2024-09-18

Программирование (Pyton) — оргвопросы

Трек 1: Хочу выучить Питон

1. Посещение занятий
2. Сдача лабораторных работ
3. Выполнение практик
4. Подготовка и сдача итогового задания

Трек 2: Хочу сдать дисциплину, при этом Питон знаю или выучу самостоятельно

1. Работа с преподавателем в режиме консультации
2. Использование Python как основного языка разработки программного продукта (MVP), которым предполагается отчитаться по дисциплине
3. Итоговый проект должен отвечать ряду требований:
    1. должно быть ТЗ (по ЕСПД) или хотя бы функциональные требования в отдельном документе;
    2. должен быть разработан проект с описанием архитектуры, API (если предусмотрено), желательно в какой-либо нотации, например, UML или BPMN
    3. разработка строго с использованием системы контроля версий с майлстоунами каждые 2-4 недели
    4. должны быть написаны автотесты
    5. проект должен быть подготовлен к развертыванию через средства CI/CD (минимум — Docker + Docker Compose)

И итоговый проект, и итоговое задание должны быть защищены в конце курса по аналогии с тем, как защищаются заявки на конкурсах поддержки от ФСИ и аналогичных фондов

Императивное программирование на Python

Императивное программирование — парадигма программирования, которая предполагает 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

```
#!/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(boo(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():
    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.

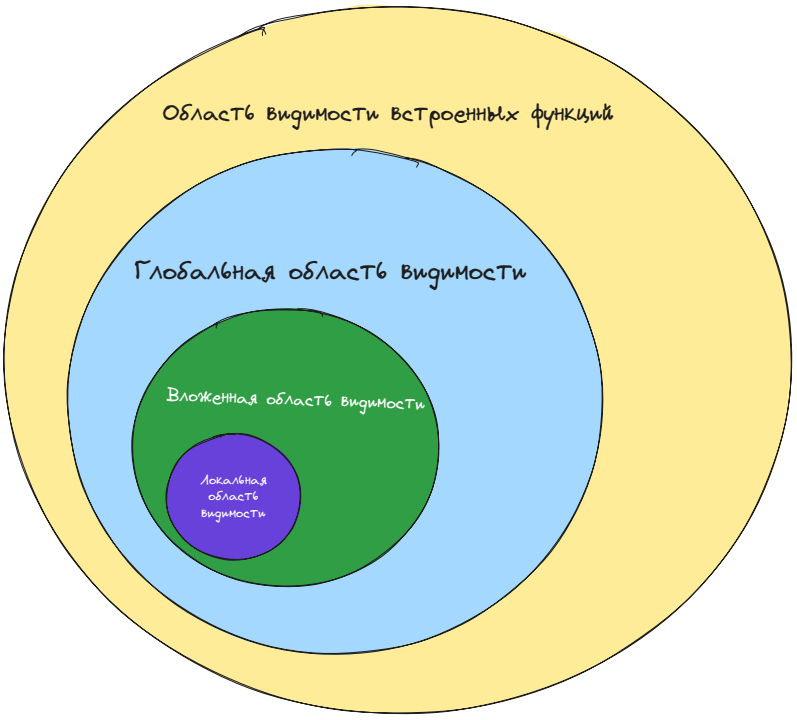
Если внутри функции мы переприсваиваем значение переменной с тем же именем, что и глобальная переменная, по умолчанию создается локальная переменная с тем же именем.

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

Практические задания

1. Написать консольную игру «Быки и коровы» (Быки и коровы — Википедия)
    1. Случайным образом загадать 4-значное число с уникальными цифрами!
    2. Просить пользователя угадывать его
    3. Для каждой цифры, которую пользователь угадал на правильном месте, говорить ему, что у него есть 1 корова
    4. Если пользователь угадал цифру, но не угадал ее позицию, говорить ему, что у него есть 1 бык
    5. Каждую попытку сообщать, сколько быков и коров у пользователя
    6. В конце сказать, за сколько ходов пользователь угадал число

Example

Допустим, сгенерировалось число 1932
Пользователь пишет догадку: 1234
Мы говорим ему: 2 коровы и 1 бык
Далее он пишет 1256
Мы говорим: 1 корова и 1 бык
...

2. Написать консольную игру «Wordle»/«5 букв» (выделяем правильные буквы `[]`, а буквы на неправильной позиции `()`)
3. Написать игру Камень-ножницы-бумага и/или Камень-ножницы-бумага-ящерица-Спок

4. Написать функцию поиска всех делителей целого числа больше 0 и проверки этого числа на то, является ли оно простым (ограничим до 100 тыс.)