O'REILLY®

Второе Издание

Простой Python

Современный стиль программирования

Билл Любанович

SECOND EDITION

Introducing Python

Modern Computing in Simple Packages

Bill Lubanovic

Beijing • Boston • Farnham • Sebastopol • Tokyo

O'REILLY®

Простой Python

Современный стиль программирования

Второе издание

Билл Любанович

ПИТЕР®

Санкт-Петербург • Москва • Екатеринбург • Воронеж
Нижний Новгород • Ростов-на-Дону
Самара • Минск

2021

ББК 32.973.2-018.1
УДК 004.43
Л93

Любанович Билл

Л93 Простой Python. Современный стиль программирования. 2-е изд. — СПб.: Питер, 2021. — 592 с.: ил. — (Серия «Бестселлеры O'Reilly»).

ISBN 978-5-4461-1639-3

«Простой Python» познакомит вас с одним из самых популярных языков программирования. Книга идеально подойдет как начинающим, так и опытным программистам, желающим добавить Python к списку освоенных языков.

Любому программисту нужно знать не только язык, но и его возможности. Вы начнете с основ Python и его стандартной библиотеки. Узнаете, как находить, загружать, устанавливать и использовать сторонние пакеты. Изучите лучшие практики тестирования, отладки, повторного использования кода и получите полезные советы по разработке. Примеры кода и упражнения помогут в создании приложений для различных целей.

16+ (В соответствии с Федеральным законом от 29 декабря 2010 г. № 436-ФЗ.)

ББК 32.973.2-018.1
УДК 004.43

ISBN 978-1492051367 англ.

Authorized Russian translation of the English edition of Introducing Python 2E ISBN 9781492051367 © 2020 Bill Lubanovic.
This translation is published and sold by permission of O'Reilly Media, Inc., which owns or controls all rights to publish and sell the same.

ISBN 978-5-4461-1639-3

Краткое содержание

Приложения

Оглавление

Часть II. Python на практике

Приложения

С любовью к Мэри, Тому и Рокси,
а также Кэрин и Эрику.

Введение

Как и обещает название, книга познакомит вас с одним из самых популярных языков программирования — Python. Издание предназначено как для начинающих программистов, так и для тех, кто уже имеет опыт в написании программ и просто желает добавить Python к списку доступных ему языков.

В большинстве случаев изучать компьютерный язык проще, чем человеческий, — в нем меньше двусмысленностей и исключений, которые приходится запоминать. Python — один из самых последовательных и понятных компьютерных языков, он сочетает в себе простоту изучения, простоту использования и большую выразительную силу.

Компьютерные языки состоят из *данных* — аналогами в разговорной речи являются существительные — и *инструкций* (или *кода*), которые можно сравнить с глаголами. Изучить нужно будет и то и другое: вы освоите основы кода и структур данных и узнаете, как их объединить. Затем перейдете к более сложным темам, а программы, которые вы будете читать и писать, станут длиннее и сложнее. Если провести аналогию с работой по дереву, мы начнем с молотка, гвоздей и небольших кусков древесины, а во второй половине книги обратимся к более специализированным инструментам, которые можно сравнить с токарными станками и другими более сложными устройствами.

Вам нужно знать не только сам язык, но и то, что с его помощью можно делать. Мы начнем с языка Python и его стандартной библиотеки, готовой к работе прямо «из коробки». Помимо этого, я покажу вам, как находить, загружать, устанавливать и использовать качественные сторонние пакеты: касаться узких тем или рассматривать сложные трюки не буду, а сделаю акцент на том, что после десяти лет работы с Python считаю действительно полезным.

Несмотря на то что книга представляет собой введение в Python, в ней мы затронем и несколько дополнительных тем, с которыми, на мой взгляд, следует ознакомиться еще на начальном этапе. Работе с базами данных и Интернетом мы тоже уделим внимание, однако не станем забывать, что технологии меняются очень быстро — теперь от программиста на Python общество ждет знаний об облачных технологиях, машинном обучении и создании потоков событий: основную информацию по этим темам вы также здесь найдете.

Язык Python имеет некоторые специальные функции, работающие лучше, чем адаптированные стили программирования из других языков. Например, использование ключевого слова `for` и *итераторов* — более прямой способ создания

цикла: пользоваться им гораздо удобнее, нежели вручную инкрементировать переменную-счетчик.

Когда вы изучаете что-то новое, бывает трудно определиться с тем, какие слова являются терминами и какие понятия на самом деле важны. Иначе говоря, тестировалась ли эта функциональность? Я выделю некоторые термины и понятия, которые имеют особое значение в Python. Код, написанный на языке Python, можно будет увидеть даже в самых первых главах.

Подобные примечания я буду делать, когда что-то может быть непонятным или же существует более питонский способ решить проблему.

Python неидеален. Я обращу ваше внимание на то, что кажется сомнительным и чего следует избегать, и предложу альтернативные варианты.

По некоторым темам, таким как наследование объектов, MVC или проектирование REST для работы с Интернетом, мое мнение может отличаться от общепринятого. Вам самим решать, к кому прислушаться.

Для кого эта книга

Эта книга для всех, кто заинтересован в изучении одного из самых популярных во всем мире языков программирования. Наличие или отсутствие опыта с другими языками программирования не имеет значения.

Что нового во втором издании

Что изменилось со времени выхода первого издания?

- Добавилось около 100 страниц, в том числе с изображением котиков.
- Количество глав удвоилось, но сами главы стали короче.
- В начале книги появилась глава, посвященная типам данных, переменным и именам.
- Рассмотрены новые особенности Python, такие как *f-строки*.
- Рекомендованы новые или улучшенные сторонние библиотеки.
- Во всей книге присутствуют новые примеры кода.
- Для начинающих разработчиков добавлен текст с описанием аппаратного и программного обеспечения.
- Более опытные разработчики могут ознакомиться с библиотекой *asyncio*.
- Рассмотрен новый стек технологий: контейнеры, облачные технологии, наука о данных и машинное обучение.

- Добавлены подсказки, с помощью которых вы сможете найти работу программиста на Python.

Что не изменилось? Примеры, в которых используются плохие стихотворения и утки. Они с нами навсегда.

Структура книги

В первой части излагаются основы языка программирования Python: главы 1–11 следует читать по порядку. Я оперирую простейшими структурами данных и кода, постепенно составляя из них более сложные и реалистичные программы. Во второй части (главы 12–22) показывается, каким образом язык программирования Python используется в определенных прикладных областях, таких как Интернет, базы данных, сети и т. д.: эти главы можно читать в любом порядке.

Вот краткое содержание всех глав и приложений и обзор новых терминов, с которыми вы там встретитесь.

- *Глава 1. «Python: с чем его едят»*. Компьютерные программы не так уж и отличаются от других инструкций, с которыми вы сталкиваетесь каждый день. Мы рассмотрим небольшие программы, написанные на Python. Они продемонстрируют синтаксис языка, его возможности и способы применения в реальном мире. Вы узнаете, как запустить программу внутри *интерактивного интерпретатора (оболочки)*, а также из текстового *файла*, сохраненного на вашем компьютере.
- *Глава 2. «Данные: типы, значения, переменные и имена»*. В компьютерных языках используются данные и инструкции. Компьютер по-разному хранит и обрабатывает разные *типы* данных. Их значения или можно изменять (такие типы называются *изменяемыми*), или нельзя (*неизменяемые* типы). В программе, написанной на Python, данные могут быть представлены как литералами (числами вроде 78 или текстовыми *строками* вроде `"waffle"`), так и именованными *переменными*. В отличие от многих других языков программирования Python относится к переменным как к *именам*, и это влечет за собой некоторые важные последствия.
- *Глава 3. «Числа»*. В этой главе показываются простейшие типы данных, применяемые в языке программирования Python: *булевы переменные*, *целые числа* и *числа с плавающей точкой*. Вы также изучите простейшую математику. В примерах этой главы интерактивный интерпретатор Python используется как калькулятор.
- *Глава 4. «Выбираем с помощью оператора if»*. С существительными (типами данных) и с глаголами (программными структурами) мы поработаем в нескольких главах. Код, написанный на Python, обычно выполняется по одной строке за раз: от начала программы до ее конца. Структура `if` позволяет запускать разные строки кода исходя из результата сравнения определенных данных.
- *Глава 5. «Текстовые строки»*. Здесь мы обратимся к существительным и миру текстовых *строк*. Вы научитесь создавать, объединять, изменять и получать строки, а также выводить их на экран.

- *Глава 6. «Создаем циклы с помощью ключевых слов while и for».* Снова глаголы. Вы научитесь создавать *цикл* двумя способами — с помощью `for` и с помощью `while`, а также узнаете, что такое *итераторы* — одно из основных понятий Python.
- *Глава 7. «Кортежи и списки».* Пришло время рассмотреть первые структуры данных более высокого уровня: списки и кортежи. Они представляют собой последовательности значений, которыми вы будете пользоваться как конструктором Lego для того, чтобы создавать более сложные структуры. Вы научитесь *проходить* по ним с помощью *итераторов*, а также быстро создавать списки с помощью *списковых включений*.
- *Глава 8. «Словари и множества».* Словари и множества позволяют сохранять данные не по позиции, а по их значению. Вы увидите, насколько это удобно, — данная особенность Python станет одной из ваших любимых.
- *Глава 9. «Функции».* Соединяйте структуры данных из предыдущих глав со структурами кода, чтобы выполнять сравнение, выборку или повторение операций. Упаковывайте код в функции и обрабатывайте ошибки с помощью *исключений*.
- *Глава 10. «Ой-ой-ой: объекты и классы».* Слово «объект» недостаточно конкретно, но имеет большое значение во многих компьютерных языках, в том числе и в Python. Если вы уже занимались объектно-ориентированным программированием на других языках, то в сравнении с ними Python покажется вам более простым. В этой главе объясняется, когда следует использовать объекты и классы, а когда лучше выбрать другой путь.
- *Глава 11. «Модули, пакеты и программы».* Вы узнаете, как перейти к более крупным структурам кода — *модулям*, *пакетам* и *программам*, а также где можно разместить код и данные, как ввести и вывести данные, обработать различные параметры, просмотреть стандартную библиотеку Python и то, что находится вне ее.
- *Глава 12. «Обрабатываем данные».* Вы научитесь профессионально обрабатывать данные и управлять ими. Эта глава полностью посвящена текстовым и двоичным данным, особенностям использования символов стандарта Unicode, а также поиску текста с помощью *регулярных* выражений. Вы познакомитесь с типами данных `byte` и `bytearray` — соперниками типа `string`, в которых содержатся необработанные бинарные значения вместо текстовых символов.
- *Глава 13. «Календари и часы».* С датой и временем работать бывает непросто. Здесь мы рассмотрим распространенные проблемы и способы их решения.
- *Глава 14. «Файлы и каталоги».* Простые хранилища данных используют *файлы* и *каталоги*. В этой главе речь пойдет о создании и использовании файлов и каталогов.
- *Глава 15. «Данные во времени: процессы и конкурентность».* Это первая глава, в которой мы приступаем к изучению системы. Начнем с данных во времени — вы научитесь использовать *программы*, *процессы* и *потоки* для того, чтобы выполнять больше работы за один промежуток времени *(конкурентность)*. Среди прочего будут упомянуты последние добавления в *async* (более подробно они рассматриваются в приложении B).

- *Глава 16. «Данные в коробке: надежные хранилища»*. Данные могут храниться в простых файлах и каталогах внутри файловых систем и структурироваться с помощью распространенных форматов, таких как CSV, JSON и XML. Однако по мере того, как объем и сложность данных будут расти, вам, возможно, придется использовать *базы данных* — как традиционные *реляционные*, так и современные базы данных *NoSQL*.
- *Глава 17. «Данные в пространстве: сети»*. Отправляйте ваш код и данные через пространство по *сетям* с помощью *служб*, *протоколов* и *API*. В качестве примеров рассматриваются как низкоуровневые *сокеты*, библиотеки *обмена сообщениями* и системы массового обслуживания, так и развертывание в *облачных* системах.
- *Глава 18. «Распутываем Всемирную паутину»*. Всемирной сети посвящена отдельная глава, в которой рассматриваются клиенты, серверы, извлечение данных, API и фреймворки. Вы научитесь *искать* сайты и *извлекать* из них данные, а затем разработаете реальный сайт, используя параметры *запросов* и *шаблоны*.
- *Глава 19. «Быть питонщиком»*. В этой главе содержатся советы для программистов, пишущих на Python: вы получите рекомендации по установке (с помощью `pip` и `virtualenv`), использованию IDE, тестированию, отладке, журналированию, контролю исходного кода и документации. Узнаете также, как найти и установить полезные пакеты сторонних разработчиков, как упаковать свой код для повторного использования и где получить более подробную информацию.
- *Глава 20. «Пи-Арт»*. При помощи языка программирования Python можно создавать произведения искусства: в графике, музыке, анимации и играх.
- *Глава 21. «За работой»*. У Python есть специальные приложения для бизнеса: визуализация данных (графики, графы и карты), безопасность и регулирование.
- *Глава 22. «Python в науке»*. За последние несколько лет Python стал главным языком науки, он используется в математике, статистике, физике, биологии и медицине. Его сильные стороны — *наука о данных* и *машинное обучение*. В этой главе демонстрируются возможности таких инструментов, как NumPy, SciPy и Pandas.
- *Приложение А. «Аппаратное и программное обеспечение для начинающих программистов»*. Если вы новичок в мире программирования, из этого приложения вы можете узнать, как на самом деле работает аппаратное и программное обеспечение и что означают некоторые термины, с которыми в дальнейшем вам придется сталкиваться.
- *Приложение Б. «Установка Python 3»*. Если вы еще не установили Python 3 на свой компьютер, в этом приложении вы найдете информацию о том, как это сделать независимо от того, какая операционная система у вас установлена: Windows, Mac OS/X, Linux или другой вариант Unix.
- *Приложение В. «Нечто совершенно иное: async»*. В разных релизах Python добавляется функциональность для работы с асинхронностью — разобраться с ней может быть сложно. Я упоминаю о ней в тех главах, в которых заходит речь об асинхронности, но в этом приложении рассматриваю тему более подробно.
- *Приложение Г. «Ответы к упражнениям»*. Здесь содержатся ответы на упражнения, приведенные в конце каждой главы. Не подглядывайте туда, пока не по-

пробуете решить задачи самостоятельно, в противном случае вы рискуете превратиться в козленочка.

- *Приложение Д. «Вспомогательные таблицы»*. В этом приложении содержатся справочные данные.

Версии Python

Языки программирования со временем изменяются — разработчики добавляют в них новые возможности и исправляют ошибки. Примеры этой книги написаны и протестированы для версии Python 3.7. Версия 3.7 является наиболее современной на момент выхода этой книги, и о самых значимых нововведениях я расскажу. Версия 3.8 вышла в конце 2019 года — я рассмотрю самую ожидаемую функциональность[1]. Узнать, что и когда было добавлено в язык программирования Python, можно, посетив страницу https://docs.python.org/3/whatsnew/: там представлена техническая информация. Она, скорее всего, покажется трудной для понимания, если вы только начинаете изучать Python, но может пригодиться в будущем, если вам нужно будет писать программы для компьютеров, на которых установлены другие версии Python.

Условные обозначения

В этой книге приняты следующие шрифтовые соглашения.

Курсив

Им обозначаются новые термины и понятия.

`Моноширинный шрифт`

Используется в листингах программного кода, а также для имен и расширений файлов, названий путей, имен функций, команд, баз данных, переменных, операторов и ключевых слов.

`Курсивный моноширинный шрифт`

Указывает текст, который необходимо заменить пользовательскими значениями или значениями, определяемыми контекстом.

Так оформлены совет, предложение или замечание.

Таким образом оформлено предупреждение.

[1] Оригинальное издание выпущено до выхода версии 3.8. Текущая версия — 3.8.2. — *Примеч. ред.*

Использование примеров кода

Примеры кода и упражнения, приведенные в тексте, доступны для загрузки по адресу https://github.com/madscheme/introducing-python. Эта книга написана, чтобы помочь вам в работе: вы можете применить код, содержащийся в ней, в ваших программах и документации и не связываться с нами, чтобы спросить разрешения, если собираетесь воспользоваться небольшим фрагментом кода. Например, если вы пишете программу и кое-где вставляете в нее код из книги, никакого особого разрешения не требуется. Однако если вы запишете на диск примеры из книги и начнете раздавать или продавать такие диски, разрешение на это получить необходимо. Если вы цитируете это издание, отвечая на вопрос, или воспроизводите код из него в качестве примера, разрешения не требуется. Но если включаете значительный фрагмент кода из данной книги в документацию по вашему продукту, необходимо разрешение.

Ссылки на источник приветствуются, но не обязательны. В такие ссылки обычно включаются название книги, имя ее автора, название издательства и номер ISBN. Например: «Простой Python. Современный стиль программирования. 2-е изд. Билл Любанович. Питер, 2020. 978-5-4461-1639-3».

При любых сомнениях относительно превышения разрешенного объема использования примеров кода, приведенных в данной книге, можете обращаться к нам по адресу permissions@oreilly.com.

От издательства

Ваши замечания, предложения, вопросы отправляйте по адресу comp@piter.com (издательство «Питер», компьютерная редакция).

Мы будем рады узнать ваше мнение!

На веб-сайте издательства www.piter.com вы найдете подробную информацию о наших книгах.

ЧАСТЬ I

Основы Python

ГЛАВА 1
Python: с чем его едят

> Популярными становятся только уродливые языки. Python — исключение из этого правила.
>
> *Дональд Кнут*

Тайны

Начнем с двух небольших тайн и их разгадок. Что, по-вашему, означают следующие две строки?

```
(Ряд 1): (RS) K18, ssk, k1, turn work.
(Ряд 2): (WS) Sl 1 pwise, p5, p2tog, p1, turn.
```

Выглядит как некая компьютерная программа. На самом деле это *схема для вязания* — точнее, фрагмент, который описывает, как связать пятку носка. Похожие носки показаны на рис. 1.1.

Рис. 1.1. Вязаные носки

Для меня эти строки имеют не больше смысла, чем судоку для одного из моих котов, но вот моя жена совершенно точно понимает написанное. Если вы вяжете, то тоже поймете.

Рассмотрим еще один таинственный текст, который можно увидеть записанным на листочке из блокнота. Вы сразу поймете его предназначение, даже если и не догадаетесь о том, каким будет конечный продукт:

```
  1/2 столовой ложки масла или маргарина;
  1/2 столовой ложки сливок;
  2 1/2 стакана муки;
  1 чайная ложка соли;
  1 чайная ложка сахара;
  4 стакана картофельного пюре (охлажденного).

Перед тем как добавить муку, убедитесь, что все ингредиенты охлаждены.
Смешайте все ингредиенты.
Тщательно замесите.
Сделайте 20 шариков.
Держите их охлажденными до следующего этапа.
Для каждого шарика:
  присыпьте разделочную доску мукой;
  раскатайте шарик при помощи рифленой скалки;
  жарьте на сковороде до подрумянивания;
  переверните и обжарьте другую сторону.
```

Даже если вы не готовите, вы сможете распознать кулинарный *рецепт*: список продуктов, за которым следуют указания по приготовлению. Но что получится в итоге? Это *лефсе*, норвежский деликатес, который напоминает тортилью (рис. 1.2). Полейте блюдо маслом, вареньем или чем-либо еще, сверните и наслаждайтесь.

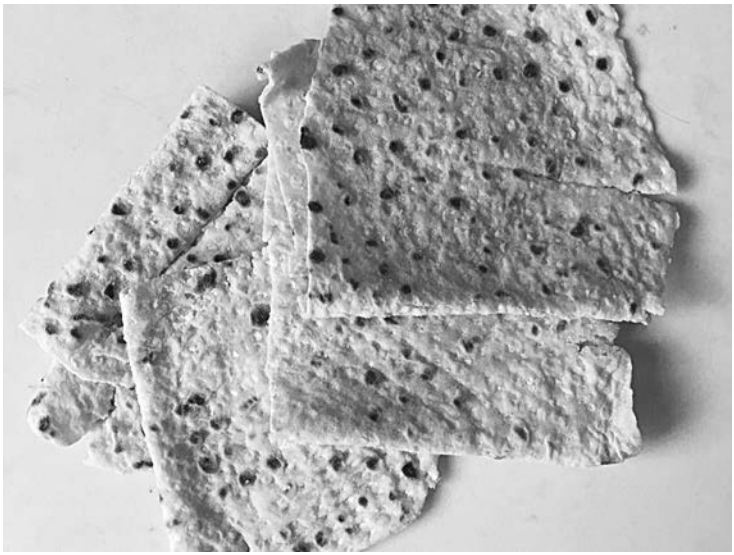

Рис. 1.2. Лефсе

Схема для вязания и рецепт имеют несколько схожих моментов:

- фиксированный *словарь*, состоящий из слов, аббревиатур и символов: какие-то могут быть вам знакомы, какие-то — нет;
- правила, описывающие, что и где можно говорить, — *синтаксис*;
- *последовательность операций*, которые должны быть выполнены в определенном порядке;
- в некоторых случаях — повторение определенных операций *(цикл)*, например способ приготовления каждого кусочка лефсе;
- в некоторых случаях — ссылка на еще одну последовательность операций (говоря компьютерными терминами, *функцию*). Например, когда вы прочтете приведенный выше рецепт, вам может понадобиться рецепт приготовления картофельного пюре;
- предполагаемое знание *контекста*. Рецепт подразумевает ваше знание о том, что такое вода и как ее кипятить. Схема для вязания подразумевает, что вы умеете держать спицы в руках;
- кое-какие *данные*, которые нужно использовать, создать или изменить, — картофель и нитки;
- *инструменты*, которые используются для работы с данными, — горшки, миксеры, духовки, вязальные спицы;
- ожидаемый *результат*. В наших примерах результатом будет предмет для ног и предмет для желудка. Главное — не перепутать.

Как ни назови — идиомы, жаргон, — примеры их использования можно встретить везде. Жаргон помогает сэкономить время тем, кто его знает, а для других людей оставляет информацию совершенно непонятной. Попробуйте расшифровать колонку газеты, посвященную бриджу, если вы не играете в эту игру, или научную статью — если вы не ученый (или ученый, но в другой области).

Маленькие программы

Подобные идеи вы встретите и в компьютерных программах, которые сами по себе являются маленькими языками: через них люди говорят компьютеру, что делать. Схему для вязания и рецепт я использовал для демонстрации того, что программы не так страшны, как может показаться, — всего лишь нужно выучить верные слова и правила.

Понять этот маленький язык гораздо легче, если в нем не очень много слов и правил и если вам не нужно изучать их все одновременно: за один раз наш мозг может воспринять только ограниченное количество знаний.

Пришло время обратиться к настоящей программе (пример 1.1). Как вы думаете, что она делает?

Пример 1.1. countdown.py

```
for countdown in 5, 4, 3, 2, 1, "hey!":
    print(countdown)
```

Если вы считаете, что это программа, написанная на языке программирования Python, которая выводит на экран следующее:

```
5
4
3
2
1
hey!
```

то вы знаете, что Python выучить проще, чем понять рецепт или схему для вязания. К тому же тренироваться писать на этом языке вы можете, сидя за удобным и безопасным столом и избегая опасностей вроде горячей воды и спиц.

Программа, написанная на языке программирования Python, содержит некоторое количество специальных слов и символов: `for`, `in`, `print`, запятые, точки с запятой, скобки и т. д. — все они являются важной частью *синтаксиса* (правил) языка. Хорошая новость заключается в том, что Python имеет более доступный и менее объемный синтаксис по сравнению с большинством других языков программирования: текст кажется почти понятным — как и рецепт.

Пример 1.2 — тоже небольшая программа на Python: она позволяет выбрать одно из заклинаний Гарри Поттера, хранящееся в *списке*, и вывести его на экран.

Пример 1.2. spells.py

```
spells = [
    "Riddikulus!",
    "Wingardium Leviosa!",
    "Avada Kedavra!",
    "Expecto Patronum!",
    "Nox!",
    "Lumos!",
    ]
print(spells[3])
```

Отдельные заклинания являются в Python *строками* (последовательностями текстовых символов, заключенных в кавычки). Они разделены запятыми и помещены в *список* — это можно определить по квадратным скобкам (`[` и `]`). Слово `spells` — это *переменная*, являющаяся именем списка, — с ее помощью мы можем работать со списком. В нашем случае на экран будет выведено четвертое заклинание:

```
Expecto Patronum!
```

Почему мы сказали `3`, если нам нужно было четвертое заклинание? Списки Python, такие как `spells`, представляют собой последовательность значений, доступ к которым осуществляется с использованием *смещения* от начала списка. Смещение для первого элемента списка равно `0`, а для четвертого — `3`.

Люди обычно считают с единицы, поэтому считать с нуля может показаться странным. Однако в программировании удобнее оперировать смещениями, а не позициями. Да, это пример того, как компьютерная программа иногда отличается от обычного языка.

Список — очень распространенная *структура данных* в языке программирования Python. О том, как им пользоваться, будет рассказано в главе 7.

Программа из примера 1.3 выводит на экран цитату одного из участников комедийного трио The Three Stooges («Три балбеса»), однако на выбор фразы влияет не позиция в списке, а то, кто ее сказал.

Пример 1.3. quotes.py

```
quotes = {
    "Moe": "A wise guy, huh?",
    "Larry": "Ow!",
    "Curly": "Nyuk nyuk!",
    }
stooge = "Curly"
print(stooge, "says:", quotes[stooge])
```

Если вы запустите эту небольшую программу, она выведет следующее:

```
Curly says: Nyuk nyuk!
```

`quotes` — переменная, которая именует *словарь* Python: коллекцию уникальных *ключей* (в примере ключом является имя участника трио) и связанных с ними *значений* (в нашем примере — значимое высказывание участника «Балбесов»). Используя словарь, вы можете сохранять элементы и выполнять их поиск по именам: зачастую это удобнее, чем работать со списком.

В примере с заклинаниями для создания списка использовались квадратные скобки (`[` и `]`), а в примере с цитатами для создания словаря — фигурные скобки (`{` и `}`). Также мы использовали двоеточие (`:`) для того, чтобы связать каждый ключ словаря с соответствующим значением. Более подробно о словарях можно прочитать в главе 8.

Надеюсь, я не перегрузил вас синтаксисом. В следующих нескольких разделах вы познакомитесь и с другими простыми правилами.

Более объемная программа

Теперь рассмотрим что-то совершенно иное: в примере 1.4 представлена программа, которая выполняет более сложную серию задач. Не рассчитывайте, что сразу поймете, как она работает, — книга для того и предназначена, чтоб научить вас этому! Таким образом я даю вам возможность увидеть и прочувствовать типичную полноразмерную программу, написанную на языке Python. Если вы знаете другие языки программирования, то можете сравнить их с Python прямо сейчас. Сможете ли вы, не зная Python и еще не прочтя расшифровку, примерно представить, что делает каждая строка? Вы уже видели примеры использования списка и словаря, а эта программа демонстрирует еще несколько новых возможностей.

В первом издании книги программа из примера подключалась к сайту YouTube и получала информацию о самых популярных роликах, таких как *Charlie Bit My Finger*. Она хорошо работала до того момента, как компания Google отключила поддержку этой службы. Во втором издании уже в новом примере (пример 1.4) мы подключаемся к другому сайту, который, очевидно, просуществует гораздо дольше, — *Wayback Machine* из Internet Archive (http://archive.org/) (бесплатного сервиса, сохраняющего миллиарды веб-страниц, в том числе фильмы, телешоу, музыкальные композиции, игры и иные цифровые артефакты за последние 20 лет). Еще несколько примеров таких *веб-API* вы увидите в главе 18.

Программа попросит вас ввести URL и дату. Затем она спросит у Wayback Machine, имеется ли копия этого веб-сайта за указанную дату. Если копия есть, API вернет информацию о ней программе, которая, в свою очередь, выведет URL и отобразит его в веб-браузере. Суть заключается в том, чтобы увидеть, как Python справляется с разнообразными задачами — принимает пользовательские данные, общается с веб-сайтами в Интернете и получает от них данные, извлекает оттуда URL и убеждает веб-браузер отобразить этот URL.

Если бы мы получали обычную веб-страницу, заполненную текстом, отформатированным как HTML, нам пришлось бы сначала придумать, как отобразить ее, а потом выполнить много действий — все это можно радостно перепоручить веб-браузеру. Мы также можем попробовать извлечь именно те данные, которые нам нужны (более подробно о *веб-скрапинге* читайте в главе 18). Любой из выбранных вариантов потребует выполнения большего количества работы и увеличит программу. Вместо этого Wayback Machine возвращает данные в формате JSON. JSON, или JavaScript Object Notation, — это читабельный для человека текстовый формат, который описывает типы и значения, а также выстраивает данные в определенном порядке. Он немного похож на языки программирования и уже стал популярным способом обмена данными между разными языками программирования и системами. Подробнее о JSON вы узнаете в главе 12.

Программы, написанные на языке Python, могут преобразовывать текст формата JSON в *структуры данных* (с которыми вы познакомитесь в следующих нескольких главах), как если бы вы написали программу для их создания самостоятельно. Наша небольшая программа выбирает лишь один фрагмент данных (URL старой веб-страницы, хранящейся в архиве). И опять же это полноценная программа, которую вы можете запустить самостоятельно. Мы почти не проверяли данные на ошибки, чтобы пример оставался коротким. Номера строк не являются частью программы и включены только для того, чтобы вам было проще следовать описанию, представленному после кода.

Пример 1.4. archive.py

```
1 import webbrowser
2 import json
3 from urllib.request import urlopen
4
5 print("Let's find an old website.")
6 site = input("Type a website URL: ")
7 era = input("Type a year, month, and day, like 20150613: ")
8 url = "http://archive.org/wayback/available?url=%s&timestamp=%s" % (site, era)
```

```
9 response = urlopen(url)
10 contents = response.read()
11 text = contents.decode("utf-8")
12 data = json.loads(text)
13 try:
14     old_site = data["archived_snapshots"]["closest"]["url"]
15     print("Found this copy: ", old_site)
16     print("It should appear in your browser now.")
17     webbrowser.open(old_site)
18 except:
19     print("Sorry, no luck finding", site)
```

Такая небольшая программа, написанная на языке Python, делает многое с помощью всего нескольких строк. Не все термины вы уже знаете, однако сможете познакомиться с ними в следующих главах.

1. *Импортируем* (делаем доступным для этой программы) весь код из модуля *стандартной библиотеки*, который называется `webbrowser`.
2. Импортируем весь код из модуля стандартной библиотеки, который называется `json`.
3. Импортируем только *функцию* `urlopen` из модуля стандартной библиотеки `urllib.request`.
4. Пустая строка (мы не хотим перегрузить восприятие).
5. Выводим на экран приветственный текст.
6. Выводим на экран вопрос об URL, считываем пользовательский ввод и сохраняем это в *переменной* с именем `site`.
7. Выводим на экран еще один вопрос и на этот раз считываем год, месяц и день, а затем сохраняем их в переменной с именем `era`.
8. Создаем строковую переменную с именем `url`, чтобы сайт Wayback Machine искал копию требуемого сайта по дате.
9. Соединяемся с сервером, расположенным по этому адресу, и запрашиваем определенный *веб-сервис*.
10. Получаем ответ и присваиваем его переменной `contents`.
11. *Дешифруем* содержимое переменной `contents` в текстовую строку формата JSON и приписываем ее переменной `text`.
12. Преобразуем переменную `text` в `data` — структуру данных языка Python, предназначенную для работы с видео.
13. Проверяем на ошибки: помещаем следующие четыре строки в блок `try` и, если находим ошибку, запускаем последнюю строку программы (она идет после ключевого слова `except`).
14. Получив совпадение по сайту и дате, извлекаем нужное значение из трехуровневого *словаря* Python. Обратите внимание на то, что в этой и двух последующих строках используются отступы — тем самым Python легче понять, что данные строки находятся в блоке `try`.

15. Выводим на экран полученный URL.
16. Сообщаем о том, что случится, когда выполнится следующая строка.
17. Отображаем полученный URL в браузере.
18. Если во время выполнения предыдущих строк что-то пошло не так, Python перейдет сюда.
19. Если программа дала сбой, выводим сообщение и имя сайта, который мы искали. Эта строка также имеет отступ, поскольку должна выполняться только в том случае, если выполняется строка `except`.

Когда я сам запустил эту программу в окне терминала, то ввел URL сайта и дату и получил следующий результат:

```
$ python archive.py
Let's find an old website.
Type a website URL: lolcats.com
Type a year, month, and day, like 20150613: 20151022
Found this copy: http://web.archive.org/web/20151102055938/http://www.lolcats.com/
It should appear in your browser now.
```

На рис. 1.3 показано то, что появилось в моем браузере.

Рис. 1.3. Результат обращения к Wayback Machine

В предыдущем примере мы задействовали стандартные библиотечные модули (программы, включаемые в Python при установке), но совсем не обязательно

ограничиваться только ими: на языке Python написано много отличного стороннего ПО. В примере 1.5 показывается та же программа, получающая доступ к архиву Интернета (Internet Archive), но в ней использован внешний пакет ПО для Python, который называется `requests`.

Пример 1.5. archive2.py

```
1 import webbrowser
2 import requests
3
4 print("Let's find an old website.")
5 site = input("Type a website URL: ")
6 era = input("Type a year, month, and day, like 20150613: ")
7 url = "http://archive.org/wayback/available?url=%s&timestamp=%s" % (site, era)
8 response = requests.get(url)
9 data = response.json()
10 try:
11     old_site = data["archived_snapshots"]["closest"]["url"]
12     print("Found this copy: ", old_site)
13     print("It should appear in your browser now.")
14     webbrowser.open(old_site)
15 except:
16     print("Sorry, no luck finding", site)
```

Новая версия короче и, как мне кажется, более читабельна для большинства людей. О `requests` вы узнаете в главе 18, а о других авторских программах для Python в главе 11.

Python в реальном мире

Стоит ли тратить время и силы на изучение Python? Язык программирования Python существует примерно с 1991 года (он старше Java, но моложе C) и является одним из пяти самых популярных языков программирования. Людям платят деньги за написание программ на Python — очень важных и значимых, которыми мы пользуемся каждый день: Google, YouTube, Instagram, Netflix и Hulu.

Я использовал Python для создания приложений в самых разных областях. Python имеет репутацию высокопроизводительного языка программирования, и это особенно нравится динамично развивающимся компаниям.

Python используется во многих компьютерных приложениях, таких как:

- командная строка на мониторе или в окне терминала;
- пользовательские интерфейсы (Graphical User Interface, GUI), включая сетевые;
- веб-приложения, как клиентские, так и серверные;
- бэкенд-серверы, поддерживающие крупные популярные сайты;
- *облака* (серверы, управляемые сторонними организациями);
- приложения для мобильных устройств;
- приложения для встроенных устройств.

Программы, написанные на Python, могут быть как одноразовыми *сценариями* — вы видели их ранее в этой главе, так и сложными системами, содержащими миллионы строк.

В опросе The 2018 Python Developers' Survey (https://www.jetbrains.com/research/python-developers-survey-2018/) вы можете увидеть числа и графики, показывающие текущее место языка Python в мире вычислительных машин.

Мы рассмотрим применение Python для создания сайтов, системного администрирования и манипулирования данными. Рассмотрим также использование Python в искусстве, науке и бизнесе.

Python против языка с планеты X

Насколько Python хорош по сравнению с другими языками программирования? Где и когда следует использовать тот или иной язык? В этом разделе я покажу примеры кода, написанные на других языках, чтобы вы могли оценить, с чем конкурирует Python. Вы *не* обязаны понимать каждый из приведенных фрагментов, если не работали с этими языками. (А когда увидите последний фрагмент, написанный на Python, то почувствуете облегчение из-за того, что не работали с некоторыми другими языками.) Если же вам интересен только Python — вы ничего не потеряете, если не станете читать этот раздел.

Каждая программа должна напечатать число и немного рассказать о языке, на котором она написана.

Если вы пользуетесь терминалом или терминальным окном, программа, которая читает то, что вы вводите, выполняет это и отображает результат, называется программой-*оболочкой*. Оболочка операционной системы Windows называется cmd (https://ru.wikipedia.org/wiki/Cmd.exe), она выполняет *пакетные* файлы, имеющие расширение `.bat`. Для Linux и других операционных систем семейства Unix (включая macOS) существует множество программ-оболочек. Самая популярная из них называется bash (https://www.gnu.org/software/bash/) или sh. Оболочка обладает простейшими возможностями вроде выполнения простой логики и разворачивания символа-джокера наподобие * в полноценные имена файлов. Вы можете сохранять команды в файлы, которые называются *сценариями оболочки*, и выполнять их позже. Подобные программы могли быть в числе самых первых в вашей карьере программиста. Проблема в том, что возможности для масштабирования у сценариев оболочки ограничиваются несколькими сотнями строк, а сами сценарии выполняются гораздо медленнее, чем программы, написанные на других языках. В следующем фрагменте кода демонстрируется небольшая программа-оболочка:

```
#!/bin/sh
language=0
echo "Language $language: I am the shell. So there."
```

Если вы сохраните этот файл под именем `test.sh` и запустите его с помощью команды `sh test.sh`, то на экране увидите следующее:

```
Language 0: I am the shell. So there.
```

ГЛАВА 2

Данные: типы, значения, переменные и имена

Доброе имя лучше большого богатства.

Притчи 22:1

Все, что хранится в компьютере, представляет собой лишь последовательность *битов* (приложение А). Одно из преимуществ вычислительной техники заключается в том, что мы можем интерпретировать эти биты любым удобным нам способом — как данные всевозможного размера и типов (например, как числа или текстовые символы) или даже как компьютерный код. Мы используем Python для определения наборов этих битов, соответствующих разным задачам, а также для отправки их в процессор и получения обратно.

Мы начнем с *типов данных*, используемых в Python, и *значений*, которые они могут содержать. Затем рассмотрим, как представить данные в виде значений-*литералов* и *переменных*.

В Python данные являются объектами

Память вашего компьютера визуально можно представить в виде длинного ряда полок. Каждый слот на этих полках имеет ширину 1 байт (8 бит). Слоты пронумерованы от 0 (первая позиция) до самого конца. Современные компьютеры имеют миллиарды байт памяти (гигабайт), поэтому полки могли бы заполнить огромный воображаемый склад.

Программа Python получает доступ к определенной области памяти вашего компьютера с помощью операционной системы. Эта память используется для кода самой программы, а также для данных, которыми программа оперирует. Операционная система гарантирует, что программа не может читать или записывать в другие области памяти без соответствующего разрешения.

Программы следят за тем, *где* (область памяти) хранятся их биты и *чем* (тип данных) они являются. С точки зрения вашего компьютера все биты одинаковы.

Одни и те же биты могут иметь разные значения в зависимости от того, какого они типа. Один и тот же вариант расстановки битов может означать как число `65`, так и текстовый символ `A`.

Разные типы используют разное количество битов. Когда вы читаете о «64-битной машине», это означает, что целое число использует 64 бита (8 байт).

Некоторые языки хранят эти необработанные значения в памяти, отслеживая их размеры и типы. Вместо непосредственной обработки таких данных Python упаковывает каждое значение — булевы значения, целые числа, числа с плавающей точкой, строки и даже крупные структуры данных, функции и программы — в память как *объекты*. Глава 10 этой книги посвящена созданию собственных объектов в Python, но сейчас мы поговорим только об объектах, которые обрабатывают встроенные типы данных.

Продолжая аналогию с полками: можно представить, что объекты — это коробки переменной длины, занимающие место на этих полках так, как показано на рис. 2.1. Python создает такие коробки, размещает их на свободных местах и убирает, когда потребность в них отпадает.

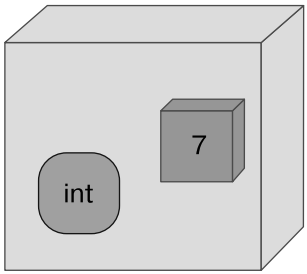

Рис. 2.1. Похожий на коробку объект — целое число со значением 7

В Python объектом является фрагмент данных, в котором содержится как минимум следующее:

- *тип*, определяющий, что объект может делать (подробнее см. в следующем разделе);
- уникальный *идентификатор*, позволяющий отличить его от других объектов;
- *значение*, соответствующее типу;
- счетчик *ссылок* для отслеживания того, как часто объект используется.

Идентификатор представляет собой адрес места на полке. *Тип* похож на фабричный штамп на коробке, который поясняет, что объект может делать. Если объект является целым числом, он имеет тип `int` и может (помимо всего прочего, о чем вы узнаете из главы 3) быть добавлен к другому объекту с типом `int`. Если мы представим, что коробка сделана из прозрачного пластика, то сможем увидеть *значение*, находящееся внутри нее. О том, зачем нужен *счетчик ссылок*, вы узнаете через несколько разделов, когда мы будем говорить о переменных и именах.

Типы

В табл. 2.1 представлены базовые типы данных в Python. Во втором столбце («Тип») содержится имя этого типа в Python. Третий столбец («Изменяемый?») указывает, можно ли изменить значение переменной после ее создания (подробнее об этом поговорим в следующем разделе). В столбце «Примеры» показываются один или

несколько примеров-литералов, соответствующих этому типу. И последний столбец («Глава») указывает на главу этой книги с наиболее подробной информацией о данном типе.

Таблица 2.1. Базовые типы данных в Python

Имя	Тип	Изменяемый?	Примеры	Глава
Булево значение	bool	Нет	True, False	Глава 3
Целое число	int	Нет	47, 25000, 25_000	Глава 3
Число с плавающей точкой	float	Нет	3.14, 2.7e5	Глава 3
Комплексное число	complex	Нет	3j, 5 + 9j	Глава 22
Текстовая строка	str	Нет	'alas', "alack", '"a verse attack"'	Глава 5
Список	list	Да	['Winken', 'Blinken', 'Nod']	Глава 7
Кортеж	tuple	Нет	(2, 4, 8)	Глава 7
Байты	bytes	Нет	b'ab\xff'	Глава 12
Массив байтов	bytearray	Да	bytearray(...)	Глава 12
Множество	set	Да	set([3, 5, 7])	Глава 8
Фиксированное множество	frozenset	Нет	frozenset(['Elsa', 'Otto'])	Глава 8
Словарь	dict	Да	{'game': 'bingo', 'dog': 'dingo', 'drummer': 'Ringo'}	Глава 8

После глав, посвященных этим базовым типам, в главе 10 вы узнаете, как создавать новые типы данных.

Изменчивость

Изменчивость одна лишь неизменна.

Перси Шелли

Тип также определяет, можно ли значение, которое хранится в ящике, изменить — тогда это будет *изменяемое значение*, или оно константно — *неизменяемое значение*. Неизменяемый объект как будто находится в закрытом ящике с прозрачными стенками (см. рис. 2.1): увидеть значение вы можете, но не в силах его изменить. По той же аналогии изменяемый объект похож на коробку с крышкой: вы можете не только увидеть хранящееся там значение, но и изменить его, не изменив его тип.

Python является *строго типизированным* языком, а это означает, что тип объекта не изменяется, даже если его значение изменяемо (рис. 2.2).

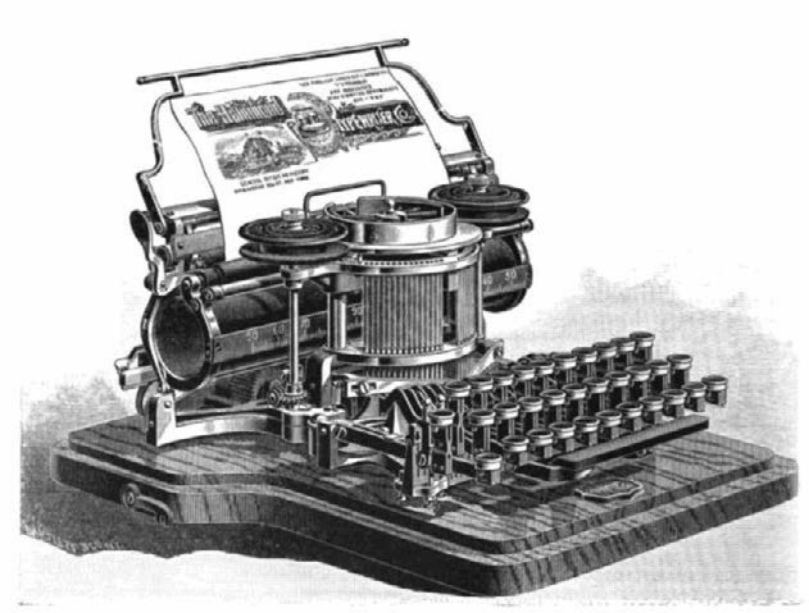

Рис. 2.2. Строгая типизация не означает, что клавиши нужно нажимать сильнее

Значения-литералы

Существует два вида определения данных в Python:

- как литералы;
- как переменные.

В следующих главах вы увидите, как указываются значения-литералы для разных типов данных — целые числа представляют собой последовательность цифр, дробные числа содержат десятичную точку, текстовые строки заключаются в кавычки и т. д. Но в примерах этой главы — чтобы избежать излишней сложности — мы будем использовать лишь короткие целые числа из десятичной системы счисления и один-два списка. Десятичные целые числа такие же, как числа в математике: они представляют собой последовательность цифр от 0 до 9. В главе 3 мы рассмотрим дополнительные детали работы с целыми числами (например, знаки и недесятичные системы счисления).

Переменные

Вот мы и добрались до ключевого понятия языков программирования.

Python, как и большинство других компьютерных языков, позволяет вам определять *переменные* — имена для значений в памяти вашего компьютера, которые вы далее будете использовать в программе.

ГЛАВА 3
Числа

Поступай так, чтобы принести счастье многим людям.

Фрэнсис Хатчесон

В этой главе мы начнем рассматривать простейшие типы данных, используемые в Python:

- *булевы значения* (они могут быть равны `True` или `False`);
- *целые числа* (например, `42` или `100000000`);
- *числа с плавающей точкой* (числа с десятичной точкой вроде `3.14159`, а иногда и экспоненты вроде `1.0e8`, что означает *десять в восьмой степени*, или `100000000.0`).

В каком-то смысле они похожи на атомы. В этой главе мы будем использовать их по отдельности, а впоследствии вы узнаете, как объединить данные простейших типов в более крупные «молекулы», такие как списки и словари.

Для каждого типа определены свои правила использования и обработки компьютером. Я также покажу вам, как использовать литералы (например, `97` или `3.1416`) и *переменные*, которые упоминались в главе 2.

Примеры кода, показанные в этой главе, написаны корректно, но являются всего лишь фрагментами. Мы будем вводить эти фрагменты в интерактивный интерпретатор Python и сразу получать результат. Попробуйте запустить их самостоятельно на своем компьютере (такие примеры вы узнаете по подсказке `>>>`).

Булевы значения

В Python объекты булева (логического) типа данных могут иметь только два значения — `True` или `False`. Иногда вы будете использовать их напрямую, а иногда оценивать «истинность» других типов по их значениям. Специальная функция `bool()` может преобразовать любой тип данных Python в булев.

Функции мы рассмотрим в главе 9, сейчас же вам нужно знать только то, что функция имеет имя, от нуля и более входных *аргументов*, взятых в скобки

и разделенных запятыми, а также от нуля и более *возвращаемых значений*. Функция `bool()` в качестве аргумента принимает любое значение и возвращает его булев эквивалент.

Для ненулевых чисел эта функция вернет значение `True`:

```
>>> bool(True)
True
>>> bool(1)
True
>>> bool(45)
True
>>> bool(-45)
True
```

Нулевые значения преобразуются в значение `False`:

```
>>> bool(False)
False
>>> bool(0)
False
>>> bool(0.0)
False
```

Чем полезны булевы значения, вы узнаете в главе 4. В последующих главах мы поговорим о том, при каких обстоятельствах списки, словари и другие типы данных могут преобразовываться как `True` или `False`.

Целые числа

Целые числа — это целые числа без дробей и десятичной точки, ничего особенного. Кроме, возможно, знака и системы счисления (если вы хотите выразить числа в других системах, а не в десятичной).

Числа-литералы

Любая последовательность цифр в Python представляет собой *число-литерал*:

```
>>> 5
5
```

Можно использовать и простой ноль (`0`):

```
>>> 0
0
```

Но не ставьте его перед другими цифрами:

```
>>> 05
  File "<stdin>", line 1
    05
     ^
SyntaxError: invalid token
```

Исключение предупреждает вас о том, что вы ввели код, который нарушает правила Python. Что это значит, я объясню в подразделе «Системы счисления». В книге вы увидите еще много примеров исключений, поскольку они являются основным механизмом обработки ошибок в Python.

Запись целого числа вы можете начать с конструкций `0b`, `0o` или `0x`. См. подраздел «Системы счисления» далее в этой главе.

Последовательность цифр указывает на целое число. Если вы поместите знак `+` перед цифрами, число останется прежним:

```
>>> 123
123
>>> +123
123
```

Чтобы указать на отрицательное число, вставьте перед цифрами знак `-`:

```
>>> -123
-123
```

При записи целого числа нельзя использовать запятые:

```
>>> 1,000,000
(1, 0, 0)
```

Вместо числа «миллион» вы получите *кортеж* с тремя значениями (более подробная информация о кортежах находится в главе 7). Однако в качестве разделителя вы *можете* использовать символ «нижнее подчеркивание»[1].

```
>>> million = 1_000_000
>>> million
1000000
```

Нижние подчеркивания, которые будут просто проигнорированы, по вашему желанию ставятся на любую позицию после первой цифры:

```
>>> 1_2_3
123
```

Операции с целыми числами

На нескольких следующих страницах вы увидите примеры использования Python в качестве простого калькулятора. Можно выполнять обычные арифметические операции с помощью Python, используя математические *операторы* из этой таблицы.

Оператор	Описание	Пример	Результат
+	Сложение	5 + 8	13
–	Вычитание	90 – 10	80

Продолжение ⇩

[1] В версиях Python 3.6 и выше.

(Продолжение)

Оператор	Описание	Пример	Результат
*	Умножение	4 * 7	28
/	Деление с плавающей точкой	7 / 2	3.5
//	Целочисленное деление	7 // 2	3
%	Вычисление остатка	7 % 3	1
**	Возведение в степень	3 ** 4	81

Сложение и вычитание будут работать так, как вы и ожидаете:

```
>>> 5 + 9
14
>>> 100 - 7
93
>>> 4 - 10
-6
```

Вы можете работать с любым количеством чисел и операторов:

```
>>> 5 + 9 + 3
17
>>> 4 + 3 – 2 – 1 + 6
10
```

Обратите внимание на то, что вставлять пробел между числом и оператором не обязательно:

```
>>> 5+9   +      3
17
```

Такой формат выглядит лучше стилистически и проще читается.

Умножение тоже довольно привычно:

```
>>> 6 * 7
42
>>> 7 * 6
42
>>> 6 * 7 * 2 * 3
252
```

Операция деления чуть более интересна, поскольку существует два ее вида:

- с помощью оператора / выполняется деление *с плавающей точкой* (десятичное деление);
- с помощью оператора // выполняется *целочисленное* деление (деление с остатком).

Даже если вы делите целое число на целое число, оператор / даст результат *с плавающей точкой* (они рассматриваются далее в этой главе):

```
>>> 9 / 5
1.8
```

Целочисленное деление вернет вам целочисленный ответ и отбросит остаток:

```
>>> 9 // 5
1
```

Вместо того чтобы проделать дыру в пространственно-временном континууме, деление на ноль с помощью любого оператора сгенерирует исключение:

```
>>> 5 / 0
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ZeroDivisionError: division by zero
>>> 7 // 0
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ZeroDivisionError: integer division or modulo by z
```

Целые числа и переменные

Во всех предыдущих примерах использовались непосредственно целочисленные значения, но можно смешивать целочисленные значения и переменные, которым было присвоено целочисленное значение:

```
>>> a = 95
>>> a
95
>>> a – 3
92
```

Как вы узнали во второй главе, `a` — это имя, которое указывает на целочисленный объект. Когда мы выполнили операцию `a – 3`, мы не присвоили результат переменной `a`, поэтому ее значение не изменилось:

```
>>> a
95
```

Если вы хотите изменить значение переменной `a`, придется сделать следующее:

```
>>> a = a - 3
>>> a
92
```

Опять же эта операция некорректна с точки зрения математики, но именно так вы выполняете повторное присваивание значения переменной в Python. В Python выражение, стоящее справа от знака `=`, вычисляется первым и только затем присваивается переменной с левой стороны.

Проще думать об этом так.

1. Вычитаем `3` из `a`.
2. Присваиваем результат этого вычитания временной переменной.
3. Присваиваем `a` значение временной переменной:

```
>>> a = 95
>>> temp = a - 3
>>> a = temp
```

ГЛАВА 4
Выбираем с помощью оператора if

О, если ты спокоен, не растерян,
Когда теряют головы вокруг...

Редьярд Киплинг. Если

В предыдущих главах вам встречались самые разные данные, но с ними вы практически не работали. В большинстве примеров использовался интерактивный интерпретатор, а сами они были достаточно короткими. Теперь вы увидите, как структурировать *код* Python, а не только данные.

В большинстве языков программирования такие символы, как фигурные скобки ({ и }) и ключевые слова (например, `begin` и `end`), применяются для разбивки кода на разделы. В этих языках хорошим тоном является использование отбивки пробелами с целью сделать программу более легкочитаемой для себя и других. Существуют даже специальные инструменты, которые помогают красиво выстраивать код.

Гвидо ван Россум, разрабатывая Python, решил, что выделения пробелами достаточно для определения структуры программы, и отказался от ввода разнообразных скобок. Python отличается от других языков тем, что *пробелы* в нем используются для того, чтобы задать структуру программы. Этот аспект новички замечают одним из первых, а тем, кто уже работал с другими языками программирования, такой подход может даже показаться странным. Однако с течением времени все становится на свои места и кажется абсолютно естественным. Вам даже понравится, что вы делаете больше, набирая меньше текста.

Наши первые примеры кода состояли из одной строки. А теперь посмотрим, как создавать комментарии и команды длиной несколько строк.

Комментируем с помощью символа

Комментарий — это фрагмент текста в вашей программе, который будет проигнорирован интерпретатором Python. Вы можете использовать комментарии, чтобы давать пояснение кода, делать какие-то пометки для себя или для чего-либо еще. Комментарий помечается символом `#`: все, что после `#` и до конца текущей строки,

является комментарием. Обычно комментарий располагается на отдельной строке, как показано здесь:

```
>>> # 60 с/мин * 60 мин/ч * 24 ч/день
>>> seconds_per_day = 86400
```

Или на той же строке, что и код, который нужно пояснить:

```
>>> seconds_per_day = 86400 # 60 с/мин * 60 мин/ч * 24 ч/день
```

Символ # имеет много имен: хеш, шарп, фунт или устрашающее *октоторп*. Как бы вы его ни назвали, эффект # действует только до конца строки, на которой он располагается.

Python не дает возможности написать многострочный комментарий. Каждую строку или раздел комментария следует начинать с символа #:

```
>>> # Я могу сказать здесь все, даже если Python это не нравится,
... # поскольку я защищен крутым
... # октоторпом.
...
>>>
```

Однако если октоторп находится внутри текстовой строки, он становится простым символом #:

```
>>> print("No comment: quotes make the # harmless.")
No comment: quotes make the # harmless.
```

Продлеваем строки с помощью символа \

Любая программа становится более удобочитаемой, если ее строки сравнительно короткие. Рекомендуемая (но не обязательная) максимальная длина строки равна 80 символам. Если вы не можете выразить свою мысль в рамках 80 символов, воспользуйтесь *символом продолжения* \ — просто поместите его в конце строки, и дальше Python будет действовать так, как будто это все та же строка.

Например, если бы я хотел создать длинную строку из нескольких коротких, я мог бы сделать это пошагово:

```
>>> sum = 0
>>> sum += 1
>>> sum += 2
>>> sum += 3
>>> sum += 4
>>> sum
10
```

Но мог бы сделать и в одно действие, используя символ продолжения:

```
>>> sum = 1 + \
...       2 + \
...       3 + \
...       4
>>> sum
10
```

Если мы оставим обратный слеш в середине выражения, будет сгенерировано исключение:

```
>>> sum = 1 +
  File "<stdin>", line 1
    sum = 1 +
              ^
SyntaxError: invalid syntax
```

Есть небольшая хитрость: если использовать парные круглые (а также квадратные или фигурные) скобки, Python не будет жаловаться на некорректное окончание строк:

```
>>> sum = (
...     1 +
...     2 +
...     3 +
...     4)
>>>
>>> sum
10
```

В главе 5 вы также увидите, как парные тройные кавычки позволяют создавать многострочные выражения.

Сравниваем с помощью операторов if, elif и else

И вот мы готовы сделать первый шаг к *структурам кода*, которые вводят данные в программы. В качестве первого примера рассмотрим небольшую программу, которая проверяет значение булевой переменной `disaster` и выводит подходящий комментарий:

```
>>> disaster = True
>>> if disaster:
...     print("Woe!")
... else:
...     print("Whee!")
...
Woe!
>>>
```

Строки `if` и `else` в Python выступают *операторами*, которые проверяют, является ли значение выражения (в данном случае переменной `disaster`) равным `True`. Помните, `print()` — это встроенная в Python *функция* для вывода информации, как правило, на экран.

Если ранее вы работали с другими языками программирования, обратите внимание на то, что здесь при проверке if ставить скобки не нужно. Например, не надо писать что-то вроде if (disaster == True). В конце строки следует поставить двоеточие (:). Если вы, как иногда и я, забудете поставить двоеточие, Python выведет сообщение об ошибке.

Каждая строка `print()` имеет отступ под соответствующей проверкой. Я использовал отступ в четыре пробела для того, чтобы выделять подразделы. Вы можете использовать любое количество пробелов, однако Python ожидает, что внутри одного раздела будет применяться одинаковое количество пробелов. Рекомендованный стиль — PEP-8 (http://bit.ly/pep-8) — предписывает использовать четыре пробела. Не применяйте табуляцию или сочетание табуляций и пробелов — это мешает считать отступы.

Все выполненные в этом примере действия позже я объясню более детально.

- Булево значение `True` присваивается переменной `disaster`.
- Производится *условное сравнение* при помощи операторов `if` и `else` с выполнением разных фрагментов кода в зависимости от значений переменной `disaster`.
- *Вызывается функция* `print()` для вывода текста на экран.

Вы можете организовать проверку в проверке столько раз, сколько посчитаете нужным:

```
>>> furry = True
>>> large = True
>>> if furry:
...     if large:
...         print("It's a yeti.")
...     else:
...         print("It's a cat!")
... else:
...     if large:
...         print("It's a whale!")
...     else:
...         print("It's a human. Or a hairless cat.")
...
It's a yeti.
```

В Python отступы определяют, какие разделы `if` и `else` объединены в пару. Наша первая проверка обращалась к переменной `furry`. Поскольку ее значение равно `True`, Python переходит к выделенной таким же количеством пробелов проверке `if large`. Поскольку мы указали значение переменной `large` равным `True`, проверка вернет результат `True`, а следующая строка `else` будет проигнорирована. Это заставит Python запустить строку, размещенную под конструкцией `if large`, и вывести на экран строку `It's a yeti`.

Если нужно проверить больше двух вариантов, используйте оператор `if` для первого варианта, `elif` (это значит `else if` — «иначе если») для промежуточных и `else` для последнего:

```
>>> color = "mauve"
>>> if color == "red":
...     print("It's a tomato")
... elif color == "green":
...     print("It's a green pepper")
... elif color == "bee purple":
...     print("I don't know what it is, but only bees can see it")
```

```
... else:
...     print("I've never heard of the color", color)
...
I've never heard of the color mauve
```

В предыдущем примере мы проверяли равенство с помощью оператора `==`. В Python используются следующие *операторы сравнения*:

- равенство (`==`);
- неравенство (`!=`);
- меньше (`<`);
- меньше или равно (`<=`);
- больше (`>`);
- больше или равно (`>=`).

Эти операторы возвращают булевы значения `True` или `False`. Посмотрим на то, как они работают, но сначала присвоим значение переменной `x`:

```
>>> x = 7
```

Теперь выполним несколько проверок:

```
>>> x == 5
False
>>> x == 7
True
>>> 5 < x
True
>>> x < 10
True
```

Обратите внимание на то, что для *проверки на равенство* используются два знака «равно» (`==`). Помните, что один знак «равно» применяется для присваивания значения переменной.

Если нужно выполнить несколько сравнений одновременно, можно использовать *логические* (или *булевы*) *операторы* `and`, `or` и `not` для определения итогового логического результата.

Булевы операторы имеют более низкий *приоритет* по сравнению с фрагментами кода, которые они сравнивают. Это значит, что результаты фрагментов сначала вычисляются, а затем сравниваются. Поскольку в данном примере мы устанавливаем значение `x` равным `7`, проверка `5 < x` возвращает значение `True`, проверка `x < 10` также возвращает `True`, поэтому наше выражение преобразуется в `True and True`:

```
>>> 5 < x and x < 10
True
```

Как указывается в подразделе «Приоритет операций» на с. 73, самый простой способ избежать путаницы — использовать круглые скобки:

```
>>> (5 < x) and (x < 10)
True
```

Рассмотрим некоторые другие проверки:

```
>>> 5 < x or x < 10
True
>>> 5 < x and x > 10
False
>>> 5 < x and not x > 10
True
```

Если вы используете оператор `and` для того, чтобы объединить несколько проверок, Python позволит вам сделать следующее:

```
>>> 5 < x < 10
True
```

Это выражение аналогично проверкам `5 < x` и `x < 10`. Вы также можете писать более длинные сравнения:

```
>>> 5 < x < 10 < 999
True
```

Что есть истина?

Что, если элемент, который мы проверяем, не является булевым? Чем Python считает `True` и `False`?

Значение `false` не обязательно должно быть явно логическим `False`. Например, к `False` приравниваются следующие значения:

- булева переменная `False`;
- значение `None`;
- целое число `0`;
- число с плавающей точкой `0.0`;
- пустая строка (`''`);
- пустой список (`[]`);
- пустой кортеж (`()`);
- пустой словарь (`{}`);
- пустое множество (`set()`).

Все остальные значения приравниваются к `True`. Программы, написанные на Python, используют это определение истинности или ложности, чтобы выполнять проверку на пустоту структуры данных наряду с проверкой на равенство непосредственно значению `False`:

```
>>> some_list = []
>>> if some_list:
...     print("There's something in here")
... else:
...     print("Hey, it's empty!")
...
Hey, it's empty!
```

Билл Любанович

Простой Python. Современный стиль программирования

2-е издание

Перевел с английского *Е. Зазноба*

Заведующая редакцией	*Ю. Сергиенко*
Ведущий редактор	*Н. Гринчик*
Литературные редакторы	*Н. Кудрейко, Н. Хлебина*
Художественный редактор	*В. Мостипан*
Корректоры	*Е. Павлович, Е. Рафалюк-Бузовская*
Верстка	*Г. Блинов*

Изготовлено в России. Изготовитель: ООО «Прогресс книга».
Место нахождения и фактический адрес: 194044, Россия, г. Санкт-Петербург,
Б. Сампсониевский пр., д. 29А, пом. 52. Тел.: +78127037373.

Дата изготовления: 10.2020. Наименование: книжная продукция. Срок годности: не ограничен.

Налоговая льгота — общероссийский классификатор продукции ОК 034-2014, 58.11.12 —
Книги печатные профессиональные, технические и научные.

Импортер в Беларусь: ООО «ПИТЕР М», 220020, РБ, г. Минск, ул. Тимирязева, д. 121/3, к. 214, тел./факс: 208 80 01.

Подписано в печать 14.10.20. Формат 70×100/16. Бумага офсетная. Усл. п. л. 47,730. Тираж 1000. Заказ 0000.