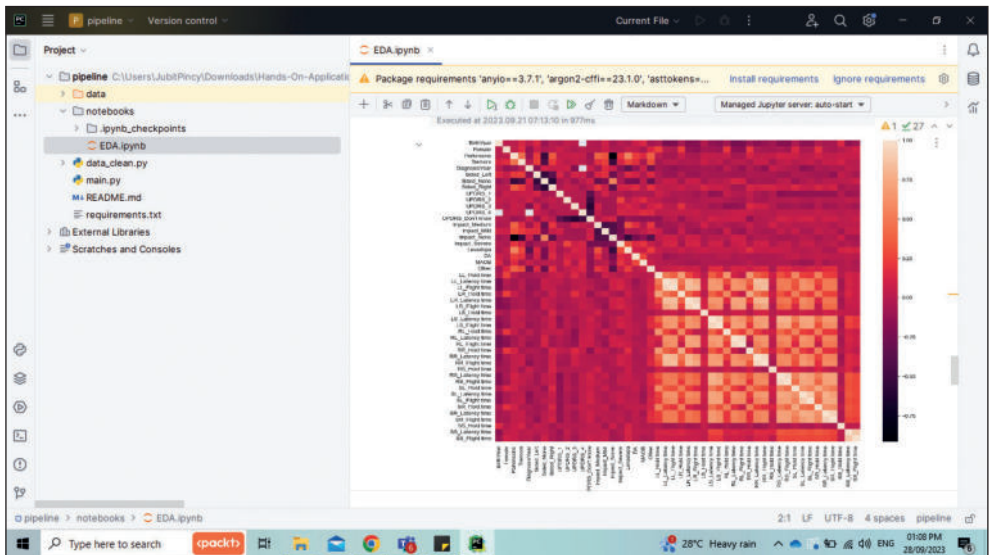


Рис. 14.25. Тепловая карта идеально подходит для визуализации корреляционных матриц

Далее попробуем применить модель ML к нашему набору данных. Вопреки распространенному мнению, во многих проектах науки о данных мы не используем преимущества моделей машинного обучения для задач прогнозирования, где мы обучаем наши модели, чтобы они могли прогнозировать будущие данные. Вместо этого мы передаем наш набор данных конкретной модели, чтобы можно было извлечь больше информации из имеющегося набора данных.

Здесь мы используем модель **классификатора опорных векторов (SVC)** из `scikit-learn` для анализа имеющихся у нас данных и возврата списка важности функций:

```

###

from sklearn.svm import LinearSVC

combined_user_df['BirthYear'].fillna(combined_user_df['BirthYear'].
mode(dropna=True)[0], inplace=True)
combined_user_df['DiagnosisYear'].fillna(combined_user_
df['DiagnosisYear'].mode(dropna=True)[0], inplace=True)

X_train = combined_user_df.drop(['Parkinsons'], axis=1)
y_train = combined_user_df['Parkinsons']

clf = LinearSVC()
clf.fit(X_train, y_train)

nfeatures = 10

coef = clf.coef_.ravel()
top_positive_coefs = np.argsort(coef)[-nfeatures :]
top_negative_coefs = np.argsort(coef)[: nfeatures]
top_coefs = np.hstack([top_negative_coefs, top_positive_coefs])

```

Обратите внимание: прежде чем передать имеющиеся у нас данные в модель ML, необходимо заполнить недостающие значения в двух столбцах, которые мы определили ранее, – BirthYear и DiagnosisYear. Большинство моделей машинного обучения не могут хорошо обрабатывать пропущенные значения, и инженеры по обработке данных должны выбирать, как эти значения следует заполнять.

Здесь мы используем **mode**, или наиболее часто встречающуюся точку данных этих двух столбцов, чтобы заполнить недостающие значения. Это связано с тем, что мода является одной из статистических характеристик, которые имеют тенденцию хорошо представлять диапазон различных типов данных, особенно для дискретных или номинальных атрибутов, которые мы здесь имеем. Если вы работаете с числовыми и непрерывными данными, такими как длина или площадь, также общепринятой практикой является использование среднего значения данного атрибута. Наконец, возвращаясь к нашему текущему процессу, этот код обучает модель на нашем наборе данных и впоследствии получает атрибут модели `coef_`.

Этот атрибут содержит список важности функций, который визуализируется в последнем разделе кода:

```

plt.figure(figsize=(15, 5))
colors = ['red' if c < 0 else 'blue' for c in coef[top_coefs]]
plt.bar(np.arange(2 * nfeatures), coef[top_coefs], color=colors)
feature_names = np.array(X_train.columns)
# Make sure the number of tick locations matches the number of tick labels.
plt.xticks(np.arange(0, 2 * nfeatures), feature_names[top_coefs], rotation=60,
ha='right')

plt.show()

```

Запуск этого кода приводит к визуализации, показанной на рис. 14.26:

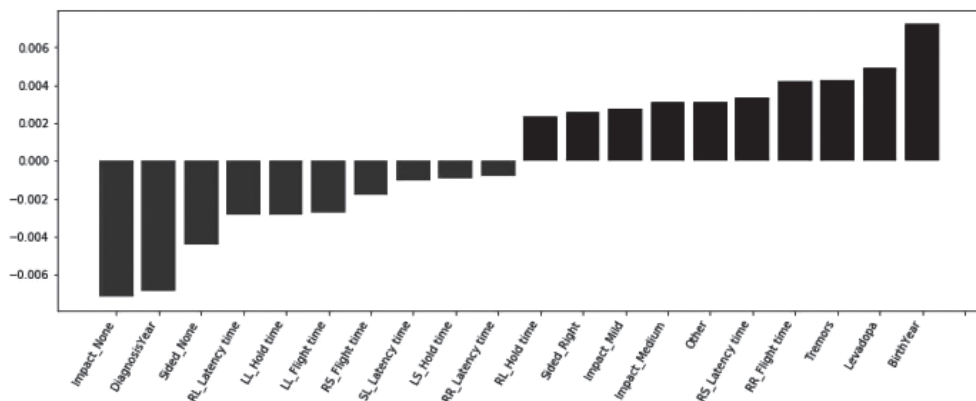


Рис. 14.26. Диаграмма списка важных функций показывает функции, которые широко используются при обучении модели машинного обучения

Из списка важности функций мы можем определить любые функции, которые широко использовались моделью ML во время обучения. Признак с очень высоким значением важности можно каким-то интересным образом соотнести с целевым атрибутом (есть ли у кого-то болезнь Паркинсона или нет). Например, мы видим, что Tremors (который, как мы знаем, вполне коррелирует с нашим целевым атрибутом) является третьей по важности функцией нашей текущей модели машинного обучения.

Это последний пункт данного обсуждения, касающийся анализа нашего набора данных. В последнем разделе этой главы мы кратко обсудим, как написать скрипт для проекта Python по науке о данных.

СКРИПТЫ ПРОТИВ БЛОКНЕТОВ В НАУКЕ О ДАННЫХ

В предыдущем конвейере обработки данных есть два основных этапа: очистка данных, где мы удаляем противоречивые данные, заполняем недостающие данные и соответствующим образом кодируем атрибуты, и анализ данных, где мы генерируем визуализации и аналитическую информацию из нашего очищенного набора данных.

Процесс очистки данных был реализован с помощью скрипта Python, а процесс анализа данных выполнялся с помощью блокнота Jupyter. В общем, решение о том, следует ли создавать программу Python в скрипте или в блокноте, является довольно важным, но часто упускаемым из виду аспектом при работе над проектом науки о данных.

Как мы обсуждали в предыдущей главе, блокноты Jupyter идеально подходят для итеративных процессов разработки, где мы можем преобразовывать и манипулировать нашими данными по ходу работы. Скрипт Python, с другой стороны, не предлагает такого динамизма. Нам нужно ввести весь необходимый код в скрипт и запустить его как законченную программу.

Однако, как показано в разделе «Очистка и предварительная обработка данных», PyCharm позволяет нам разделить традиционный скрипт Python на отдельные ячейки кода и проверять имеющиеся у нас данные по ходу работы с помощью панели **SciView**. Динамику, которую предлагают блокноты Jupyter, можно также найти в PyCharm.

Еще одно основное различие между обычными скриптами Python и блокнотами Jupyter заключается в том, что печатные данные и визуализации включаются в блокнот вместе с ячейками кода, которые их сгенерировали. Глядя на это с точки зрения специалиста по данным, мы видим, что эта функция весьма полезна при создании отчетов и презентаций.

Допустим, вам поручено найти действенную информацию из набора данных в проекте компании, и нужно представить окончательные результаты, а также то, как вы пришли к ним со своей командой. Блокнот Jupyter может эффективно служить основной платформой для вашей презентации. Люди не только смогут увидеть, какие конкретные команды использовались для обработки и манипулирования исходными данными, но вы также сможете включать тексты Markdown для дальнейшего объяснения любых тонких моментов обсуждения.

Обычные Python-скрипты можно просто использовать для низкоуровневых задач, где общий рабочий процесс уже согласован, и вам не нужно будет его кому-либо представлять. В нашем текущем примере я решил очистить набор данных с помощью скрипта Python, поскольку большинство изменений очистки и форматирования, которые мы применили к набору данных, не дают никакой практической информации, которая могла бы ответить на наш первоначальный вопрос. Я использовал блокнот только для задач по анализу данных, где было много визуализаций и идей, достойных дальнейшего обсуждения.

В целом решение использовать традиционный скрипт Python или блокнот Jupyter зависит исключительно от ваших задач и целей. Нам просто нужно помнить, что, какой бы инструмент мы ни хотели использовать, PyCharm предлагает невероятную поддержку, которая может сильно упростить наш рабочий процесс.

КРАТКОЕ СОДЕРЖАНИЕ

В этой главе мы рассмотрели практический процесс работы над конвейером обработки данных. Во-первых, мы обсудили важность контроля версий не только для нашего кода и файлов, связанных с проектом, но и для наших наборов данных; затем научились использовать Git LFS для применения контроля версий к большим файлам и наборам данных.

Далее рассмотрели различные методы очистки и предварительной обработки данных, специфичные для примера набора данных. Используя панель в PyCharm, мы можем динамически проверять текущее состояние наших данных и переменных и видеть, как они меняются после каждой команды.

Наконец, мы рассмотрели несколько методов создания визуализаций и извлечения информации из нашего набора данных. Используя редактор Jupyter в PyCharm, мы смогли избежать работы с сервером Jupyter и работать над нашим блокнотом полностью внутри PyCharm. Пройдя этот процесс, вы теперь готовы решать реальные проблемы и проекты в области науки о данных, используя те же инструменты и функции, которые мы обсуждали до сих пор.

Итак, мы завершили обсуждение использования PyCharm в контексте научных вычислений и обработки данных. В следующей главе, наконец, рассмотрим тему, о которой мы неоднократно упоминали в предыдущих главах, – плагины PyCharm.

Вопросы

Ответьте на следующие вопросы, чтобы проверить свои знания по этой главе.

1. Каковы основные способы сбора данных для проекта по науке о данных?
2. Можно ли использовать Git LFS с Git? Если да, то каков общий процесс?
3. Для какого типа атрибута недостающие значения могут быть заполнены средним значением? А что насчет моды?
4. Какую проблему решает горячее кодирование? Какая проблема может возникнуть при использовании горячего кодирования?
5. Какой тип атрибута может быть полезен при использовании гистограмм? А как насчет участков распределения?
6. Почему важно учитывать матрицу корреляции признаков для набора данных?
7. Помимо задач прогнозирования, для чего мы можем использовать модели машинного обучения (как мы это делали в данной главе)?

Дальнейшее чтение

Более подробную информацию можно найти в следующих статьях и материалах для чтения:

- Adams, W. R. (2017). *High-accuracy detection of early Parkinson’s Disease using multiple characteristics of finger movement while typing*. PloS one, 12(11), e0188226,
- The Tappy Keystroke Data with Parkinson’s Patients data, uploaded by Patrick DeKelly: <https://www.kaggle.com/valking/tappy-keystroke-data-with-parkinsons-patients>,
- *Building a Data Pipeline from Scratch*, by Alan Marazzi: <https://medium.com/the-data-experience/building-a-data-pipeline-from-scratch-32b712cfb1db>,
- *DataScienceforStartups:DataPipelines*, by Ben Weber: <https://towardsdatascience.com/data-science-for-startups-data-pipelines-786f6746a59a>,
- Documentation for the pandas library: <https://pandas.pydata.org/pandas-docs/stable/>.

Часть V

Плагины и заключение

Эта часть познакомит читателей с концепцией плагинов PyCharm и расскажет о процессе загрузки плагинов и их добавления в среду PyCharm. Также будут подробно описаны наиболее популярные плагины и то, как они могут еще больше оптимизировать производительность программиста. Мы также коснемся важных тем, обсуждавшихся в предыдущих главах книги, и дадим всесторонний обзор наиболее популярных функций PyCharm.

Эта часть состоит из следующих глав:

- глава 15 «Больше возможностей с плагинами»,
- глава 16 «Ваши следующие шаги с PyCharm».

Больше возможностей с плагинами

Любую достойную IDE можно расширить с помощью архитектуры плагинов. Плагины поддерживают такие известные компании, как Microsoft, Eclipse и JetBrains, а также небольшие некоммерческие проекты IDE с открытым исходным кодом, такие как Geany и Ninja IDE. Даже vim, который настолько прост, насколько это возможно, учитывая отсутствие графического пользовательского интерфейса, имеет множество доступных плагинов, которые позволяют вам настроить свой опыт за пределами возможностей поставляемого продукта. В случае с IDE JetBrains архитектура плагинов лежит в основе всего, что они создают.

Учтите, что JetBrains начала с IntelliJ и на основе этого создала около дюжины специализированных IDE. Причина, по которой это стало возможным, заключается в том, что их архитектура основана на плагинах. PyCharm Professional включает в себя все основные функции двух других IDE, WebStorm и DataGrip, поскольку эти функции представляют собой просто наборы плагинов.

В этой главе мы собираемся обсудить некоторые важные плагины как от JetBrains, так и от сторонних разработчиков. К концу этой главы вы будете знать следующее:

- как использовать функции удаленной разработки JetBrains, которые позволяют вам работать на локальном ноутбуке, одновременно используя удаленную среду, такую как виртуальная машина, рабочая станция или сервер;
- как работать совместно с *Code With Me* от JetBrains;
- как работать с Docker с помощью плагина JetBrains Docker;
- как использовать инфраструктуру разработки в виде кода с помощью HashiCorp Vagrant в PyCharm;
- я также расскажу о некоторых небольших, качественных плагинах, включая темы, специализированные маркеры, дополнительные языки и, конечно же, формater кода Black.

ТЕХНИЧЕСКИЕ ТРЕБОВАНИЯ

Чтобы пройти эту главу, как и остальную часть книги, вам понадобится следующее:

- установленная и рабочая копия PyCharm. Установка была описана в главе 2;
- многие разделы этой главы требуют подключения к интернет-сервисам. Для некоторых демонстраций требуется высокоскоростное соединение или много терпения;
- для раздела об удаленной разработке вам понадобится удаленный компьютер, доступный через SSH и работающий под управлением операционной системы Linux. Это может быть физическая машина или виртуальная машина, работающая через облачного провайдера. Я рассказываю только о подключении по удаленному доступу. Я не буду вдаваться в подробности настройки компьютера или виртуальной машины Linux в качестве удаленного устройства;
- для раздела о Docker я буду использовать Docker Desktop. Инструкции по его установке для операционной системы вашего компьютера можно найти по адресу <https://www.docker.com/products/docker-desktop/>;
- пример исходного кода этой книги с GitHub. Мы рассмотрели клонирование кода в главе 2. Соответствующий код этой главы вы найдете на <https://github.com/PacktPublishing/Hands-On-Application-Development-with-PyCharm---Second-Edition/tree/main/chapter-15>.

ПЛАГИНЫ В КОМПЛЕКТЕ И JETBRAINS MARKETPLACE

В книге было несколько случаев, когда я указывал на полезные плагины. Я просто не мог дождаться, и плагины, которые я вам показал, были либо очень актуальными для конкретной главы игры, либо просто очень крутыми и захватывающими. Чтобы стать ответственным наставником по PyCharm, я возьму себя в руки, сдержу свой энтузиазм (насколько смогу) и сделаю вид, что вы сразу перешли к этой главе, которая может оказаться лучшей в книге. Или нет, вам решать.

Многие функции продуктов JetBrains IDE созданы в виде плагинов. Вы встретите два типа плагинов: созданные JetBrains и сторонние плагины, доступные в JetBrains Marketplace. Многие плагины, созданные JetBrains, включены в различные IDE, которые они создают, но некоторые необходимо добавить или активировать. Начнем с изучения панели **Plugins**, доступной в настройках IDE.

Окно плагинов

Есть несколько способов попасть в окно **Plugins**. Возможно, проще всего просто закрыть любой проект, открытый в PyCharm, и появится окно, с которого мы начали 15 глав назад, на рис. 15.1.

В верхней части окна **Plugins** вы найдете две важные вкладки: одна показывает установленные вами плагины (2), а другая ведет на Marketplace, где вы можете купить и установить новые (1). Многие плагины в Marketplace бесплатны, но многие из лучших – нет.

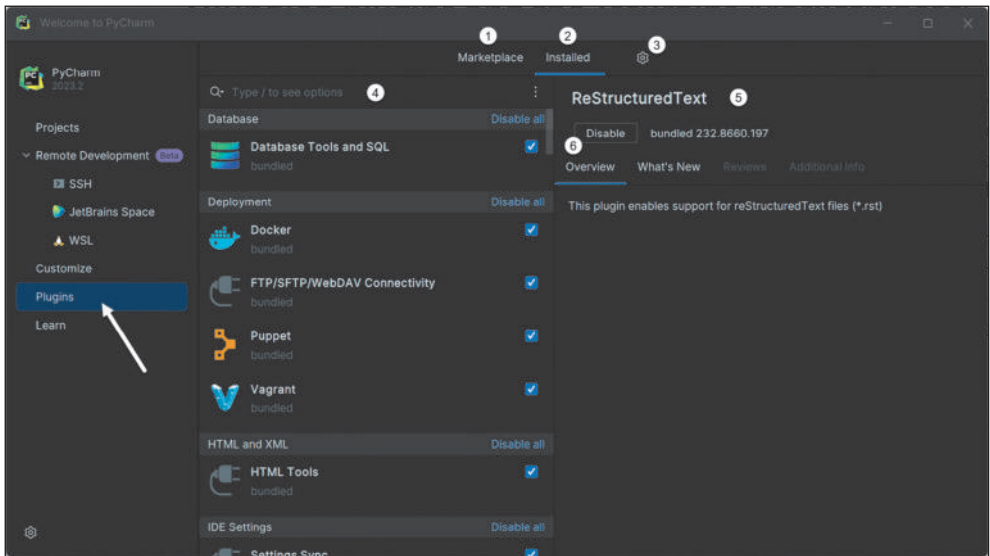


Рис. 15.1. Окно Plugins содержит все инструменты, необходимые для управления плагинами в PyCharm

Значок шестеренки (3) позволяет настроить дополнительные репозитории плагинов. Мне это никогда не было нужно, но оно есть, если понадобится. Вы также можете установить плагин из локальной папки, используя соответствующее меню под шестеренкой.

Поле поиска (4) позволяет найти плагины в списке. Выбирая плагин, вы обычно получаете некоторую информацию о нем справа (5). Плагины можно включать и отключать (6), поэтому, если ваша IDE после установки плагина ведет себя нестабильно, можете временно отключить ее или полностью удалить.

Связанные плагины

Если вы изучите список на рис. 15.2, то обнаружите, что большинство плагинов на этом экране включены в состав PyCharm. Это плагины, созданные JetBrains и включенные в обычную установку PyCharm. Неплохая идея просмотреть этот список и отключить плагины, которые вы вряд ли будете использовать. Например, я использую **Terraform** и **Ansible** для своих DevOps и **Infrastructure as Code (IaC)**. Я вряд ли буду использовать Puppet, поэтому мне следует отключить этот плагин.

При этом вы можете обнаружить, что PyCharm предупреждает вас о зависимых плагинах. Например, есть плагин для Less, который представляет собой метафреймворк CSS. Я им не пользуюсь, поэтому заманчиво его отключить. Однако плагин JSX Styled Components, который я использую в проектах React, зависит от Less (см. рис. 15.2).

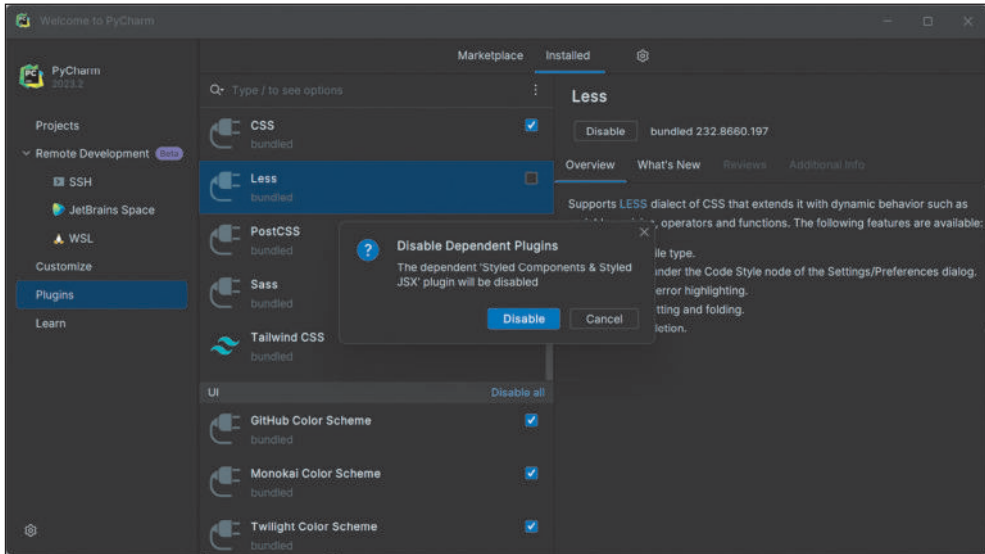


Рис. 15.2. Меня предупреждают, когда я пытаюсь отключить плагин, который необходим другому плагину

В этом случае лучше оставить параметр **Less** включенным.

JetBrains Marketplace

Если вам нужен розничный IDE-шоппинг, можете перейти на вкладку **Marketplace** и отправиться за покупками. Некоторые плагины бесплатны, некоторые нет. С этого экрана вы можете легко искать и устанавливать плагины из Marketplace. Если вы купите один из них, вы, скорее всего, будете перенаправлены на веб-сайт JetBrains Marketplace, где сможете контролировать свою лицензию.

Создание собственных плагинов

Любой может создать плагин IDE, используя язык программирования Java и версии IntelliJ Community или Ultimate. Я не буду рассказывать, как создать плагин, но покажу, с чего начать. На рис. 15.3 у меня открыт IntelliJ Community Edition в диалоговом окне нового проекта. Учитывая, что у вас большой опыт создания проектов в PyCharm, скажу вам, что все IDE идентичны. IntelliJ – это оригинальный флагманский продукт IDE от JetBrains.

Помимо шаблона проекта, имеется также гиперссылка на руководство по созданию проекта плагина.

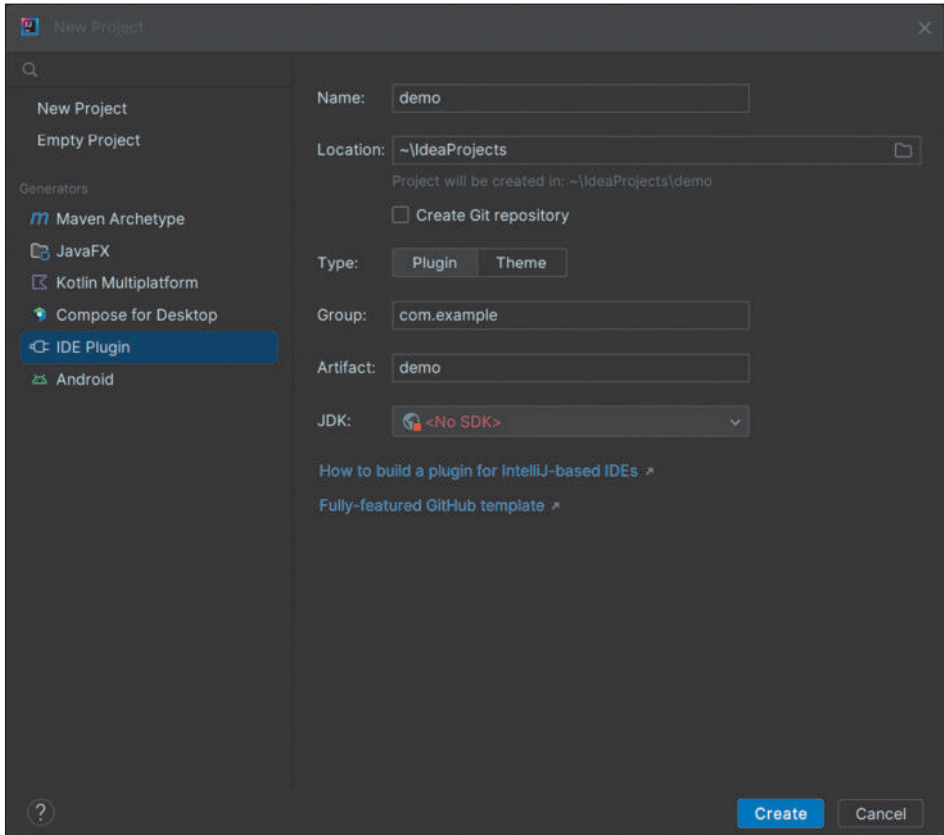


Рис. 15.3. В IntelliJ Community Edition имеется шаблон проекта для создания собственных плагинов IDE

Плагины, необходимые для ваших проектов

Точно так же, как у нас есть текстовый файл `requirements.txt`, в котором перечислены имена и версии пакетов, необходимых для данного проекта Python, мы можем указать определенные плагины в качестве требуемых для проекта PyCharm. Это гарантирует, что другие разработчики будут иметь доступ к любым функциям плагина, которые, по вашему мнению, имеют решающее значение для их успеха в совместной работе над вашим проектом. На рис. 15.4 показано, как установить необходимые плагины для вашего проекта.

Сначала зайдите в настройки вашего проекта. В **Build, Execution, Deployment** найдите пункт меню **Required Plugins** (1) и кликните его. Затем нажмите значок + (2) и выберите необходимые плагины в появившемся диалоговом окне (3). Нажмите **OK**.

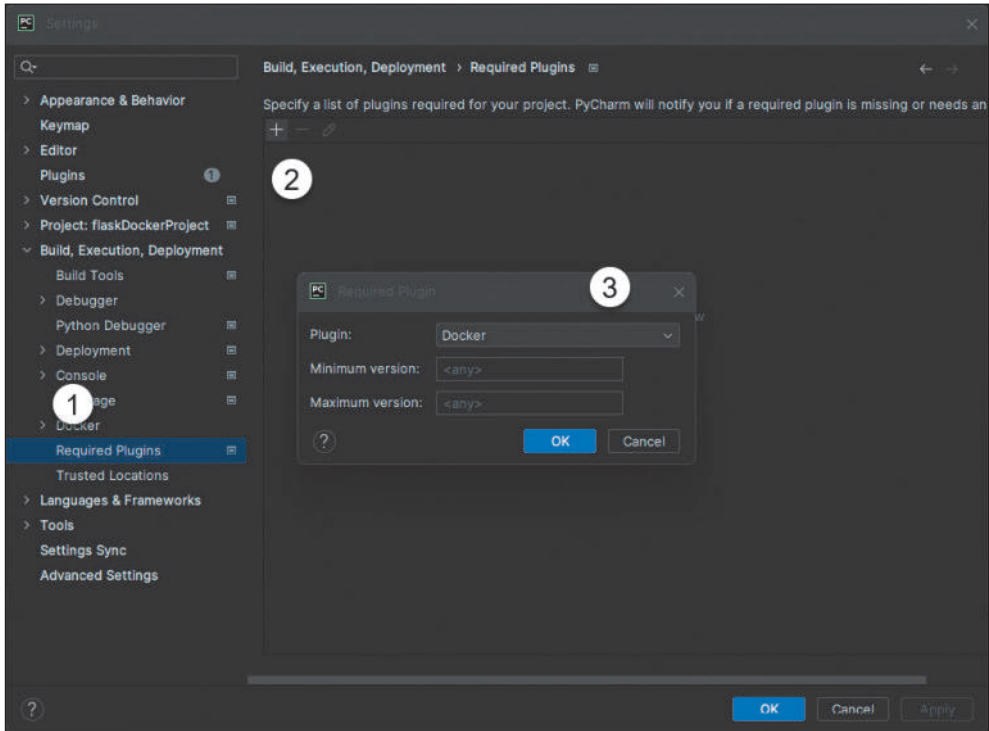


Рис. 15.4. Действия по определению требуемых плагинов

Разные полезные плагины

Я хочу посвятить большую часть этой главы некоторым очень эффективным плагинам со сложными функциями. Однако, прежде чем я это сделаю, давайте разберемся с простыми вещами. Я расскажу о небольших интересных плагинах, о которых вы, скорее всего, вспомните, когда будете искать плагины IDE. Некоторые из них мы видели в других главах.

- Плагин CSV был рассмотрен в главе 13.
- Плагин Markdown. Раньше его не было в комплекте, но теперь он есть, и он определенно то, что нужно.
- Плагин режима ячеек, который мы также рассмотрели в главе 13.

Давайте поговорим еще о нескольких, которые я сгруппирую в зависимости от того, чем они занимаются.

Плагины Theme

Вы проводите от 8 до 16 ч в день, глядя на IDE. С таким же успехом вы могли бы потратить некоторое время на то, чтобы это выглядело круто. Доступно множество цветowych «тем». На рис. 15.5 показан мой нынешний фаворит: темно-фиолетовая тема.

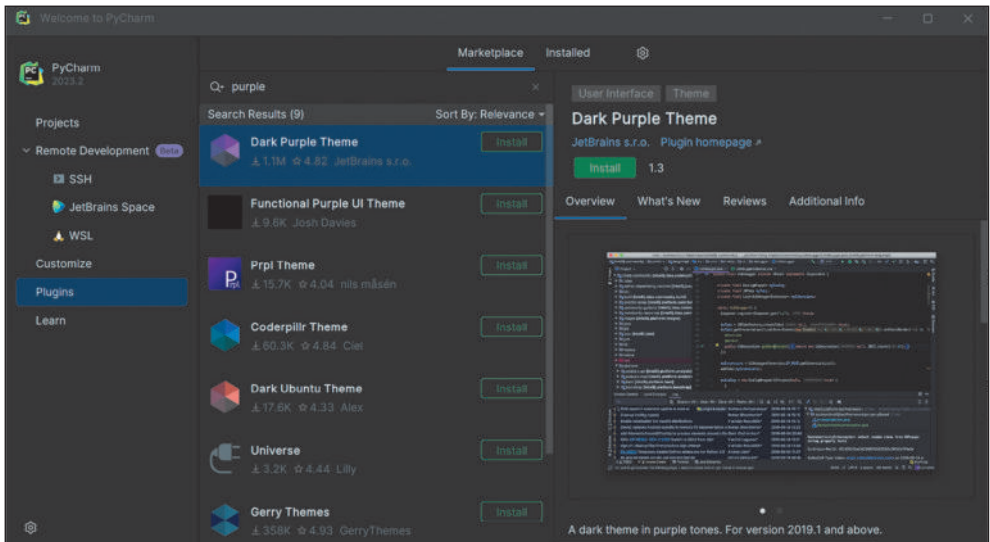


Рис. 15.5. Темно-фиолетовая тема – одна из моих любимых, поскольку она отличается от стандартной темной темы VS Code

Когда люди проходят мимо и видят такое IDE, они сразу поймут, что вы не один из тех рабов VS Code с их вездесущей готовой темной темой Microsoft. Вы могли бы использовать стандартную тему Darcula, поставляемую с PyCharm, но для такого выпускника Университета Оклахомы¹, как я, в этой теме слишком много оранжевого цвета. Выберите что-нибудь, что вы не видите каждый день, и вскоре все на кубической ферме² узнают, что вы настоящий профессионал!

Мне также нравится пользовательский интерфейс Material Theme в приятной подсветке синим цветом. Этот плагин поставляется в облегченной (бесплатной) и платной версиях. Если вы посмотрите вокруг, то найдете цветовую тему, которая вам подходит, поскольку их очень много. Самый простой способ найти нового фаворита – ввести тег в поле поиска, показанное на рис. 15.6.

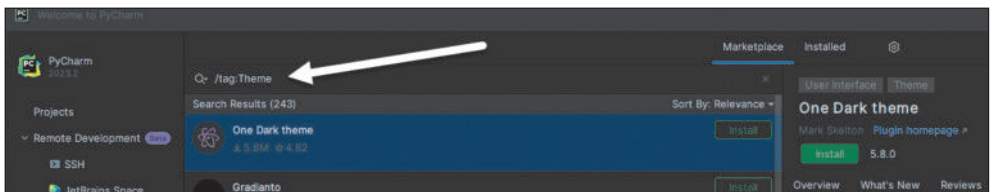


Рис. 15.6. Вы можете фильтровать поиск плагинов, используя такие теги, как “Theme”

¹ «Выпускник Университета Оклахомы» – в США – уничижительный и даже оскорбительный эпитет, применяемый для обозначения неразумных, необразованных, неграмотных, нечистоплотных, безуспешных и неряшливых людей. – *Прим. ред.*

² «Кубическая ферма» – жаргонный термин, обозначающий офисную среду, где сотрудники работают в небольших кабинках или офисах с ограниченным пространством и малой приватностью. Это место, где умирает творчество, а мечты разбиваются под тяжестью бюрократии и однообразия. Это место, где преследуются амбиции и прославляется посредственность. – *Прим. ред.*

Я рекомендую вам попробовать это, потому что по какой-то причине оставление поля поиска пустым не дает очень длинного списка опций.

Плагины для обработки файлов

Этот класс плагинов обеспечивает цветное выделение некоторых файлов, не связанных с кодом, которые мы используем каждый день.

Плагин `.ignore` обеспечивает удобное форматирование для любых распространенных файлов игнорирования, таких как `.gitignore` и `.dockerignore`.

Плагины интеграции инструментов

Пол Эверитт, один из разработчиков, поддерживающих PyCharm, часто напоминает нам, что I в IDE означает *Integrated*. Вам никогда не придется покидать PyCharm по какой-либо причине. Помимо написания кода, разработчикам необходимо взаимодействовать с системами отслеживания проблем, системами контроля версий, средами непрерывной сборки и, возможно, множеством инфраструктурных инструментов. Неудивительно, что существуют плагины, помогающие с этими типами интеграции. На рис. 15.7 показан тег, используемый для поиска этих плагинов.

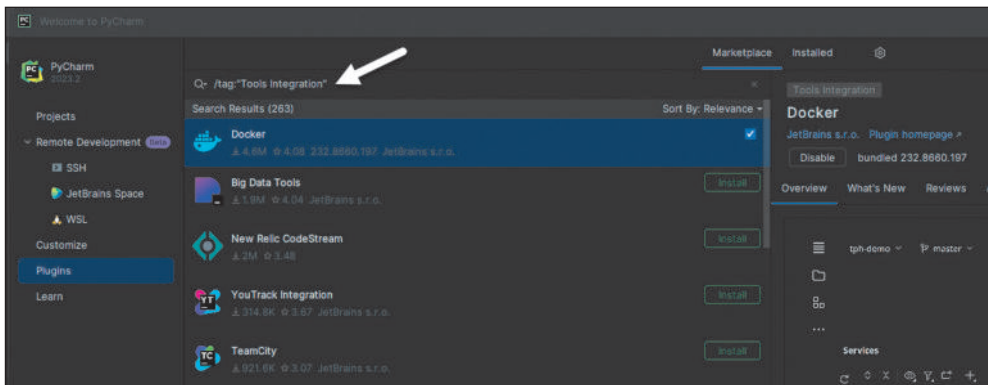


Рис. 15.7. Используйте тег Tools Integration, чтобы найти интеграцию с вашей любимой инфраструктурой и инструментами DevOps

Раньше я был активным пользователем TeamCity. Если вы используете TeamCity или любой другой командный инструмент JetBrains, вы найдете плагины, которые позволяют отслеживать ваши сборки, решать проблемы и многое другое прямо из вашей IDE. Для справки, Team City по-прежнему остается моим любимым сервером **непрерывной интеграции (CI)**.

Плагины продуктивности

Производительность разработки – это основная причина использования IDE в первую очередь. Под этим тегом вы найдете множество дополнительных функций и интеграций. На рис. 15.8 показан этот тег в действии.

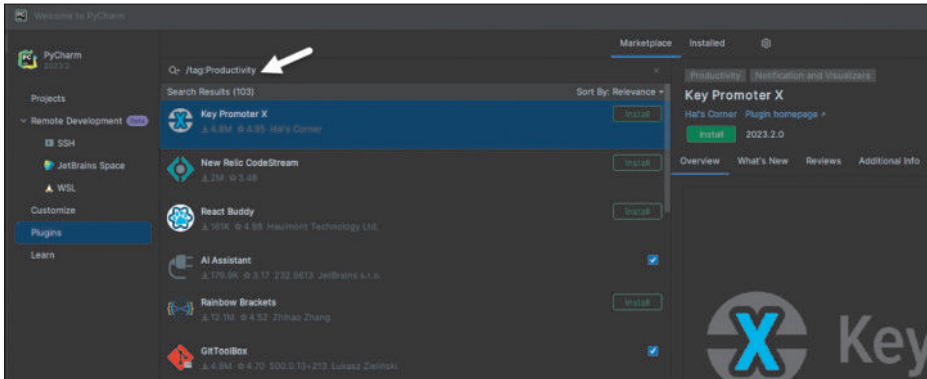


Рис. 15.8. Используйте тег продуктивности, чтобы найти инструменты, которые сделают вашу работу более продуктивной

Мой любимый плагин DevOps – GitToolbox. Взгляните на один из моих проектов, открытых в PyCharm с включенным GitToolbox.

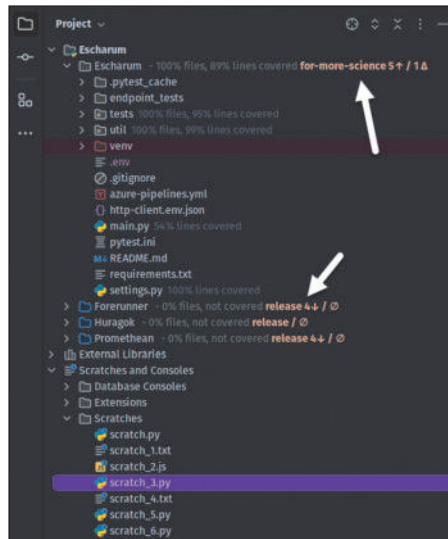


Рис. 15.9. Плагин GitToolbox предоставляет много информации о репозитории вашего проекта

С первого взгляда я могу увидеть ветку, которую сейчас использую, а также количество коммитов впереди или позади относительно моего удаленного доступа. Плагин поможет вам очистить старые ветки, которые вы давно не использовали, выполнять регулярную выборку через определенные промежутки времени и проверять сообщения о фиксации. Моя любимая особенность – это то, что вы видите информацию о `git blame` для каждой строки. Я не из тех, кто жалуется, но мне, как ведущему разработчику, очень полезно видеть, кто из моих коллег работал над тем или иным файлом. Это определенно помогает проверкам кода идти немного быстрее.

Плагины завершения

ИИ стал революционной неотъемлемой частью разработки программного обеспечения. При использовании тега `/tag: Completion` будут перечислены десятки традиционных плагинов автодополнения кода, а также плагинов на основе искусственного интеллекта, как показано на рис. 15.10.

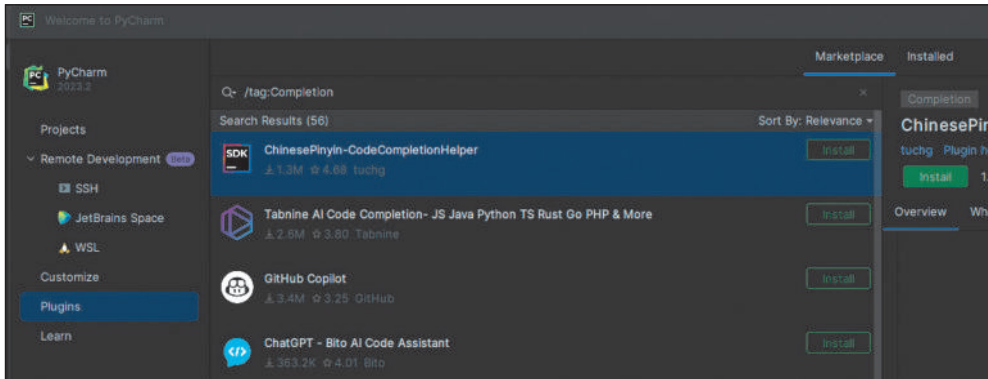


Рис. 15.10. Плагины завершения, включая наиболее популярные реализации ИИ

На данный момент GitHub Copilot и любой из десятков плагинов ChatGPT являются очень популярными плагинами автодополнения. Прежде чем платить за них деньги, ознакомьтесь с моим обсуждением плагина **JetBrains AI Assistant** в главе 17. В настоящее время он находится на стадии закрытого бета-тестирования, поэтому я не могу включить его сюда, но он действительно хорош и более тесно интегрирован, чем любой из этих плагинов.

Кодируйте со мной (и вы никогда больше не будете одиноким)

Парное программирование – популярная практика, введенная много лет назад с помощью гибкой методологии, известной как **экстремальное программирование (XP)**. Хотя XP не так известен, как Scrum или Kanban, он имеет многие из тех же практик и целей, но с парным программированием в качестве обязательного компонента. В XP два программиста сидят перед одним компьютером и вместе пишут код. Один человек печатает, в то время как другой наблюдает, тренирует, ищет информацию и, как правило, вносит свой вклад, помогая. Через определенные промежутки времени программисты меняются местами. Приверженцы этой практики доверяют ей, утверждая, что она устраняет необходимость в проверке кода и в целом приводит к улучшению кода. Недоброжелатели, в число которых входят все ныне живущие менеджеры, не являющиеся техническими специалистами, говорят, что нет смысла иметь двух дорогостоящих специализированных разработчиков, выполняющих вдвое меньше работы, чем могли бы.

Нравится вам это или нет, но вы должны признать, что прямое сотрудничество имеет огромную пользу. Реальная проблема заключается в том, что в мире после Covid, где очень много разработчиков работают удаленно, идея двух разработчиков за одним экраном больше не осуществима, по крайней мере, до сих пор.

У JetBrains есть интегрированный инструмент под названием *Code With Me*. Этот инструмент позволяет нескольким разработчикам присоединиться к общему онлайн-сеансу, где все одновременно работают над проектом на одном компьютере. Этот плагин поставляется в комплекте со всеми IDE JetBrains, включая PyCharm. Вы можете активировать это, кликнув значок, показанный на рис. 15.11.

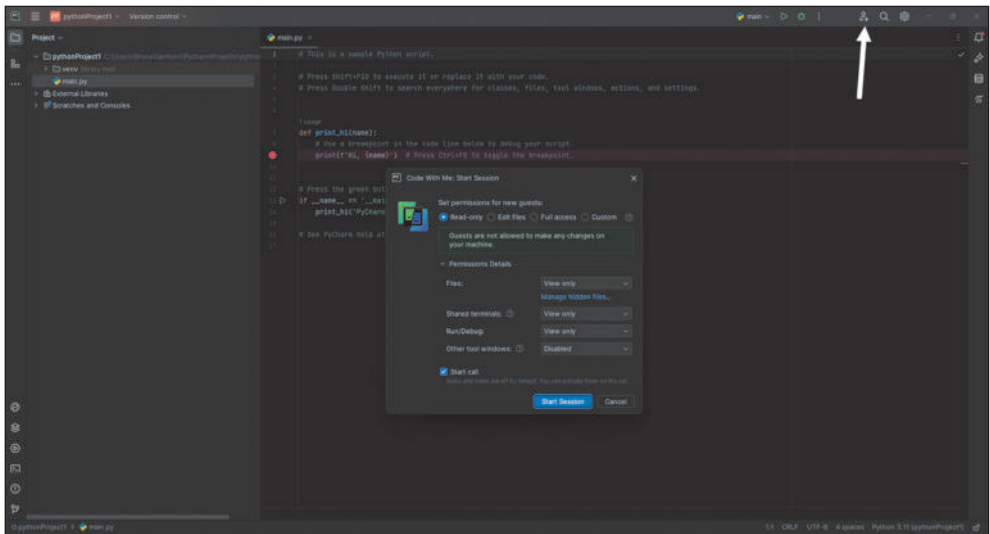


Рис. 15.11. Чтобы начать сеанс Code With Me, кликните значок, указанный стрелкой

Можете установить допуски для своих гостей. Помните, вы предоставляете доступ к своему компьютеру! Вам следует участвовать в сеансах только с людьми, которым вы доверяете, а также отправлять ссылки на сеансы только тем людям, которым вы доверяете.

После того как вы нажмете **Start Session**, PyCharm создаст гиперссылку, которой вы сможете поделиться по своему усмотрению. Можете отправить ее по электронной почте, но я чаще всего вижу, что ее отправляют другим разработчикам через Slack или аналогичные службы частного чата. Любой, у кого есть ссылка, может нажать ее, чтобы присоединиться к сеансу. Помимо совместного использования IDE, у вас также есть обычные инструменты для проведения конференций. Участники могут видеть и слышать друг друга, как и следует ожидать от таких продуктов, как Zoom или WebEx. На рис. 15.12 показано, как я ни с кем не совещаюсь, но это дает вам представление о том, чего ожидать.

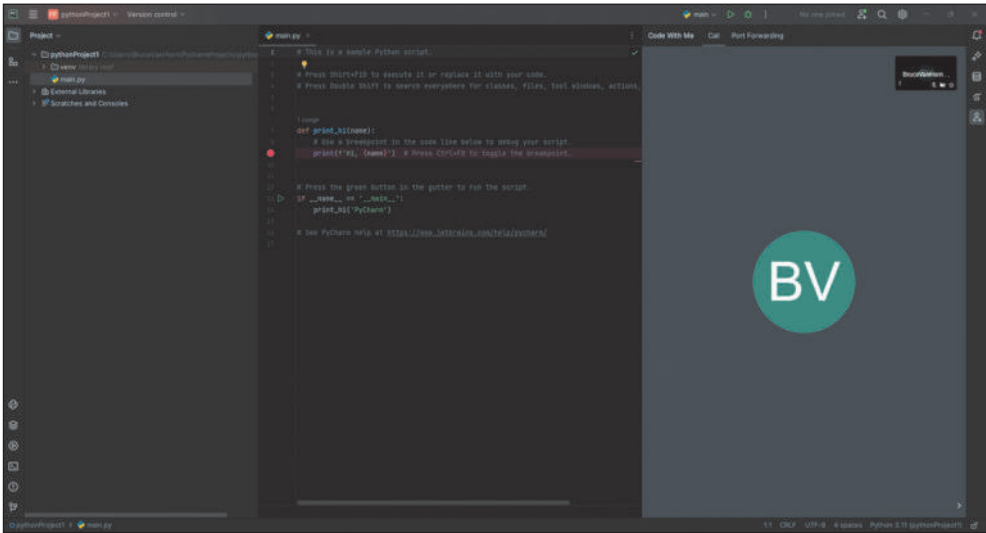


Рис. 15.12. Выполняется сеанс Code With Me. К сожалению, поскольку я провожу все свое время за написанием книг, у меня нет друзей, которым можно позвонить, так что это только я

Эта функция полезна во многих скриптах, выходящих за рамки очевидного варианта использования парного программирования. Вы можете использовать это для каждого класса или провести собеседование по программированию, используя что-то более реалистичное, чем Google Doc. Если вы собираетесь часто использовать эту услугу, понадобится подписка, поскольку существуют ограничения на продолжительность сеанса и количество участников, которых можно пригласить. Вы можете узнать больше по адресу <https://www.jetbrains.com/code-with-me/buy/#commercial>.

Удаленная разработка

В моем офисе есть большая, мощная рабочая станция HP с корпусом Full Tower весом 60 фунтов, двумя процессорами Intel Xeon (всего 48 ядер), 128 Гб памяти, графическим процессором NVidia профессионального уровня и графическим ускорителем 24 GB GPU. Она даже имеет **последовательно подключаемый системный интерфейс малого компьютера (SAS)** для работы с хранилищем корпоративного уровня. Невозможно иметь такую мощь на ноутбуке, и бывают дни, когда я выжимаю из этой станции все.

Наша индустрия любит суп из алфавита!

Я просто собираюсь прояснить это для всех инженеров, которые пытаются понять, как трехбуквенная аббревиатура SAS отображает шесть слов. Это потому, что аббревиатура SAS содержит одну букву, которая сама по себе обозначает другую аббревиатуру. Это последовательно подключаемый SCSI (SAS). SCSI означает системный интерфейс малого компьютера. Диски SAS представляют собой

вращающиеся диски, которые способны развивать более высокие скорости вращения, чем стандартные диски с последовательным интерфейсом Advanced Technology Attachment (SATA), но они все равно далеко не так быстры, как твердотельные накопители (SSD) или экспресс-накопители с энергонезависимой памятью (NVME) с памятью NOT-AND или NAND. Однако диски SAS предназначены для использования в резервных массивах недорогих дисков (RAID), что делает их полезными для недорогого и отказоустойчивого хранения очень больших файлов.

Все эти аббревиатуры и жаргонизмы – часть общеотраслевого заговора, призванного заставить ваших родственников, понятия не имеющих, чем вы занимаетесь, одновременно гордиться и чувствовать себя некомфортно на семейных посиделках и праздниках.

Но, как и любой другой разработчик, я хотел бы иметь возможность работать по удаленке. Очевидно, я не хочу таскать с собой свою 60-фунтовую рабочую станцию. Даже если бы это было так, нашей команде безопасности не понравится, что наш исходный код покидает здание. Да, я мог бы использовать удаленную технологию, такую как VNC Viewer или Microsoft Remote Desktop, но они могут работать медленно и не иметь таких удобств, как поддержка четырех мониторов, к которым я уже привык.

Я также мог бы просто работать по SSH, используя **neovim**. Но ведь мы с вами уже это проходили, верно? Если вы считаете, что это хорошая идея, вернитесь к главе 1, где я говорил, что это плохая идея. Итак, что мне делать?

К счастью, JetBrains меня спас. Используя удаленные плагины, которые предварительно установлены в PyCharm Professional, я могу запускать свою IDE на ПК дома, но работать на своей рабочей станции в офисе. Это делается с помощью **защищенной оболочки (SSH)**, которая представляет собой зашифрованный протокол связи, обычно используемый для администрирования серверов на базе Linux и Unix.

Проще говоря, вы можете настроить PyCharm на использование виртуальной среды (venv), которая установлена и настроена на вашем удаленном компьютере вместе с кодом на удаленном жестком диске. Все работает на удаленном компьютере, кроме самой IDE. Ваши конфигурации запуска будут выполнять ваш код удаленно. При отладке дебаггер запускается на удаленном компьютере, а не на локальном. Это довольно бесшовный процесс, но у него есть пара недостатков.

Во-первых, вы должны иметь определенные знания о сетях, чтобы настроить проект для этого. Вам необходим IP-адрес удаленного компьютера, а также действительные учетные данные доступа, которые позволят вам получить доступ к удаленному компьютеру. В моем случае мне также нужен работающий клиент **виртуальной частной сети (VPN)** для доступа к сети в моем офисе, и это является внешним по отношению к PyCharm. Хотя VPN технически не является обязательным требованием, в большинстве скриптов оно должно быть таковым.

Второй компромисс – это производительность. PyCharm в иной день не очень быстр. Нам нужно дождаться завершения индексации, прежде чем мы сможем использовать любую из тех замечательных функций, о которых гово-

рили на протяжении всей книги. Нередко возникают ошибки нехватки памяти для PyCharm, что требует от вас увеличения объема памяти, которую PyCharm разрешено использовать. Обновление или установка плагинов требует перезапуска IDE, и даже начальный экран загрузки висит дольше, чем многим из нас хотелось бы. В общем, иногда PyCharm может быть несколько неуклюжим, но мы миримся с этим, потому что он облегчает наш общий рабочий процесс.

Я склонен думать о PyCharm как о дизельном грузовике. Такой грузовик не сможет выиграть спринтерскую гонку на четверть мили против Porsche или Tesla Model 3. Будучи родом из сельской Оклахомы, я могу вам сказать, что там, откуда я родом, ни у кого нет спортивных автомобилей, потому что наши дизельные грузовики – это больше, чем просто транспорт, они являются частью нашей жизни. Я не могу использовать Теслу, чтобы отбуксировать коров на аукцион, и не могу использовать Порше, чтобы возить сено на задний двор, чтобы кормить лошадей. Точно так же мы используем тяжелые IDE, потому что готовы обменять «быстроту» минимального редактора, такого как VS Code, на очень большую ценность, предоставляемую индексацией PyCharm. Возможность находить варианты использования, переходить непосредственно к определениям функций и иметь такой тесно интегрированный набор инструментов, охватывающий все, от редактирования до управления базами данных, стоит небольшого отставания в скорости.

С учетом вышесказанного это отставание может стать более очевидным, когда вы работаете над удаленным проектом. Вы можете ожидать, что время индексирования будет больше, поскольку индексирование происходит удаленно, а затем передается обратно вам. Задержка в сети и VPN всегда замедляют работу, и это не вина PyCharm. Мы миримся с этим, потому что вы можете работать дома в пижаме с включенной музыкой рядом с собственной кухней и закусками, и вы можете ограничить свое взаимодействие с другими людьми, что обычно только замедляет работу таких людей, как мы. Это стоит того. Теперь, когда ваши ожидания определены, давайте посмотрим, как заставить все это работать.

Настройка удаленной разработки в PyCharm

Я горжусь тем, что во всех моих книгах и курсах все получается максимально реалистично. Например, в главе 7 у меня возникли проблемы с функцией PyCharm, которая должна автоматически загружать Node.js. Я этого не скрывал, потому что это реально произошло со мной, а значит, может случиться и с вами. Имеет смысл научиться решать подобные проблемы.

Возможно, вы уже заметили, что снимки экрана в этой главе отличаются от снимков в предыдущих главах. Я пишу эту главу в тот день, когда работаю дома и использую компьютер с Windows 11, который до сегодняшнего дня не был настроен для удаленной работы. Поскольку это мой домашний компьютер, снимки экрана могут немного отличаться от снимков из предыдущих глав, особенно в отношении темного режима. Я отключил темный режим для большей части книги, потому что светлый режим легче читать в печатной книге, и я экономлю деньги издателя, которые он тратит на красители. Но не сегодня. Потому что я хочу, чтобы все было как есть.

Первое, что я сделаю, – это установлю и открою PyCharm. Очевидно, мы уже рассмотрели это в главе 2, поэтому нет необходимости повторять это снова. Экран приветствия предоставляет возможность открытия удаленного проекта, как показано на рис. 15.13.

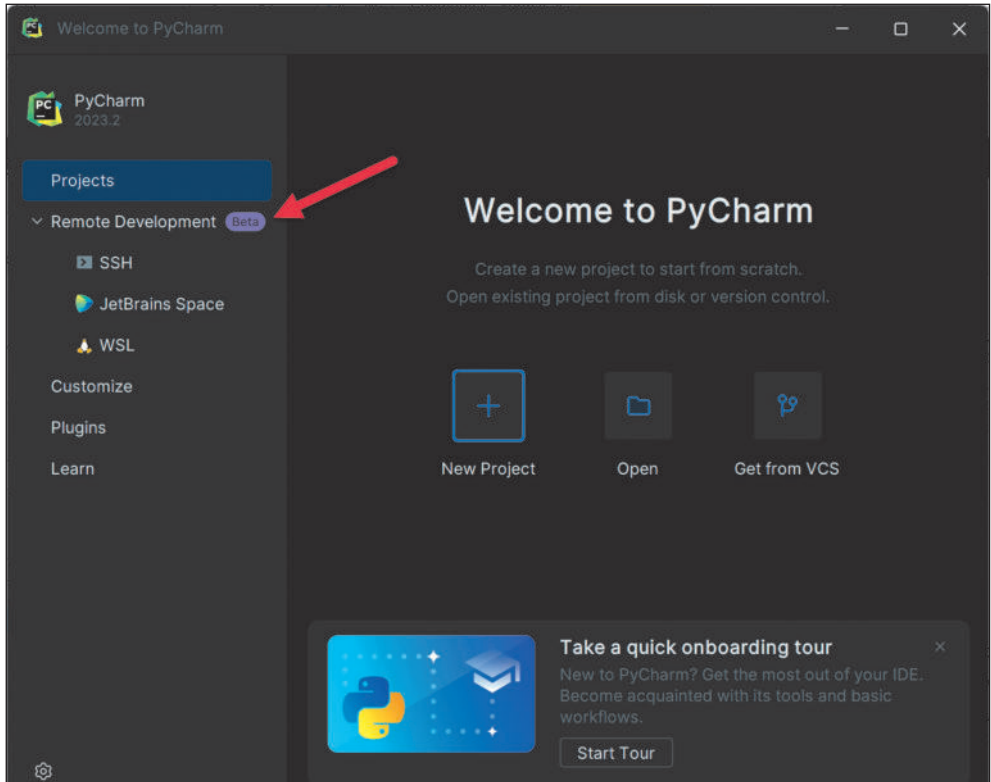


Рис. 15.13. Параметры удаленной разработки доступны в правой части окна приветствия

Здесь есть несколько вариантов, включая SSH, который я и буду использовать. JetBrains Space – это отдельный продукт, я им не владею, поэтому не могу о нем рассказать. **WSL** или **Windows Subsystem** для **Linux**, скорее всего, появится только в том случае, если вы используете Windows. WSL позволяет запускать виртуальную машину Linux непосредственно в Windows без использования очевидного гипервизора, такого как VMware или Virtual Box. Я говорю «очевидного», потому что, конечно, здесь все равно задействован гипервизор – это Microsoft **HyperV**, встроенный в профессиональные версии продуктов Windows 11 и Windows Server. Windows 11 упрощает эту задачу, позволяя вам выбрать одну или несколько установок Linux с помощью программы Windows Terminal, которая появилась в Windows 11.

Поскольку WSL технически является виртуальной машиной, это считается удаленной разработкой, даже если вы работаете на вашем локальном компьютере. На данный момент я собираюсь сосредоточиться на реальной удаленной разработке, получая доступ к моей рабочей станции, которая в настоящее время на-

ходится примерно в 12 милях от меня, через SSH. Чтобы внести ясность: на моем локальном компьютере установлена Windows 11, а на моей рабочей станции – **Pop_OS! 22**, которая является вариантом Ubuntu Linux. Вы видели это на скриншотах на протяжении всей книги. Это добавляет волшебства! Я могу использовать операционную систему потребительского уровня, такую как Windows или macOS, для подключения к рабочей лошадке Linux профессионального уровня.

Прежде чем вы зайдете в Твиттер (теперь он называется X), чтобы раскритиковать меня за неуважение к вашей любимой ОС, поймите, что это единственно возможная комбинация. Вам необходимо подключиться к удаленному Linux, поскольку Windows и Mac еще не поддерживаются. Я не знаю, в чем проблема с macOS, поскольку она поставляется с предустановленным SSH-сервером. Этот рабочий процесс имеет смысл для случая, когда у вас есть Mac Pro на работе и что-то еще дома. Следующие экраны настройки указывают на то, что эта функция предназначена для будущих выпусков.

На рис. 15.14 показано, что происходит, когда мы выбираем опцию SSH, показанную на рис. 15.13. Нам предоставляется экран, который позволяет создать новый проект на нашем удаленном ресурсе, но только после того, как мы сначала настроим наше соединение. Это будет похоже на создание SSH-соединения для публикации наших веб-проектов, о котором мы говорили в главе 7.

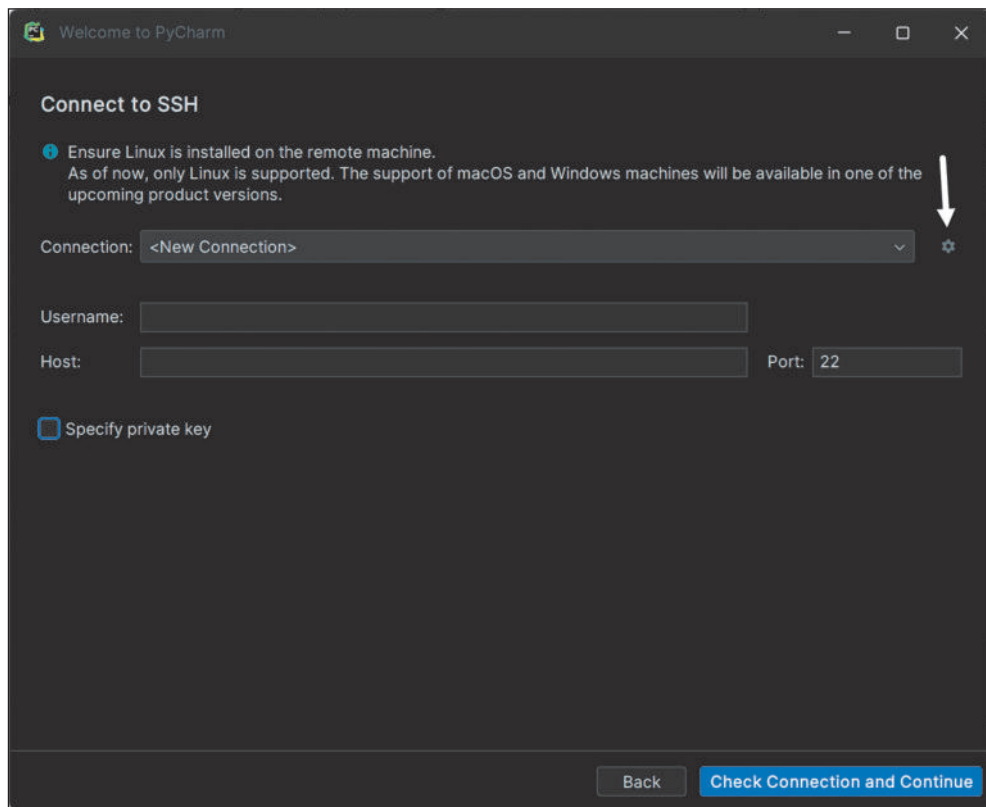


Рис. 15.14. Диалоговое окно настройки удаленной работы можно улучшить, только разместив поверх него другое диалоговое окно конфигурации

Прежде чем вы сможете что-либо сделать, необходимо настроить SSH-соединение, кликнув значок шестеренки, указанный стрелкой на рис. 15.14. Это подводит нас к еще одному диалоговому окну, показанному на рис. 15.5.

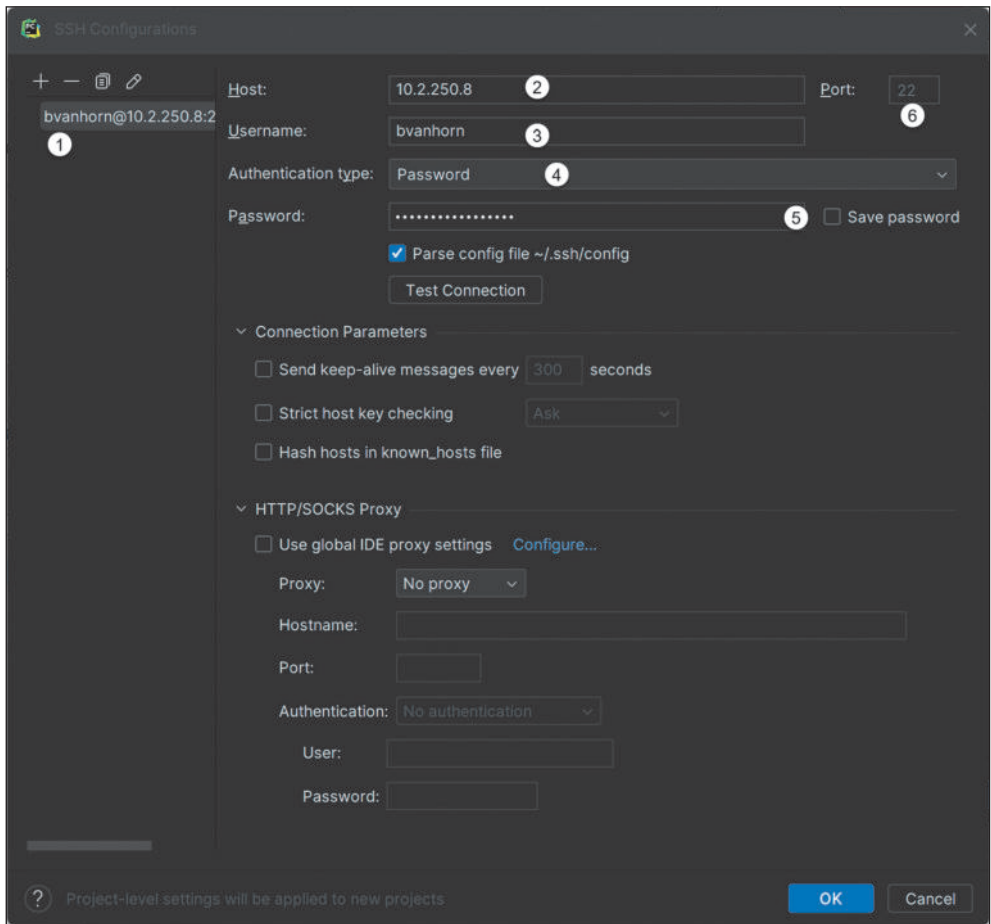


Рис. 15.15. Здесь я создаю конфигурацию SSH, которая будет использоваться в моей удаленной настройке

Я пошел дальше и нажал кнопку +, указанную в позиции 1, что открыло мне остальную часть формы, которую я заполняю. Нам нужно указать IP-адрес нашего хоста или **полное доменное имя (FQDN)**, если оно у вас есть. Это редкость, вы, скорее всего, будете использовать IP-адрес. Адрес на рисунке является частным IP-адресом, поэтому не пытайтесь получить к нему доступ. Это вполне реально, поскольку ваш удаленный IP-адрес, вероятно, также будет частным в удаленной сети, защищенной VPN. Я ввел свое имя пользователя (3) и для аутентификации использую пароль (4). Введя свой пароль, я могу сохранить его (5). Порт по умолчанию для SSH – 22, поэтому я оставил значение по умолчанию (6).

Если вы вернетесь к первой позиции (1), то сможете кликнуть значок карандаша и внести другое имя, если вам не нравится эстетика имени пользователя и IP-адреса. Я переименовываю свой компьютер на рис. 15.16 на более подходящее имя, учитывая его аппаратные характеристики.

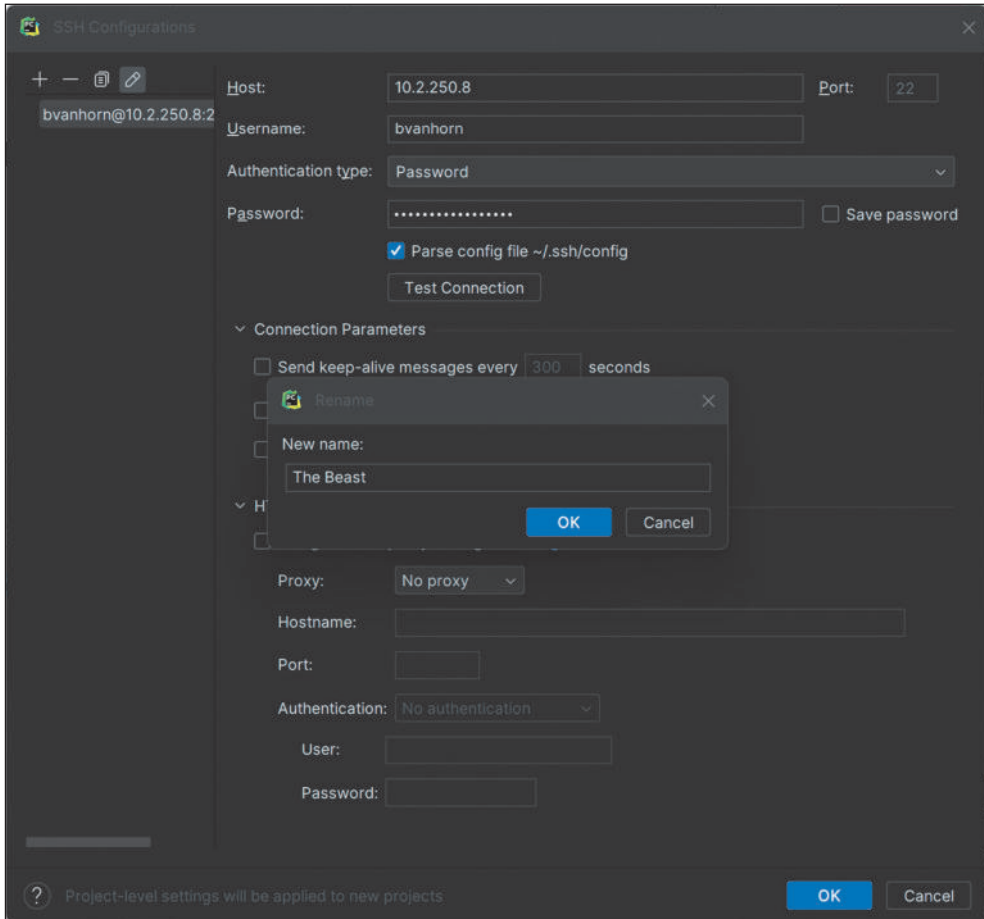


Рис. 15.16. Beast (Зверь) – это имя моей удаленной рабочей станции

Остальные параметры для меня не важны, поскольку мне не нужно использовать прокси-сервер, чтобы добраться до своей рабочей станции. Я пошел дальше и развернул все закрытые параметры на рис. 15.15, чтобы вы могли видеть любые дополнительные настройки, которые могут вам понадобиться.

Нажмите **ОК**, чтобы вернуться к предыдущему диалоговому окну. Нужно будет выбрать только что созданное соединение из раскрывающегося списка. Для меня переименование не прижилось, поэтому я вернулся к имени пользователя и IP, как показано на рис. 15.17.

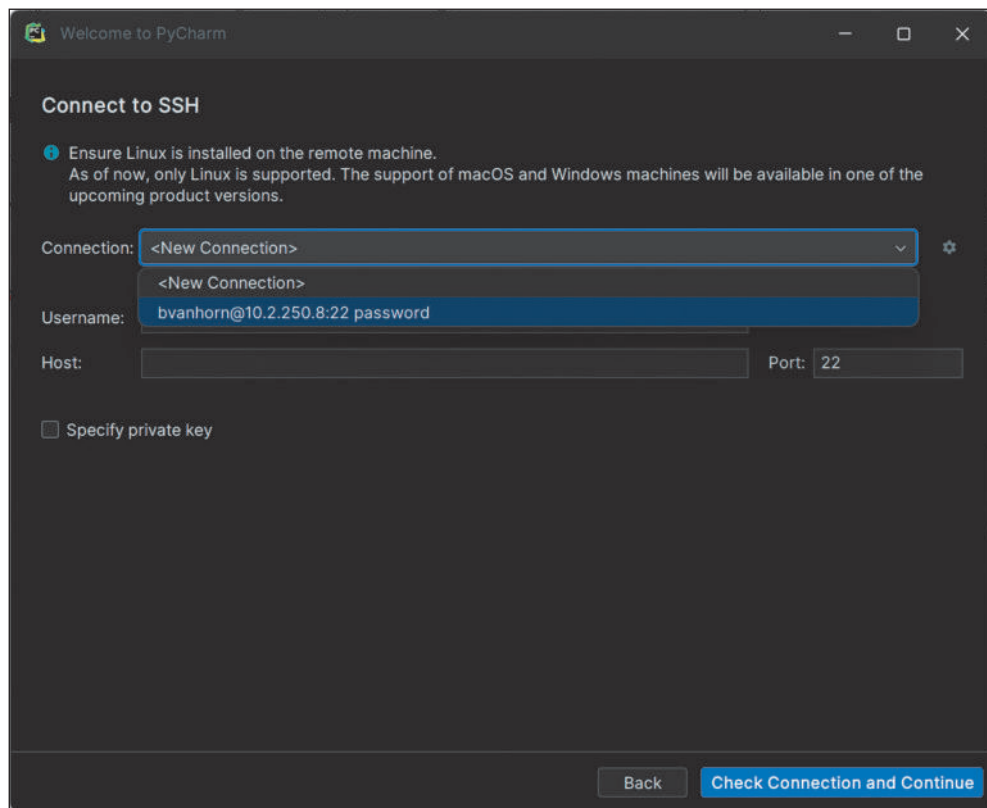


Рис. 15.17. Создав соединение, вам необходимо выбрать его здесь и поля имени пользователя и хоста будут заполнены за вас

Обратите внимание, что здесь у вас есть возможность отказаться от пароля и использовать закрытый ключ. Использование ключей SSH потенциально ограничивает необходимость в пароле, но вам все равно может быть предложено разблокировать ключ. В любом случае PyCharm может запомнить ваш пароль или ключевую фразу SSH, если вы захотите. Если вы работаете в компании, заботящейся о безопасности, можете проверить свои рекомендации, чтобы понять, где приложению считается целесообразным хранить ваши пароли и ключевые фразы.

Нажмите кнопку с надписью **Check Connection and Continue**. Предполагая, что вы получили доступ, PyCharm копирует некоторое программное обеспечение и настройки на хост, а затем представляет вам исходное диалоговое окно, как показано на рис. 15.18.

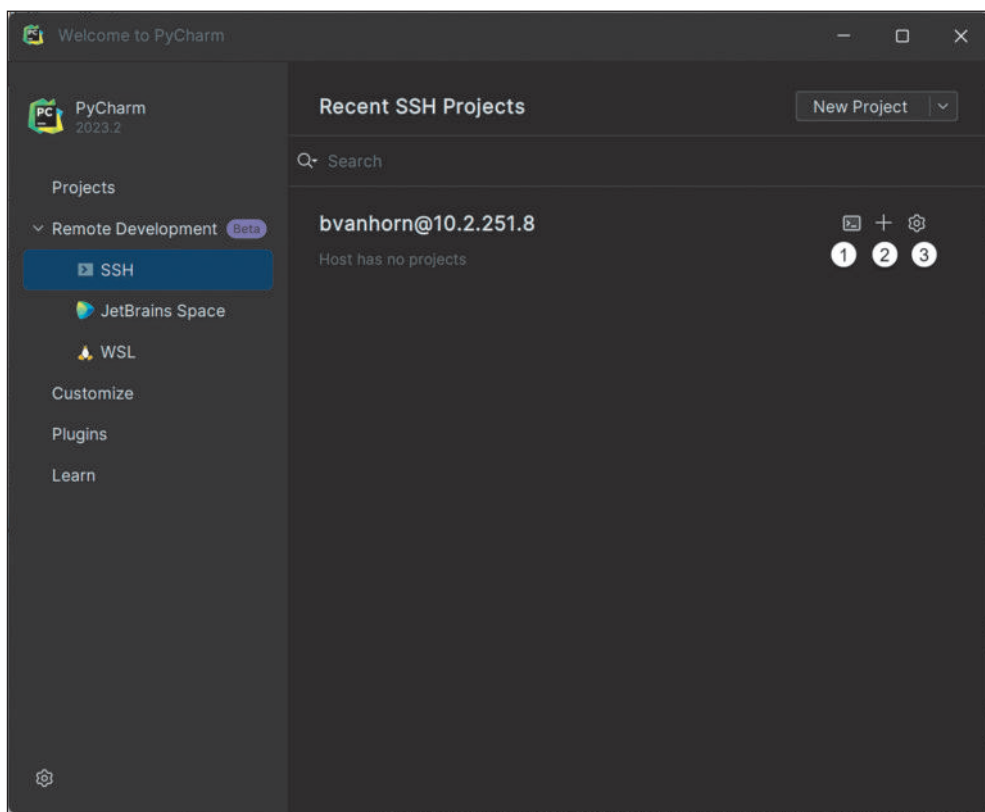


Рис. 15.18. Теперь у нас есть подключенная система с тремя опциями!

Теперь у нас есть три варианта. Вы можете подключиться к хосту с помощью сеанса терминала, кликнув его значок, показанный в (1). Нажатие значка + в (2) позволяет вам создать или открыть проект на удаленном компьютере. Шестеренка (3) позволяет вам либо удалить хост, либо управлять серверными модулями на хосте. Последний вариант объясняется моим предыдущим замечанием. Когда мы впервые подключались, я сказал вам, что PyCharm скопировал на хост какое-то программное обеспечение. Рассматриваемое программное обеспечение во многом похоже на наше приложение Toolbox. Это поможет нам управлять серверными модулями IDE, которые мы будем загружать в процессе создания проекта. Есть несколько других IDE, которые можно использовать таким образом, и вы можете управлять этими бэкендами здесь. Если вы нажмете на них прямо сейчас, то их не будет, потому что мы еще не создали ни одного проекта. Давайте это исправим.

Создание удаленного проекта

Нажмите кнопку +, указанную в позиции (2) на рис. 15.18. В награду вы увидите экран, показанный на рис. 15.19.

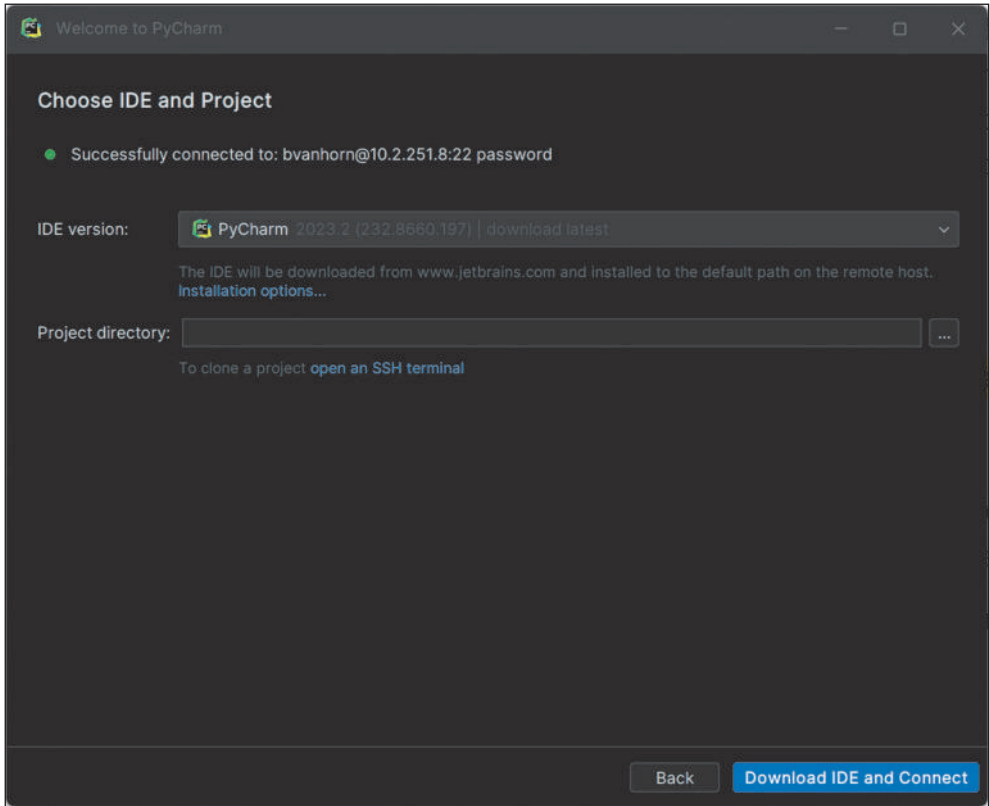


Рис. 15.19. Мы собираемся создать новый удаленный проект, для которого необходимо загрузить серверную часть IDE и выбрать папку проекта на удаленном компьютере

Для создания удаленного проекта вам необходимо указать, какую IDE вы хотите использовать и где хотите хранить свой код на удаленном жестком диске. Учитывая, что мы вызываем PyCharm для создания проекта, может показаться странным, что он просит нас выбрать, какая IDE нам нужна. Однако помните, что на самом деле это не PyCharm. Это плагин, который можно использовать во многих IDE JetBrains. Прямо сейчас вы на самом деле взаимодействуете с другим программным обеспечением, которое случайно размещено в PyCharm.

Мы получаем PyCharm по умолчанию, так что это нечто. Обратите внимание на версию. У меня по умолчанию используется версия EAP, и это может быть не то, что вам нужно. Лично я не использую EAP, если только не проверяю его на обратную совместимость проекта. В своей повседневной работе я использую стабильную версию. Выберите соответствующую запись из раскрывающегося списка, как показано на рис. 15.20.

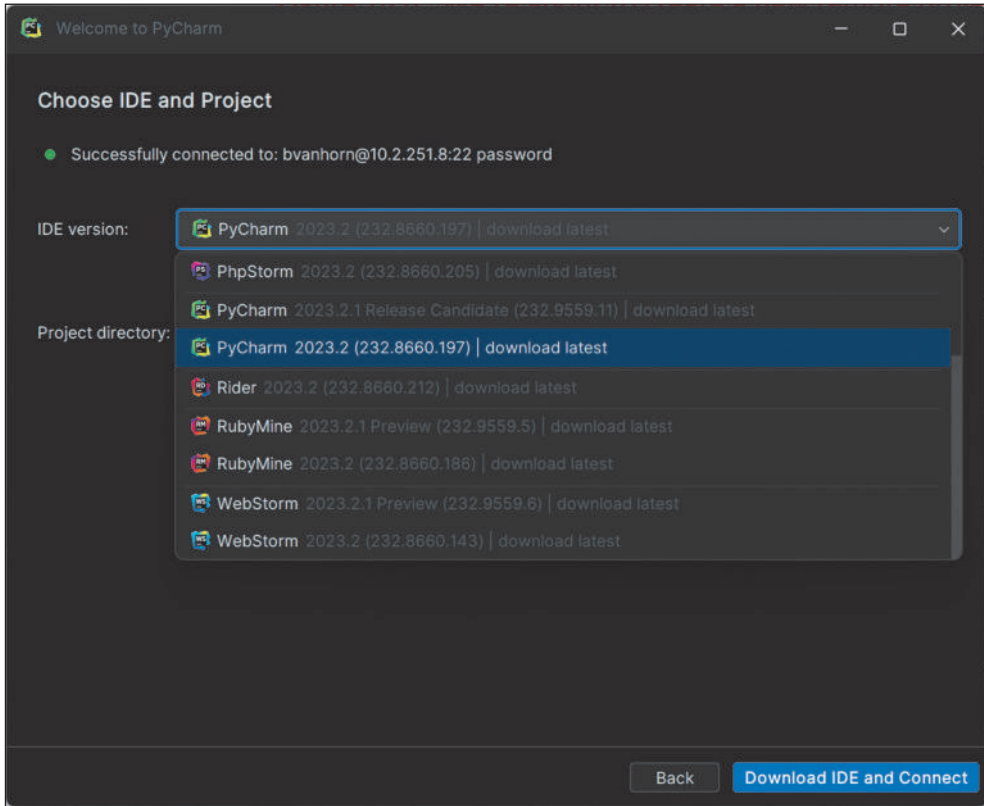


Рис. 15.20. Выберите IDE, которую вы хотите использовать на удаленном компьютере

Другое, возможно, очевидное соображение заключается в том, что вам потребуется лицензия на любую IDE, которую вы выберете. Далее нам нужно выбрать место. Это может сбить вас с толку, поскольку структура файловой системы Linux отличается от структуры Windows. Linux не использует буквы дисков и использует косую черту для разделения элементов пути, тогда как Windows использует обратную косую черту. К счастью, я могу просто посмотреть это, кликнув три точки рядом с настройкой каталога проекта. Посмотрите на рис. 15.21.

По умолчанию браузер каталогов отображает вашу домашнюю папку на удаленном компьютере. Вы можете перейти к папке по вашему выбору. К сожалению, на данный момент отсутствует одна большая функция – возможность создать новую папку для вашего проекта. Я уверен, ее установят, но в данный момент ее нет. Нам придется поработать над этим. Нажмите **Cancel**, затем **Back**, чтобы вернуться к экрану, показанному на рис. 15.5. Нажмите кнопку терминала, чтобы запустить сеанс терминала SSH на хосте, как показано на рис. 15.22.

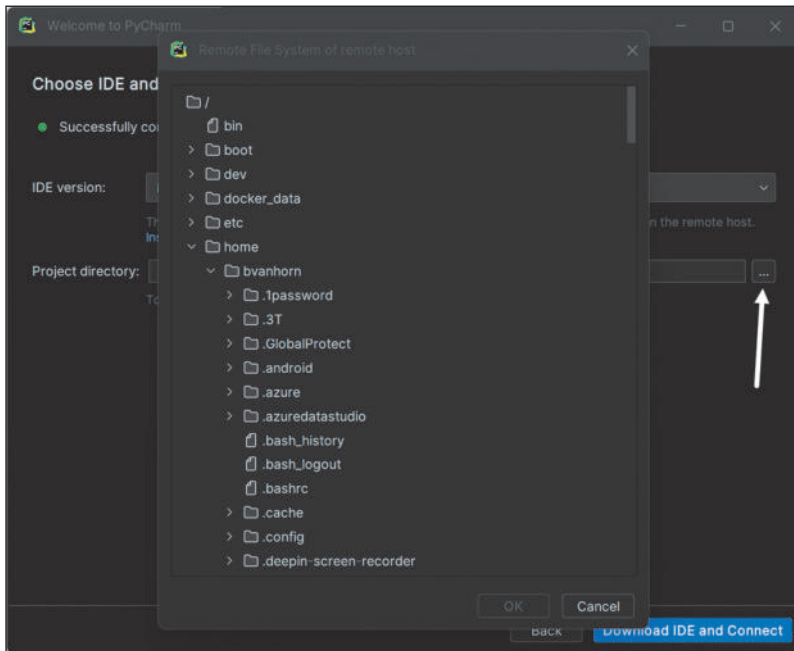


Рис. 15.21. Нажмите три точки, чтобы просмотреть файловую систему на удаленном компьютере и выбрать каталог проекта

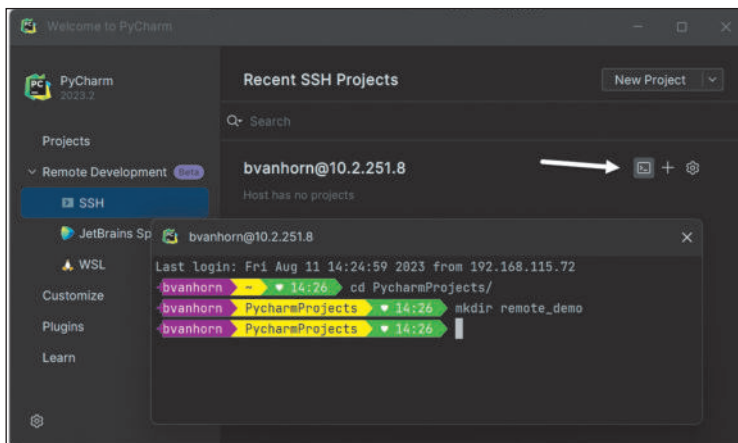


Рис. 15.22. Используйте сеанс терминала для создания папки

Используйте показанные команды, чтобы создать новую папку. У меня уже была папка PyCharmProjects на удаленном компьютере в моей домашней папке. Это связано с тем, что PyCharm фактически установлен на удаленном компьютере, поскольку это мой компьютер для ежедневного использования. Вы можете создать там любую подходящую вам папку. Можете выйти из сеанса терминала, набрав команду `exit`.

Давайте попробуем еще раз

Как и раньше, кликните значок +, выберите свою IDE из раскрывающегося списка и перейдите к только что созданной папке.

Просто думайте об этом как о PyCharm без пользовательского интерфейса. Мы подключаем интерфейс на нашем компьютере к загруженному серверу PyCharm на удаленном компьютере.

Давайте снова сосредоточимся на удаленной работе над проектом. Нажмите кнопку **Download IDE and Connect**, показанную на рис. 15.23.

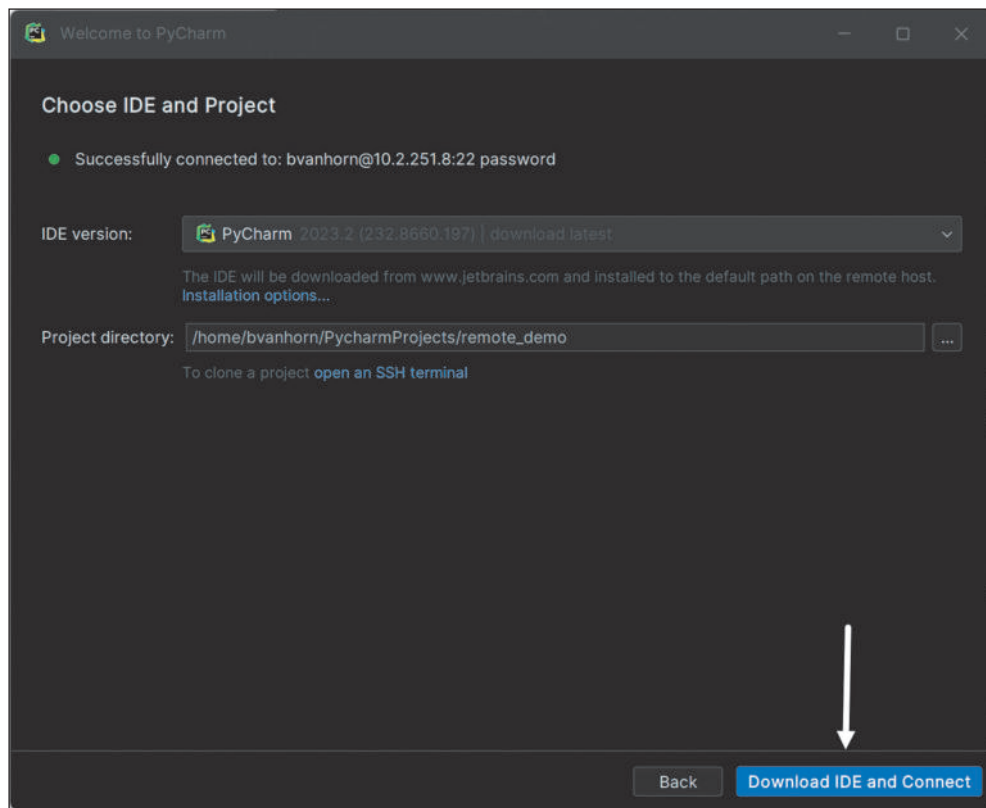


Рис. 15.23. Нажмите кнопку, чтобы создать проект на удаленном компьютере

Следующая часть займет несколько минут. После того как вы нажмете кнопку **Download IDE and Connect**, на вашем удаленном компьютере запустится воспитательная оркестровка команд. В результате получается работающая серверная часть IDE, как показано на рис. 15.24.

Кликните ссылку, показанную на рис. 15.11, чтобы подключиться к удаленной IDE. Скорее всего, вы увидите проблему с паролем и, возможно, сообщение от Windows с вопросом, можно ли пропускать трафик через брандмауэр, но в конечном итоге вы будете работать на удаленном компьютере, как и я на рис. 15.25.

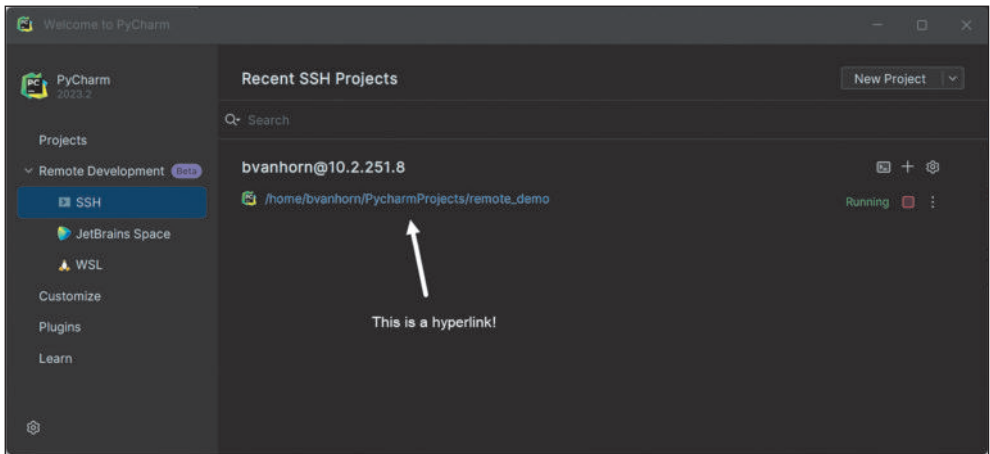


Рис. 15.24. Теперь у вас есть бэкэнд-сервер IDE, работающий на удаленном компьютере

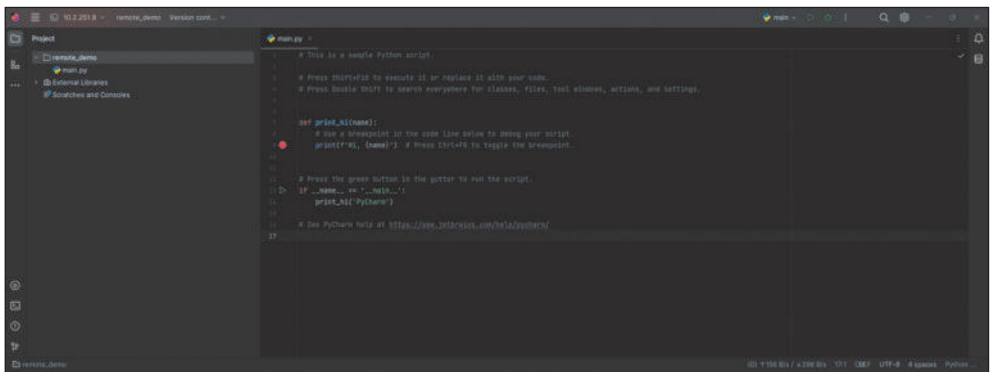


Рис. 15.25. Волшебство! Это выглядит так же, как работа локально!

На данный момент вы в основном знаете, что делать. Ваша IDE будет работать нормально, за исключением одного пропущенного шага: вам никогда не предлагалось создать виртуальную среду.

Создание виртуальной среды на удаленном компьютере

Мы уже делали это раньше, в главе 3, поэтому я не буду здесь углубляться в подробности, но есть один момент, который сбивает с толку. Давайте посмотрим на это.

Чтобы создать виртуальную среду, нужно зайти в настройки проекта и найти настройку интерпретатора проекта. Мой показан на рис. 15.26.

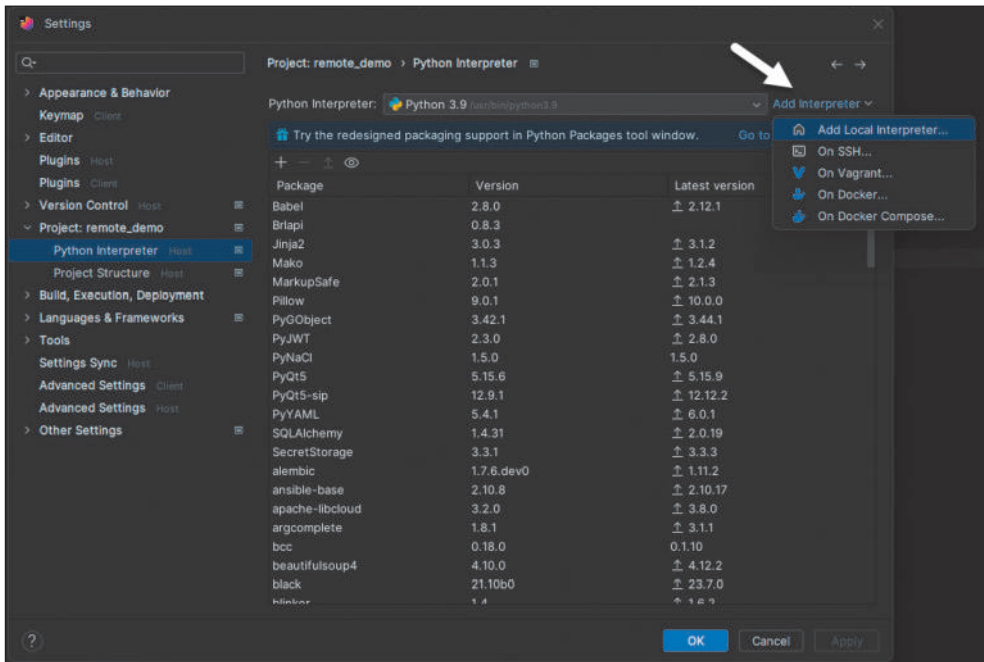


Рис. 15.26. Когда вы добавляете интерпретатор, помните, что, хотя он и удаленный, он рассматривается как локальный

Интерпретатор по умолчанию имеет значение `/usr/bin/python3.9`, которое является глобальным экземпляром, а не тем, что мне нужно. Когда я нажимаю кнопку **Add Interpreter**, это становится немного запутанным. У вас может возникнуть соблазн использовать **On SSH**, потому что мы используем SSH для подключения к удаленному устройству. Однако мы уже подключены к удаленному компьютеру, поэтому PyCharm работает на удаленном компьютере, что делает его локальным интерпретатором. Когда вы нажмете **Add Local Interpreter**, обратите внимание, как на рис. 15.14, что путь похож на Linux, а не на путь Windows. Для пользователей macOS это будет менее очевидно, поскольку путь выглядит одинаково.

Другие соображения

Теперь, когда у вас есть эта настройка, вы фактически запускаете PyCharm на удаленном компьютере. Все ваши пути к файлам будут отображаться как пути Linux. Git и другие клиенты контроля версий работают с удаленной папкой. Если вы используете ключи SSH для аутентификации на GitHub или на хосте контроля версий, вам потребуется настроить ключи удаленного устройства, а не просто использовать локальные открытые ключи. По сути, полностью забудьте о своем локальном компьютере. Вы работаете полностью на удаленном доступе.

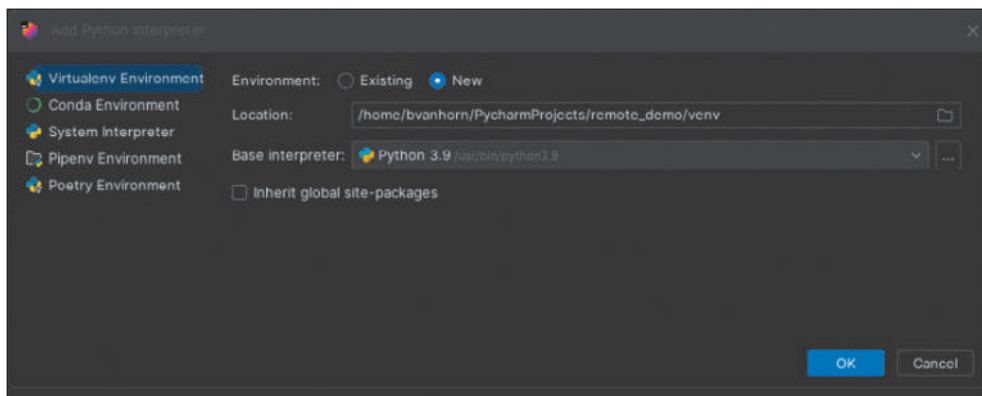


Рис. 15.27. Несмотря на то что мы выбрали локальный вариант, все происходит на удаленном компьютере

Нажмите **ОК**, и вы обнаружите, что для проекта создано подходящее `venv`. Теперь вы готовы начать работу над своим проектом!

РАБОТА С DOCKER

Контейнерная технология – одно из наиболее важных достижений в технологии DevOps со времен виртуальной машины. **Docker** и подобные контейнерные технологии произвели революцию в способах разработки, развертывания и управления программными приложениями. Представьте себе контейнеры как легкие автономные модули, инкапсулирующие все необходимые компоненты для запуска приложения, включая код, среду выполнения, системные библиотеки и настройки. Такая инкапсуляция гарантирует, что приложение будет работать согласованно в различных средах – от ноутбука разработчика до рабочих серверов.

В частности, Docker – это широко используемая платформа для создания, доставки и запуска контейнеров. Он обеспечивает стандартизированный способ упаковки приложений и их зависимостей, изолируя их от базовой хост-системы. Такая изоляция позволяет разработчикам избежать пресловутой проблемы «на моей машине это работает», поскольку контейнеры ведут себя одинаково в различных средах.

Еще одним важным преимуществом контейнеров является их мобильность. Разработчики могут упаковать приложение и все его зависимости в образ контейнера, которым затем можно будет легко поделиться с другими. Это делает совместную работу более эффективной и снижает вероятность проблем с совместимостью. Кроме того, контейнеры хорошо масштабируются. Их можно быстро реплицировать для обработки возросших рабочих нагрузок, что является отличным решением для современных, динамичных и ресурсоемких приложений.

Docker также улучшает использование ресурсов. Поскольку контейнеры используют ядро операционной системы хоста, они более легкие, чем традиционные виртуальные машины, для которых требуются отдельные операционные системы. Это приводит к более быстрому запуску, меньшим затратам памяти и более высокой плотности контейнерных приложений на одном хосте.

Входящий в комплект PyCharm плагин Docker

PyCharm Professional поставляется с уже включенным плагином JetBrains Docker. Чтобы использовать плагин Docker, вам не нужно ничего больше. Однако необходимо его установить. На своих компьютерах для разработки я обычно запускаю Docker Desktop, потому что считаю графический интерфейс продукта полезным. Сегодня это будет полезно, поскольку я покажу вам, как много можно сделать в PyCharm с помощью Docker.

Давайте создадим проект Flask и запустим его в Docker.

Создаем проект

В PyCharm Professional создайте новый проект Flask. Если вы пропустили, я показал вам, как это сделать, в главе 8. Настройки моего проекта вы можете увидеть на рис. 15.28.

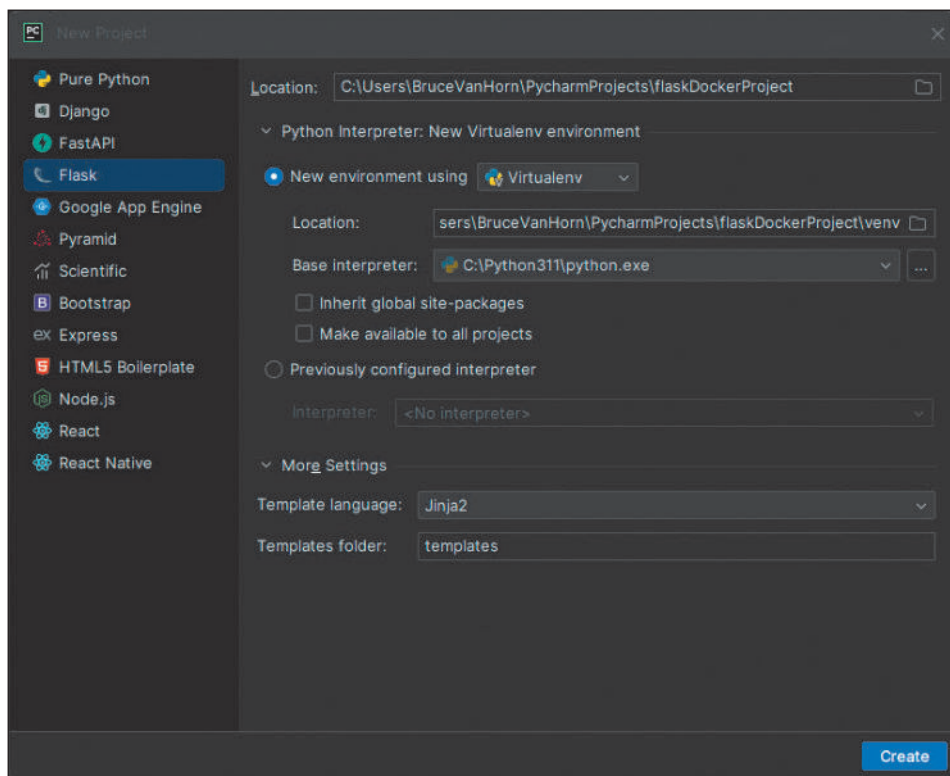


Рис. 15.28. Создайте проект Flask, чтобы мы могли запустить его в Docker

Мы не будем вносить никаких изменений в сгенерированный проект, кроме добавления **Dockerfile**. На рис. 15.29 показано, как я это делаю.

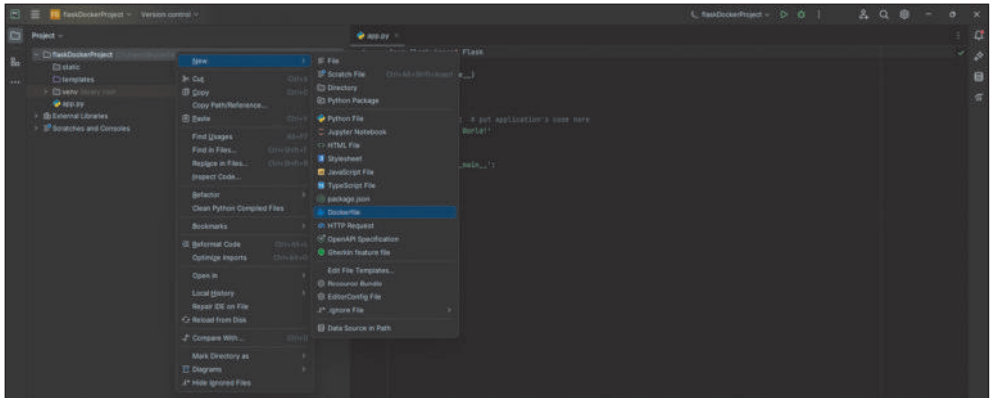


Рис. 15.29. Кликните проект правой кнопкой мыши и затем кликните New | Dockerfile

На предыдущем рисунке я кликаю проект правой кнопкой мыши, навожу курсор на **New** и выбираю пункт меню **Dockerfile**. Это создает для меня шаблон Dockerfile со следующим содержанием:

```
FROM ubuntu:latest
LABEL authors="BruceVanHorn"

ENTRYPOINT ["top", "-b"]
```

В первой строке указано, какое ядро Linux будет служить основой для нашего образа контейнера. Я большой поклонник Ubuntu, но есть и другие возможности. Alpine Linux также очень популярен.

Вторая строка – это именно то, что написано: ярлык для изображения, содержащий имена авторов. На самом деле нам это не нужно, как и последняя строка, по крайней мере, в ее нынешнем виде. Давайте заменим сгенерированный шаблон чем-то более полезным. Корпорация, стоящая за Docker, ведет большой реестр готовых контейнеров, которые вы можете использовать в качестве отправной точки в своей работе. Среди них есть контейнер с уже установленной какой-либо версией Python 3. Я собираюсь использовать Python 3.9, хотя это и не самая последняя версия, потому что я случайно заметил, что на моем компьютере установлена версия 3.9, когда я создавал пример проекта:

```
# Use the official Python 3.9 image as the base image
FROM python:3.9
```

В следующей строке я устанавливаю рабочий каталог внутри контейнера. Контейнер имеет минимальную файловую систему с ограниченным пространством. Здесь будет жить код нашего приложения:

```
WORKDIR /app
```

Далее мне нужно поручить Docker скопировать мой текстовый файл с требованиями и использовать `pip` для установки требований моего проекта:

```
# Copy the requirements file and install the necessary packages
COPY requirements.txt requirements.txt
RUN pip install -r requirements.txt
```

Далее я скопирую свой код в контейнер:

```
# Copy the Flask application code into the container
COPY . .
```

Инструкция `COPY` скопирует все содержимое папки моего проекта, включая папку `venv`. Поскольку команда `pip` установит наши требования, папка `venv` нам не нужна. Я исправлю это через мгновение. Мой следующий шаг в этом файле – открыть порт 5000 на брандмауэре контейнера, чтобы я мог получить доступ к приложению:

```
# Expose the port on which the Flask app will run
EXPOSE 5000
```

Наконец, я определяю команду для запуска приложения Flask при запуске контейнера:

```
CMD ["python", "app.py"]
```

Чтобы решить проблему копирования, я создам в папке проекта файл с именем `.dockerignore`. Это работает так же, как работает `.gitignore`, за исключением того, что он, очевидно, используется для установки исключений файлов для Docker. Содержимое файла `.dockerignore` просто следующее:

```
venv
```

Кстати, в Marketplace вы можете найти плагин под названием `.ignore`, который добавляет форматирование к массиву игнорируемых файлов.

Добавьте конфигурацию запуска Docker

Мы рассмотрели создание и использование конфигураций запуска в главе 3, поэтому здесь мы не будем рассматривать это снова. Я просто укажу на возможность создания конфигурации запуска с помощью Docker, как показано на рис. 15.30.

Большая часть вашей конфигурации запуска будет заполнена за вас. PyCharm увидит ваш `Dockerfile` и будет использовать его по умолчанию. Мои настройки показаны на рис. 15.31.

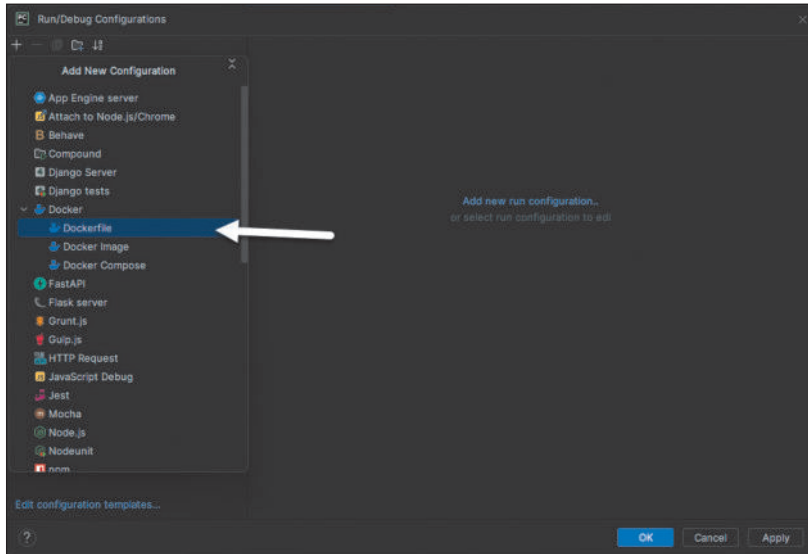


Рис. 15.30. Используйте параметр Dockerfile для создания конфигурации запуска, использующей Docker

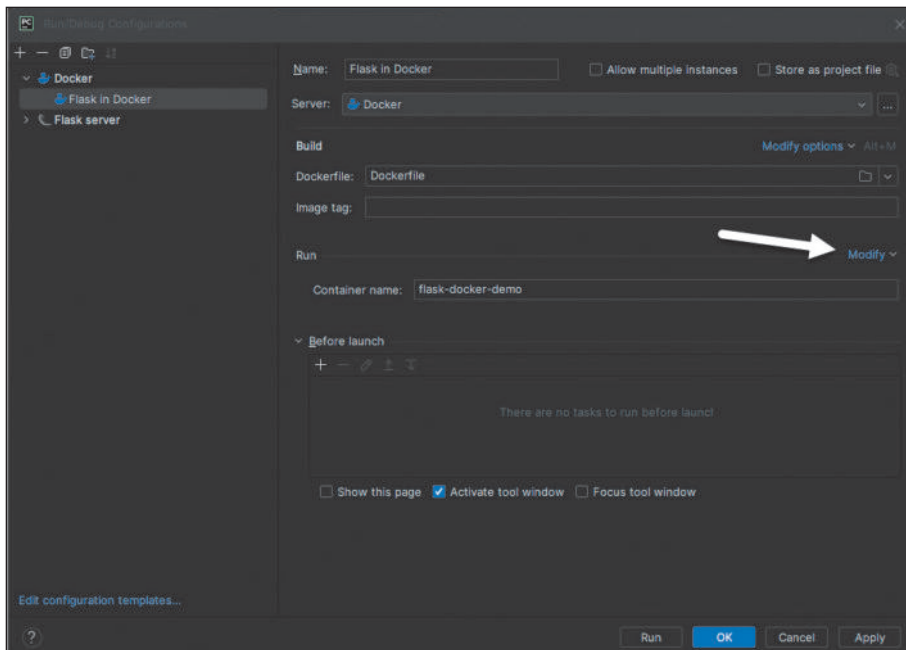


Рис. 15.31. Моя конфигурация Docker

Если бы мы это запустили, Docker извлек бы образ Python 3.9, создал бы контейнер и запустил приложение. Если мы действительно хотим использовать приложение, нужно сделать еще одну вещь. Стрелка на рис. 15.31 указывает на меню, которое позволяет вам настроить команду запуска Docker. Обычно здесь есть вещи, которые вы захотите сделать в зависимости от вашего приложения, например мапирование томов постоянного хранилища или, в нашем случае, мапирование порта в контейнере с портом на нашем локальном компьютере.

Нажмите **Modify | Bind ports**, как показано на рис. 15.32.

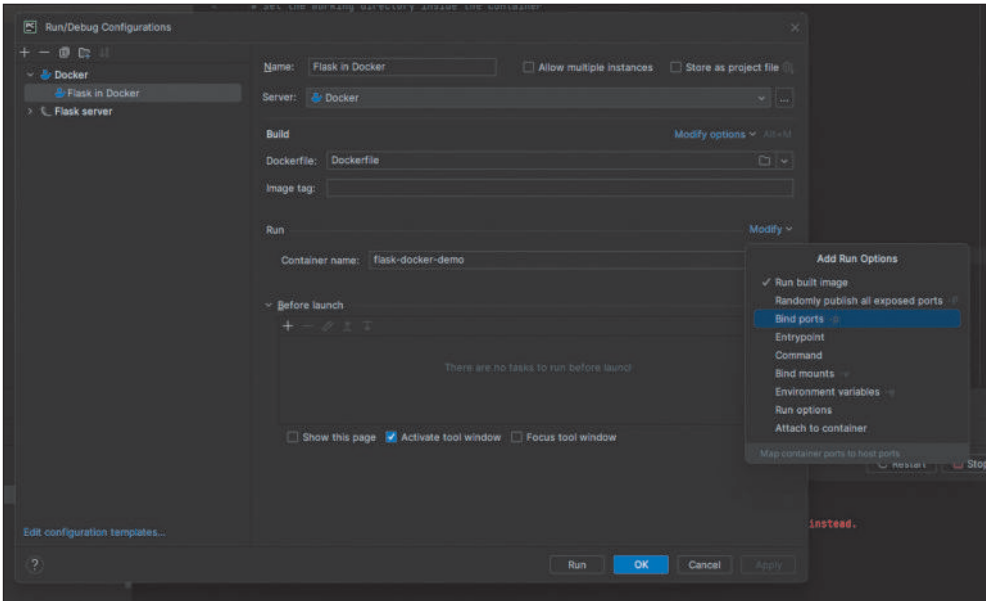


Рис. 15.32. Вам необходимо добавить порт привязки для контейнера Docker, чтобы видеть приложение, работающее из вашего браузера

После того как вы выберете опцию **Bind ports**, PyCharm добавит параметр конфигурации для **Bind ports**. Как ни странно, на рис. 15.33 есть значок папки, обозначенный стрелкой вправо.

Это странно, потому что нам не придется искать файл или папку. При нажатии на нее открывается диалоговое окно **Bind ports**. По умолчанию ваше приложение Flask работает на порту 5000, поэтому все, что нужно сделать, – это указать привязку между 5000 в контейнере и 5000 на вашем компьютере. Нажимайте **OK**, пока не выйдете из конфигурации запуска. Запустите конфигурацию запуска. Мы объяснили конфигурации запуска еще в главе 3, поэтому, если вы присоединяетесь к нам поздно, вернитесь и просмотрите ее.

Вы увидите текстовое световое шоу, пока Docker загружает фрагменты в образ контейнера, создает контейнер и, наконец, запускает приложение. Вы можете увидеть работу моего на рис. 15.34.

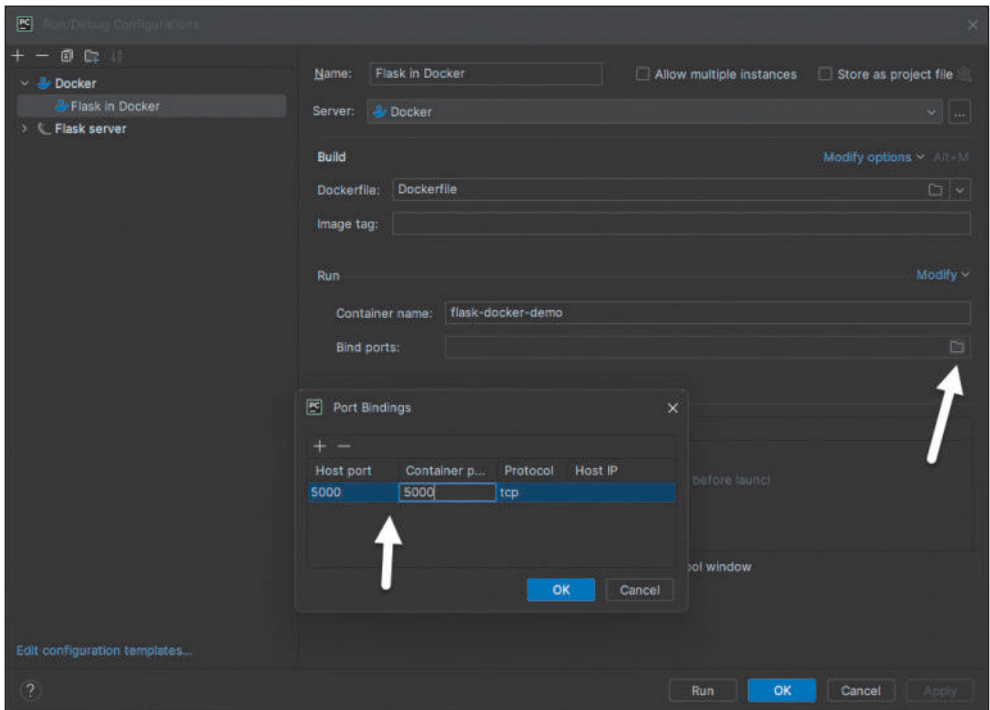


Рис. 15.33. Кликните папку, чтобы открыть диалоговое окно Bind ports и настроить маппинг портов

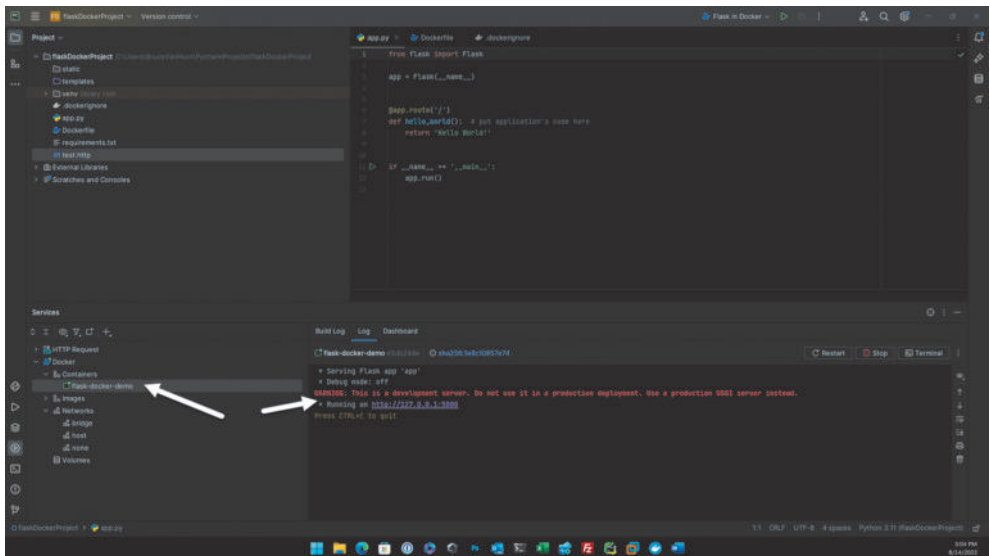


Рис. 15.34. Приложение My Flask работает в контейнере Docker

Если вы захотите сравнить пользовательский интерфейс Docker и Docker Desktop, вы увидите, что многие функции одинаковы. Краткий обзор показан на рис. 15.35.

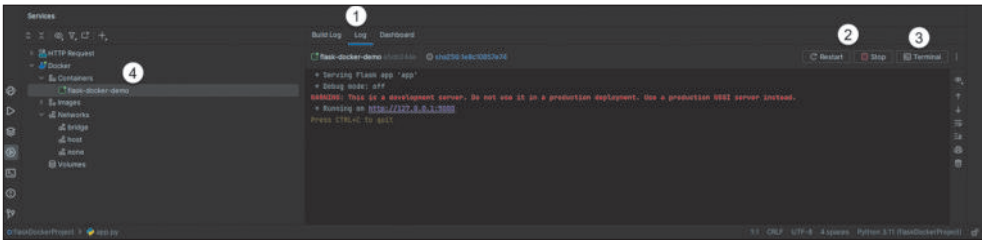


Рис. 15.35. PyCharm обладает многими функциями, которые можно ожидать от компетентного пользовательского интерфейса Docker

В позиции (1) мы видим три вкладки. **Build Log** позволяет нам проверять выходные данные, созданные при создании контейнера Docker из нашего файла Dockerfile. У нас есть некоторые базовые элементы управления для остановки и перезапуска контейнера в точке (2). Естественно, когда контейнер будет остановлен, здесь также появится возможность его запуска. Тогда, возможно, наиболее полезно будет использовать кнопку **Terminal** (3), чтобы запустить сеанс терминала внутри контейнера.

С левой стороны (4) мы видим полный дисплей, показывающий все наши запущенные контейнеры, загруженные образы, сети и смонтированные тома.

В общем, в PyCharm вы действительно можете делать почти все, что можно делать в Docker Desktop. Еще раз, миссия PyCharm – предоставить вам интегрированный опыт работы с одной из наиболее важных технологий, используемых сегодня, – полностью реализована!

КРАТКОЕ СОДЕРЖАНИЕ

Плагины PyCharm – это настраиваемые надстройки, способные еще больше расширить список функций и возможностей, которыми можно воспользоваться при использовании PyCharm. Мы увидели, как просматривать, загружать и управлять различными плагинами в среде PyCharm. Воспользовавшись этими плагинами, мы можем дополнительно настроить наше рабочее пространство и повысить собственную производительность. Управление плагинами в PyCharm можно выполнить на вкладке **Plugins** в настройках.

Мы рассмотрели несколько простых плагинов, которые улучшат качество вашей жизни при кодировании, например плагины тем (theme) и некоторые полезные файловые хендлеры. Мы рассмотрели один из моих любимых инструментов – GitToolbox, который позволяет вам увидеть гораздо больше деталей из вашего репозитория Git прямо в окне IDE.

Затем мы расширили тему и рассмотрели некоторые плагины с очень большими функциональными возможностями, которые являются довольно новыми для PyCharm 2023. Удаленная разработка очень полезна в постпандемическом мире, где все больше и больше из нас работают удаленно. Даже если это не так,

скорее всего, вы все чаще работаете с облачными серверами и виртуальными машинами, а не всегда используете жесткий диск на локальном ноутбуке.

Еще одним большим достижением является поддержка PyCharm Docker. Входящий в комплект плагин Docker обеспечивает полную интеграцию: от автозаполнения и подсветки синтаксиса в файлах Dockerfiles и файлах композера (composer) до полноценного графического интерфейса для управления вашими контейнерами. Вам никогда не придется покидать PyCharm, чтобы перейти на Docker Desktop и выполнять операции Docker там. Естественно, если захотите, вы обнаружите, что обе среды полностью синхронизированы, и также можно использовать окно **Terminal** в PyCharm для взаимодействия с вашими контейнерами именно таким образом, если вы это предпочитаете.

Плагины отвечают за добавление множества очень интересных и очень полезных функций в нашу IDE. Есть несколько плагинов, о которых я не рассказывал, поскольку, честно говоря, они еще не совсем готовы. У меня есть некоторый доступ к функциям закрытого бета-тестирования и некоторым скрытым сокровищам, о которых я хочу рассказать в следующей главе. Мне не терпится показать вам кое-что из того, что появится в будущих версиях!

Ваши следующие шаги с PyCharm

Возможности PyCharm настолько обширны, а улучшения от JetBrains появляются так быстро, что за ними трудно уследить.

В этой завершающей главе я хочу сделать еще кое-что, чтобы отдать должное этому удивительному программному обеспечению. Здесь мы рассмотрим следующее:

- я подведу итоги, сделав краткий обзор и прокомментировав наше совместное путешествие;
- расскажу о нескольких интересных функциях, которые помогут повысить вашу производительность, они не вписались в темы представленных до сих пор глав;
- расскажу о некоторых новых функциях, представленных за те 8 месяцев, которые мне потребовались для написания книги. Я рассматривал версию 2023.1 и не стал ее обновлять, чтобы сохранить соответствие со скриншотами пользовательского интерфейса, используемыми на протяжении всей книги. Вышла версия 2023.2, в которой есть несколько потрясающих изменений, включая бета-версию нового **AI Assistant!**
- подниму бокал и пролью слезу, когда мы будем расставаться до выхода следующей книги.

Я сделаю все возможное, чтобы не проявлять к вам эмоций. Мы уже некоторое время вместе, и нам будет трудно пройти этот победный круг, зная, что книга почти закончена. Я тут немного запутался, просто подумав об этом, так что давайте продолжим.

Чтобы продолжить работу с этой главой, вам понадобится следующее :

- установленный и работающий интерпретатор Python. Я буду использовать последнюю версию <https://python.org>;
- установленные копии `pip` и `virtualenv`. Вы получаете их автоматически при установке Python в Windows, а macOS включает их в каждую систему. При использовании Linux вам необходимо отдельно установить

менеджеры пакетов (например, `pip`) и инструменты виртуальной среды (например, `virtualenv`). В наших примерах будут использоваться `pip` и `virtualenv`;

- установленная и рабочая копия PyCharm. Установка была описана в главе 2. Снимки экрана в этой главе взяты из версии 2023.2.1.

Пример исходного кода этой книги с GitHub можно найти по адресу <https://github.com/PacktPublishing/Hands-On-Application-Development-with-PyCharm---Second-Edition/tree/main/chapter-16>. Мы рассмотрели клонирование кода в главе 2.

РАЗНООБРАЗИЕ ВОЗМОЖНОСТЕЙ PYCHARM

Как я уже сказал, объем возможностей этой IDE ошеломляет. В этом разделе я расскажу о нескольких вещах, которые не вписались в другие главы. Я буду представлять их скорее в духе фразы «Эй, посмотрите на это!», а не представлять полное руководство по каждой функции.

Удаленные виртуальные среды

Эта функция существует у нас уже некоторое время, и я думаю, что она, вероятно, затмевается новыми функциями удаленной разработки, которые мы рассмотрели в главе 15. Эта функция позволяет работать с интерпретатором на другом компьютере, доступном через SSH. Для этого есть несколько вариантов использования. Я использовал эту функцию для отладки копии приложения, используя виртуальную среду на рабочем сервере. У меня на компьютере есть копия производственного кода, но виртуальная среда находится в производственной системе. Это позволяет мне воспроизвести баг и исправить его локально, используя именно ту среду, в которой он и был пойман. Это было особенно эффективно, когда я использовал ноутбуки Mac и Windows в качестве основной среды разработки, а производственная среда работала в Linux. Как мы уже отмечали, многие пакеты развертываются по-разному в зависимости от операционной системы. У меня всегда были проблемы с **NumPy**, **pandas** и **pymssql** (драйвер, поддерживаемый Microsoft для SQL Server), которые работают по-разному в разных средах, особенно в Windows. Для создания сторонних пакетов, использующих код C, вам понадобится компилятор. В Linux это *cmake*. В Windows это компилятор Microsoft C++. Между ними есть большая разница, и тестирование приложения только на Windows мне не подходит. Мне нужен более надежный тест, и с помощью этой функции я могу использовать именно ту среду, в которой он будет выполняться.

Вы можете создать или использовать виртуальную среду на удаленном компьютере, настроив с ней SSH-соединение. Мы рассмотрели создание SSH-соединения еще в главе 15, поэтому я не буду повторять инструкции здесь. Опции для использования удаленной среды находятся рядом с обычными опциями в настройках интерпретатора проекта, показанных на рис. 16.1.

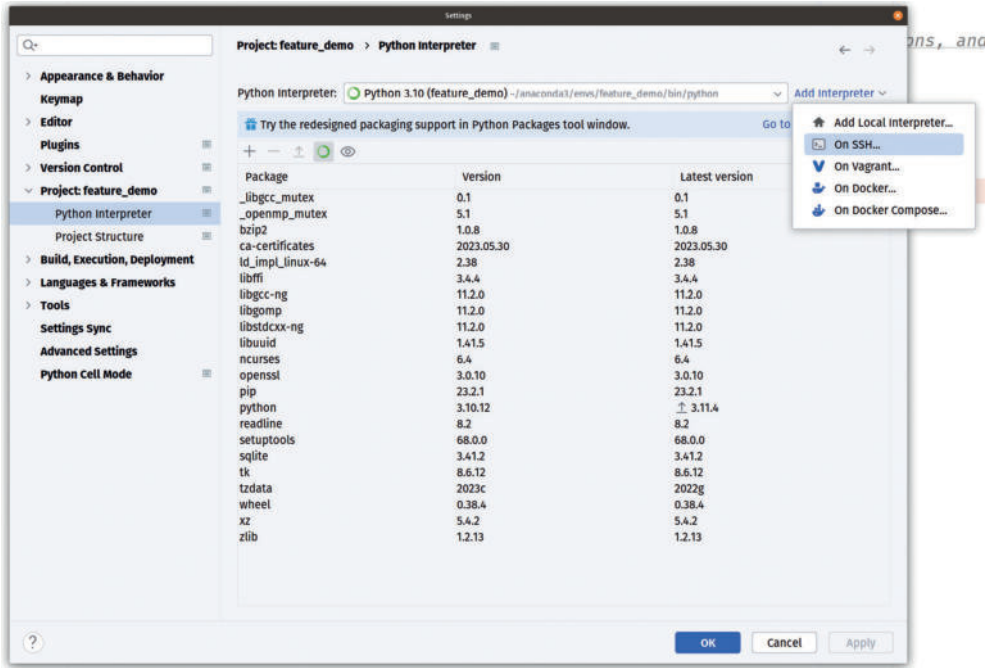


Рис. 16.1. Используйте диалоговое окно настроек интерпретатора вместе с ранее настроенным SSH-соединением для настройки виртуальной среды в удаленной системе Linux

Естественно, такой подход приведет к значительному снижению производительности; по крайней мере, у меня было так. Но в моем случае оно того стоило!

Обратите внимание, что существует также опция **On Vagrant**, которая, по сути, является тем же, но вместо случайного удаленного сервера она находится на локальной виртуальной машине, управляемой **Vagrant** от **HashiCorp**. Что такое Vagrant, спросите вы.

Работа с HashiCorp Vagrant

В области DevOps очень важным стало движение под названием «**Инфраструктура как код**» (**Infrastructure as Code, IaC**). Идея состоит в том, что никто больше ничего не запускает на «голом железе»; все запускается на виртуальной машине. Тем не менее одна из проблем, с которыми я сталкивался на протяжении многих лет, заключается в том, что, когда я собираюсь настроить среду тестирования и производственную среду, приходится делать все это вручную, и это занимает довольно много времени. Моя цель – создать как минимум две или три практически идентичные среды.

Допустим, я создаю продукт с помощью Flask, который использует React в качестве внешнего интерфейса и Microsoft SQL Server в качестве базы данных. Чтобы запустить это, используя лучшие практики, мне нужно несколько серверов.

- Нужен веб-сервер, такой как Nginx или Apache, для обслуживания моего приложения React вместе со всеми статическими ресурсами, которые могут понадобиться.

- Сервер приложений. Сервер разработки Flask не собирается сокращать его производительность. Мне нужен сервер, на котором работает что-то вроде **Green Unicorn** (сокращенно *Gunicorn*), который представляет собой сервер WSGI производственного уровня, способный обрабатывать производственные нагрузки. **WSGI** означает **интерфейс шлюза веб-сервера** в области разработки Python. Это спецификация стандартизированного интерфейса между веб-серверами и веб-приложениями или платформами Python, такими как Flask, FastAPI и Django. WSGI определяет набор правил и соглашений, которые позволяют веб-серверам взаимодействовать с приложениями Python согласованным и совместимым способом.
- Мне нужен выделенный сервер базы данных для SQL Server.

Проще говоря, это три сервера, каждый из которых реплицируется три раза: один для тестирования разработки, один для промежуточного или **пользовательского приемочного тестирования (UAT)** и один для производства. Для каждого сервера мне нужно установить, пропатчить и усилить операционную систему. Затем мне нужно настроить каждый тип сервера. Веб-сервер прост: мне просто нужно установить nginx и настроить его так, чтобы он использовал SSL, перенаправлял любые запросы, не использующие HTTPS, кешировал статический контент и выполнял обратные прокси-запросы API на сервер API, на котором работает Flask.

Сервер Flask более задействован, поскольку передовая практика требует наличия nginx и в этой системе, перенаправляющего трафик на Green Unicorn. Обычно я настраиваю на этом сервере правила брандмауэра, чтобы отклонять любой трафик, не поступающий с веб-сервера, и блокировать все ненужные порты.

Сервер базы данных является самым сложным. В идеале мне нужны отдельные разделы для ОС, журналов и данных, чтобы избежать узких мест в производительности ввода-вывода. В сумме мне может легко потребоваться неделя, чтобы настроить и проверить небольшую среду.

Если бы я использовал IaC, я мог бы написать скрипт, который автоматизирует настройку и удаление каждой из трех сред. Написание скрипта займет столько же времени, сколько и настройка всего вручную, но, когда придет время масштабировать или обновить мою инфраструктуру, наличие его в коде будет очень полезно. Поскольку это есть в коде, то все документировано. Если я хочу обновить ОС, скажем, с Ubuntu 20 на 22, я просто меняю переменную и могу пересобрать все новые серверы или обновить те, которые у меня есть. IaC стоит потраченного времени в каждом отдельном случае, за исключением одноразового прототипа.

Одной из наиболее популярных платформ IaC является **Terraform** от **HashiCorp**. В JetBrains Marketplace вы найдете плагины для языка **HashiCorp (HCL)**, которые позволяют работать с файлами Terraform с обычной раскраской синтаксиса и функцией автозаполнения. Terraform разработан, чтобы помочь вам настроить виртуальные машины через провайдеров. Есть провайдер VMware, и именно он преобразует ваш IaC в настоящие виртуальные машины. Есть также провайдеры облачных услуг, включая Azure, AWS и Digital Ocean.

HashiCorp также производит продукт под названием **Vagrant**. Vagrant предназначен для использования в качестве IaC для вашего локального компью-

тера. Я все время говорил, что вам следует использовать Linux для разработки, если вы планируете запускать свое окончательное приложение на сервере Linux. Но большинство людей предпочли бы не отказываться от работы на Windows или Mac. Поскольку Mac основан на Linux-подобной операционной системе, Mac, по крайней мере, близок к этому, но он не будет идентичен, если вы работаете в Ubuntu, Red Hat или Fedora.

Vagrant – это программа, которая в сочетании с локальным гипервизором, таким как VMware Workstation, Oracle VirtualBox или Microsoft Hyper-V, позволяет вам создать скрипт, генерирующий локальную виртуальную машину, называемую *ящик* (box, далее VM), которая может соответствовать вашей производственной среде. По сути, вы автоматизируете создание и обслуживание виртуальных машин на своем локальном компьютере так же, как если бы вы использовали Terraform для своих серверов. Vagrant делает некоторые полезные вещи, например автоматически мапирует папку вашего проекта с виртуальной машиной, поэтому вам не нужно постоянно копировать код на виртуальную машину вручную. Ваш терминал в PyCharm автоматически подключается к виртуальной машине с помощью SSH, поэтому при выполнении команд вы используете локальный терминал в IDE, но виртуальная машина обрабатывает выполнение. Вы могли бы сделать это с помощью **Windows Subsystem for Linux (WSL)**, но это потребует больше усилий, а возможности использования WSL довольно ограничены по сравнению с созданием отдельных виртуальных машин для каждого проекта.

Хотя Vagrant очень полезен, большинство разработчиков для подобных рабочих процессов переходят на Docker, но я все еще встречаю разработчиков, особенно в области кибербезопасности и в образовательных учреждениях, которые продолжают работать с виртуальными машинами. Преимущество Vagrant заключается в том, что контейнеры Docker являются неизменяемыми, т. е. их нельзя изменить после создания. Несмотря на то что это превосходит эксплуатационные требования, поскольку неизменяемость развернутого приложения обеспечивает системе большую стабильность, это может быть менее желательно для повседневной разработки, поскольку разработчику полезно иметь возможность возиться со средой без ограничений.

Vagrant создает VM, который представляет собой просто виртуальную машину и полноценный сервер с полноценной операционной системой. Вы можете делать с ним все, что захотите, и, поскольку он находится на вашем ноутбуке, если ваши махинации сломают систему, вы можете просто убить VM и сделать еще один сброс в исходное состояние за считанные минуты.

Vagrant дает вам возможность создавать веб-приложения, используя реальный IP-адрес вместо использования адреса обратной связи, также известного как localhost. Это обеспечивает преимущества локального тестирования, поскольку приложения, работающие на локальном хосте, часто ведут себя иначе, чем приложения, привязанные к реальному IP-адресу. Кроме того, вы можете сделать IP-адрес вашего Vagrant Box доступным не только в локальной сети, но и в общедоступном интернете. Когда вы будете готовы продемонстрировать свои успехи, можете пригласить других в свой общий VM Vagrant, получить отзывы, а затем отключить общий доступ.

Вы найдете инструменты для Vagrant в меню **Tools**, показанном на рис. 16.2.

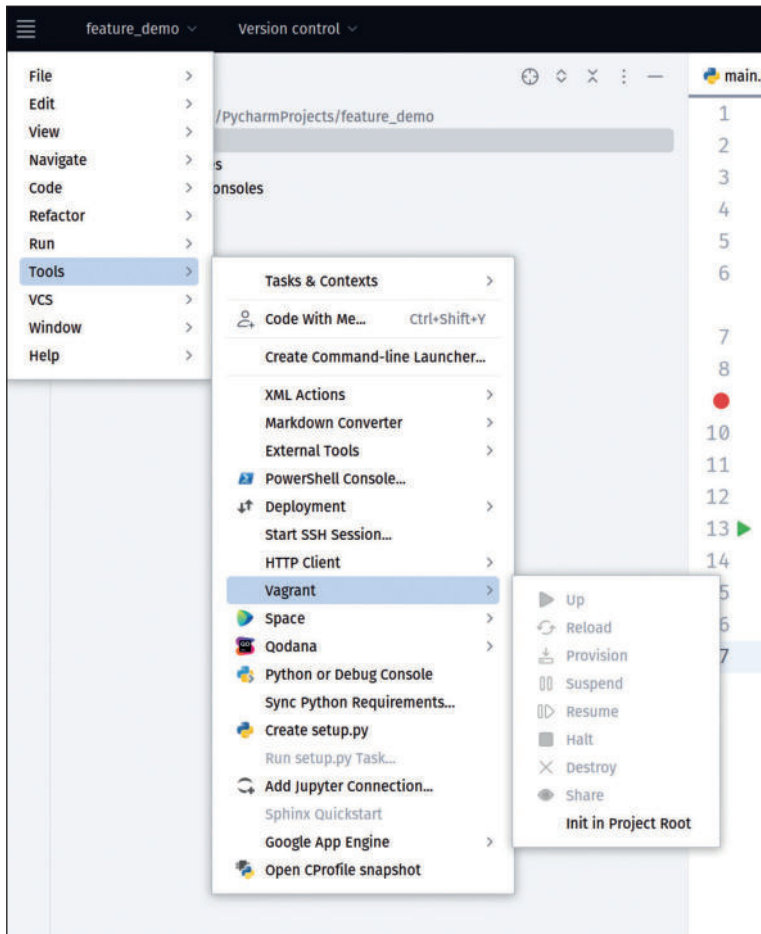


Рис. 16.2. PyCharm поддерживает все команды Vagrant напрямую через параметры меню

Чтобы использовать Vagrant, вам необходимо установить его вместе с поддерживаемым гипервизором. Инструкции по установке можно найти по адресу <https://developer.hashicorp.com/vagrant/tutorials/getting-started/getting-started-install>. Что касается гипервизора, то по умолчанию используется Oracle VirtualBox. Дополнительную информацию о VirtualBox можно найти на странице <https://www.virtualbox.org/wiki/Downloads>.

После установки этих программ вы можете использовать опцию **Init in Project Root**, показанную на рис. 16.2.

Вам будет предложено создать файл с именем Vagrantfile в корне вашего проекта, выбрав имя для вашего поля Vagrant и указав URL-адрес изображения. Диалоговое окно показано на рис. 16.3.

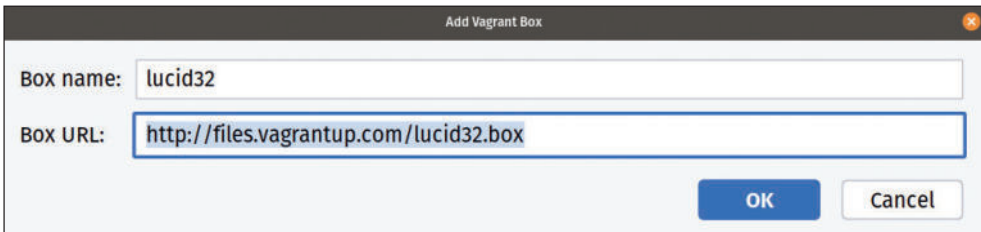


Рис. 16.3. Вам будет предложено создать VM Vagrant с очень старыми настройками по умолчанию

Понятно, что это очень старое значение по умолчанию, поскольку `lucid32` относится к Ubuntu 10. На момент написания текущей версии Ubuntu является 23, а версия с долгосрочной поддержкой – 22.

Вероятно, вам следует найти более новое определение VM, если вы серьезно относитесь к использованию более современной виртуальной машины. Определения VM можно найти на странице <https://app.vagrantup.com>. Я сделал это на рис. 16.4, где искал Jammy Jellyfish, кодовое имя проекта для **Ubuntu 22**, которое я использую для всех своих текущих производственных проектов.

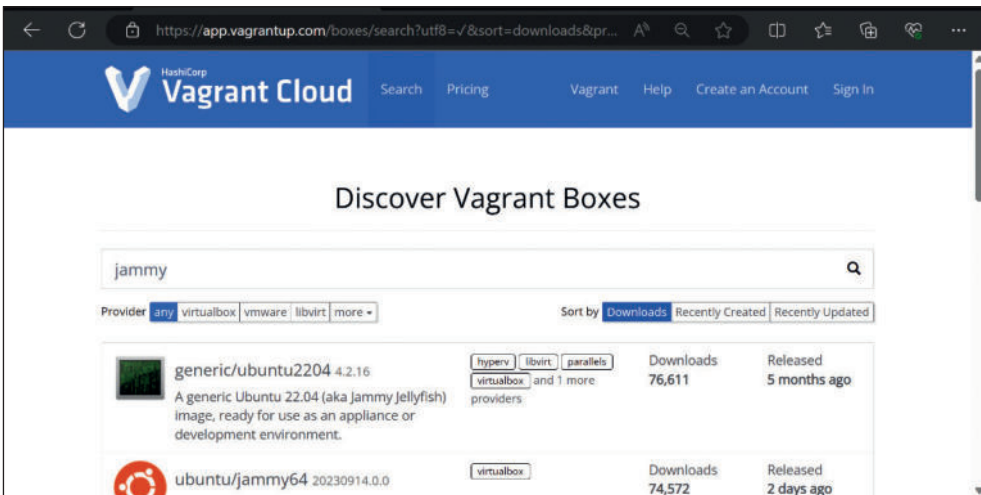


Рис. 16.4. В реестре Vagrant Cloud можно найти множество виртуальных машин с предопределенной конфигурацией

Как только вы найдете ту, которую хотите использовать, нажмите на нее. На рис. 16.5 я нажимаю на вторую запись в списке.

Теперь, когда у нас есть определение VM, вернемся к рис. 16.3. В первом поле вам нужно ввести имя VM, указанное темной стрелкой на рис. 16.5, а именно `ubuntu/jammy64`. Затем скопируйте имя URL-адреса определения VM, указанное светлой стрелкой на рис. 16.5, в наш случай <https://app.vagrantup.com/ubuntu/boxes/jammy64>. Нажатие кнопки **ОК** загрузит образ VM и создаст виртуальную машину. Когда все будет готово, вы обнаружите, что остальные инструменты Vagrant больше не выделены серым цветом в меню, как показано на рис. 16.6.

Сгенерированный Vagrantfile открыт (не автоматически, это сделал я), и вы даже можете видеть, что PyCharm предлагает мне добавить новый файл в репозиторий Git.

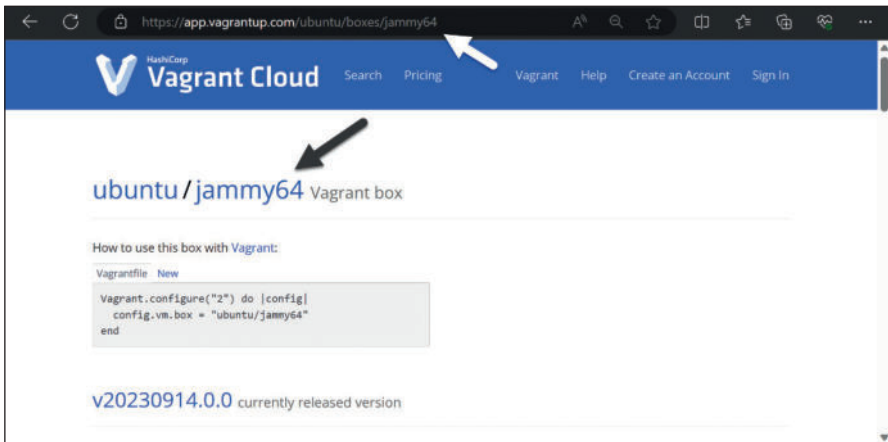


Рис. 16.5. Это определение конфигурации VM для Ubuntu Jammy 64

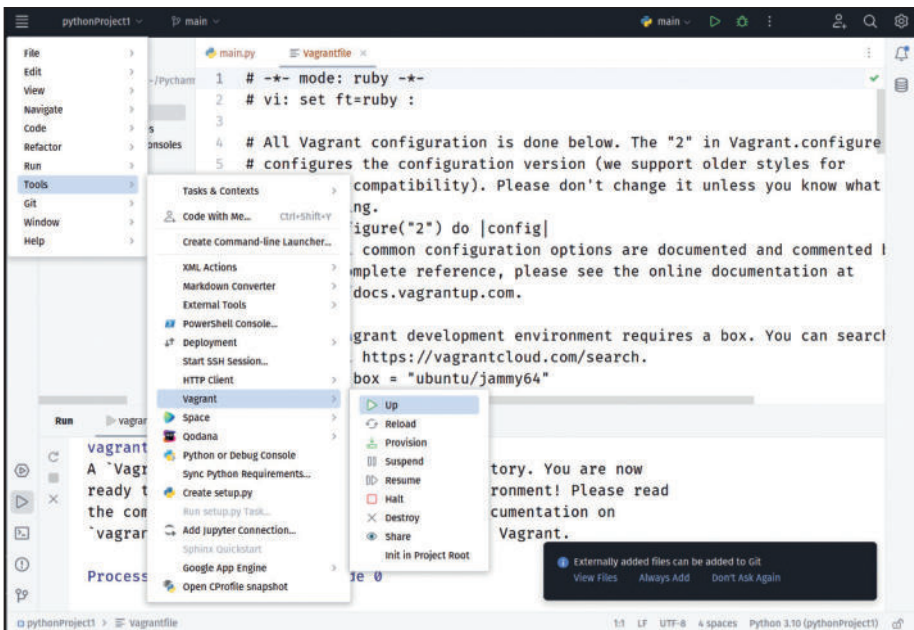


Рис. 16.6. Теперь доступны параметры инструмента Vagrant

Если вы углубитесь в использование Vagrant, то обнаружите, что, как и многие другие интеграции PyCharm, IDE предоставляет вам графический способ выполнения того, что в противном случае было бы рабочим процессом из командной строки. Вы можете запускать команды, используя меню, показанное на рис. 16.6, или можете вводить команды в терминале для достижения того

же результата. У Packt есть несколько хороших книг по разработке с помощью Vagrant. Я обязательно оставлю несколько предложений в разделе «Дальнейшее чтение» этой главы.

Использование Vagrant имеет несколько больших преимуществ. Вы можете поручить всей своей команде разработчиков использовать идентичную среду для разработки и тестирования вашего приложения, используя ту же среду, которую вы будете использовать в рабочей среде. Больше никаких фраз «На моей машине это работает», что означает, что с кодом все в порядке, а виновата проблема с конфигурацией. Поскольку конфигурация среды стандартизирована и поддерживается как часть базы кода, проект должен работать везде одинаково при условии, что код и любые внешние файлы конфигурации (например, файлы `.ini` или `.env`) или переменные среды также идентичны.

Использование IaC в Vagrant защищает менее опытных разработчиков, которые не знают, как настраивать виртуальные машины, а также стандартизирует сборку, поэтому все VM одинаковы. Конфигурация отслеживается в вашей системе контроля версий и поэтому отслеживается и обновляется так же, как код приложения.

Отслеживание вашего времени

За свою долгую карьеру я несколько раз работал с правительством США и вел собственную внештатную практику. В каждом случае мне нужен был способ отслеживать свое время. В случае госконтрактов мне приходится отслеживать набор кодов биллинга с интервалом в 7 мин. Учет времени можно использовать и в более разумных условиях. Я стараюсь вести учет времени, чтобы сопоставить неуловимую историю, используемую в методологиях гибкой разработки, таких как Scrum, с тем, сколько времени потребуется для выполнения задачи. Согласно большинству гибких методологий люди очень плохо умеют оценивать время. Но они довольно хорошо оценивают относительный размер. На рис. 16.7 вам будет сложно определить, какой именно высоты (в дюймах или сантиметрах) находится коробка соевого молока рядом со стаканчиком латте.



Рис. 16.7. Насколько высока упаковка соевого молока?

Ее высота 8 дюймов или 20? Но если вы попытаетесь оценить размер, используя кофейный стаканчик в качестве единицы измерения, вы сможете быть более точными. Высота коробки примерно равна двум с половиной стаканчикам. То же самое и с сюжетными точками, но, в конце концов, босс действительно захочет знать, сколько времени займет то или иное дело. Вы не можете сказать ему, что сделаете задачу за две чашки кофе, и ожидать, что он этим удовлетворится. Это может повлечь за собой очевидный ответ: «Отлично, значит, это будет сделано после *моей* второй чашки кофе!»

Я делаю свою обычную оценку по пунктам истории, затем отслеживаю свое время, чтобы увидеть, сколько времени на самом деле все это заняло, чтобы я мог дать боссу ответ и в следующий раз оценить время лучше.

Чтобы использовать функции отслеживания времени в PyCharm, вам необходимо подключиться к серверу. Под сервером в данном случае я подразумеваю один из сервисов отслеживания проектов, поддерживаемых PyCharm, как показано на рис. 16.8.

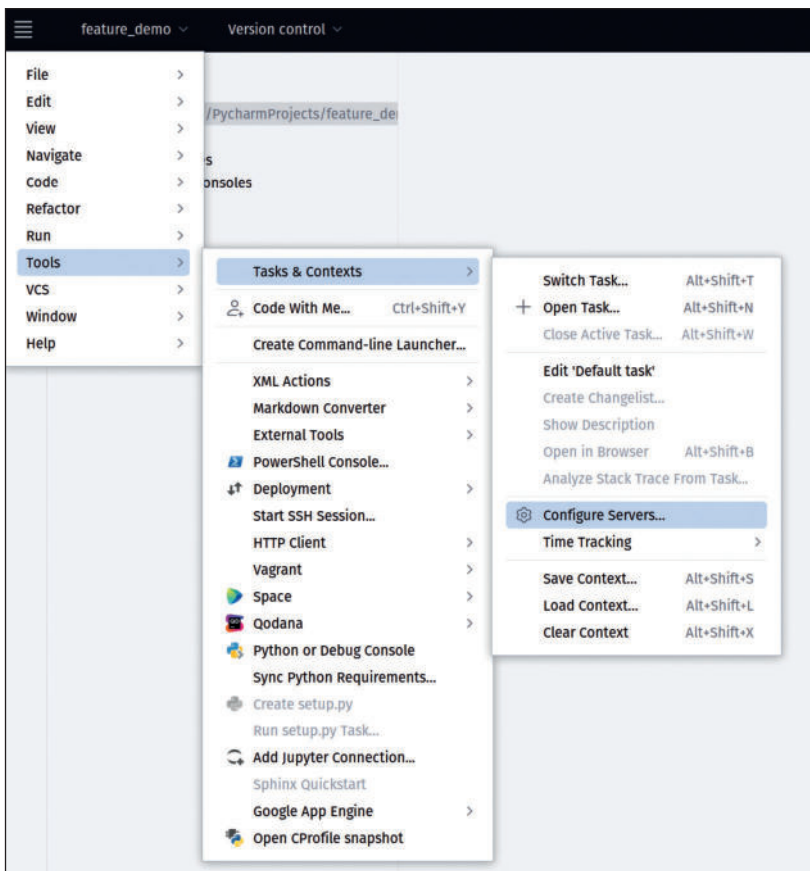


Рис. 16.8. Кликните элемент меню *Configure Servers*, чтобы настроить трекер проекта

При выборе этого пункта меню вы должны увидеть экран, показанный на рис. 16.9.

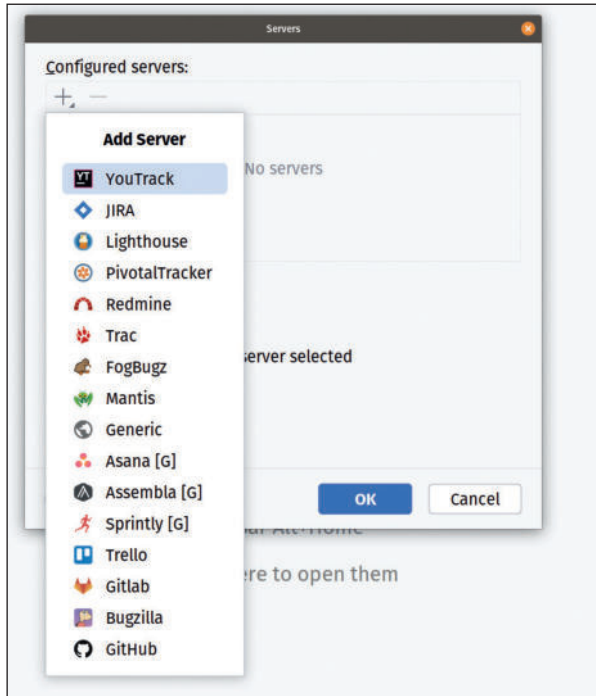


Рис. 16.9. Выберите из списка службу отслеживания проектов, которую вы используете

Как только ваш сервер будет зарегистрирован, вы сможете выбрать элемент своей незаконченной работы или новую задачу, назначенную вам. На рис. 16.10 показана интеграция моего сервера YouTrack с номером выпуска **DEMO-20: Finish** главы 16 в **PyCharm Book**.

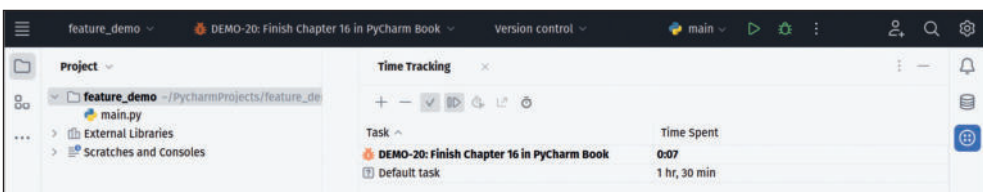


Рис. 16.10. Время, потраченное на решение этой задачи, отслеживается автоматически

Переключаясь между задачами в течение дня, вы можете переключаться прямо в своей IDE, которая будет отслеживать, сколько времени вы тратите на каждую задачу. Затем можете синхронизировать свое рабочее время с сервером, чтобы вести точный учет того, как вы тратите свое время.

TODO – список дел

Может показаться, что это связано с функцией отслеживания времени, но на самом деле это не так. Панель списка TODO отображает все комментарии TODO в вашем коде в виде списка. Пример вы можете увидеть на рис. 16.11.

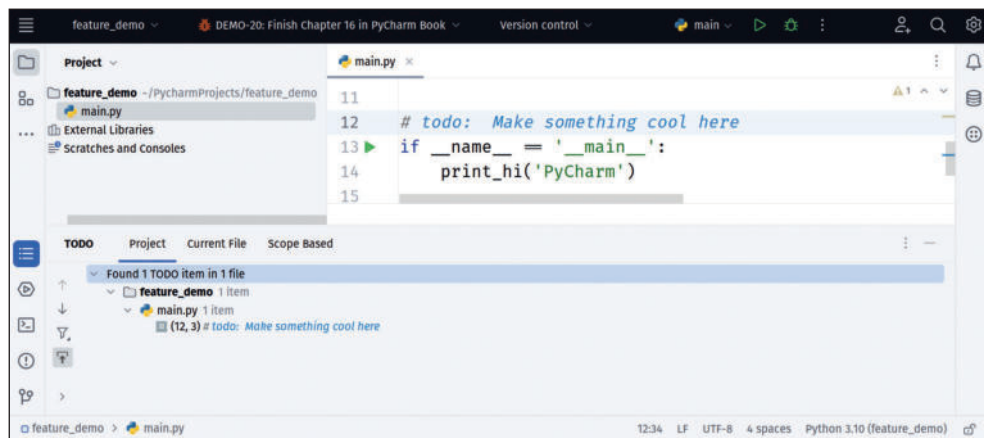


Рис. 16.11. Панель TODO в PyCharm перечисляет все комментарии к задачам в вашем коде и перенаправляет вас прямо к заданной строке при двойном щелчке мыши

Вы можете дважды кликнуть элемент TODO, чтобы перейти прямо к этой строке кода, что позволит вам быстро вернуться к тому, от чего вам пришлось временно отвлечься.

Макросы

Макрос – это скрипт записанных нажатий клавиш, который можно воспроизвести для создания полезной автоматизации. По моему мнению, научиться их использовать – необходимый навык выживания в джунглях разработки программного обеспечения.

Допустим, нам нужен список вещей, которые не изменяются, чтобы заполнить раскрывающийся список в HTML. В этом примере я собираюсь создать тег SELECT со всеми 50 штатами США. Это не изменилось с 1959 года, хотя, если меня когда-нибудь изберут императором, первыми добавлениями в моем списке будут Канада и Мексика. Они не знают, что их ждет.

Вернувшись в реальность, мне нужно это сделать, и я чертовски уверен, что не хочу все это печатать. Я попросил своего лучшего друга ChatGPT дать мне список из 50 штатов в алфавитном порядке. Результат – в us-states.txt в исходном коде главы.

В исходном коде вы также найдете HTML-файл с именем states.html. Я включил следующий код:

```
<form>
  <p>Pick a state:</p>
  <select id="state" name="state">

    </select>
</form>
```

Как видите, в теге <select> нет ничего, что предназначено для отображения раскрывающегося списка выбора на HTML-странице, определенной осталь-

ной частью кода в этом файле. Мне нужно добавить штаты, чтобы они выглядели так:

```
<option>Alabama</option>
<option>Arkansas</option>
```

И так для всех 50 штатов. Я могу сделать это макросом. Конечно, я мог бы просто попросить ChatGPT сгенерировать его для меня, но это было бы неприятно, поскольку это книга о PyCharm, а не о ChatGPT. Откройте проект `feature_demo` в исходном коде главы, затем откройте файл `united-states.txt`. Начните с размещения курсора напротив `Alabama` в файле `united-states.txt`, как показано на рис. 16.12.

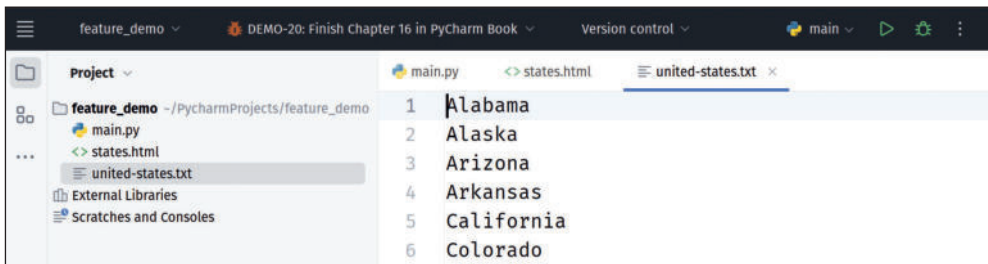


Рис. 16.12. Поместите курсор перед Алабамой

Далее нажмите **Edit | Macro | Start Macro Recording**, как показано на рис. 16.13.

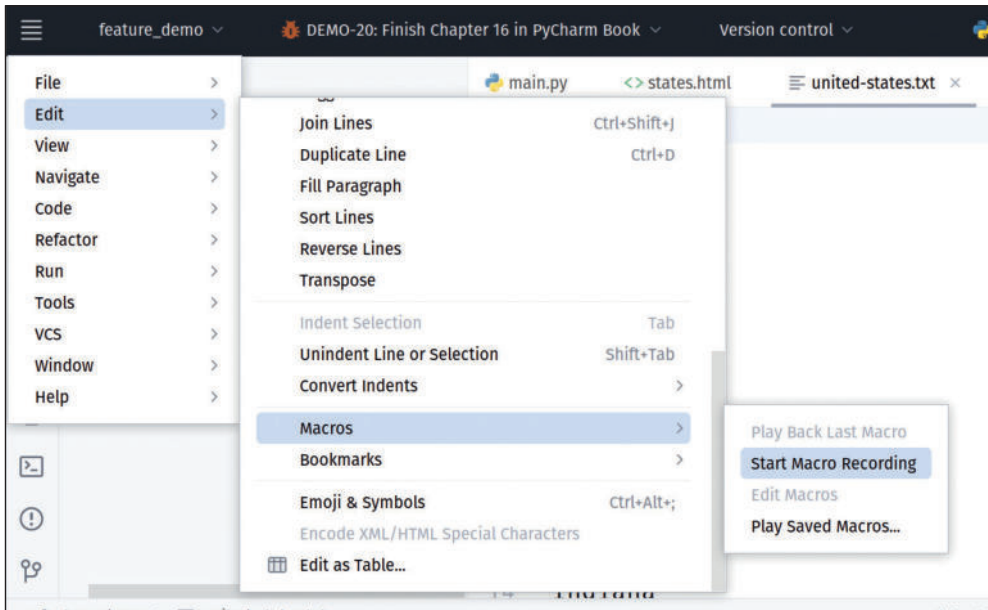


Рис. 16.13. Начните запись макроса

При записи макроса введите `<option>`. Затем нажмите клавишу *End* (*Cmd + стрелка вправо* на Mac) на клавиатуре, чтобы перейти к концу строки, а затем введите `</option>`. Нажмите клавишу *со стрелкой вниз* на клавиатуре, а затем клавишу *Home* (*Cmd + Стрелка влево* на Mac). Курсор теперь должен находиться в начале второй строки, *Arkansas*. Вернитесь к тому же пункту меню, показанному на рис. 16.10, но обратите внимание: **Start Macro Recording** теперь означает **Stop Macro Recording**. Нажмите на него. Вам будет предложено дать имя макросу или оставить его пустым, если он является временным. Я назвал свой `option-list`.

Вернитесь в меню, показанное на рис. 16.10, и вы увидите либо имя сохраненного макроса, либо просто **Play Back Last Macro**. Когда вы нажмете любую опцию, ваши нажатия клавиш будут воспроизведены, и вы перейдете на следующую строку, преобразовав *Arkansas* в `тег option`.

Можете сопоставить свой макрос с комбинацией клавиш и повторять свои действия с помощью сочетания клавиш, что позволяет легко повторять действия снова и снова.

Если честно, функция макросов в PyCharm неплохая, но она не так хороша, как в некоторых других инструментах. *Ultraedit* и *Notepad++*, хотя и не являются IDE, но имеют очень сложные функции макросов, такие как возможность воспроизводить макрос до тех пор, пока он не достигнет конца файла, или запускать макрос определенное количество раз. Макросы PyCharm подходят для небольших задач, но для более крупных можно воспользоваться другими инструментами.

Уведомления

Иногда PyCharm отображает уведомления в виде всплывающих окон в правом нижнем углу экрана. Эти сообщения обычно исчезают через несколько секунд, но вы можете увидеть их на панели уведомлений. Вы найдете эту панель на правой панели инструментов, как показано на рис. 16.14.

Можете переключить панель, кликнув значок *колокольчика* на правой панели инструментов. На панели есть опция **Timeline**, которая позволяет отображать уведомления по давности. Вы можете отклонить любые уведомления, которые больше не нужны, или очистить их все.

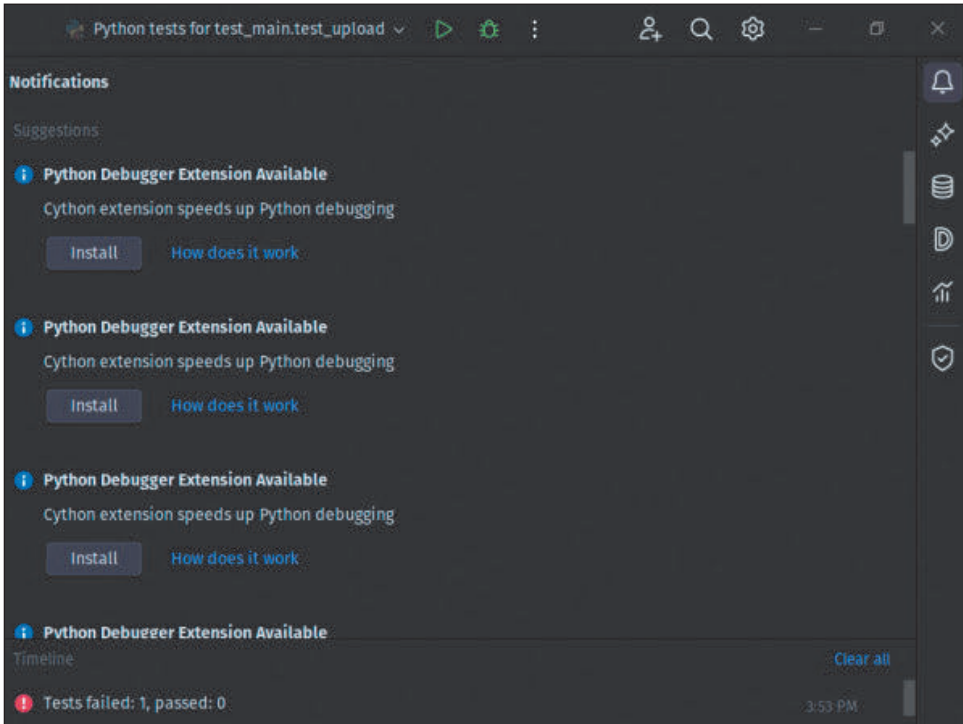


Рис. 16.14. Панель уведомлений

НОВЫЕ ВОЗМОЖНОСТИ ВЕРСИИ 2023.2

За время написания этой книги многое изменилось, особенно в пользовательском интерфейсе. Я использовал сборку #PY-231.8109.197, или, как мне нравится ее называть, *боевой 197-й*. Я все время поддерживал одну и ту же версию, не устанавливая никаких обновлений, потому что хотел, чтобы все было согласованно. Примерно в то время, когда я начал работать над главой 13, JetBrains провела онлайн-мероприятие, на котором представила новую версию – 2023.2. Новая версия наполнена новыми функциями, но сегодня, когда я пишу главу 16 (эту главу), вышли еще две важные версии. Такой темп, вероятно, во многом обусловлен обновленным пользовательским интерфейсом JetBrains.

Если вы помните, в начале книги нам пришлось включить «new UI» в настройках. К тому времени, как вы это прочитаете, новый пользовательский интерфейс будет использоваться по умолчанию. JetBrains не остановилась на достигнутом. Пользовательский интерфейс всегда имел широкие возможности настройки, и мы подробно рассмотрели это в первых главах книги.

Одним из основных направлений работы над новым пользовательским интерфейсом стала оптимизация процесса разработки. Большинство IDE проходят цикл роста, начиная с простого и целенаправленного пользовательского интерфейса, который объединяет минимальные функции, необходимые для того, чтобы считаться IDE, а не расширенным текстовым редактором. Со временем добавляется все больше и больше функций с целью предоставить вам

инструмент, который делает все, что может понадобиться. Буквально на днях мне пришло в голову, что в те дни, когда я на 100 % сосредоточен на разработке Python или даже JavaScript, веб-разработке или разработке баз данных, все, что мне действительно нужно, – это легкая ОС и PyCharm. Мне даже не понадобится оконный менеджер, кроме того, что необходим для запуска PyCharm!

Проблема, вызванная богатством такого инструмента, заключается в том, что он становится устрашающим для новых пользователей и неорганизованным для опытных пользователей. В главе 15 мы узнали, что большинство новых функций любой IDE JetBrains реализованы в виде плагинов. Это может привести к неорганизованному пользовательскому интерфейсу и создать впечатление, будто IDE представляет собой кучу несвязанных друг с другом частей, собранных вместе, что дает не самый лучший опыт пользования.

Новый пользовательский интерфейс призван решить эту проблему, делая его более лаконичным, предоставляя наиболее часто используемые инструменты и скрывая (но не удаляя) более сложные параметры. Сборка, которую я использовал для книги, справилась с этой задачей очень хорошо. По мере выхода новых выпусков я замечаю дальнейшие попытки сгладить сложность. Взгляните на рис. 16.15.

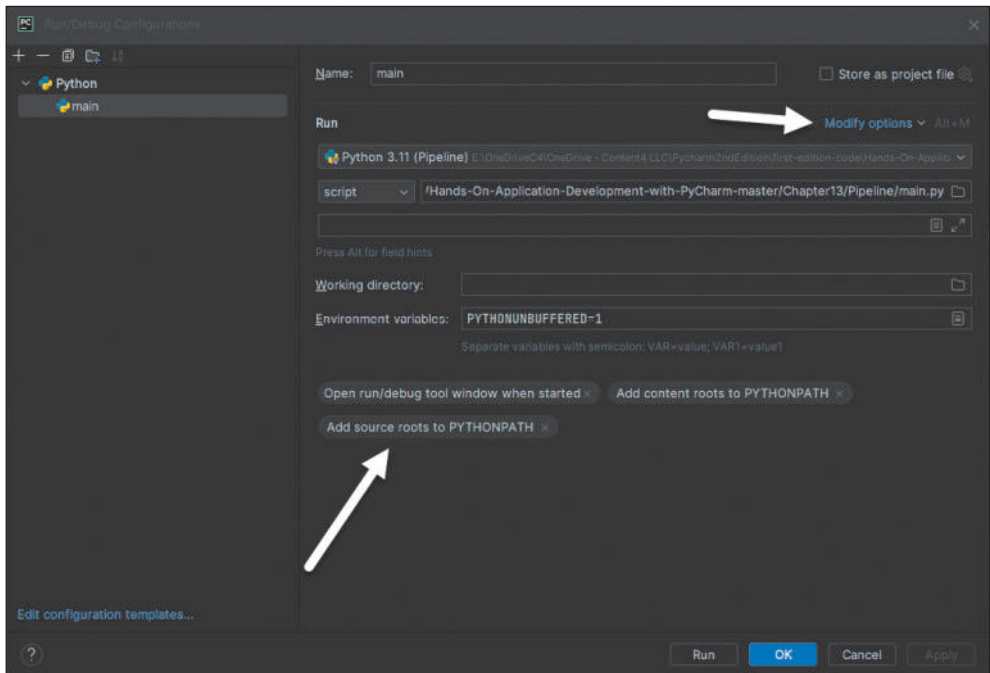


Рис. 16.15. JetBrains переместил множество настроек в теги вместо десятков флажков

Это конфигурация запуска из последнего обновления, выпущенного всего несколько дней назад и помеченного как сборка № PY-232.9559.58, созданного 22 августа 2023 года. Есть два видимых различия. Внизу имеется набор тегов и раскрывающийся список **Modify Options**, оба обозначены стрелками на рис. 16.15. Сравните это с рис. 16.16.

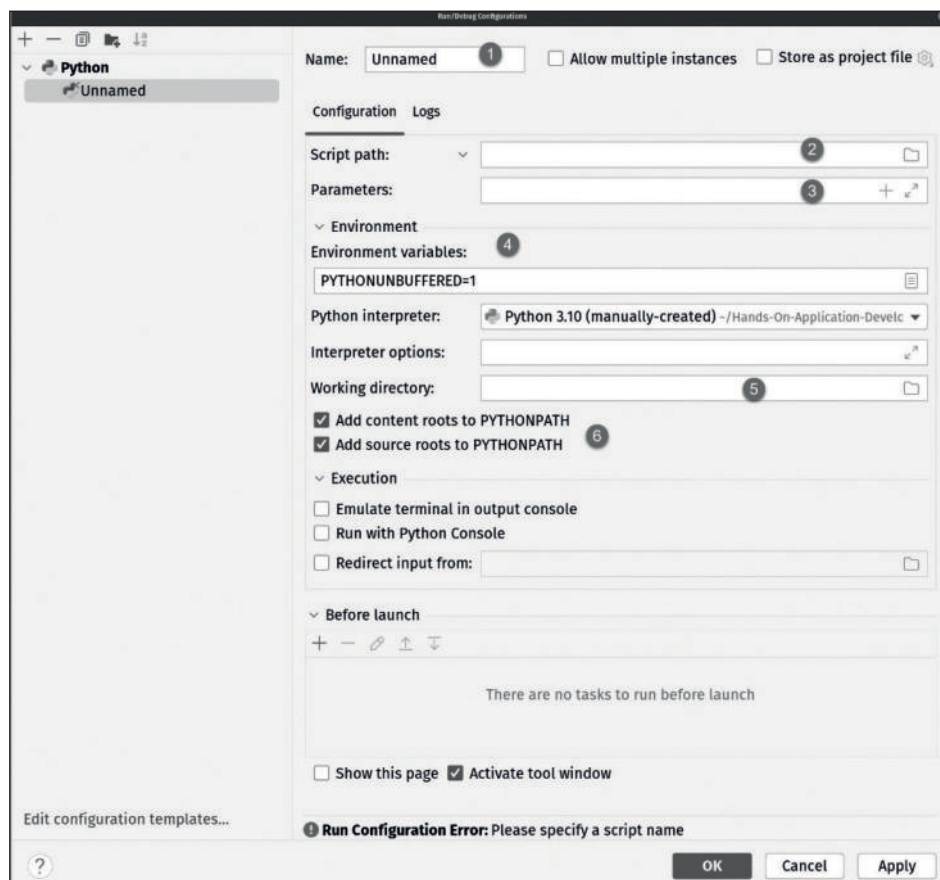


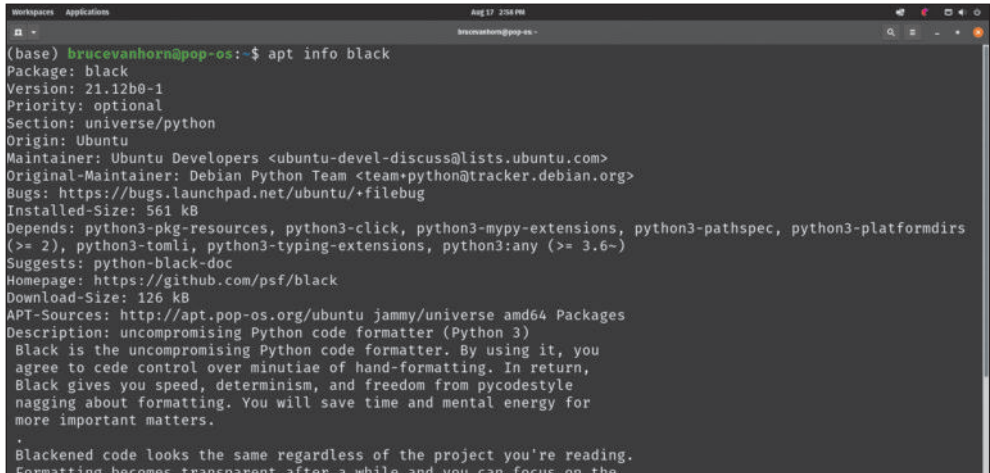
Рис. 16.16. Это старый диалог из главы 3

Все эти параметры по-прежнему доступны, но они были сведены в раскрывающийся список **Modify Options**. Настройки, которые вы фактически использовали, отображаются в виде тегов внизу рис. 16.15. Количество изменений, произошедших всего за несколько месяцев, ошеломляет!

Помимо изменений продукта, есть несколько интересных новых функций и улучшений.

Интеграция с Black

Black – бескомпромиссное средство форматирования кода, впервые представленное на PyCon 2019. С момента своего выпуска он приобрел большое внимание и популярность, а JetBrains добавила поддержку PyCharm. Для работы этой функции необходимо установить Black. Подробности об установке и многое другое можно найти на странице <https://github.com/psf/black>. Если вы используете Linux, проверьте менеджер пакетов. Я использую Pop_OS, основанную на Ubuntu. На рис. 16.17 показано, что я нашел Black в виде пакета и предпочитаю установить его таким образом.



```
(base) brucevanhorn@pop-os:~$ apt info black
Package: black
Version: 21.12b0-1
Priority: optional
Section: universe/python
Origin: Ubuntu
Maintainer: Ubuntu Developers <ubuntu-devel-discuss@lists.ubuntu.com>
Original-Maintainer: Debian Python Team <team+python@tracker.debian.org>
Bugs: https://bugs.launchpad.net/ubuntu/+filebug
Installed-Size: 561 kB
Depends: python3-pkg-resources, python3-click, python3-mypy-extensions, python3-pathspect, python3-platformdirs
(>= 2), python3-tomli, python3-typing-extensions, python3:any (>= 3.6~)
Suggests: python-black-doc
Homepage: https://github.com/psf/black
Download-Size: 126 kB
APT-Sources: http://apt.pop-os.org/ubuntu jammy/universe amd64 Packages
Description: uncompromising Python code formatter (Python 3)
 Black is the uncompromising Python code formatter. By using it, you
 agree to cede control over minutiae of hand-formatting. In return,
 Black gives you speed, determinism, and freedom from pycodestyle
 nagging about formatting. You will save time and mental energy for
 more important matters.
 .
 Blackened code looks the same regardless of the project you're reading.
 Formatting becomes transparent after a while and you can focus on the
```

Рис. 16.17. В Linux вы, вероятно, можете установить Black с помощью менеджера пакетов

После установки Black вам необходимо включить его для каждого проекта. Вы найдете эту опцию при поиске Black в диалоговом окне **Settings**, показанном на рис. 16.18.

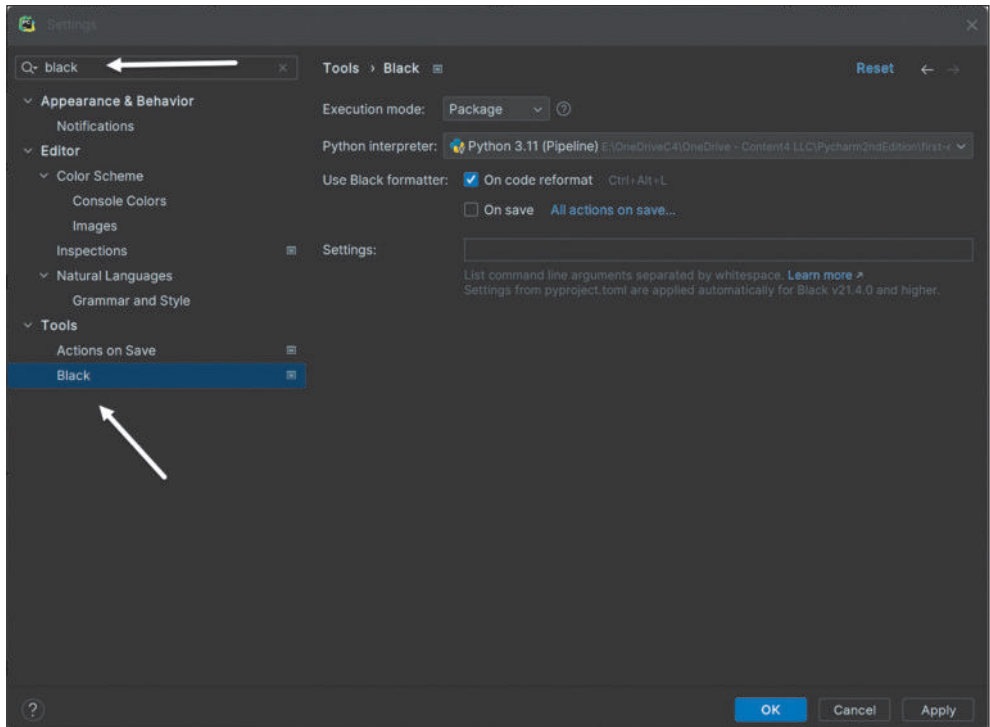


Рис. 16.18. Найдите Black в диалоговом окне Settings, и вы найдете его в разделе Tools

Возможные варианты форматирования: **On code reformat** и **On save**. Если вы работаете в команде и у вас есть существующая база кода, используйте его с осторожностью. Одна из целей разработки Black – последовательно форматировать ваш код, чтобы уменьшить различия во время слияний и проверок кода. В первый день использования вы, скорее всего, столкнетесь с огромными различиями. Я объясню почему. Допустим, у меня есть ветка в Git под названием `development`. В моей команде мы индивидуально разделяем разработку на функциональные ветки. Если я работаю над какой-то новой функцией, я прекращаю `development` и создаю новую ветку под названием `feature/amazing-thing`. Дальше я создам свою потрясающую вещь, но мне нужно будет попутно коснуться и некоторых других файлов, написанных другими разработчиками. Естественно, прежде чем я сделаю фиксацию, мышечная память автоматически использует команду `Ctrl + Alt + L` для переформатирования. Black переформатирует весь файл. Когда вы перейдете к объединению, можете обнаружить, что другие разработчики изменили некоторые из тех же файлов, которые вы переформатировали, но они были менее добросовестны в переформатировании. Вам предстоит столкнуться с проблемой некорректного слияния, даже если между двумя разработчиками на самом деле было изменено всего несколько строк, и в противном случае они могли бы не столкнуться.

Это не вина формatera Black! То же самое произойдет, если вы используете форматар кода PyCharm по умолчанию. Тем не менее форматар Black является лучшим решением, поскольку он нацелен на детерминизм – модное слово для кодеров, означающее, что запуск операции всегда дает один и тот же результат. Бескомпромиссный детерминированный форматар должен давать очень стабильные результаты при каждом использовании.

Мой совет: если вы хотите начать использовать Black, соберитесь в команду. Пусть каждый продвинет свою работу, объединит ее в ветку, а затем применит Black ко всему проекту. В моем примере я бы разрешил свои конфликты, а затем снова слился бы с веткой разработки. Тогда я бы попросил всех своих коллег сделать то же самое. Как только в разработке появятся все изменения, заморозьте код и используйте Black для всех исходных файлов. Во время заморозки никому не разрешается работать с каким-либо отдельным файлом до тех пор, пока форматирование не будет применено и не отправлено. Если у вас очень большая база кода, возможно, вам придется со временем систематизировать свои усилия. Результаты будут того стоить! Ваш код будет совместим с PEP-8, и вы увидите меньше конфликтов слияния, особенно вызванных форматированием.

Black можно настроить для запуска при сохранении, а за пределами PyCharm Black можно использовать как привязку в вашем контроле версий, так и в вашем процессе CI/CD.

Интеграция с GitLab

GitHub – самое популярное и широко используемое облачное решение для управления кодом. Его ахиллесовой пятой является тот факт, что он основан на облаке. Организации, которые серьезно относятся к безопасности, например те, чьи торговые партнеры требуют сертификации, такой как сертификация **Service Organization Control 2 (SOC2)**, часто не удовлетворены тем, что их

интеллектуальная собственность размещается на общедоступной платформе. Хотя вы действительно можете размещать частные репозитории на GitHub, сам факт того, что файлы находятся не на сервере, контролируемом сертифицированной организацией, может стать проблемой.

К счастью, с помощью GitLab можно разместить собственный частный сервер GitHub. Вы можете запустить GitLab локально за брандмауэром и осуществлять полный контроль над своей безопасностью и инфраструктурой, при этом используя то, что во всех других смыслах является опытом GitHub. Интеграцию с GitLab вы найдете в настройках **Version Control**, как показано на рис. 16.9.

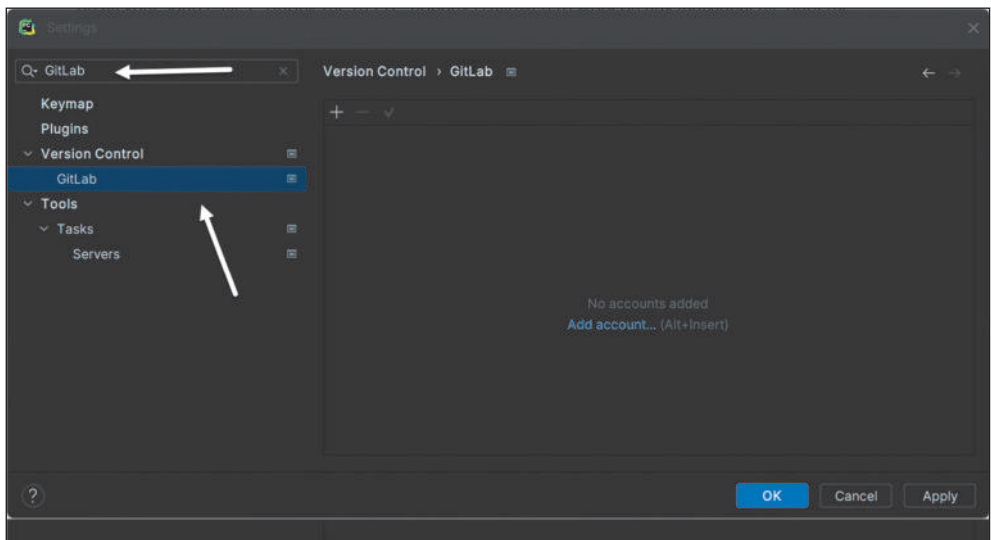


Рис. 16.19. Настройки GitLab можно найти рядом с другими настройками VCS

Интеграция достаточно глубокая. Помимо основ, которые мы рассмотрели в главе 5, вы можете обрабатывать все свои запросы на получение, утверждения и многое другое прямо из IDE.

Запускайте все что угодно!

У нас уже давно есть **Search everywhere**. Если вы пропустили это, можете дважды нажать клавишу **Shift** и открыть диалоговое окно, которое позволит вам найти что-нибудь в любом месте вашего проекта. Это работает, даже если то, что вы ищете, не находится в коде. Это может быть в документации, настройках или где-то еще.

Новым в PyCharm, хотя и не обязательно новым для IntelliJ является **Run Anything**, активируемый двойным нажатием клавиши **Ctrl**. В пользовательском интерфейсе вы также найдете подсказки, освещающие эту новую (для PyCharm) функцию. На рис. 16.20 вы увидите, что я дважды кликнул клавишу **Ctrl**, что вызвало небольшое диалоговое окно. Поскольку я новичок в этой функции, я ввел в диалоговом окне вопросительный знак (?).

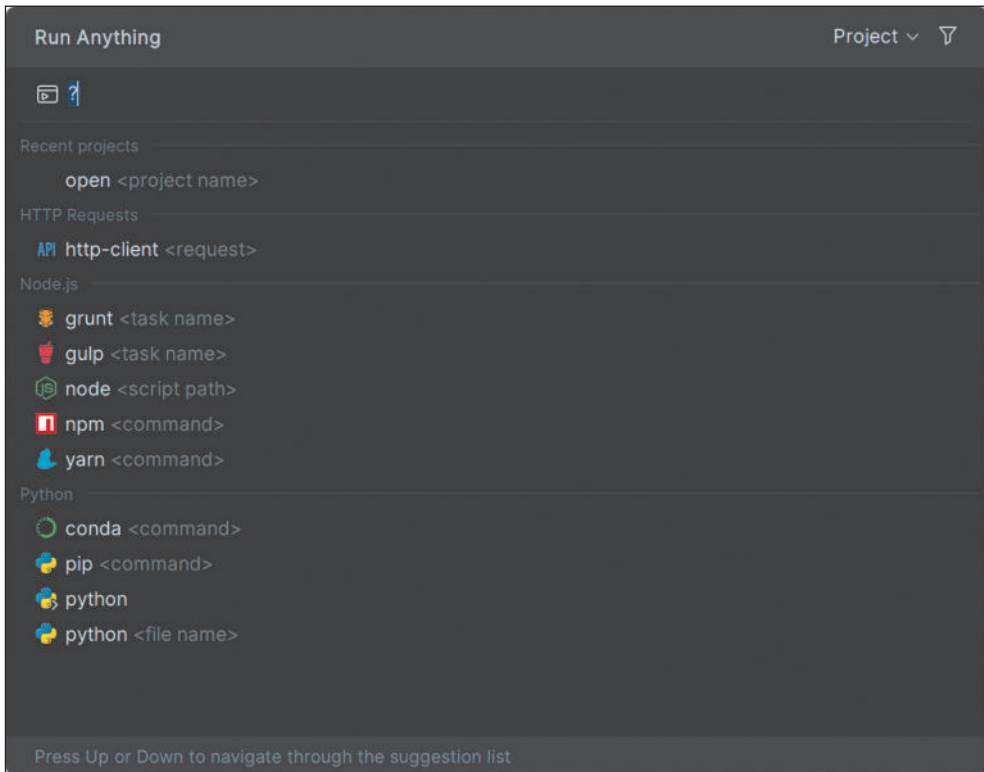


Рис. 16.20. Дважды кликните клавишу Ctrl, чтобы открыть диалоговое окно Run Anything

Приятно услышать: «Вы можете запустить ВСЕ ЧТО УГОДНО!» Но все вы, критически мыслящие люди, сразу же спросите: «Что вы подразумеваете под ВСЕ ЧТО УГОДНО?» Здесь ответ на то, что можно запустить. Мне понятно, что я могу запустить любой скрипт Python в своем проекте, просто введя его имя. Мне понятно, что я также могу запускать команды `pip` и `conda`. Веб-разработчики могут запустить тестовый код HTTP-запроса, который мы рассмотрели в главе 9. Они также могут запускать все популярные менеджеры пакетов, такие как **npm** и **Yarn**, а также распространенные инструменты сборки, включая **gulp** и **grunt**. Эти два инструмента были заменены **webpack** в большинстве современных цепочек инструментов, но на самом деле нам не нужна конфигурация для этого, поскольку большинство людей запускают веб-пакет из конфигурации в файле `package.json` через `npm`, который явно поддерживается. Естественно, мы также можем запустить любой процесс **Node** так же легко, как запускаем Python.

AI-помощник

Гвоздем большинства недавних мероприятий с участием PyCharm стал AI Assistant. На момент написания этой статьи эта функция находится в стадии закрытого бета-тестирования, а это означает, что вам необходимо зарегистрироваться и получить одобрение, чтобы опробовать эту функцию. Имейте в виду, что все, что я здесь показываю, вероятно, изменится.

Самый простой способ реализации (и существует множество плагинов, которые делают это) – это просто предоставить интегрированный интерфейс для некоторого онлайн-API AI.

Напротив, вы найдете AI Assistant, интегрированный в PyCharm. Чтобы использовать AI Assistant, используйте правое меню, показанное на рис. 16.21.

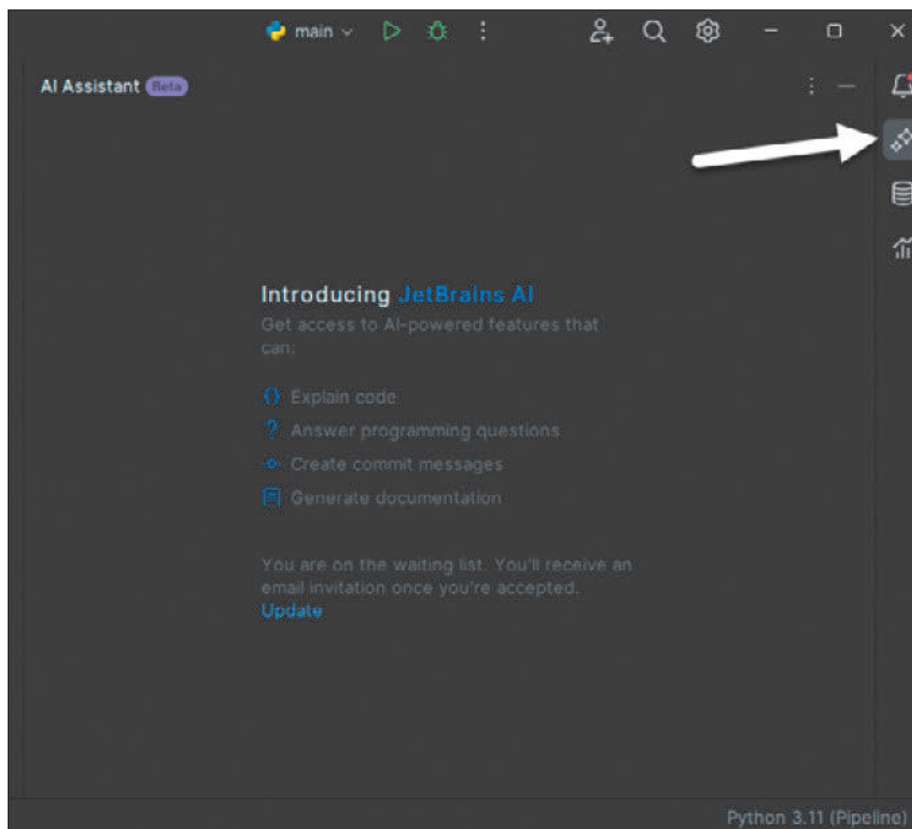


Рис. 16.21. AI Assistant можно активировать с помощью кнопки в правом меню

В предыдущем диалоговом окне обобщаются доступные функции. Давайте взглянем.

Эта функция не является локальной

Если в вашей компании или среде есть ограничение на отправку кода в API, например ChatGPT, вам необходимо знать, что эта функция делает именно это. Функциональность не является автономной в PyCharm, и **любой вводимый вами код или информация будут отправлены различным внешним сторонним API**. Это задокументировано в соглашении, которое вы слепо прокрутили и проигнорировали, чтобы иметь возможность использовать PyCharm и новые интересные функции искусственного интеллекта.

Объяснение кода

Существует множество обстоятельств, при которых вы можете столкнуться с кодом, который не понимаете. Мой обычный способ описания этих обстоятельств довольно красочный, особенно в два часа ночи накануне крайнего срока. Я постараюсь сохранить профессионализм.

- Это написал кто-то, кто явно понятия не имеет, что делает.
- Это написал кто-то, кто явно разбирается в этом лучше, чем я. Это может быть один и тот же человек, в зависимости от времени суток.
- Это было написано человеком, который любит лаконичный код, несмотря на то что это антипаттерн в Python.
- Код содержит регулярное выражение¹, которое вы не запомнили, поскольку ни один нормальный человек не понимает регулярные выражения, даже если он и притворяется таковым.

В подобных обстоятельствах помощь находится на расстоянии одного клика мыши. Выделите соответствующий код, кликните правой кнопкой мыши и выберите **AI Assistant Action | Explain code**. Через несколько секунд AI предоставит вам информацию о выбранном коде.

Примечание

Имейте в виду, что AI-помощник не всегда прав и не всегда дает адекватные комментарии.

Ответы на вопросы по программированию

По правде говоря, AI Assistant ответит на любой ваш вопрос. Я задавал ему вопросы о конфигурации nginx, Docker и общие вопросы о сети. Конечно, он также отвечает на вопросы по программированию.

Создание сообщений о коммитах

Этот действительно круто! Я большой поклонник онлайн-комиксов XKCD², и отношение к Git во многих комиксах просто идеальное. Вот что приходит на ум прямо сейчас: <https://xkcd.com/1296/>.

Поскольку IDE может видеть ваш список изменений, она может сгенерировать для вас сообщение о коммитах, как показано на рис. 16.22.

¹ Регулярное выражение в Python – это последовательность символов, описывающая текстовые шаблоны. – *Прим. ред.*

² XKCD – это веб-комикс о романтике, сарказме, математике и языке. – *Прим. ред.*

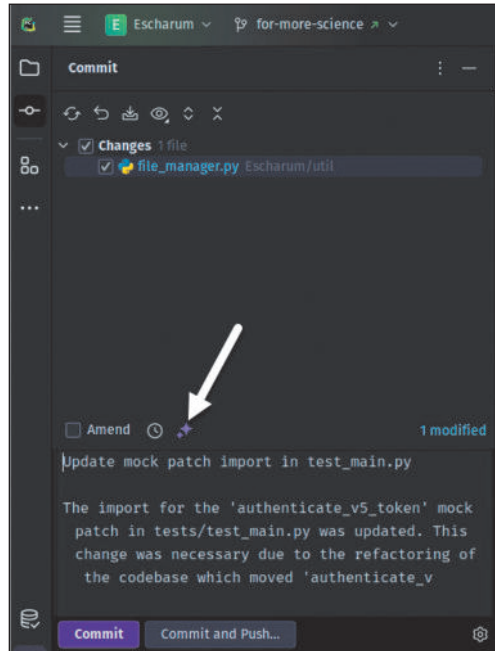


Рис. 16.22. Генерируются сообщения о коммитах, которые являются более глубокими и содержательными, чем «Я изменил кучу вещей»

Возможно, вы не придаете особого значения сообщениям о коммитах. Я работал над проектами правительства США, где проверялись сообщения о коммитах. Об этом обычно не знала команда разработчиков, которая, узнав об этом, решила перебазировать и воссоздать каждый коммит, чтобы удалить свой более красочный язык из сообщений коммита. Аудит на самом деле не стремился к этому; он использовался для проверки процесса контроля изменений. Моя команда в настоящее время проходит сертификацию SOC2, и я могу сказать вам из первых рук, что полные и описательные сообщения о коммитах очень полезны. Поскольку кибербезопасность продолжает вторгаться в рабочие процессы разработчиков, можно поспорить, что эта функция рано или поздно докажет вам свою ценность.

Создание документации

Вы ненавидите писать строки документации? Это вопрос с подвохом. Если вы разработчик программного обеспечения, вы, вероятно, думаете, что сам код – это вся документация, которая может кому-либо понадобиться. Но вы также увидели силу хорошо документированного кода при использовании функций, документации как в основном пользовательском интерфейсе, так и в SciView. Разве было бы плохо, если бы ваши собственные функции были также хорошо документированы, и вам не нужно было бы писать эту документацию самостоятельно? На рис. 16.23 показано, как я генерирую строку документации для функции в одном из моих проектов.

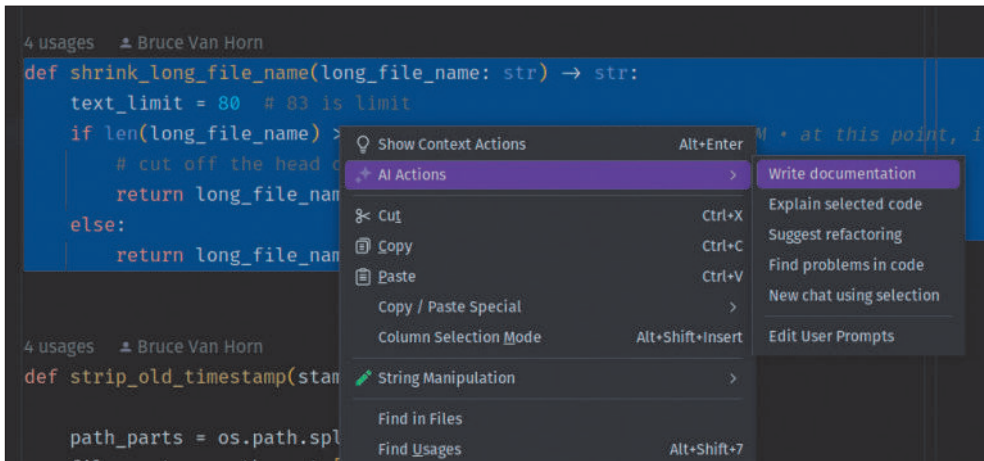


Рис. 16.23. Не пишите документацию, а генерируйте ее

Документация в этом случае на самом деле представляет собой еще одну форму шаблона. Вы должны позволить IDE сгенерировать ее за вас, а затем, возможно, улучшить то, что она создала, если это необходимо.

Поддержка Jupyter Notebook для Polars

Хотя pandas хороши, для больших данных они слишком медленны. Spark и другие платформы больших данных могут помочь, но только за счет горизонтального масштабирования, которое предполагает настройку нескольких вычислительных узлов в вашей сети. **Polars** – это библиотека обработки данных, позволяющая вам работать с большими объемами данных, не прибегая к использованию нескольких компьютеров, и она уже давно является востребованной функцией в PyCharm. Если ваш DataFrame помещается в памяти вашего компьютера, можете использовать его в Polars и просматривать в PyCharm так же, как и любой другой DataFrame.

Кроме того, PyCharm поддерживает библиотеку **Plotly** для визуализации, которая уникально работает с датафреймами Polars. Объединив поддержку этих расширенных библиотек с PyCharm, вы сможете преодолеть многие препятствия, которые недоброжелателями обычно называют недостатками использования Polars.

РЕЗЮМЕ И ЗАКЛЮЧИТЕЛЬНЫЕ ЗАМЕЧАНИЯ

За короткое время мы сделали многое! По крайней мере, мне так кажется, поскольку во многих из этих глав было чрезвычайно заманчиво перейти к учебнику по теме, которую мы освещали. Например, в главах Flask и Django я чувствовал себя «не в своей тарелке», не предоставив вам руководство по Flask и Django. Я не сделал этого, потому что это на самом деле отдельные книги как внутри библиотеки Packt, так и за ее пределами.

Мы узнали, что цель IDE – предложить вам полный набор инструментов для каждой задачи разработки, которую вы можете выполнять ежедневно. Если

смотреть через эту призму, PyCharm – это потрясающая IDE, у которой действительно очень мало конкурентов.

Мы увидели, как основы настройки простого проекта Python становятся проще благодаря автоматизации таких процессов, как создание виртуальной среды. Профессиональная версия PyCharm имеет множество дополнительных функций для создания проектов, в основном сосредоточенных на шаблонах проектов для наиболее популярных типов проектов в области общей разработки программного обеспечения, полнофункциональной веб-разработки и обработки данных. Существует даже специальная версия для образовательного использования, позволяющая создавать интерактивные уроки, которые воспроизводятся непосредственно в IDE. Об этом мы не говорили, но, если вам интересно, я оставлю ссылку в разделе «Дальнейшее чтение».

Полнофункциональные веб-разработчики могут использовать огромную часть инструментов PyCharm, предназначенных только для их работы. Мы видели, что поддерживаются все основные платформы, и поддержка даже не ограничивается Python. PyCharm может делать все, что может делать WebStorm, в отношении разработки JavaScript и работы с современными фреймворками пользовательского интерфейса JavaScript, такими как Angular, Vue, Vite и React.

PyCharm обладает действительно полным набором инструментов для взаимодействия с реляционными и нереляционными базами данных. Этот набор функций в основном взят из DataGrip, другой IDE от JetBrains, полностью ориентированной на базы данных, в частности на создание и разработку реляционных структур. Он имеет полную поддержку MongoDB и Redis, которые я использую ежедневно. Обычно мне приходится использовать такие инструменты, как Studio 3T или Redis Insights, вместо этого я могу просто остаться в PyCharm. Разнообразие поддерживаемых платформ ошеломляет! Существуют десятки поддерживаемых баз данных, и иногда может быть интересно просто просмотреть список и найти те, о которых вы никогда не слышали.

Рассмотрев базы данных, мы узнали о преимуществах, которые PyCharm приносит в область науки о данных. Вы можете использовать то, что я назову облегченной версией Jupyter Notebooks, используя ячейки кода в IPython. Я не занимаюсь ежедневным анализом данных на работе, но иногда мне нравится притворяться, что так и есть. Функции ячеек кода, описанные в главах 12 и 13, полезны для таких задач, как интеграция нескольких API или микросервисов. Я могу создать прототип скрипта, используя ячейки, которые извлекают данные из различных API или микросервисов, перебросить их в DataFrame pandas, и я могу использовать SciView, чтобы увидеть результаты, даже если мне не нужно сохранять вызовы DataFrame в моей реальной программе. Простая возможность визуализировать данные в нескольких простых диаграммах, расположенных рядом друг с другом, может оказаться большим подспорьем.

Если вы занимаетесь наукой о данных, ваши возможности возрастают благодаря таким инструментам, как интеграция с Jupyter Notebooks, что делает PyCharm универсальным инструментом для всего вашего рабочего процесса. В этом суть инструмента для трех разных групп разработчиков.

Когда Packt попросил меня написать эту книгу, я был взволнован и напуган. Я был взволнован, потому что только что узнал, что LinkedIn Learning удалил мой курс PyCharm, а это был один из любимых курсов. Эта книга дала мне воз-

возможность погрузиться глубже, чем вы можете сделать в двухчасовом видеокурсе. Если вам нравятся двухчасовые видеокурсы, у меня остался один в LinkedIn, посвященный разработке API с помощью Flask. Сомневаюсь, что они продержатся долго, поэтому, если вам интересно, обязательно посмотрите, прежде чем его снимут. Я помещу ссылку в раздел «Дальнейшее чтение». Мои новые видеокурсы можно найти на сайте <https://maddevskilz.com>. Естественно, все, что связано с Python, будет использовать PyCharm в качестве IDE.

Идея (.idea?) написания этой книги была пугающей, потому что в эту IDE упаковано (Pack?) так много всего, и я действительно знаю то, чего не знаю. Я не знаю всего. Я определенно нашел функции, о существовании которых не подозревал, когда начинал. Наверное, я что-то неправильно понял. Это нормально (я надеюсь). Завершение этой книги – это отдельный момент в путешествии, которое для меня началось семь лет назад, когда мне надоела Visual Studio и веб-разработка на C# в целом. Когда я увидел, как легко создавать приложения в Flask, а затем еще раз, как легко PyCharm позволяет таким простым людям, как я, выполнять настоящую работу, меня зацепило. Написание такой книги – огромная ответственность, потенциально имеющая огромное значение. Программирование – это легальная суперсила. В отличие от большинства сверхспособностей в фильмах, этой сверхспособности можно научить. Не обязательно прилетать с далекой планеты, быть укушенным радиоактивным пауком или вколотся экспериментальной сывороткой. Хотя кое-что из этого может помочь, все, что вам действительно нужно, – это любопытство, настойчивость и упорный труд. Если вы хотя бы немного овладеете этим ремеслом, то сможете творить великие дела.

У меня была возможность создать программное обеспечение, которое помогает выбирать методы лечения пациентов с раком простаты. Проект ни к чему не привел, но через год после того, как я отложил проект, мне позвонил отец и сказал, что у него только что диагностировали рак простаты. Дополнительная мораль этой истории – никогда не удалять свои старые репозитории! Мой отец сам был патологоанатомом, и с помощью врача, создавшего алгоритм, мы прогнали данные моего отца через программу, и схема лечения, полученная в результате, дала мне еще 8 лет жизни с ним.

Я также участвовал в проектах, призванных помочь обеспечить безопасность моей страны, обучать пилотов Корпуса морской пехоты США и управлять ИТ-активами. Когда я работал над этими проектами, это были просто проекты. Были даны сроки, в которые мне нужно было уложиться, и это было источником стресса. Но каждая опубликованная часть программного обеспечения, даже каждая строка кода, которую мы пишем, потенциально могут помочь сделать мир вокруг нас лучше! Иногда преимущества очевидны, как в случае с программным обеспечением для лечения рака. Даже программное обеспечение с обыденными целями, такое как планирование емкости склада предприятия, может иметь далеко идущие преимущества, поскольку оно также является инструментом, позволяющим многим делать то, что они делают, и создавать то, что они создают.

Учтите также, что наша суперспособность работает в большом масштабе. Супермен может находиться одновременно только в одном месте; этот факт слишком часто используют различные суперзлодеи. Мы же – не ограничены

в этом плане. Во многих магазинах приложений есть приложения, которые могут помочь детям с дислексией научиться читать. Это одно приложение, загруженное десятки тысяч раз, может дать образование поколениям, которые были бы маргинализированы или проигнорированы системами государственного образования большинства стран, которые ценят результаты тестов и финансирование выше результатов.

У вас есть суперсила! Используйте это! Эти инструменты облегчают задачу. Если я зацепил хотя бы одного человека и он получит пользу от этого инструмента и этого языка, которые я получил с тех пор, как начал, то для меня это означает, что книга достигла бешеного успеха.

ДАЛЬНЕЙШЕЕ ЧТЕНИЕ

Обязательно посетите следующие полезные ресурсы:

- Branton, A. (2018) *Hands-On DevOps with Vagrant*. Packt,
- RESTful APIs with Python 3 and Flask (linkedin.com): <https://www.linkedin.com/learning/building-restful-apis-with-flask/restful-apis-with-python-3-and-flask-4>,
- Polars library: <https://www.pola.rs/>,
- The Plotly Library for Python: <https://plotly.com/python/>,
- Educational edition of PyCharm: <https://www.jetbrains.com/pycharm-edu/>,
- HashiCorp Vagrant: <https://www.vagrantup.com/>,
- HashiCorp Terraform: <https://www.terraform.io/>,
- Docker Desktop: <https://www.docker.com/products/docker-desktop/>,
- Oracle VirtualBox: <https://www.virtualbox.org/>,
- JetBrains AI Assistant: <https://blog.jetbrains.com/idea/2023/06/ai-assistant-in-jetbrains-ides/>.

Предметный указатель

А

активные серверные страницы
(ASP Classic) 292

В

ветвление (branching) 202
виртуальная машина 94, 551, 590
виртуальная среда 94, 122, 587
внедрение зависимостей (DI) 217
выражения немедленно вызванной
функции (IIFE) 274
вычисления с сокращенным набором
команд (RISC) 294

Г

глобальная блокировка
интерпретатора (GIL) 94
графический пользовательский
интерфейс (GUI) 28

З

завершение хиппи (hippie
completion) 138
защищенное копирование (SCP) 258

И

индекс пакетов Python (PyPI) 104
интеграционное тестирование 213
интегрированная система контроля
версий (VCS) 55
интегрированная среда разработки
(IDE) 28, 132
интерфейс общего шлюза (CGI) 291
интерфейс прикладного
программирования (API) 28
искусственный интеллект 35, 133, 339

К

каскадные таблицы стилей (CSS) 259
контроль источников (source control) 177

Л

лицензионное соглашение с конечным
пользователем (EULA) 60

М

магическая команда 481
машинное обучение (ML) 499
менеджер пакетов Node (NPM) 108
менеджер пакетов узлов (npm) 258
модель-представление-контроллер
(MVC) 50
модульное тестирование 213, 221, 334

Н

намерения (intentions) 150

О

общий интерфейс шлюза (CGI) 108
объектная модель документа (DOM) 260
объектно ориентированное
программирование (OOP) 137
одностраничные приложения (SPA) 320

П

передача репрезентативного
состояния (REST) 324
препроцессор гипертекста (PHP) 292
принцип единой ответственности
(SRP) 532
принцип единственной
ответственности (SRP) 215
принцип инверсии зависимостей
(DIP) 215
принцип открытости-закрытости
(OCP) 215
принцип подстановки Лискова (LSP) 215
принцип разделения интерфейса
(ISP) 215
программа раннего доступа (EAP) 58
протокол безопасной передачи файлов
(SFTP) 48
протокол передачи гипертекста
(HTTP) 108
протокол передачи файлов
(FTP) 48, 258
протокол управления передачей/
интернет-протокол (TCP/IP) 295

Р

разработка через поведение (BDD) 213
 разработка через тестирование (TDD) 213
 распределенные системы контроля версий (DVCS) 178

С

серверные страницы Java (JSP) 292
 система контроля версий (VCS) 176, 177
 сквозное тестирование 213
 сортировка пузырьком (bubble sort) 45
 среда выполнения (runtime) 93
 среда непрерывной интеграции (CI) 344

Т

тестирование пользовательского интерфейса 213

У

удаленный компьютер (remote) 178

Ф

фулстек 258

Ц

циклическое расширение слов 138

Э

экстремальное программирование (XP) 560

Я

ядро для эффективного, удаленного и множественного взаимодействия компьютеров (KERMIT) 295
 язык гипертекстовой разметки (HTML) 109
 язык манипулирования данными (DML) 396
 язык определения данных (DDL) 396
 язык разметки Cold Fusion (CFML) 292
 язык структурированных запросов (SQL) 138

А

Active Server Pages (ASP) 108
 ActiveState ActivePython 94
 AI Assistant 586
 AJAX 321
 Anaconda 94
 Angular 288

В

Bash shell 179
 Bootstrap 286, 313
 Bourne Again Shell (Bash) 95

С

Chocolately 356
 Cold Fusion 291
 Composer 108
 CRUD 332

Д

DataSpell 441
 DigitalOcean 402
 Django 365, 371, 383
 Docker 22, 50, 102, 344, 346, 394, 406, 551, 577, 608

Е

Emmet 304

Ф

FastAPI 322, 328, 355, 361
 Flask 79, 115, 292, 297, 301, 302, 309, 316, 361, 385, 390, 580
 foreign key 374

Г

Git 77, 176, 179, 181, 188, 193, 508, 510
 gitbash 179
 GitHub 57, 183
 Gnu Image Manipulation Program (GIMP) 278

И

IntelliJ IDEA 35, 128, 199
 IntelliSense 40
 IronPython 94

Ж

JavaScript Object Notation (JSON) 323
 Java ServerPages (JSP) 108
 JUnit 214
 Jupyter 51, 462, 481, 486, 496, 545, 610
 Jupyter notebook 459, 464, 485
 Jython 94

М

Mercurial 178
 MicroPython 94
 Microsoft SQL Server 257
 MongoDB 257, 411
 MySQL 394

N

Node.js 257

P

pandas DataFrame 468

PEP-8 41, 148

Perforce 178

PowerShell 95

Pull 185

Push 185

Pyramid 386, 388

Python Debugger (pdb) 43

Python Enhancement Proposal #8 148

R

React 258, 259, 275, 276, 287, 301, 356, 553

React Native 288

Read Execute Print Loop (REPL) 105

Redis 257

RESTful API 292, 347, 361

S

schema 408

Scientific mode 455

SciView 51, 454, 464, 465, 470

Secure Shell (SSH) 179

Secure Sockets Layer (SSL) 325

Space 193

SQL 431, 436

Subversion 35, 178

Subversion (SVN) 66

T

Team Foundation Server 176

theme 556

Transport Layer Security (TLS) 325

V

Vagrant 50, 551, 588, 590

Vite 289

Vue 289

W

Werkzeug 298

Windows Terminal 95

Windows для Linux (WSL) 102

X

XMLHttpRequest (XHR) 327

Z

Z shell (zsh) 179

Книги издательства «ДМК Пресс» можно заказать
в торгово-издательском холдинге «КТК Галактика» наложенным платежом,
выслав открытку или письмо по почтовому адресу:
115487, г. Москва, пр. Андропова д. 38 оф. 10.
При оформлении заказа следует указать адрес (полностью),
по которому должны быть высланы книги;
фамилию, имя и отчество получателя.
Желательно также указать свой телефон и электронный адрес.
Эти книги вы можете заказать и в интернет-магазине: www.galaktika-dmk.com.
Оптовые закупки: тел. (499) 782-38-89.
Электронный адрес: books@aliants-kniga.ru.

Брюс М. Ван Хорн
Куан Нгуен

PyCharm: профессиональная работа на Python

Главный редактор *Мовчан Д. А.*
dmkpress@gmail.com
Перевод *Люско И. Л.*
Корректор *Абросимова Л. А.*
Верстка *Луценко С. В.*
Дизайн обложки *Мовчан А. Г.*

Формат 70×100 1/16.
Гарнитура «PT Serif». Печать цифровая.
Усл. печ. л. 50,21. Тираж 100 экз.

Веб-сайт издательства: www.dmkpress.com

Раскройте возможности PyCharm для создания деловых, научных и веб-приложений на Python!

PyCharm – лучшая профессиональная среда разработки для программистов Python среди множества доступных интегрированных сред. Независимо от того, в каких целях используется Python – для общих задач автоматизации, создания утилит, веб-приложений, анализа данных, машинного обучения или бизнес-приложений, – инструменты PyCharm упрощают выполнение сложных задач и оптимизируют общую производительность.

В процессе чтения вы:

- изучите базовые и расширенные функции PyCharm;
- установите, сконфигурируете и настроите в PyCharm свои проекты Python;
- узнаете, как разрабатывать веб-приложения с помощью Flask, Django, FastAPI и Pyramid;
- откроете для себя возможности PyCharm по управлению базами данных и визуализации данных;
- освоите автоматизацию написания кода, отладку и удаленную разработку в PyCharm;
- научитесь выполнять задачи по обработке данных с помощью блокнотов Jupyter, NumPy и pandas.

Книга адресована как опытным разработчикам на Python, так и новичкам.



Брюс Ван Хорн II – архитектор и ведущий разработчик в Visual Storage Intelligence. Специализируется на веб-разработке на Python, C# и JavaScript. Автор нескольких книг и серий видео, опубликованных в Packt, Skillsoft, Lynda.com и LinkedIn Learning, включая оригинальный видеокурс LinkedIn по PyCharm.



Куан Нгуен – доктор философии в области компьютерных наук в Вашингтонском университете в Сент-Луисе, где занимается исследованиями байесовских методов в машинном обучении, а также проблемами принятия решений, связанных с неопределенностью. Автор нескольких книг по программированию на Python и научным вычислениям.

Интернет-магазин:
www.dmkpress.com

Оптовая продажа:
КТК «Галактика»
books@aliens-kniga.ru

Packt
DMK
ИЗДАТЕЛЬСТВО
www.dmk.ru

ISBN 978-5-93700-274-7



9 785937 002747 >