1 Разработка стратегии прагматичного обучения

В этой главе

- ✓ Что такое «прагматичный»
- ✓ На что способен Python
- ✓ Когда стоит подумать об использовании альтернативных языков
- ✓ Чему может научить эта книга

Python — замечательный язык программирования. Распространение с открытым кодом, универсальность, независимость от платформы способствовали формированию огромного сообщества разработчиков и невероятной экосистемы, включающей десятки тысяч находящихся в свободном доступе библиотек для веб-разработки, машинного обучения (МО), data science и многих других областей. Надеюсь, вы согласитесь с тем, что умение писать код на Python важно, но умение создавать по-настоящему эффективные, безопасные и простые в сопровождении приложения дает вам поистине огромное преимущество. Эта книга поможет вам перейти с уровня начинающего программиста Python на уровень уверенного владения языком.

При практической работе в экосистеме Python мы применяем инструменты, предназначенные для конкретной предметной области, например веб-фреймворки или библиотеки для машинного обучения. Эффективное применение этих

инструментов — нетривиальное дело, для которого потребуется хорошее владение базовыми навыками программирования на Python: обработкой текста, работой со структурированными данными, созданием потоков управления и операций с файлами. Python-разработчики могут создавать разные решения для одних и тех же задач. Среди нескольких решений, как правило, можно найти наилучшее: наиболее компактное, или удобочитаемое, или эффективное. Все эти свойства часто объединяются под словосочетанием «питонический код»: имеется в виду идиоматический стиль программирования, овладеть которым стремятся все Python-разработчики. В этой книге рассказывается о том, как писать питонический код для решения различных задач программирования.

Python предоставляет разработчику столько возможностей, что было бы нереально или неразумно пытаться изучить их все по одной книге. Поэтому при выборе материала для своей книги я остановился на прагматичном подходе, а именно на обучении читателя тем основным навыкам, которые наиболее вероятно пригодятся в реальных проектах. Не менее важным я считаю регулярно обращаться к тому, как писать код, учитывая его удобочитаемость и простоту сопровождения. Это поможет вам выработать хороший стиль кодинга, который, я уверен, будет высоко оценен и вами, и вашими коллегам по команде.

> **ПРИМЕЧАНИЕ** В книге часто встречаются такие врезки. Многие из них содержат полезные советы относительно удобочитаемости и простоты сопровождения кода. Не пропускайте их!

1.1. О ПОЛЬЗЕ ПРАГМАТИЧНОГО ПОДХОДА

Мы программируем с определенной целью — для построения веб-сайта, обучения моделей МО или анализа данных. Какой бы ни была наша цель, нам нужно действовать прагматично, поскольку мы пишем код для решения реальных задач. Поэтому следует четко сформулировать свои цели, прежде чем изучать программирование с нуля или работать над повышением своей квалификации где-то в середине карьеры. Но даже если вы пока не уверены в том, чего именно хотите добиться при помощи Python, базовые средства Python универсально полезны. Освоив эти базовые средства, вы сможете применять их с любыми предметно-ориентированными инструментами Python.

Цель «стать прагматичным программистом» означает, что вам следует сосредоточиться на наиболее полезных приемах. Впрочем, освоение этих навыков — лишь первая веха на долгом пути, а вашей долгосрочной целью должно быть написание удобочитаемого кода, который не только работает, но и прост в сопровождении.

1.1.1. Написание удобочитаемого кода Python

Я разработчик, который фанатично стремится к удобочитаемости кода. Написание кода можно сравнить с разговором на обычном человеческом языке. Когда вы говорите на каком-нибудь языке, вы хотите, чтобы другие вас понимали? Если вы согласны, то, вероятно, согласитесь и с тем, что наш код должен быть понятным для других. Вопрос о том, обладают ли читатели нашего кода достаточной технической квалификацией для его понимания, находится вне зоны нашей ответственности. От нас зависит лишь то, как мы пишем код, — насколько удобочитаемым он получается. Попробуйте ответить на несколько простых вопросов:

- *Указывают ли имена ваших переменных на их назначение*? Никто не похвалит код, в котором полно переменных с именами вида `var0`, `temp_var` или `x`.
- *Указывают ли сигнатуры ваших функций на то, что делают эти функции*? Люди приходят в замешательство при виде функций с именами вида `do_data(data)` или `run_step1()`.
- *Насколько последовательно вы распределяете свой код по файлам*? Люди ожидают, что разные файлы одного типа используют сходную структуру. Например, размещаете ли вы инструкции `import` в начале файлов?
- *Хранятся ли конкретные файлы в правильных папках в структуре вашего проекта*? С увеличением масштаба проекта следует создавать отдельные папки для взаимосвязанных файлов.

Все эти вопросы относятся к области удобочитаемости. Не следует вспоминать о ней лишь время от времени, мы должны помнить про удобочитаемость кода постоянно, на протяжении всего проекта. Причина проста: к совершенству ведет правильная практика. Будучи нейробиологом по образованию, я точно знаю, как работает мозг, когда дело доходит до поведенческого обучения. Мы тренируем нейронную сеть нашего мозга, практикуясь в улучшении читабельности и постоянно задавая себе вопросы для самоконтроля. В конце концов вы обучите мозг распознавать хорошую практику программирования и начнете писать удобочитаемый, простой в сопровождении код, даже не задумываясь об этом.

1.1.2. Думайте о опровождаемости еще до написания кода

В отдельных случаях мы создаем код, предназначенный для одноразового использования. Когда мы пишем скрипт, нам почти всегда удается убедить себя в том, что он не предназначается для повторного применения, а значит, не нужно беспокоиться о создании понятных имен переменных, о правильном структурировании кода или рефакторинге функций и моделей данных

(не говоря уже том, что мы не пишем комментарии или оставляем устаревшие). Но сколько раз выяснялось, что тот же скрипт приходится использовать на следующей неделе и даже на следующий день? Вероятно, такое случалось с каждым из нас.

В предыдущем абзаце описана проблема простоты сопровождения (сопровождаемости) в малом масштабе. В данном случае она влияет только на вашу собственную производительность на коротком отрезке времени. Но если вы работаете в командной среде, сложности, создаваемые отдельными участниками, накапливаются и превращаются в крупномасштабные проблемы с сопровождаемостью. Участники команды не следуют одинаковым правилам выбора имен переменных, функций и файлов. В коде сплошь и рядом встречаются закомментированные фрагменты. Повсюду оставлены неактуальные комментарии.

Чтобы разрешить проблемы с сопровождаемостью на более поздних стадиях проектов, следует сформировать правильные установки еще во время обучения программированию. Ниже приводятся некоторые вопросы, которые стоит учитывать для выработки правильного отношения к простоте сопровождения в долгосрочной перспективе.

- *Свободен ли ваш код от устаревших комментариев и закомментированных участков*? Если ответ будет отрицательным, обновите или удалите их! Такие комментарии хуже их полного отсутствия, потому что они могут содержать противоречивую информацию.
- *Значительное ли место в коде занимает дублирование*? Если ответ будет положительным, вероятно, код требует рефакторинга. В программировании нередко упоминается практическое правило DRY (Don't Repeat Yourself, то есть «не повторяйтесь»). Удалив дубликаты из кода, вы получите один совместно используемый блок, что снизит риск ошибок по сравнению с внесением изменений во все повторяющиеся фрагменты.
- *Используете ли вы такие системы контроля версий, как Git*? Если ответ будет отрицательным, изучите расширения или плагины для вашей интегрированной среды разработки (IDE). Для Python часто используются IDE PyCharm и Visual Studio Code. Во многих IDE имеются интегрированные средства контроля версий, которые заметно упрощают управление версиями.

Каждый прагматичный программист Python должен взять на вооружение эти приемы, упрощающие сопровождение. Ведь почти все инструменты Python распространяются с открытым кодом и быстро развиваются. Следовательно, учет будущей сопровождаемости должен занимать центральное место в любом реальном проекте. В этой книге мы постараемся затронуть вопросы реализации практик сопровождения при повседневном программировании на Python. Помните, что удобочитаемость кода является ключевым фактором

долгосрочной простоты сопровождения. Если вы сосредоточитесь на написании удобочитаемого кода, это будет способствовать улучшению сопровождаемости вашей кодовой базы.

1.2. ЧТО PYTHON ДЕЛАЕТ ХОРОШО — ИЛИ НЕ ХУЖЕ, ЧЕМ ДРУГИЕ ЯЗЫКИ

Растущая популярность Python обусловлена характеристиками самого языка. Хотя ни одна из этих характеристик не уникальна для Python, именно их гармоничное сочетание объясняет повсеместное его распространение. Ниже приведена краткая сводка важнейших характеристик Python.

- *Кросс-платформенность* — Python работает на всех основных платформах, включая Windows, Linux и MacOS. Любой код Python, который вы напишете на своей платформе, будет работать на других компьютерах без ограничений, связанных с различиями между платформами.
- *Выразительность и удобочитаемость* — синтаксис Python проще синтаксиса многих других языков. Выразительный, понятный стиль программирования широко распространен среди питонистов. Вы увидите, что хорошо написанный код Python легко читается, как хорошо написанный текст.
- *Быстрота построения прототипов* — благодаря простоте синтаксиса код Python обычно получается более компактным, чем код, написанный на других языках. Следовательно, для построения работоспособного прототипа на Python требуется меньше работы, чем при использовании других языков.
- *Автономность* — после того, как вы установите Python на своем компьютере, он будет готов к использованию сразу же после «распаковки». Базовый установочный пакет Python содержит все основные библиотеки, необходимые для выполнения любых задач повседневного программирования.
- *Открытый код, бесплатное распространение и расширяемость* — хотя Python работает автономно, это не мешает вам создавать и использовать собственные пакеты. Если другой разработчик опубликовал какой-либо нужный вам пакет, вы сможете установить его однострочной командой, не беспокоясь о лицензии или оплате подписки.

Эти ключевые характеристики привлекли многих программистов, в результате чего сформировалось огромное сообщество разработчиков. Модель открытого кода Python позволяет заинтересованным пользователям вносить свой вклад в язык и экосистему в целом. В табл. 1.1 суммируются сведения о некоторых важных предметных областях и средствах, предоставляемых для них Python. Приведенный список далеко не полон, и вам предстоит самостоятельно исследовать, какие средства предлагает Python для деятельности в интересующей вас области.

Таблица 1.1. Обзор предметно-ориентированных инструментов Python

Предметная область	Инструмент	Краткая характеристика
Веб-разработка	Flask	Микрофреймворк для веб-разработки; хорошо подходит для построения облегченных веб-приложений; гибкий механизм расширения для сторонней функциональности
	Django	Полнофункциональный веб-фреймворк; хорошо подходит для построения веб-приложений, управляемых базами данных; обладает высокой масштабируемостью для корпоративных решений
	FastAPI	Веб-фреймворк для построения прикладных интерфейсов (API); проверка и преобразование данных; автоматическое генерирование веб-интерфейсов API
	Streamlit	Веб-фреймворк для простого построения приложений, ориентированных на данные; популярен среди специалистов data science и МО-инженеров
Data science	NumPy	Специализируется на обработке больших многомерных массивов; высокая вычислительная эффективность; интегрирован во многие библиотеки
	pandas	Гибкий пакет для обработки двумерных данных, сходных с электронными таблицами; широкий набор средств для работы с данными
	statsmodels	Популярный пакет статистических вычислений: линейная регрессия, корреляция, байесовские модели, анализ выживаемости
	Matplotlib	Объектно-ориентированная парадигма для построения гистограмм, точечных диаграмм, круговых диаграмм и других стандартных диаграмм с широкими возможностями настройки
	Seaborn	Простая в использовании библиотека визуализации для создания привлекательной графики; высокоуровневый API на базе Matplotlib
Машинное обучение	Scikit-learn	Широкий диапазон средств предварительной обработки для построения МО-моделей; реализация распространенных алгоритмов МО
	TensorFlow	Фреймворк с высокоуровневым и низкоуровневым API; система визуализации Tensor board; хорошо подходит для построения сложных нейронных сетей
	Keras	Высокоуровневые API для построения нейронных сетей; простота использования; хорошо подходит для построения низкопроизводительных моделей
	PyTorch	Фреймворк для построения нейронных сетей; более интуитивный стиль программирования, чем у TensorFlow; хорошо подходит для построения сложных нейронных сетей
	FastAI	Высокоуровневые API для построения нейронных сетей на базе PyTorch; простота использования

Фреймворки, библиотеки, пакеты и модули

При обсуждении инструментария используются тесно связанные термины — *фреймворки*, *библиотеки*, *пакеты* и *модули*. В разных языках могут использоваться термины с несколько различающимися значениями. Стоит разобраться в том, какой смысл вкладывает в эти термины большинство программистов Python.

Термин «*фреймворк*» является самым широким. Фреймворки предоставляют полный набор функциональных средств, специально спроектированных для выполнения определенной работы на высоком уровне (например, веб-разработки).

Библиотеки являются структурными элементами фреймворков и состоят из пакетов. Предоставляемая библиотеками функциональность разрешает пользователям не разбираться в подробностях работы с используемыми пакетами.

Пакеты предоставляют конкретную функциональность. Точнее говоря, пакеты состоят из модулей, и каждый модуль представляет собой набор тесно связанных структур данных и функций в одном файле (например, файле .py).

1.3. ЧЕГО PYTHON НЕ ДЕЛАЕТ ИЛИ ЧТО ДЕЛАЕТ НЕДОСТАТОЧНО ХОРОШО

Все имеет свои ограничения, есть они и у возможностей Python. Существует много всего, чего Python делать не может или, по крайней мере, не может делать так же хорошо, как альтернативные ему средства. Хотя некоторые люди пытаются «протолкнуть» Python как якобы используемый для всех целей язык, признаем, что на данный момент возможности Python в двух важных областях ограниченны:

- *Мобильные приложения* — в мобильную эпоху у всех есть смартфоны, а приложения используются практически во всех аспектах жизни: для банковских операций, покупок в интернете, заботы о здоровье, общения и, конечно, для игр. К сожалению, на сегодня хороших фреймворков Python для разработки приложений для смартфонов пока нет, несмотря на существование Kivy и BeeWare. Если вы работаете в сфере мобильной разработки, в качестве альтернативы стоит рассмотреть более развитые средства, такие как Swift для приложений iOS или Kotlin для приложений Android. Прагматичный программист выбирает язык, который позволяет создавать продукты, нравящиеся пользователю.
- *Низкоуровневая разработка* — при разработке программных продуктов, взаимодействующих напрямую с оборудованием, Python оказывается не лучшим вариантом. Из-за интерпретируемой природы Python общая скорость выполнения недостаточно высока для разработки низкоуровневых продуктов (скажем, драйверов устройств), требующих моментальной реакции. Если вас интересует

низкоуровневая разработка, вам стоит обратить внимание на альтернативные языки, которые успешнее взаимодействуют с аппаратным уровнем. Например, C и C++ хорошо подходят для разработки драйверов устройств.

1.4. О ЧЕМ ВЫ УЗНАЕТЕ ИЗ КНИГИ

Мы немного поговорили о том, что значит быть прагматичным программистом. Теперь обсудим, как прийти к этой цели. В ходе написания программ вы неизбежно столкнетесь с новыми проблемами из области программирования. В книге определены методы программирования, нужные для решения задач, с которыми вы с большой вероятностью встретитесь на практике.

1.4.1. Ориентация на предметно-независимые знания

Все вещи прямо или косвенно связаны друг с другом, это относится и к знанию Python. В контексте Python эта связь проиллюстрирована на рис. 1.1. На концептуальном уровне функциональность Python и его практическое применение можно представить как три взаимосвязанные сущности.

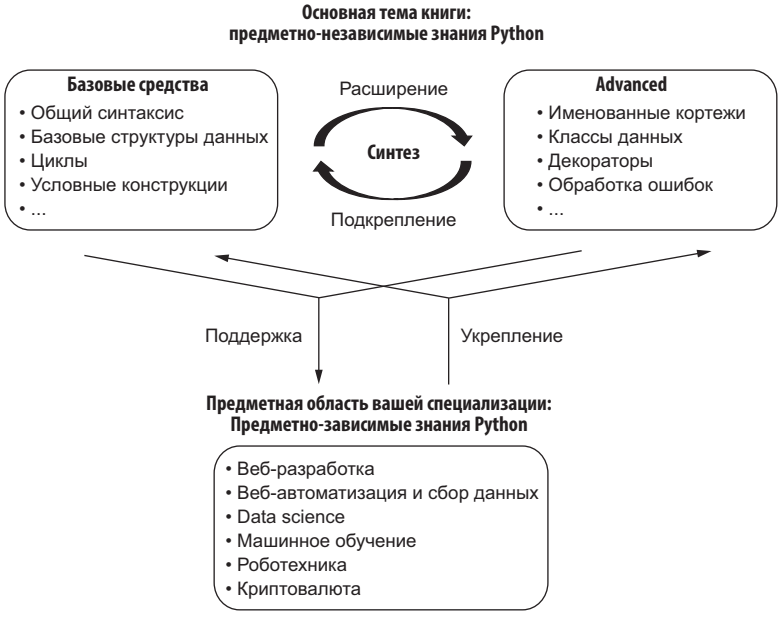

Рис. 1.1. Отношения между предметно-независимыми и предметно-зависимыми областями знаний Python. К предметно-независимым знаниям относятся базовые и расширенные средства Python, тесно связанные друг с другом. Совместно они формируют основу для предметно-зависимых знаний в разных предметных областях

Большинство людей изучают Python, чтобы применять язык для решения задач предметной области, в которой они работают. Для этого необходимы *предметно-зависимые* знания Python, например навыки веб-разработки и анализа данных. Необходимым условием успешного выполнения работы являются базовые навыки Python, а именно *предметно-независимые* знания. Даже если ваши должностные обязанности меняются или развиваются со временем, вы всегда можете применить базовые навыки Python в новой роли.

В этой книге мы сосредоточимся на предметно-независимых знаниях Python. Чтобы упростить процесс обучения, разделим предметно-независимые знания Python условно на два компонента: базовые и расширенные.

Что касается первого компонента, необходимо знать базовые структуры данных и операции с ними. Также нужно уметь вычислять результаты условий для построения инструкций `if...else`... Для выполнения многократно повторяющихся операций можно воспользоваться циклами `for` и `while`, а для повторного использования блоков кода переработать их в функции и классы. Овладения этими основами будет достаточно для написания полезного кода Python, который решает ваши рабочие задачи. Когда вы освоите большую часть базовых навыков, можно переходить к расширенным.

Расширенные навыки позволяют писать еще лучший код, более эффективный и использующий универсальные возможности Python. Чтобы оценить его универсальность, рассмотрим простой пример. При переборе объекта `list` в цикле `for` часто требуется вывести не только сам элемент, но и его позицию:

```
prime_numbers = [2, 3, 5]

# Желательный вывод:
Prime Number #1: 2
Prime Number #2: 3
Prime Number #3: 5
```

Ограничиваясь только базовыми возможностями, мы придем к приведенному ниже решению. В этом решении создается объект `range`, который позволяет извлечь индекс (с начинающейся с 0 нумерацией) для получения позиционной информации. Для вывода применяется конкатенация строк:

```
for num_i in range(len(prime_numbers)):
    num_pos = num_i + 1
    num = prime_numbers[num_i]
    print("Prime Number #" + str(num_pos) + ": " + str(num))
```

Но после прочтения этой книги вы станете более опытным пользователем Python и получите следующее решение, более элегантное и питоническое:

```
for num_pos, num in enumerate(prime_numbers, start=1):
    print(f"Prime Number #{num_pos}: {num}")
```

В этом решении используются три приема: распаковка кортежа для получения `num_pos` и `num` (раздел 4.4), создание объекта `enumerate` (раздел 5.3) и форматирование вывода с использованием f-строк (раздел 2.1). Я не стану подробно описывать эти приемы здесь, потому что они будут рассмотрены в соответствующих разделах. Однако этот пример показывает, чему посвящена данная книга, — *применению различных навыков для создания питонических решений*.

Кроме этих приемов, вы научитесь применять расширенные концепции функций (например, декораторы и замыкания). При определении классов вам будет ясно, как организовать их совместную работу для минимизации объема кода и сокращения риска ошибок. А когда программа будет завершена, вы уже будете знать, как протестировать ваш код, чтобы он был готов к эксплуатации.

Вся эта книга — о синтезе предметно-независимых знаний Python. Вы освоите не только практически полезные расширенные возможности, но и базовые средства Python и фундаментальные концепции программирования (в случаях, где они применимы). Ключевую роль в этом играет термин «*синтез*», что и показано в разделе 1.4.2.

1.4.2. Решение задач посредством синтеза

Начинающие программисты нередко оказываются в сложной ситуации: вроде бы они знают множество разнообразных методов, но не представляют, когда и как использовать их для решения задач. Для каждого рассматриваемого в книге приема я покажу, как он работает автономно, а также продемонстрирую его применение совместно с другими методами. Надеюсь, вскоре вы начнете понимать, как из этих разнородных компонентов строится бесконечное множество новых программ.

Необходимо сделать принципиальное замечание по поводу изучения и синтеза различных приемов: будьте готовы к тому, что путь постижения программирования нелинеен. Ведь и технические возможности Python тесно связаны друг с другом: хотя мы и сосредоточимся на изучении промежуточных и расширенных средств Python, их невозможно полностью изолировать от базовых тем. Вы заметите, что я часто отмечаю базовые средства или намеренно возвращаюсь к уже рассмотренным темам.

1.4.3. Изучение навыков в контексте

Как упоминалось ранее, в книге основное внимание уделяется изучению навыков, основанных на предметно-независимых знаниях Python. Определение «предметно-независимый» означает, что навыки, рассмотренные в книге, применимы к любой области, в которой вы хотите использовать Python. Впрочем, без примеров вряд ли можно что-либо изучить. Многие средства, описанные в книге, будут продемонстрированы на примере текущего проекта, который послужит единым контекстом для обсуждения конкретных навыков. Если какие-то навыки

вами уже освоены, можете переходить к подразделу «Обсуждение», в котором разбираются некоторые ключевые аспекты рассмотренной темы.

Забегая вперед, сообщу, что общим проектом станет веб-приложение для управления задачами (таск-менеджер). В этом приложении пользователь должен выполнять различные операции, включая добавление, редактирование и удаление задач. В нем задействовано все, что может быть реализовано на «чистом» Python: модели данных, функции, классы и вообще все, что обычно встречается в подобных приложениях. Сразу скажу, что целью вовсе не является построение идеально работающего совершенного приложения. В процессе работы над веб-приложением вы будете изучать важнейшие средства Python, чтобы потом применить предметно-независимые знания к вашим рабочим проектам.

ИТОГИ

- Очень важно сформировать стратегию прагматичного обучения. Сосредоточившись на предметно-независимых аспектах Python, вы сможете подготовиться к любой профессиональной роли, связанной с программированием на Python.
- Python — язык программирования общего назначения, распространяемый с открытым кодом. Вокруг него сформировалось огромное сообщество разработчиков, которые создают и распространяют пакеты Python.
- Python конкурентоспособен во многих областях, включая веб-разработку, data science и машинное обучение. В каждой области существуют свои фреймворки и пакеты Python, которыми вы сможете пользоваться.
- Возможности Python не безграничны. Если вы планируете разрабатывать мобильные приложения или низкоуровневые драйверы устройств, используйте Swift, Kotlin, Java, C, C++, Rust или любой другой подходящий для этого язык.
- В книге проводится различие между *предметно-независимыми* и *предметно-зависимыми* знаниями Python. При этом на первое место ставится изучение предметно-независимых знаний Python.
- Путь изучения программирования не линеен. Хотя в книге будут рассматриваться расширенные средства, я также буду часто упоминать о базовых. Кроме того, вы столкнетесь с некоторыми сложными темами, изучение которых требует движения по восходящей спирали.
- Важнейшим рецептом для изучения Python (или любого другого языка программирования) является синтез отдельных технических навыков для формирования их разностороннего набора. В процессе синтеза вы будете изучать язык с прагматичной точки зрения, зная, какие методы подойдут для решаемой вами проблемы.