# PYTHON

## ЗА 7 ДНЕЙ

КРАТКИЙ КУРС
ДЛЯ НАЧИНАЮЩИХ

ЭНДРЮ ПАРК

ЭНДРЮ ПАРК

# PYTHON

## ЗА 7 ДНЕЙ
## КРАТКИЙ КУРС ДЛЯ НАЧИНАЮЩИХ

ПИТЕР®

Санкт-Петербург • Москва • Минск

2023

*Эндрю Парк*

# Python за 7 дней. Краткий курс для начинающих

*Серия «Библиотека программиста»*

*Перевел с английского Е. Матвеев*
*Научный редактор А. Алимова*

ББК 32.973.2-018.1я7 УДК 004.43(07)

**Парк Эндрю**

П18 Python за 7 дней. Краткий курс для начинающих. — СПб.: Питер, 2023. — 256 с.: ил. — (Серия «Библиотека программиста»).

ISBN 978-5-4461-2057-4

Хотите за одну неделю освоить самый популярный язык программирования? Надоело разбираться в море хаотичной и неструктурированной информации из многочисленных бесплатных онлайн-источников?

Хорошая новость! Не нужно тратить время, чтобы осилить сложные академические тексты, неоправданно дорогие онлайн-курсы или видеотуториалы, которые содержат слишком много технических деталей, непонятных для начинающих.

Книга «Python за 7 дней» написана специально для новичков в программировании. Ее основные принципы — это простота и практичность.

Вы познакомитесь с кратким введением в Python, чтобы понять, какую пользу можно извлечь, изучая его; узнаете, как установить Python и какой дистрибутив лучше использовать; разберетесь с объектами и методами (включая ООП), чтобы эффективно использовать этот удобный язык и его простой синтаксис.

Практические упражнения в конце каждой главы идеально подойдут для отработки навыков программирования.

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

ISBN 979-8836767464 англ.
ISBN 978-5-4461-2057-4 рус.



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: 06.2023. 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 02.05.23. Формат 60x90/16. Бумага офсетная. Усл. п. л. 16,000. Тираж 1000. Заказ

# Содержание

# Введение

Компьютеры помогли миру выйти на новый технологический уровень. С нынешнем уровнем производительности и надежности они уже правят миром. Хотя компьютеры иногда называют глупыми машинами, они могут делать то, для чего предназначены, практически идеально. Чтобы компьютеры стали такими производительными, люди, которых мы сегодня называем разработчиками, общались с ними на разных языках программирования годами. Существует несколько разновидностей языков программирования. Подобно тому как люди используют разные языки для разговора в зависимости от региона, компьютеры и разработчики используют языки программирования в зависимости от системы, в которой они работают.

В компьютерной отрасли существует много высокоуровневых языков программирования, но Python особенно популярен и удобен для начинающих. Эта книга, доступно объясняющая базовые идеи Python, поможет начинающим войти в IT-сферу, даже если у них вообще нет опыта программирования.

# Что такое Python

Python — высокоуровневый язык программирования, который завоевал популярность в сообществе разработчиков благодаря своей гибкости, простоте и большому количеству сторонних библиотек и фреймворков, помогающих создавать программные продукты в любой области. Кроме того, Python является одним из самых популярных современных языков, потому что он подходит для начинающих.

Во многих университетах Python преподается как вводный язык программирования для студентов бакалавриата по computer science. Многие онлайн-курсы, изучающие основы программирования, часто используют Python для представления материала. Я рад, что вы выбрали эту книгу — она поможет вам быстро и легко освоить Python.

# Обо мне

Вероятно, поиск в интернете вернет вам тысячи ресурсов, посвященных языку программирования Python. Однако многие новички заходят в тупик в начале изучения языка, потому что у них нет четкого руководства, которому они могли бы следовать.

Меня зовут Эндрю Парк. Я опытный программист с более чем 20-летним стажем в области разработки ПО на Python. Моя любовь к программированию возникла в то время, когда я увлекся видеоиграми. Все началось с моего желания модифицировать игру Pokemon, за которой я проводил много времени. Попытка написать небольшой блок кода, чтобы почувствовать себя чемпионом, еще в юном возрасте разожгла во мне желание разобраться в программной логике и переменных. Имея опыт создания разных игровых режимов, я понял, как работают программы, и стал экспериментировать с разными языками программирования.

Через несколько лет я стал писать небольшие скрипты, автоматизирующие рабочие задачи. Однако в то время я еще не выбрал язык программирования и вряд ли мог считать себя настоящим разработчиком. Все языки программирования которые я опробовал (включая C и Perl), были довольно сложными, из-за чего я неоднократно со злости чуть не бросал программирование. К счастью, в те бурные времена я открыл для себя Python, который только-только появился. Поначалу реализация Python была весьма несовершенной, так как это был простой любительский проект одного разработчика. Но по мере того, как Python привлекал к себе внимание других программистов, все больше людей начинали участвовать в этом проекте с открытым кодом, и Python стал таким

производительным языком программирования, каким мы его знаем сейчас.

Через несколько месяцев изучения азов я начал переписывать свой код на Python. Меня поразило, насколько портируемым и свободным от всего лишнего становился этот код. Когда я освоился с Python, пути назад уже не было. Я начал писать собственные приложения и публиковать их в разных онлайн-магазинах. И хотя моя основная работа была связана с созданием веб-приложений, благодаря Python я успешно реализовал ряд сторонних проектов в других областях.

Хотя сам я неплохо владею Python, мне также хочется помочь другим людям, у которых еще нет такого опыта. Еще с тех времен, когда я занимался простой модификацией игр, мне всегда нравилось помогать другим изучать программирование. Я старался использовать доступные термины для простого объяснения сложных тем и помогал многим своим друзьям и коллегам. Страсть к программированию и обучению подтолкнула меня к тому, чтобы написать эту книгу для тех, кто только начинает знакомство с Python.

## В чем эта книга поможет вам

Программы на Python кажутся очень простыми, но это впечатление обманчиво. В принципе, читателю было бы полезно разбираться в фундаментальных темах Python и приемах, которые позволяют применять их для решения практических задач. Книга предоставляет теоретические знания, которые помогут понять основы и накопить практический опыт работы на Python.

Чтобы получить максимум пользы от книги, рекомендую некоторые приемы когнитивного обучения, которые повысят эффективность усвоения материала.

- Используйте графические схемы связей для установления соответствия между различными концепциями и их быстрой реализации в ваших проектах. Графические схемы связей с помощью наглядного представления на тривиальных диаграммах позволяют запомнить большой объем данных.

- Используйте мнемонические приемы (такие как «дворец памяти» или «метод локусов») для осмысленного запоминания данных. Примитивная зубрежка очень сильно отличается от запоминания только необходимой информации с применением когнитивных методов.

- Используйте метод пассивного повторения для быстрого возвращения ко всем темам, которые вы узнаете из книги. Пассивное повторение поможет закрепить основы.

- Используйте метод Фейнмана и объясните все базовые концепции программирования, о которых вы узнаете в книге, кому-то, кто не разбирается в теме. Если вы сможете объяснить концепцию простыми словами, значит, вы достаточно хорошо усвоили основные положения.

- Не ограничивайтесь использованием кода, приведенного в книге. Реализуйте собственный код с использованием аналогичных стратегий. Простое копирование не научит вас создавать собственный код.

Python — язык программирования, который ожидает от вас нового подхода. Отнеситесь к кодированию на Python как к головоломке, и вскоре вы найдете способы заставить ваш мозг создавать сложную логику реальных задач. Эта книга поможет вам эффективно освоить программирование на языке Python. И я собираюсь отправиться в это путешествие вместе с вами. Готовы?

## Чем вы можете помочь этой книге

Написать эту книгу было непросто. Иногда мне кажется, что провести много часов за отладкой проще, чем написать книгу. Не буду скрывать, что впервые в своей жизни я испытывал состояние творческого тупика. Думаю, это в основном объяснялось необъятностью тем, которые размещались у меня в голове. Тем не менее изложить их в логичном, компактном и упорядоченном виде оказалось намного сложнее.

Стоит упомянуть о том, что я предпочитаю обходиться без услуг издательств. Таким образом, я могу называть себя «независимым автором». Это мое личное решение.

Но теперь я с гордостью могу сказать, что моя *одержимость* идеей помочь тем, кто делает первые шаги в мире программирования, победила. Мне доставит огромное удовольствие, если вы оставите положительный отзыв на Amazon. Для меня это очень много значит, и такие отзывы сильно помогут в распространении материала.

Приятного чтения!

# Глава 1

# Введение в Python

Python — высокоуровневый язык программирования. Он прост, надежен и поддерживает мультипарадигменные рабочие процессы. Python справедливо считается отличной отправной точкой для новичков, желающих приобщиться к миру программирования. Успех Python в основном обусловлен тем, что он избавлен от всего лишнего, а объем рутинного кода сводится к минимуму.

Например, если вы захотите написать простую игру «Змейка» на C или C++, программа будет занимать около 300 строк, а на языке Python количество строк кода удается сократить до 200. Столь заметные различия в программных реализациях помогли Python стать самым популярным языком для проектов с открытым исходным кодом во всем мире. Тысячи энтузиастов создали тысячи библиотек для разных компьютерных областей, благодаря чему Python стал важной вехой на пути революции проектов с открытым кодом.

# История Python

Создатель Python Гвидо ван Россум реализовал Python как любительский проект во время рождественских праздников. Он использовал свой опыт работы над языком программирования ABC, чтобы создать интерпретируемый язык, интуитивно понятный и удобный для программистов. Имея опыт разработки под UNIX, он прежде всего хотел с помощью Python произвести впечатление на хакеров в онлайн-сообществе.

Однако из-за отклика, полученного от коллег-программистов, он начал доводить проект до ума и через несколько месяцев создал язык программирования, который был лаконичным, простым и быстрым. Вследствие своего вклада в проект Python Гвидо ван Россум получил титул «великодушного диктатора» сообщества Python — высочайшая награда, которую может завоевать разработчик проекта с открытым кодом.

Прямо с момента релиза Python неизменно входит в число десяти самых популярных языков программирования согласно рейтингам TIOBE[1]. Минималистский подход к решению задач помог Python опередить другие языки программирования (такие как Perl) и стать одним из самых доступных языков для начинающих.

Python использует философию «у задачи есть только одно решение», что противоречит философии таких языков программирования, как Perl: «существует множе-

[1] Рейтинг, оценивающий популярность языков программирования на основе подсчета результатов поисковых запросов, содержащих название языка. См. https://ru.wikipedia.org/wiki/Индекс_TIOBE. — *Примеч. ред.*

ство решений одной задачи». Python придал сообществу программистов необходимую дисциплину, вследствие чего объем разработки ПО стал расти в геометрической прогрессии.

Чтобы понять, какое влияние оказал Python на программистов по всему миру, взгляните на перечисленные ниже области практического применения этого языка.

# Применение Python

Влияние Python проявилось в целом ряде отраслей современной науки и технологий.

## Веб-программирование

Большая часть влияния Python пришлась в первую очередь на область веб-технологий. Пока в веб-пространстве правил язык Java, Python не пользовался особой популярностью. Со временем сторонние фреймворки (такие как Django и Tornado) помогли Python завоевать популярность у веб-разработчиков.

По прошествии двух десятилетий Python стал одним из самых популярных скриптовых языков для веб-разработки; единственным его конкурентом может считаться только JavaScript. Многие крупные компании — Google, Facebook, Netflix — используют Python в своих продуктах. Знаменитый фреймворк Django помогает программистам писать бэкенд-код для разных API.

Так как язык Python удобен для автоматизации, он часто используется для разработки различных ботов, а также инструментов автоматического поиска и сбора информации.

## Научные вычисления

Язык Python также популярен в научном сообществе из-за своей специфики открытого кода. Такие библиотеки,

как Numpy и SciPy, помогают ученым в области компьютерных наук проводить вычислительные эксперименты с меньшим объемом кода. Кроме того, Python лучше других языков работает с математическими вычислениями и математическими программными продуктами, поэтому в наши дни для ученых практически не существует других альтернатив.

## Машинное обучение и искусственный интеллект

В наше время машинное обучение и искусственный интеллект предоставляют наибольшее количество вакансий для разработчиков. Для Python существует множество сторонних библиотек (например, TensorFlow), направленных исключительно на реализацию алгоритмов машинного обучения.

Python также прекрасно подходит для глубокого обучения и технологий обработки естественных языков, что делает его одним из основных претендентов на роль лучшего языка для разработки технологий, связанных с искусственным интеллектом.

## Linux и управление базами данных

Со стремительным развитием компаний по всему миру растет спрос на инженеров-разработчиков, способных эффективно управлять базами данных и внутренними системами. Хотя инженеры-разработчики должны хорошо разбираться в разных операционных системах (таких как

Linux), они также должны хорошо знать Python для автоматизации разных процедур, необходимых для проверки производительности систем во внутренней сети.

## Тестирование на проникновение и хакерские атаки

Python также используется хакерами по обе стороны баррикады — как «белыми», так и «черными». «Белые» хакеры используют популярные средства Python для проверки методов противодействия проникновению. «Черные» хакеры же, напротив, используют Python-скрипты для создания эксплойтов[1], которые позволяют автоматически добывать конфиденциальную информацию у жертв.

Вследствие высокой адаптируемости Python почти во всех компьютерных областях появился ряд других родственных высокоуровневых языков программирования — таких как Go, Groovy и Swift. Именно благодаря Python минималистская философия программирования стала более популярной.

[1] Фрагменты кода, использующие уязвимости в ПО. — *Примеч. ред.*

# Разные версии Python

Когда Python только появился в начале 90-х, он еще не был полностью отточен. В библиотеке присутствовал ряд ошибок и нестыковок, так как она разрабатывалась Россумом без чьей-либо помощи. Вследствие немедленного успеха, который Python обрел в сообществе программистов в течение двух лет с момента первого выпуска, сотни независимых разработчиков стали помогать Россуму создать более масштабный проект.

Специфика проекта с открытым кодом также помогла Python объединить множество интеллектуалов, которые проверяли и изменяли код там, где это необходимо. За два последних десятилетия базовая команда разработки Python выпустила две основные версии — Python 2 и Python 3.

В 2022 году Python 2 продолжает использоваться многими программистами, хотя эта версия официально не поддерживается основными разработчиками. Выбор версии полностью зависит от проекта, над которым вы работаете.

## Python 2

Python 2 — старая версия, выпущенная в 2000 году. Она считалась самой популярной версией Python на протяжении долгого времени. Python 2 относительно прост, и для него существует намного больше сторонних фреймворков и библиотек для разработки.

Несмотря на отсутствие официальных обновлений с 2021 года, Python 2.7 остается рекомендуемой версией для нескольких программных областей. Миграция всех

фреймворков и библиотек с Python 2 на Python 3 — трудоемкий процесс, поэтому многие компании продолжают использовать Python 2.

## Python 3

Python 3.11.3 — новейшая версия Python от основной команды разработки Python[1]. Python 3 работает быстрее и предоставляет множество дополнительных классов программистам, работающим со стандартной библиотекой. Также ее легче поддерживать по сравнению с Python 2.

## Какую версию выбрать

Выбор версии Python должен зависеть от области, в которой вы работаете. Например, многие специалисты по анализу и обработке данных используют Python 3, тогда как разработчики, применяющие унаследованное ПО, используют Python 2 для интеграции компонентов.

> **ПРИМЕЧАНИЕ**
>
> Весь код Python, встречающийся в книге, написан для Python 3, так как эта версия более актуальна и новичкам логичнее начинать с последней версии.

[1] На момент подготовки русского издания книги. — *Примеч. ред.*

# Преимущества изучения Python

Популярность Python начала расти в начале 1990-х, когда компании по всему миру стали пользоваться доступностью интернета для построения сложных веб-приложений. Традиционные языки — такие как C и C+ — были достаточно сложными, а программисту было трудно за короткое время написать высококачественный код. В это время некоторые компании применяли Python для создания библиотек, которые могли легко использоваться с существующими библиотеками C и C++. Программисты замечали, как удобно работать на Python по сравнению с другими высокоуровневыми языками, и стали переходить на него, чтобы быстрее получить рабочую версию своего кода.

Понимание преимуществ, которые предоставляет Python, поможет вам осознать, насколько простым и надежным Python может быть для разработчиков из различных областей компьютерных наук.

## Python — интерпретируемый язык

В отличие от предшествующих языков программирования, которые пользовались компилятором для выполнения команд, в Python используется новый вычислительный компонент, называемый интерпретатором. Вместо того чтобы тратить время на обработку программы компилятором, интерпретатор применяет современные вычислительные методы для разбора кода еще до итогового выполнения программы. Динамическая обработка кода может сократить время ожидания при выполнении программы. Python также использует

элементы естественного языка для исключения любых непроизводительных процедур, увеличивающих время разработки. Специфика структуры программ также упрощает автоматизацию программирования, благодаря чему язык Python выбирают многие системные разработчики и администраторы Linux.

## Python — язык с открытым кодом

Python — одна из главных причин революции проектов с открытым кодом. Благодаря его открытому характеру вы можете изменить любой код на Python и распространить его самостоятельно. Культура разработки с открытым кодом помогает программистам по всему миру делиться своими знаниями и ресурсами для разработки библиотек и фреймворков, упрощающих создание новых проектов.

Доступ к сложным и простым проектам с помощью одного щелчка кнопкой мыши поможет начинающим понять логику программирования и создать новые инновационные проекты.

## Python поддерживает разные парадигмы

Разные языки программирования используют разные парадигмы создания и развертывания кода. Например, в Java применяется парадигма объектно-ориентированного программирования, тогда как C использует процедурную парадигму. Парадигма программирования изменяет рабочий процесс и методологию, которая применяется разработчиками для решения задачи.

В языке Python поддерживаются различные парадигмы программирования: структурная, функциональная,

объектно-ориентированная и т. д., вследствие чего он становится хорошим вариантом для программистов, применяющих разные подходы для разных задач.

## В Python используется механизм сборки мусора

Управление памятью — важнейшая область для разработчиков приложений. В высокоуровневых языках (таких как Java и C) используются сложные механизмы управления данными. Хотя эти механизмы прекрасно работают, их обслуживание требует значительного времени и усилий. В свою очередь, Python для управления памятью использует сборщик мусора. С этой стратегией в программе можно легко использовать данные и переменные, на которые не существует ссылок.

## Доступность Python

Одна из главных причин популярности Python среди разработчиков — удобочитаемость кода. Весь код хорошо читается, а следовательно, его легко поддерживать. Удобочитаемость способствует повышению качества кода, написанного на Python, а качество сокращает время отладки.

## Портируемость Python

Еще одна важная особенность Python — выполнение в любой операционной системе — позволяет легко развернуть код в разных системах всего за несколько часов. Чтобы программы Python заработали, от пользователя потребуется лишь установить интерпретатор в своей системе.

Например, если разработчик пишет Linux-программу для автоматизации управления базами данных SQL, то каждый пользователь, имеющий доступ к коду, сможет развернуть его на машине с Windows или Mac, изменив всего несколько компонентов кода.

## Превосходные специализированные библиотеки

Чтобы любой язык программирования стал действительно популярным в области современных технологий, ему необходимы хорошие библиотеки. Для Python написано очень много библиотек, с которыми можно экспериментировать.

Кроме специализированных библиотек, в распоряжении программистов также имеются стандартные библиотеки, предоставленные основной командой разработки Python, — они тоже помогают создавать перспективные программы.

## Поддержка интеграции компонентов

Python упрощает интеграцию нового кода с кодом, который был написан ранее. Благодаря расширенным возможностям интеграции компонентов Python хорошо подходит для реализации расширенных средств настройки для разных приложений.

Интеграция компонентов также позволяет разработчикам добавлять новую функциональность в устаревшие программные продукты, чтобы они работали в новых версиях операционных систем.

## Замечательное сообщество

Вокруг Python сформировалось доброжелательное сообщество. Оно помогает новичкам справляться с проблемами, с которыми те сталкиваются при написании кода. Наряду с форумами Python существуют многочисленные ресурсы и руководства, написанные опытными программистами, которые помогают преодолевать любые препятствия в процессе разработки.

В GitHub существует множество Python-проектов с открытым кодом. Любой программист может просмотреть их, чтобы разобраться в реализации сложной логики в своем продукте.

# Установка Python

Чтобы установить программное обеспечение Python, сначала необходимо установить в системе интерпретатор Python. Без интерпретатора разработчик не сможет ни создавать, ни запускать программы на языке Python. Благодаря своей портируемости Python устанавливается в любой современной операционной системе. В этом разделе мы рассмотрим установку Python в трех операционных системах — Linux, Mac и Windows.

## Как установить Python в Linux

Так как большинство программистов использует Linux в качестве основной операционной системы, начнем с установки Python на локальную машину с Linux. Linux — операционная система с открытым кодом, которая используется в основном программистами и государственными организациями. Во многих дистрибутивах Linux Python устанавливается по умолчанию.

Чтобы проверить, установлен ли Python в вашей системе Linux, откройте новый терминал командой Ctrl+Alt+N.

Когда терминал откроется, введите следующую команду.

**КОД В ТЕРМИНАЛЕ:**

```
$ python --version
```

Если Python установлен в вашей системе, терминал выведет лицензионную информацию для установленной версии.

Сообщение command not found[1] будет означать, что Python в вашей системе не установлен. В этом случае вы можете воспользоваться одним из менеджеров пакетов Linux для установки выбранного дистрибутива Python.

Прежде чем устанавливать какие-либо программы в Linux, сначала обновите все служебные инструменты Linux и убедитесь в отсутствии конфликтов, которые могли бы помешать установке Python.

**КОД В ТЕРМИНАЛЕ:**

```
$ sudo apt update
```

Эта команда может использоваться для обновления файлов пакетов, присутствующих в системе Linux на базе Debian.

Если вы хотите обновить пакеты в системе на базе Arch, воспользуйтесь командой `pacman`.

**КОД В ТЕРМИНАЛЕ:**

```
$ sudo pacman -S
```

После того как пакеты будут обновлены, введите одну из приведенных ниже команд для установки Python в системе Linux.

**КОД В ТЕРМИНАЛЕ ДЛЯ СИСТЕМ DEBIAN:**

```
$ sudo apt install python3
```

**КОД В ТЕРМИНАЛЕ ДЛЯ СИСТЕМ ARCH:**

```
$ sudo pacman -S python3
```

[1] «Команда не найдена». — *Примеч. ред.*

За информацией об установке Python в других дистрибутивах Linux (например, Gentoo и kali) обращайтесь к официальной документации Python.

## Как установить Python в macOS

Операционная система macOS используется в устройствах, производимых компанией Apple. Так как система macOS строится с поддержкой UNIX, Python 2 часто устанавливается как часть встроенного программного обеспечения.

Чтобы проверить, установлена ли поддержка Python в системе macOS на оборудовании Apple, откройте терминал командой Settings ▸ Utilities ▸ Terminal.

В открывшемся терминале введите следующую команду.

**КОД В ТЕРМИНАЛЕ:**

```
$ python3 --version
```

Если на экране не появится сообщение с версией Python, это означает, что Python в вашей системе не установлен и вам придется установить его с нуля при помощи менеджера пакетов Homebrew.

**КОД В ТЕРМИНАЛЕ:**

```
$ brew install python3
```

## Как установить Python в Windows

Windows — самая популярная операционная система в мире. Многие программисты и рядовые пользователи работают в Windows, потому что система проста в использовании и для нее написано множество приложений, упрощающих развертывание кода Python-разработчикам.

Чтобы установить Python в системе Windows, необходимо сначала загрузить исполняемый файл с официального сайта Python. После того как файл будет загружен, дважды щелкните на нем, чтобы установить программу. В некоторых версиях Windows также необходимо изменить переменные среды с помощью Панели управления (Control Panel), чтобы система работала со средствами разработки кода Python.

После того как вы сделаете все необходимое, откройте окно командной строки и проверьте правильность установки интерпретатора Python.

**КОД В ТЕРМИНАЛЕ:**

```
>> python --version
```

Если команда выводит информацию об установленной версии, значит, установка Python прошла успешно. В противном случае стоит поискать информацию об ошибке в Google или обратиться за помощью на форумы Python.

# Глава 2

# PyCharm и IDLE

После установки программного обеспечения Python вам понадобится специализированная среда разработки для создания программ в вашей системе. И хотя теоретически ничто не мешает работать в простейшей среде IDLE, входящей в базовую установку Python, программистам рекомендуется пользоваться интегрированной средой разработки (IDE, Integrated Development Environment), например PyCharm, чтобы рабочий процесс проходил на более высоком уровне. Интегрированные среды разработки повышают производительность и упрощают отладку существующего кода в программных продуктах.

# Преимущества интерпретатора Python

Сильные стороны интерпретатора Python — его универсальность и высокий технологический уровень по сравнению с традиционными компиляторами. Например, интерпретатор Python обеспечивает более короткое время ожидания, чем компиляторы. Если компиляторы обрабатывают готовый код, в котором уже устранены ошибки, интерпретатор автоматически проверяет код во время его написания и сообщает программисту о возможных проблемах еще до начала обработки. Получение информации об ошибках в реальном времени удобнее для начинающих программистов, которые учатся прямо в процессе программирования.

При установке Python в системе наряду с основными программными средствами загружается среда IDLE (Integrated Development and Learning Environment). Чтобы запустить ее, введите команду idle в интерфейсе терминала, которым вы предпочитаете пользоваться. Если команда не сработала, IDLE можно найти в каталоге с дистрибутивом Python. IDLE использует механизм REPL[1] для вывода результатов на экран компьютера. REPL — основной метод, который используется интерпретаторами Python для

[1] REPL — сокращение от Read (прочитать ввод от пользователя), Eval (выполнить введенный код), Print (вывести на экран результат), Loop (снова войти в режим ожидания). — *Примеч. ред.*

проверки/разбора введенных команд и вывода результатов на основании пользовательского ввода.

Python IDLE может стать отличным инструментом для тех, кто только начинает осваивать программирование. И хотя большинство серьезных проектов разработки корпоративного ПО ведется в таких интегрированных средах, как PyCharm, знание Python IDLE поможет вам понять, как работает механизм интерпретации Python.

# Как использовать оболочку Python IDLE

После того как Python будет установлен в вашей системе, откройте терминал или командную строку и введите следующую команду, чтобы запустить IDLE.

**КОМАНДА:**

```
$ idle
```

При нажатии клавиши Enter или Return открывается новая командная оболочка.

```
>>>
```

В ней можно ввести простейшие арифметические выражения или команду `print`, чтобы проверить работоспособность Python IDLE в вашей системе.

**ПРОГРАММНЫЙ КОД[1]:**

```
>>> print("This is a sample to check
➥  functioning  of IDLE")
```

**ВЫВОД:**

```
This is a sample to check the functioning of IDLE
```

[1] Здесь и далее значок ➥ показывает перенос строк кода, которые не помещаются на ширину печатной страницы. Следует иметь в виду, что в PEP-8 (руководство по стилю Python) не рекомендуется писать строки кода длиннее чем 79 символов. — *Примеч. ред.*

Нажатие клавиши Enter переводит программу в режим REPL, а текст, заключенный в кавычки, выводится на экран. IDLE распознает функцию `print()`, используемую для вывода строк в окне командной оболочки.

Также для проверки IDLE можно воспользоваться арифметическими операциями.

**ПРОГРАММНЫЙ КОД:**

```
>>> 2 + 5
```

**ВЫВОД:**

```
7
```

## Упражнение

Самостоятельно проверьте вывод других арифметических операций (например, умножения и деления) в окне IDLE.

> **ПРИМЕЧАНИЕ**
>
> Помните, что весь код будет уничтожен сразу же после выхода из окна оболочки, а значит, весь введенный код необходимо сохранить в Python-файле.

# Как открыть файл Python в IDLE

IDLE предоставляет возможность открывать и читать уже существующие файлы с кодом Python (с расширением .py) прямо в терминале. Помните, что приведенная команда сработает только в том случае, если выполнить ее из каталога с файлом Python.

**ПРОГРАММНЫЙ КОД:**

```
$ idle имя_файла.py
```

Эта команда открывает файл с предварительно написанным кодом, чтобы программист мог прочитать его.

Обратите внимание:

- IDLE умеет автоматически подсвечивать элементы синтаксиса;
- IDLE помогает разработчику в написании кода, давая подсказки;
- IDLE упрощает расстановку отступов в коде.

Также можно воспользоваться средствами графического интерфейса — для этого в левом верхнем углу окна оболочки IDLE выберите File ▸ Open, а затем найдите нужный Python-файл.

## Как редактировать файлы

После того как файл будет открыт в IDLE, вы можете приступить к редактированию кода. IDLE отображает

нумерацию строк программы, что позволяет разработчику легко работать с кодом без отступов. После того как файл будет отредактирован, используйте клавишу F5 или щелкните по Run ▸ Run Module для выполнения кода в терминале.

Если в файле нет ошибок, вы получите результаты его выполнения, а если есть — на экране появляется содержимое стека с ошибками.

Хотя среда Python IDLE уступает более современным средам разработки, представленным на рынке, она остается отличным средством отладки. IDLE предоставляет несколько функций быстрой отладки, таких как размещение конечных точек, перехват исключений и разбор кода. Впрочем, эта среда неидеальна и с ростом библиотеки вашего проекта могут возникнуть проблемы.

Пожалуй, при всей своей примитивности IDLE остается лучшим инструментом разработки для начинающих.

## Упражнение

Создайте в Python IDLE программу для суммирования двух чисел. Выполните отладку с использованием точек останова. При решении этой простой задачи можно пользоваться любыми источниками информации в интернете, если какие-то концепции программирования вам неизвестны.

# Интегрированная среда разработки (IDE)

Оболочка Python IDLE не справляется с требованиями сложных проектов, поэтому ее не рекомендуется использовать в реальной разработке. Вместо этого разработчики создают программный код в специализированных программных системах, которые называются интегрированными средами разработки или IDE. Интегрированные среды разработки предоставляют средства тесной интеграции с различными библиотеками.

## Возможности IDE

### Простая интеграция с библиотеками и фреймворками

Одно из важнейших преимуществ IDE — простота интеграции библиотек и фреймворков в приложениях. В IDLE вам пришлось бы подключать их вручную при каждом использовании; IDE выполняет рутинные операции за вас с помощью автозаполнения команд импортирования.

Многие IDE также обеспечивают прямую интеграцию с репозиториями git.

### Интеграция с объектно-ориентированным проектированием

Многие программисты Python, занимающиеся разработкой приложения, используют объектно-ориентированную

парадигму. Python IDLE не предоставляет никаких средств, которые бы упрощали создание приложений на базе принципов объектно-ориентированного программирования. Все современные интегрированные среды предоставляют такие средства, как диаграммы иерархий классов, которые помогают построить более совершенную программную логику на начальном этапе работы над проектом.

## Подсветка синтаксиса

Подсветка синтаксиса улучшает производительность работы программистов и помогает им избежать простых и очевидных ошибок. Например, зарезервированные ключевые слова (`if` и т. д.) не могут использоваться в качестве имен переменных. IDE автоматически распознает эту ошибку и привлекает к ней внимание разработчика при помощи подсветки элементов синтаксиса.

## Автозавершение кода

Во всех современных IDE применяются передовые методы искусственного интеллекта и машинного обучения, которые автоматически завершают программные конструкции за разработчика. IDE собирает большой объем информации из используемых пакетов и предлагает разработчику разные переменные и методы в зависимости от ввода и логики, над которой он работает. Впрочем, при всей полезности автозавершения никогда не стоит полностью полагаться на него, так как иногда оно нарушает ход выполнения программы и порождает ошибки.

## Управление версиями

Управление версиями — один из главных источников проблем для разработчиков. Например, если вы используете в своем приложении частные библиотеки и фреймворки, их обновления могут привести к сбоям в работе приложения. Вам как разработчику необходимо знать об этих изменениях и реализовывать новую логику выполнения, чтобы приложение продолжало работать. Механизм управления версиями позволяет разработчикам легко обновлять код приложений без нарушения уже написанной логики. IDE предоставляет механизмы интеграции управления версиями с такими веб-сайтами, как GitHub.

Кроме этой функциональности, IDE также могут предоставлять различные средства отладки для разработчиков. PyCharm и Eclipse — самые популярные IDE для Python, доступные как для независимых разработчиков, так и для организаций.

В данной книге мы будем в основном использовать PyCharm, так как эта интегрированная среда намного эффективнее Eclipse и проще в настройке.

# PyCharm

PyCharm — специализированная IDE для языка Python — создана JetBrains, одной из передовых компаний в области разработки ПО. Изначально PyCharm разрабатывалась командой JetBrains с целью управления IDE для других языков программирования.

Позднее команда JetBrains выпустила PyCharm как отдельный продукт для пользователей всего мира. Среда PyCharm доступна для всех популярных операционных систем в двух версиях — Community и Professional.

**Community** — бесплатная версия с открытым кодом, которая может использоваться любым желающим для написания кода на Python. Впрочем, она обладает ограниченной функциональностью, особенно в отношении управления версиями и интеграции со сторонними библиотеками.

**Professional** — платная IDE, предоставляющая разработчикам расширенную функциональность и многочисленные возможности интеграции. В версии Professional разработчики могут легко создавать веб-приложения или приложения обработки/анализа данных.

## Какие возможности доступны в PyCharm

Популярность среды PyCharm отчасти обусловлена и другими уникальными возможностями, которые она предоставляет разработчикам Python.

### Редактор кода

Редактор кода, поставляемый с Pycharm, — один из самых лучших в отрасли. Его мастерство в автозавершении кода производит впечатление на каждого, кому довелось работать с новыми проектами в этом редакторе. Специалисты JetBrains использовали сложные модели машинного обучения и наделили среду IDE достаточно высоким интеллектом, чтобы она понимала сложные программные блоки и предоставляла рекомендации для пользователя.

Если вы серьезно занимаетесь разработкой, редактор PyCharm можно настроить для более удобного просмотра кода. Светлая и темная темы оформления позволяет пользователю выбрать внешний вид под свое настроение.

### Навигация по коду

В PyCharm программист может легко управлять файлами, образующими сложную иерархическую систему. Такие специальные возможности, как закладки и режим увеличения, помогают эффективно управлять важнейшими программными блоками и логикой кода.

### Мощные средства рефакторинга

PyCharm предоставляет мощные средства рефакторинга, чтобы разработчик мог легко изменять имена файлов, классов или методов без нарушения работоспособности программы. Если вы попытаетесь провести рефакторинг в IDLE, код немедленно перестает работать, потому что

стандартная оболочка Python IDLE не понимает различий между старыми и новыми именами.

Многие разработчики Python применяют средства рефакторинга для обновления кода или при переходе на новую стороннюю библиотеку, которая лучше подходит для одного из их программных компонентов.

### Интеграция с веб-технологиями

Многие Python-разработчики работают в области веб-технологий, так как она формирует значительную часть индустрии программирования. PyCharm позволяет разработчикам легко интегрировать продукты с такими веб-фреймворками Python, как Django. PyCharm также понимает код HTML, CSS и JavaScript, который обычно используется веб-разработчиками при создании веб-сервисов.

Все эти возможности упрощают интеграцию существующего кода с фреймворками Python.

### Интеграция с научными библиотеками

Среда PyCharm также известна своей качественной поддержкой библиотек для научных и сложных математических расчетов, таких как SciPy и NumPy. И хотя PyCharm не заменит интеграцию и очистку данных, она поможет создать базовую псевдологику для всех проектов обработки и анализа данных.

### Тестирование

PyCharm позволяет применять высокоуровневые стратегии модульного тестирования даже в больших и сложных

проектах с множеством участников. Среда предоставляет современные средства отладки и удаленной настройки для рабочих процессов альфа- и бета-тестирования.

## Как работать с PyCharm

Хочется надеяться, что мне удалось убедить вас в том, что PyCharm является важнейшим инструментом разработки. В этом подразделе приводится информация, которая поможет вам установить PyCharm и понять, как использовать эту среду для совершенствования процессов управления проектами Python.

### Шаг 1. Установка PyCharm

Установка PyCharm в любой операционной системе проходит достаточно прямолинейно. От вас потребуется лишь загрузить пакет установки с официального сайта или с помощью одного из менеджеров пакетов.

Зайдите на официальный веб-сайт JetBrains[1], в меню Developer Tools перейдите на страницу PyCharm и нажмите кнопку Download. Загрузите исполняемый файл или файл dmg (в зависимости от операционной системы) и после завершения загрузки щелкните на нем; затем выполните инструкции, появившиеся на экране.

[1] www.jetbrains.com — *Примеч. ред.*

Если вы хотите загрузить профессиональную версию продукта, то перед загрузкой пробной версии необходимо ввести реквизиты для платежа. После завершения пробного периода произойдет списание средств, и вы сможете пользоваться профессиональной версией без каких-либо проблем.

**ПРИМЕЧАНИЕ**

Чтобы успешно установить среду PyCharm IDE в вашу систему, необходимо убедиться в том, что Python был установлен правильно. Процесс установки IDE автоматически определяет путь к Python для установки основных библиотек продукта.

## Шаг 2. Создание нового проекта

После того как среда PyCharm будет установлена, откройте ее из меню приложений или воспользуйтесь ярлыком на рабочем столе. После запуска PyCharm открывается новое окно для создания проекта с нуля. В левом верхнем углу программного интерфейса расположена команда для открытия нового проекта (File). Также там находятся команды импортирования и экспортирования для загрузки существующих проектов и быстрого сохранения текущих.

Каждый раз, когда вы впервые открываете проект в PyCharm, вам предлагается выбрать интерпретатор Python, который будет использоваться для всех программных операций. Если вы не уверены в том, где находится интерпретатор Python,

выберите вариант virtualenv — в этом случае среда проведет автоматический поиск в системе и найдет интерпретатор Python за вас.

### Шаг 3. Структура проектов PyCharm

Когда вы начинаете проект в PyCharm, очень важно создавать новые папки и ресурсы для ваших программных файлов, чтобы упростить доступ к ним.

Выберите команду new-->folder, чтобы создать новую папку в интерфейсе проекта. В ней можно разместить любые Python-скрипты или дополнительные файлы, используемые в программе.

Каждый раз, когда вы создаете новый файл в отдельной папке, ему присваивается расширение .py. Если вы захотите создать отдельные файлы классов или шаблонов, необходимо явно сообщить об этом при создании файла в папке.

### Шаг 4. Расширенные возможности PyCharm

Когда код написан и интегрирован, вы сможете легко открыть встроенный интерфейс IDLE или специальный интерфейс вывода PyCharm.

Весь написанный вами код будет автоматически сохраняться в реальном времени, а значит, вам не придется беспокоиться о потере критических данных проекта из-за плохого сетевого подключения или сбоя питания. От вас потребуется лишь нажать клавиши Ctrl+S или Cmd+S для сохранения копии проекта в локальной системе.

Завершив работу над программой, нажмите Shift+F10, чтобы запустить и скомпилировать код с помощью интерпретатора.

Комбинации клавиш Ctrl+F или Cmd+F используются для поиска любых методов, переменных или фрагментов, применяемых в проекте. Просто нажмите клавиши и введите уточняющую информацию о том, что вы ищете.

После того как код Python будет импортирован и развернут в нужной операционной системе, следует настроить среду отладки для устранения ошибок. Просто нажмите клавиши Shift+F9, чтобы установить точки останова и исправить логические ошибки без нарушения программной логики или внесения новых дефектов.

# Руководство по стилю Python

Программирование Python стало чрезвычайно популярным среди программистов из-за философии, которую поддерживал и продолжает поддерживать этот язык. Python стремился к простоте там, где другие высокоуровневые языки напрасно усложняют задачу для среднего программиста. Perl — прекрасный пример такого неудобного языка.

Создатели Python предлагали первым энтузиастам-питонистам следовать простому набору четко сформулированных принципов (объединенных под названием «Дзен Python») для создания кода, который не только хорошо работает, но и хорошо выглядит. Даже через 20 лет после публикации эти принципы остаются актуальными, и каждый программист Python должен знать их.

Чтобы прочитать формулировку этих принципов, введите в терминале следующий код.

**КОД В ТЕРМИНАЛЕ:**

```
$ python
$ import this
```

Рассмотрим самые важные из принципов, чтобы вы лучше поняли философию, которую Python продвигает среди разработчиков.

- **Красивое лучше, чем уродливое.**

Всем Python-разработчикам рекомендуется писать семантически симметричный код и следить за тем, чтобы этот

код был красивым. Красивый код должен иметь четкую структуру; это означает, что программисты должны писать условные конструкции без усложнения кода. Кроме того, правильная расстановка отступов также визуально улучшает код. Таким образом, он лучше читается, а иногда и быстрее выполняется.

- **Явное лучше, чем неявное.**

Многие разработчики по неизвестной причине пытаются сделать свою программную логику неявной, что затрудняет ее понимание другими программистами. Python старается переломить эту привычку: разработчикам рекомендуется явно формулировать логику своего кода, чтобы она была понятна всем. Кроме того, это одна из причин, почему открытый код более популярен в библиотеках и фреймворках Python.

- **Простое лучше, чем сложное.**

Вы как Python-разработчик должны стремиться к написанию простого кода. Сознательное желание упростить код повысит вашу квалификацию в используемом языке программирования. По мере накопления практического опыта ваше умение писать менее запутанный код также улучшится.

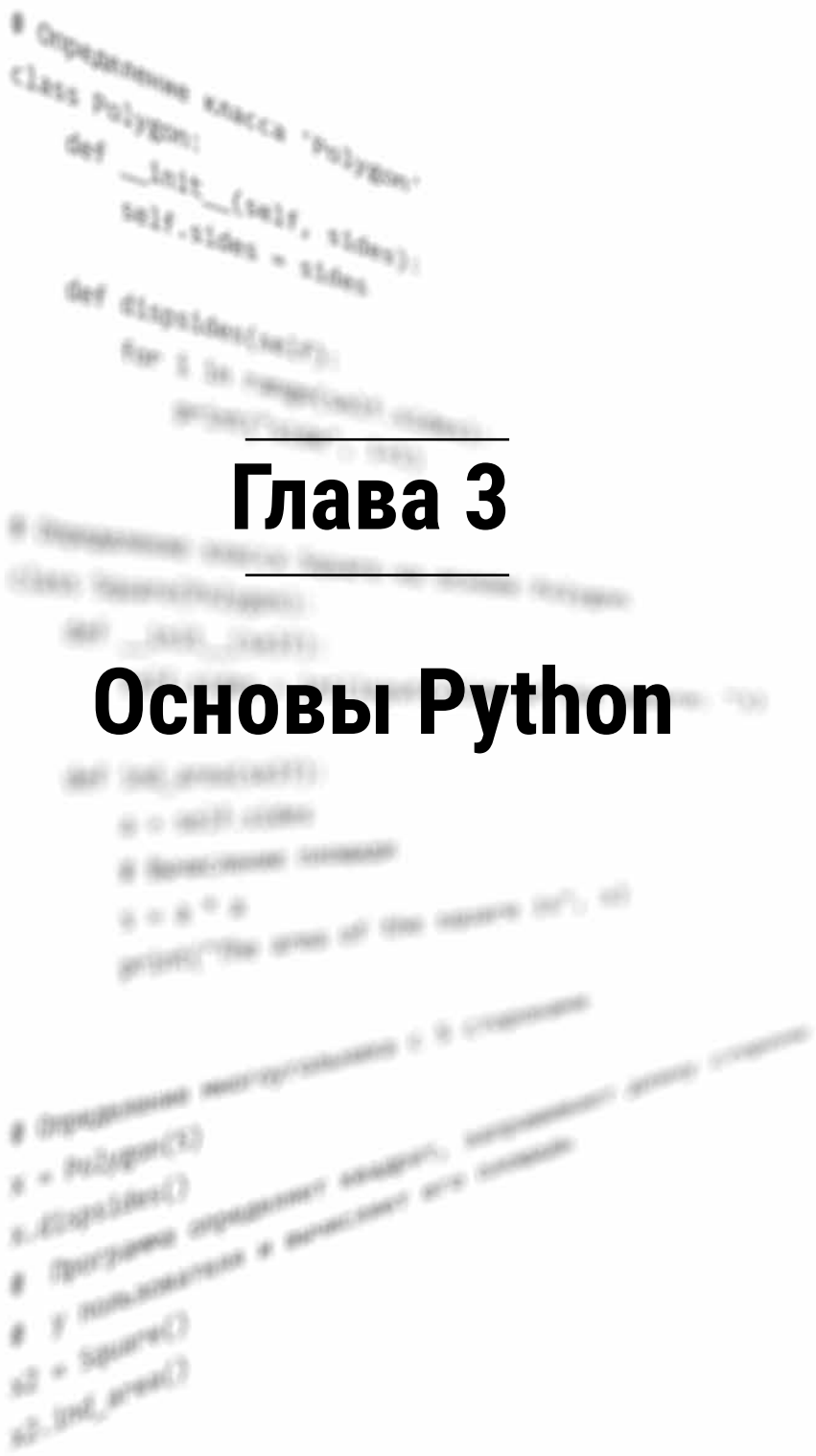
- **Сложное лучше, чем запутанное.**

В любом проекте иногда приходится писать сложный код, способный решать сразу несколько задач. Работая над сложным кодом, следите за тем, чтобы он не становился запутанным. Эффективное использование исключений и файлов поможет быстро свести запутанный код,

в котором позднее могут обнаружиться коварные ошибки, к минимуму.

- **Должно быть одно — и желательно только одно — очевидное решение.**

Python поощряет единообразие, в отличие от своих языков-предшественников C и C++. Python-разработчик использует одну логику для разных экземпляров, которые использовались в программе. Единообразие обеспечивает гибкость и упрощает поддержку кода.



# Глава 3

# Основы Python

Приложения Python должны быть динамическими, иначе говоря, они должны получать входные данные непосредственно от пользователя и выдавать соответствующий результат. Интерпретатор Python и все функции вашей программы могут обращаться к этим входным значениям.

В главе я приведу несколько примеров программ и покажу, как сделать их более удобными для пользователя при помощи операций ввода и вывода.

# Почему входные данные необходимы

Входные данные обеспечивают практическую пользу ваших приложений. Все программы — от веб-приложений до новейших приложений виртуальной реальности — работают на основании входных данных, предоставляемых пользователем. Например, при входе в социальную сеть необходимо ввести адрес электронной почты и пароль. Эти значения называются входными данными, и доступ к своей учетной записи удается получить только в том случае, если введенная информация верна.

Даже такие сложные приложения, как системы распознавания лиц, используют информацию о лицах в качестве входных данных. Каждое реальное приложение в наши дни получает и собирает данные, вводимые пользователем, чтобы настроить работу приложения под его потребности.

## Сценарии использования

Допустим, вы разработали приложение на Python для взрослой аудитории. Это значит, что оно должно быть недоступно для пользователей младше 18 лет.

В таком случае можно реализовать условную проверку возраста, введенного пользователем. Если введенное значение больше или равно 18, приложение становится

доступным для пользователя, а если возраст меньше 18 — в доступе будет отказано. Конечно, для проверки того, может ли пользователь работать с вашим приложением, Python может получать данные любых поддерживаемых типов — мы всего лишь рассмотрели один реальный пример. Получение входных данных от пользователей имеет бесконечное практическое применение.

# Функция input()

Если вызвать функцию `input()` во время выполнения программы Python, интерпретатор приостанавливает работу и ожидает, когда пользователь введет данные с устройства ввода (с помощью клавиатуры, мыши или на сенсорном экране мобильного устройства).

При вводе данных пользователь обычно руководствуется подсказкой, которая выдается приложением. При разработке реальных приложений необходимо создать хороший графический интерфейс для вывода подсказок. В этой главе мы рассмотрим некоторые варианты создания текстовых подсказок, которые могут использоваться разработчиками.

После того как данные будут введены, пользователь должен нажать клавишу Enter, чтобы интерпретатор возобновил работу и обработал логические конструкции, использованные в программе.

**ПРИМЕР:**

```
sample = input("Which country do you belong to?: ")
print(sample + " is a great country"))
```

Если запустить эту программу, пользователь сначала увидит текст подсказки, как показано ниже.

**ВЫВОД:**

```
Which country do you belong to?: China
China is a great country
```

Попробуйте ввести название другой страны и посмотрите, что произойдет.

**ВЫВОД:**

```
Which country do you belong to?: France
France is a great country
```

## Как составлять подсказки для пользователя

При применении функции `input()` для получения данных от пользователя желательно использовать содержательные подсказки, чтобы привлечь внимание.

> **ПРИМЕЧАНИЕ**
>
> Не включайте в текст лишнюю информацию. Подсказки должны быть по возможности простыми и короткими.

**ПРОГРАММНЫЙ КОД:**

```
example = input("Which is your favorite football
➥  team?: ")
print("So you are a "+ example + " fan. Hurray!")
```

**ВЫВОД:**

```
Which is your favorite football team?: Liverpool
So you are a Liverpool fan. Hurray!
```

Функция `input()` также может выводить подсказки, состоящие из нескольких строк.

**ПРОГРАММНЫЙ КОД:**

```
prompt = "This is a simple question to know what
➥  you like."
prompt += "\nSo, please say your favorite place: "
example = input(prompt)
print(example + " is a great place to visit")
```

**ВЫВОД:**

```
This is a simple question to know what you like
So, please say your favorite place: Paris
Paris is a great place to visit
```

С самого начала книги мы будем использовать функцию `print()` для вывода результатов. `print()` — один из рекомендуемых способов вывода текста на экран.

Любая информация, переданная функции `print()`, преобразуется в строковый литерал и выводится на экран. Знать аргументы функции `print()` обычно не обязательно, но некоторые из них помогут вам в форматировании кода, и их желательно изучить.

## Что такое экранированная последовательность

Экранированные последовательности — особые серии символов, предназначенные для быстрого форматирования данных. Например, `\n` — часто используемая последовательность, которая позволяет вывести данные с новой строки.

`\t` и `\b` — еще две популярные экранированные последовательности. `\t` позволяет выводить данные со следующей позиции табуляции, а `\b` удаляет один символ перед курсором. Если последовательность `\b` находится в конце строки, удаления не происходит.

## Для чего нужна команда end

Функция `print()` также принимает аргумент `end` для текста, который добавляется после строки, как в следующем примере.

**ПРОГРАММНЫЙ КОД:**

```
print("France is a beautiful country,", end=" Isn't
➥  it true?")
```

**ВЫВОД:**

```
France is a beautiful country, Isn't it true?
```

В этом примере `'Isn't it true'` — присоединенный текст.

# Комментарии в Python

Когда команда программистов трудится над сложным проектом, участникам приходится постоянно обмениваться информацией друг с другом, чтобы понять суть проекта. Комментарии позволяют разработчикам делиться информацией, не нарушая работу программы.

Интерпретатор игнорирует комментарии и переходит к следующей строке. Поскольку на Python написано множество проектов с открытым кодом, комментарии могут помочь разработчикам понять, как интегрировать сторонние библиотеки и фреймворки в код.

Комментарии также упрощают чтение кода, а следовательно, делают его более понятным. Казалось бы, программисту не нужно напоминать, как работает написанный им код но вы будете удивлены тем, насколько часто разработчик забывает логику своей же программы. Краткие напоминания о том, как работает алгоритм, оказываются очень полезными.

Python поддерживает две разновидности комментариев: однострочные и многострочные.

## Однострочные комментарии

Однострочные комментарии особенно популярны у Python-разработчиков, так как они могут легко чередоваться с кодом.

Чтобы добавить в программу однострочный комментарий, вставьте символ #. Все строчные символы, следующие после него, будут игнорироваться компилятором.

**ПРОГРАММНЫЙ КОД:**

```
# Пример однострочного комментария
print("This is just an example.")
```

**ВЫВОД:**

```
This is just an example.
```

Интерпретатор проигнорировал однострочный комментарий и выполнил только команду `print`.

### Для чего используются однострочные комментарии

Однострочные комментарии используются прямо в коде. Они помогают другим программистам понять, как работает логика программы, и описывают назначение реализованных переменных.

## Многострочные комментарии

Вообще говоря, ничто не мешает использовать однострочные комментарии для длинной записи, состоящей из трех-четырех строк. Тем не менее поступать так не рекомендуется, потому что Python предоставляет более удобный способ записи многострочных комментариев.

Программисты Python могут использовать строковые литералы для создания многострочных комментариев, как показано в следующем примере.

**ПРОГРАММНЫЙ КОД:**

```
"""
This is a comment
In Python
with multiple lines
Author: Python Rookie
"""
print("This is just an example.")
```

**ВЫВОД:**

```
This is just an example.
```

Как и в случае с однострочными комментариями, при запуске этой программы выполняется только команда `print`.

### Для чего используются многострочные комментарии

Программисты часто используют многострочные комментарии для определения условий лицензии или изложения подробной информации о разных пакетах и функциях с примерами реализации. Такие блоки хорошо воспринимаются программистом, читающим код.

# Зарезервированные ключевые слова

Зарезервированные ключевые слова представляют собой стандартные ключевые слова языка программирования, которые не могут использоваться разработчиками при написании кода в качестве идентификаторов: имен переменных, классов и функций.

Если вы попытаетесь использовать зарезервированное ключевое слово в качестве имени в своей программе, интерпретатор не позволит этого сделать и сообщит об ошибке. Например, если присвоить переменной имя `for`, программа работать не будет, потому что в Python ключевое слово `for` обычно используется для определения специальной разновидности циклов.

В настоящее время существует около 30 зарезервированных ключевых слов, которые не могут использоваться в качестве имен в программах. Программистам Python желательно знать их, чтобы не допускать лишних ошибок при создании сложных проектов.

## Упражнение

Чтобы лучше понять суть команд Python, которые приводились ранее, попробуйте самостоятельно найти ключевые слова Python в терминале.

# Операторы Python

Программисты обычно используют операторы для совершения различных операций и формирования команд или выражений.

**ПРИМЕР:**

```
2x + 3z = 34
```

Здесь `2x`, `3z` и `34` — операнды, а `+` и `=` — операторы, используемые с операндами для формирования выражения.

Изначально операторы использовались в математике для построения математических выражений. Первые пользователи языков программирования позаимствовали операторы и некоторые другие базовые компоненты программирования для удобства присваивания и изменения значений. Операторы позволяют создавать сложные выражения, которые помогают программистам в реализации нетривиальных алгоритмов.

**ПРИМЕР:**

```
a = 32
b = 34
print(a + b)
```

**ВЫВОД:**

```
66
```

Здесь `a` и `b` — операнды, а `=` и `+` — операторы.

# Разновидности операторов

Существует несколько разновидностей операторов, которые используются программистами для реализации программной логики. Самая популярная категория — арифметические операторы, позволяющие реализовать математическую логику для разных операндов например переменных, определяемых в коде.

Операторы сложения, вычитания, умножения и деления — арифметические операторы, которые нужны для реализации простейших вычислений в программе.

## Сложение

Оператор сложения предназначен для суммирования двух операндов в программе. Операндами могут быть переменные и объекты разных типов; кроме того, в некоторых случаях возможно суммирование данных двух отличающихся типов. Интерпретатор Python достаточно разумен, чтобы выполнить необходимое преобразование типа и предоставить результат программисту. Оператор сложения обозначается символом +.

**ПРОГРАММНЫЙ КОД:**

```
x = 54
y = 34
z = x + y
# Знак + является оператором сложения
print(z)
```

При выполнении программы в IDE или IDLE интерпретатор суммирует значения двух переменных и сохраняет результат в переменной z.

**ВЫВОД:**

```
88
```

## Вычитание

Оператор вычитания вычитает одно значение из другого. Операндами могут быть переменные и объекты разных типов; кроме того, в некоторых случаях возможно вычитание данных двух отличающихся типов. Оператор вычитания обозначается символом -.

**ПРОГРАММНЫЙ КОД:**

```
x = 54
y = 34
z = x - y
# Знак - является оператором вычитания
print(z)
```

При выполнении программы в IDE или IDLE интерпретатор вычисляет разность двух операндов и сохраняет ее в переменной z.

**ВЫВОД:**

```
20
```

## Умножение

Оператор умножения вычисляет произведение двух значений в программе. Операндами могут быть переменные и объекты разных типов; кроме того, в некоторых случаях возможно умножение данных двух отличающихся типов. Операция умножения обозначается символом *.

**ПРОГРАММНЫЙ КОД:**

```
x = 5
y = 3
z = x * y
# Знак * является оператором умножения
print(z)
```

При выполнении программы в IDE или IDLE интерпретатор вычисляет произведение двух операндов и сохраняет его в переменной z.

**ВЫВОД:**

```
15
```

## Деление

Оператор деления вычисляет частное двух операндов. Операндами могут быть целые числа и числа с плавающей точкой, а оператор деления обозначается символом /.

**ПРОГРАММНЫЙ КОД:**

```
x = 6
y = 3
z = x / y
# Знак / является оператором деления
print(z)
```

При выполнении программы в IDE или IDLE интерпретатор вычисляет частное двух операндов и сохраняет его в переменной z.

**ВЫВОД:**

```
2.0
```

### Остаток от деления

Данный оператор определяет остаток от целочисленного деления. Вычисление остатка часто применяется в программной логике и осуществляется с помощью символа %.

**ПРОГРАММНЫЙ КОД:**

```
x = 7
y = 3
z = x % y
# Знак % — оператор вычисления остатка от деления
print(z)
```

При выполнении программы в IDE или IDLE интерпретатор вычисляет остаток от целочисленного деления двух операндов и сохраняет его в переменной z.

**ВЫВОД:**

```
1
```

В данном случае частное от деления двух чисел равно 2,12, а остаток от целочисленного деления равен 1, поэтому это число выводится как результат работы программы.

Чтобы получить результат деления без дробной части, используйте операцию целочисленного деления.

## Целочисленное деление

Целочисленное деление — альтернативный арифметический оператор, который часто используется разработчиками, если для них не важна точность результата. Обычно этот оператор выводит целочисленное значение, ближайшее к частному, получаемому в результате деления.

Оператор целочисленного деления обозначается символами `//`.

**ПРОГРАММНЫЙ КОД:**

```
x = 12
y = 5
z = x // y
# Оператор целочисленного деления — //
print(z)
```

**ВЫВОД:**

```
2
```

В действительности результат деления в этой программе равен 2,4, но, так как мы используем оператор целочисленного деления, программа в качестве результата возвращает ближайшее целое число.

## Побитовые операторы

Побитовые операторы часто используются опытными программистами в таких специальных областях, как сжатие, шифрование и обнаружение ошибок.

Во всех высокоуровневых языках программирования поддерживаются следующие побитовые операторы.

1. `AND` (`&`)
2. `OR` (`|`)
3. `XOR` (`^`)
4. `NOT` (`~`)

Все эти побитовые операторы работают по принципам, которые должны быть известны вам по традиционной (булевой) логике.

# Приоритет операторов

При построении математических выражений образуются комбинации разных операторов. Когда вы создаете нетривиальные математические выражения в ходе разработки реальных приложений, ситуация быстро усложняется. Система приоритетов предоставляет четкие правила, определяющие порядок применения операторов в математических операциях.

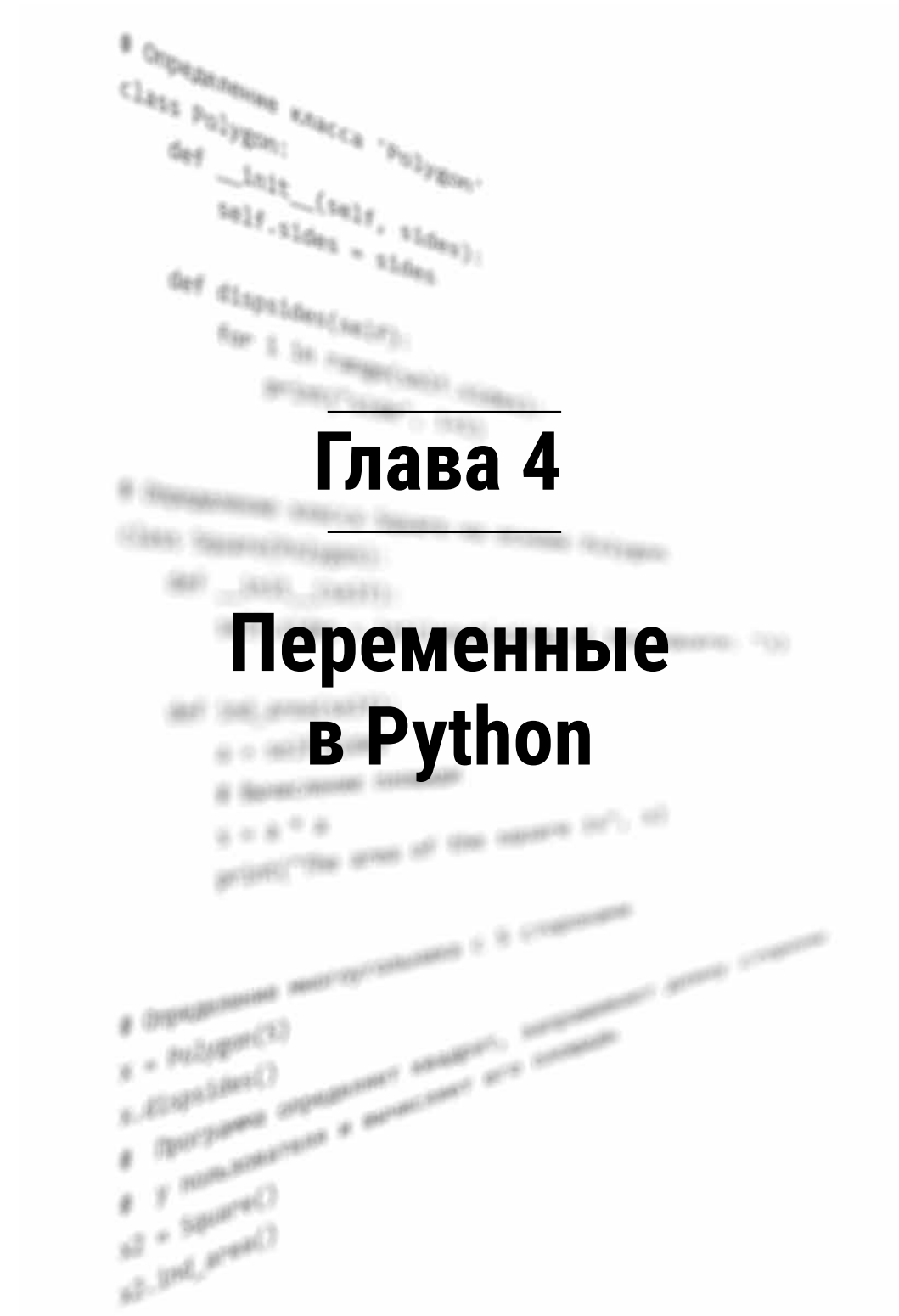
Если разработчик не следит за приоритетами операций, результат может полностью измениться, что, вероятно, приведет к фатальному сбою приложения.

## Правила приоритета операторов в Python

- Если в выражении присутствуют операторы, заключенные в круглые скобки, то интерпретатор начинает с применения этих операторов, а потом переходит к остальным.
- Затем выполняются побитовые операторы.
- Далее по приоритету идут математические операторы умножения и деления. Операторы `*`, `/`, `%` и `//` имеют такой же приоритет.
- На следующем уровне приоритета находятся остальные арифметические операции, такие как сложение и вычитание (`+` и `–`).
- Самым низким приоритетом обладают операторы сравнения и логические операторы.

## Упражнения

- Напишите программу Python, которая получает данные от пользователя. Выполните с полученными данными различные арифметические операции (умножение, деление и т. д.). Также можете вычислить остаток от деления.
- Напишите команду `print()`, которая выводит на экран ваше любимое стихотворение.
- Напишите программу Python для преобразования десятичного числа в шестнадцатеричное.
- Напишите программу Python, которая получает от пользователя три числа и сохраняет их в переменных `x`, `y` и `z`, а затем вычисляет значение выражения `x^2 (2y + 5z)`.



# Глава 4

# Переменные в Python

Чтобы программа Python работала так, как планировал разработчик, ей необходимы основные структурные блоки — переменные и операторы. Они помогают начинающим программистам понять логику сложных программных продуктов.

# Что такое переменные

Эффективная обработка данных — неотъемлемая часть любой программы. Как пользователи, так и программы взаимодействуют через данные. Без данных приложения не будут иметь смысла и ценности для конечного пользователя. В частности, переменные используются для отправки и получения данных по сети.

Концепция переменных изначально появилась в области математики, которая называется алгеброй. Переменные применяются в ней для определения значений. Так что это понятие не является новшеством языка программирования Python. С первых дней существования высокоуровневых языков программирования переменные использовались для хранения данных в конкретной ячейке памяти компьютера. Первые программисты сталкивались с определенными неудобствами при чтении данных по конкретному адресу памяти компьютера, и они позаимствовали концепцию переменных из алгебры, чтобы хранить значения в памяти компьютера и использовать их тогда, когда потребуется.

Возьмем математическое выражение `2x + 3y`.

1. Если `x` = 3 и `y` = 4, то результат выражения равен 18.

2. Если `x` = 2 и `y` = 6, то результат выражения равен 22.

Аналогичным образом вывод программы зависит от значения переменной, которое можно легко изменить. Если значение какой-то переменной не должно изменяться во время выполнения программы, в терминологии программирования такая переменная называется константой.

Чтобы понять, как работают переменные, необходимо сначала понять, как происходит выполнение программ Python. Чтобы объяснение стало более наглядным, я воспользуюсь командой `print`.

**ПРОГРАММНЫЙ КОД:**

```
print("This is a sample analysis.")
```

**ВЫВОД:**

```
This is a sample analysis.
```

В этом примере при выполнении команды `print` результат немедленно выводится на экран, но, кроме того, происходят многие вещи, которые остаются незаметными для пользователя.

## Как работает код

- При первоначальном выполнении программы интерпретатор читает каждую строку и сопоставляет ее с доступными для него библиотеками.

- Интерпретатор выполняет этот процесс сопоставления достаточно продуманно. Он не только определяет, что представляет собой каждый символ программы, но и анализирует информацию о переменных и читает данные, хранящиеся в соответствующей ячейке памяти, для проверки программной логики.

- Если даже после сложного разбора и анализа кода интерпретатору не удается найти определенные методы или переменные, программа выдает ошибки или исключения.

- В приведенном выше примере при разборе команды `print` интерпретатор немедленно узнает, что речь идет об одном из основных методов, определенных в библиотеке Python, и выводит на экран строковый литерал, который был заключен в круглые скобки.

Если вы в полной мере понимаете вышеприведенное объяснение, можно переходить к использованию переменных в Python.

**ПРОГРАММНЫЙ КОД:**

```
program = "This is a sample analysis."
print(program)
```

**ВЫВОД:**

```
This is a sample analysis.
```

## Как работает код

- При запуске программы интерпретатор обычно разбирает все строки кода, написанные программистом.

- Вместо команды `print`, за которой следует блок текста, интерпретатор теперь видит идентификатор — переменную с именем `program`. Интерпретатор просматривает предшествующий код и видит, что переменная уже инициализирована некоторым текстом и хранится в определенной ячейке памяти.

- Интерпретатор выводит переменную на экран, как того требует программист. Для этого он читает данные, хранящиеся в переменной.

Все переменные работают по этой базовой схеме, даже если они используются в сложной программной логике.

Значение переменной можно легко изменить. Этот факт очень важен для программистов Python, потому что все динамические программы изменяют значения своих переменных в соответствии с данными, введенными пользователем, и это может происходить даже непосредственно во время выполнения программы.

**ПРОГРАММНЫЙ КОД:**

```
sample = "This is an example"
print(sample)
sample = "This is a second example"
print(sample)
```

**ВЫВОД:**

```
This is an example
This is a second example
```

Мы знаем, что интерпретатор Python обрабатывает код последовательно, строку за строкой, соответственно первая команда выводит результат с первым значением переменной, а вторая — со вторым.

# Выбор имен переменных

При создании переменных все Python-разработчики должны соблюдать основные правила, сформулированные сообществом Python. Если вы откажетесь от соблюдения этих правил, в программе могут возникнуть ошибки, а приложение может неожиданно завершиться. Кроме того, соблюдение правил при написании программ может упростить чтение кода.

## Правила выбора имен переменных

- Согласно правилам Python, имена переменных могут содержать только цифры, алфавитные символы и символы подчеркивания. Например, строка `sample1` может использоваться как имя переменной, а строка `$sample1` именем переменной быть не может, потому что она начинается с запрещенного символа `$`.

- Имя переменной в языке Python не может начинаться с цифры. Например, строка `sample1` соответствует формату имен переменных, а строка `1sample` именем переменной быть не может.

- В качестве имен переменных запрещено использовать зарезервированные слова, предназначенные для выполнения различных программных процедур в коде Python. В настоящее время существует около 30 таких ключевых слов, например, к ним относится `for`.

- Хотя это не является жестким требованием, для лучшей читаемости рекомендуется отдавать предпочтение присвоению переменным простых имен. Сложные или запутанные имена загромождают код. Хотя практика длинных имен хорошо подходит для таких высокоуровневых языков, как C, C++ и Perl, язык Python не разделяет эту философию.

# Как определить переменную

Всем переменным, определяемым в языке Python, необходимо присвоить исходное значение оператором присваивания (=) перед первым использованием.

**СИНТАКСИС:**

```
имя_переменной = значение_переменной
```

**ПРИМЕР:**

```
# Переменная с целым типом данных
example = 343
# Переменная со строковым типом данных type
example1 = "Russia"
```

В этом примере `example` — имя создаваемой переменной, а `343` — значение, присваиваемое ей при инициализации.

В приведенном выше способе определения переменной ее тип данных не указывается явно — Python достаточно умен, чтобы определять типы данных автоматически.

# Как узнать адрес переменной в памяти

Каждая переменная хранится в конкретной ячейке памяти. Каждый раз, когда в программе упоминается имя переменной, интерпретатор Python читает информацию, хранящуюся в соответствующей ячейке памяти. Если приказать интерпретатору Python заменить значение переменной, он просто запишет новое значение на место старого. Старое значение будет уничтожено (или в отдельных случаях сохранено механизмом сборки мусора для уничтожения в будущем).

Такие языки программирования, как C, обычно используют указатели для получения информации о местонахождении переменной в памяти. Однако в Python указатели не поддерживаются, так как часто их сложно реализовать. Кроме того, требуется искусная компиляция, с чем интерпретатор не всегда справляется.

Вместо этого для получения адреса переменной в памяти разработчик Python может воспользоваться встроенной функцией `id()`.

**ПРОГРАММНЫЙ КОД:**

```
# Сначала создаем переменную
sample = 64
# Затем вызываем встроенную функцию id()
print(id(sample))
```

**ВЫВОД:**

```
1x37372829x
```

Здесь `1x37372829x` — адрес переменной в памяти в шестнадцатеричном формате.

Теперь попробуем заменить значение переменной и проверить, изменился ли результат вызова `id()`.

**ПРОГРАММНЫЙ КОД:**

```
sample = 64
print(id(sample))
# Значение переменной заменяется новым
sample = 78
# Снова выводим адрес переменной в памяти
print(id(sample))
```

**ВЫВОД:**

```
1x37372829x
```

Нетрудно заметить, что адрес переменной в памяти остался прежним, а проверка с помощью `print` показывает, что значение переменной действительно изменилось.

**ПРОГРАММНЫЙ КОД:**

```
sample = 64
print(id(sample))
sample = 78
print(sample)
```

**ВЫВОД:**

```
78
```

# Локальные и глобальные переменные в Python

В зависимости от логики, которую вы реализуете в своей программе, переменные могут быть локальными или глобальными. Теоретически локальные переменные могут применяться только в конкретных функциях или классах, в которых они были созданы. Глобальные переменные, напротив, можно использовать в любой части программы без каких-либо проблем. При обращении к локальной переменной за пределами функции, в которой она определяется, интерпретатор Python обычно выдает сообщение об ошибке.

**ПРОГРАММНЫЙ КОД:**

```
# Пример функции с локальной переменной
def sample():
    example = "This is a trail"
    print(example)


sample()
```

**ВЫВОД:**

```
This is a trail
```

В этом примере переменная определяется как локальная внутри функции. А значит, при любых попытках использовать ее в любой другой функции интерпретатор выдаст сообщение об ошибке.

**ПРОГРАММНЫЙ КОД:**

```
# Пример функции с локальной переменной
def sample():
    example = "This is a trail"
    print(example)


def second_sample():
    # Обращение к переменной из другой функции
    print(example)


second_sample()
```

**ВЫВОД:**

```
This is a trail
NameError: name 'example' is not defined
```

К глобальным же переменным можно обращаться из любой части программы.

**ПРОГРАММНЫЙ КОД:**

```
# Создание глобальной переменной
example = "This is a trail"


def sample():
    print(example)


def second_sample():
    # Обращение из другой функции
    print(example)


sample()
second_sample()
```

**ВЫВОД:**

```
This is a trail
This is a trail
```

Так как функции могут обращаться к глобальным переменным, в обоих случаях `print` выводит сообщение на экран.

> **ПРИМЕЧАНИЕ**
>
> Выбор между локальными или глобальными переменными зависит исключительно от вас. Многие программисты предпочитают локальные переменные, потому что с ними приложения выполняются быстрее. Если же вы хотите избежать хлопот с управлением памятью, можно использовать глобальные переменные.

# Глава 5

# Типы данных Python

Python-разработчики обычно используют разные типы данных в зависимости от задачи. Программист должен понимать, насколько важна эта тема для процесса разработки.

# Что такое типы данных

Типы данных представляют собой заранее определенные интервалы значений, используемые программистами при создании переменных. Важно помнить, что Python не является языком со статической типизацией, поэтому разработчик не обязан явно определять типы данных при создании переменных. Все языки со статической типизацией — такие как C или C++ — обычно требуют от программистов определять типы данных, хранящихся в переменных.

Хотя в Python определять тип данных не обязательно, все же для построения сложных программ, активно взаимодействующих с пользователем, необходимо разбираться в них.

Вот пример языка *со статической типизацией* и того, как в нем определяются переменные.

**ПРОГРАММНЫЙ КОД:**

```
int age = 12:
```

Здесь `int` — тип данных, используемый для определения переменной, `age` — имя создаваемой переменной, а `12` — значение, которое должно быть сохранено в переменной `age`.

В Python же переменные определяются *без явного указания типа*.

**ПРОГРАММНЫЙ КОД:**

```
age = 12
```

Здесь программист указывает только имя переменной и ее значение, но не определяет тип данных, потому что интерпретатор Python уже понимает, что передаваемое значение является целым числом.

## Составные элементы кода

Прежде чем рассматривать типы данных, поддерживаемые Python, немного поговорим об основных компонентах, которые используются разработчиками при написании кода.

Начнем с простых выражений и команд. При их создании в языках программирования используются три основных компонента.

### Идентификаторы

Идентификатор — это имя объекта, например переменной, функции, класса, модуля и т. д.

**ПРИМЕР:**

```
x = 34
```

В этом примере `x` — идентификатор переменной, в которой хранятся данные.

## Литералы

Это значения, которые присваиваются любым фрагментам данных, создаваемым в программах.

**ПРИМЕР:**

```
x = 34
```

В этом примере `34` — литерал, который сохраняется в созданном фрагменте данных.

## Операторы

Операторы используются для реализации математических операций при написании программной логики.

**ПРИМЕР:**

```
x = 34
```

В этом примере `=` — оператор присваивания. При создании кода Python также часто применяются и другие арифметические операторы, например `+`, `-`, `*` и `/`.

Рассмотрим некоторые популярные типы данных, используемые программистами Python в приложениях.

# Строки

Строки — тип данных, который обычно используется для представления фрагментов текста. Программист, который хочет применять строковые данные для хранения текста в программе, может использовать одинарные или двойные кавычки. Каждый раз, когда в программе создается строковый тип данных, на самом деле создается объект `str`, содержащий последовательность символов.

Люди обычно общаются с помощью текста, поэтому строки являются важнейшим типом данных, который должен знать разработчик для создания полезных программ. Важно понимать, что компьютеры всегда работают с двоичными данными, следовательно, для них строки представляют собой комбинацию нулей и единиц. Каждый символ преобразуется с помощью кодировки ASCII или Юникод, и программист должен хорошо понимать их.

В Python 3 появился усовершенствованный механизм кодирования текста для таких иностранных языков, как китайский, японский и корейский, вследствие чего расширяются возможности использования строк.

Как же строки представляются в программе?

**ПРИМЕР:**

```
x = 'This is an example'
print(x)
```

**ВЫВОД:**

```
This is an example
```

Все символы, заключенные между одинарными или двойными кавычками, относятся к строковому типу данных. В примере выше строковые данные сохраняются в переменной `x`. Область памяти и размер переменной со строковым типом данных обычно определяются количеством битов, необходимых для хранения переменной. Количество символов в строковом типе данных прямо пропорционально количеству битов, например, в приведенном выше примере строка `'This is an example'` содержит 18 символов (включая пробелы).

Поскольку Python-разработчики могут объявлять строки с помощью разных видов кавычек, для сохранения целостности при работе над реальными проектами рекомендуется использовать во всем коде один способ, который вам кажется наиболее удобным.

**ПРОГРАММНЫЙ КОД:**

```
# Определение строк в двойных кавычках
a = "This is an example"
print(a)
# Определение строк в утроенных одинарных кавычках
b = '''This is an example'''
print(b)
# Определение строк в утроенных двойных кавычках
c = """This is an example
but with more than one line"""
print(c)
```

**ВЫВОД:**

```
This is an example
This is an example
This is an example
but with more than one line
```

В этой программе продемонстрированы три разных способа определения строк. Между кавычками также допускается использование специальных символов, знаков и разрывов строк. Python поддерживает экранированные последовательности, которые используются в большинстве других языков программирования. Например, `\n` – экранированная последовательность, применяемая программистами для создания новых строк.

## Как обращаться к символам строки

Так как строки являются наиболее часто используемым типом данных в Python, основная библиотека предоставляет несколько встроенных функций для удобной работы с ними.

Чтобы обратиться к отдельным символам строки, необходимо знать их индексы. Нумерация индексов обычно начинается с 0, а не с 1. Также можно использовать отрицательные индексы и срезы для обращения к отдельным частям строки.

**ПРИМЕР:**

```
# Обращение к частям строки
first = 'Programming'
# Выводит всю строку
print('Example used = ', first)
# Выводит первый символ
print('first character = ', first[0])
# Выводит последний символ с помощью отрицательного
# индекса
print('last character = ', first[-1])
# Выводит последний символ с помощью положительного
# индекса
print('last character = ', first[10])
# Выводит срез от нулевого до третьего индекса
print('Sliced character = ', first[0:3])
```

**ВЫВОД:**

```
Example used = Programming
first character = P
last character = g
last character = g
Sliced character = Pro
```

Так как строковый тип данных является неизменяемым, заменить символы в строковом литерале невозможно. Если вы попытаетесь заменить отдельные символы, будет выдана ошибка `TypeError`.

**ПРОГРАММНЫЙ КОД:**

```
first = 'programming'
first[1] = 'c'
print(first)
```

**ВЫВОД:**

```
TypeError: 'str' object does not support item
➥  assignment
```

## Форматирование строк

Python предоставляет простые средства форматирования строк с использованием символа `%` (этот же символ применяется при вычислении остатка от деления). В этом контексте `%` также называется оператором форматирования строк.

**ПРОГРАММНЫЙ КОД:**

```
print("Italy won FIFA cup %d times" % 4)
```

**ВЫВОД:**

```
Italy won FIFA cup 4 times
```

Для форматирования строк с помощью целых чисел используется обозначение `%d`, а для добавления строковых символов — `%s`.

## Операции со строками

Строки являются самым популярным типом данных, поэтому в стандартную библиотеку Python включена поддержка операций со строками. С их помощью вы сможете легко извлечь нужные данные из большого объема информации. Строковые операции особенно важны для специалистов по анализу и обработке данных.

## Конкатенация

Конкатенацией называется соединение двух фрагментов строк. Для соединения двух строк можно воспользоваться арифметическим оператором +. Если вы хотите, чтобы команда лучше читалась, разделите строки пробелами.

**ПРОГРАММНЫЙ КОД:**

```
example = 'This is ' + 'a great example'
print(example)
```

**ВЫВОД:**

```
This is a great example
```

Помните, что при конкатенации объединяемые строки не разделяются пробелами. Пробелы необходимо добавить самостоятельно, как показано в следующем примере.

**ПРОГРАММНЫЙ КОД:**

```
example = 'This is' + ' ' + 'a great example'
print(example)
```

**ВЫВОД:**

```
This is a great example
```

## Умножение строк

Операция умножения строк многократно повторяет строковое значение. Для ее выполнения используется оператор *.

**ПРОГРАММНЫЙ КОД:**

```
example = 'Great ' * 4
print(example)
```

**ВЫВОД:**

```
Great Great Great Great
```

### Присоединение

Эта операция присоединяет любую строку в конец другой строки; для ее выполнения используется арифметический оператор `+=`. Помните, что присоединяемая строка добавляется именно в конец другой строки.

**ПРОГРАММНЫЙ КОД:**

```
example = "France is a beautiful country "
example += "you need to visit at least once"
print(example)
```

**ВЫВОД:**

```
France is a beautiful country you need to visit at
➥  least once
```

### Определение длины строки

Кроме строковых операций, вы также можете пользоваться встроенными функциями и методами из основной библиотеки. Например, функция `len()` позволяет узнать количество символов в строке.

> **ПРИМЕЧАНИЕ**
>
> Пробелы также учитываются при подсчете символов в строке.

**ПРОГРАММНЫЙ КОД:**

```
example = 'Today it will rain'
print(len(example))
```

**ВЫВОД:**

```
18
```

## Поиск в строке

При работе со строками нередко встречается задача поиска фрагмента текста. Для решения этой задачи можно воспользоваться встроенной функцией `find()`. Она возвращает индекс позиции первого вхождения искомой подстроки.

> **ПРИМЕЧАНИЕ**
>
> При использовании функции `find()` интерпретатор Python возвращает только положительные индексы.

**ПРОГРАММНЫЙ КОД:**

```
example = "This is great"
sample = example.find('gr')
print(sample)
```

**ВЫВОД:**

```
8
```

Если подстрока не найдена, интерпретатор возвращает значение `-1`.

**ПРОГРАММНЫЙ КОД:**

```
example = "This is great"
sample = example.find('f')
print(sample)
```

**ВЫВОД:**

```
-1
```

## Преобразование регистра

Методы `lower()` и `upper()` используются для приведения символов строки к нижнему или верхнему регистру.

**ПРОГРАММНЫЙ КОД:**

```
example = "China is the most populous country"
sample = example.lower()
print(sample)
```

**ВЫВОД:**

```
china is the most populous country
```

**ПРОГРАММНЫЙ КОД:**

```
example = "China is the most populous country"
sample = example.upper()
print(sample)
```

**ВЫВОД:**

```
CHINA IS THE MOST POPULOUS COUNTRY
```

### Метод title()

Метод `title()` используется для преобразования строки, в результате которого каждое слово начинается с буквы в верхнем регистре.

**ПРОГРАММНЫЙ КОД:**

```
example = "China is the most populous country"
sample = example.title()
print(sample)
```

**ВЫВОД:**

```
China Is The Most Populous Country
```

# Целые числа

Числовые значения в коде Python используются для выполнения арифметических операций или получения подробной информации о статистической величине. Когда интерпретатор Python встречает значение целочисленного типа, он немедленно создает объект `int` с указанным значением. Ни одно значение объекта `int` не может заменяться другим значением, поскольку этот тип является неизменяемым.

Разработчики используют тип данных `int` для реализации многих нетривиальных возможностей в своих программах. Например, плотность пикселей в графических или видеофайлах обычно представляется целыми числами.

> **ПРИМЕЧАНИЕ**
>
> Разработчику также следует знать об унарных операторах (+, -), которые используются для представления положительных и отрицательных чисел соответственно. Для положительных целых чисел указывать унарный оператор не обязательно, но для отрицательных целых чисел это необходимо.

**ПРОГРАММНЫЙ КОД:**

```
x = 25
y = -45
print(x)
print(y)
```

**ВЫВОД:**

```
25
-45
```

# Числа с плавающей точкой

Не все числовые значения являются целыми. Иногда приходится иметь дело с данными, имеющими дробную часть. Чтобы разработчики могли оперировать такими данными, в Python поддерживаются числа с плавающей точкой. Они позволяют работать с дробными значениями с точностью до семи знаков.

**ПРОГРАММНЫЙ КОД:**

```
x = 4.2324324
y = 67.32323
print(x)
print(y)
```

**ВЫВОД:**

```
4.2324324
67.32323
```

Числа с плавающей точкой также могут использоваться для представления данных в шестнадцатеричной системе.

**ПРОГРАММНЫЙ КОД:**

```
a = float.hex(23.232)
print(a)
```

**ВЫВОД:**

```
0x1.73b645a1cac08p+4
```

Многие программисты Python также используют типы данных с плавающей точкой для представления комплексных чисел и чисел в экспоненциальной записи.

# Логический тип данных

К логическому типу данных относятся специальные типы данных, которые обычно используются для представления результатов `True` или `False`, например при сравнении двух значений.

**ПРОГРАММНЫЙ КОД:**

```
a = 32
b = 64
print (a > b)
```

**ВЫВОД:**

```
False
```

В этом примере результат равен `False`, потому что значение `a` не больше значения `b`. Логические типы данных удобны при выполнении логических операций.

# Глава 6

# Сложные структуры данных в Python

Python-разработчикам часто приходится работать с большими объемами данных, для которых не всегда удобно использовать переменные. В частности, специалистам по анализу нередко необходимо обрабатывать огромные объемы информации, и количество таких динамических данных может выходить из-под контроля. Списки из стандартной библиотеки Python будут очень полезными для программистов, работающих со сложными проектами с высокими требованиями к данным. Эта структура данных напоминает массивы из языка C.

Хорошее знание поддерживаемых в Python структур данных, а также методов добавления и изменения элементов в этих структурах относится к числу важнейших навыков для программиста.

# Списки

Списки — структура данных Python, способная хранить элементы разного типа в определенном порядке. У списков, как и у переменных, есть имя, и их тоже можно передавать в функции, однако у списков есть и уникальные свойства и возможности. Например, их можно изменять, извлекать из них отдельные элементы или обрабатывать при помощи методов из стандартной библиотеки Python.

Списки обычно представляются в Python в следующем формате:

```
[32, 33, 34]
```

В этом фрагменте кода `32`, `33` и `34` — элементы списка. Важно понимать, что здесь элементы списка относятся к целочисленному типу данных, который не определяется явно, потому что интерпретатор Python может делать это автоматически.

Наблюдательный читатель заметит, что списки заключаются в квадратные скобки, а элементы разделяются запятыми. Если элемент списка относится к строковому типу данных, он записывается в кавычках.

**ПРИМЕР:**

```
['Nevada', 'Ohio', 'Colorado']
```

В данном контексте `Nevada`, `Ohio` и `Colorado` — элементы списка.

Списки могут присваиваться переменным, как в следующем примере:

```
sample = ['Nevada', 'Ohio', 'Colorado']
```

Списки выводятся на экран точно так же, как и все остальные типы данных.

**ПРОГРАММНЫЙ КОД:**

```
print(sample)
```

**ВЫВОД:**

```
['Nevada', 'Ohio', 'Colorado']
```

## Пустой список

Список Python, не содержащий ни одного элемента, называется пустым. Обычно пустые списки представляются в виде `[]`.

**ПРОГРАММНЫЙ КОД:**

```
example = []
```

## Индексы в списках

Python поддерживает простые и удобные средства для обработки и замены элементов списка. Индексы обычно начинаются с 0; они используются при выполнении многих операций (например, сегментирования и поиска).

Возьмем список, который был показан в одном из предыдущих примеров:

```
['Ohio', 'Nevada', 'Colorado']
```

Все элементы этого списка можно вывести с указанием их индексов.

**ПРОГРАММНЫЙ КОД:**

```
example = ['Ohio', 'Nevada', 'Colorado']
print(example[0])
print(example[1])
print(example[2])
```

**ВЫВОД:**

```
Ohio
Nevada
Colorado
```

Получая индекс 0, интерпретатор Python выводит первый элемент списка. С увеличением индекса увеличивается позиция элемента в списке.

Также при выводе элемент списка можно объединить со строковым литералом.

**ПРОГРАММНЫЙ КОД:**

```
example = ['Ohio', 'Nevada', 'Colorado']
print(example[2] + ' is a great city')
```

**ВЫВОД:**

```
Colorado is a great city
```

Если при обращении к списку указывается индекс, превышающий количество элементов, интерпретатор Python выдает сообщение об ошибке.

**ПРОГРАММНЫЙ КОД:**

```
example = ['Ohio', 'Nevada', 'Colorado']
print(example[3])
```

**ВЫВОД:**

```
Index error: list index out of range
```

**ПРИМЕЧАНИЕ**

Следует помнить, что числа с плавающей точкой не могут использоваться в качестве индексов.

**ПРОГРАММНЫЙ КОД:**

```
example = ['Ohio', 'Nevada', 'Colorado']
print(example[3.2])
```

**ВЫВОД:**

```
TypeError: list indices must be integers or slices,
➥  not float
```

Элементами списков также могут быть другие списки. Список, хранящийся внутри другого списка, называется вложенным.

**ПРОГРАММНЫЙ КОД:**

```
x = [[1, 223], [2, 45, 63, 22]]
print(x)
```

**ВЫВОД:**

```
[[1, 223], [2, 45, 63, 22]]
```

Для обращения к элементу вложенного списка используется запись вида `list[][]`.

**ПРОГРАММНЫЙ КОД:**

```
x = [[1, 223], [245, 63, 22]]
print(x[1][2])
```

**ВЫВОД:**

```
22
```

Третий элемент второго списка равен 22, что и отражено в выводе.

К элементам списка также можно обращаться с указанием отрицательного индекса. Обычно значение `-1` соответствует индексу последнего элемента, `-2` — предпоследнего и т. д.

**ПРОГРАММНЫЙ КОД:**

```
example = ['Ohio', 'Nevada', 'Colorado']
print(example[-1])
```

**ВЫВОД:**

```
Colorado
```

До настоящего момента рассматривалось объявление списков и его вывод элементов на экран. В следующем разделе речь пойдет об операциях, выполняемых со списками как структурами данных.

## Срезы

Срезы избавляют программиста от необходимости работать с элементами списка, которые его не интересуют. Выделяя сегмент, можно сосредоточиться на той части списка, которая важна для логики программы.

**СИНТАКСИС:**

```
имя_списка[начальный индекс: конечный индекс]
```

Индекс начального элемента обычно отделяется двоеточием от индекса конечного элемента списка.

**ПРОГРАММНЫЙ КОД:**

```
sample = [12, 32, 21, 24, 65]
print(sample[0:2])
```

**ВЫВОД:**

```
[12, 32]
```

При определении среза не обязательно указывать начало или конец списка. Если это не указано в явном виде, интерпретатор автоматически предполагает, что имеется в виду первый или последний элемент списка.

**ПРИМЕР:**

```
sample = [12, 32, 21, 24, 65]
print(sample[:3])
```

**ВЫВОД:**

```
[12, 32, 21]
```

Так как в этом примере индекс перед двоеточием не указан, интерпретатор делает вывод, что срез начинается с первого элемента.

**ПРИМЕР:**

```
sample = [12, 32, 21, 24, 65]
print(sample[2:])
```

**ВЫВОД:**

```
[21, 24, 65]
```

В этом примере интерпретатор предполагает, что индекс после двоеточия должен соответствовать концу списка.

Если не указаны оба значения, на выходе будет получен полный список, как в следующем примере.

**ПРОГРАММНЫЙ КОД:**

```
sample = [12, 32, 21, 24, 65]
sample[:]
```

**ВЫВОД:**

```
[12, 32, 21, 24, 65]
```

## Получение длины списка

Чтобы быстро определить длину списка, используйте встроенную функцию `len()`.

**ПРОГРАММНЫЙ КОД:**

```
sample = [12, 32, 21, 24, 65]
print(len(sample))
```

**ВЫВОД:**

```
5
```

## Изменение значений элементов списка

Как показано ниже, значения в списке можно легко изменить при помощи оператора присваивания.

**ПРОГРАММНЫЙ КОД:**

```
sample = [12, 32, 21, 24, 65]
sample[2] = 34
print(sample)
```

**ВЫВОД:**

```
[12, 32, 34, 24, 65]
```

Также значение элемента в списке можно заменить значением другого, уже существующего элемента, как в следующем примере.

**ПРОГРАММНЫЙ КОД:**

```
sample = [12, 32, 21, 24, 65]
sample[2] = sample[1]
print(sample)
```

**ВЫВОД:**

```
[12, 32, 32, 24, 65]
```

## Конкатенация списков

Два списка легко объединяются арифметическим оператором +.

**ПРОГРАММНЫЙ КОД:**

```
sample = [12, 32, 21, 24, 65]
example = [11, 22, 33]
print(sample + example)
```

**ВЫВОД:**

```
[12, 32, 21, 24, 65, 11, 22, 33]
```

## Дублирование списков

Оператор * позволяет продублировать элементы списков.

**ПРОГРАММНЫЙ КОД:**

```
print([2, 4, 6] * 3)
```

**ВЫВОД:**

```
[2, 4, 6, 2, 4, 6, 2, 4, 6]
```

## Удаление элементов

Из списка также можно удалить отдельные элементы командой `del`.

**ПРОГРАММНЫЙ КОД:**

```
sample = [2, 3, 4, 6, 8]
del sample[2]
print(sample)
```

**ВЫВОД:**

```
[2, 3, 6, 8]
```

## Операторы in и not in

Python предоставляет простой способ проверки присутствия значений в списке. Для этого используются логические операторы `in` и `not in`. Их результатом является логическое значение `True` или `False`.

**ПРОГРАММНЫЙ КОД:**

```
print('Football' in ['Cricket', 'Football',
➥  'Hockey'])
```

**ВЫВОД:**

```
True
```

## Метод index()

Метод `index()` позволяет легко найти индекс элемента в списке.

**ПРОГРАММНЫЙ КОД:**

```
x = [32, 23, 12]
print(x.index(23))
```

**ВЫВОД:**

```
1
```

Если указанный элемент отсутствует в списке, будет выдана ошибка `ValueError`.

**ПРОГРАММНЫЙ КОД:**

```
x = [32, 23, 12]
print(x.index(49))
```

**ВЫВОД:**

```
ValueError: 49 is not in list
```

## Метод insert()

С помощью метода `insert()` можно добавить новый элемент на любую позицию списка.

**СИНТАКСИС:**

```
имя_списка.insert(индекс, элемент)
```

**ПРОГРАММНЫЙ КОД:**

```
x = [32, 23, 12]
x.insert(2, 11)
print(x)
```

**ВЫВОД:**

```
[32, 23, 11, 12]
```

В этом примере новый элемент добавляется на третью позицию (индекс 2), а прежний третий элемент сдвигается на четвертую позицию.

## Метод sort()

При помощи метода `sort()` Python-разработчики могут легко выстроить все элементы списка по возрастанию или по убыванию.

**ПРОГРАММНЫЙ КОД:**

```
x = [23, 12, 11, 45]
x.sort()
print(x)
```

**ВЫВОД:**

```
[11, 12, 23, 45]
```

Если в списке содержатся строки, то список будет отсортирован в алфавитном порядке.

**ПРОГРАММНЫЙ КОД:**

```
x = ['USA', 'China', 'Russia', 'UK']
x.sort()
print(x)
```

**ВЫВОД:**

```
['China', 'Russia', 'UK', 'USA']
```

Для сортировки элементов списка по убыванию в метод добавляется аргумент `reverse = True`.

# Кортежи

Списки — популярная структура данных, которая часто используется программистами Python. Тем не менее их реализации присущи некоторые проблемы. Так как все списки, создаваемые в Python, являются изменяемыми объектами, повышается риск случайной замены элементов, удаления или иных операций с ними.

Разработчику иногда бывает удобнее создать неизменяемую структуру данных. В такой ситуации стоит обратиться к кортежам. Значения элементов кортежа невозможно изменить. При попытке изменить содержимое кортежа выдается ошибка `TypeError`.

**ПРОГРАММНЫЙ КОД:**

```
# Создание кортежа в Python
example = ('Earth', 'Venus', 'Mars')
print(example)
```

**ВЫВОД:**

```
('Earth', 'Venus', 'Mars')
```

В этом примере мы просто создаем кортеж и используем функцию `print` для вывода его содержимого на экран.

> **ПРИМЕЧАНИЕ**
>
> Кортежи, в отличие от списков, записываются не в квадратных, а в круглых скобках.

Проведем эксперимент, чтобы вы поняли, как работают кортежи. Попробуем изменить элемент в приведенном примере, вывести его содержимое и посмотреть, что при этом произойдет.

**ПРОГРАММНЫЙ КОД:**

```
# Создание кортежа в Python
example = ('Earth', 'Venus', 'Mars')
print(example)
example[2] = 'Jupiter'
# Вывод содержимого кортежа после замены элемента
print(example)
```

**ВЫВОД:**

```
('Earth', 'Venus', 'Mars')
TypeError: 'tuple' object does not support item
➥  assignment
```

Как только вы попытаетесь изменить элемент кортежа, интерпретатор выдаст ошибку. Это доказывает, что все элементы кортежа неизменяемы, вследствие чего замена, удаление или добавление новых элементов становятся невозможными.

## Конкатенация кортежей

С кортежами также можно выполнять различные операции, сходные с операциями со списками, которые рассматривались ранее. Например, как и в случае со списками, с кортежами можно выполнять операции сложения или умножения.

**ПРОГРАММНЫЙ КОД:**

```
sample1 = (45, 34, 23)
sample2 = (32, 12, 11)
# Сложение двух кортежей
print(sample1 + sample2)
```

**ВЫВОД:**

```
(45, 34, 23, 32, 12, 11)
```

В этом примере два кортежа объединяются оператором сложения. Аналогичным образом можно воспользоваться оператором умножения для быстрого дублирования элементов в кортеже.

Также в кортеже могут храниться другие кортежи. Обычно внутренние кортежи называются вложенными.

**ПРОГРАММНЫЙ КОД:**

```
a = (23, 32, 12)
b = ('Tokyo', 'Paris', 'Washington')
c = (a, b)
print(c)
```

**ВЫВОД:**

```
((23, 32, 12), ('Tokyo', 'Paris', 'Washington'))
```

В этом примере два кортежа вложены в один общий кортеж.

## Дублирование

Программисты также могут дублировать содержимое кортежа при помощи оператора *.

**ПРОГРАММНЫЙ КОД:**

```
a = (2, 3, 4) * 3
print(a)
```

**ВЫВОД:**

```
(2, 3, 4, 2, 3, 4, 2, 3, 4)
```

Как было сказано ранее, содержимое кортежей невозможно изменить, так как кортежи проектировались как неизменяемые. Посмотрим, что произойдет, если попытаться заменить одно значение другим.

**ПРОГРАММНЫЙ КОД:**

```
x = (32,64,28)
x[2] = 12
print(x)
```

**ВЫВОД:**

```
TypeError: 'tuple' object does not support item
➥  assignment
```

## Сегментирование кортежей

Программист может легко извлечь часть кортежа операцией сегментирования, которая выделяет часть кортежа по граничным индексам.

**ПРОГРАММНЫЙ КОД:**

```
x = (12, 13, 14, 15, 16)
print(x[1:3])
```

**ВЫВОД:**

```
(13, 14)
```

## Как удалить кортеж

Невозможно удалить конкретный элемент, присутствующий в кортеже, но сам кортеж можно полностью удалить следующей командой.

**ПРОГРАММНЫЙ КОД:**

```
x = (12, 13, 14, 15, 16)
del x
print(x)
```

**ВЫВОД:**

```
NameError: name 'x' is not defined
```

# Словари

Словари — специальные структуры данных, которые существуют в Python для хранения значений в парах (в отличие от одиночных значений, используемых списками и кортежами). В словарях хранятся пары «ключ — значение», что улучшает оптимизацию обращений к данным и повышает эффективность. Содержимое словарей записывается в фигурных скобках, чтобы они отличались от списков и кортежей.

## Как создать словарь

Как говорилось ранее, словари определяются как наборы пар «ключ — значение», разделенные запятыми. Все последовательно перечисляемые элементы должны разделяться запятыми.

**СИНТАКСИС:**

```
словарь = {ключ: значение, ключ: значение, ...}
```

Разработчик может включить в словарь любое количество таких пар.

**ПРИМЕР:**

```
capitals = {'USA': 'Washington', 'Russia':
➥  'Moscow', 'China': 'Beijing'}
print(capitals)
```

**ВЫВОД:**

```
{'USA': 'Washington', 'Russia': 'Moscow',
➥  'China': 'Beijing'}
```

Также поддерживается возможность создания вложенных словарей, то есть словарей, которые являются элементами других словарей:

```
capitals = {'USA': 'Washington', 'Russia':
'Moscow', 'China': 'Beijing',
'Australia': {'Australia': 'Canberra',
➥  'New_Zealand': 'Wellington'}}
```

В этом примере значение последней пары представляет собой словарь, состоящий из двух пар «ключ — значение».

## Упражнения

- Напишите программу для игры в «перепутанные слова»[1]. Программа должна создавать несколько списков, взаимодействующих друг с другом.

- Напишите программу, которая переставляет в обратном порядке все элементы списка и подсчитывает длину списка.

- Напишите программу для упорядочения содержащихся в словаре пар «ключ — значение» по возрастанию или по убыванию.

[1] Игра, в которой ученикам дается список слов с переставленными буквами, которые необходимо восстановить.

- Напишите программу, которая создает словарь из N пар популярных синонимов (можно начать с N = 10.)

- Напишите программу, которая переставляет элементы словаря в обратном порядке и заменяет в их элементах значения цветов в формате RGB («красный/зеленый/синий») цветами в формате «синий/зеленый/оранжевый».

# Глава 7

# Условные конструкции и циклы

В ходе работы с мобильным или веб-приложением пользователь принимает разнообразные решения. Поэтому в любой компьютерной программе должна быть возможность реализовать эти решения. Например, приложение должно быть достаточно умным для того, чтобы сформировать интерфейс в соответствии с выбором пользователя. Подобный динамический подход имеет много общего с тем, как мыслят люди.

Чтобы научиться программировать динамическое поведение ваших приложений, необходимо понимать, как работают условные конструкции и циклы — высокоуровневые программные структуры.

Условные конструкции и циклы также помогают ускорить выполнение программ. Каждый Python-разработчик должен уметь пользоваться ими, так как эти конструкции необходимы для более сложных тем, которые будут рассматриваться далее (таких как функции и модули).

# Операторы сравнения

Чтобы понимать условные конструкции и циклы и уметь пользоваться ими на практике, необходимо знать разные операторы сравнения, которые поддерживаются языком Python.

Операторы сравнения, как следует из названия, сравнивают два операнда и выдают результат в форме логического значения (`True` или `False`).

> **ПРИМЕЧАНИЕ**
>
> `True` и `False` — специальные логические значения, поддерживаемые языком Python для принятия решений в программах. Эти логические значения базируются на принципах работы логических элементов, содержащихся в микропроцессорах.

## Оператор «меньше» (<)

Оператор «меньше» проверяет, меньше ли значение левого операнда, чем значение правого операнда.

**ПРОГРАММНЫЙ КОД:**

```
x = 64 < 83
print(x)
```

**ВЫВОД:**

```
True
```

**ПРОГРАММНЫЙ КОД:**

```
x = 73 < 45
print(x)
```

**ВЫВОД:**

```
False
```

Если присмотреться к двум приведенным примерам, в первом случае выводится `True`, потому что 64 меньше 83, а во втором примере выводится `False` — ведь 73 точно не меньше 45.

Аналогичным образом оператор «меньше» может использоваться для сравнения чисел с плавающей точкой.

**ПРОГРАММНЫЙ КОД:**

```
x = 8.3 < 43
print(x)
```

**ВЫВОД:**

```
True
```

Оператор «меньше» также может использоваться для сравнения строк в формате ASCII.

**ПРОГРАММНЫЙ КОД:**

```
x = 'Sample' < 'sample'
print(x)
```

**ВЫВОД:**

```
True
```

В этом примере выводится логическое значение `True`, потому что ASCII-код буквы нижнего регистра обычно больше ASCII-кодов букв верхнего регистра.

## Упражнение

Вычислите сумму ASCII-кодов букв слова `sample` из этого примера. Код символа можно определить при помощи функции `ord()`.

Операторы сравнения также могут использоваться с другими структурами данных, например кортежами. Однако перед сравнением следует убедиться в том, что все значения в кортеже имеют одинаковые типы данных.

**ПРОГРАММНЫЙ КОД:**

```
print((22, 25, 34) < (32, 34, 46, 76))
```

**ВЫВОД:**

```
True
```

Если кортежи содержат значения разных типов данных, в терминал выводится ошибка.

**ПРОГРАММНЫЙ КОД:**

```
print((1, 2, 3) < ('one', 34, 56))
```

**ВЫВОД:**

```
TypeError: '<' not supported between instances of
➥  'int' and 'str'
```

## Оператор «больше» (>)

Оператор «больше» проверяет, больше ли значение левого операнда, чем значение правого операнда.

**ПРОГРАММНЫЙ КОД:**

```
print(48 > 64)
```

**ВЫВОД:**

```
False
```

**ПРОГРАММНЫЙ КОД:**

```
print(33 > 21)
```

**ВЫВОД:**

```
True
```

В первом примере выводится логическое значение `False`, потому что значение левого операнда (48) не больше правого (64).

Во втором примере же выводится логическое значение `True`, потому что значение правого операнда (33) больше значения левого операнда (21).

Аналогичным образом оператор «больше» может использоваться для сравнения чисел с плавающей точкой и других типов данных (например, кортежей).

## Оператор «равно» (==)

Оператор «равно» проверяет, равны ли значения двух операндов — левого и правого. Если значения равны, возвращается логическое значение `True`, а если нет — значение `False`.

**ПРОГРАММНЫЙ КОД:**

```
print(64 == 64)
```

**ВЫВОД:**

```
True
```

**ПРОГРАММНЫЙ КОД:**

```
print(43 == 42)
```

**ВЫВОД:**

```
False
```

# Операторы управления

Теперь вы знаете, как работают операторы сравнения, и мы можем перейти к операторам управления, знать которые разработчикам просто необходимо. Программисты используют их даже в относительно простом коде.

## Последовательная структура

При последовательной структуре программного кода все действия вашей программы обычно выполняются линейно. Многие программы используют последовательную структуру, чтобы избежать усложнения кода. Тем не менее создание последовательного кода требует значительного мастерства от программиста, потому что линейный подход часто оказывается непросто реализовать.

**ПРИМЕР:**

```
a = "34"
print(a + " is my favorite number")
```

**ВЫВОД:**

```
34 is my favorite number
```

В этом примере интерпретатор Python выполняет код последовательно, строку за строкой и в результате выводит на экран текст.

## Условная конструкция

Условие — известная программная структура, при которой, в зависимости от результата логической проверки, определенная часть программы выполняется, а другая часть пропускается.

В условной конструкции выполняются только части команд, что помогает интерпретатору Python экономить время, так как ему не нужно разбирать весь написанный код.

Чаще всего программисты Python используют условные структуры `if` и `if-else`.

## Циклы

Если некоторая команда или блок логики в программе должны выполняться снова и снова на основании логических проверок, используйте циклы. Интерпретатор Python позволяет многократно выполнять действие, пока выполняется некоторое условие.

Чтобы эффективно использовать циклы, разработчик должен правильно определять логику их начала и завершения. `while` и `for` — популярные циклы, с которыми могут экспериментировать в своем коде даже начинающие Python-разработчики.

# Условные операторы if/else

Условные операторы зависят от принятия решений при выполнении конкретных операций. Если условие не выполняется, соответствующий блок условной логики пропускается.

Для выбора между двумя блоками в Python существует базовая команда `if/else`.

**СИНТАКСИС:**

```
if условие:
    выполняемая команда
else:
    выполняемая команда
```

**ПРОГРАММНЫЙ КОД:**

```
number = 43
if number % 3 == 0:
    print ("This number is divided by 3")
else:
    print ("This number is not divisible by 3")
```

**ВЫВОД:**

```
This number is not divisible by 3
```

## Объяснение

- Сначала необходимо определить переменную, которая будет хранить данные, проверяемые условием `if/else`.
- Код, который выполняется после команды `if (else)`, должен иметь отступ (четыре знака «пробел»).
- В более сложных программах используются средства ввода (input) для получения данных непосредственно от пользователя.
- После того как переменная будет сохранена, интерпретатор обрабатывает условие блока `if`.
- Интерпретатор Python выполняет операцию вычисления остатка и определяет, кратно ли 3 проверяемое число.
- Если число кратно 3, выполняется блок, расположенный под ключевым словом `if`.
- Так как в данном случае условие ложно, интерпретатор пропускает блок `if` и выполняет код блока `else`. В результате выводится то значение, которое вы видите.

В следующем примере выполняется код блока `if`.

**ПРОГРАММНЫЙ КОД:**

```
number = 40
if number % 4 == 0:
    print ("This number is divided by 4")
else:
    print ("This number is not divisible by 4")
```

**ВЫВОД:**

```
This number is divided by 4
```

Так как условие истинно, выполняется команда `print` блока `if`, а блок `else` пропускается интерпретатором.

## Операторы if, elif, else

Чтобы сделать серию проверок более эффективной, можно добавить несколько проверок выполнения условий.

**ПРОГРАММНЫЙ КОД:**

```
sample = 45
if sample % 3 == 0:
    print ("This number is divisible by 3")
elif sample % 4 == 0:
    print("This number is divisible by 4")
else:
   print ("This number is not divisible by 3 and
➥  4")
```

**ВЫВОД:**

```
This number is divisible by 3
```

В этом примере интерпретатор Python должен проверить три условия. Определив, что первое условие истинно, интерпретатор Python выводит его и пропускает два других условия.

# Цикл for

Циклы, как и условные конструкции, относятся к числу основных строительных блоков программ Python. Вместо того чтобы многократно выполнять одну и ту же операцию, например проверять условие, можно воспользоваться циклом `for` или `while`.

В циклах `for` могут использоваться любые разновидности структур данных: списки, кортежи, строки, словари.

**СИНТАКСИС:**

```
for val in list:
    {Здесь размещается тело цикла}
```

Цикл `for` обеспечивает перебор всех элементов.

**ПРИМЕР:**

```
x = [32, 12, 11]
sample = 0
for val in x:
    sample = sample + val
print ("The sum of numbers is", sample)
```

**ВЫВОД:**

```
The sum of the numbers is 55
```

Вместо того чтобы выполнять арифметические операции с каждым элементом списка по отдельности, мы просто используем цикл `for` для автоматизации обработки.

# Цикл while

Цикл `for` хорошо подходит для автоматизации, но он несколько усложняет реализацию логики в коде, потому что на уровне цикла нельзя задать проверяемое условие завершения цикла. В таких ситуациях используется цикл `while`.

В заголовке цикла `while` задается условие, которое будет проверяться при каждой итерации цикла.

**СИНТАКСИС**

```
while условие
    {тело цикла}
```

**ПРИМЕР:**

```
y = 0
z = 1
x = int(input("Enter number: "))
while z <= x:
    y = y + z
    z = z + 1
print("The sum of numbers is: ", y)
```

**ВЫВОД:**

```
Enter number: 3
The sum of numbers is: 6
```

При создании сложных программ часто используются вложенные условные конструкции и циклы.

# Операторы break и continue

Циклы позволяют выполнять сложную программную логику за меньшее время. Они удобны во многих ситуациях, но иногда расходуют слишком много памяти, что приводит к неожиданным сбоям программ.

Для решения этой проблемы Python предоставляет два оператора: `break` и `continue`.

## Как работает break

Каждый раз, когда интерпретатор Python встречает в программе оператор `break`, он немедленно завершает цикл и продолжает выполнение программы со строки, следующей за циклом. Если `break` встречается во вложенном цикле, то внутренний цикл завершается, а внешний продолжает выполняться.

**СИНТАКСИС:**

```
break
```

**ПРОГРАММНЫЙ КОД:**

```
n = 4
i = 1
while i <= n:
    if i % 2 == 0:
        print(i, "is divided by 2")
    if i % 3 == 0:
        print(i, "is divisible by 3")
        break
    i = i + 1
```

**ВЫВОД:**

```
2 is divided by 2
3 is divisible by 3
```

В этом примере интерпретатор завершает программу, когда встречает команду `break`.

А какой результат был бы выведен без `break`?

## Как работает continue

Каждый раз, когда интерпретатор Python встречает в программе команду `continue`, он немедленно завершает текущую итерацию цикла и переходит к следующей. Помните, что эта команда не завершает цикл.

Команда `continue` экономит время и вычислительные ресурсы за счет пропуска нежелательных команд в цикле.

**ПРИМЕР:**

```
for var in 'computer':
    if var == 'r':
        continue
    print('Letter now:', var)
```

**ВЫВОД:**

```
Letter now: c
Letter now: o
Letter now: m
Letter now: p
Letter now: u
Letter now: t
Letter now: e
```

## Упражнения

- Напишите программу для вывода чисел, не превышающих 2000, кратных 12 и делящихся на 5. Используйте разделители при выводе элементов.

- Напишите программу для преобразования фунтов в килограммы с использованием циклов `for` и `while`.

- Напишите на Python генератор случайных чисел в диапазоне от 1000 до 10 000.

- Используйте циклы для вывода по крайней мере пяти узоров ранголи, используя буквы алфавита.

- Напишите программу для вычисления последовательности Фибоначчи с использованием команды `continue`.

- Используя циклы, напишите программу, которая может переводить суммы в долларах США в евро и фунты.

- Напишите программу для проверки вводимых учетных данных с паролем. Убедитесь в том, что при проверке соблюдаются стандарты выбора паролей.

# Глава 8

# Функции и модули

Язык Python поддерживает различные парадигмы. Парадигма процедурного программирования — самая популярная из всех, используемых разработчиками при написании кода. Процедурное программирование универсально; оно легко реализуется в простых проектах и требует меньшего количества разработчиков для завершения кода. Для процедурной парадигмы также характерна ускоренная реализация различных программных компонентов.

Процедурное программирование основано на вызове функций. Изучение процедурного программирования на нескольких примерах поможет вам строить сложные программы с меньшим объемом кода.

# Для чего нужны функции

Первоначально функции использовались для решения сложных задач в дискретной математике. Позднее программисты стали реализовывать эту концепцию, чтобы код можно было использовать повторно, не переписывая.

Для демонстрации работы функций мы воспользуемся примером простого мобильного приложения.

Picsart — популярное мобильное приложение для редактирования фотографий, предоставляющее ряд фильтров и инструментов для обработки изображений. Например, один из инструментов помогает пользователям легко обрезать края изображения.

Разработчики Picsart в процессе написания кода обычно используют разные библиотеки, фреймворки и функции. Например, обрезка оформляется в виде отдельной функции, так как она требует выполнения множества сложных операций для деления пикселей и вывода результата.

Теперь допустим, что разработчики решили обновить приложение и включить в него поддержку обрезки видео.

В текущем состоянии программистам доступны два варианта.

- Написать функцию обрезки с нуля.

- Воспользоваться функцией обрезки, созданной для фотографий, и дополнить ее новой функциональностью.

Многие разработчики предпочитают второй вариант, потому что он проще реализуется и экономит много времени. Впрочем, создавать функции не так просто. Зачастую требуется сложная логика, связывающая функции с инфраструктурой разработки приложений и другими сторонними библиотеками.

# Разновидности функций

Функции обычно делятся на две категории — системные и пользовательские.

Системные функции предоставляются основной библиотекой Python и часто используются разработчиками для выполнения стандартных операций. Например, `print` — системная функция, которая используется для вывода данных на экран.

Пользовательские функции создаются самими разработчиками, обычно в них реализуется уникальная логика. Разработчики также могут импортировать функции из сторонних библиотек для интеграции со своим кодом.

Какую бы разновидность вы ни применяли, помните: главная цель функций, с точки зрения программиста, — решение задач с меньшим объемом кода и возможность его повторного использования.

# Как работают функции

По своему принципу функции в программировании напоминают математические функции. Разработчик сначала определяет функцию со сложной логикой и присваивает ей имя, по которому ее можно будет вызвать из любой точки программы. При создании функции определяются специальные программные компоненты, которые называются параметрами. Для снижения риска ошибок разработчики явно указывают параметры, которые функция может принимать.

Если функция не вызывается в программе, пользователь не сможет воспользоваться логикой, написанной разработчиком. В интерфейсе приложений вызов функций часто представляется кнопками, вкладками и другими графическими элементами. И хотя такие элементы могут быть всего лишь вспомогательными средствами для пользователя, на программном уровне компонент вызывает функцию, благодаря чему алгоритм работает так, как планировалось.

# Как определять собственные функции

Стандартные системные функции определять не придется, так как они уже созданы заранее. От вас потребуется лишь вызывать их. И хотя программисты могут вносить изменения в системные функции, делать этого не рекомендуется, так как обычно они весьма сложны и любое вмешательство может нарушить работоспособность вашего кода.

Особо увлеченные разработчики могут определять собственные функции с помощью ключевого слова `def`.

Простой пример поможет понять, как выглядит объявление функции в Python.

**ПРОГРАММНЫЙ КОД:**

```
def sample():
    # Функция выводит приветственное сообщение для
# пользователя
    print("Hello! Hope you are fine. Good morning")


sample()
```

**ВЫВОД:**

```
Hello! Hope you are fine. Good morning
```

## Объяснение

- Хотя эта программа очень проста, основная схема остается той же, что и в самых сложных функциях. При создании реальных программ увеличивается только количество операций.

- В первой строке ключевое слово `def` указывает на начало функции. Если пропустить ключевое слово `def`, программа работать не будет — интерпретатор просто не поймет, что вы определяете функцию.

- После `def` указывается имя функции. В данном примере функции присваивается имя `sample`. Имена функций подчиняются тем же правилам, что и имена переменных.

- Все, что следует после комментария, называется телом функции. Тело функции может содержать переменные, константы и вызовы других функций. Обычно в теле определяется основная логика функции.

- Тело функции обычно начинается с комментария или doc-строки. В приведенном выше примере используется комментарий. Также возможно предоставить информацию о функции, заключенную в утроенные одинарные или двойные кавычки — это называется doc-строкой.

**ПРИМЕР:**

```
'''This is a sample function we are explaining
➥  for beginners.'''
```

Если комментарий состоит из нескольких строк, его можно оформить так:

```
"""
Author: Sam
Function: Sample
What does it do?: It just prints
"""
```

В третьей строке примера содержится команда `print` для вывода информации на экран. В теле пользовательской функции может быть сколько угодно встроенных функций, чтобы ваша программа выглядела более естественной и понятной. Хотя в данном примере выводимая информация статична, он помогает понять, как работают традиционные приложения.

В последней строке демонстрируется вызов функции разработчиком. Здесь `sample()` — вызов функции. Так как это очень простая программа, в круглых скобках параметры не указываются. В сложных программах при вызове может передаваться набор параметров. Встречаясь с вызовом функции, интерпретатор немедленно обращается к коду функции и выполняет логику.

# Использование параметров в функциях

В рассмотренном примере функция вызывалась без параметров. В реальных приложениях, как правило, параметры присутствуют, так как логика таких программ часто оказывается сложной и запутанной. Чтобы оценить все преимущества функций, следует создать код, который получает параметры и выполняет какие-то содержательные операции.

Теперь представьте, что у приложения два пользователя и мы хотим поприветствовать их по именам.

**ПРОГРАММНЫЙ КОД:**

```
def sample(name):
    # Функция выводит приветственное сообщение
# для пользователя
    if name == "Sam":
        print("Hello! Hope you are fine Sam.
➥ Good morning")
    elif name == "Tom":
        print("Hello! Hope you are fine Tom.
➥ Good morning")


sample()
```

Для того чтобы в вышеприведенном примере вывести соответствующий результат, приходится создать две команды `print`, также необходимо использовать условные проверки

для определения имени пользователя. Логика получается слишком сложной, и на самом деле она избыточна, так как параметры позволят динамически построить приветствие для конкретного человека — и не для двух, а для тысяч потенциальных пользователей — с минимальными изменениями при создании функции.

Например, следующая функция с одним параметром позволяет строить сообщения динамически.

**ПРОГРАММНЫЙ КОД:**

```
def sample(name):
    '''Пример функции с одним параметром'''
    print("Hello " + name + "." + " Glad that you
➥  are back here. Good Morning")


sample('Sam')
sample('Tom')
sample('Rick')
sample('Damon')
```

**ВЫВОД:**

```
Hello Sam. Glad that you are back here.
➥  Good Morning
Hello Tom. Glad that you are back here.
➥  Good Morning
Hello Rick. Glad that you are back here.
➥  Good Morning
Hello Damon. Glad that you are back here.
➥  Good Morning
```

### Объяснение

- Функция создается с именем `sample`, а в круглых скобках определяется параметр `name`. Тип данных параметра указывать не нужно — интерпретатор Python достаточно умен, чтобы разобрать любое значение данных, переданное пользователем.

- При вызове `print` программист указывает имя параметра и объединяет строки оператором `+`. Таким образом, какое бы значение ни ввел пользователь, оно будет выводиться между строками с неизменным текстом.

- В следующих строках функция вызывается с указанием значения параметра. В нетривиальных приложениях оно обычно зависит от ввода пользователя. В нашем примере используются значения `Sam`, `Tom`, `Rick` и `Damon`, заданные разработчиком.

## Передача аргументов

Практически в любом современном приложении при вызове функций используются параметры. Каждому параметру соответствует аргумент, передаваемый функции. Хотя существует несколько способов передачи аргументов, которые определяют значения параметров функции, на практике чаще всего применяются позиционные и именованные аргументы.

## Позиционные аргументы

При использовании позиционных аргументов программисты обычно напрямую перечисляют значения. На первый взгляд последовательность значений может показаться непонятной, но именно этот способ часто применяется программистами из-за простоты реализации.

С позиционными аргументами необходимо помнить порядок, в котором должны передаваться значения.

**ПРОГРАММНЫЙ КОД:**

```
def football(country, number):
    # Сообщает, сколько раз страна выигрывала кубок
# мира FIFA
    print(country + " has won FIFA " + str(number)
➥  + " times")


football('Argentina', 4)
football('England', 2)
```

**ВЫВОД:**

```
Argentina has won FIFA 4 times
England has won FIFA 2 times
```

Здесь в первом случае передаются значения `'Argentina'` и `4`. Так как типы данных не указаны, интерпретатор Python определяет тип значения автоматически и передает его функции.

Программист не всегда может с первого взгляда понять тип данных, который должен использоваться в каждом конкретном случае, поэтому имена параметров играют важную роль. При чтении этого кода вы сразу понимаете,

что `country` (страна) содержит строку, тогда как для `number` (номер) используется целочисленный тип данных. Аргументы обычно разделяются запятыми. Итак, при вызове функции в обоих случаях второй параметр имеет целочисленный тип. Чтобы в выводимом на экран сообщении объединить целое число и строку, необходимо привести параметр `number` к строковому типу.

Как видно из следующего примера, при использовании позиционных аргументов легко допустить ошибку.

**ПРОГРАММНЫЙ КОД:**

```
def football(country, number):
    # Сообщает, сколько раз страна выигрывала кубок
# мира FIFA
    print(country, "has won FIFA", number, "times")


football(4, 'Argentina')
football(2, 'England')
```

**ВЫВОД:**

```
4 has won FIFA Argentina times
2 has won FIFA England times
```

Хотя функция выдает результат, особого смысла в нем нет, потому что при вызове разработчик передал аргументы в обратном порядке.

Для решения подобных проблем с позиционными аргументами разработчики могут при вызове функций использовать механизм именованных аргументов.

## Именованные аргументы

С именованными аргументами можно напрямую передавать значения для конкретных параметров функций. Такие аргументы передаются в формате «*параметр = значение*».

Именованные аргументы снижают вероятность ошибок, но сложнее реализуются, поэтому они относительно редко используются разработчиками в сложных проектах с большим объемом кода.

**ПРОГРАММНЫЙ КОД:**

```
def football(country, number):
    # Сообщает, сколько раз страна выигрывала кубок
# мира FIFA
    print(country + " has won FIFA " + str(number)
➥  + " times")


football(country='Argentina', number=4)
football(country='England', number=2)
```

**ВЫВОД:**

```
Argentina has won FIFA 4 times
England has won FIFA 2 times
```

Здесь ключевые аргументы определяются в формате «*параметр = значение*». Например, в `country='Argentina'` `country` — параметр, а `'Argentina'` — передаваемый аргумент.

## Аргументы по умолчанию

В процессе написания программы на Python или любом другом языке программирования не все значения обязательно должны быть динамическими. Иногда при передаче аргументов для параметров функции разработчики используют фиксированные значения, то есть константы. Использовать фиксированные значения для параметров функций не обязательно, это делается исключительно по решению разработчика.

Однако программистам Python рекомендуется определять значения по умолчанию, потому что они сокращают объем шаблонного кода и упрощают управление данными в сложных проектах. Шаблонным кодом называется код, без которого, в принципе, можно было бы обойтись, но который должен быть написан разработчиком, чтобы интерпретатор работал без проблем. Хотя Python не загромождается лишним кодом по сравнению с другими языками высокого уровня, для улучшения его удобочитаемости стоит внести некоторые изменения, включая определение значений по умолчанию.

**ПРОГРАММНЫЙ КОД:**

```
def football(country, number=4):
    print(country + " has won FIFA cup " +
➥  str(number) + " times")


football('Argentina')
football('Brazil')
```

**ВЫВОД:**

```
Argentina has won FIFA cup 4 times
Brazil has won FIFA cup 4 times
```

В этом примере для параметра уже определено значение по умолчанию, поэтому вызов функции упрощается и занимает меньше времени.

> **ПРИМЕЧАНИЕ**
>
> Следует помнить, что интерпретатор Python при переопределении аргумента заменит его, несмотря на то, что для аргумента определено значение по умолчанию.

**ПРОГРАММНЫЙ КОД:**

```
def football(country, number=4):
    print(country + " has won FIFA cup " +
➥  str(number) + " times")


football('Argentina')
football('Brazil', 5)
```

**ВЫВОД:**

```
Argentina has won FIFA cup 4 times
Brazil has won the FIFA cup 5 times
```

Для аргумента определено значение по умолчанию `4`. Тем не менее, когда при вызове для `Brazil` передается значение `5`, интерпретатор Python заменяет значение по умолчанию новым значением аргумента.

# Область видимости в Python

Концепция области видимости чрезвычайно важна, так как с ней разработчик может понять, какие функции ему доступны и как пользоваться ими без проблем. Как упоминалось ранее, для функций — как и для переменных — существует локальная и глобальная область видимости.

Все переменные, которые были созданы внутри функции и могут использоваться только внутри нее, называются переменными с локальной областью видимости или просто локальными переменными. Переменные, которые могут использоваться в любой точке программы, называются переменными с глобальной областью видимости или глобальными переменными. Таким образом, каждая переменная, использованная в функции, должна быть либо локальной, либо глобальной.

## Почему важна область видимости

Главная причина для использования концепции области видимости заключается в том, что она помогает более эффективно использовать механизм сборки мусора. Все значения, которые были заменены или не используются в течение долгого времени, обычно уничтожаются для ускорения работы программы. Хотя они могут быть созданы повторно при вызове функции, это все равно расходует вычислительные ресурсы.

Когда программист создает переменную в глобальной области видимости, вероятно, он планирует обращаться к этой переменной многократно; глобальная область види-

мости поможет ему избежать многократной инициализации переменных. Какую бы программу вы ни разрабатывали, сознательное использование области видимости повысит эффективность вашей работы над сложными проектами.

## Локальная и глобальная область видимости

**Правило 1:** переменные с локальной областью видимости не могут использоваться в глобальной области видимости.

**ПРОГРАММНЫЙ КОД:**

```
def sample():
    example = 24
    print(example)


sample()
print(example)
```

**ВЫВОД:**

```
NameError: name 'example' is not defined
```

В этом примере в локальной области видимости создается переменная со значением `24`. Если вызвать эту функцию и попытаться вывести значение переменной из глобальной области видимости, вы получите сообщение об ошибке, потому что программист может обращаться к локальным переменным внутри функции, но не в глобальной области видимости.

**ПРОГРАММНЫЙ КОД:**

```
def sample():
    example = 24
    print(example)


sample()
```

**ВЫВОД:**

```
24
```

Теперь программа работает, потому что обращение к переменной происходит из локальной области видимости. Соответственно, программа выполняется без каких-либо проблем, а значение локальной переменной выводится командой `print`.

**Правило 2:** все локальные функции могут использовать как локальные, так и глобальные переменные.

**ПРОГРАММНЫЙ КОД:**

```
trail = 62


def sample():
    example = 24
    print(trail)


sample()
print(trail)
# При попытке выполнения команды print(example) вне
# функции произойдет ошибка
```

**ВЫВОД:**

```
62
62
```

В этом примере значение переменной `trail` выводится оба раза — при обращении к переменной как из локальной, так и из глобальной области видимости.

**Правило 3:** локальные переменные одной функции не могут использоваться другой функцией.

**ПРОГРАММНЫЙ КОД:**

```
def sample():
    example = 24
    print(example)


sample()


def sample2():
    print(example)


sample2()
```

**ВЫВОД:**

```
24
NameError: name 'example' is not defined
```

Функция `print` работает при первом вызове, потому что в ней используется переменная из локальной функции. Однако во второй раз выдается сообщение об ошибке, потому что функция не может обратиться к локальной переменной из другой функции.

**ПРИМЕЧАНИЕ**

Переменным из локальной и глобальной области видимости можно присвоить одинаковые имена, это не создаст никаких конфликтов. Однако ради сохранения хорошего стиля программирования и для предотвращения путаницы лучше присваивать разные имена локальным и глобальным переменным.

# Модули

Группа взаимосвязанных функций образует *модуль*. Всякий раз, когда вы захотите использовать эту группу функций в каком-либо программном компоненте, можно просто импортировать модуль и вызвать содержащиеся в нем функции с нужными значениями аргументов.

В Python механизм импортирования модулей намного удобнее, чем в языках C. Многие программисты импортируют модули, чтобы использовать содержащиеся в них методы и строить на их основе новую функциональность

**СИНТАКСИС:**

```
import { имя модуля }
```

**ПРИМЕР:**

```
import clock
```

Этот синтаксис импортирует все встроенные функции, содержащиеся в модуле `clock`, в вашу программу. После импортирования вы можете вызывать эти методы и передавать им нужные аргументы.

## Что делает import

Встроенная библиотечная функция `import` копирует все функции, содержащиеся в некотором файле, и связывает их с текущим файлом. По сути, она дает вам возможность

использовать методы, отсутствующие в текущем файле. Создание модулей избавляет вас от многократного написания одного и того же кода.

## Как создать модуль

Импортирование модулей сторонних библиотек и фреймворков экономит время, но разработчик также должен уметь создавать собственные модули.

Допустим, вы создаете веб-приложение для торрент-сервиса. Чтобы приложение заработало, необходимо написать множество функций. Для улучшения организации кода можно создать сетевой модуль и объединить в нем все функции, относящиеся к передаче данных по сети. Затем можно создать модуль, относящийся к графическому интерфейсу, и написать функции, которые помогут создать приложение с хорошим внешним видом.

Модуль Python создается, когда текстовому файлу с кодом присваивается расширение .py.

После того как файл .py будет создан, в нем сохраняются все функции модуля.

Например, в только что созданном файле .py можно сохранить приведенную ниже функцию для умножения двух целых чисел.

**ФАЙЛ SAMPLE_MODULE.PY:**

```
def product(x, y):
    # Используется для вычисления произведения двух
# чисел
    z = x * y
    # Функция возвращает произведение
    return z
```

Когда модуль создан, можно написать программу, которая импортирует эту функцию.

**ПРОГРАММНЫЙ КОД:**

```
import sample_module
```

Все функции указанного модуля становятся доступными в текущем файле.

**ПРОГРАММНЫЙ КОД:**

```
sample_module.product(3, 6)
```

**ВЫВОД:**

```
18
```

Программа автоматически находит функцию `product`, и на экран выводится произведение заданных аргументов.

# Встроенные функции и модули

Разработчики могут использовать встроенные функции и модули при создании сложных, нетривиальных приложений. Хотя пользовательские функции полезны, поскольку они предоставляют разработчику свободу при решении сложных задач, они все же создают определенные сложности с реализацией, причем иногда пользовательские функции излишни, потому что встроенные функции могут выполнить работу за вас.

## print()

Пожалуй, это самая популярная встроенная функция в библиотеке Python. Все, от новичков до опытных программистов, пользуются командой `print()` для вывода информации на экран. Обычно текст, который должен быть выведен на экран, заключается в кавычки.

**ПРОГРАММНЫЙ КОД:**

```
print("This is an example")
```

**ВЫВОД:**

```
This is an example
```

## abs()

Встроенная функция `abs()` возвращает абсолютное значение любого целого числа. Как правило, при получении отрицательного целого числа эта функция меняет его знак.

**ПРОГРАММНЫЙ КОД:**

```
x = -24
print(abs(x))
```

**ВЫВОД:**

```
24
```

## round()

Встроенная математическая функция `round()` возвращает целое число, ближайшее к заданному числу с плавающей точкой.

**ПРОГРАММНЫЙ КОД:**

```
x = 2.46
y = 3.12
print(round(x))
print(round(y))
```

**ВЫВОД:**

```
2
3
```

## max()

Встроенная функция `max()` используется для нахождения наибольшего числа в группе. Функция может использоваться с любым типом данных, включая переменные и списки.

**ПРОГРАММНЫЙ КОД:**

```
a = 45
b = 43
c = 23
solution = max(a, b, c)
print(solution)
```

**ВЫВОД:**

```
45
```

## min()

Встроенная функция `min()` используется для нахождения наименьшего числа в группе.

**ПРОГРАММНЫЙ КОД:**

```
a = 45
b = 43
c = 23
solution = min(a, b, c)
print(solution)
```

**ВЫВОД:**

```
23
```

## sorted()

Встроенная функция `sorted()` используется для сортировки всех элементов, содержащихся в итерируемом объекте, например в списке, кортеже, множестве или в словаре, по возрастанию или убыванию (по усмотрению программиста).

**ПРОГРАММНЫЙ КОД:**

```
x = [2, 323, 21, 5, 242, 11]
y = sorted(x)
print(y)
```

**ВЫВОД:**

```
[2, 5, 11, 21, 242, 323]
```

## sum()

Специальная встроенная функция `sum()` суммирует все элементы последовательности. Прежде чем вызывать эту встроенную функцию, убедитесь в том, что все элементы объекта относятся к одному типу. В противном случае программа выдает сообщение об ошибке, так как суммирование значений двух разных типов данных невозможно.

**ПРОГРАММНЫЙ КОД:**

```
x = (32, 43, 11, 12, 19)
y = sum(x)
print(y)
```

**ВЫВОД:**

```
117
```

## len()

Встроенная функция `len()` предоставляет информацию о количестве элементов, содержащихся в последовательности, например в списке, словаре, строке или кортеже.

**ПРОГРАММНЫЙ КОД:**

```
x = (1, 23, 32, 11, 12)
y = len(x)
print(y)
```

**ВЫВОД:**

```
5
```

## type()

Встроенная функция `type()` предоставляет информацию о том, какой тип данных хранится в объекте.

Если это функция, также выводится информация о ее параметрах.

**ПРОГРАММНЫЙ КОД:**

```
x = 23.2121
print(type(x))
```

**ВЫВОД:**

```
<class 'float'>
```

# Строковые методы

Строковый тип данных чрезвычайно популярен, поэтому он требует от программиста большего внимания, чем другие типы данных.

Основная библиотека Python предоставляет десятки встроенных функций, которые помогают программисту наиболее эффективно использовать данные, хранящиеся в строковом типе.

## strip()

Встроенный метод `strip()` удаляет символы из аргумента, переданного функции. Удаляются все вхождения заданной подстроки.

**ПРОГРАММНЫЙ КОД:**

```
x = "Welcome"
print(x.strip("me"))
```

**ВЫВОД:**

```
Welco
```

## replace()

Встроенный метод `replace()` заменяет часть строки другой строкой. Если строка состоит из нескольких слов, можно указать количество заменяемых слов в параметре.

**ПРОГРАММНЫЙ КОД:**

```
example = "This is not a good sign"
print(example.replace("good", "bad"))
```

**ВЫВОД:**

```
This is not a bad sign
```

## split()

Встроенный метод `split()` автоматически разбивает строку по вхождениям заданной подстроки.

**ПРОГРАММНЫЙ КОД:**

```
example = "There are nine planets"
print(example.split("re"))
```

**ВЫВОД:**

```
['The', 'a', 'nine planets']
```

Так как переданная подстрока встречается в строке дважды, строка разбивается на три части по разделителю `"re"`.

## join()

Специальный метод `join()` объединяет элементы последовательности. По желанию можно добавить разделитель.

**ПРОГРАММНЫЙ КОД:**

```
x= [23, 11, 12, 56]
sample ="~"
sample = sample.join(x)
```

**ВЫВОД:**

```
23 ~ 11 ~ 12 ~ 56
```

## Упражнения

- Напишите программу Python, которая генерирует десять случайных чисел, а потом автоматически находит наибольшее из них. Используйте метод `max()` для решения задачи.

- Создайте список, переставьте все элементы в обратном порядке и последовательно просуммируйте их.

- Напишите программу Python, которая принимает на вход десять строк и переставляет их символы в обратном порядке.

- Напишите рекурсивную функцию для вычисления факториала 100.

- Создайте текст из трех страниц и потренируйтесь на нем в использовании строковых операций. Строки должны быть в том же виде, в котором они обычно представляются на бумаге. Постарайтесь использовать как можно больше разных методов.

- Напишите программу Python для построения последовательности рядов значений, образующих треугольник Паскаля.

- Напишите программу Python, которая автоматически загружает из «Википедии» статью, указанную пользователем.

- Напишите программу Python, создающую цветовую схему для всех цветов в формате RGB.

# Глава 9

# Объектно-ориентированное программирование

В предыдущей главе мы рассмотрели парадигму процедурного программирования с несколькими примерами кода. Хотя парадигма процедурного программирования весьма популярна среди независимых разработчиков, ее применение заметно усложняется при работе в команде, где участники должны взаимодействовать друг с другом для эффективного использования кода, написанного другими людьми.

С другой стороны, при процедурном подходе, несмотря на меньшую громоздкость кода, приходится импортировать модули при создании каждого нового файла. Импортирование большого количества модулей также стремительно увеличивает время выполнения программы.

Из-за этих проблем многие программисты на момент выхода первой версии Python предпочитали использовать объектно-ориентированные языки (такие как Java). Однако при выпуске Python 2 разработчики с радостью узнали, что в Python появилась поддержка объектно-ориентированного программирования, вследствие чего он стал мультипарадигменным языком.

В этой главе мы сосредоточимся на принципах объектно-ориентированного программирования, которые будут рассмотрены на нескольких примерах.

# Что такое объектно-ориентированное программирование

Объектно-ориентированное программирование (ООП) — популярная парадигма программирования, для которого характерна логическая группировка функций, определяемых в программе, по классам и объектам.

Класс состоит из набора полей данных и методов, к которым программист может легко обращаться через нотацию с точкой. Объекты позволяют переменным и методам за пределами класса работать с этим классом.

## Пример использования

Допустим, вы создаете приложение, которое выводит информацию о разных видах транспорта и их моделях.

В процедурном программировании разработчик создал бы функцию для каждого вида транспорта, а затем для каждой модели. При небольшом количестве моделей такое решение кажется простым, но с ростом их числа повторное использование кода усложняется.

В объектно-ориентированном программировании разработчик сначала создает класс `Vehicle`, представляющий транспортное средство, и определяет различные свойства и значения. Затем разработчик определяет отдельный класс для каждого типа транспортного средства. Вместо

того чтобы снова создавать функции для каждого свойства, разработчик может обращаться к этим свойствам через нотацию с точкой — такую возможность ему предоставляет парадигма объектно-ориентированного программирования.

ООП экономит много времени и позволяет организовать повторное использование кода с помощью таких механизмов, как полиморфизм и наследование.

# Как создать класс в Python

Классы, по сути, представляют собой «чертежи», на основании которых создаются объекты. Классы содержат такие логические сущности, как атрибуты и методы.

При создании класса необходимо знать некоторые правила.

- Все классы, определяемые в программе, должны помечаться ключевым словом `class`.
- Переменные, создаваемые внутри класса, представляют собой атрибуты класса.
- Все атрибуты класса являются общедоступными, и к ним всегда можно обратиться при помощи оператора `.` (точка).

**СИНТАКСИС СОЗДАНИЯ КЛАССА:**

```
class ИмяКласса:
    # Команды класса
```

**ПРОГРАММНЫЙ КОД:**

```
# Демонстрация создания классов
class Country:
    # Команды класса
```

> **ПРИМЕЧАНИЕ**
>
> Зарезервированные ключевые слова не могут использоваться в качестве имен классов в Python. Если вы попытаетесь это сделать, будет выведено сообщение об ошибке и программа завершит работу.

# Как создаются объекты

Объект в программировании на языке Python представляет собой сущность, с которой связывается состояние и поведение. Все, что находится внутри класса, можно считать объектом. Например, переменная, созданная внутри класса, может использоваться как объект. Программисты часто пользуются объектами, даже не подозревая об этом.

## Что содержат объекты

- Каждый объект обладает состоянием. Состояние обычно отражает значения свойств, связанных с объектом.

- Каждый объект обладает поведением. Поведение объекта изменяется в соответствии с методом, в котором он используется.

- Все объекты обладают идентичностью (то есть идентификационными характеристиками), которая позволяет им взаимодействовать с другими объектами.

Допустим, существует класс `Dog`, который описывает разные породы собак и их поведение. Этот класс может представлять объекты разных видов.

- Кличку собаки можно использовать для идентификации объекта.

- Такие атрибуты, как порода, возраст и цвет шерсти, могут быть отнесены к состоянию объекта.

- Разные виды поведения, присущие собакам, — лай, сон, бег — можно отнести к поведению объекта.

## Пример создания объекта

Чтобы создать объект, достаточно указать его имя и класс. Например, если ранее был определен класс `Dog`, можно использовать следующий код.

**ПРОГРАММНЫЙ КОД:**

```
obj = Dog()
```

Эта команда создает объект с именем `obj`, принадлежащий классу `Dog`.

## Параметр self

Программисты Python должны знать о существовании параметра `self`, который используется для ссылки на экземпляр класса.

Параметр `self` имеет много общего с указателями `this`, используемыми в высокоуровневых языках программирования, таких как C и C++, однако, в отличие от `this`, `self` не является ключевым словом, это просто соглашение, и его можно называть как угодно.

# Метод __init__

Метод `__init__` можно считать аналогом конструкторов в C++ и Java. Каждый раз, когда в программе создается объект класса, этот метод выполняется по умолчанию. Таким образом, если вы хотите создать объект, инициализированный конкретным значением, это значение необходимо передать методу `__init__`.

Создадим программу Python с использованием метода `___init__`.

**ПРОГРАММНЫЙ КОД:**

```
class Geography:
    # Создание атрибута класса
    attr1 = "country"

    # Создание атрибута экземпляра
    def __init__(self, name):
        self.name = name


# Создание объекта совмещается с инициализацией
USA = Geography("USA")
UK = Geography("UK")

# Доступ к атрибутам класса
print ("USA is a {} ".format(USA.attr1))
print("UK is a {} ".format(UK.attr1))

# Доступ к атрибутам экземпляров класса
print("Country name is {}".format(USA.name))
print("Country name is {}".format(UK.name))
```

**ВЫВОД:**

```
USA is a country
UK is a country
Country name is USA
Country name is UK
```

В этом примере создается класс, а затем — атрибуты класса и экземпляра. Делать это каждый раз при создании класса не обязательно; я привел эту программу только для того, чтобы вы лучше поняли, как работают классы и объекты.

- Укажите имя класса.
- Создайте хотя бы один атрибут.
- Предоставьте метод `__init__` с аргументом `self`.
- Создайте объекты класса.

## Как создаются классы и объекты с методами

Напишем код по типичной схеме, которая применяется разработчиками для создания методов и вызова их с использованием объектов.

**ПРОГРАММНЫЙ КОД:**

```
class Geography:
    # Создание атрибута класса
    attr1 = "country"

    # Создание атрибута экземпляра
    def __init__(self, country_name):
        self.country_name = country_name

    def governance(self):
        print("This country is {}".format
➥  (self.country_name))


# Создание объекта
USA = Geography("USA")
UK = Geography("UK")
USA.governance()
UK.governance()
```

**ВЫВОД:**

```
This country is USA
This country is UK
```

### Объяснение

В этом примере сначала создается атрибут класса, а затем в методе `__init__` выполняется инициализация. Далее программа создает объекты и вызывает для них метод, обращаясь через точку.

# Наследование

Наследование — один из важнейших механизмов объектно-ориентированного программирования. Под этим термином понимается возможность определения новых классов на основе других, уже существующих. Новый класс обычно называется дочерним, а класс, от которого он порождается, — родительским.

## Пример использования

Наследование полезно во многих ситуациях. Представьте, что вы создаете мобильное приложение для работы с камерой на платформе iOS. В процессе разработки приложения вам пришлось создать несколько модулей для различных функций, предоставляемых приложением. За несколько месяцев разработки вы заметили, что при процедурном подходе разработки код графического интерфейса дублируется.

Для экономии времени и денег вы решаете реализовать объектно-ориентированную парадигму в своем проекте. Это позволяет вам взять код, уже написанный для графических интерфейсов, и связать его с новыми классами, которые вы создаете. Тем самым экономятся время и силы, а программисты могут реализовать новые возможности без переписывания старой, уже существующей функциональности.

**СИНТАКСИС НАСЛЕДОВАНИЯ В PYTHON:**

```
class BaseClass:
    { Тело базового класса }


class DerivedClass(Baseclass):
    { Тело производного класса }
```

**ПРИМЕЧАНИЕ**

При использовании как базовых, так и производных классов необходимо соблюдать все правила, упоминавшиеся ранее.

**ПРОГРАММНЫЙ КОД:**

```
# Определение класса 'Polygon'
class Polygon:
    def __init__(self, sides):
        self.sides = sides

    def dispsides(self):
        for i in range(self.sides):
            print("side", i+1)


# Определение класса Square на основе Polygon
class Square(Polygon):
    def __init__(self):
        self.sides = int(input("Side of the square:
➥ "))

    def ind_area(self):
        a = self.sides
        # Вычисление площади
        s = a * a
        print("The area of the square is", s)


# Определение многоугольника с 5 сторонами
x = Polygon(5)
x.dispsides()
# Программа определяет квадрат, запрашивает длину
# стороны у пользователя и вычисляет его площадь
x2 = Square()
x2.ind_area()
```

## Объяснение

В этой программе мы сначала определяем класс `Polygon` (многоугольник), а затем создаем объект, представляющий многоугольник с пятью сторонами (углами). Для него можно вывести размеры сторон функцией `dispsides`.

Затем создается класс `Square` (квадрат), производный от `Polygon`. В данном примере сразу же после создания объекта пользователь должен задать размер стороны квадрата.

Когда для объекта вызывается метод `find_area`, он использует данные, переданные пользователем, и выводит площадь квадрата. В будущем вы можете создать другой класс, производный от `Polygon`, и от вас потребуется лишь создать метод для вычисления его площади.

**ВЫВОД:**

```
side 1
side 2
side 3
side 4
side 5
Side of the square: 15
The area of the square is 225
```

При достаточно хорошем владении средствами ООП вы сможете создавать классы и объекты, которые взаимодействуют друг с другом, и строить программы, которые используют разные компоненты и решают различные задачи. Чтобы больше узнать об объектно-ориентированном программировании, попробуйте просмотреть проекты с открытым кодом в GitHub.

# Глава 10

# Операции с файлами в Python

Python использует переменные для хранения данных — как статических, так и динамических. Хотя переменные хорошо подходят для хранения данных во время выполнения программы, они могут создать проблемы, если данные, содержащие конфиденциальную информацию, должны использоваться многократно. Переменные могут уничтожаться для освобождения памяти, что вряд ли устроит пользователей, которые желают сохранить свои данные или использовать их повторно для других целей. Чтобы программисты могли взаимодействовать с данными независимо от их размера и формата, Python поддерживает операции с файлами. Каждый Python-разработчик должен понимать, как работают операции с файлами, и уметь применять их в своих программах.

# Файлы и пути к файлам

При работе с файлами Python-разработчики обычно используют два параметра: имя файла, необходимое для удобства обращения, и путь к файлу, описывающий его местоположение. Например, example.pdf — имя файла, а C:/users/downloads/example.pdf — путь к файлу. В имени файла example.pdf часть .pdf называется расширением. Для работы с файлами операционные системы обычно используют эффективную систему управления файлами.

**ПРИМЕЧАНИЕ**

Чтобы знать о различных методах работы с файлами, необходимо хотя бы в общих чертах познакомиться с менеджерами файлов, которые используются в разных операционных системах. Например, в операционной системе Windows имеется Explorer (Проводник), а в системах Mac для работы с файлами существует программа Finder. Независимо от того, какую операционную систему и менеджер файлов вы используете, файлы обычно размещаются в иерархической структуре с корневым каталогом и его подкаталогами.

## Иерархическая структура файлов

Python-разработчик должен знать, как указывать полный путь к местоположению файла. Обычно путь записывается в иерархическом виде, чтобы по нему можно было определить каталог и подкаталоги.

Например, в пути C://users/sample/example.pdf C: — корневой каталог системы, а sample и users называются подкаталогами этого корневого каталога. Так как в разных каталогах могут находиться файлы с одинаковыми именами, при выполнении операции для определения местоположения файла необходимо использовать полный путь.

> **ПРИМЕЧАНИЕ**
>
> Важно учитывать, что в файловых системах Windows корневые каталоги отделяются от подкаталогов символом \ (обратная косая черта). В других операционных системах (в частности, в Mac и Linux) для разделения корневых каталогов и подкаталогов используется символ / (косая черта).

Если вам не хочется по отдельности обрабатывать символы / и \ при вводе кода в терминале, воспользуйтесь функцией `os.path.join`.

**ПРОГРАММНЫЙ КОД:**

```
import os
print(os.path.join('D', 'first', 'second'))
```

**ВЫВОД:**

```
D\first\second
```

## Определение текущего рабочего каталога

При написании сложного кода программисту часто приходится взаимодействовать с разными файлами, находящимися в одном каталоге. Чтобы было удобно работать с файлами из одного каталога, можно воспользоваться функцией `os.getcwd()`.

**ПРИМЕР:**

```
print(os.getcwd())
D:\linux\sampleiles\python
```

В выходных данных содержится полный путь к текущему рабочему каталогу. Чтобы вывести список файлов, находящихся в текущем каталоге, можно воспользоваться командами операционной системы (например, `ls`).

## Создание новых каталогов

Часто при написании программ приходится создавать файлы в разных каталогах. Например, файл сохранения для компьютерной игры автоматически генерируется программой без вмешательства пользователя. Каждый Python-разработчик должен знать, как написать код для создания нового каталога. Для этого в Python существует функция `os.makedirs()`.

**ПРОГРАММНЫЙ КОД:**

```
import os
os.makedirs('D:/user1/python/sample')
```

В этом примере сначала импортируется модуль, в котором хранится код упомянутой выше системной функции. На следующем шаге мы вызываем функцию `makedirs()`, передавая ей в качестве параметра путь к желаемому месту хранения файла. `sample` — новый каталог, который создается в каталоге `python`. Чтобы убедиться в том, что он был создан успешно, откройте менеджер файлов или введите команду `cd` в командной строке.

**ПРИМЕЧАНИЕ**

Следите за тем, чтобы при создании нового каталога указывался абсолютный (полный) путь.

# Управляющие функции

Файловая система весьма сложна, а для работы с ней необходимо знать множество встроенных функций. Программист Python может легко открывать/закрывать файлы и оперировать ими прямо из IDE или терминала. Python поддерживает как текстовые файлы (например, с расширениями .txt, .csv, .html), так и двоичные. К категории двоичных файлов относятся изображения, аудио- и видеофайлы. Если вам потребуется открыть или обработать файлы таких типов, как pdf и jpg, для них необходимо установить сторонние библиотеки.

Чтобы вы быстрее поняли основные концепции работы с файлами, начнем с создания файла example.txt с полным путем D:/users/python/example.txt. Вы можете использовать любой другой путь. На примере этого текстового файла разберем такие функции, как `open()`, `close()`, `write()` и `read()`.

Допустим, файл example.txt содержит текст, приведенный ниже.

**СОДЕРЖИМОЕ ФАЙЛА:**

```
This is a Python file manipulation sample sheet.
```

## Как открыть файл функцией open()

Открыть файл при помощи функции Python совсем не сложно. Вам нужно лишь знать абсолютный путь к файлу и уметь пользоваться функцией `open()`.

**ПРОГРАММНЫЙ КОД:**

```
# Функция открывает файл из терминала или IDE
file_management = open('D:/users/python/example.
➥ txt')
```

В этом примере функция `open()` используется с параметром, в котором передается путь к открываемому файлу. Когда файл будет открыт, интерпретатор Python сможет читать и записывать в него данные.

### Как работает open()

Обнаружив функцию `open()`, интерпретатор создает новый объект файла, и все изменения, выполненные на этой фазе, необходимо сохранить, чтобы они отразились в исходном файле. Если файл не будет сохранен, то все внесенные изменения будут потеряны.

## Как читать файлы методом read()

После того как интерпретатор Python откроет файл функцией `open()`, он создает новый объект. После этого интерпретатор может легко прочитать все содержимое файла с помощью метода `read()`.

**ПРОГРАММНЫЙ КОД:**

```
file_management = open('D:/users/python/example.
➥ txt')
# Метод read() читает все содержимое файла
reading = file_management.read()
print(reading)
```

**ВЫВОД:**

```
This is a Python file manipulation sample sheet.
```

В этом примере мы использовали метод `read()` и сохранили все данные, прочитанные из файла, в новую переменную с именем `reading`. Информацию из файлов можно сохранять в виде строки, списка, кортежа, словаря и т. д., в зависимости от сложности файла, с которым вы работаете.

Метод `read()` читает все содержимое файла, однако для чтения данных по строкам удобнее воспользоваться методом `readlines()`. Разберем его на простом примере.

Создайте в рабочем каталоге новый файл с именем new_file.txt. Введите и сохраните в нем несколько строк произвольного текста.

**NEW_FILE.TXT:**

```
This is a sample document.
We are just creating lines.
We will use this data to manipulate text.
Python interpreter is efficient.
enough to make this possible
```

Теперь вызовите метод `readlines()` из терминала.

**ПРОГРАММНЫЙ КОД:**

```
# Переменная для открытия нового файла с заданным
именем
adv = open('new_file.txt')
print(adv.readlines())
```

**ВЫВОД:**

```
['This is a sample document.\n', 'We are just ➥
creating lines.\n', 'We will use this data to ➥
manipulate text.\n', 'Python interpreter is
efficient.\n', 'enough to make this possible\n']
```

В выводе содержатся все строки файла, завершающиеся символом новой строки `\n`.

## Как записывать данные методом write()

Программист Python может также записать новые данные в любой файл при помощи метода `write()`. Метод `write()` в целом похож на функцию `print()`, которая используется для вывода на экран, только она «выводит» данные в заданный файл.

Программисты могут открыть файл в режиме записи с помощью метода `open()`. Потребуется лишь передать интерпретатору специальный аргумент, чтобы он понял, что файл нужно открыть в режиме добавления новых данных.

Завершив запись данных в файл, закройте его методом `close()`. Обновленный файл будет автоматически сохранен в прежнем месте.

**ПРОГРАММНЫЙ КОД:**

```
# Файл открывается в режиме записи
example = open('file.txt', 'w')
example.write('This is how you need to open write
➥  mode\n')
```

Программа не выводит текст на экран, а сразу сохраняет его в файле.

Также можно использовать аргумент `'a'` для присоединения текста.

**ПРИМЕР:**

```
# Файл открывается в режиме записи
example = open('file.txt', 'a')
# Текст добавляются к указанному файлу
example.write('This is new version')
example.close()
```

Чтобы проверить, был ли текст присоединен к файлу, можно воспользоваться методом `read`.

```
example = open('file.txt')
txt = example.read()
print(sample)
```

**ВЫВОД:**

```
This is how you need to open write mode
This is new version
```

## Копирование файлов и каталогов

Обычно файлы и каталоги копируются в файловом менеджере, например в Проводнике Windows или Mac Finder. Однако некоторые Python-разработчики используют встроенную библиотеку shutil для создания программных компонентов, которые могут использоваться для быстрого копирования, перемещения и удаления файлов.

Чтобы использовать функции, включенные в библиотеку shutil, сначала ее необходимо импортировать.

**СИНТАКСИС:**

```
import shutil
```

Чтобы скопировать файлы или каталоги из одного места в другое, можно просто воспользоваться методом `shutil.copy()`. Этот метод обычно получает два параметра: путь к источнику и путь к месту назначения.

**ПРИМЕР:**

```
shutil.copy('C:\user1\python\sample.txt',
➥'C:\user2\python')
```

В этом примере файл с именем sample.txt, находящийся в подкаталоге python каталога user1, копируется в подкаталог python каталога user2.

Если вы хотите скопировать файл в новый файл с другим именем, укажите его во втором параметре, как показано ниже.

**ПРИМЕР:**

```
shutil.copy('C:\user1\python\sample.txt',
➥'C:\user2\python\sample1.txt')
```

Все содержимое файла sample.txt будет скопировано и добавлено в файл sample1.txt. Прежние данные из принимающего файла удалятся.

## Перемещение и переименование файлов и каталогов

Перемещение файла или каталога занимает меньше времени, чем копирование, но считается более рискованным, так как после перемещения не остается резервной копии. Когда вы перемещаете файлы, они полностью удаляются из текущего каталога и переносятся в новый каталог.

Для быстрого перемещения файлов между каталогами в Python используется метод `shutil.move()`.

**ПРОГРАММНЫЙ КОД:**

```
shutil.move('C:\user1\python\sample.txt',
➥'C:\user2\python')
```

В этом примере файл sample.txt будет перемещен в другой каталог.

Если вас беспокоит возможность того, что в каталоге, в который перемещается файл, уже существует файл с тем же именем, используйте следующий синтаксис.

**ПРОГРАММНЫЙ КОД:**

```
shutil.move('C:\user1\python\sample.txt',
➥'C:\user2\python\sample_2.txt')
```

Этот пример также можно назвать простым переименованием перемещенного файла.

## Удаление файлов и каталогов

Python также предоставляет три функции для удаления файлов.

```
os.unlink(path)
```

Функция удаляет только файл, указанный в пути.

**ПРИМЕР:**

```
os.unlink('C:\user1\python\arithmetic.text')
```

Файл с именем arithmetic.txt будет удален навсегда.

```
os.rmdir(path)
```

Функция удаляет весь каталог, указанный в параметре.

**ПРИМЕР:**

```
os.rmdir('C:\user1\python')
```

Папка с именем python будет удалена навсегда.

```
shutil.rmtree(path)
```

Функция строит иерархический путь и удаляет все файлы и каталоги, входящие в эту иерархию.

**ПРИМЕР:**

```
shutil.rmtree('C:\user1')
```

Все файлы и папки в каталоге user1 будут удалены.

# Глава 11

# Обработка исключений

Любое приложение время от времени может завершить работу из-за некорректного ввода пользователя или ошибки приложения. Разработчик несет ответственность за то, чтобы сообщить пользователю причину сбоя. Если проблему исправить невозможно, приложение по крайней мере должно обнаружить сбой и отправить логи с информацией о работе приложения на сервер, чтобы программист мог найти решение позже. Предупреждение об ошибке — меньшее, что могут сделать разработчики современных приложений для улучшения пользовательского опыта.

Обработка исключений — механизм программирования, который позволяет разработчикам реализовать обработку аномальных ситуаций и уведомить пользователя о возможном решении.

Наверняка пользователям Windows знакомо уведомление «Программа не отвечает» и красный значок X. Это один из самых известных интерфейсов обработки исключений, когда-либо существовавших в любой системе.

Обработку исключений не обязательно реализовывать на высочайшем уровне, но она должна по крайней мере улучшить впечатления пользователей от вашего приложения.

Написание обработчиков исключений считается продвинутым навыком разработки на Python. Обработка исключений также помогает программистам обнаруживать ошибки и логические дефекты в программе с самых первых стадий рабочего процесса. Кроме того, это экономит немало времени в ходе тестирования и поддержки.

# Пример обработки исключений

- Зайдите под своей учетной записью в Твиттер, перейдите в свой профиль и попробуйте создать твит с изображением, размер которого превышает 24 Мбайт. Через некоторое время после начала загрузки на веб-странице или в мобильном приложении появится сообщение о невозможности загрузки из-за слишком большого размера изображения.

- Разработчики Твиттера создали интерфейс обработки исключений, который объясняет пользователю, почему его изображение не было загружено. Таким образом, обработка исключений — отличный инструмент для улучшения пользовательского опыта.

- Все известные сторонние библиотеки включают набор методов обработки исключений, которые можно импортировать для использования в своих приложениях.

Рассмотрим обработку исключений на примере ошибки деления на ноль.

Обычно результат деления на ноль не может быть определен — считается, что его значение бесконечно. Таким образом, когда пользователь вашего приложения пытается выполнить такое деление, следует предоставить обработчик ошибки `ZeroDivisionError`.

Для обработки ошибки или вывода сообщения о ней используются команды `try` и `except`.

# Как работают команды try и except

try и except — программные конструкции, которые должен знать разработчик для реализации обработки исключений. В блоке try разработчик размещает код, в котором могут возникнуть ошибки. А в блоке except содержится информация о том, что нужно делать при возникновении одной из определенных вами ошибок в ходе выполнения программы.

**ПРОГРАММНЫЙ КОД:**

```
# Блок try/except в функции
def divide64(number):
    try:
        x = 64/number
        print(x)
    except ZeroDivisionError:
        print ("Cannot divide by 0")


divide64(2)
divide64(0)
divide64(64)
```

**ВЫВОД:**

```
32.0
Cannot divide by 0
1.0
```

## Как сработал код

Мы определили блоки try и except, которые сообщают интерпретатору, где может возникнуть ошибка, и предоставили информацию, которая должна быть выведена при возникновении ошибки.

# Разновидности ошибок

В документации Python перечислено великое множество системных ошибок. В предыдущем примере при обсуждении ошибки деления на ноль был продемонстрирован наиболее прямолинейный подход. Для разных ошибок существуют разные способы решения, даже при продолжении работы программы.

Если вы будете знать некоторые системные ошибки и причины их возникновения, это поможет вам понять основные принципы отладки ваших приложений.

## Ошибки значений (ValueError)

Эти ошибки возникают при передаче функции аргументов, тип которых отличается от предполагаемого. Ошибки значений могут привести к неожиданному завершению работы программы.

### Пример

Отправка файла документа там, где разрешены только графические файлы.

## Ошибки импортирования (ImportError)

Эти ошибки возникают, когда модуль не удается напрямую импортировать в код программы. Ошибки импортирования в основном возникают из-за сбоев сетевых подключений или проблем с сетевыми менеджерами пакетов.

### Пример

Невозможность синхронизации данных в учетных записях облачных серверов из-за ошибки импортирования.

## Ошибки ОС (OSError)

Иногда проблемы возникают из-за несовместимости программного кода с версией операционной системы или из-за того, что системное ядро не может понять информацию, которую ему передает приложение. Эти ошибки достаточно часто встречаются при использовании дистрибутивов Linux. Еще одна причина — заполненный диск компьютера.

### Пример

Аварийное завершение приложения из-за того, что хост использует неподдерживаемую версию операционной системы.

## Ошибки типов (TypeError)

Обычно эти ошибки обусловлены тем, что функция или операция применяются к значению неподходящего типа.

## Ошибки имен (NameError)

Ошибка образуется при обращении пользователя или программиста к переменной или функции, не определенной в программе.

## Ошибки индексирования (IndexError)

Ошибки индексирования обычно возникают при использовании индекса, выходящего за границы созданного вами итерируемого объекта (например, списка).

# Глава 12

# Расширенные возможности

Популярность Python стремительно растет благодаря тому, что многочисленные сторонние библиотеки предоставляют разработчикам разные функциональные возможности. Достаточно лишь импортировать эти библиотеки в свой код. Они помогают разработчикам создавать реальные приложения, доступные для массового пользователя. Вы как Python-разработчик должны знать о некоторых популярных библиотеках, которые помогают создавать сложные программы без построения всей функциональности с нуля.

Исходный код многих библиотек доступен на сайтах проектов с открытым кодом, таких как GitHub или Bitbucket.

Давайте рассмотрим некоторые из таких библиотек.

# Requests

Requests — библиотека Python, которая позволяет отправлять HTTP-запросы. HTTP-запрос возвращает объект Response Object со всеми данными ответа от сервера (содержимое, код ответа, статус и т. д.). Такую информацию легко обрабатывать.

Обычно данные ответа от сервера возвращаются в формате JSON. С этим форматом разработчику не всегда удобно работать, однако библиотека Requests разбирает данные JSON и выводит содержимое в читаемой форме. Также библиотека Requests может использоваться для автоматизации сбора данных с разных сайтов.

## Установка Requests

Чтобы установить любую библиотеку Python, воспользуйтесь стандартным менеджером пакетов pip.

**ПРОГРАММНЫЙ КОД:**

```
pip install requests
```

# Scrapy

Scrapy — Python-фреймворк, разработанный специально для сбора данных. Обычно поисковые системы и динамические сайты используют автоматические скрипты для сбора и анализа содержимого веб-страниц. Scrapy помогает разработчикам создавать более продвинутые скрипты, способные на интуитивном уровне извлекать данные веб-страниц.

Установить Scrapy можно с помощью любого менеджера пакетов, в том числе pip.

**ПРОГРАММНЫЙ КОД:**

```
pip install scrapy
```

# TensorFlow

TensorFlow — популярная библиотека машинного применяемая, в частности, для создания нейронных сетей. Многие разработчики также используют TensorFlow для создания программных компонентов на основе глубокого обучения, например программ распознавания лиц или обработки естественного языка. Изначально библиотека TensorFlow разрабатывалась компанией Google для построения сложных моделей машинного обучения, но позднее команда открыла доступ к проекту разработчикам-энтузиастам, желающим внести в него свой творческий вклад.

TensorFlow можно установить с помощью любого менеджера пакетов, включая pip.

**ПРОГРАММНЫЙ КОД:**

```
pip install tensorflow
```

# scikit-learn

scikit-learn — еще одна популярная библиотека для создания моделей машинного обучения. Многие разработчики также используют ее для создания аналитических приложений. scikit-learn упрощает реализацию сложных моделей машинного обучения, таких как случайный лес, метод опорных векторов, кластеризация и алгоритм k-средних.

scikit-learn также поддерживает нетривиальные алгоритмы нейронных сетей, используемые в научных исследованиях (например, в разработке генетических алгоритмов). Ее главное преимущество заключается в возможности оценки производительности моделей.

scikit-learn можно установить с помощью любого менеджера пакетов, включая pip.

**ПРОГРАММНЫЙ КОД:**

```
pip install scikit-learn
```

# Pandas

Pandas — одна из самых популярных сторонних библиотек для анализа данных. Хотя язык R популярнее Python в этой области, Pandas остается достойным вариантом для разработчиков, занимающихся созданием нетривиальных расширенных алгоритмов анализа данных на Python. Pandas позволяет легко экспортировать и импортировать большие наборы данных в разных входных форматах, включая JSON, CSV и Excel. Pandas поддерживает такие высокоуровневые средства анализа данных, как очистка, преобразование и группировка данных с более высокой точностью, чем у других библиотек, доступных для разработчиков.

Для установки Pandas в локальной системе используйте менеджер пакетов, например pip.

**ПРОГРАММНЫЙ КОД:**

```
pip install pandas
```

# Pygame

Python также используется для разработки игр. Pygame считается одним из самых популярных сторонних игровых фреймворков среди независимых разработчиков по всему миру. Pygame предоставляет в распоряжение разработчика мультимедийные и физические библиотеки для создания 2D- и 3D-игр. Pygame также предоставляет компоненты для звуковых устройств, клавиатуры, мыши, геймпада и акселерометра, для создания интерактивных игр.

Многие разработчики, пользующиеся Pygame, разрабатывают игры для смартфонов Android и планшетов, потому что фреймворк SDL, применяемый в Pygame, хорошо адаптируется к этим устройствам.

Для установки Pygame в локальной системе используйте менеджер пакетов, например pip.

**ПРОГРАММНЫЙ КОД:**

```
pip install pygame
```

# Beautiful Soup

Beautiful Soup — одна из самых популярных библиотек Python, предназначенных для автоматического извлечения данных из HTML- и XML-файлов. Beautiful Soup строит эффективное дерево разбора для навигации по структурам данных, что позволяет упорядочить извлеченную информацию.

При извлечении данных Beautiful Soup успешно распознает элементы HTML 5. Некоторые сторонние программные продукты, такие как Ahrefs, используют Beautiful Soup для своих главных средств, например инструмента определения ключевых слов (Keywords Explorer), которому часто приходится извлекать данные из миллиардов страниц, доступных в интернете.

Для установки Beautiful Soup в локальной системе используйте менеджер пакетов, например pip.

**ПРОГРАММНЫЙ КОД:**

```
pip install beautifulsoup4
```

# Pillow

Pillow — одна из многочисленных библиотек Python, которая предоставляет удобные средства для работы с графикой. Улучшение качества графики необходимо во многих компьютерных областях, а Pillow в своей работе использует унаследованные проекты PIL — более совершенной библиотеки для работы с графикой, написанной на языке C.

Библиотека Pillow появилась как ответвление проекта PIL, работа над которым была прекращена. Pillow предоставляет средства для работы с разными графическими форматами, такими как jpeg, gif, ttf и png. С помощью встроенных методов Pillow можно выполнять разные операции редактирования, включая обрезку, повороты, изменение размеров и смену фильтров.

Для установки Pillow в локальной системе используйте менеджер пакетов, например pip.

**ПРОГРАММНЫЙ КОД:**

```
pip install Pillow
```

# Matplotlib

Matplotlib — знаменитая библиотека Python, предназначенная для создания статических, анимированных и интерактивных графиков. Наряду с Matplotlib используется библиотека SciPy. Она позволяет интегрировать высокоуровневые математические расчеты в Python-код: алгоритмы оптимизации, интегрирования и интерполяции, алгебраические и дифференциальные уравнения, решение статистических и многих других задач. Также в анализе данных используется библиотека NumPy, предназначенная для создания многомерных массивов и для решения реальных научных задач. Многие ученые и специалисты по анализу данных используют эти библиотеки в рабочем процессе.

Matplotlib строит красивые графики на основании данных, что позволяет лучше понять распределение данных, определить тенденции, аномалии и выбросы значений. Если SciPy и NumPy в большей степени ориентируются на технические и научные вычисления, то Matplotlib уделяет основное внимание содержательному упорядочению данных и визуализации.

Для установки Matplotlib в локальной системе используйте менеджер пакетов, например pip.

**ПРОГРАММНЫЙ КОД:**

```
pip install matplotlib
```

Для работы некоторых сложных функций Matplotlib необходимо установить SciPy и NumPy.

**ПРОГРАММНЫЙ КОД:**

```
pip install scipy
pip install numpy
```

# Twisted

Чтобы создавать Python-приложения, связанные с веб-технологиями, разработчик должен понимать сетевые концепции. Хотя стандартная библиотека Python содержит достаточно ресурсов и методов для написания эффективного сетевого кода, всегда желательно использовать такие библиотеки, как Twisted, для реализации сложной функциональности с меньшим объемом кода. Twisted позволяет работать с различными сетевыми протоколами (такими как TCP, UDP и HTTP) с минимальными усилиями со стороны разработчика. Некоторые веб-сайты, такие как Twitch, используют Twisted как стандартную библиотеку сетевых компонентов.

Twisted можно установить с помощью любого менеджера пакетов, например pip.

**ПРОГРАММНЫЙ КОД:**

```
pip install twisted
```

# GitHub

Репозиторий Github важен для программистов, так как он упрощает сотрудничество в командах даже в условиях удаленной работы. В основе GitHub лежит репозиторий Git, который основан на одноранговой модели; таким образом, ваши изменения в коде отразятся на компьютерах ваших коллег, как только они подключатся к интернету.

GitHub полностью бесплатный. Ваш проект может быть приватным или открытым. Обратиться к приватному репозиторию может только пользователь GitHub, которому вы предоставили доступ, например участники вашей команды. Приватные репозитории используют алгоритмы шифрования для защиты данных. Открытый репозиторий, в свою очередь, доступен любому пользователю GitHub.

## Почему GitHub так важен для Python-разработчиков

В какой бы компьютерной отрасли вы ни работали, при создании проектов вам придется использовать сторонние библиотеки и фреймворки из GitHub. Используйте GitHub напрямую либо обратитесь к сторонним клиентам, которые помогут вам взаимодействовать с локальными репозиториями.

GitHub и все клиенты, поддерживающие Git, используют зависимости для простой синхронизации библиотек и модулей в коде. Для фиксации изменений кода в рабочем каталоге используется команда `commit`. Сохраненные и отправленные на сервер изменения смогут видеть другие

участники команды. Чтобы создать новый репозиторий, необходимо ввести следующую команду в консоли.

**КОМАНДА PYTHON:**

```
$ git init
```

После ввода команды в консоли в текущем рабочем каталоге будет создан новый проект (подкаталог `.git`), который содержит все необходимые метаданные Git для нового репозитория. Теперь вы сможете создать каталоги и файлы для своего проекта и делать ревизии проекта.

Если вы хотите получить информацию о состоянии проекта и о внесенных изменениях, введите следующую команду в консоли:

```
$ git status
```

Для того чтобы добавить все созданные в локальном репозитории файлы к проекту, выполните следующую команду:

```
$ git add
```

Затем зафиксируйте изменения командой `git commit` и отправьте их на сервер с помощью `git push`.

После этих подготовительных операций можно переходить к созданию собственного проекта с открытым кодом, который поможет вашим коллегам-программистам в их работе.

# Менеджер пакетов pip

Все операционные системы предоставляют приложения для своих пользователей. Python не операционная система, а всего лишь интерпретатор, который умеет выполнять программы, написанные на языке Python. Программу, написанную на другом языке, интерпретатор Python выполнить не сможет.

Существует великое множество бесплатных и коммерческих программ Python, которые можно загрузить из разных источников. Простой поиск в Google принесет вам тысячи результатов с программами, написанными для области вашего интереса. Тем не менее, если вы захотите установить найденную программу в своей системе, вам понадобятся хотя бы минимальные знания об исполняемых файлах.

Чтобы помочь программистам в установке нужных программ, Python предоставляет менеджеры пакетов. С их помощью разработчик может загрузить пакет в свою операционную систему и немедленно выполнить его. Хотя существует много сторонних менеджеров пакетов для Python, самым популярным остается pip — этот инструмент должен быть знаком каждому Python-разработчику.

## Что можно сделать с помощью pip

- Установить новые пакеты и зависимости.
- Найти на серверах pip каталог всех доступных пакетов для Python.

- Ознакомиться со списком требований перед установкой программы.
- Удалить пакеты и зависимости, которые стали ненужными.

Прежде всего необходимо проверить, установлен ли pip в вашей системе. Обычно pip устанавливается вместе с пакетом Python.

**КОМАНДА В ТЕРМИНАЛЕ:**

```
$ pip --version
```

Если команда выведет информацию о версии pip, значит, менеджер пакетов установлен в системе. Если нет — вероятно, вам придется загрузить его с официального веб-сайта и установить вручную.

## Как установить пакет

Для установки пакетов всегда используется один и тот же синтаксис команды:

```
$ pip install ИмяПрограммы
```

Например, если вы хотите установить в системе пакет TensorFlow, можно воспользоваться следующей командой:

```
$ pip install tensorflow
```

Если вы хотите проверить метаданные устанавленного пакета, введите следующую команду:

```
$ pip show tensorflow
```

В выходных данных будет выведена обширная информация, включая имя автора, название пакета и его местонахождение.

Чтобы удалить пакеты, установленные в системе, при помощи менеджера пакетов pip, введите следующую команду.

**СИНТАКСИС:**

```
$ pip uninstall ИмяПакета
```

Например, если вы хотите удалить пакет TensorFlow, который был установлен ранее, введите команду:

```
$ pip uninstall tensorflow
```

# Виртуальная среда

Обычно при установке пакета также устанавливается набор зависимостей. Иногда эти зависимости могут конфликтовать с зависимостями других продуктов, тогда пакет не установится. Чтобы упростить разработку независимых проектов, можно создать изолированную виртуальную среду при помощи пакета virtualenv.

Сначала необходимо установить virtualenv при помощи менеджера пакетов pip.

**КОМАНДА В ТЕРМИНАЛЕ:**

```
$ pip install virtualenv
```

После того как пакет будет установлен, введите следующую команду для создания нового каталога, использующего виртуальную машину:

```
$ virtualenv -p python3 sample
```

В результате будет создан каталог `sample` с некоторыми подкаталогами, например `bin`, `lib` и `include`.

С этого момента все файлы, программы и пакеты, которые вы устанавливаете в терминале, будут сохраняться в новом каталоге; тем самым предотвращаются конфликты со встроенными зависимостями или пакетами, существующими в вашей системе.

Однако сначала необходимо активировать виртуальную машину одной из следующих команд.

В Unix или MacOS:

```
$ source sample/bin/activate
```

В Windows:

```
$ source sample\Scripts\activate.bat
```

Когда все пакеты будут установлены, просто деактивируйте виртуальную среду следующей командой.

**КОМАНДА В ТЕРМИНАЛЕ:**

```
(sample) $ deactivate
```

# Модуль sys

Python-разработчик должен хорошо понимать, как работает интерпретатор Python. Интерпретатор обычно разбирает все переменные, литералы, методы, встречающиеся в коде, и выполняет программу с проверкой ошибок синтаксиса, типа и индексирования. Разработчик достаточно часто проверяет, как функционирует интерпретатор, и хранит конфиденциальную информацию, необходимую для использования конкретного ПО.

Python позволяет разработчику легко получить эту информацию при помощи модуля sys.

```
import sys
```

После выполнения этой команды вы сможете вызывать все методы, входящие в библиотеку sys.

- `path`

  Этот метод из библиотеки sys возвращает путь установки интерпретатора Python в вашей системе:

  ```
  print(sys.path)
  ```

- `argv`

  Метод выводит список всех аргументов командной строки, передаваемых скрипту:

  ```
  print(sys.argv)
  ```

- `copyright`

  Метод выводит информацию об авторских правах для интерпретатора Python или другого программного продукта:

  ```
  print(sys.copyright)
  ```

- `getrefcount`

  Метод сообщает о количестве ссылок на объект, например на переменную:

  ```
  print(sys.getrefcount(variable))
  ```

# Модульное тестирование

Каждый программист перед выпуском продукта должен убедиться в том, что программа соответствует всем рекомендациям Python. Даже если вся программная логика теоретически верна, в будущем все равно могут возникнуть проблемы из-за недочетов в коде. Подобных критических ситуаций следует избегать, чтобы улучшить впечатление пользователя от работы с продуктом.

Python предоставляет программистам возможность тестировать свой код с помощью фреймворков модульного тестирования. Фреймворк unittest устанавливается по умолчанию, чтобы программисты могли создавать тестовые условия для своих программ.

## Как работают модульные тесты

Многие программисты впадают в ступор при тестировании кода, потому что в документации Python нет конкретного набора правил проведения модульных тестов. Тем не менее опытные программисты всегда подчеркивают, что тестирование лучше начать с методов, а затем распространить его на другие программные компоненты.

- Эта методология может применяться для тестирования любых частей программы.

- Тестируемый код можно легко передать другим разработчикам. Вместе с ним передаются все ошибки построения и времени выполнения.

- Разработчик может объединить группу тестов в коллекцию, а затем вручную организовать и сопровождать их.

- Программист может установить сторонние фреймворки, чтобы расширить возможности модульного тестирования.

# Заключение

Прежде всего поздравляю вас с завершением вводного курса Python! В этой книге представлены различные темы, которые помогут вам создавать качественный код для ваших проектов.

Чтобы выйти за пределы основ, изложенных в книге, необходимо постоянно практиковаться. Вы получите практический опыт только в том случае, если будете работать над проектами или принимать участие в соревнованиях по программированию.

Опытные программисты обладают некоторыми отличительными особенностями, которые помогают им добиться успеха. По сути, это привычки, которые повышают эффективность их работы. Всем начинающим важно знать эти особенности и применять их в своих рабочих процессах, чтобы повысить эффективность в какой-то области или комбинации нескольких областей.

# Полезные привычки программистов

## Уделяйте внимание основам

Каждый программист должен стремиться как можно лучше овладеть основами, то есть фундаментальными принципами программирования. При хорошем знании основ становится проще писать код для сложных задач за меньшее время. Чтобы улучшить знание основ, обязательно изучите руководство по стилю Python, ориентированное на простоту кода. Написание простого кода и соблюдение правил «Дзена Python» поможет вам улучшить знание основ.

## Разбивайте задачу

Программист должен находить решения для сложных и запутанных задач. Не все задачи можно решить применением какой-то единой логики. Иногда программист должен разбить задачу на подзадачи, чтобы решить ее с большей эффективностью. Этот подход помогает разработчикам создать программу, которая содержит меньше ошибок и требует минимальной стратегии модульного тестирования.

## Найдите свою нишу

Невозможно в совершенстве разбираться во всех компьютерных областях. Программист должен знать, какая область представляет для него наибольший интерес. Опробуйте разные компьютерные системы, чтобы понять, какая область вызывает у вас наибольший энтузиазм. Например, Python обладает достаточными возможностями для того, чтобы сделать вас веб-разработчиком, аналитиком данных или системным инженером. Не заставляйте себя изучать все понемногу, а сосредоточьтесь на изучении конкретной области и достижении в ней совершенства.

## Ошибки бывают полезными

Ошибки бывают весьма неприятными, особенно для начинающих. Каждый раз, когда вы сталкиваетесь с ошибкой, попробуйте скопировать информацию из трассировки и поискать описание в Google. Возможно, вы найдете несколько решений проблемы; самостоятельное решение поможет вам лучше понять основы Python.

## Изучайте алгоритмы

Каждый программист Python должен изучать алгоритмы (например, алгоритмы сортировки и поиска), чтобы научиться лучше реализовывать программную логику. Базовое понимание математических концепций также поможет вам подходить к решению сложных задач на интуитивном уровне. Хотя решения, выбираемые программистами-теоретиками, обычно отличаются от решений

разработчиков-практиков, понимание теоретического подхода к решению задач поможет вам преодолеть препятствия, с которыми вы сталкиваетесь в ходе разработки.

На языке Python можно реализовать алгоритмы графов, бинарного поиска и сложные структуры данных (такие как стеки и очереди). Я рекомендую пользоваться такими веб-сайтами, как LeetCode, чтобы ознакомиться с алгоритмически-ориентированными возможностями Python.

## Начните пользоваться GitHub

GitHub — один из главных ресурсов, которые должны быть известны каждому программисту. Если вы хотите вносить какие-либо изменения в любые из репозиториев Git, вам придется знать, как работают команды `push` и `commit`. Все коммерческие организации, нанимающие разработчиков, также отдают предпочтение кандидатам с опытом работы с GitHub, чтобы они быстрее интегрировались в команду.

## Не перенапрягайтесь

Хотя это вряд ли можно отнести к технологическим советам, разработчики должны знать философию «тише едешь — дальше будешь», которой следуют их опытные коллеги. Не пытайтесь усвоить большой объем информации за один раз. На начальных стадиях карьеры важнее логическая последовательность. Это означает, что эффективнее выделять несколько часов на изучение Python ежедневно, вместо того чтобы пытаться усвоить всю информацию за несколько дней. Участие в таких программах, как «100 Days of Python», на социальных платформах

(например, Твиттер) поможет вам поднять мотивацию и обеспечить последовательный подход к изучению.

## Изучите механизмы тестирования

Прежде чем устанавливать программы в системах конечных пользователей, необходимо их тщательно протестировать. Владение приемами модульного тестирования (такими как альфа- и бета-тестирование) поможет разработчикам выпускать более качественные и работоспособные продукты, не содержащие ошибок. Используйте стратегию получения отчетов от пользователей, чтобы легко воспроизвести ошибки на своей рабочей машине и устранить их как можно быстрее. Исправление ошибок требует большого практического опыта, а иногда и привлечения экспертов. Не стесняйтесь обращаться за помощью на форумы.

## Соблюдайте баланс между работой и личной жизнью

Какую бы карьеру вы ни выбрали, очень важно выдержать правильное соотношение между работой и личной жизнью — особенно это важно для программистов. Вы должны владеть методами управления задачами и временем, чтобы с максимальной эффективностью использовать свое рабочее время. Если вы принадлежите к числу независимых специалистов, используйте для управления задачами такие приложения, как Things и Session.

Применение специальных таймеров, например Pomodoro, позволит вам лучше распоряжаться своим рабочим временем.

# Что дальше

Я рад, что вы вместе со мной сделали свои первые шаги на пути изучения Python. Программирование — интересное занятие, и, как бы быстро вы ни учились, только практика сделает из вас хорошего разработчика. Вооружившись знаниями, полученными из книги, переходите к самостоятельному созданию проектов.

Если вы не можете придумать, с какими проектами лучше поэкспериментировать, предложу несколько идей.

- Создайте систему учета книг для местной библиотеки.
- Создайте систему бронирования билетов на пригородные электрички.
- Создайте простой веб-сайт с использованием библиотеки Django.
- Создайте какую-нибудь классическую игру с использованием библиотеки Pygame.
- Сделайте парсинг данных Твиттер и создайте бот, который автоматически делает ретвиты популярных сообщений.

Желаю успехов на пути разработки!

# Благодарности

Для меня написать сотню строк кода проще, чем сотню слов в технической книге. Работа над книгой — непростое дело. В какие-то дни работа останавливалась из-за творческого тупика или из-за того, что мне не удавалось подобрать аналогию, которая была бы полезна. Работа над книгой заняла много времени... но я все же справился с задачей, потому что мне хотелось помочь людям, которые только начинают свой путь в мире программирования и порой чувствуют себя беспомощно из-за многочисленных препятствий.

Я бы никогда не довел работу над книгой до конца, если бы не моя жена Хелен, которая помогала мне на протяжении всего времени написания книги. Она подбадривала меня, когда я застревал на каком-то программном блоке, который собирался включить в книгу.

Также хочу поблагодарить одного из моих профессоров, Дэвида Тейлора, который всегда поддерживал меня во время учебы. Его наставления и энтузиазм в программировании помогли мне лучше представить, чем бы я хотел заниматься в будущем. Подход Дэвида к решению задач помог мне преодолевать препятствия как в профессии, так и в жизни.

Напоследок я хочу поблагодарить своего друга Питера, без которого я бы никогда не написал эту книгу. Питер — один из моих коллег, который сейчас работает в компании Apple. Я всегда обращался к нему, когда мне хотелось обсудить какие-то вопросы, возникавшие в процессе работы. Без него моя книга никогда бы не была дописана, и я от всей души благодарю его за то, что он помог мне отправиться в это путешествие.

И наконец, спасибо вам. Да, вам — читателям, для которых я задумал написать эту книгу. Именно вам мне хотелось

помочь, и надеюсь, у меня это получилось. Хочется верить, что книга оказалась полезной и помогла в изучении новых концепций и приемов. В конце концов, мы никогда не должны прекращать учиться, и, если бы я сказал, что с приведенным в книге материалом вы станете экспертом, это было бы неправдой. Итак, я рекомендую всегда сохранять любознательность и готовность пройти дополнительный путь, который поможет вам глубже проникнуть в суть дела.

Мне бы хотелось, чтобы вы расценивали эту книгу как отправную точку для погружения в удивительный, непрерывно развивающийся мир Python.

Если вы полагаете, что какую-то часть книги можно улучшить, я предлагаю связаться со мной напрямую по адресу andrewpark.py@gmail.com. Мне хотелось бы получать конструктивную обратную связь, чтобы я мог немедленно обновлять материал. В этом случае я с радостью пришлю вам новую версию! Спасибо за то, что дочитали до конца.

*Эндрю*