2025-09-17

Императивное программирование на Python

Императивное программирование — парадигма программирование, которая подход к структуре программы как последовательному набору инструкций.

В императивном программирование важны два концепта:

- именованные переменные;
- оператор присваивания

> **Note**
>
> Эти два понятия необходимы для корректного хранения и использования промежуточных результатов
> Одна инструкция выполнилась, мы получили ее результат — и передали следующей инструкции

Также важны понятия:

- составных выражений
- функции (подпрограммы) — для повторяющихся наборов инструкций

Лучший курс по Python 0: Мета информация - YouTube

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

import os
import json

if os.path.exists('./spoon.json'):
    with open('./spoon.json', 'r', encoding='utf-8') as f:
        my_data = json.load(f)  # именованная переменная и оператор
присваивания
        print(my_data)  # вызов встроенной функции
else:
    raise FileNotFoundError('There is no spoon.json')
```

Области видимости переменных в Python

Локальная область видимости

Локальная область видимости **совпадает с телом функции**, в которой определена переменная. Мы можем обратиться к переменной в локальной области видимости из любой точки этой функции, но никак не можем вне границ функции

```
def foo(x: int) -> float:
    return x / 2

def bar(x: int) -> float:
    a = x + 2
    b = a ** 2
    return b / 3

def boo(x: int) -> float:
    a = x / 4
    return a ** 5

if __name__ == '__main__':
    print(boo(4)) # вернет 1.0
    print(a) # вернет ошибку NameError: name 'a' is not defined
```

Вложенная область видимости (Enclosing или Nested)

Область видимости функции **относительно ее дочерних функций**.

```
def outer(x: int) -> float:
    a = x / 4

    def inner() -> None:
        b = f'25% от {x} равно {a}'
        print(b)
    inner()
    print(b) # здесь будет ошибка
    return a ** 5

if __name__ == '__main__':
    print(outer(4)) # вывалится с ошибкой NameError: name 'b' is not
defined, т.к. внешняя функция не знает о переменных внутренней, а вот
внутренняя за счет вложенной области видимости все видит
```

Если нужно переприсвоить значение переменной во вложенной области видимости (т.е. присвоить что-либо во внутренней функции переменной из внешней функции), то нужно использовать ключевое слово `nonlocal`

```
def outer_func_nonlocal() -> None:
    outer_var = 100
    print(f'Outer func: {outer_var=}') # выведет Outer func: outer_var=100
    def inner_func():
```

```
        nonlocal outer_var
        print(f'Inner func: {outer_var=}')
        outer_var = 200
    inner_func()  # выведет Inner func: outer_var=100
    print(f'Outer func: {outer_var=}')  # выведет Outer func: outer_var=200
```

Глобальная область видимости

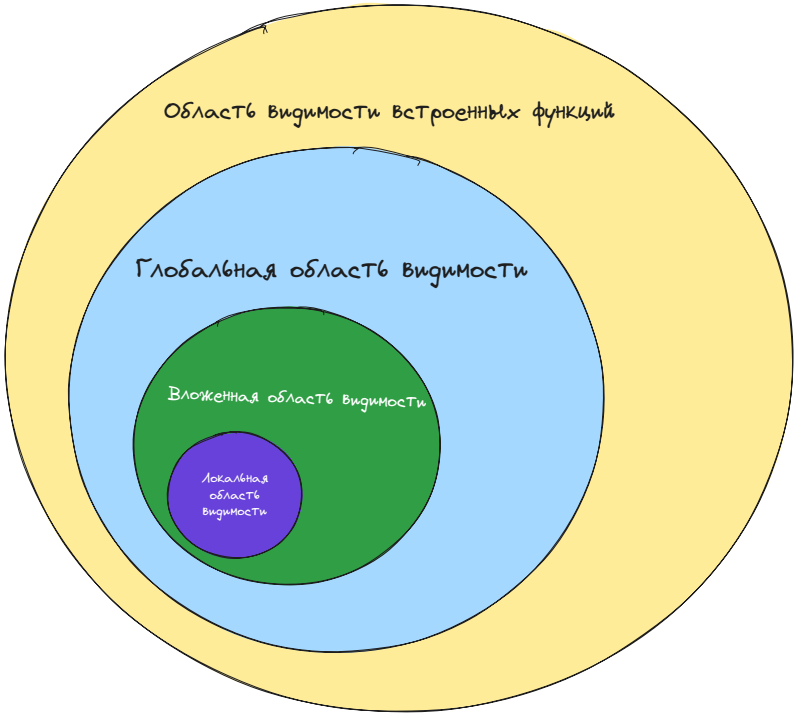
Доступ к таким переменным могут получить любые другие переменные, объекты и функции.

Если внутри функции мы переприсваиваем значение переменной с тем же именем, что и глобальная переменная, по умолчанию создается локальная переменная с тем же именем.

Если нужно изменить глобальную переменную внутри функции, то необходимо использовать ключевое слово `global`

```
y:int = 20

def foo() -> float:
    return y / 2

def bar() -> float:
    a = y + 2
    b = a ** 2
    return b / 3

def boo() -> float:
    global y
    y /= 4
    return y

if __name__ == '__main__':
    print(foo())  # y == 20; вернет 10.0
    print(boo())  # y == 5.0; вернет 5.0
    print(bar())  # y == 5.0; вернет 16.333333333333332
```

Резюмируя области видимости



1. встроенные функции, типы данных и коллекции (`__builtins__`)
2. глобальная область видимости (все могут получить доступ к глобальным переменным, объектам и методам)
3. вложенная область видимости (область видимости внешней функции относительно дочерних функций)
4. локальная область видимости (к этим переменным и объектам имеют доступ только переменные и объекты внутри тела функции или класса, в котором они определены)

Стилистические рекомендации по оформлению кода на Python

Императивное программирование является де-факто парадигмой Python по умолчанию. Стилистическое руководство, изложенное в одном из ранних Python Enhancement Proposals (PEP), описывает принципы оформления хорошо читаемого кода на Питоне с применением указанной парадигмы.

PEP 8 – Style Guide for Python Code | peps.python.org

Для облегчения проверки на соответствие стилистическим рекомендациям активно применяются линтеры:

- Pylint
- ruff
- Flake8
- mypy

Лабораторная работа №1: Основы императивного программирования в Python

Цель: освоить базовые концепции императивного стиля (переменные, ветвления, циклы, функции) и работы со структурами данных (строки, списки, множества, словари) через разработку консольных приложений с обязательной обработкой пользовательского ввода.

Общие требования ко всем программам:

1. **Обработка некорректного ввода:** программы должны устойчиво работать при любом пользовательском вводе (например, ввод букв вместо цифр, слишком короткое слово, неверный символ), запрашивать данные повторно и не завершаться с ошибкой.
2. **Чистота кода:** код должен быть читаемым: использовать осмысленные имена переменных, добавлять комментарии, разделять логические блоки.
3. **Функциональная декомпозиция:** решение должно быть разбито на функции. Главный код (`if __name__ == "__main__":`) должен быть кратким.

Задание 1: Быки и коровы

Правила: компьютер загадывает число из `n` уникальных цифр. Игрок пытается его угадать, предлагая свои варианты. За каждую догадку он получает ответ:

- Корова — цифра угадана и стоит на правильной позиции.
- Бык — цифра угадана, но стоит на неправильной позиции.

Требования к реализации:

1. Реализовать классическую игру для числа из 4 цифр.
2. Добавить возможность выбора уровня сложности — длины загадываемого числа (3, 4 или 5 цифр).
3. После каждой угаданной цифры вести статистику по сессии: количество сыгранных игр, минимальное/максимальное/среднее число попыток.
4. После окончания игры спрашивать, хочет ли пользователь сыграть еще.

Пример

```
Загадано число из 4 цифр.
Ваш вариант: 1234
Найдено 1 коров и 2 быков
Ваш вариант: 1567
Найдено 0 коров и 1 бык
...
Ваш вариант: 1932
```

```
Вы угадали число 1932 за 5 попыток.

Хотите сыграть еще? (да/нет): да
...
Всего игр сыграно: 2
Лучший результат: 5 попыток
Худший результат: 8 попыток
Средний результат: 6.5 попыток
```

Задание 2: Анализатор чисел

Правила: программа должна анализировать введенное пользователем целое число.

Требования к реализации:

1. Запросить у пользователя целое число N ($N > 0$). Обработать некорректный ввод.
2. Найти и вывести все делители числа N.
3. Проверить и вывести, является ли число N простым (делится только на 1 и на себя).
4. Проверить и вывести, является ли число N совершенным (равно сумме всех своих делителей, кроме себя самого).

Пример

```
Введите целое число больше 0: 28
Делители числа 28: [1, 2, 4, 7, 14, 28]
Число 28 не является простым
Число 28 является совершенным (1+2+4+7+14=28)
```

Задание 3: Wordle/5 букв

Правила: компьютер загадывает слово из 5 букв. У игрока есть 6 попыток его угадать. После каждой попытки программа дает обратную связь:

- Буква угадана и стоит на правильном месте — выделяется `[X]`.
- Буква есть в слове, но стоит на другом месте — выделяется `(X)`.
- Буквы нет в слове — она остается без изменений.

Важно: реализовать корректную логику для случаев с повторяющимися буквами.

Требования к реализации:

1. Использовать заранее подготовленный список слов из 5 букв (например, `['лотос', 'столп', 'комод']`).

2. Реализовать вывод подсказок после каждой попытки в формате, указанном выше.
3. Предусмотреть 6 попыток. Если слово не угадано — показать загаданное слово.

Пример

```
Загадано слово из 5 букв. У вас 6 попыток.
Попытка 1: полет
Результат: (п) [о] (л) (е) (т)
Попытка 2: лотос
Результат: [л] [о] [т] [о] [с]
Вы угадали слово "лотос" за 2 попытки!
```

Задание 4: Камень-Ножницы-Бумага-Ящерица-Спок

Правила: Расширенная версия игры. Выбор компьютера делается случайно.

- Ножницы режут бумагу.
- Бумага накрывает камень.
- Камень давит ящерицу.
- Ящерица травит Спока.
- Спок ломает ножницы.
- Ножницы убивают ящерицу.
- Ящерица ест бумагу.
- Бумага подставляет Спока.
- Спок испаряет камень.
- Камень затупляет ножницы.

Требования к реализации:

1. Реализовать игру против компьютера, который делает случайный выбор.
2. Вести счет побед между компьютером и пользователем.
3. Реализовать возможность играть до определенного количества побед (например, до 3).

Пример

```
До скольки побед играем? 3
Ваш ход (камень/ножницы/бумага/ящерица/спок): спок
Ход компьютера: камень
Спок испаряет камень! Вы победили!
Счет: Вы - 1, Компьютер - 0
...
```

Задание 5: Статистика текста

Правила: программа должна анализировать введенный пользователем текст и предоставлять статистику.

Требования к реализации:

1. Запросить у пользователя произвольный текст (не менее 100 символов).
2. Реализовать следующие функции анализа:

- Подсчет общего количества символов (с пробелами и без)
- Подсчет количества словоформ
- Поиск 5 самых частых словоформ и их частоту
- Поиск 5 самых длинных словоформ
- Подсчет средней длины словоформы

3. Реализовать обработку текста: приведение к нижнему регистру, удаление знаков препинания.

Пример

```
Введите текст для анализа (не менее 100 символов):
> "В Python есть несколько структур данных для хранения коллекций.
Список — это изменяемая последовательность элементов. Кортеж — это
неизменяемая последовательность. Словарь — это коллекция пар ключ-
значение."

Результаты анализа:
Общее количество символов: 145 (без пробелов: 121)
Количество словоформ: 21
Самые частые словоформы:
- 'это': 3 раза
- 'последовательность': 2 раза
- 'коллекций': 1 раз
- 'хранения': 1 раз
- 'несколько': 1 раз
Самые длинные словоформы:
- 'последовательность' (17 букв)
- 'изменяемая' (10 букв)
- 'неизменяемая' (12 букв)
- 'коллекция' (9 букв)
- 'структур' (8 букв)
Средняя длина словоформы: 7.2 символа
```