**Глава 1**

Python описывается как мощный и простой для изучения язык программирования. Он предоставляет хорошо продуманные высокоуровневые структуры данных и поддерживает объектно-ориентированное программирование. Благодаря изящному синтаксису, динамической типизации и интерпретируемой природе, Python идеально подходит для написания сценариев и ускоренной разработки приложений. Его также можно использовать для связи различных программных компонентов.

Создателем Python является Гвидо ван Россум, который начал работу над языком в 1991 году. Название "Python" было выбрано в честь комедийного шоу "Летающий цирк Монти-Питона". Язык активно развивается при поддержке организаций, в которых работал Гвидо, а сейчас над ним работает большое мировое сообщество программистов. Однако последнее слово в направлении развития остается за Гвидо ван Россумом.

Программисты используют Python по нескольким ключевым причинам:

* Качество кода: Код на Python легко читается, что упрощает его повторное использование и обслуживание. Единообразие оформления делает код понятным даже для тех, кто не участвовал в его создании.
* Высокая скорость разработки: Python значительно повышает производительность разработчика по сравнению с компилируемыми или строго типизированными языками, такими как C++ или Java. Объем кода на Python обычно в разы меньше.
* Переносимость: Программы на Python часто работают без изменений на всех основных платформах.
* Библиотеки поддержки: Язык поставляется с большой стандартной библиотекой, предоставляющей готовые функции для различных задач (от поиска текста до сетевых операций). Также можно использовать собственные и сторонние библиотеки.
* Интеграция компонентов: Сценарии Python легко взаимодействуют с другими частями приложений, могут вызывать функции из C/C++ библиотек, интегрироваться с Java-компонентами и обмениваться данными по сети.

Python активно используется многими крупными компаниями и организациями, включая Google, Youtube, Яндекс, Mail.ru, NASA и другие. С его помощью можно создавать системные скрипты, программы с графическим интерфейсом, веб-сайты и веб-приложения, приложения баз данных, быстро создавать прототипы, заниматься математическими и научными вычислениями, а также разрабатывать игры, приложения для обработки изображений, искусственный интеллект и многое другое.

Сильные стороны Python включают его объектно-ориентированность, бесплатность и свободное распространение, переносимость, а также мощность:

* Динамическая типизация: Типы переменных определяются во время выполнения.
* Автоматическое управление памятью (сборщик мусора).
* Поддержка модульного программирования.
* Встроенные типы объектов, такие как списки, словари, кортежи и строки.
* Большая коллекция инструментов в стандартной библиотеке и от сторонних разработчиков.
* Возможность интеграции с программами на C/C++.
* Удобство использования, поскольку код не требует компиляции.
* Простота изучения.

Существуют две основные ветки версий: Python 2.x и Python 3.x. Они не совместимы между собой, и для работы с обеими версиями часто требуются значительные изменения в коде. Несмотря на выход 3.x, много существующего ПО написано на 2.x.

В Python переменные объявляются просто путем присваивания значения; тип переменной не нужно указывать явно в исходном коде, он отслеживается во время выполнения. Язык чувствителен к регистру ('a' и 'A' - разные переменные). Конец строки является концом оператора, точка с запятой в конце инструкции не требуется (хотя ее можно использовать, но это не лучший стиль). Комментарии начинаются с символа # и продолжаются до конца строки.

Исходные файлы Python имеют расширение ".py". Программу можно запустить из консоли командой python имя\_файла.py.

Типичная структура программы часто включает импорт модулей, определение главной функции (например, main) и использование конструкции if \_\_name\_\_ == '\_\_main\_\_': main(). Эта конструкция позволяет коду работать как самостоятельная программа при прямом запуске, но также быть импортированным как модуль в другую программу.

Функции определяются с помощью ключевого слова def. Тело функции определяется уровнем отступа. Первой строкой может быть строка документации ("docstring"), описывающая назначение функции. Переменные, определенные внутри функции, являются локальными для этой функции.

В Python нет отдельного типа для символов (как Char в некоторых других языках). Строки обладают множеством полезных методов для работы с ними, таких как преобразование регистра (lower(), upper()), удаление пробелов (strip()), поиск (find()), замена (replace()), разделение на список (split()) и объединение элементов списка в строку (join()). Строки также поддерживают форматирование с помощью метода format(), используя позиционные или именованные аргументы. Существуют также методы для проверки содержимого строки (например, isalnum(), isdigit()).

Для определения блоков кода в условных операторах (if), циклах и функциях Python использует двоеточие (:) и отступы, а не фигурные скобки {}. Логические условия в if не требуют скобок. "Нулевые" значения (None, 0, пустые строки, списки, словари) считаются Ложью (False). Логические операторы записываются словами: and, or, not.

Цикл for ... in позволяет легко итерировать по элементам коллекции, такой как список. Также с его помощью можно перебирать символы в строке. Конструкция value in collection позволяет проверить наличие элемента в коллекции.

Функция range() используется в циклах for для генерации последовательности чисел. range(n) генерирует числа от 0 до n-1, а range(a, b) — от a до b-1.

В Python также присутствует стандартный цикл while, а операторы break и continue позволяют управлять его выполнением.

Списки — это изменяемые коллекции элементов. У списков есть множество методов для работы с их содержимым, таких как append() (добавить элемент в конец), insert() (вставить элемент по индексу), extend() (добавить элементы из другого списка), index() (найти индекс элемента), remove() (удалить элемент по значению), pop() (удалить и вернуть элемент по индексу), count() (подсчитать количество вхождений элемента). Важно помнить, что некоторые методы, такие как append, sort, reverse, изменяют исходный список и не возвращают новый.

Для сортировки списков предпочтительно использовать функцию sorted(list), которая возвращает новый отсортированный список, не изменяя исходный. Можно сортировать в обратном порядке (reverse=True) или использовать параметр key для задания пользовательской функции сортировки (например, по длине элементов).

Если вам нужна помощь при работе с Python, вы можете использовать встроенные функции help() и dir() в интерактивной оболочке Python. help() показывает документацию по модулям, функциям и методам, а dir() — список доступных методов объекта.

**Глава 2**

.

Дополнительные числовые типы:

Decimal (десятичное число): Предназначен для выполнения арифметических операций над дробными числами с высокой точностью, подобно школьной арифметике. В отличие от стандартных вещественных чисел (float), которые могут давать неточные результаты при работе с десятичными дробями (например, 0.1 + 0.1 + 0.1 - 0.3 не равно нулю для float), числа Decimal могут быть представлены точно. Такие неточности могут накапливаться и приводить к неприятным последствиям. По этой причине, числа Decamal предпочтительнее для бухгалтерских приложений, для которых важно четкое равенство. Decimal имеет изменяемую пользователем точность, по умолчанию до 28 знакомест, которая может быть насколько большой, насколько это необходимо для данной задачи. Десятичное число может быть создано из целого числа, строки, числа с плавающей запятой или кортежа. Начиная с Python 3.3, производительность Decimal сопоставима с int и float благодаря ускорителю.

Fraction (дробное число): представляет дробные числа. Может быть создано из пары целых чисел (числитель, знаменатель), другого дробного числа, вещественного числа или строки.

Кортежи:

Кортеж — это неизменяемый список.

Зачем нужны кортежи?

Защита от изменений: его удобно использовать в случаях, когда важно быть уверенным, что данные не изменялись в ходе выполнения программы.

Занимают меньше памяти по сравнению со списками.

Могут использоваться в качестве ключей словаря, так как они хешируемы (в отличие от списков).

Работа с кортежами: допускаются те же операции, что и для списков, за исключением операций, пытающихся изменить кортеж. Возможны конкатенация (+), индексация ([ ]), срезы (:) и методы вроде count(). Попытка использовать методы для изменения, такие как append(), вызовет ошибку AttributeError.

Словари:

Словарь — это неупорядоченная коллекция произвольных объектов с доступом по ключу.

Способы создания словаря:

литерал ({}).

функция dict() (из пар ключ-значение или последовательности пар).

метод fromkeys() (создает словарь с ключами из последовательности и опциональным значением по умолчанию).

генератор словарей.

Основные операции со словарями:

len(d): число элементов в словаре.

d[key]: возвращает элемент словаря с ключом key.

d[key] = value: присваивает значение элементу словаря.

del d[key]: удаляет элемент по ключу; вызывает KeyError, если ключа нет.

key in d: возвращает True, если у словаря есть ключ key.

key not in d: эквивалентно not key in d.

Методы словарей:

get(key[, default]): возвращает элемент словаря с ключом key, если ключ есть, иначе default (по умолчанию None); никогда не вызывает исключение KeyError.

items(): возвращает список пар (ключ, значение); удобно использовать для цикла по словарю.

keys(): возвращает список ключей словаря.

pop(key[, default]): если в словаре есть элемент с ключом key — удаляет его и возвращает его значение, в противном случае возвращает default; если default не задан — вызывает исключение KeyError.

popitem(): удаляет и возвращает произвольную пару (ключ, значение); если словарь пуст, возбуждает исключение KeyError.

setdefault(key[, default]): если ключ key есть в словаре — возвращает его значение; если нет — создает ключ со значением из default (по умолчанию None) и возвращает default.

update([other]): обновляет словарь, добавляя пары (ключ, значение) из other; существующие ключи перезаписываются; возвращает None.

values(): возвращает копию списка значений словаря.

copy(): возвращает копию словаря.

clear(): удаляет все элементы из словаря.

Множества:

Множество — это контейнер, содержащий не повторяющиеся элементы в случайном порядке.

Создание множества: функция set() (из итерируемого объекта), литерал ({}) (но пустые скобки {} создают словарь), генератор множеств.

Операции с множествами:

len(s): число элементов в множестве.

x in s: принадлежит ли x множеству s.

s.isdisjoint(t): истина, если set и other не имеют общих элементов.

s == t: равенство множеств.

s.issubset(t) или s <= t: s является подмножеством t.

s.issuperset(t) или s >= t: s является надмножеством t.

Операции объединения, пересечения, разности и симметричной разности:

объединение: s.union(t, ...) или s | t.

пересечение: s.intersection(t, ...) или s & t.

вычитание: s.difference(t, ...) или s - t. Возвращает множество из всех элементов set, не принадлежащих ни одному из other.

симметричная разность: s.symmetric\_difference(t) или s ^ t. Возвращает множество из элементов, встречающихся в одном множестве, но не встречающиеся в обоих.

Эти операции также имеют версии с обновлением множества на месте (s.update(), s.intersection\_update(), s.difference\_update(), s.symmetric\_difference\_update()) с соответствующими операторами (|=, &=, -=, ^=).

Методы, изменяющие множества:

add(elem): добавляет элемент в множество.

remove(elem): удаляет элемент из множества; вызывает KeyError, если такого элемента не существует.

discard(elem): удаляет элемент, если он находится в множестве, без ошибки.

pop(): удаляет и возвращает произвольный элемент из множества; вызывает KeyError, если множество пустое.

clear(): очистка множества.

Frozenset — полностью похож на set, но является неизменяемым типом данных. Это аналогия с парами списки и кортежи.

Файлы:

В Python есть встроенные средства для работы с файлами: открытие, закрытие, чтение, запись.

Открытие файла: используется функция open().

Первый аргумент — имя файла (может быть относительным или абсолютным путем).

Второй аргумент — режим открытия:

'r' - чтение (по умолчанию)

'w' - запись (содержимое удаляется, новый файл создается если не существует)

'a' - дозапись (добавляет в конец файла)

'b' - бинарный режим

't' - текстовый режим (по умолчанию)

'+' - чтение и запись Режимы могут быть объединены, то есть, к примеру, ‘rb’ — чтение в двоичном режиме. По умолчанию режим равен ‘rt’.

Последний аргумент encoding используется только в текстовом режиме для указания кодировки.

Методы работы с файлами:

f.read(): чтение файла целиком в единственную строку.

f.read(N): чтение следующих N символов (или байтов) в строку.

f.readline(): чтение следующей текстовой строки (включая символ конца строки) в строку.

f.readlines(): чтение файла целиком в список строк (включая символ конца строки).

f.write(string): запись строки символов (или байтов) в файл.

f.writelines(list): запись всех строк из списка в файл.

f.close(): закрытие файла вручную (выполняется по окончании работы с файлом).

f.flush(): выталкивает выходные буферы на диск, файл остается открытым.

f.seek(N): изменяет текущую позицию в файле для следующей операции, смещая ее на N байтов от начала файла.

По файлу можно итерироваться построчно (for line in open(...)). Поддерживается работа с файлами Unicode текста (encoding=...) и бинарными файлами ('rb').

Генераторы:

Генераторы используются для создания списков, кортежей, словарей и множеств.

Генератор списка возвращает готовый список ([i for i in range(10)]).

Генератор кортежа ((i for i in range(10))) возвращает не объект кортеж, а объект-генератор. Это позволяет существенно экономить занимаемую кортежем память.

Генератор словаря возвращает словарь ({i : i+i for i in range(7)}).

Генератор множества возвращает множество ({i for i in range(10)}).

Генераторы особенно удобно использовать со встроенными функциями, такими как sum(), min(), max(). Например, можно подсчитать максимальную длину строки в файле.

**Глава 3**

**Функциональное программирование**

Ввод данных пользователя и интерактивные циклы. Для ввода данных с клавиатуры в Python используется встроенная функция input(). Она может выводить подсказку, если ей передать аргумент.

Типичный шаблон для создания интерактивных циклов, которые считывают данные, обрабатывают их и выводят результат, основан на бесконечном цикле while True. Ввод считывается функцией input(), и цикл прерывается по определенному условию, например, если пользователь ввел 'stop'.

Обработка ошибок при вводе

Можно проверять введенные данные перед использованием, например, с помощью строкового метода isdigit() для проверки, состоит ли строка только из цифр. Если ввод некорректный, можно вывести сообщение об ошибке.

Более универсальный подход — использовать конструкцию try...except...else.

Код, который потенциально может вызвать ошибку (например, преобразование строки в число функцией int()), помещается в блок try.

Если в блоке try возникает исключение (ошибка), выполняется код в блоке except.

Если в блоке try исключение не возникло, выполняется код в блоке else.

Этот подход предполагает успешное выполнение операции и обрабатывает ошибку, если она происходит, вместо явной предварительной проверки каждого возможного случая.

Анонимные функции (lambda)

В Python можно создавать функции в виде выражений с использованием ключевого слова lambda.

Такие функции называются анонимными или безымянными, потому что они не связываются с именем переменной, как функции, определенные через def.

Общий синтаксис: lambda аргументы: выражение. Выражение вычисляется, и его результат возвращается при вызове lambda-функции.

Lambda-функции могут принимать аргументы, включая аргументы со значениями по умолчанию.

Они полезны для создания небольших встроенных функций или для отложенного выполнения кода в местах, где требуется функция, но нецелесообразно определять ее отдельно с помощью def.

Встроенные функции для работы с последовательностями: map, filter, reduce

Эти функции являются инструментами функционального программирования и позволяют обрабатывать элементы последовательностей без явных циклов for в некоторых случаях.

* map()
  + Применяет заданную функцию к каждому элементу последовательности (например, списка).
  + Собирает результаты применения функции для каждого элемента.
  + В Python 3 возвращает итерируемый объект (не список напрямую, в отличие от Python 2), поэтому часто используется в сочетании с list() для получения списка результатов.
  + Часто используется с lambda-выражениями для определения функции, применяемой к элементам.
* filter()
  + Используется для отбора элементов из последовательности.
  + Принимает функцию, которая должна возвращать истину (True) или ложь (False) для каждого элемента.
  + В список результатов включаются только те элементы, для которых функция вернула истину.
  + Как и map, filter в Python 3 возвращает итерируемый объект.
  + Является эффективной встроенной альтернативой циклу for с условной проверкой и добавлением элементов в новый список.
* reduce()
  + Применяет функцию к элементам последовательности накопительно, чтобы свести последовательность к одному единственному значению.
  + В Python 3 функция reduce была перемещена в модуль functools и требует явного импорта (from functools import reduce).
  + На каждом шаге функция reduce передает результат предыдущего шага и следующий элемент последовательности функции, которая выполняет операцию (например, сложение или умножение).
  + По умолчанию первый элемент последовательности используется как начальное значение для накопительной операции.
  + Эквивалентна циклу, который накапливает результат, последовательно обрабатывая элементы списка.

**Глава 4**

Материал посвящен основам системного программирования в Python, рассматривая регулярные выражения, работу с файловой системой и взаимодействие с URL.

Регулярные выражения (модуль re)

Регулярные выражения — это мощный инструмент для поиска соответствий строк заданным шаблонам. В Python для этого используется модуль re.

Поиск соответствия: Основной метод — re.search(pattern, string). Он ищет первое вхождение шаблона в строке. Если найдено, возвращает объект соответствия (match object), иначе — None.

Объект соответствия: если re.search() находит соответствие, возвращается объект match. С помощью метода match.group() можно получить найденный текст.

"Сырые" строки: рекомендуется использовать "сырые" строки (r'...') для шаблонов регулярных выражений, чтобы обратные слэши не обрабатывались как escape-последовательности Python.

Базовые шаблоны:

Обычные символы (a, X, 9) соответствуют сами себе.

Метасимволы имеют специальное значение (., ^, $, \*, +, ?, {, }, [, ], \, , (, )).

. соответствует любому символу, кроме перевода строки (\n).

Специальные последовательности:

\w: соответствует букве, цифре или знаку подчеркивания ([A-Za-z0-9\_]).

\W: соответствует любому символу, кроме букв, цифр и знаков подчеркивания ([^a-zA-Z0-9\_]).

\s: соответствует "пробельному" символу (пробел, табуляция, новая строка и т.д. [ \t\n\r\f\v]).

\S: соответствует любому "не-пробельному" символу ([^ \t\n\r\f\v]).

\d: соответствует цифре.

\D: соответствует нечисловому символу ([^0-9]).

\b: соответствует границе слова (позиция между \w и \W символами или началом/концом строки).

\t, \n, \r: соответствуют табуляции, новой строке, возврату каретки.

^: начало строки.

$: конец строки.

\: экранирует специальные символы, заставляя их соответствовать самим себе (например, \. для точки).

Метасимволы повторения: применяются к предыдущему символу или группе.

\*: 0 или более повторений (жадный).

+: 1 или более повторений (жадный).

?: 0 или 1 повторение.

Наборы символов: используются квадратные скобки [] для указания набора символов, любой из которых может соответствовать. Метасимволы внутри [] теряют свое специальное значение. Можно указывать диапазоны ([a-z]) или инвертировать набор с помощью ^ в начале ([^ab] — любой символ, кроме 'a' или 'b').

Группы: Круглые скобки () создают логические группы. Они не влияют на сам поиск, но позволяют извлечь части найденного текста. match.group(1), match.group(2) и т.д. содержат текст, соответствующий содержимому соответствующей группы. match.group() или match.group(0) содержат всё найденное соответствие. Группы (?:...) являются незахватывающими и не попадают в результат извлечения.

re.findall(pattern, string): находит все непересекающиеся соответствия шаблону в строке и возвращает их в виде списка. Если шаблон содержит группы, возвращает список кортежей, где каждый кортеж соответствует одному полному совпадению, а его элементы — содержимому групп. Удобно использовать для поиска в содержимом целого файла.

Флаги: изменяют поведение регулярных выражений. Задаются как дополнительные аргументы функциям модуля re или методу compile().

re.I или re.IGNORECASE: игнорировать регистр.

re.M или re.MULTILINE: ^ и $ соответствуют началу/концу каждой строки, а не только всей обрабатываемой строки.

re.S или re.DOTALL: символ. соответствует всем символам, включая \n.

re.X или re.VERBOSE: разрешает использовать пробелы и комментарии (#) в шаблоне для улучшения читаемости.

Компиляция: если шаблон используется многократно, его можно скомпилировать с помощью re.compile(pattern, flags) для небольшого ускорения.

Отладка: рекомендуется тестировать шаблоны на небольших текстах, начиная с более слабых шаблонов, получая много соответствий, и затем постепенно их ужесточать.

Жадность: По умолчанию операторы повторения (\*, +) являются "жадными" — они пытаются захватить как можно больше текста. Добавление? (\*?, +?) делает их "не жадными", заставляя остановиться как можно раньше.

Замена: re.sub(pattern, replacement, string, count=0, flags=0) ищет все вхождения шаблона в строке и заменяет их на replacement. В строке replacement можно использовать \1, \2 и т.д. для вставки текста из соответствующих групп исходного совпадения.

Файловая система (модули os, os.path, shutil)

Модули os и os.path предоставляют функции для взаимодействия с файловой системой. Модуль shutil — для операций копирования и удаления.

os.listdir(dir): возвращает список имен файлов и директорий в указанной директории.

Создание/удаление директорий:

os.mkdir(dir\_path): создает одну директорию.

os.makedirs(dir\_path): создает все промежуточные директории в пути, если они не существуют.

os.rmdir(path): удаляет пустую директорию.

shutil.rmtree(path): удаляет директорию со всем ее содержимым (осторожно, необратимо!).

Работа с путями:

os.path.join(path1, path2, ...): объединяет части пути, учитывая особенности операционной системы.

os.path.abspath(path): преобразует относительный путь в абсолютный.

os.path.dirname(path): возвращает путь к директории из полного пути к файлу/директории.

os.path.basename(path): возвращает имя файла/директории из полного пути.

os.path.exists(path): проверяет, существует ли указанный путь (файл или директория).

Копирование файлов: shutil.copy(source\_path, dest\_path): копирует файл (целевая директория должна существовать).

Работа с URL (модуль urllib)

Модуль urllib.request позволяет работать с URL как с файлами, а urllib.parse — разбирать и собирать URL.

urllib.request.urlopen(url): открывает URL и возвращает файлоподобный объект, из которого можно читать данные методами. read(), .readlines() и т.д. Полученные данные являются байтовой строкой, которую нужно декодировать (например, .decode('utf-8')).

Метаинформация: Объект, возвращаемый urlopen, имеет методы для получения метаинформации:

.info(): возвращает объект с метаинформацией (заголовками).

.info().gettype(): возвращает MIME-тип (например, 'text/html').

.geturl(): возвращает базовый URL запроса (учитывая редиректы).

.getcode(): возвращает код ответа HTTP (например, 200 для успешного ответа).

Скачивание файлов:

urllib.request.urlretrieve(url, file\_name): скачивает URL и сохраняет содержимое в указанный файл.

Также можно использовать комбинацию urlopen и shutil.copyfileobj для скачивания.

Разбор и сборка URL:

urllib.parse.urlencode(query): преобразует словарь или список пар в строку параметров URL (например, ключ1=значение1&ключ2=значение2).

urllib.parse.urlparse(urlstring): разбирает URL на составные части (схема, сетевое расположение, путь, параметры, запрос, фрагмент).

**Глава 5**

Хорошо, вот конспект файла на русском языке, без ссылок на источники:

Программы на Python: Модули, Пакеты, Классы и Объекты

Модули Программа на Python обычно организована как набор файлов. Один файл главный, а другие выступают как модули – библиотеки инструментов (переменных, функций и других объектов). Чтобы использовать инструменты из модуля, его необходимо импортировать. Инструкция import module\_name загружает модуль, выполняя его код и создавая объекты. Инструменты модуля называются его атрибутами. Python ищет модули в путях поиска, включающих каталог программы, переменную окружения PYTHONPATH, стандартную библиотеку и другие места. Эти пути формируют список sys.path. Существует два основных способа импорта:

import module: загружает модуль целиком. Доступ к атрибутам осуществляется через module.attribute.

from module import name: загружает конкретные атрибуты из модуля. Доступ к ним осуществляется напрямую по имени name.

from module import \*: копирует все имена из модуля в текущую область видимости (используется реже). Инструкции импорта выполняются во время выполнения программы, а не на этапе компиляции. Модули представляют собой пространства имен. У каждого модуля есть специальные атрибуты, такие как \_\_name\_\_ (полное имя), \_\_doc\_\_ (строка документации), \_\_file\_\_ (путь к файлу) и, для пакетов, \_\_path\_\_ и \_\_package\_\_.

Пакеты — это способ организации модулей в директории. Директория может стать пакетом и быть импортирована как пространство имен, где поддиректории и модули являются ее атрибутами. Ключевое условие для пакета: каждая директория в пути импорта (кроме самой верхней) должна содержать файл с именем \_\_init\_\_.py. Этот файл автоматически выполняется при первом импорте пакета и используется для инициализации, например, настройки соединений с базами данных или создания структур данных.

Классы и Объекты В объектно-ориентированном подходе Python есть два типа объектов: объекты классов и объекты экземпляров. Классы выступают как фабрики, способные создавать множество экземпляров. Инструкция class создает объект класса. Присваивания внутри определения класса создают атрибуты класса. Функции, определенные внутри класса, называются методами класса. Метод класса всегда получает в первом аргументе (обычно называемом self) ссылку на объект экземпляра, для которого он вызван. Присваивания атрибутам через self создают атрибуты экземпляра. Вызов класса создает новый объект экземпляра. Экземпляры наследуют атрибуты класса. У каждого экземпляра есть свое собственное пространство имен. При поиске атрибута интерпретатор сначала ищет его в экземпляре, затем в классе экземпляра, а потом во всех его суперклассах. Наследование позволяет классам наследовать от других классов (называемых суперклассами), создавая иерархию классов. Подкласс наследует атрибуты суперкласса. Иерархия позволяет специализировать поведение в подклассах. Метод \_\_init\_\_ — это конструктор класса. Он вызывается автоматически при создании нового экземпляра и используется для его инициализации. Метод \_\_str\_\_ вызывается, когда объект преобразуется в строку для вывода, например, с помощью функции str.

Принципы ООП

Инкапсуляция: Сосредоточение логики операций внутри методов класса, скрывая детали реализации за интерфейсом. Это упрощает изменение кода, так как модификация требуется только в одном месте.

Наследование: позволяет адаптировать существующий код (классов и экземпляров), а не копировать его.

Вызов метода экземпляра, например instance.method(args), автоматически преобразуется интерпретатором в вызов метода класса, передавая экземпляр в качестве первого аргумента: Class.method(instance, args). Внутри подкласса можно явно вызвать метод суперкласса, включая конструктор \_\_init\_\_.

Для минимизации конфликтов имен используются соглашения: методы, предназначенные только для внутреннего использования в классе, часто начинаются с одного подчеркивания (\_method). Атрибуты с двойным подчеркиванием (\_\_attribute) получают автоматически искаженное имя, включающее имя класса, что делает их "псевдо-приватными".

**Глава 6**

Декораторы:

Декораторы — это обычные функции, которые принимают в качестве аргумента другую функцию.

Они используются для "обертывания" функции, добавляя некоторый код до или после ее выполнения, не изменяя саму функцию.

Самый удобный способ применить декоратор — использовать символ @ перед определением функции. Запись @decorator def func(): pass эквивалентна def func(): pass; func = decorator(func).

Можно использовать несколько декораторов для одной функции. Они применяются сверху вниз.

Декораторам можно передавать параметры. В этом случае декоратор представляет собой функцию, которая возвращает другой декоратор.

Декораторы можно использовать для методов классов. Это ничем не отличается от использования для обычных функций.

Существуют предопределённые декораторы для классов: @staticmethod (для статических методов, не получающих первым аргументом ни экземпляр, ни класс) и @classmethod (для методов класса, получающих первым аргументом сам класс).

Исключения:

В Python бывают два типа ошибок: ошибки синтаксиса и исключения.

Ошибки синтаксиса возникают, когда код грамматически неверен.

Исключения возникают во время выполнения программы, даже если синтаксис корректен (например, деление на ноль, обращение к неопределенной переменной, некорректное преобразование типов).

Исключения — это события, которые могут изменить ход выполнения программы. Они могут возбуждаться интерпретатором автоматически или программно самим кодом.

Для обработки исключений используются четыре инструкции:

try/except: Перехватывает исключения и выполняет код для восстановления.

try/finally: Выполняет завершающие операции независимо от возникновения исключения.

raise: Позволяет возбудить исключение программно.

assert: Позволяет возбудить исключение программно при выполнении определенного условия.

Полный формат try включает блоки try, except (может быть несколько для разных типов исключений, включая except NameError as err для доступа к объекту исключения, и общий except для всех остальных), else (выполняется, если в блоке try не возникло исключения) и finally (выполняется всегда при выходе из блока try).

Блок else в try полезен для выполнения кода только тогда, когда исключение *не* было возбуждено, что делает логику более очевидной.

Исключения можно возбуждать вручную с помощью инструкции raise. При этом можно передавать аргументы, которые доступны через атрибут .args у объекта исключения.

Можно создавать собственные типы исключений, наследуя их от класса Exception или его потомков.

Итераторы и Генераторы:

Итерация — это процесс считывания элементов объекта один за другим.

Объекты, по которым можно пройтись в цикле for...in..., называются итерируемыми объектами (например, списки, строки, файлы). Они удобны, но могут хранить все данные в памяти.

Итератор — это объект, представляющий поток данных. Он возвращает последовательные элементы при каждом вызове метода \_\_next\_\_() (или встроенной функции next()). Когда данных больше нет, итератор вызывает исключение StopIteration. Итератор также должен иметь метод \_\_iter\_\_(), который возвращает сам себя.

Протокол итератора требует наличия методов \_\_iter\_\_() и \_\_next\_\_().

Генераторы — это тоже итерируемые объекты, но они не хранят значения в памяти, а генерируют их на лету. Поэтому их обычно можно прочитать только один раз.

Генераторы создаются с помощью генераторных выражений или генераторных функций.

Генераторная функция выглядит как обычная функция, но содержит оператор yield, который возвращает следующее значение и временно приостанавливает выполнение функции, запоминая ее состояние (включая локальные переменные). При следующем запросе значения выполнение возобновляется с места последнего yield.

**Глава 7**

Здесь рассматриваются модули Python: itertools, collections, а также работа с форматами данных CSV, SQLite, JSON и XML.

Модуль itertools предоставляет специальные методы для работы с итерируемыми объектами.

Бесконечные последовательности:

itertools.count(start=0, step=1): возвращает бесконечный генератор, начинающийся со значения start с шагом step.

itertools.cycle(iterable): бесконечно повторяет элементы из заданной последовательности iterable.

itertools.repeat(element, times=None): Повторяет element. Если указано times, повторяет times раз. Без times повторяет бесконечно.

Контроль за итераторами:

itertools.takewhile(predicate, iterable): создает итератор, который возвращает элементы из iterable до тех пор, пока функция predicate возвращает истинное значение для текущего элемента.

itertools.islice(iterable, stop) / itertools.islice(iterable, start, stop[, step]): создает итератор, который возвращает выбранные элементы из iterable, подобно срезам списков, но работает с любыми итераторами.

Операции с итераторами:

itertools.permutations(iterable, r=None): возвращает итератор, генерирующий все возможные перестановки длины r (если r не указано, то длины исходной последовательности) из элементов iterable.

itertools.combinations(iterable, r): возвращает итератор, генерирующий все возможные комбинации длины r без повторений из элементов iterable.

itertools.groupby(iterable, key=None): создает итератор, возвращающий последовательность пар "ключ и группа элементов". Он последовательно перебирает iterable и создает новую группу каждый раз, когда значение ключа (определяемого функцией key или самим элементом, если key нет) меняется. Возвращаемая группа сама является итератором.

itertools.chain(\*iterables): создает итератор, который последовательно возвращает элементы из каждого переданного итератора iterables.

itertools.product(\*iterables, repeat=1): возвращает итератор, генерирующий декартово произведение входных итераторов. Аргумент repeat позволяет повторить итераторы несколько раз.

Модуль collections Модуль collections содержит специализированные типы данных, дополняющие встроенные типы.

Counter (Счетчик):

Это подкласс словаря, используемый для подсчета хешируемых объектов.

Создается из последовательности, словаря или с помощью именованных аргументов.

Метод update() позволяет добавлять элементы из другой последовательности или словаря.

При доступе к несуществующему элементу возвращает 0, а не вызывает ошибку KeyError.

Метод most\_common(n) возвращает список из n самых часто встречающихся элементов и их количества, отсортированный по убыванию количества.

Поддерживает операции как с множествами (+, -, & для пересечения, | для объединения).

OrderedDict (Отсортированный словарь):

Словарь, который запоминает порядок добавления своих элементов.

Если существующий элемент перезаписывается, его позиция не меняется. Удаление и последующая вставка элемента перемещает его в конец словаря.

При сравнении двух OrderedDict учитывается порядок элементов.

Метод move\_to\_end(key, last=True) перемещает элемент с ключом key в конец (по умолчанию) или в начало (last=False) словаря.

defaultdict (словарь со значением по умолчанию):

Подкласс словаря, который при попытке доступа к несуществующему ключу автоматически создает его со значением, возвращаемым функцией default\_factory, указанной при создании defaultdict.

namedtuple (именованный кортеж):

Фабричная функция для создания подклассов кортежей с именованными полями. Поля доступны как по индексу, так и по имени атрибута, что делает код более читаемым.

Создается с указанием имени типа (typename) и имен полей (field\_names, строка через пробел/запятую или список строк).

Метод \_make(iterable) создает экземпляр из последовательности значений.

Метод \_asdict() возвращает содержимое как OrderedDict.

Метод \_replace(\*\*kwargs) создает новый именованный кортеж с замененными значениями полей.

Атрибут \_fields содержит кортеж имен полей.

Можно объединять поля из разных namedtuple.

Работа с CSV CSV (Comma-Separated Values) — текстовый формат для представления табличных данных, где значения колонок разделены разделителем (часто запятой). Модуль csv облегчает работу с такими файлами.

Чтение:

Используется csv.reader(), который возвращает итератор по строкам файла. Каждая строка представлена списком строк (значений полей).

Можно указать dialect (диалект) или параметры чтения (например, delimiter, quotechar, quoting).

Диалекты:

Набор параметров, определяющих формат CSV (разделитель, символ кавычек и др.).

Стандартные диалекты (например, "excel") доступны.

Можно зарегистрировать собственный диалект с помощью csv.register\_dialect().

Доступные параметры диалекта включают delimiter, doublequote, escapechar, lineterminator, quotechar, quoting, skipinitialspace.

Значения параметра quoting определяют правила использования кавычек (QUOTE\_ALL, QUOTE\_MINIMAL, QUOTE\_NONNUMERIC, QUOTE\_NONE).

csv.Sniffer().sniff(data) может попытаться автоматически определить диалект файла по его содержимому.

Чтение в namedtuple:

Можно комбинировать csv.reader с collections.namedtuple и функцией map для чтения строк непосредственно в объекты именованных кортежей.

Работа со словарями (DictReader):

csv.DictReader считывает файл CSV, используя значения первой строки как ключи словаря. Каждая строка файла затем представляется как словарь.

Запись:

Используется csv.writer().

Метод writerow(row) записывает одну строку (которая может быть списком или кортежем).

Можно указать диалект или параметры записи при создании csv.writer.

Работа с SQLite SQLite — легковесная встраиваемая реляционная база данных, не требующая отдельного сервера. Модуль sqlite3 предоставляет интерфейс для работы с базами данных SQLite.

Подключение к базе данных или ее создание осуществляется функцией sqlite3.connect('database\_name.db'). Если файл не существует, он создается.

Для выполнения команд SQL используется объект соединения (connection) или курсор (cursor), который создается из соединения.

SQL-запросы выполняются методами execute() (для одной команды) или executemany() (для нескольких команд).

Запросы с параметрами используют заполнители (? Илит :name) для безопасной передачи данных.

Для выборки данных (SELECT) после выполнения запроса можно получить строки с помощью fetchone() (одна строка), fetchmany(size) (несколько строк) или fetchall() (все оставшиеся строки).

По умолчанию строки возвращаются как кортежи. Атрибут conn.row\_factory = sqlite3.Row позволяет получать строки как объекты Row, которые доступны как по индексу, так и по имени колонки.

Для сохранения изменений в базе данных после команд INSERT, UPDATE, DELETE или CREATE TABLE необходимо вызвать conn.commit(). В контекстном менеджере with sqlite3.connect(...) фиксация происходит автоматически при успешном выходе из блока.

Работа с JSON JSON (JavaScript Object Notation) — легкочитаемый текстовый формат обмена данными. Модуль json в Python позволяет кодировать структуры данных Python в JSON и декодировать JSON в структуры данных Python.

json.dumps(obj): Сериализует объект Python (obj) в строку формата JSON.

json.loads(string): Десериализует строку формата JSON (string) в объект Python.

Дополнительные параметры json.dumps():

sort\_keys=True: сортирует ключи словарей в JSON.

indent=n: форматирует вывод с отступами для лучшей читаемости.

separators=(',', ':'): указывает символы-разделители для плотной упаковки данных.

skipkeys=True: позволяет пропускать ключи словарей, которые не являются строками (по стандарту JSON ключи должны быть строками). Иначе будет вызвано исключение TypeError.

Работа с XML XML (eXtensible Markup Language) — язык разметки, используемый для структурирования данных. Стандартная библиотека Python включает модуль xml.etree.ElementTree для работы с XML-документами.

Парсинг документа:

ET.parse(source) парсит XML-файл.

tree.getroot() возвращает корневой элемент дерева.

Навигация по дереву:

Можно обходить дочерние элементы с помощью цикла for child in root:.

Доступ к элементам по индексу (например, root).

elem.tag возвращает имя тега, elem.attrib возвращает словарь атрибутов.

elem.text возвращает текстовое содержимое элемента.

Поиск элементов:

tree.iter(tag): возвращает итератор по всем элементам с указанным тегом во всем дереве.

elem.find(match): находит первого дочернего элемента, соответствующего пути или тегу.

elem.findall(match): находит все дочерние элементы, соответствующие пути или тегу.

elem.get(key): возвращает значение атрибута по имени.

Изменение документа:

Изменение текстового содержимого: elem.text = new\_text.

Изменение или добавление атрибутов: elem.set(key, value).

Удаление элементов: parent.remove(child).

Запись в файл:

tree.write(filename) записывает измененное дерево XML в файл.

Построение XML документов:

ET.Element(tag) создает новый элемент.

ET.SubElement(parent, tag) создает дочерний элемент и добавляет его к родителю.

ET.dump(elem) печатает XML-структуру элемента для отладки.