

Московский государственный технический университет
имени Н. Э. Баумана

В. И. Кузнецов

Язык программирования Python

*Методические указания к выполнению
лабораторной работы №7 по дисциплине
«Языки интернет программирования»*

Редакция от 1.09.2025

Факультет «Информатика и системы управления»
Кафедра «Компьютерные системы и сети»

Кузнецов В.И.

Язык программирования Python: методические указания к выполнению практикума № 7 и лабораторной работы №7 по дисциплине «Языки интернет-программирования» / В. И. Кузнецов.

В методическом пособии приведены основы языка программирования Python. Так же приведены синтаксиса и примеры использования языка.

Для студентов МГТУ им. Н.Э. Баумана, обучающихся по направлению «Информатика и вычислительная техника».

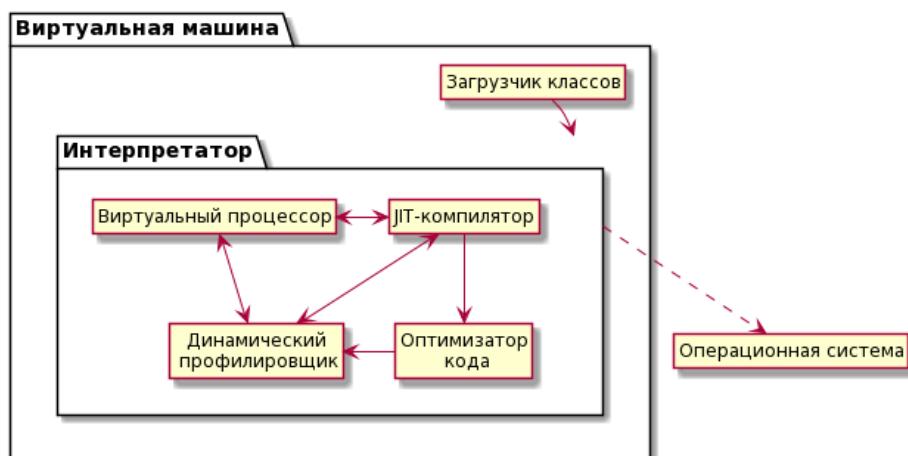
Теоретическая часть

Python — популярный высокоуровневый язык программирования для создания самых разных приложений: от веб-сервисов и игр до настольных программ и систем, работающих с базами данных. Особенно широко он используется в машинном обучении и исследованиях в области искусственного интеллекта.

Язык разработал голландский программист Гвидо ван Россум; первый анонс состоялся в 1991 году. Далее Python активно развивался: в 2000 году вышла ветка 2.0, в 2008 — 3.0. Между крупными релизами регулярно публикуются минорные обновления. На момент написания этого текста актуальной является версия 3.13, выпущенная в октябре 2024 года.

Ключевые особенности Python:

- Скриптовый язык: программы пишутся в виде текстовых скриптов.
- Поддерживает разные парадигмы программирования, включая объектно-ориентированную и функциональную.
- Интерпретируемый: для запуска кода требуется интерпретатор.
- Схема выполнения такая: мы создаём скрипт в текстовом редакторе, передаём его интерпретатору, который компилирует исходник в промежуточный байткод; затем виртуальная машина исполняет байткод, превращая его в инструкции, выполняемые операционной системой.
- Формально компиляция в байткод и его исполнение виртуальной машиной — разные этапы, но на практике они объединены в работе интерпретатора.



Синтаксис

Python отличается простым и понятным синтаксисом.

- Конец строки означает конец инструкции — ставить точку с запятой не нужно.
- Блоки формируются за счёт отступов. Величина отступа может быть любой, но внутри одного блока она должна быть одинаковой. Для лучшей читаемости используйте 4 пробела (в крайнем случае — табуляцию); один пробел — плохая идея.
- Составные (вложенные) конструкции записываются по общему правилу: заголовок заканчивается двоеточием, а под ним, с отступом, идёт соответствующий блок кода.

Основная инструкция:
Вложенный блок инструкций

Дополнительно:

- Несколько инструкций можно разместить в одной строке, разделяя их точкой с запятой, но лучше так не делать ради читаемости.

```
a = 1; b = 2; print(a, b)
```

- Одну инструкцию можно переносить на несколько строк, если обрамить её круглыми, квадратными или фигурными скобками.

```
if (a == 1 and b == 2 and  
    c == 3 and d == 4): # Не забываем про двоеточие  
    print('spam' * 3)
```

- Тело составной инструкции допускается писать в той же строке, что и её заголовок, но только если внутри нет других составных конструкций.

```
if x > y: print(x)
```

Стандартные операторы

Переменные

Переменные предназначены для хранения данных. Название переменной в Python должно начинаться с алфавитного символа или со знака подчеркивания и может содержать алфавитно-цифровые символы и знак подчеркивания. И кроме того, название переменной не должно совпадать с названием ключевых слов языка Python. Ключевых слов не так много, их легко запомнить:

```
False    await      else      import    pass
None     break      except    in       raise
True     class      finally   is       return
and      continue   for      lambda   try
as       def        from     nonlocal while
assert   del        global   not      with
async    elif       if       or       yield
```

В Python применяется два типа наименования переменных: camel case и snake case.

Camel case подразумевает, что каждое новое подслово в наименовании переменной начинается с большой буквы. Например:

```
userName = "Tom"
```

Snake notation подразумевает, что подслова в наименовании переменной разделяются знаком подчеркивания. Например:

```
user_name = "Tom"
```

И также надо учитывать регистрозависимость, поэтому переменные name и Name будут представлять разные объекты.

```
# две разные переменные
name = "Tom"
Name = "Tom"
```

Определив переменную, мы можем использовать в программе. Например, попытаться вывести ее содержимое на консоль с помощью встроенной функции print:

```
name = "Tom" # определение переменной name
print(name)   # вывод значения переменной name на консоль
```

Переменная хранит данные одного из типов данных. В Python существует множество различных типов данных. Базовые типы `bool`, `int`, `float`, `complex` и `str`.

Python является языком с динамической типизацией. А это значит, что переменная не привязана жестко к определенному типу.

Тип переменной определяется исходя из значения, которое ей присвоено. Так, при присвоении строки в двойных или одинарных кавычках переменная имеет тип `str`. При присвоении целого числа Python автоматически определяет тип переменной как `int`. Чтобы определить переменную как объект `float`, ей присваивается дробное число, в котором разделителем целой и дробной части является точка.

С помощью встроенной функции `type()` динамически можно узнать текущий тип переменной.

Логические операторы

Условная конструкция `if-elif-else` — главный механизм ветвления в Python. Она позволяет программе выбирать, какое действие выполнить, исходя из результата проверки условий и текущих значений переменных.

```
if test1:  
    state1  
elif test2:  
    state2  
else:  
    state3
```

- Любое число, не равное 0, или непустой объект - истина.
- Числа, равные 0, пустые объекты и значение `None` - ложь
- Операции сравнения применяются к структурам данных рекурсивно
- Операции сравнения возвращают `True` или `False`
- Логические операторы `and` и `or` возвращают истинный или ложный объект-операнд

Логические операторы:

`X and Y`

Истина, если оба значения X и Y истинны.

`X or Y`

Истина, если хотя бы одно из значений X или Y истинно.

```
not X
```

Истина, если X ложно.

Цикл while

While - один из самых универсальных циклов в Python, поэтому довольно медленный. Выполняет тело цикла до тех пор, пока условие цикла истинно.

```
i = 5
while i < 15:
    print(i)
    i = i + 2
```

Цикл for

Цикл for уже чуточку сложнее, чуть менее универсальный, но выполняется гораздо быстрее цикла while. Этот цикл проходится по любому итерируемому объекту (например строке или списку), и во время каждого прохода выполняет тело цикла.

```
for i in 'hello world':
    print(i * 2, end=' ')
...
hheelllloo  wwoorrlldd
```

Оператор continue

Оператор continue начинает следующий проход цикла, минуя оставшееся тело цикла (for или while)

```
for i in 'hello world':
    if i == 'o':
        continue
    print(i * 2, end=' ')
hheellllo  wwrrlldd
```

Оператор break

Оператор break досрочно прерывает цикл.

```
for i in 'hello world':
    if i == 'o':
        break
    print(i * 2, end=' ')
hheelll
```

Волшебное слово else

Слово `else`, примененное в цикле `for` или `while`, проверяет, был ли произведен выход из цикла инструкцией `break`, или же "естественным" образом. Блок инструкций внутри `else` выполнится только в том случае, если выход из цикла произошел без помощи `break`.

```
for i in 'hello world':  
    if i == 'a':  
        break  
else:  
    print('Буквы а в строке нет')
```

Функции

Функции представляют блок кода, который выполняет определенную задачу и который можно повторно использовать в других частях программы. В предыдущих статьях уже использовались функции. В частности, функция `print()`, которая выводит некоторое значение на консоль. Python имеет множество встроенных функций и позволяет определять свои функции. Формальное определение функции:

```
def имя_функции ([параметры]):  
    инструкции
```

Одни функции могут определяться внутри других функций - внутренние функции еще называют локальными. Локальные функции можно использовать только внутри той функции, в которой они определены.

Функция может возвращать результат. Для этого в функции используется оператор `return`, после которого указывается возвращаемое значение:

```
def имя_функции ([параметры]):  
    инструкции  
    return возвращаемое_значение
```

Лямда-выражения

Лямда-выражения в языке Python представляют небольшие анонимные функции, которые определяются с помощью оператора `lambda`. Формальное определение лямбда-выражения:

```
message = lambda: print("hello")  
  
message() # hello
```

Здесь лямбда-выражение присваивается переменной `message`. Это лямбда-выражение не имеет параметров, ничего не возвращает и просто

выводит строку "hello" на консоль. И через переменную message мы можем вызвать это лямбда-выражение как обычную функцию.

Объект

Python имеет множество встроенных типов, например, int, str и так далее, которые мы можем использовать в программе. Но также Python позволяет определять собственные типы с помощью классов. Класс представляет некоторую сущность. Конкретным воплощением класса является объект.

Можно еще провести следующую аналогию. У нас у всех есть некоторое представление о человеке, у которого есть имя, возраст, какие-то другие характеристики. Человек может выполнять некоторые действия - ходить, бегать, думать и т.д. То есть это представление, которое включает набор характеристик и действий, можно назвать классом. Конкретное воплощение этого шаблона может отличаться, например, одни люди имеют одно имя, другие - другое имя. И реально существующий человек будет представлять объект этого класса.

В языке Python класс определяется с помощью ключевого слова class:

```
class название_класса:  
    атрибуты_класса  
    методы_класса
```

Для создания объекта класса используется конструктор. Конструктор по умолчанию, который не принимает параметров и который неявно имеют все классы. Однако мы можем явным образом определить в классах конструктор с помощью специального метода, который называется `__init__()` (по два прочерка с каждой стороны). К примеру, изменим класс Person, добавив в него конструктор:

```
class Person:  
    # конструктор  
    def __init__(self):  
        print("Создание объекта Person")  
  
tom = Person()      # Создание объекта Person
```

Деструктор представляет собой метод `__del__(self)`, в который, как и в конструктор, передается ссылка на текущий объект. В деструкторе

определяются действия, которые надо выполнить при удалении объекта, например, освобождение или удаление каких-то ресурсов, которые использовал объект.

Исключения

При программировании на Python мы можем столкнуться с двумя типами ошибок. Первый тип представляют синтаксические ошибки (syntax error). Они появляются в результате нарушения синтаксиса языка программирования при написании исходного кода. При работе в какой-либо среде разработки, например, в PyCharm, IDE сама может отслеживать синтаксические ошибки и каким-либо образом их выделять.

Второй тип ошибок представляют ошибки выполнения (runtime error). Они появляются в уже в процессе выполнения программы. Подобные ошибки еще называются исключениями.

При возникновении исключения работа программы прерывается, и чтобы избежать подобного поведения и обрабатывать исключения в Python есть конструкция try..except.

try..except

Конструкция try..except имеет следующее формальное определение:

```
try:  
    инструкции  
except [Тип_исключения] :  
    инструкции
```

Весь основной код, в котором потенциально может возникнуть исключение, помещается после ключевого слова try. Если в этом коде генерируется исключение, то работа кода в блоке try прерывается, и выполнение переходит в блок except.

После ключевого слова except опционально можно указать, какое исключение будет обрабатываться (например, ValueError или KeyError). После слова except на следующей строке идут инструкции блока except, выполняемые при возникновении исключения.

При обработке исключений также можно использовать необязательный блок finally. Отличительной особенностью этого блока является то, что он выполняется вне зависимости, было ли сгенерировано исключение.

Как правило, блок finally применяется для освобождения используемых ресурсов, например, для закрытия файлов.

Стоит отметить, что блок finally не обрабатывает исключения, и если мы используем этот блок без блока except, то при возникновении ошибки приложение аварийно завершится, как в следующем случае при делении числа на ноль.

Структуры данных

Для работы с наборами данных Python предоставляет такие встроенные типы как списки, кортежи и словари.

Список (list) представляет тип данных, который хранит набор или последовательность элементов. Во многих языках программирования есть аналогичная структура данных, которая называется массив.

Списки

Для создания списка применяются квадратные скобки [], внутри которых через запятую перечисляются элементы списка. Например, определим список чисел:

```
numbers = [1, 2, 3, 4, 5]
```

Подобным образом можно определять списки с данными других типов, например, определим список строк:

```
people = ["Tom", "Sam", "Bob"]
```

Также для создания списка можно использовать функцию-конструктор list():

```
numbers1 = []
numbers2 = list()
```

Оба этих определения списка аналогичны - они создают пустой список.

Кортежи

Кортеж (tuple) представляет последовательность элементов, которая во многом похожа на список за тем исключением, что кортеж является неизменяемым (immutable) типом. Поэтому мы не можем добавлять или удалять элементы в кортеже, изменять его.

Для создания кортежа используются круглые скобки, в которые помещаются его значения, разделенные запятыми:

```
tom = ("Tom", 23)
print(tom)      # ("Tom", 23)
```

Также для определения кортежа мы можем просто перечислить значения через запятую без применения скобок:

```
tom = "Tom", 23
print(tom)      # ("Tom", 23)
```

Если вдруг кортеж состоит из одного элемента, то после единственного элемента кортежа необходимо поставить запятую:

```
tom = ("Tom",)
```

Для создания кортежа из другого набора элементов, например, из списка, можно передать список в функцию tuple().

Словари

Словарь (dictionary) в языке Python хранит коллекцию элементов, где каждый элемент имеет уникальный ключ и ассоциированое с ним некоторое значение.

Определение словаря имеет следующий синтаксис:

```
dictionary = { ключ1:значение1, ключ2:значение2, ... }
```

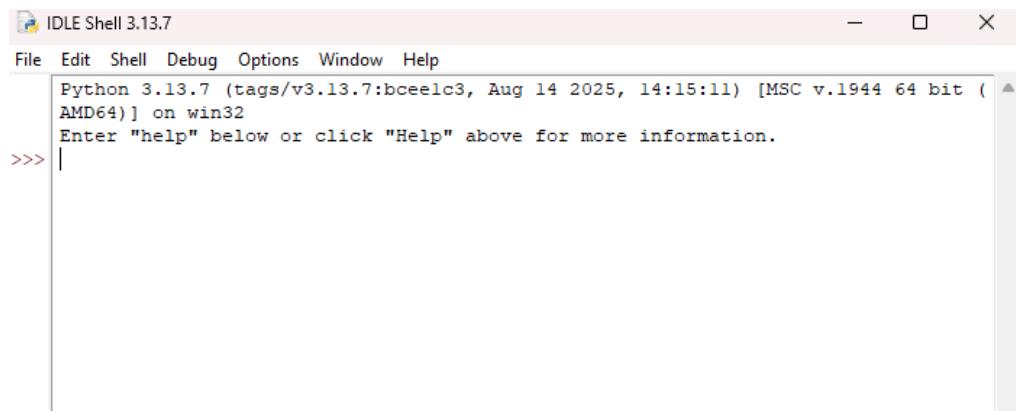
В фигурных скобках через запятую определяется последовательность элементов, где для каждого элемента сначала указывается ключ и через двоеточие его значение.

Практическая часть

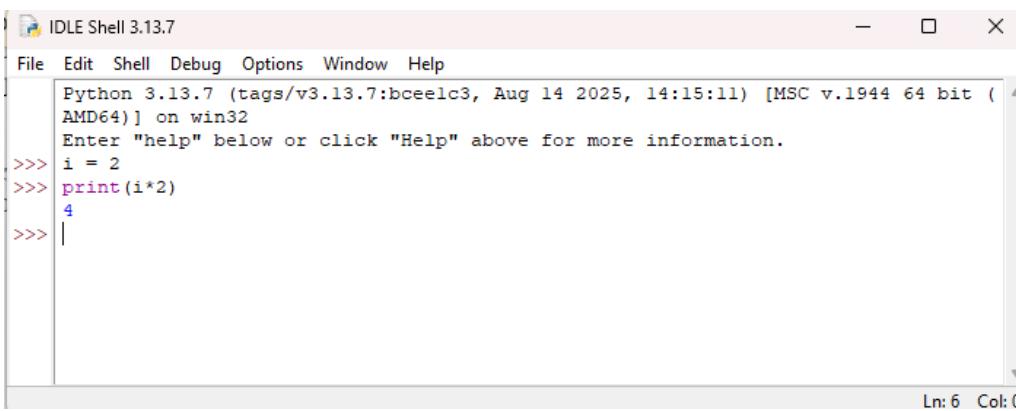
Для работы с языком программирования Python необходимо установить интерпретатор языка. Скачать его можно через [официальный сайт](#) для всех операционных систем или через пакетный менеджер для конкретных операционных систем.

Для разработки можно использовать любую стороннюю IDE: PyCharm или VsCode, но также можно использовать базовую поставляемую вместе с интерпретатором называемая IDLE. Дальнейшая работа с языком будет рассматриваться на её примере.

Язык python интерпретируемый и из-за особенностей работы позволяет запускать программу по отдельным строкам. При старте IDLE открывается консоль, в которой можно исполнять функции языка.



The screenshot shows the title bar "IDLE Shell 3.13.7" and a menu bar with "File", "Edit", "Shell", "Debug", "Options", "Window", and "Help". Below the menu is a status bar showing "Python 3.13.7 (tags/v3.13.7:bceelc3, Aug 14 2025, 14:15:11) [MSC v.1944 64 bit (AMD64)] on win32". The main area displays the text "Enter "help" below or click "Help" above for more information." followed by a red "">>>>" prompt and an empty line for input.



The screenshot shows the same window setup as the first one. In the main area, there is a sample Python session:
>>> i = 2
>>> print(i*2)
4

A status bar at the bottom right indicates "Ln: 6 Col: 0".

The screenshot shows the IDLE Shell 3.13.7 interface. The title bar reads "IDLE Shell 3.13.7". The menu bar includes File, Edit, Shell, Debug, Options, Window, and Help. The main window displays a Python script being run in a shell environment:

```
AMD64) on win32
Enter "help" below or click "Help" above for more information.

>>> i = 2
>>> print(i*2)
4
>>> while(i<5):
...     print(i)
...     i += 1
...
2
3
4
```

The status bar at the bottom right indicates "Ln: 13 Col: 0".

Метод реализации программы через консоль удобен в случае разового запуска простых скриптов. Для удобной отладки, декомпозиции больших проектов или необходимости повторного запуска в IDLE присутствует режим работы с файлами с программным кодом.

The screenshot shows the IDLE Shell 3.13.7 interface with the "File" menu open. The menu bar includes File, Edit, Shell, Debug, Options, Window, and Help. The "File" menu contains the following items:

- New File (Ctrl+N)
- Open... (Ctrl+O)
- Open Module... (Alt+M)
- Recent Files (with a submenu arrow)
- Module Browser (Alt+C)
- Path Browser
- Save (Ctrl+S)
- Save As... (Ctrl+Shift+S)
- Save Copy As... (Alt+Shift+S)
- Print Window (Ctrl+P)
- Close Window (Alt+F4)

The status bar at the bottom right indicates "Ln: 13 Col: 0".

The screenshot shows the IDLE Editor window with the title "*untitled*". The menu bar includes File, Edit, Format, Run, Options, Window, and Help. The main editor area contains the following Python code:i = 2
print (i*2)
while(i<5):
 print(i)
 i += 1

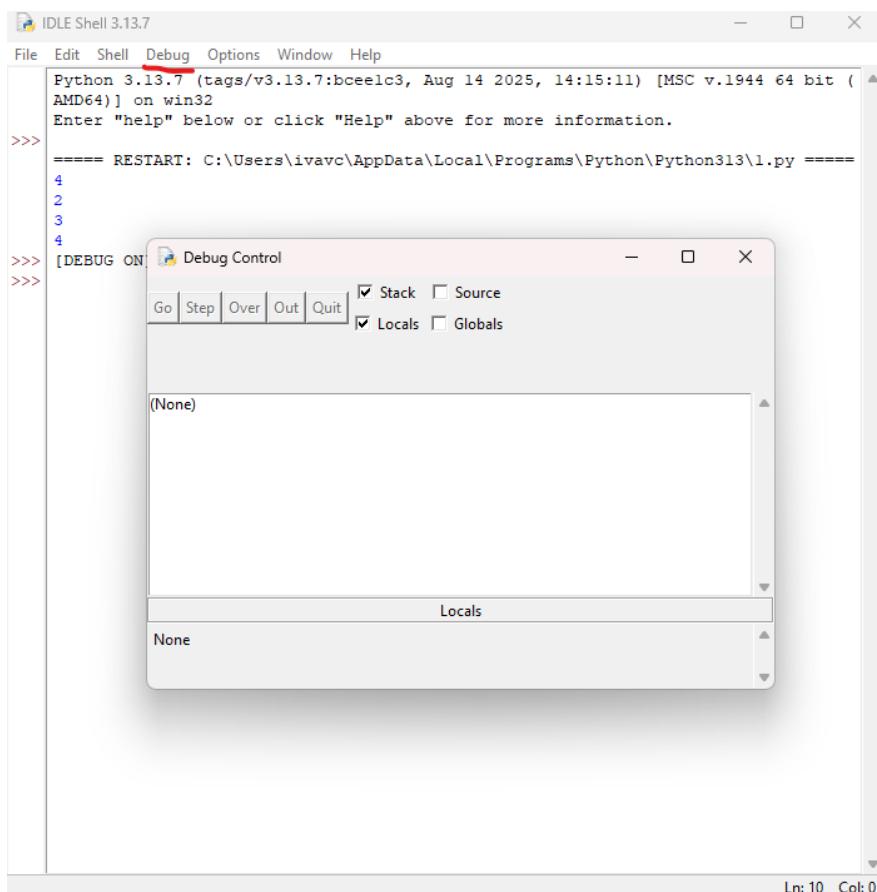
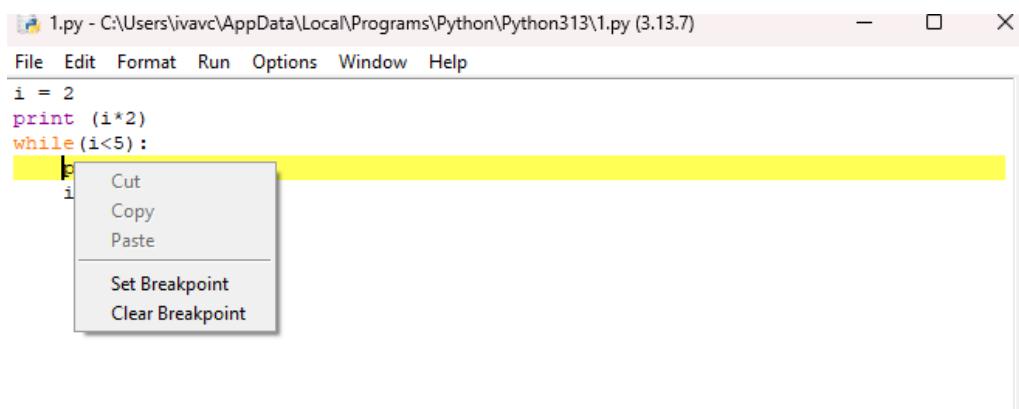
Работа программы будет выполняться по строчкам внутри интерпретатора, но код запуститься последовательно в консоли интерпретатора.

IDLE Shell 3.13.7

```
File Edit Shell Debug Options Window Help
Python 3.13.7 (tags/v3.13.7:bceelc3, Aug 14 2025, 14:15:11) [MSC v.1944 64 bit (AMD64)] on win32
Enter "help" below or click "Help" above for more information.

>>> ===== RESTART: C:\Users\ivavc\AppData\Local\Programs\Python\Python313\1.py =====
4
2
3
4
>>> |
```

IDLE при простоте среды так же имеет средства отладки. Точку останова можно установить нажатием правой кнопкой мыши на интересуемой, а режим отладки включается в окне консоли.



Задания

Задание 1 Топ-N слов из файла

Что сделать: написать функцию/CLI, которая читает текстовый файл и выводит N самых частых слов.

Требования:

- Нормализуйте регистр и очистите пунктуацию (учтите Unicode).
- Игнорируйте короткие слова длиной < 3
- Используйте collections.Counter.

Задание 2 Слияние списка словарей с накоплением значений

Что сделать: есть список словарей вида {'user': 'ann', 'sum': 120}. Соберите итоговый словарь {'ann': 420, 'bob': 300, ...}.

Требования:

- Используйте defaultdict(int) или Counter.
- Обработайте пропуски полей и некорректные типы (тихая фильтрация или try/except).

Задание 3 Контекст-менеджер timer(name)

Что сделать: реализовать контекст-менеджер, который измеряет время выполнения блока и печатает [{name}] took 12.3 ms.

Требования:

- Реализовать класс с __enter__ / __exit__ или через contextlib.contextmanager.
- Использовать time.perf_counter().
- Поддержать вывод в произвольный поток (sys.stdout по умолчанию).

Контрольные вопросы

1. Что такое список (list) в Python, и чем он отличается от кортежа (tuple)?
2. Что такое функции высшего порядка и как они используются в Python?
3. Объясните разницу между операторами == и is в Python.
4. Что такое декораторы и как они работают?
5. Как работает механизм обработки исключений, и зачем использовать try / except / else / finally?