МИНОБРНАУКИ РОССИИ
федеральное государственное бюджетное образовательное учреждение высшего образования
“Московский государственный технологический университет “СТАНКИН””
(ФГБОУ ВО МГТУ “СТАНКИН”)

Институт
информационных технологий

Кафедра
информационных систем

Отчет по лабораторной работе №2

по дисциплине “**Веб-программирование**”
на тему: “Основы языка Python”

Студент **Абдурафики А.**
группы ИДБ-20-06

_____
подпись

Проверил _____ **Сакович С.**
подпись

Москва 2022 г.

GitHub: https://github.com/Abdusharif0095/Laboratory-work-2

ОГЛАВЛЕНИЕ

ГЛАВА 1. ВВЕДЕНИЕ

Python - высокоуровневый язык программирования общего назначения с динамической строгой типизацией и автоматическим управлением памятью, ориентированный на повышение производительности разработчика, читаемости кода и его качества, а также на обеспечение переносимости написанных на нём программ. Язык является полностью объектно-ориентированным в том плане, что все является объектами. Необычной особенностью языка является выделение блоков кода пробельными отступами. Синтаксис ядра языка минималистичен, за счёт чего на практике редко возникает необходимость обращаться к документации. Сам же язык известен как интерпретируемый и используется в том числе для написания скриптов.

ГЛАВА 2. ВЫПОЛНЕНИЕ ЛАБОРАТОРНОЙ РАБОТЫ

Цель лабораторной работы: рассмотреть основные функции и возможности языка программирования Python.

2.1 Решение задачи №1

Задача

Написать функцию, которая на вход принимает int и возвращает true или false в зависимости является ли это число палиндром. Число является палиндромом, если оно читается справа налево и слева направо одинаково.

Решение

```
def palindrome(num: int) -> bool:
   return (str(abs(num)) == "".join(reversed(str(abs(num)))))
```

2.2 Решение задачи №2

Задача

Написать функцию, которая принимает на вход список из положительных.

целочисленных элементов и возвращает три списка: (25)

a. в первом - числа, которые делятся на 2.

b. во втором - числа, которые делятся на 3.

c. с третьем - числа, которые делятся на 5.

Решение

```
def lists(nums):
   lst = [[], [], []]
   mods = [2, 3, 5]

   for num in nums:
       for i in range(3):
           if num % mods[i] == 0:
               lst[i].append(num)

   return lst[0], lst[1], lst[2]
```

2.3 Решение задачи №3

Задача

Написать функцию, принимающую на вход int, и число, обратное этому int.

Решение

```
def reverse_num(num: int) -> int:
```

```
    return (-(num < 0) + (num > 1)) *
int(("".join(reversed(str(abs(num))))))
```

2.4 Решение задачи №4

Задача

Написать функцию, которая будет рассчитывать квадратный корень n-ой степени методом Ньютона (https://ru.wikipedia.org/wiki/ Алгоритм нахождения корня n-ной степени)

Решение

```
def NewtonMethod(a, n):
    xk = 1
    xk1 = (1 / n) * ((n - 1) * xk + a / xk ** (n - 1))
    while abs(xk1 - xk) > 0.00001:
        xk = xk1
        xk1 = (1 / n) * ((n - 1) * xk + a / xk ** (n - 1))

    return xk1
```

2.5 Решение задачи №5

Задача

Написать функцию, принимающую 1 аргумент — число от 0 до 100000, и возвращающую true, если оно простое, false если нет.

Решение

```
def is_prime(num: int) -> bool:
    for i in range(int(num ** (1 / 2) + 1)):
        if num % i == 0:
            return False

    return True
```

2.6 Решение задачи №6

Задача

Написать декоратор, который будет кэшировать результат вызова функции и отдавать его при последующих вызовах данной функции.

Решение

```
import functools


def cache_function(func):
    @functools.wraps(func)
    def wrapper(*args, **kwargs):
        global cache_map
```

```
        args_repr = [repr(a) for a in args]
        kwargs_repr = [f"{k}={v!r}" for k, v in kwargs.items()]
        key = ", ".join(args_repr + kwargs_repr)
        if key in cache_map:
            return cache_map[key]
        else:
            value = func(*args, **kwargs)
            cache_map[key] = value
            return value

    return wrapper
```

Пример кэширования функции:

```
@cache_function
def factorial(n: int) -> int:
    return n * factorial(n - 1) if n >= 1 else 1
```

2.7 Решение задачи №7

Задача

Усложненный вариант - написать тот же самый декоратор, но с параметром, который будет показывать сколько раз отдавать кешируемый результат. Если данный счетчик обнуляется, то выполняем функцию и вновь кэшируем ее результат.

Решение

```
def cache_function_cnt(func):
    @functools.wraps(func)
    def wrapper(*args, **kwargs):
        global cnt
        global cache_map_cnt
        global cache_map_val
        args_repr = [repr(a) for a in args]
        kwargs_repr = [f"{k}={v!r}" for k, v in kwargs.items()]
        key = ", ".join(args_repr + kwargs_repr)
        if key in cache_map_cnt and cache_map_cnt[key] != 0:
            cache_map_cnt[key] -= 1
            return cache_map_val[key]
        else:
            value = func(*args, **kwargs)
            cache_map_val[key] = value
            cache_map_cnt[key] = cnt - 1
            return value

    return wrapper
```

Пример кэширования функции:

```
@cache_function_cnt
def factorial2(n: int) -> int:
    return n * factorial2(n - 1) if n >= 1 else 1
```

ГЛАВА 3. ВЫВОД

В процессе выполнения лабораторной работы я изучил основные функции и возможности языка программирования Python.