# Тема 4.2. Основы работы с Python

# Введение

В этой лекции мы рассмотрим широкий спектр тем, связанных с программированием на Python в контексте системного администрирования linux. Начнем с истории языка Python, его места в среде обитания других языков программирования, рассмотрим синтаксис и основные конструкций, чтобы заложить прочную основу для понимания его возможностей. Затем мы изучим, как работать с файловой системой, читать и записывать файлы, а также управлять путями и директориями. Мы также обсудим способы запуска и контроля процессов с помощью модуля subprocess, что позволит вам автоматизировать задачи и взаимодействовать с операционной системой. Далее, мы перейдем к объектно-ориентированному программированию, рассмотрим принципы инкапсуляции, наследования и полиморфизма, а также работу с сетевыми сокетами и чтение системных журналов. Эта лекция предоставит вам целостное представление о различных аспектах программирования на Python, помогая улучшить ваши навыки и применять их на практике системного администрирования.

# 1. История и развитие Python

Python был создан Гвидо ван Россумом. Первый релиз Python (версия 0.9.0) был выпущен в феврале 1991 года. Язык был задуман как высокоуровневый, простой в использовании и читаемый. Python быстро завоевал популярность благодаря своей простоте и мощным возможностям, что сделало его одним из самых популярных языков программирования в мире.

С января 1994 по июль 2010 года вышло несколько версий, последняя из которых стала Python 2.7. Эта версия включала многие функции из Python 3.x и получила длительную поддержку, которая продолжалась до января 2020 года.

В декабре 2008 года был выпущен Python 3.0. Это была не обратимо совместимая версия с Python 2.x, что позволило внести значительные улучшения в язык. Среди этих улучшений были: Улучшенная поддержка Unicode, изменения в синтаксисе, например, print стал функцией, Объединение long int и int типов данных.

Далее с декабря 2016 года по 2022 продолжали добавлять новые функции и улучшения производительности.

В октябре 2022 года был выпущен Python 3.11, включающий значительные улучшения производительности и новые синтаксические функции.

Python прошел долгий путь от небольшого проекта в конце 1980-х годов до одного из самых популярных языков программирования в мире. Он широко используется в веб-разработке, научных исследованиях, искусственном интеллекте, автоматизации и системном администрировании. Сообщество Python продолжает активно развивать и улучшать язык, делая его еще более мощным и удобным инструментом для разработчиков.

# 2. Python в системном администрировании

Python стал одним из самых популярных инструментов для системных администраторов по ряду причин. Вот основные преимущества использования Python в системном администрировании:

**1. Простота и читаемость кода**

* **Легкий синтаксис**: Python известен своим простым и интуитивно понятным синтаксисом, который делает код легко читаемым и поддерживаемым. Это особенно важно для системных администраторов, которые часто пишут скрипты для автоматизации задач.
* **Быстрое обучение**: Благодаря своей простоте, Python легко изучается, что позволяет администраторам быстро осваивать язык и начинать его использовать.

**2. Многообразие стандартных библиотек**

* **Стандартная библиотека**: Python поставляется с обширной стандартной библиотекой, включающей модули для работы с файлами, сетями, процессами и многими другими задачами, что позволяет быстро и эффективно решать повседневные задачи системного администрирования.
* **Сторонние библиотеки**: Существует огромное количество сторонних библиотек и модулей, доступных через PyPI (Python Package Index), которые значительно расширяют возможности языка.

**3. Кроссплатформенность**

* **Работа на различных платформах**: Python работает на всех основных операционных системах, включая Linux, Windows и macOS. Это позволяет администраторам использовать один и тот же код на различных платформах.
* **Обратная совместимость**: Python поддерживает обратную совместимость между версиями, что облегчает переход на новые версии языка.

**4. Автоматизация и скриптинг**

* **Автоматизация рутинных задач**: Python позволяет автоматизировать множество рутинных задач, таких как управление конфигурацией, мониторинг систем, резервное копирование данных и обновление программного обеспечения.
* **Планирование задач**: С помощью Python можно создавать скрипты для автоматического выполнения задач по расписанию, используя инструменты планирования, такие как cron.

**5. Сетевое программирование**

* **Работа с сетями**: Python предоставляет простые и мощные инструменты для работы с сетями, что позволяет легко создавать скрипты для управления сетевыми устройствами, автоматизации развертывания и управления сетевой инфраструктурой.
* **Сетевые библиотеки**: Стандартные библиотеки, такие как socket, позволяют решать широкий круг задач, связанных с сетевым программированием.

**6. Управление системными процессами и журналами**

* **Процессы**: Python позволяет управлять системными процессами, запускать и останавливать их, а также анализировать их состояние с помощью модуля subprocess.
* **Журналы**: С помощью Python можно легко читать и анализировать системные журналы, что упрощает диагностику и устранение проблем.

**7. Интеграция с другими инструментами**

* **Интеграция с DevOps инструментами**: Python легко интегрируется с другими инструментами управления конфигурацией и автоматизации, такими как Ansible, Puppet, SaltStack и другие.
* **API и веб-сервисы**: Python позволяет быстро и эффективно взаимодействовать с различными API и веб-сервисами, что упрощает интеграцию с внешними системами.

**8. Сообщество и поддержка**

* **Активное сообщество**: У Python большое и активное сообщество разработчиков, что обеспечивает обширную базу знаний, примеров кода и поддержку.
* **Документация**: Обширная и хорошо структурированная документация делает Python доступным и понятным даже для начинающих пользователей.

Использование Python в системном администрировании позволяет значительно упростить и автоматизировать множество задач, повысить эффективность работы и снизить количество ошибок. Благодаря своей простоте, гибкости и широким возможностям, Python стал неотъемлемым инструментом для многих системных администраторов.

Python достаточно прост для освоения, даже если у вас нет опыта в разработке: начать можно с простых скриптов или программ "Hello, World!". Это интерпретируемый язык, который позволяет писать скрипты. Он отлично справляется с задачами автоматизации, когда нужно выполнить несколько действий последовательно, особенно если это сложнее, чем может Bash.

С помощью Python можно выполнять важные операции на production-серверах. Это намного удобнее, чем делать это вручную. При правильной автоматизации вероятность ошибки значительно снижается по сравнению с ручным выполнением одной и той же операции многократно. Скрипты полезны для простых задач, когда нужен инструмент, выполняющий определенные действия.

Python подходит для многих задач. Например, он хорошо справляется с обработкой логов. Также можно автоматизировать простые задачи и запускать их с помощью cron. Например, можно написать скрипт, который каждые 5 минут проверяет Active Directory («Активный каталог») и добавляет новых пользователей на сервер.

Python также хорошо справляется с задачами оркестрации, т.е автоматизированным управлением задачами и координацией различных процессов.

Еще одна важная область применения Python — ML-Ops. Он используется как язык поддержки для специалистов по машинному обучению и как системный язык для инженеров. Python служит удобным API (набор способов и правил, по которым различные программы общаются между собой и обмениваются данными). Большое преимущество Python — его развитая экосистема, которой нет у других языков.

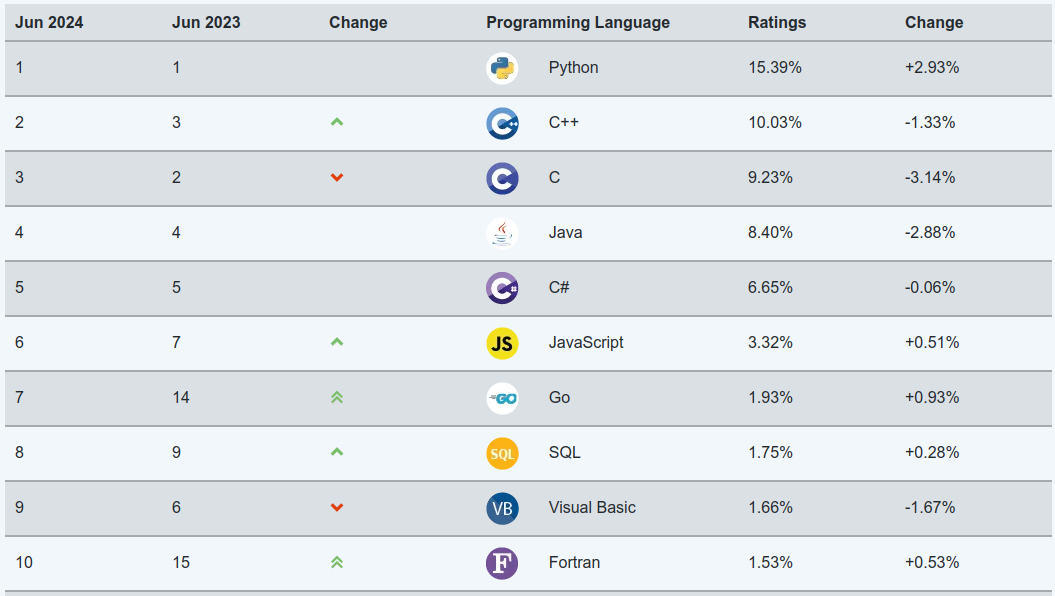
# 3. Рейтинг Python

Кратко рассмотрим место python в it индустрии на основании известных источников.

##### 3.1 Индекс TIOBE

Индекс TIOBE — это показатель популярности языков программирования, который обновляется ежемесячно. Он основан на данных, собранных из поисковых систем, таких как Google, Bing, Yahoo!, Wikipedia, Amazon, YouTube и Baidu. TIOBE измеряет количество поисковых запросов, связанных с конкретными языками программирования, что позволяет определить их популярность среди разработчиков. В настоящее время Python занимает первое место в индексе TIOBE, его популярность продолжает расти.

<https://www.tiobe.com/tiobe-index/>



##### 3.2 IEEE Spectrum

Рейтинг языков программирования IEEE Spectrum — это ежегодное исследование, которое оценивает популярность языков программирования на основе различных метрик. Рейтинг считается одним из наиболее всеобъемлющих и авторитетных, так как учитывает широкий спектр данных и мнений из различных источников. Разработчики и компании часто обращаются к этому рейтингу, чтобы понять, какие языки стоит изучать или использовать в новых проектах.

Python в 2023 и 2022 два года подряд занимал первое место в рейтинге языков программирования IEEE Spectrum. Его популярность продолжает расти благодаря широкому спектру применения, от науки о данных и машинного обучения до веб-разработки и автоматизации.

<https://skillbox.ru/media/code/ieee-spectrum-opublikoval-reyting-populyarnosti-yazykov-programmirovaniya-za-2023-god/>

##### 3.3 Опрос StackOverflow

Рейтинг языков программирования по опросу StackOverflow ежегодно составляется на основе ответов тысяч разработчиков со всего мира. Этот опрос считается одним из самых крупных и влиятельных в IT-индустрии, предоставляя глубокий взгляд на предпочтения и опыт программистов. В опросе оцениваются различные аспекты языков программирования, включая их популярность, любовь разработчиков, и желаемость изучения.

В последнем опросе Stack Overflow Python занял третье место по популярности среди языков программирования.

<https://survey.stackoverflow.co/2023/#most-popular-technologies-language>



# 4. Дистрибутивы Linux с Python

Python — это очень популярный язык программирования, и он предустановлен на многих, но не на всех дистрибутивах Linux.

* *Ubuntu: Python предустановлен, так как многие системные утилиты и пакеты зависят от него.*
* *Debian: Python также предустановлен, поскольку является важной частью системы.*
* *Fedora: Python включен по умолчанию, поскольку многие инструменты Fedora используют его.*
* *CentOS/RHEL: Python обычно установлен, так как он необходим для работы многих системных скриптов и утилит.*
* *Arch Linux: Python может быть не предустановлен в минимальной установке, но легко доступен для установки.*
* *Alpine Linux: Этот дистрибутив нацелен на минимализм, поэтому Python может не быть установлен по умолчанию, но его можно легко установить через пакетный менеджер apk.*
* *Gentoo: Python может быть не установлен в базовой системе, но его можно легко добавить при необходимости.*
* *Tiny Core Linux: Дистрибутив минималистичный, и Python обычно не включен, но его можно установить из репозиториев.*



# 5. Установка Python

Установка Python на Linux может варьироваться в зависимости от дистрибутива. Мы рассмотрим установку Python на Ubuntu.

Ubuntu обычно поставляется с предустановленной версией Python. Чтобы проверить, какая версия Python установлена, выполните команду: python3 --version

Если у вас не установлен python то можете использовать инструкции приведенные на слайде.

Проверка установленной версии Python:

python3 --version

Установка Python из официального репозитория:

sudo apt update

sudo apt install python3

Установка pip и venv:

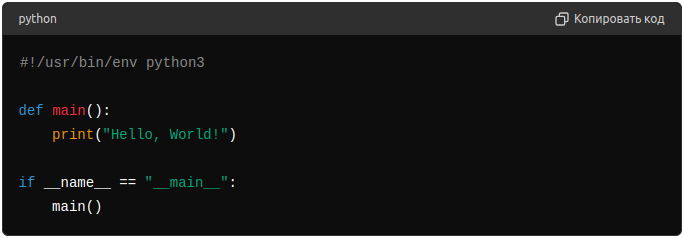
sudo apt install python3-pip python3-venv

# 6. Запуск Python скриптов

Для запуска можно использовать интерпретатор передав ему путь к файлу, например: python3 main.py где main.py это файл со скриптом на языке Python.

Альтернативно для запуска скрипта без непосредственного вызова интерпретатор можно указав шебанг в первой строке скрипта #!/usr/bin/env python3 и сделав файл исполняемым.

Например: Создайте файл со скриптом, например, hello.py.



Измените разрешения файла, чтобы он стал исполняемым.



Теперь вы можете запустить скрипт непосредственно, указав его имя.



# 7. Основы языка Python

Отступы в Python играют ключевую роль и являются обязательной частью синтаксиса языка. Они используются для определения блоков кода, таких как тела функций, циклы, условные операторы и другие конструкции. В отличие от многих других языков программирования, где для этих целей используются фигурные скобки, в Python структуру кода определяют именно отступы.

### Основные правила отступов в Python

1. **Использование пробелов и табуляций:**

* + Python позволяет использовать как пробелы, так и табуляции для отступов, но не рекомендуется смешивать их в одном проекте, так как это может привести к ошибкам. По умолчанию принято использовать пробелы.
  + Рекомендовано использовать 4 пробела для одного уровня отступа.

2. **Консистентность:**

* + Все строки кода в одном блоке должны иметь одинаковый отступ. Это позволяет Python корректно интерпретировать, какие строки кода принадлежат какому блоку.

3. **Ошибки из-за отступов:**

* + Ошибка IndentationError возникает, если отступы используются некорректно, например, если блок кода отступлен неправильно.

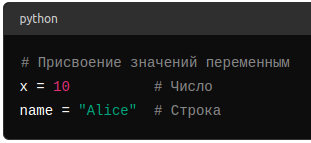


В этом примере:

* print("Hello, " + name) и блок if-else отступлены на один уровень (4 пробела) внутри функции greet.
* Строки print("Welcome back, Alice!") и print("Nice to meet you, " + name) отступлены на второй уровень (еще 4 пробела), так как они являются частью блока if-else.

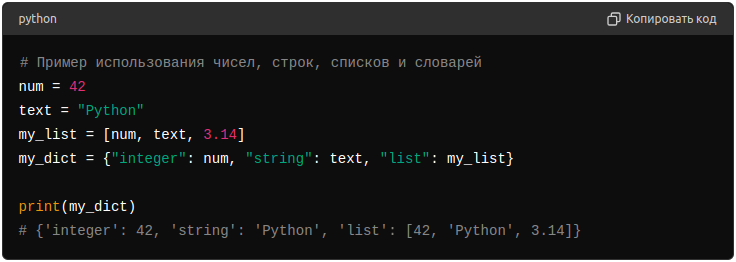
##### 6.1 Переменные

Переменные в Python используются для хранения данных. Переменная создается, когда ей присваивается значение.



##### 6.2 Типы данных

Python поддерживает несколько основных типов данных, каждый из которых предназначен для хранения и обработки различных видов информации.



6.2.1 Числа (Numbers)

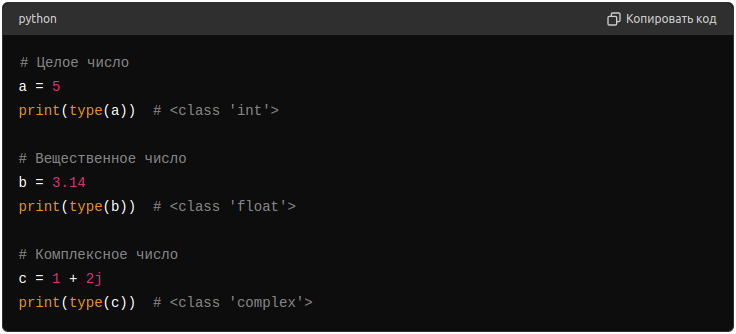
Python поддерживает три основных числовых типа данных: целые числа (int), вещественные числа (float) и комплексные числа (complex).

Целые числа представляют собой числа без дробной части, такие как 42 или -7.

Вещественные числа используются для представления чисел с плавающей запятой, например, 3.14 или -0.001.

Комплексные числа включают действительную и мнимую части, записываются в виде 1+2j, где j обозначает мнимую единицу.

Эти типы данных позволяют выполнять широкий спектр математических операций, обеспечивая гибкость и точность вычислений в Python.



*6.2.2 Строки (Strings)*

Строки в Python используются для хранения и обработки текстовой информации и заключаются в одинарные, двойные или тройные кавычки. Строки поддерживают индексацию и срезы, что позволяет легко извлекать отдельные символы или подстроки. Встроенные методы строк, такие как upper(), lower(), replace(), и split(), обеспечивают удобные инструменты для манипуляции текстом. Строки в Python являются неизменяемыми, что означает, что после создания они не могут быть изменены, однако можно создать новые строки на основе существующих. Для форматирования строк используются такие методы, как f-строки (f"{name}"), метод format(), и операторы % для старого стиля форматирования.





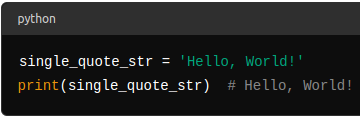
6.2.2.1 Кавычки

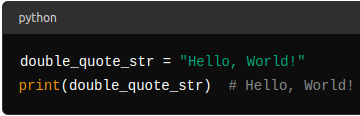
В Python кавычки используются для создания строковых литералов. Существует несколько типов кавычек, которые можно использовать в зависимости от контекста и требований к строке.

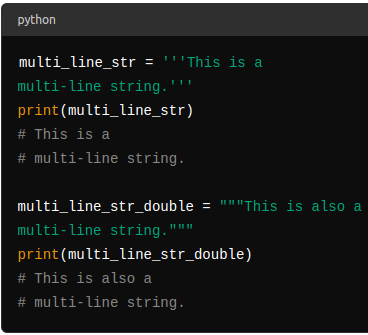
Одинарные кавычки ('): Используются для создания строковых литералов.

Двойные кавычки ("): Используются аналогично одинарным кавычкам.

Тройные кавычки (''' или """): Используются для создания многострочных строк или строк, содержащих кавычки.







6.2.2.2 Использование кавычек внутри строк

Иногда необходимо использовать кавычки внутри строк. Это можно сделать несколькими способами:

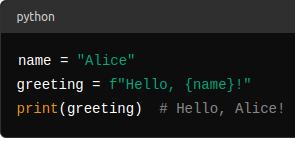
* Экранирование кавычек с помощью обратной косой черты (\)
* Использование разных типов кавычек для вложенных строк

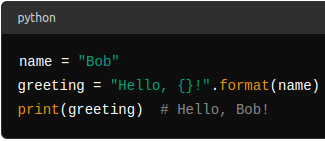
6.2.2.3 Кавычки и форматирование строк

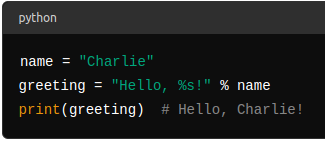
Для форматирования строк используется несколько способов:

* F-строки (f-strings): Позволяют вставлять значения переменных в строки. F-строки начинаются с буквы f перед строкой и используют фигурные скобки {} для включения выражений.
* Метод format(): Позволяет форматировать строки, вставляя значения в места, обозначенные {}.
* Оператор %: Более старый способ форматирования строк, который использует % для включения значений.

Кавычки в Python предоставляют гибкие возможности для создания и работы со строками. В зависимости от ситуации можно использовать одинарные, двойные или тройные кавычки для создания строк, включения кавычек внутрь строк, создания многострочных текстов и форматирования строк.



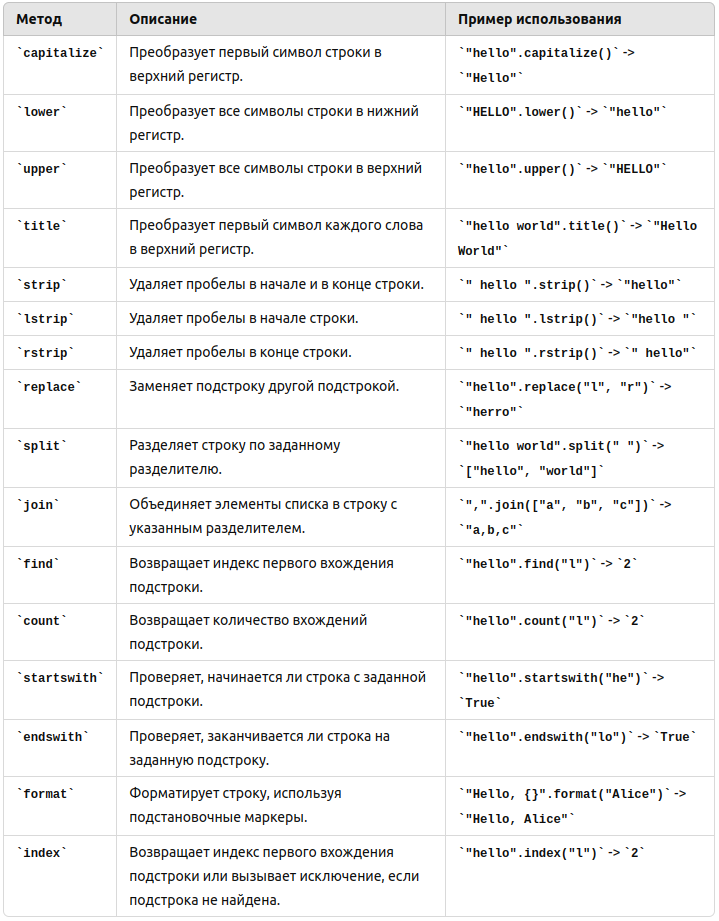




6.2.2.4 Методы для работы со строками

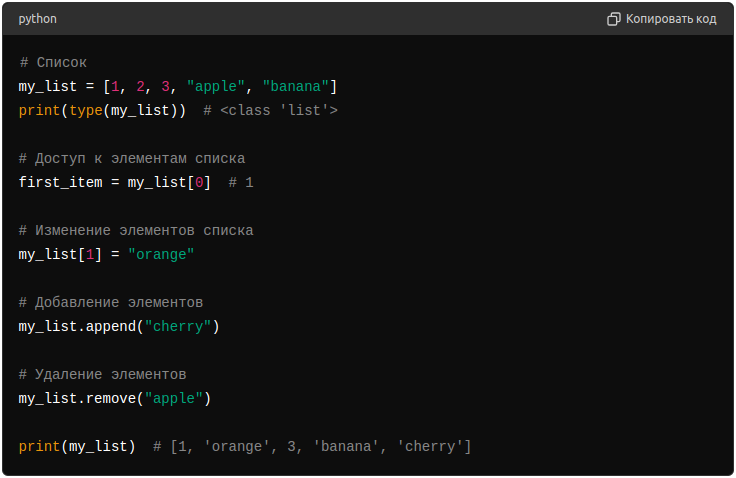
На слайде приведена таблица с методами для работы со строками.

Эти методы позволяют выполнять различные операции со строками, такие как изменение регистра, удаление пробелов, замена подстрок, разбиение строки и объединение элементов, поиск подстрок и форматирование строк.



6.2.3 Списки (Lists)

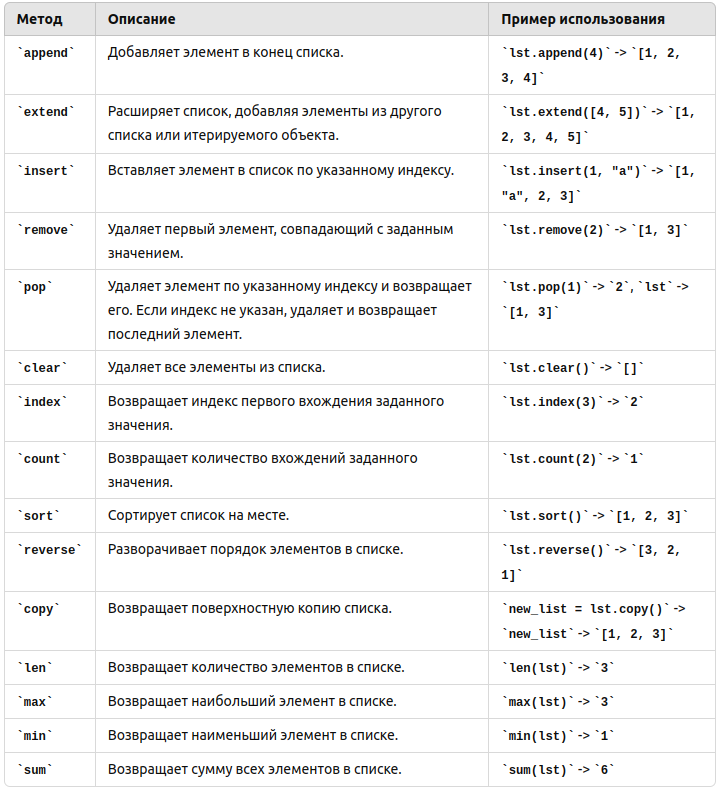
Списки в Python представляют собой упорядоченные коллекции объектов, которые могут содержать элементы разных типов, включая другие списки. Списки являются изменяемыми, что позволяет добавлять, удалять или изменять элементы после их создания. Основные методы работы со списками включают append(), remove(), pop(), и extend(), обеспечивающие гибкость при управлении элементами. Списки поддерживают индексацию и срезы, что позволяет легко извлекать и изменять подмножества элементов. Списки также могут быть отсортированы с помощью метода sort() или функции sorted(), что упрощает работу с данными.



6.2.3.1 Методы для работы со списками

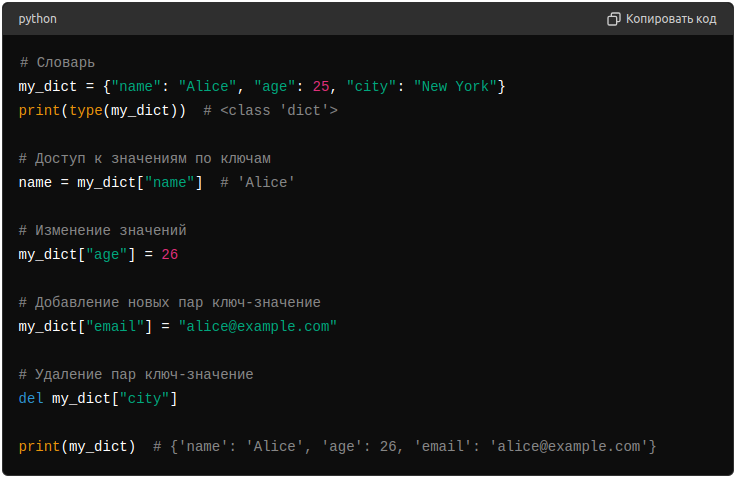
На слайде приведена таблица с методами для работы со списками.

Эти методы позволяют эффективно управлять элементами списка, выполнять операции добавления, удаления, поиска, сортировки, копирования и вычисления значений.



6.2.4 Словари (Dictionaries)

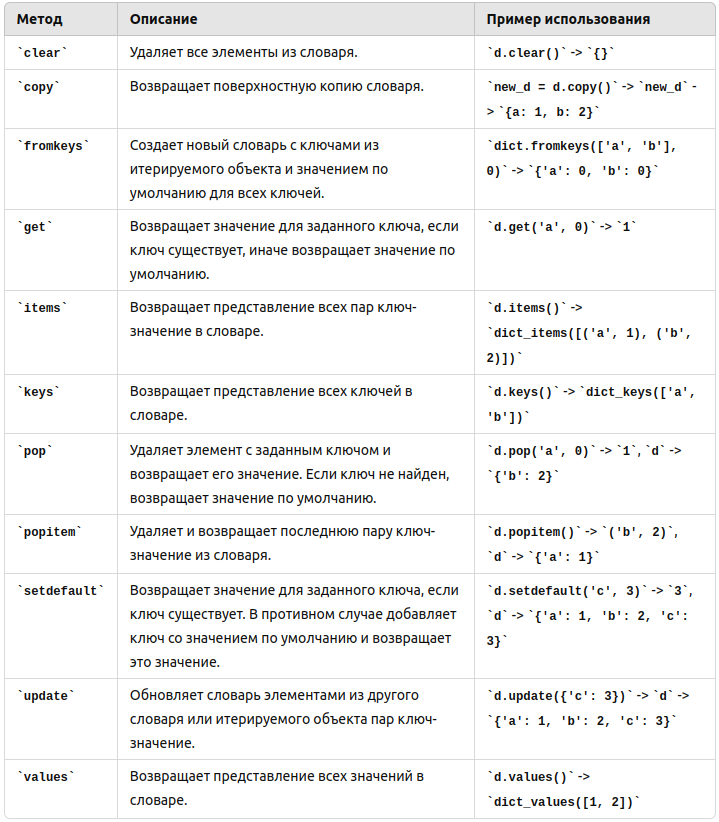
Словари (dict) в Python представляют собой неупорядоченные коллекции пар "ключ-значение", где ключи должны быть уникальными и неизменяемыми. Словари позволяют быстро находить значения по ключам, обеспечивая эффективный доступ к данным. Основные методы работы со словарями включают keys(), values(), и items(), которые возвращают ключи, значения и пары "ключ-значение" соответственно. Метод get() позволяет безопасно извлекать значения по ключу, возвращая значение по умолчанию, если ключ не найден. Словари поддерживают добавление, изменение и удаление элементов с помощью синтаксиса присваивания, методов update() и pop().



6.2.4.1 Методы для работы со словарями

На слайде приведена таблица с методами для работы со словарями.

Эти методы позволяют эффективно управлять элементами словаря, выполнять операции добавления, удаления, поиска, итерирования и обновления данных в словаре.



##### 6.3 Управляющие конструкции

Управляющие конструкции позволяют управлять потоком выполнения программы. В Python основные управляющие конструкции включают условные операторы и циклы.

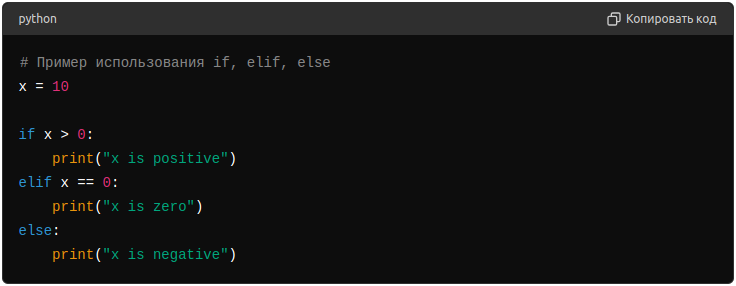
6.3.1 Условные операторы (if, elif, else)

Условные операторы позволяют выполнять различные действия в зависимости от условий.

if: Проверяет условие, и если оно истинно, выполняет блок кода.

elif: Проверяет дополнительное условие, если предыдущее условие было ложным.

else: Выполняет блок кода, если все предыдущие условия были ложными.



6.3.2 Циклы

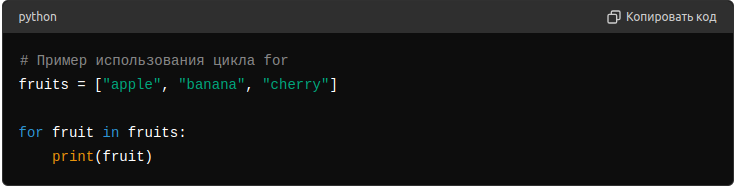
Циклы позволяют повторять выполнение блока кода несколько раз.

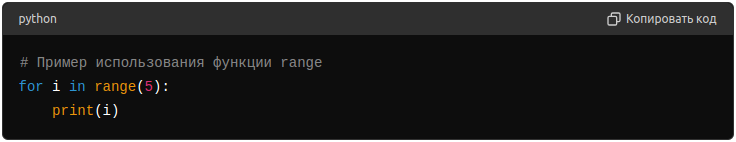
6.3.2.1 Цикл for

Цикл for итерирует по элементам последовательности (список, строка, диапазон и т.д.).

Цикл for также может использоваться с функцией range для выполнения итераций по числовому диапазону:

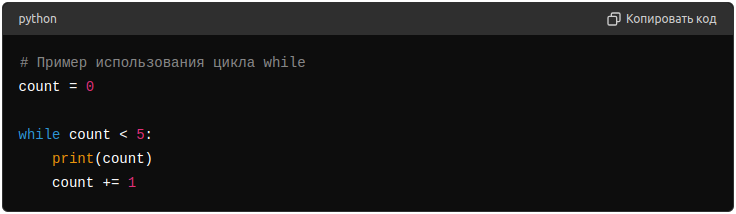
* **range(n)**: Создает последовательность чисел от 0 до n-1.
* **range(start, stop)**: Создает последовательность чисел от start до stop-1.
* **range(start, stop, step)**: Создает последовательность чисел от start до stop-1 с шагом step.





6.3.2.2 Цикл while

Цикл while выполняет блок кода, пока условие истинно.

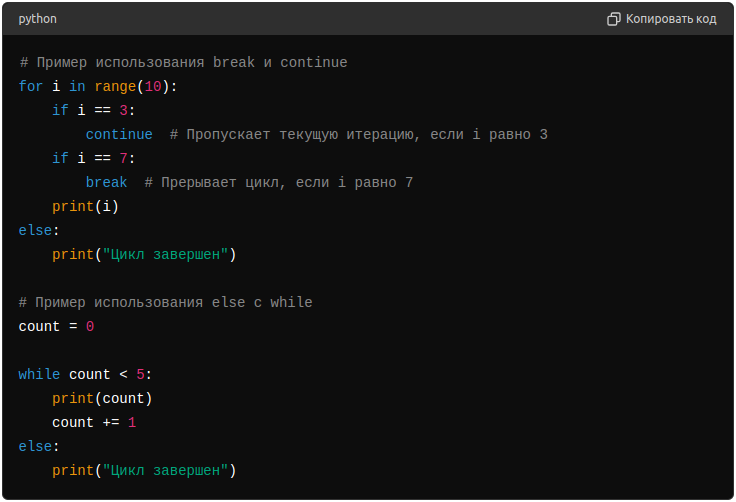


6.3.2.3 Операторы управления циклами

Python предоставляет несколько операторов для управления циклами: **break**, **continue** и **else**.

* **break**: Прерывает цикл досрочно.
* **continue**: Пропускает оставшуюся часть кода внутри цикла для текущей итерации и переходит к следующей итерации.
* **else**: Выполняет блок кода после завершения цикла, если цикл не был прерван оператором break.

Эти конструкции позволяют создавать гибкие и сложные алгоритмы, управляя потоком выполнения программы в зависимости от условий и требований.



##### 6.4 Функции

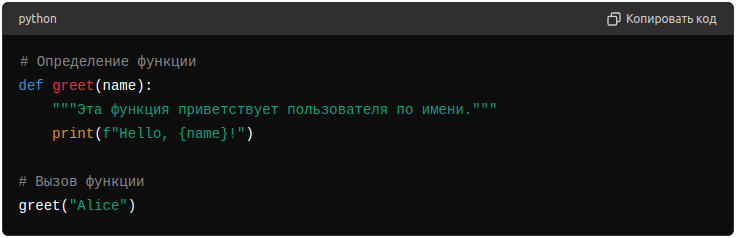
Функции в Python позволяют организовать код в блоки, которые можно многократно использовать. Они помогают структурировать программу, делая ее более читаемой и удобной для поддержки.

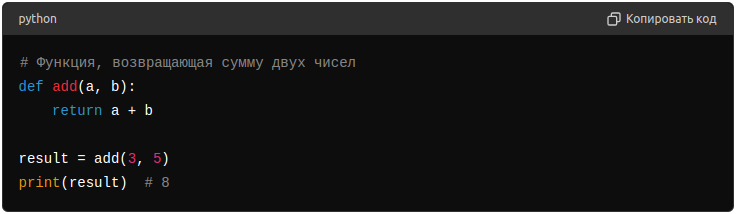
Функции определяются с помощью ключевого слова def, за которым следует имя функции, список параметров в круглых скобках и двоеточие. Тело функции начинается с новой строки и должно быть с отступом.

На слайде представлен пример работы с функциями, где:

* **name**: Параметр функции.
* **""" """**: Строка документации (docstring), описывающая функцию.
* **print()**: Оператор, выполняющий действия внутри функции.

Функции могут возвращать значения с помощью ключевого слова return.





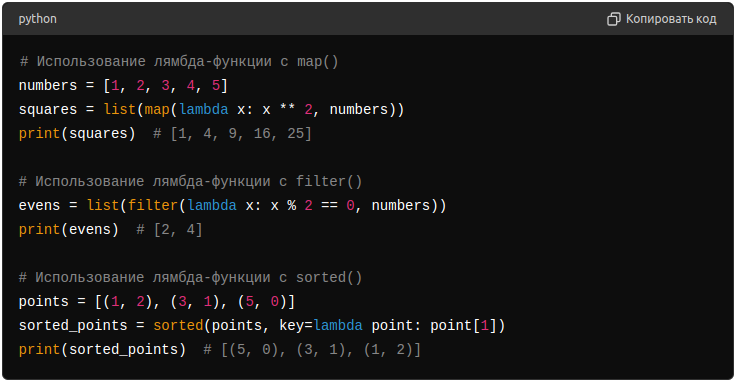
6.4.1 Лямбда-функции

Лямбда-функции — это анонимные функции, которые определяются с помощью ключевого слова lambda. Они могут иметь любое количество аргументов, но содержат только одно выражение. Лямбда-функции часто используются для создания небольших одноразовых функций.

Лямбда-функции полезны в сочетании с такими функциями, как map(), filter(), и sorted().

Лямбда-функции делают код более компактным и удобным для использования в ситуациях, когда требуется небольшая функция, которая будет использована один раз.



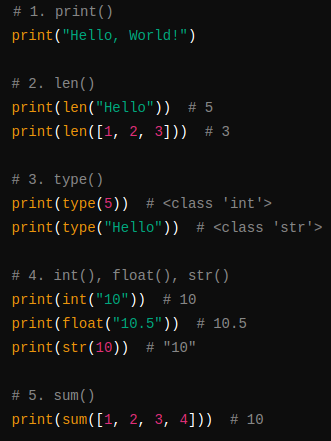


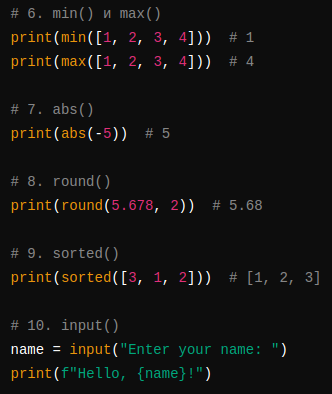
6.4.2 Встроенные функции

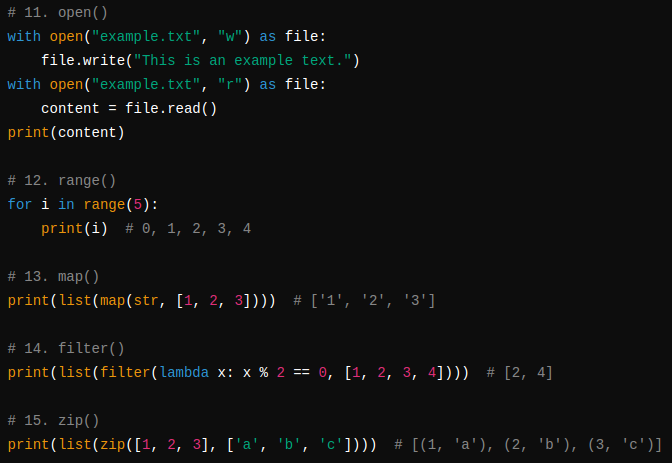
Python предоставляет широкий набор встроенных функций, которые можно использовать без необходимости их определения. Вот некоторые из наиболее часто используемых встроенных функций:

1. *print(): Выводит данные на стандартный вывод (обычно на экран).*
2. *len(): Возвращает количество элементов в объекте (например, в строке, списке или словаре).*
3. *type(): Возвращает тип объекта.*
4. *int(), float(), str(): Преобразуют объекты к целому числу, числу с плавающей запятой или строке соответственно.*
5. *sum(): Возвращает сумму элементов последовательности.*
6. *min() и max(): Возвращают минимальное и максимальное значение в последовательности соответственно.*
7. *abs(): Возвращает абсолютное значение числа.*
8. *round(): Округляет число до заданного количества десятичных знаков.*
9. *sorted(): Возвращает отсортированный список из элементов заданной последовательности.*
10. *input(): Считывает строку с ввода от пользователя.*
11. *open(): Открывает файл и возвращает соответствующий объект файла.*
12. *range(): Возвращает последовательность чисел, используемую для итерирования в циклах.*
13. *map(): Применяет функцию к каждому элементу последовательности и возвращает итератор с результатами.*
14. *filter(): Отфильтровывает элементы последовательности, для которых функция возвращает True.*
15. *zip(): Объединяет несколько последовательностей в одну, образуя пары элементов.*

Эти встроенные функции делают Python мощным и удобным инструментом для различных задач, от базовых операций до сложной обработки данных.







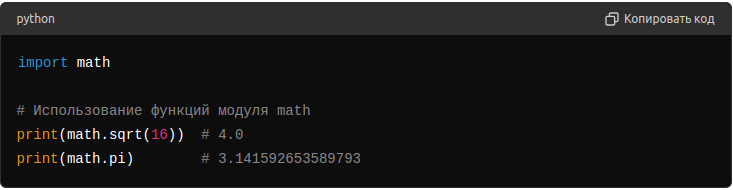
##### 6.6 Модули

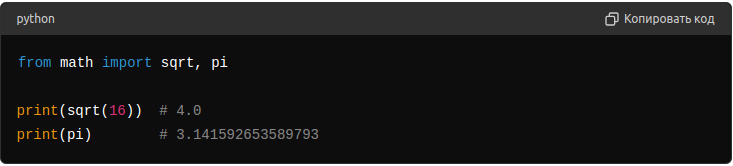
Модули в Python позволяют организовать код в логические блоки, которые можно легко использовать повторно и делиться ими с другими. Модуль в Python — это просто файл с расширением .py, который содержит определения функций, классов и переменных.

Чтобы использовать функции и классы из модуля, необходимо импортировать его с помощью ключевого слова import.

Вы можете импортировать конкретные функции или классы из модуля, чтобы не писать имя модуля каждый раз.

Чтобы сделать код более компактным или избежать конфликтов имен, вы можете импортировать модуль с псевдонимом.

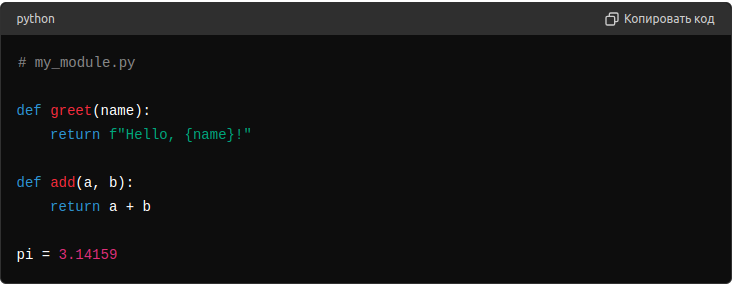






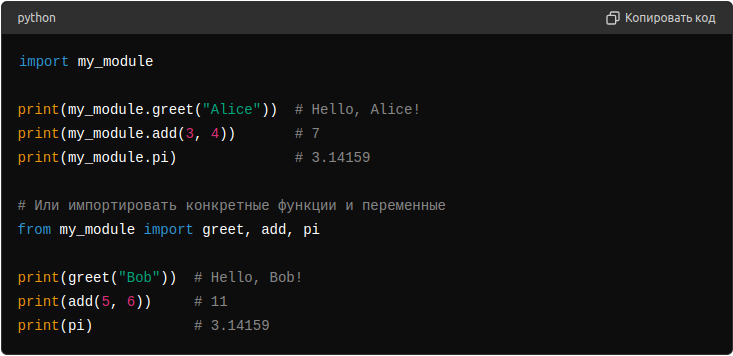
6.6.1 Создание собственного модуля

Создание собственного модуля в Python очень просто. Нужно создать файл с расширением .py и определить в нем функции, классы и переменные.



6.6.2 Использование собственного модуля

Теперь вы можете импортировать и использовать функции и переменные из вашего модуля как показано на слайде.

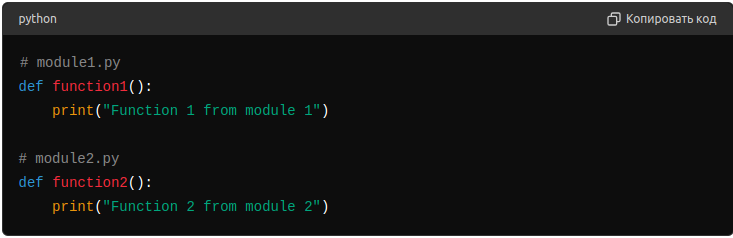


6.6.3 Пакет

Пакет — это структура, позволяющая организовать модули в иерархии каталогов. Пакет — это директория, содержащая файлы модулей и файл \_\_init\_\_.py. Структура модуля представлена на слайде.

\_\_init\_\_.py может быть пустым или содержать код инициализации пакета.

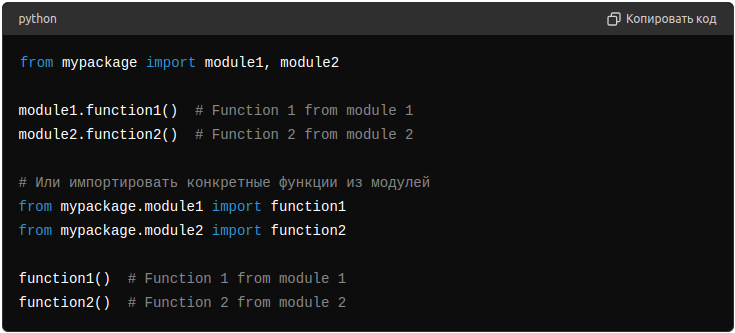




6.6.4 Использование пакета

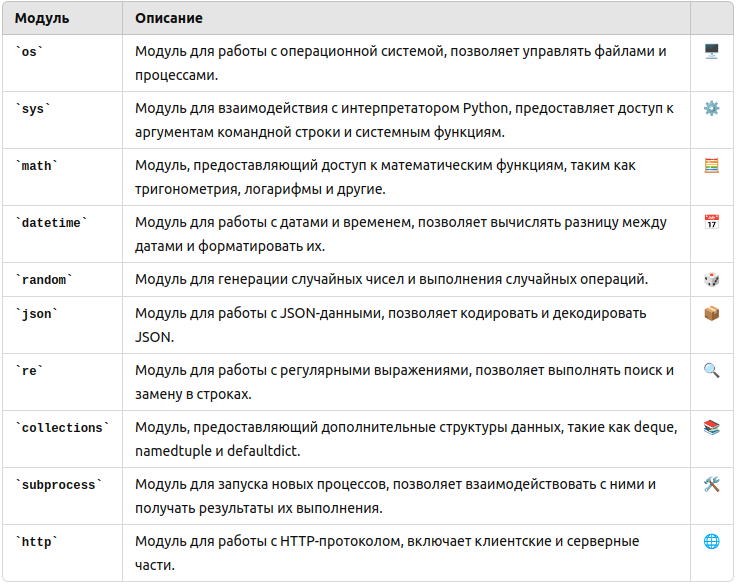
Пример использования пакета показан на слайде, где в первой строке происходит импорт модулей из пакета, а в дальнейшем происходит обращение к функциям импортированных модулей.

Python поставляется с большим количеством встроенных модулей, таких как os, sys, datetime, json, и многие другие. Вы также можете устанавливать сторонние модули с помощью менеджера пакетов pip, который мы рассмотрим далее.



6.6.5 Примеры встроенных модулей

Примеры некоторых встроенных модулей представлены в таблице на слайде.



6.6.6 Внешние пакеты

pip — это менеджер пакетов для Python, который позволяет устанавливать и управлять внешними библиотеками и модулями. Он автоматически загружает и устанавливает пакеты из Python Package Index (PyPI) и других репозиториев.

На большинстве современных дистрибутивов Python pip установлен по умолчанию. Чтобы проверить, установлен ли pip, выполните в терминале команду: pip --version

Проверить версию установленного pip:

pip --version

Установка pip:

sudo apt install python3-pip

6.6.6.1 Команды pip

На слайде приведены часто используемые команды для работы с pip.

Установка пакета: pip install <package-name>

Установка пакета конкретной версии: pip install requests==2.25.1

Обновление пакета: pip install --upgrade requests

Удаление пакета: pip uninstall requests

Список пакетов: pip list

Поиска ппакета: pip search requests

6.6.6.2 requirements.txt

requirements.txt — это файл, в котором перечислены все зависимости проекта. Этот файл облегчает установку необходимых пакетов для проекта.

Чтобы создать файл requirements.txt с текущими зависимостями, выполните команду pip freeze > requirements.txt

Чтобы установить все пакеты, перечисленные в requirements.txt, используйте команду: pip install -r requirements.txt

создать файл requirements.txt с текущими зависимостями:

pip freeze > requirements.txt

Установка пакетов из requirements.txt:

pip install -r requirements.txt

6.6.7 venv

venv — это встроенный модуль Python для создания и управления виртуальными окружениями. Виртуальное окружение — это изолированная среда для установки зависимостей проекта, что позволяет избежать конфликтов между пакетами и версиями, используемыми в разных проектах.

venv нужен для:

* **Изоляция проектов**: Виртуальные окружения изолируют зависимости разных проектов, что предотвращает конфликты между версиями пакетов.
* **Управление зависимостями**: Удобно управлять зависимостями, специфичными для проекта, без воздействия на глобальную установку Python.
* **Воспроизводимости**: Легко воспроизводить рабочую среду проекта на разных машинах.

Для создания виртуального окружения выполните команду: python -m venv myenv

Здесь myenv — это имя директории, в которой будет создано виртуальное окружение.

Для активации виртуального окружения используйте source myenv/bin/activate

После активации виртуального окружения в командной строке появится название окружения, например (myenv), что указывает на активное виртуальное окружение.

После активации виртуального окружения все команды pip install будут устанавливать пакеты именно в это окружение.

Чтобы деактивировать виртуальное окружение и вернуться к глобальной установке Python, выполните команду: deactivate

Для удаления виртуального окружения достаточно удалить директорию, в которой оно находится: rm -rf myenv.

venv — это мощный инструмент для управления зависимостями Python-проектов. Он обеспечивает изоляцию окружений, что предотвращает конфликты между пакетами и версиями, и делает работу над проектами более удобной и безопасной. Использование venv вместе с pip и файлами requirements.txt позволяет легко создавать, управлять и воспроизводить рабочие среды для Python-проектов.

Создание виртуального окружения:

python -m venv myenv

Активация виртуального окружения:

source myenv/bin/activate

Деактивация виртуального окружения:

deactivate

Удаление виртуального окружения:

rm -rf myenv

##### 6.7 Исключения и их обработка

Исключения в Python используются для управления ошибками, возникающими во время выполнения программы. Исключения позволяют программам обрабатывать ошибки и продолжать работу вместо аварийного завершения.

**Исключение**: Ошибка, возникающая во время выполнения программы.

**Обработка исключений**: Механизм для перехвата и обработки ошибок, чтобы программа могла продолжить работу.

6.7.1 Зачем нужны исключения?

Рассмотрим зачем нужны исключения:

**Управление ошибками**: Исключения позволяют программам управлять ошибками, возникающими в процессе выполнения, и продолжать работу, вместо того чтобы аварийно завершаться.

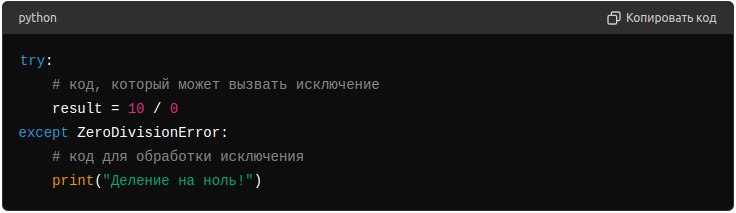
**Разделение кода обработки ошибок**: Исключения помогают отделить код обработки ошибок от основного кода логики, делая программу более читаемой и поддерживаемой.

**Повышение надежности**: Использование исключений делает программы более надежными, так как позволяет явно обрабатывать необычные или неожиданные ситуации.

**Диагностика проблем**: Исключения могут предоставлять подробную информацию об ошибках, что облегчает диагностику и исправление проблем.

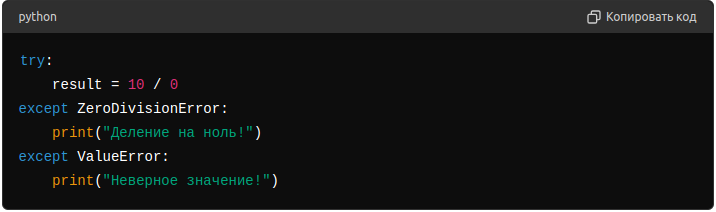
6.7.2 try-except

Исключения в Python обрабатываются с помощью конструкции try-except. В этом примере блок try содержит код, который может вызвать исключение. Если возникает исключение ZeroDivisionError, выполняется блок except.



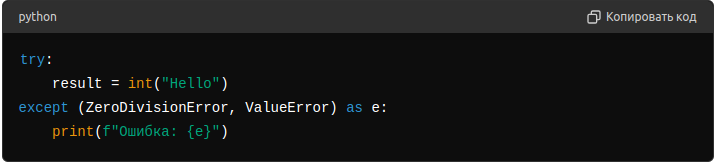
6.7.3 Множественные блоки except

Вы можете обрабатывать различные типы исключений с помощью нескольких блоков except.



6.7.4 Обработка нескольких исключений в одном блоке

Вы можете обрабатывать несколько типов исключений в одном блоке except, указав их в виде кортежа.

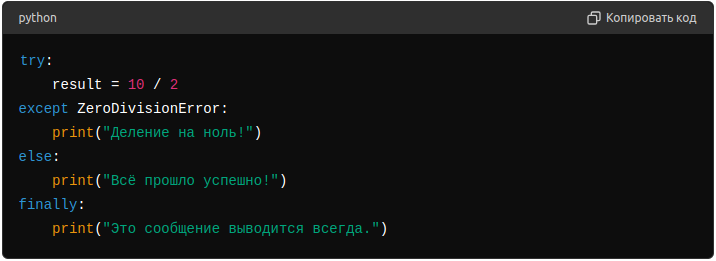


6.7.5 Блоки else и finally

else: Выполняется, если исключение не возникло.

finally: Выполняется всегда, независимо от того, возникло исключение или нет.

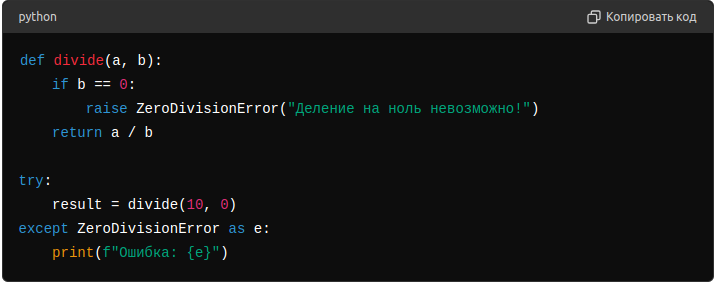
В этом примере блок else выполняется только если исключение не возникло, а блок finally выполняется в любом случае.



6.7.6 Выбрасывание исключений

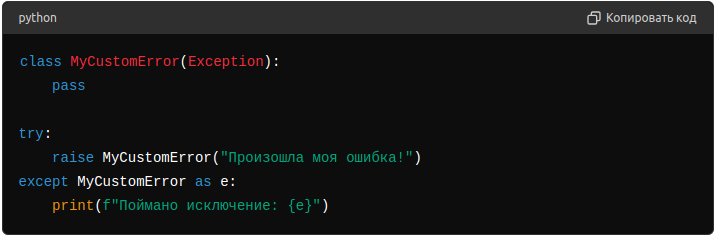
В Python исключения могут быть выброшены с помощью ключевого слова raise. Это позволяет вам сигнализировать о возникновении ошибки или необычной ситуации, которую ваш код не может обработать самостоятельно.

В этом примере выбрасывается исключение ZeroDivisionError с сообщением "Деление на ноль невозможно!", если делитель равен нулю.



6.7.7 Создание собственных исключений

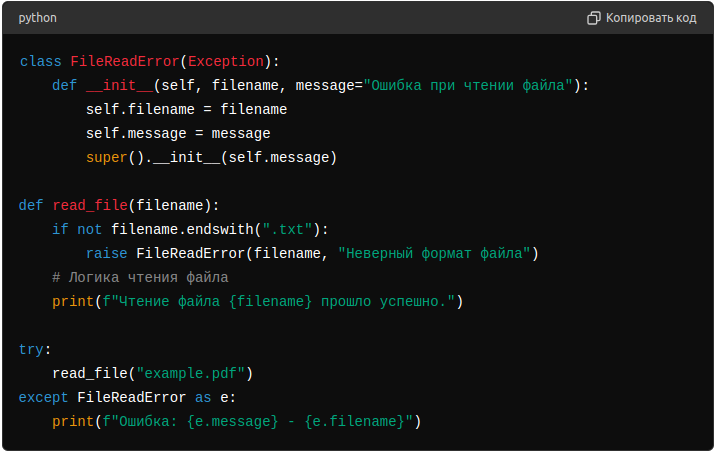
Вы можете создавать собственные классы исключений, наследуя от встроенного класса Exception.



6.7.8 Выбрасывание исключений с дополнительными данными

Вы можете передавать дополнительные данные при выбрасывании исключения, что позволяет предоставить больше информации о возникшей ошибке.

В этом примере исключение FileReadError передает имя файла и сообщение об ошибке при выбрасывании.

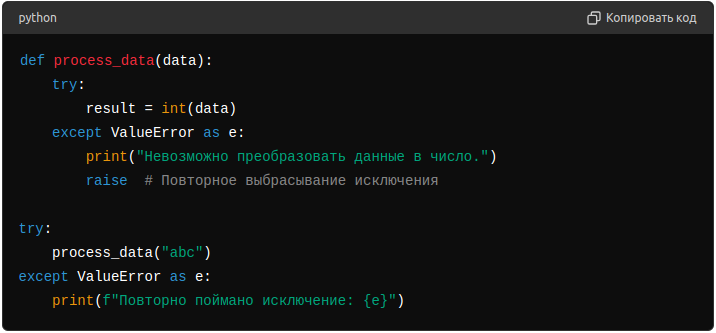


6.7.9 Повторное выбрасывание исключений

Иногда требуется повторно выбросить пойманное исключение для обработки на более высоком уровне.

В этом примере исключение ValueError повторно выбрасывается после его перехвата и обработки в функции process\_data.

Обработка исключений — важный механизм для создания надежных и устойчивых к ошибкам программ. Использование try, except, else и finally позволяет гибко управлять возникновением и обработкой ошибок, обеспечивая корректное выполнение программы даже в случае возникновения непредвиденных ситуаций.



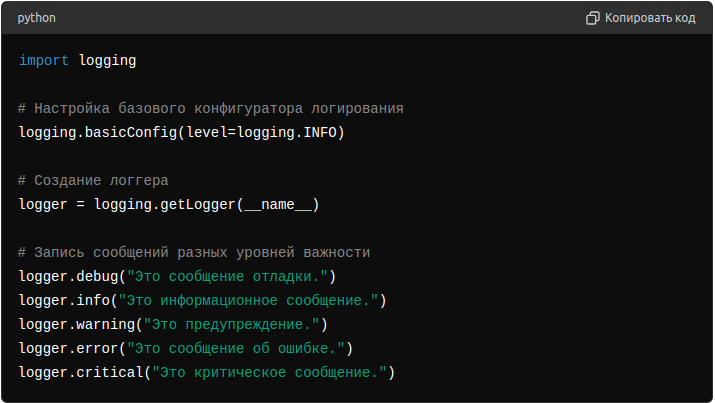
# 8. Логирование

Логирование — это процесс записи сообщений о работе программы. Эти сообщения могут включать информацию о текущем состоянии программы, ошибках, предупреждениях, и другие важные события. Логирование помогает отладке, мониторингу и обслуживанию программного обеспечения.

Основные концепции логирования

* **Логгер (Logger)**: Объект, который используется для записи сообщений.
* **Обработчик (Handler)**: Компонент, который определяет, куда отправляются лог-сообщения (например, в файл или на консоль).
* **Фильтр (Filter)**: Компонент, который фильтрует лог-сообщения на основе определенных критериев.
* **Форматтер (Formatter)**: Компонент, который определяет формат лог-сообщений.

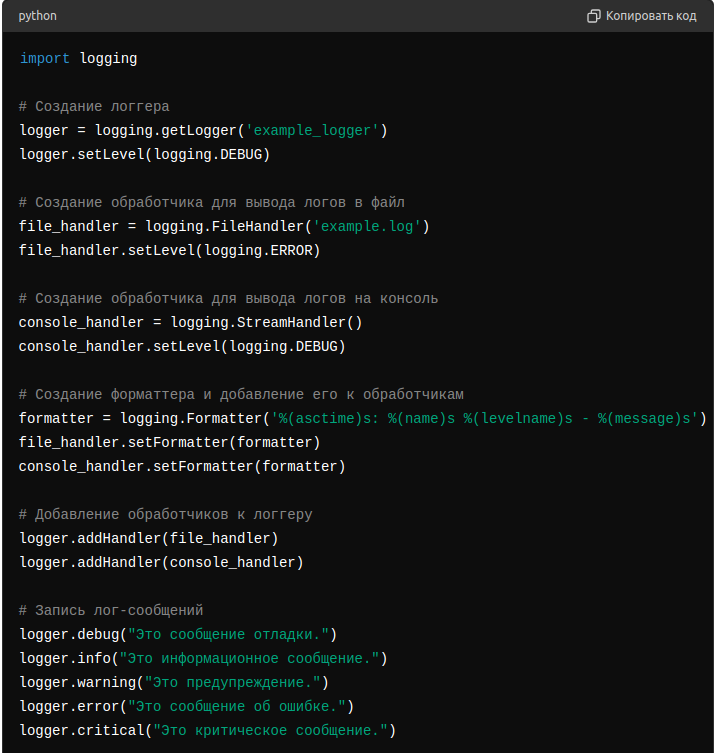
Модуль logging предоставляет встроенные средства для логирования в Python.



Существует несколько уровней логирования:

* DEBUG: Подробная информация, предназначенная для диагностики.
* INFO: Подтверждение того, что все работает как ожидалось.
* WARNING: Указание на что-то неожиданное или потенциальную проблему.
* ERROR: Более серьезная проблема, программа не смогла выполнить некоторую функцию.
* CRITICAL: Очень серьезная ошибка, указывающая на то, что программа может не продолжить работу.

Вы можете настроить логирование, указав обработчики, форматтеры и уровни логирования как показано на слайде.



В этом примере:

* Логгер example\_logger создается с уровнем логирования DEBUG.
* Обработчик файла file\_handler создается для записи логов в файл example.log с уровнем ERROR.
* Обработчик консоли console\_handler создается для вывода логов на консоль с уровнем DEBUG.
* Форматтер используется для форматирования лог-сообщений, включая время, имя логгера, уровень и сообщение.

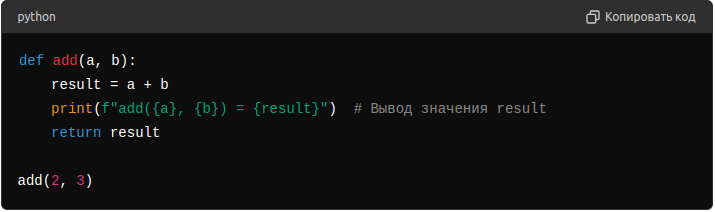
Логирование является важным аспектом разработки приложений, позволяя отслеживать и анализировать поведение программы. Модуль logging в Python предоставляет гибкие и мощные средства для настройки и использования логов, обеспечивая поддержку различных уровней логирования, форматирования и направлений вывода лог-сообщений.

# 9. Отладка скриптов

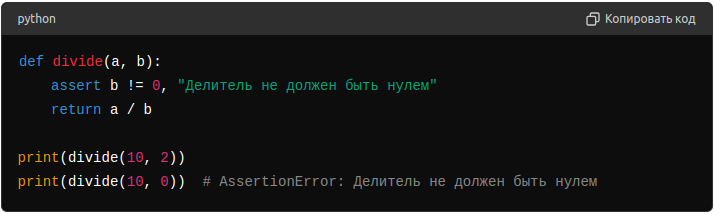
Отладка — это процесс нахождения и устранения ошибок (багов) в программе. Python предоставляет несколько инструментов и методов для эффективной отладки скриптов, которые помогут разработчикам выявлять и исправлять ошибки в коде.

Простейшие методы отладки:

**1. Использование print()**: Самый простой способ отладки — это вывод значений переменных и состояния программы с помощью функции print().



**2. Использование assert**: Утверждения помогают проверять, что определенное условие верно. Если условие ложно, программа выбрасывает исключение AssertionError.

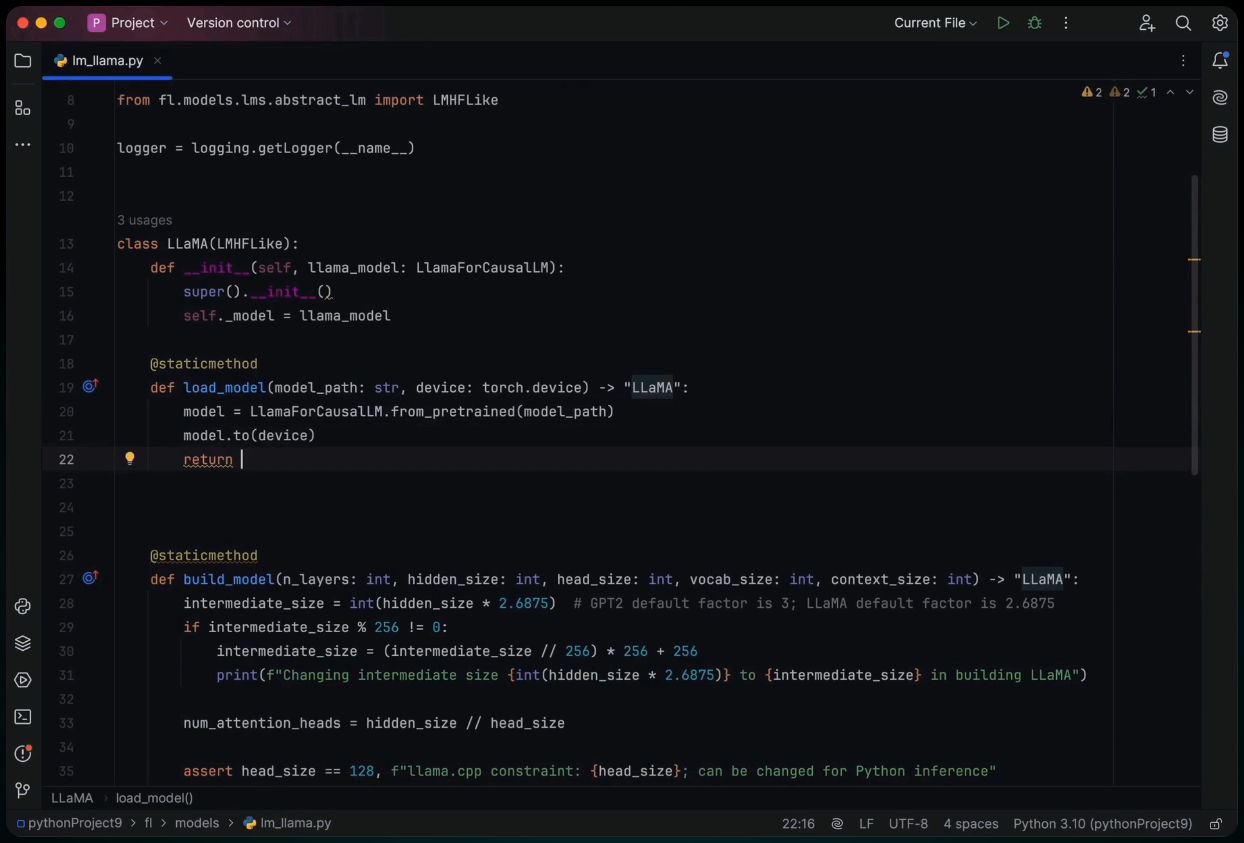


Многие современные интегрированные среды разработки (IDE) поддерживают отладку. Рассмотрим несколько популярных IDE и их возможности:

**PyCharm**:

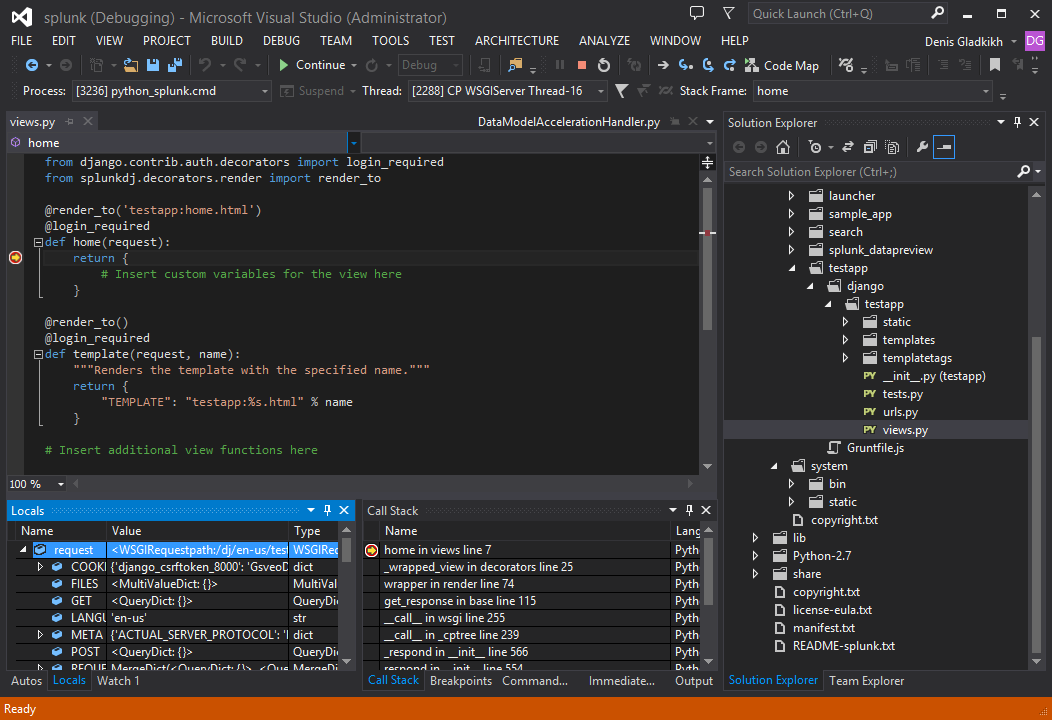
* Поддержка графического интерфейса для отладки.
* Возможность ставить точки останова, выполнять шаги, проверять значения переменных и выражений.
* Поддержка отладки удаленных приложений.

**PyCharm Community Edition поставляется бесплатно. Скачать можно с официального сайта jetbrains.com.**

****

**Visual Studio Code**:

* Плагин Python для отладки (https://code.visualstudio.com/docs/python/debugging).
* Удобный интерфейс для установки точек останова, выполнения шагов и просмотра значений переменных.
* Поддержка конфигурации задач отладки с помощью launch.json.



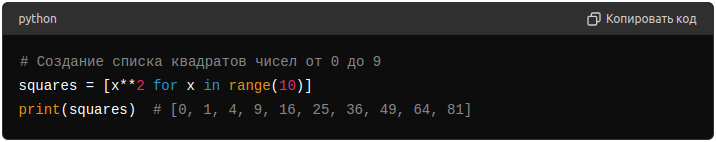
Отладка — это важный этап разработки программного обеспечения. Python предоставляет множество инструментов и методов для эффективной отладки, включая простое использование print() и возможности отладки в современных IDE. Эти инструменты помогают разработчикам находить и исправлять ошибки, обеспечивая правильную работу программы.

# 10. Синтаксический сахар

Синтаксический сахар — это конструкции языка программирования, которые не добавляют новых возможностей, но делают код более понятным и удобочитаемым. Python 3 включает множество таких конструкций, которые облегчают написание и понимание кода.

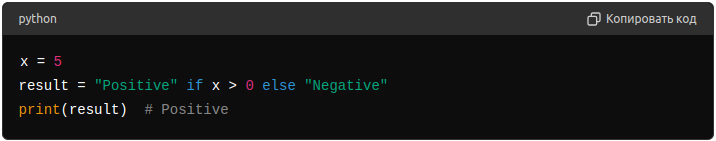
##### 1. Генераторы списков (List Comprehensions)

Генераторы списков позволяют создавать новые списки из итерируемых объектов в более компактной форме.



##### 2. Тернарный оператор (Conditional Expressions)

Тернарный оператор позволяет писать условные выражения в одной строке.



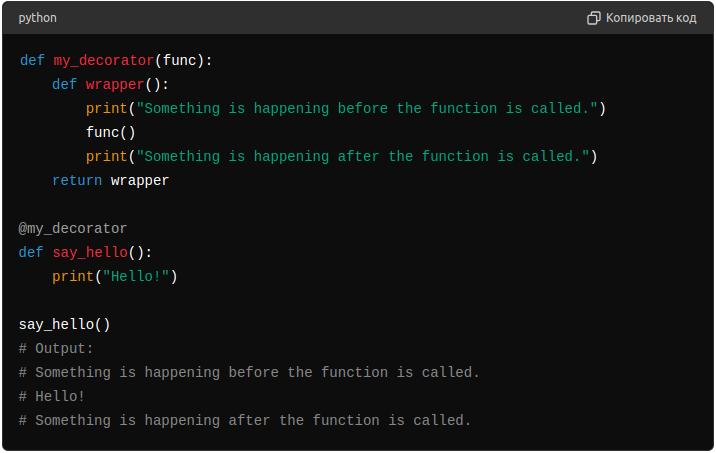
##### 3. Расширенная распаковка (Extended Unpacking)

Python 3 поддерживает расширенную распаковку для удобного извлечения элементов из коллекций.



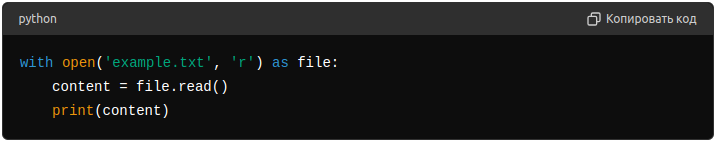
##### 4. Декораторы (Decorators)

Декораторы позволяют оборачивать функции или методы для добавления к ним дополнительной функциональности.



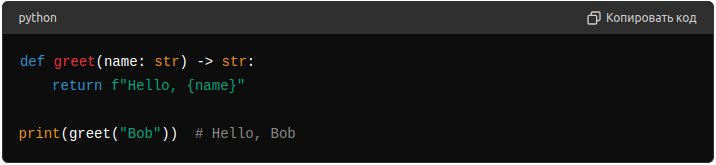
##### 5. Контекстные менеджеры (Context Managers)

Контекстные менеджеры позволяют управлять ресурсами (например, файлами) с помощью конструкции with, автоматически освобождая ресурсы после использования.



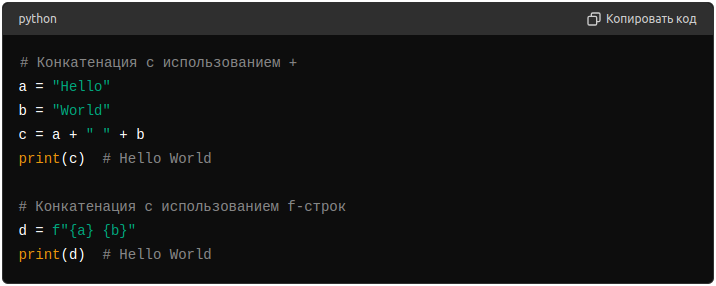
##### 6. Аннотации типов (Type Annotations)

Аннотации типов позволяют указывать типы аргументов и возвращаемого значения функции.



##### 7. Конкатенация строк (String Concatenation)

Python поддерживает несколько удобных способов конкатенации строк.

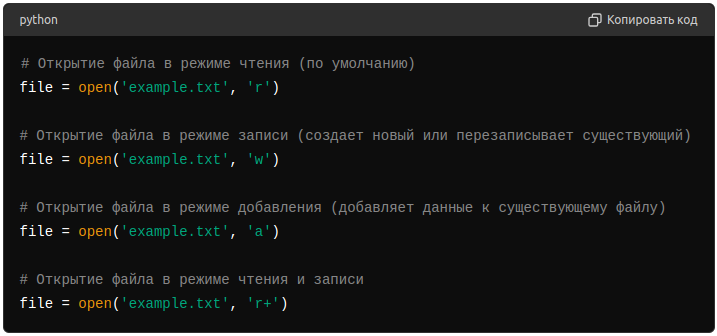


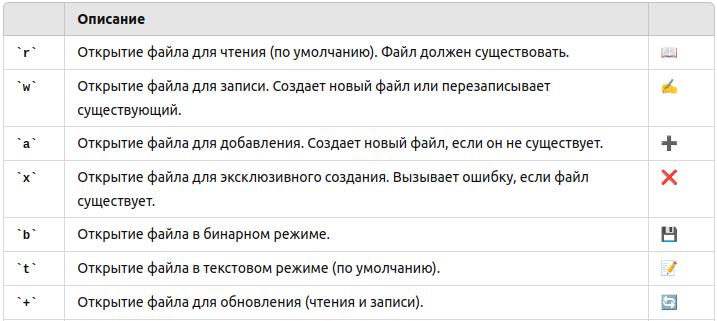
Синтаксический сахар в Python 3 помогает сделать код более лаконичным, читаемым и поддерживаемым. Использование этих удобных конструкций может значительно улучшить ваш опыт программирования и упростить выполнение повседневных задач.

# 11. Работа с файлами

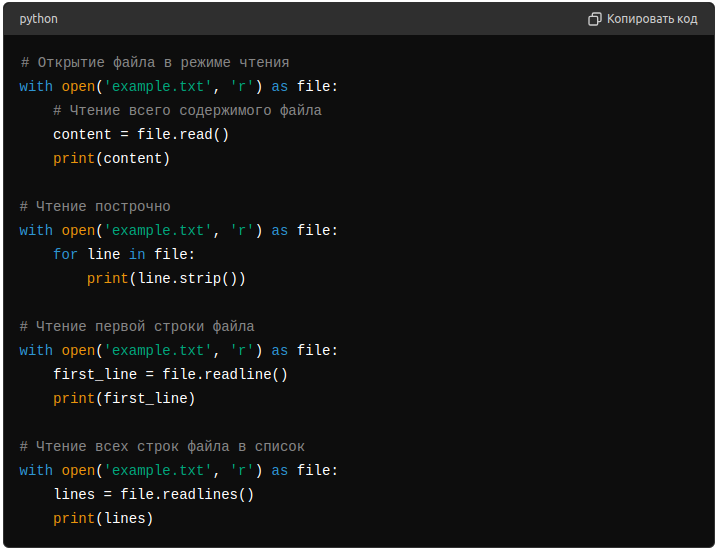
Работа с файлами является одной из важных задач в программировании. Python предоставляет простой и удобный интерфейс для открытия, чтения и записи файлов. Основной инструмент для работы с файлами — это встроенная функция open().

Функция open() используется для открытия файлов. Она принимает два основных аргумента: имя файла и режим открытия.

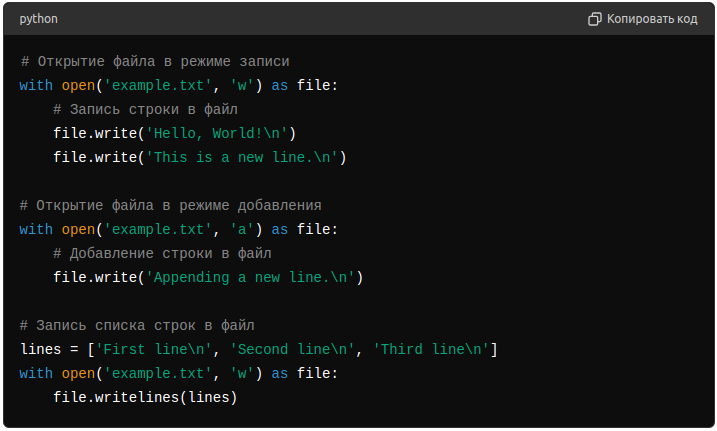




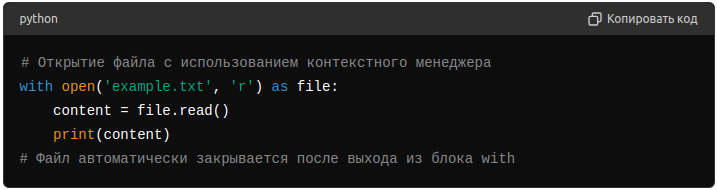
В Python режимы открытия файлов определяют, как файл будет использоваться: для чтения, записи, добавления и т.д. Например, режим 'r' открывает файл только для чтения, а режим 'w' открывает файл для записи, создавая новый файл или перезаписывая существующий. Режим 'a' позволяет добавлять данные в конец файла, сохраняя его содержимое. Режимы могут быть объединены, то есть, к примеру, 'rb' - чтение в двоичном режиме. По умолчанию режим равен 'rt'.



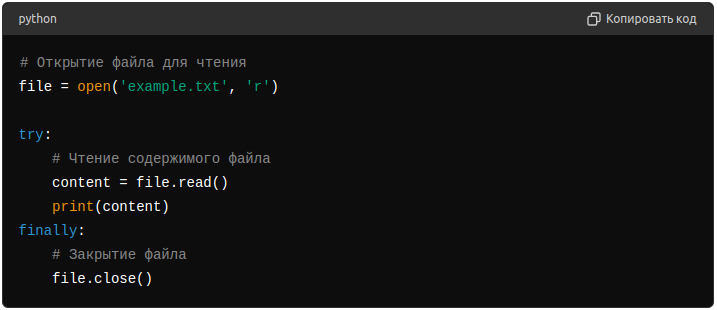
Методы чтения файлов в Python позволяют читать содержимое файла разными способами. Метод read() считывает весь файл целиком и возвращает его содержимое в виде строки. Метод readline() читает одну строку из файла за раз, что удобно для поочередного чтения строк в цикле. Метод readlines() считывает все строки файла и возвращает их в виде списка, где каждая строка является элементом списка.



Методы записи файлов в Python позволяют записывать данные в файл. Метод write() записывает строку или байты в файл, добавляя их к текущему содержимому или заменяя его в зависимости от режима открытия файла. Метод writelines() принимает список строк и записывает их в файл подряд, без добавления символов новой строки, поэтому строки должны уже содержать символы новой строки, если это необходимо. Оба метода требуют открытия файла в режимах, которые позволяют запись, таких как 'w', 'a', или 'r+'.



При работе с файлами необходимо закрывать их после завершения работы. Использование контекстного менеджера with автоматически закрывает файл после завершения блока кода.



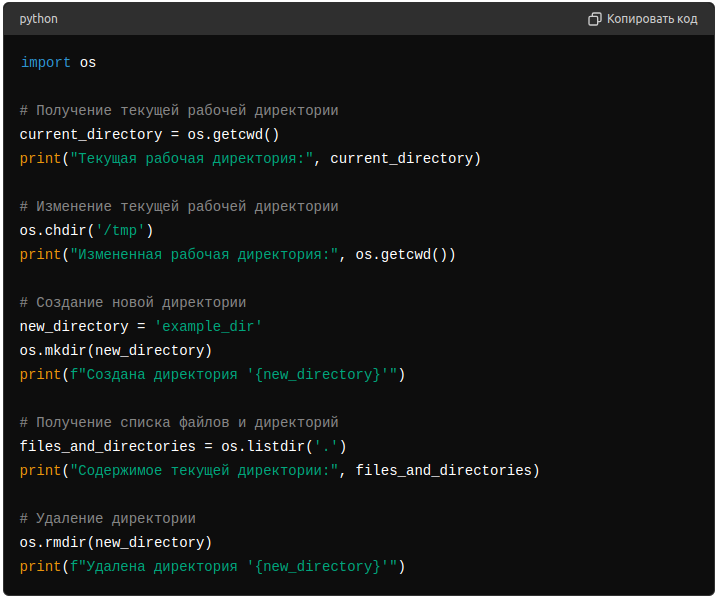
В Python закрытие файлов без использования контекстного менеджера может быть выполнено с помощью метода close(). Важно всегда закрывать файл после завершения работы с ним, чтобы освободить ресурсы. Хотя использование контекстного менеджера with является предпочтительным способом работы с файлами в Python, использование метода close() в сочетании с блоком try...finally также является допустимым способом обеспечить правильное закрытие файлов и освобождение ресурсов.

Работа с файлами в Python проста и удобна благодаря встроенной функции open() и методам для чтения и записи. Использование контекстного менеджера with обеспечивает автоматическое закрытие файлов, что делает код более безопасным и устойчивым к ошибкам.

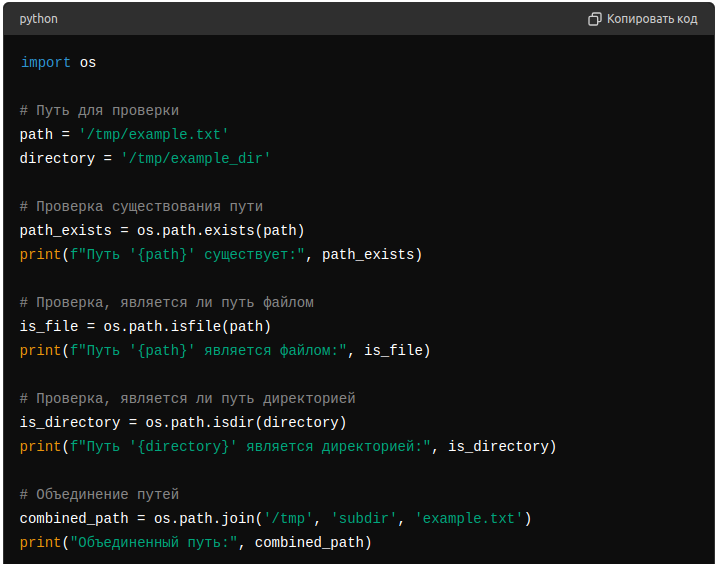
# 12. Пути и директории

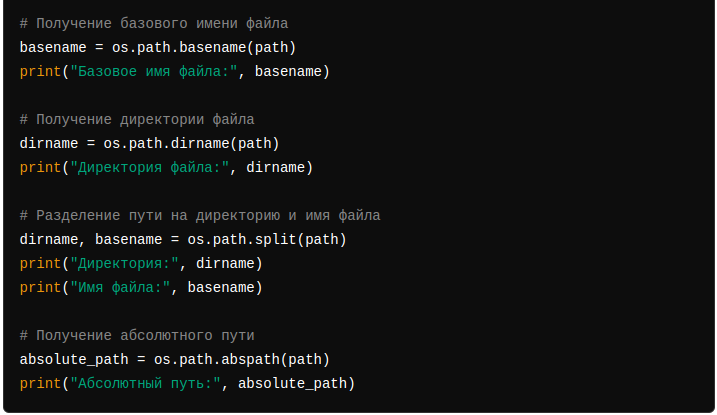
Работа с файлами часто включает в себя управление путями и директориями. Python предоставляет несколько модулей, которые значительно упрощают эту задачу: os, os.path, и pathlib.

Модуль os предоставляет множество функций для работы с операционной системой.

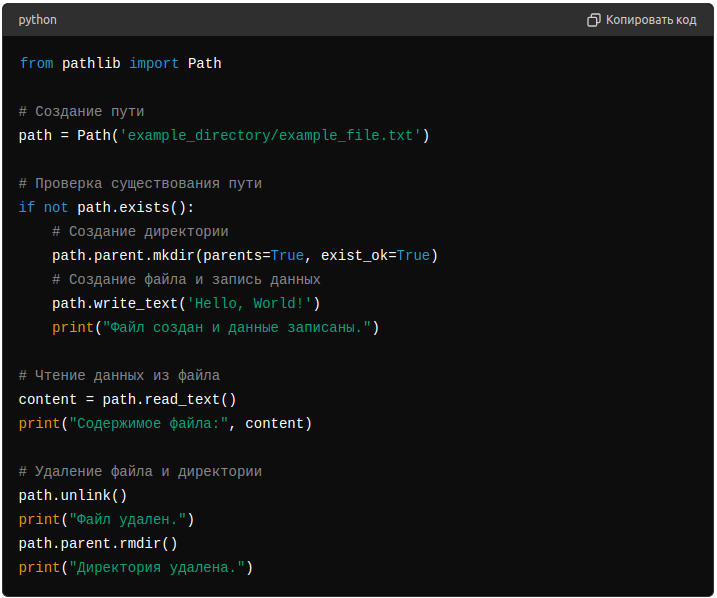


А модуль os.path предоставляет функции для работы с путями.





Модуль pathlib предоставляет объектно-ориентированный интерфейс для работы с путями. Это более современный и удобный способ работы с файлами и директориями.



# 13. Основы работы с сетевыми сокетами в Python

Сетевые сокеты — это основной механизм для реализации сетевого взаимодействия между приложениями. Python предоставляет мощные возможности для работы с сокетами через модуль socket.

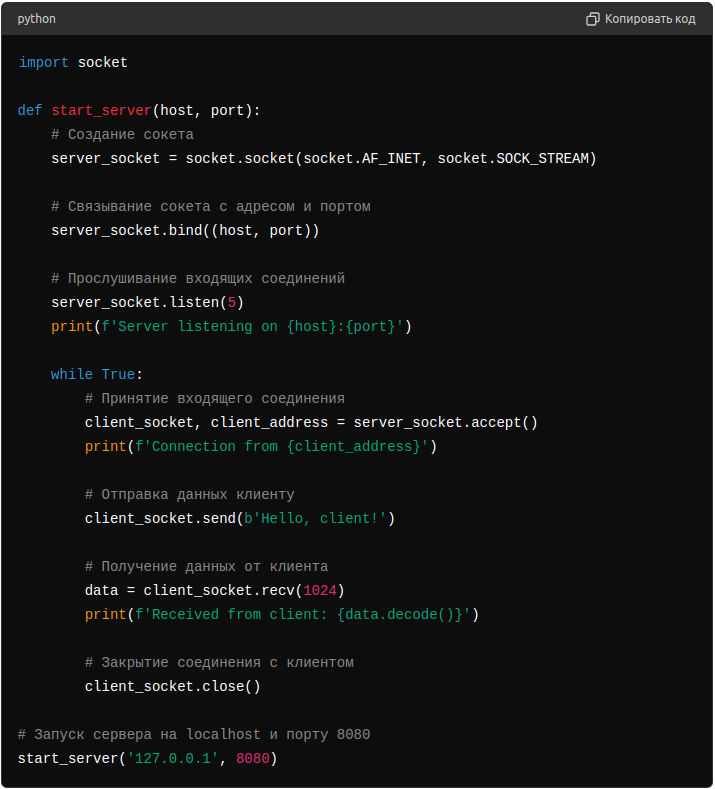
Сокеты — это точки для сетевого соединения, которые позволяют приложениям обмениваться данными по сети. Сокеты могут работать на основе различных протоколов, таких как TCP (Transmission Control Protocol) и UDP (User Datagram Protocol).

Основные функции модуля socket

* socket(): Создание нового сокета.
* bind(): Связывание сокета с адресом и портом.
* listen(): Прослушивание входящих соединений (только для TCP-серверов).
* accept(): Принятие входящего соединения (только для TCP-серверов).
* connect(): Установка соединения с сервером (только для TCP-клиентов).
* send() и recv(): Отправка и получение данных через сокет (для TCP).
* sendto() и recvfrom(): Отправка и получение данных через сокет (для UDP).
* close(): Закрытие сокета.

Здесь на слайде показан пример tcp сервера, который:

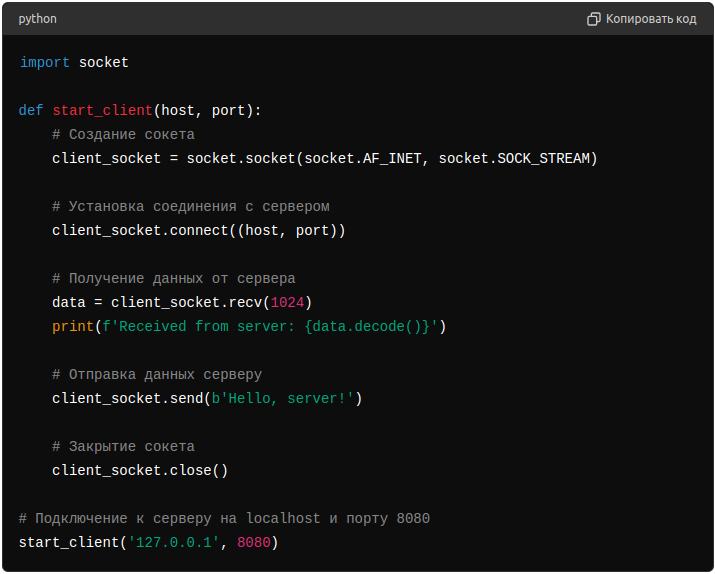
* Создает сокет с помощью socket.socket().
* Привязывает его к указанному адресу и порту с помощью bind().
* Начинает прослушивание входящих соединений с помощью listen().
* Принимает входящие соединения с помощью accept().
* Обрабатывает соединения, отправляя и получая данные с помощью send() и recv().
* Закрывает соединение с клиентом с помощью close().



При запуске этого сервера он начнет прослушивать адрес 127.0.0.1 на порту 8080.

А здесь на слайде показан пример tcp-клиента, который:

* Создает сокет с помощью socket.socket().
* Подключается к серверу с помощью connect().
* Получает данные от сервера с помощью recv().
* Отправляет данные серверу с помощью send().
* Закрывает сокет с помощью close().



При запуске клиента, он отправит сообщение Hello server на сервер и в ответ от сервера получит сообщение Hello client.



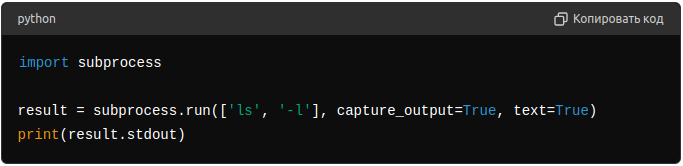


Эти примеры демонстрируют основные принципы работы с сетевыми сокетами в Python, позволяя создавать простые клиент-серверные приложения.

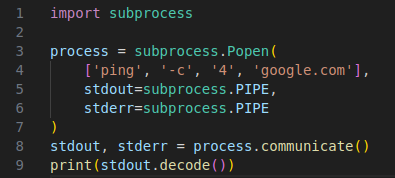
# 14. Управление процессами

Python предоставляет несколько модулей для управления процессами, такие как os, subprocess, и multiprocessing. Эти модули позволяют запускать новые процессы, взаимодействовать с ними и контролировать их выполнение. Модуль subprocess предоставляет более гибкий способ запуска новых процессов и взаимодействия с ними.

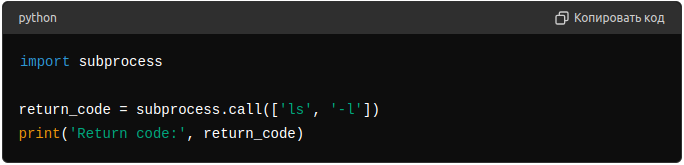
subprocess.run: Запускает команду, ожидает её завершения и возвращает результат.



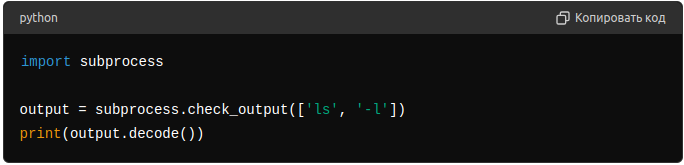
subprocess.Popen: Запускает новый процесс и позволяет взаимодействовать с ним через потоки ввода, вывода и ошибок.



subprocess.call: Запускает команду и ожидает её завершения, возвращая код завершения.



subprocess.check\_output: Запускает команду и возвращает её вывод. Если команда завершилась с ненулевым кодом, вызывает исключение.



Модуль subprocess в Python предоставляет мощные средства для создания и управления новыми процессами, что позволяет выполнять системные команды и взаимодействовать с ними. Он заменяет старые функции, такие как os.system, os.spawn\*, os.popen\*, предоставляя более гибкий и надежный способ управления процессами. Основные функции модуля включают subprocess.run(), subprocess.Popen(), subprocess.call(), и subprocess.check\_output(), каждая из которых предлагает различные уровни контроля над выполнением команд. С subprocess можно легко перенаправлять стандартный ввод, вывод и поток ошибок, а также передавать данные между процессами. Это делает subprocess незаменимым инструментом для автоматизации задач и написания скриптов, взаимодействующих с операционной системой.

# 15. Чтение системных журналов

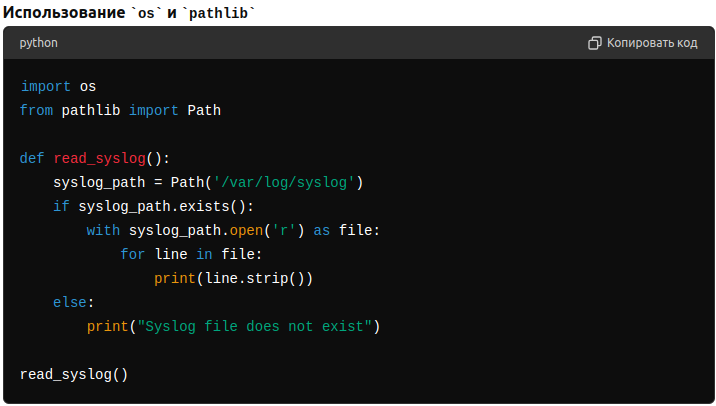
Системные журналы содержат важную информацию о событиях, происходящих в операционной системе. В Linux и Unix-подобных системах такие журналы часто хранятся в файлах, расположенных в директории /var/log. Для чтения и анализа этих журналов можно использовать стандартные инструменты Python.

Основные модули и методы

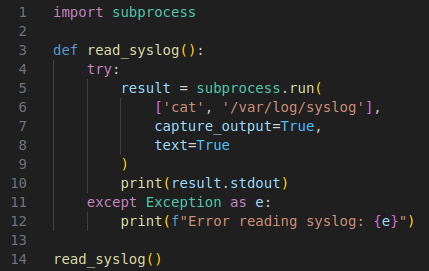
* os: Для взаимодействия с операционной системой.
* subprocess: Для выполнения системных команд.
* pathlib: Для удобной работы с путями.
* re: Для обработки и анализа строк с помощью регулярных выражений.

Рассмотрим пример чтения журнала syslog, который обычно находится в файле /var/log/syslog разными способами.

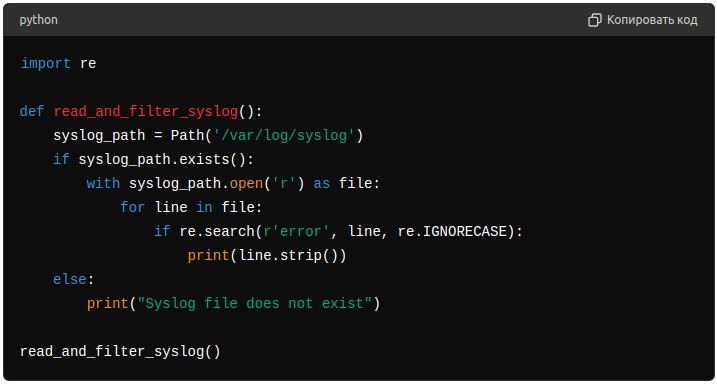
Здесь на слайде приведен пример построчного чтения при помощи модулей os и pathlib:



Вы можете использовать модуль subprocess для выполнения команды cat или tail для чтения содержимого журнала:



Вы можете использовать регулярные выражения для фильтрации и анализа содержимого журналов при помощи модуля re:



Чтение и анализ системных журналов в Python может быть выполнено с помощью стандартных библиотек, таких как os, subprocess, pathlib, и re. Эти инструменты позволяют эффективно обрабатывать журналы, извлекать важную информацию и анализировать события, происходящие в системе.

# 16. Магические переменные

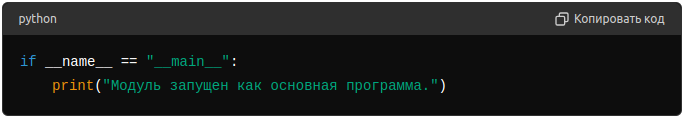
В Python существует несколько специальных переменных (также называемых "магическими"), которые начинаются и заканчиваются двойным подчеркиванием (\_\_). Эти переменные часто используются для предоставления информации о текущем состоянии объекта или контексте выполнения программы. Они также позволяют изменить поведение встроенных функций и операторов.

Рассмотрим некоторые:

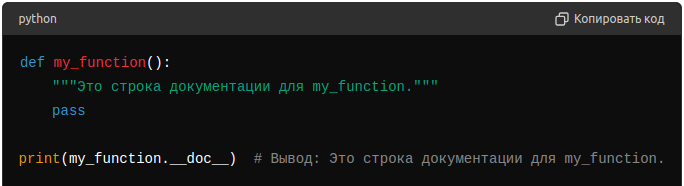
\_\_name\_\_: Имя модуля.

Значение '\_\_main\_\_', если модуль был запущен как основная программа.

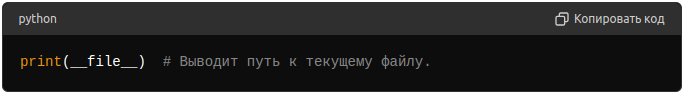
Имя модуля, если модуль был импортирован.



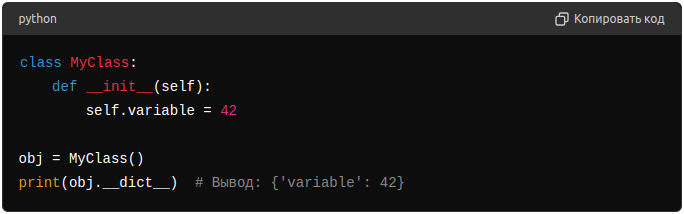
\_\_doc\_\_: Строка документации модуля, класса или функции.



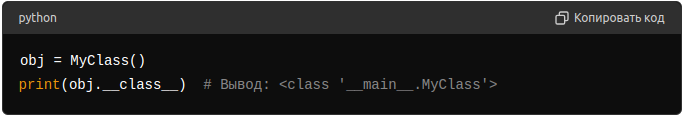
\_\_file\_\_: Путь к файлу текущего модуля.



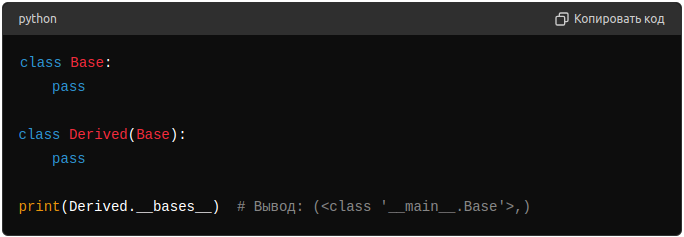
\_\_dict\_\_: Словарь, содержащий атрибуты модуля, класса или объекта.



\_\_class\_\_: Указывает класс, к которому принадлежит экземпляр.



\_\_bases\_\_: Кортеж базовых классов.

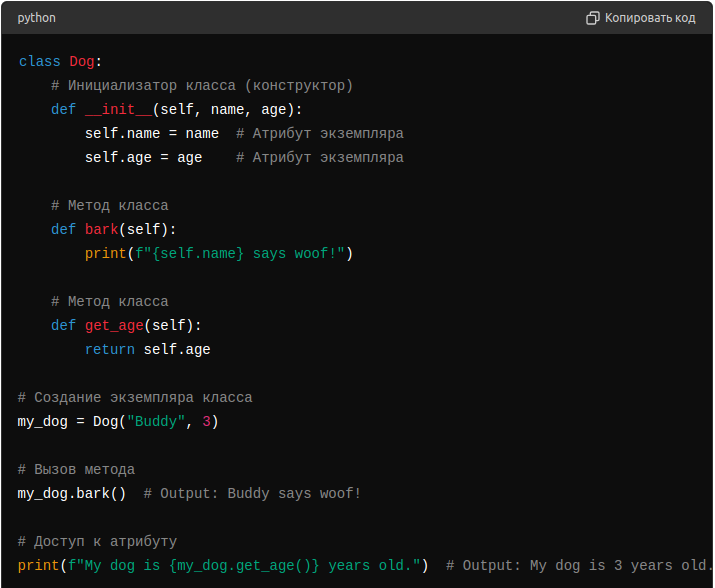


Специальные переменные в Python предоставляют мощные средства для управления поведением модулей. Они предоставляют дополнительную информацию о состоянии программы и позволяют организовывать код более эффективно и читаемо. Использование этих переменных помогает разработчикам создавать более гибкие и интуитивно понятные API.

# 17. Классы

Классы в Python являются основой объектно-ориентированного программирования (ООП). Классы позволяют создавать собственные типы данных, объединяя данные и методы, которые работают с этими данными, в единую структуру.

Создание класса в Python начинается с ключевого слова class, за которым следует имя класса и двоеточие. Внутри класса определяются методы и атрибуты.



Основные компоненты класса:

* Конструктор (\_\_init\_\_ метод): Используется для инициализации экземпляра класса. Принимает аргументы, которые задают начальные значения атрибутов экземпляра.



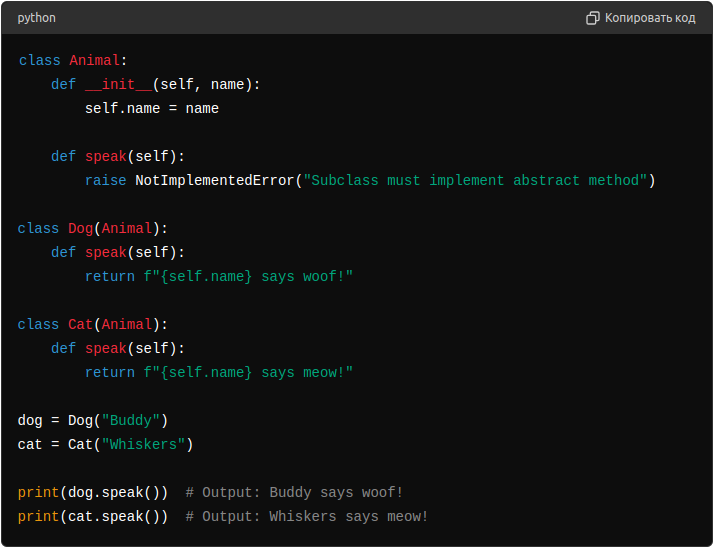
* Атрибуты: Переменные, принадлежащие классу или экземпляру класса. Экземплярные атрибуты задаются в методе \_\_init\_\_ с помощью self



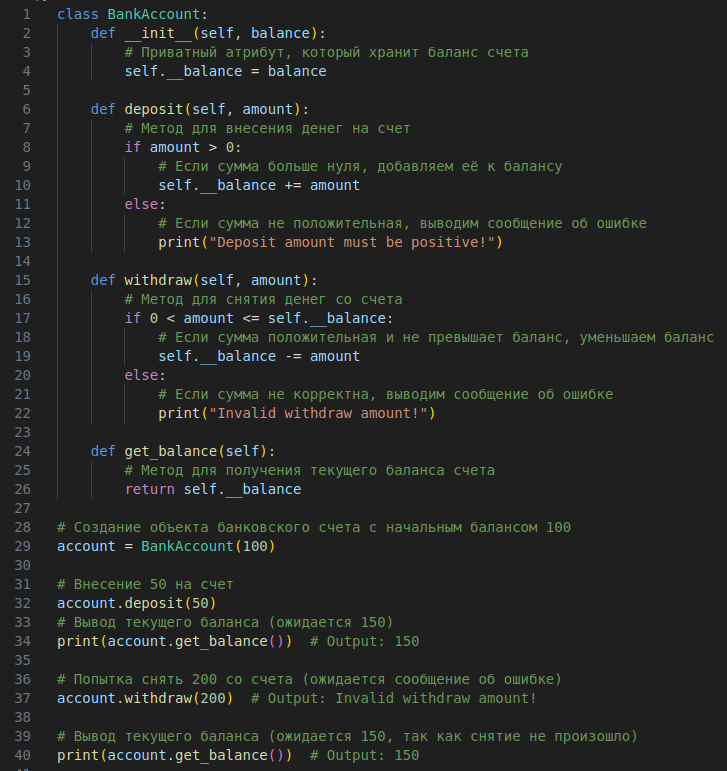
* Методы: Функции, определенные внутри класса. Принимают первым аргументом ссылку на экземпляр (self)



Классы в Python поддерживают наследование, что позволяет создавать новый класс на основе существующего.

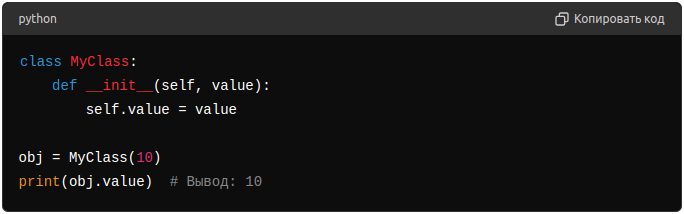


Инкапсуляция заключается в сокрытии внутреннего состояния объекта и предоставлении методов для работы с этим состоянием. Этот пример демонстрирует, как инкапсуляция позволяет скрывать внутреннее состояние объекта и управлять доступом к нему через методы класса, обеспечивая целостность данных и корректность операций.



Магические методы (или методы с двойным подчеркиванием) позволяют перегружать операторы и изменять поведение встроенных функций.

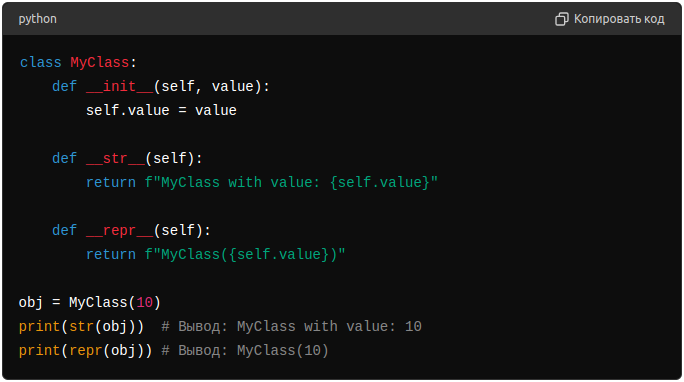
\_\_init\_\_: Метод инициализации экземпляра класса.



\_\_str\_\_ и \_\_repr\_\_: Методы для строкового представления объекта.

\_\_str\_\_ используется функцией str() и print().

\_\_repr\_\_ используется функцией repr() и отладчиком.



Классы в Python предоставляют мощные средства для создания и управления объектами. Использование ООП позволяет писать более структурированный, удобочитаемый и поддерживаемый код. Основные концепции, такие как наследование, полиморфизм и инкапсуляция, помогают организовать код и упростить его развитие и поддержку.

# Заключение

В завершение нашей лекции мы получили всестороннее понимание Python и его применения в различных областях. Мы начали с изучения истории Python, узнали его рейтинг в среде других языков программирования, рассмотрели синтаксис и основные конструкций, что дало нам прочную базу для дальнейшего изучения. Мы рассмотрели, как эффективно работать с файловой системой и управлять путями и директориями с помощью модулей os и pathlib. Изучили методы управления процессами с помощью модуля subprocess, что позволило нам автоматизировать и контролировать выполнение системных команд. В рамках объектно-ориентированного программирования мы разобрали ключевые концепции инкапсуляции, наследования и полиморфизма. Также мы познакомились с основами работы с сетевыми сокетами и чтением системных журналов, что расширяет наши возможности по взаимодействию с сетью и операционной системой. Эти знания значительно расширят ваши возможности как системного администратора, помогая вам создавать более эффективные и надежные скрипты обслуживания операционных систем на базе linux.