

МУХАММАД АЗИФ

PYTHON

ДЛЯ

ГИКОВ

Санкт-Петербург

«БХВ-Петербург»

2024

УДК 004.43
ББК 32.973.26-018.1
А35

Азиф М.

А35 Python для гиков: Пер. с англ. — СПб.: БХВ-Петербург, 2024. — 432 с.: ил.
ISBN 978-5-9775-0956-5

Книга подробно рассказывает о разработке, развертывании и поддержке крупномасштабных проектов на Python. Представлены такие концепции, как итераторы, генераторы, обработка ошибок и исключений, обработка файлов и ведение журналов. Приведены способы автоматизации тестирования приложений и разработки через тестирование (TDD). Рассказано о написании приложений с использованием кластера Apache Spark для обработки больших данных, о разработке и развертывании бессерверных программ в облаке на примере Google Cloud Platform (GCP), о создании веб-приложений и REST API, использовании среды Flask. Показаны способы применения языка для создания, обучения и оценки моделей машинного обучения, а также их развертывания в облаке, описаны приемы использования Python для извлечения данных с сетевых устройств и систем управления сетью (NMS).

Для программистов

УДК 004.43
ББК 32.973.26-018.1

Группа подготовки издания:

Руководитель проекта	<i>Павел Шалин</i>
Зав. редакцией	<i>Людмила Гауль</i>
Перевод с английского	<i>Марины Попович</i>
Редактор	<i>Александр Морозов</i>
Компьютерная верстка	<i>Натальи Смирновой</i>
Корректор	<i>Анна Брезман</i>
Оформление обложки	<i>Зои Канторович</i>

© Packt Publishing 2021. First published in the English language under the title 'Python for Geeks – (9781801070119)'
Впервые опубликовано на английском языке под названием 'Python for Geeks – (9781801070119)'

Подписано в печать 05.09.23.
Формат 70×100^{1/16}. Печать офсетная. Усл. печ. л. 34,83.
Тираж 1500 экз. Заказ №
"БХВ-Петербург", 191036, Санкт-Петербург, Гончарная ул., 20.
Отпечатано с готового оригинал-макета
ООО "Принт-М", 142300, М.О., г. Чехов, ул. Полиграфистов, д. 1

ISBN 978-1-80107-011-9 (англ.)
ISBN 978-5-9775-0956-5 (рус.)

© Packt Publishing, 2021
© Перевод на русский язык, оформление.
ООО "БХВ-Петербург", ООО "БХВ", 2024

Содержание

Об авторе.....	14
О рецензентах.....	15
Предисловие	16
Для кого предназначена эта книга.....	16
О чем эта книга.....	17
Как получить максимальную отдачу от книги.....	18
Загрузка файлов с примерами кода.....	18
Условные обозначения.....	19
РАЗДЕЛ 1. PYTHON ПОМИМО ОСНОВ	21
Глава 1. Оптимальный жизненный цикл разработки на Python.....	23
Культура и сообщество Python.....	23
Этапы проекта Python.....	26
Стратегия процесса разработки.....	27
Итерация по этапам.....	28
Стремление к MVP в первую очередь.....	28
Стратегия разработки для специализированных предметных областей.....	29
Эффективное документирование кода Python.....	32
Комментарии Python.....	32
Docstring.....	32
Документация на уровне функций или классов.....	34
Разработка эффективной схемы именования.....	35
Методы.....	36
Переменные.....	36
Константы.....	37
Классы.....	37
Пакеты.....	38
Модули.....	38
Соглашения об импорте.....	38
Аргументы.....	38
Полезные инструменты.....	38

Системы контроля версий	39
Что не стоит хранить в репозитории системы контроля версий	39
Понимание стратегий развертывания кода	40
Пакетная разработка	40
Среды разработки Python	42
IDLE	42
Sublime Text	42
Atom	42
PyCharm	42
Visual Studio Code	43
PyDev	43
Spyder	43
Заключение	43
Вопросы	44
Дополнительные ресурсы	44
Ответы	44
Глава 2. Использование модулей для сложных проектов	45
Технические требования	46
Знакомство с модулями и пакетами	46
Импорт модулей	46
Оператор import	48
Оператор <code>__import__</code>	52
Инструмент <code>importlib.import_module</code>	52
Абсолютный и относительный импорт	53
Загрузка и инициализация модуля	55
Загрузка модуля	55
Установка параметров для специальных переменных	55
Выполнение кода	56
Стандартные модули	57
Написание многоразовых модулей	58
Независимая функциональность	58
Генерализация функционала	59
Традиционный стиль программирования	60
Четко определенная документация	61
Сборка пакетов	62
Именованье	63
Файл инициализации пакета	63
Сборка пакета	63
Доступ к пакетам из любого расположения	66
Общий доступ к пакету	70
Создание пакета в соответствии с рекомендациями PyPA	70
Установка из локального исходного кода с помощью <code>pip</code>	73
Публикация пакета в Test PyPI	75
Установка пакета из PyPI	76

Содержание	7
Заключение	77
Вопросы	77
Дополнительные ресурсы	77
Ответы	78
Глава 3. Расширенное объектно-ориентированное программирование на Python	79
Технические требования	80
Знакомство с классами и объектами	80
Различия между атрибутами класса и атрибутами экземпляра	80
Конструкторы и деструкторы классов	83
Различия между методами класса и методами экземпляра	84
Специальные методы	85
Принципы ООП	86
Инкапсуляция данных	87
Объединение данных и действий	87
Соккрытие информации	89
Защита данных	91
Традиционный подход к использованию геттеров и сеттеров	91
Использование декоратора <code>property</code>	92
Расширение классов с помощью наследования	94
Простое наследование	94
Множественное наследование	96
Полиморфизм	97
Перегрузка метода	97
Переопределение метода	98
Абстракция	100
Композиция как альтернативный подход к проектированию	102
Утиная типизация в Python	104
Когда не стоит использовать ООП в Python	105
Заключение	106
Вопросы	106
Дополнительные ресурсы	107
Ответы	107
РАЗДЕЛ 2. РАСШИРЕННЫЕ КОНЦЕПЦИИ ПРОГРАММИРОВАНИЯ	109
Глава 4. Библиотеки Python для продвинутого программирования	111
Технические требования	111
Введение в контейнеры данных Python	112
Строки	112
Списки	113
Кортежи	114
Словари	114
Множества	115

Итераторы и генераторы для обработки данных.....	116
Итераторы.....	116
Генераторы.....	120
Обработка файлов в Python.....	122
Операции с файлами.....	123
Обработка ошибок и исключений.....	126
Работа с исключениями в Python.....	127
Вызов исключений.....	129
Определение пользовательских исключений.....	130
Модуль logging в Python.....	131
Основные компоненты системы логирования.....	132
Работа с модулем logging.....	134
Что стоит и не стоит записывать в журнал.....	140
Заключение.....	141
Вопросы.....	141
Дополнительные ресурсы.....	141
Ответы.....	141
Глава 5. Тестирование и автоматизация с помощью Python.....	143
Технические требования.....	144
Понимание различных уровней тестирования.....	144
Модульное тестирование.....	145
Интеграционное тестирование.....	145
Системное тестирование.....	145
Приемочное тестирование.....	146
Работа с тестовыми фреймворками Python.....	146
Работа с фреймворком unittest.....	148
Фреймворк тестирования pytest.....	157
Разработка через тестирование.....	165
Красный.....	165
Зеленый.....	166
Рефакторинг.....	166
Автоматизированная непрерывная интеграция.....	167
Заключение.....	168
Вопросы.....	168
Дополнительные ресурсы.....	168
Ответы.....	169
Глава 6. Дополнительные советы и приемы Python.....	170
Технические требования.....	170
Расширенные приемы использования функций в Python.....	171
Функции counter, itertools и zip для итерационных задач.....	171
Использование методов filter, map и reduce для преобразования данных.....	175
Создание лямбда-функций.....	178
Внедрение одной функции в другую.....	179
Изменение поведения функции с помощью декораторов.....	181

Расширенные концепции структур данных	187
Внедрение словаря в словарь	187
Использование включений	190
Введение в Pandas DataFrame	192
Операции с объектом DataFrame	193
Сложные случаи использования DataFrame	198
Заключение	203
Вопросы	204
Дополнительные ресурсы	204
Ответы	204
РАЗДЕЛ 3. МАСШТАБИРОВАНИЕ ЗА ПРЕДЕЛЫ ОДНОГО ПОТОКА	205
Глава 7. Многопроцессорная обработка, многопоточность и асинхронное программирование	207
Технические требования	208
Многопоточность в Python и ее ограничения	208
Слепое пятно Python	209
Ключевые компоненты многопоточного программирования на Python	210
Практический пример: многопоточное приложение для загрузки файлов с Google Диска	218
Многопроцессорная обработка	221
Создание нескольких процессов	221
Обмен данными между процессами	224
Обмен объектами между процессами	228
Синхронизация процессов	230
Практический пример: многопроцессорное приложение для загрузки файлов с Google Диска	231
Асинхронное программирование для адаптивных систем	233
Модуль asyncio	234
Распределение задач с помощью очередей	236
Практический пример: асинхронное приложение для загрузки файлов с Google Диска	238
Заключение	240
Вопросы	240
Дополнительные ресурсы	241
Ответы	241
Глава 8. Масштабирование Python с помощью кластеров	242
Технические требования	243
Возможности кластеров для параллельной обработки	243
Nadoop MapReduce	244
Apache Spark	246
Устойчивые распределенные наборы данных (RDD)	249
Операции с RDD	249
Создание RDD	250

PySpark для параллельной обработки данных	251
Создание программ SparkSession и SparkContext	253
PySpark для операций с RDD	254
PySpark DataFrames	257
PySpark SQL	261
Практические примеры использования Apache Spark и PySpark	262
Пример 1: калькулятор числа π в Apache Spark	262
Заключение	268
Вопросы	269
Дополнительные ресурсы	269
Ответы	270
Глава 9. Программирование на Python для облака	271
Технические требования	271
Знакомство с облачными возможностями для приложений Python	272
Среды разработки Python для облака	272
Облачные среды выполнения для Python	274
Создание веб-сервисов Python для облачного развертывания	276
Использование Google Cloud SDK	277
Использование веб-консоли GCP	284
Использование Google Cloud Platform для обработки данных	287
Введение в основы Apache Beam	287
Конвейеры Apache Beam	289
Создание конвейеров для Cloud Dataflow	294
Заключение	298
Вопросы	299
Дополнительные ресурсы	299
Ответы	300
РАЗДЕЛ 4. PYTHON ДЛЯ ВЕБ-РАЗРАБОТКИ, ОБЛАКА И СЕТИ	301
Глава 10. Использование Python для разработки веб-приложений и REST API	303
Технические требования	304
Требования к веб-разработке	304
Веб-фреймворки	304
Пользовательский интерфейс	305
Веб-сервер/сервер приложений	306
База данных	307
Безопасность	307
API	307
Документация	307
Знакомство с фреймворком Flask	308
Создание базового веб-приложения с маршрутизацией	308
Обработка запросов с разными типами HTTP-методов	310
Отображение статического и динамического контента	312

Извлечение параметров из HTTP-запроса	313
Взаимодействие с системами управления базами данных	315
Обработка ошибок и исключений в веб-приложениях	318
Создание REST API.....	321
Использование Flask для REST API.....	322
Разработка REST API для доступа к базе данных	324
Пример: создание веб-приложения с помощью REST API	326
Заключение	331
Вопросы.....	332
Дополнительные ресурсы	332
Ответы	332
Глава 11. Разработка микросервисов на Python	334
Технические требования.....	334
Введение в микросервисы	335
Практические рекомендации по созданию микросервисов	337
Создание приложений на базе микросервисов.....	338
Варианты разработки микросервисов на Python.....	339
Варианты развертывания микросервисов.....	340
Разработка приложения на основе микросервисов	341
Заключение	352
Вопросы.....	352
Дополнительные ресурсы	352
Ответы	353
Глава 12. Создание бессерверных функций на Python.....	354
Технические требования.....	355
Знакомство с бессерверными функциями	355
Преимущества бессерверных функций.....	356
Варианты использования	356
Варианты развертывания бессерверных функций	357
Написание бессерверных функций.....	358
Создание облачной функции на основе HTTP с помощью консоли GCP	359
Практический пример: создание приложения для уведомлений о событиях в облачном хранилище	363
Заключение	367
Вопросы.....	367
Дополнительные ресурсы	367
Ответы	367
Глава 13. Python и машинное обучение	369
Технические требования.....	370
Введение в машинное обучение.....	370
Использование Python для машинного обучения.....	372
Библиотеки машинного обучения в Python	372
Рекомендации по обучающим данным	374

Создание и оценка модели машинного обучения	375
Процесс построения модели машинного обучения	375
Создание примера машинного обучения	376
Оценка модели с помощью кросс-валидации и тонкой настройки гиперпараметров	381
Сохранение ML-модели в файл	384
Развертывание и прогнозирование ML-модели в GCP Cloud	385
Заключение	388
Вопросы	388
Дополнительные ресурсы	388
Ответы	389
Глава 14. Python для автоматизации сети	390
Технические требования	391
Введение в автоматизацию сети	391
Плюсы и минусы автоматизации сети	392
Варианты использования	393
Взаимодействие с сетевыми устройствами	394
Протоколы для взаимодействия с сетевыми устройствами	394
Взаимодействие с сетевыми устройствами с помощью библиотек Python на основе SSH	397
Взаимодействие с сетевыми устройствами с помощью NETCONF	404
Интеграция с системами управления сетью	408
Использование конечных точек сервиса определения местоположения	409
Получение токена аутентификации	410
Получение сетевых устройств и инвентаризация интерфейсов	411
Обновление порта на сетевом устройстве	412
Интеграция с событийно-ориентированными системами	414
Создание подписок для Apache Kafka	416
Обработка событий от Apache Kafka	417
Продление и удаление подписки	418
Заключение	418
Вопросы	419
Дополнительные ресурсы	419
Ответы	420
Предметный указатель	421

*Моей жене, Сайме Аруж, без любящей поддержки которой
было бы невозможно завершить эту книгу.
Моим дочерям, Сане Азиф и Саре Азич и моему сыну Зейну Азиф,
которые вдохновляли меня на протяжении всего этого путешествия.
В память о моих отце и матери за их жертвы и за то,
что показали силу решимости.
А также моим братьям и сестрам за всю их поддержку,
которую они оказывали в отношении моей учебы и работы.*

- Мухаммад Азиф

Об авторе

Мухаммад Азиф — главный архитектор решений с большим опытом в области веб-разработки, виртуализации, машинного обучения, а также автоматизации сетей и облачных систем. Благодаря огромному опыту в разных областях он привел многие крупномасштабные проекты к успешному развертыванию. В последние годы Мухаммад перешел на более высокие руководящие должности, но ему по-прежнему нравится самому писать код и использовать современные технологии для решения реальных задач. В 2012 году он получил докторскую степень по компьютерным системам в Карлтонском университете в Оттаве, а сейчас руководит разработкой решений в Nokia.

Я хочу поблагодарить всех, кто поддерживал меня, особенно Имрана Ахмеда, который помог мне написать главу 1.

О рецензентах

Харшит Джейн (**Harshit Jain**) — специалист по data science с пятилетним опытом в сфере программирования, помогает компаниям создавать и применять сложные алгоритмы машинного обучения для повышения эффективности бизнеса. Он имеет широкий спектр знаний — от классического машинного обучения до глубокого обучения и компьютерного зрения. В свободное время он наставляет будущих специалистов data science. Его доклады и статьи, в том числе «*Learning the Basics of Data Science*» и «*How to Check the Impact on Marketing Activities — Market Mix Modeling*», показывают, что наука о данных — это его страсть. Рецензирование этой книги — его первая, но, определенно, не последняя попытка укрепить свои знания о Python для помощи в вашем обучении.

Сураб Бхавсар (**Sourabh Bhavsar**) — старший fullstack-работчик, практикует Agile и облачные технологии, имеет опыт в разработке программного обеспечения более 7 лет. Он закончил аспирантуру по искусственному интеллекту и машинному обучению в Техасском университете в Остине, получил степень магистра делового администрирования (MBA) по маркетингу и степень бакалавра инженерных наук в области информационных технологий в университете Пуны, Индия. В настоящий момент он работает ведущим инженером в PayPal, где отвечает за проектирование и разработку решений на основе микросервисов, а также реализацию различных движков рабочих процессов и оркестрации. Сураб верит, что учиться никогда не поздно, и любит изучать новые веб-технологии. В свободное время он играет на барабанах табла и изучает астрологию.

Предисловие

Python — многоцелевой язык, который помогает решать задачи любой сложности в множестве областей. «*Python для гиков*» научит вас, как продвигаться по карьерной лестнице с помощью советов и приемов экспертов.

Сначала вы исследуете различные способы работы с Python как с точки зрения проектирования, так и с точки зрения реализации. Далее вы рассмотрите жизненный цикл масштабного проекта Python. По мере продвижения сфокусируетесь на создании элегантной структуры с помощью ее модульного разделения. Вы узнаете, как масштабировать свой код за пределы одного потока, а также как реализовать многопроцессорную и многопоточную обработки. В дополнение поймете, как можно использовать язык не только для развертывания на одной машине, но и для использования целых кластеров в облачной вычислительной среде. Затем вы познакомитесь с методами обработки информации, рассмотрите многообразные и масштабируемые конвейеры данных и узнаете, как использовать эти методы для автоматизации сети, бессерверных функций и машинного обучения. В завершение вы научитесь, как составить эффективный стратегический план веб-разработки, используя практики и рекомендации из этой книги.

К концу этой книги вы сможете уверенно программировать на Python для крупных и сложных проектов.

Для кого предназначена эта книга

Эта книга будет полезна Python-разработчикам среднего уровня в любой области, которые стремятся развить навыки проектирования и управления масштабными сложными проектами. Она также пригодится программистам, которые хотят создавать многообразные модули и библиотеки Python и разрабатывать приложения для облачного развертывания. Предыдущий опыт работы с этим языком поможет вам извлечь максимальную пользу из книги.

О чем эта книга

Глава 1 «Оптимальный жизненный цикл разработки на Python» поможет понять жизненный цикл стандартного проекта Python и его этапы, а также здесь приведены лучшие практики и советы по написанию кода.

Глава 2 «Использование модулей для сложных проектов» посвящена модулям и пакетам Python.

Глава 3 «Расширенное объектно-ориентированное программирование на Python» описывает, как можно реализовать продвинутые концепции объектно-ориентированного программирования с помощью Python.

Глава 4 «Библиотеки Python для продвинутого программирования» познакомит с такими концепциями, как итераторы, генераторы, обработка ошибок и исключений, обработка файлов и ведение журналов в Python.

Глава 5 «Тестирование и автоматизация с помощью Python» помогает понять разные типы автоматизации тестов, например, модульное, интеграционное и системное тестирование, а также как реализовать модульные тесты с помощью популярных фреймворков.

Глава 6 «Дополнительные советы и приемы Python» посвящена расширенным возможностям языка по преобразованию данных, созданию декораторов, а также использованию структур данных, включая *pandas DataFrame*, для аналитических приложений.

Глава 7 «Многопроцессорная обработка, многопоточность и асинхронное программирование» приводит рекомендации по созданию многопоточных и многопроцессорных приложений с помощью встроенных библиотек Python.

Глава 8 «Масштабирование Python с помощью кластеров» рассказывает о работе *Apache Spark* и написании приложений с использованием кластера *Apache Spark* для обработки больших данных.

Глава 9 «Программирование на Python для облака» описывает, как разрабатывать и развертывать приложения на облачной платформе, а также как использовать *Apache Beam* в целом и для *Google Cloud Platform* в частности.

Глава 10 «Использование Python для разработки веб-приложений и REST API» посвящена использованию среды *Flask* для разработки веб-приложений, взаимодействия с базами данных и создания *REST API* для веб-сервисов.

Глава 11 «Разработка микросервисов на Python» рассматривает общие понятия, а также использование фреймворка *Django* для создания примера микросервиса и интеграции его с микросервисом на базе *Flask*.

Глава 12 «Создание бессерверных функций на Python» описывает роль бессерверных функций в облачных вычислениях и способы их создания с помощью Python.

Глава 13 «Python и машинное обучение» поможет понять использование языка для создания, обучения и оценки моделей машинного обучения, а также их развертывания в облаке.

Глава 14 «Python для автоматизации сети» посвящена библиотекам Python для извлечения данных с сетевых устройств и систем управления сетью (NMS), а также для отправки конфигурации на устройства и NMS.

Как получить максимальную отдачу от книги

Для получения реальной пользы от этой книги необходимо предварительно изучить Python. Вам понадобится Python версии 3.7 или более поздней. Все примеры кода были протестированы в версиях 3.7 и 3.8 и должны работать с более поздними «3.x» версиями.

Аккаунт *Google Cloud Platform* (подойдет пробная бесплатная версия) будет полезен для развертывания некоторых примеров кода в облаке.

Таблица 1.1. Программное обеспечение и оборудование, описанное в книге

ПО и оборудование для книги	Требования к операционной системе
Python 3.7 или выше	Windows
Apache Spark 3.1.1	macOS
Cisco IOS XR 7.12 (сетевое устройство)	Linux
Nokia Network Services Platform (NSP) 21.6	
Google Cloud SDK 343.0.0	

Если вы читаете цифровую версию книги, мы советуем набирать код самостоятельно или использовать его из репозитория **GitHub** (ссылка доступна в следующем подразделе). Это поможет избежать ошибок, связанных с копированием и вставкой кода.

Загрузка файлов с примерами кода

Файлы с примерами кода можно скачать по адресу:

<https://github.com/PacktPublishing/Python-for-Geeks>.

Если выйдет обновление кода, оно будет отражаться в репозитории GitHub.

Мы также предлагаем другие пакеты из нашего обширного каталога книг и видео:
<https://github.com/PacktPublishing/>.

Вы можете найти там что-то интересное.

Условные обозначения

В книге используется ряд текстовых условных обозначений и соглашений.

Код в тексте: указывает кодовые слова в тексте, имена таблиц баз данных, имена папок и файлов, расширения файлов, пути, фиктивные URL-адреса и дескрипторы Twitter. Пример: «Смонтируйте загруженный файл образа диска `WebStorm-10*.dmg` как еще один диск в системе».

Блок кода выглядит следующим образом:

```
resource = {
  "api_key": "AIzaSyDYKmm85kebxddKrGns4z0",
  "id": "0B8TxHW2Ci6dbckVwTRtTl3RUU",
  "fields": "files(name, id, webContentLink)",
}
```

Для привлечения вашего внимания к определенной части кода мы будем выделять строки **жирным шрифтом**:

```
#casestudy1.py: Pi calculator
from operator import add
from random import random

from pyspark.sql import SparkSession

spark = SparkSession.builder.
master("spark://192.168.64.2:7077") \
  .appName("Pi calculator app") \
  .getOrCreate()

partitions = 2
n = 10000000 * partitions

def func(_):
  Preface xxi
  x = random() * 2 - 1
  y = random() * 2 - 1
  return 1 if x ** 2 + y ** 2 <= 1 else 0
count = spark.sparkContext.parallelize(range(1, n + 1),
  partitions).map(func).reduce(add)
print("Pi is roughly %f" % (4.0 * count / n))
```

Входные и выходные данные в командной строке:

```
pi is roughly 3.141479
```

Жирным шрифтом и *курсивом* выделяются новые термины и важные слова. Например, пункты меню и сообщения в диалоговых окнах. Пример: «Как мы уже говорили, в Cloud Shell есть редактор, который можно запустить, нажав на кнопку **Open editor**».

СОВЕТЫ И ВАЖНЫЕ ПРИМЕЧАНИЯ

Выделены таким образом.

Раздел 1

Python помимо основ

Мы начинаем наше путешествие с изучения различных способов, как оптимально использовать Python как с точки зрения проектирования, так и с точки зрения реализации. Подробно рассмотрим жизненный цикл крупномасштабного проекта и его этапов. Изучим возможные способы, как создать элегантный дизайн с помощью модульности проекта. Везде, где это необходимо, заглянем «под капот» для лучшего понимания внутренностей Python. И в конце глубоко погрузимся в объектно-ориентированное программирование.

Этот раздел содержит следующие главы:

- ◆ «Глава 1: Оптимальный жизненный цикл разработки на Python».
 - ◆ «Глава 2: Использование модулей для сложных проектов».
 - ◆ «Глава 3: Расширенное объектно-ориентированное программирование на Python».
-

Оптимальный жизненный цикл разработки на Python

Эта книга предназначена для разработчиков, у которых уже есть опыт работы с Python, поэтому самые азы мы опустим. Начнем с нескольких слов о сообществе разработчиков и их уникальной культуре, которая отражается в коде. Затем поговорим об этапах стандартного проекта и посмотрим, как сформировать стратегию для его разработки.

После познакомимся со способами документирования кода и эффективными схемами именования, которые упрощают его обслуживание. Также рассмотрим системы контроля версий в проектах Python, в том числе **Jupyter Notebook**. В конце обсудим рекомендации по развертыванию кода после разработки и тестирования.

Темы этой главы:

- ◆ Культура и сообщество Python.
- ◆ Этапы проекта Python.
- ◆ Стратегия процесса разработки.
- ◆ Эффективное документирование кода Python.
- ◆ Разработка эффективной схемы именования.
- ◆ Системы контроля версий.
- ◆ Понимание стратегий развертывания кода.
- ◆ Среды разработки Python.

Культура и сообщество Python

Python — это высокоуровневый интерпретируемый язык, созданный Гвидо ван Россумом в 1991 году. Язык имеет собственное сообщество программистов, которое

уделяет особое внимание написанию кода и придерживается своей уникальной философии разработки. Сегодня Python используется в самых разных отраслях — от образования до медицины, и в каждом проекте ощущается его характерная культура.

Разработчики считают, что код должен быть максимально простым. Если задачу можно решить несколькими способами, следует выбрать вариант, наиболее соответствующий соглашениям и философии Python. Настоящие фанаты даже создают *артефакты*, максимально соответствующие правилам языка, а нарушителей этих правил считают плохими программистами. В этой книге мы постараемся не отступать от философии Python.

Существует даже официальный документ, написанный Тимом Питерсом, — «Дзен Python» (*The Zen of Python*), который призывает поддерживать принципы языка. В нем сказано, что Python должен быть максимально аккуратным, понятным и хорошо задокументированным. Прочсть его можно, выполнив в консоли команду `import this` (рис. 1.1):

```
[1] import this
    this

☐ The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
<module 'this' from '/usr/lib/python3.6/this.py'>
```

Рис. 1.1. Дзен Python

Красивое лучше, чем уродливое.

Явное лучше, чем неявное.

Простое лучше, чем сложное.

Многосоставное лучше, чем запутанное.

Последовательное лучше, чем вложенное.

Умеренное лучше, чем нагроможденное.

Читаемость имеет значение.

Особые случаи не настолько особые, чтобы нарушать правила.

При этом практичность важнее безупречности.

Ошибки никогда не должны замалчиваться.

Если они не замалчиваются явно.

Встретив двусмысленность, не пытайся угадать.

Должен существовать, предпочтительно, только один очевидный способ решить задачу.

Хотя он поначалу может быть и не очевиден.

Сейчас лучше, чем никогда.

*Хотя никогда зачастую лучше, чем **прямо** сейчас.*

Если реализацию сложно объяснить, то идея плохая.

Если реализацию легко объяснить, то идея, возможно, хорошая.

Пространство имен — отличная штука! Будем делать его больше!

Текст выглядит загадочно, будто выбит на стене в гробнице фараона, но это сделано намеренно. При этом каждая строчка имеет глубокое значение. Мы будем ссылаться на «Дзен Python» на протяжении всей книги, поэтому обсудим несколько отрывков:

- ◆ *«Красивое лучше, чем уродливое»*: важно, чтобы код был хорошо написан, читаем и не требовал разъяснений; он должен не только отлично работать, но и красиво выглядеть; не следует использовать кратчайший путь при написании кода, это может затруднить его чтение.
- ◆ *«Простое лучше, чем сложное»*: не нужно усложнять; если есть выбор, следует использовать более простой вариант; лишних сложностей нужно избегать; если для большего удобства необходимо удлинить код, лучше предпочесть этот вариант.
- ◆ *«Должен существовать, предпочтительно, только один очевидный способ решить задачу»*: у проблемы должно быть только одно *наилучшее* решение, которое нужно постараться найти; работая над структурой кода, стоит стремиться к такому решению.
- ◆ *«Сейчас лучше, чем никогда»*: не стоит ждать идеальных условий; решать проблему следует *сейчас*, опираясь на имеющуюся информацию, предположения, навыки, инструменты и инфраструктуру; после уже можно улучшать решение; не нужно ждать подходящего момента, он может никогда не наступить.
- ◆ *«Явное лучше, чем неявное»*: код не должен требовать пояснений; к выбору имен переменных, классов и структуры функций, а также к созданию общей архитек-

туры следует подходить разумно; здесь следует проявить чрезмерную бдительность и сделать код максимально понятным.

- ◆ «Последовательное лучше, чем вложенное»: вложенная структура занимает меньше места, но затрудняет чтение; по возможности используйте последовательные структуры.

Этапы проекта Python

Для начала определим основные этапы проекта. Каждый из них включает несколько схожих действий, как показано на схеме (рис. 1.2):

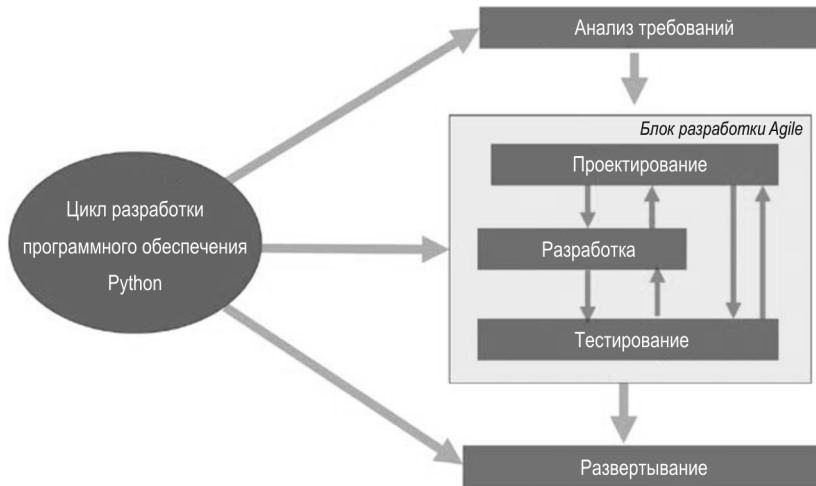


Рис. 1.2. Этапы проекта Python

Стандартный проект Python состоит из следующих этапов:

1. **Анализ требований:** на этом этапе мы общаемся с ключевыми заинтересованными лицами и анализируем их требования. Важно понять, *что* нужно делать, прежде чем решить, *как* это делать. Такими лицами могут быть пользователи или руководители компании. Необходимо наиболее подробно записать все требования, изучить их и обсудить с конечными пользователями перед следующим этапом; анализ требований не входит в цикл **проектирования**, **разработки** и **тестирования**, его нужно завершить до этих этапов. Важны как *функциональные*, так и *нефункциональные* требования. Первые следует разделить на модули, которые должны максимально соответствовать будущим модулям кода.
2. **Проектирование:** это техническое решение в ответ на требования из предыдущего этапа. Здесь идет обдумывание, *как* мы будем создавать наше решение. Это творческий процесс, в ходе которого используются опыт и навыки для создания подходящего набора и структуры модулей и оптимальных связей между

ними. На этом этапе особенно важно не допускать ошибок. Любые неверные решения на этом этапе обходятся дороже, чем на более поздних этапах. На исправление ошибок потребуется в 20 раз больше усилий, чем на устранение ошибок аналогичного масштаба на этапе *написания кода* (разработка). Например, последствия при неверном подборе подходящих данных или неправильно рассчитанном масштабе проекта будут более серьезными, по сравнению с ошибками при реализации функции. Проектирование часто лежит в более абстрактной плоскости идей. Ошибки здесь не так очевидны, их невозможно обнаружить в ходе тестирования, но можно легко перехватывать с помощью хорошо продуманной системы обработки исключений.

3. На этапе *проектирования* мы:

- создаем структуру кода и определяем его модули;
- выбираем фундаментальный подход (функциональное программирование, объектно-ориентированное программирование или их гибрид);
- определяем классы и функции и выбираем имена для этих высокоуровневых компонентов;

4. **Разработка** (Написание кода): здесь мы воплощаем проект на Python, начиная с высокоуровневых абстракций, компонентов и модулей, а затем переходя к деталям.

5. **Тестирование**: на этом этапе выполняется проверка, правильно ли работает код.

6. **Развертывание**: после тщательного тестирования мы предоставляем наше решение конечному пользователю, которому неважно, чем мы занимались на предыдущих этапах. Он просто решает свои задачи, описанные в требованиях. Допустим, мы работаем над проектом по машинному обучению, цель которого — прогнозировать осадки в Оттаве, тогда в ходе развертывания мы пытаемся определить, как предоставить пользователю удобное решение.

Теперь вы понимаете, из каких этапов состоит проект, и мы можем перейти к стратегическим аспектам.

Стратегия процесса разработки

Стратегическое планирование разработки охватывает каждый этап и переходы между ними. Для составления плана мы должны ответить на следующие вопросы:

1. Собираемся ли мы использовать подход с минимальным проектированием и сразу перейти к написанию кода?
2. Выберем ли мы *разработку через тестирование* (**Test-Driven Development, TDD**), где сначала пишутся тесты на основе требований, а затем создается код?
3. Хотим ли мы сначала создать *минимально жизнеспособный продукт* (**Minimum Viable Product, MVP**), а затем постепенно развивать его?

4. Какой стратегии мы будем придерживаться при проверке таких нефункциональных требований, как безопасность и производительность?
5. Будем ли мы разрабатывать под конкретные устройства или же развернем целый кластер или облако?
6. Какими будут объем, скорость и типы *входных* и *выходных* данных? Будем использовать *распределенную файловую систему Hadoop (Hadoop distributed file system, HDFS)* или файловую структуру **Simple Storage Service (S3)** от **Amazon**? Базу данных будем использовать **SQL** или **NoSQL**? Данные будут храниться локально или в облаке?
7. Работаем ли мы над такими специализированными вариантами использования, как *машинное обучение (Machine Learning)* с особыми требованиями к созданию конвейеров данных и моделей для тестирования, развертывания и обслуживания?

Ответы на эти вопросы помогут определить этапы процесса разработки. В последнее время предпочтительно использовать процессы *итеративного* подхода в том или ином виде. Также на старте популярна концепция MVP. Обо всем этом мы поговорим в следующих подразделах.

Итерация по этапам

Современный подход к разработке предусматривает короткие циклы проектирования, разработки и тестирования. Традиционная *каскадная модель* уже давно мертва. Выбор правильной детализации, акцента и частоты этапов зависит от характера проекта и избранной стратегии разработки. Если мы хотим выбрать стратегию с минимальной разработкой и сразу перейти к написанию кода, этап проектирования будет коротким. Но даже в этом случае необходимо обдумать структуру будущих модулей.

Независимо от выбора стратегии этапы связаны между собой. Мы начинаем с проектирования, реализуем план в коде, затем тестируем его. Отметив все недостатки, мы возвращаемся на этап проектирования.

Стремление к MVP в первую очередь

Рекомендуется отбирать только самые важные требования для создания минимально жизнеспособного продукта и постепенного его улучшения. Циклы проектирования, написания кода и тестирования повторяются снова и снова, пока не будет готов итоговый продукт, который можно развернуть и использовать.

Рассмотрим, как реализовать решение некоторых специализированных предметных областей на Python.

Стратегия разработки для специализированных предметных областей

Python используется в самых разных сценариях. Рассмотрим пять важных областей, где он применяется, и узнаем, как составлять стратегию разработки в зависимости от специфики каждого из них:

- ◆ машинное обучение;
- ◆ облачные и кластерные вычисления;
- ◆ системное программирование;
- ◆ сетевое программирование;
- ◆ бессерверные вычисления.

Рассмотрим каждый вариант поподробнее.

Машинное обучение

На сегодняшний день Python — самый популярный язык для написания алгоритмов машинного обучения, где требуется хорошо структурированная среда. Язык имеет обширную коллекцию высококачественных библиотек для реализации ML.

В стандартном проекте обычно используется методология **CRISP-DM (Cross-Industry Standard Process for Data Mining)** — стандарт, описывающий общие процессы и подходы к аналитике данных. Схема жизненного цикла CRISP-DM выглядит следующим образом (рис. 1.3):

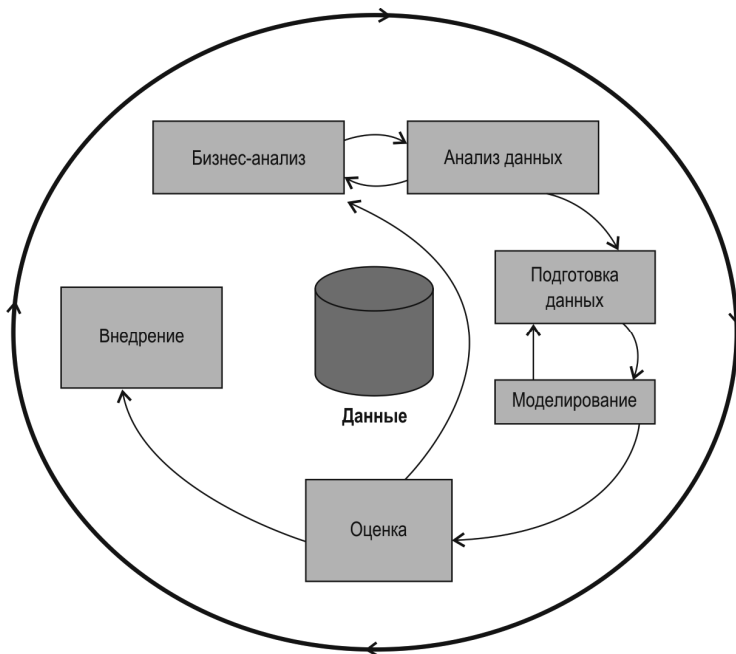


Рис. 1.3. Жизненный цикл CRISP-DM