Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт цифрового развития
Кафедра инфокоммуникаций

ОТЧЕТ
ПО ЛАБОРАТОРНОЙ РАБОТЕ №2.12
дисциплины «Основы кроссплатформенного программирования»

Выполнила:
Мурашко Анастасия Юрьевна
1 курс, группа ИТС-б-о-22-1,
11.03.02 «Инфокоммуникационные технологии и системы связи»,
направленность (профиль) «Инфокоммуникационные системы и сети», очная форма обучения

_____
(подпись)

Руководитель практики:
Воронкин Р. А., доцент кафедры инфокоммуникаций

_____
(подпись)

Отчет защищен с оценкой _____ Дата защиты_____

Ставрополь, 2023 г.

Тема: Декораторы в языке Python

Цель: приобретение навыков по работе с декораторами функций при написании программ с помощью языка программирования Python версии 3.x.

Ход работы:

Пример №1:

```
def decorator_function(func):
    def wrapper():
        print('Функция-обёртка!')
        print('Оборачиваемая функция: {}'.format(func))
        print('Выполняем обёрнутую функцию...')
        func()
        print('Выходим из обёртки')

    return wrapper


@decorator_function
def hello_world():
    print('Hello world!')


hello_world()
```

Рисунок 1. Код

```
C:\Users\MonstR\PycharmProjects\pythonProject\venv\Scripts\python.exe "D:/User
Функция-обёртка!
Оборачиваемая функция: <function hello_world at 0x0000018F80C8DD30>
Выполняем обёрнутую функцию...
Hello world!
Выходим из обёртки

Process finished with exit code 0
```

Рисунок 2. Результат выполнения

Вариант№9

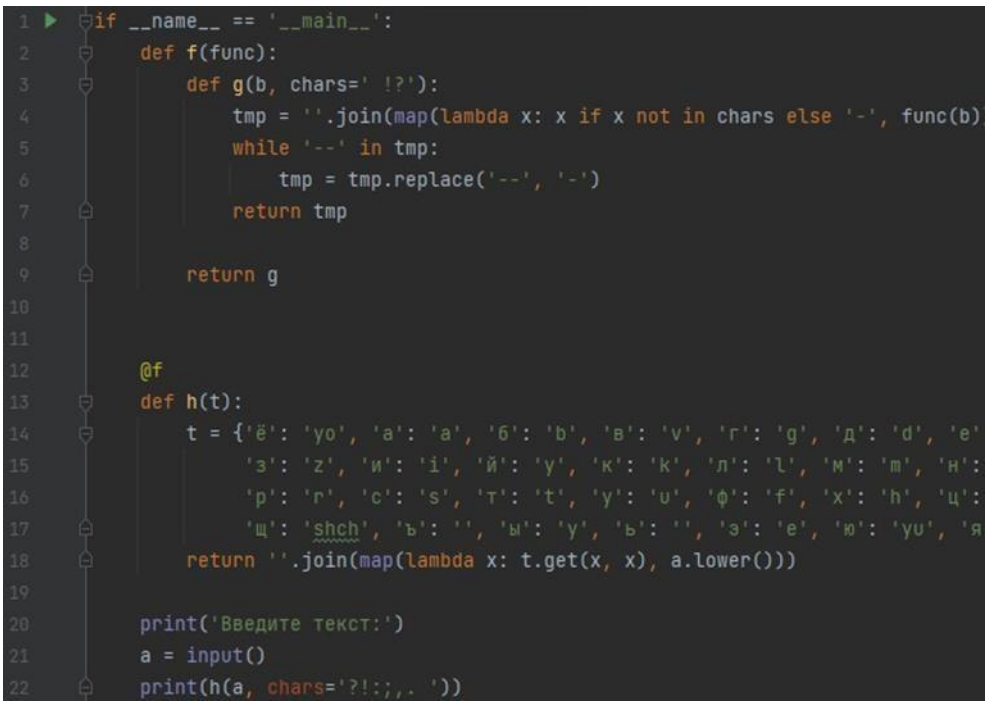
Индивидуальное задание:

9. Объявите функцию, которая принимает строку на кириллице и преобразовывает ее в латиницу, используя следующий словарь для замены русских букв на соответствующее латинское написание:

```
t = {'ё': 'yo', 'а': 'a', 'б': 'b', 'в': 'v', 'г': 'g', 'д': 'd', 'е': 'e',
'ж': 'zh',
'з': 'z', 'и': 'i', 'й': 'y', 'к': 'k', 'л': 'l', 'м': 'm', 'н': 'n', 'о':
'o', 'п': 'p',
'р': 'r', 'с': 's', 'т': 't', 'у': 'u', 'ф': 'f', 'х': 'h', 'ц': 'c', 'ч':
'ch', 'ш': 'sh',
'щ': 'shch', 'ъ': '', 'ы': 'y', 'ь': '', 'э': 'e', 'ю': 'yu', 'я': 'ya'}
```

Функция должна возвращать преобразованную строку. Замены делать без учета регистра (исходную строку перевести в нижний регистр – малые буквы). Определите декоратор с параметром `chars` и начальным значением `" !?"`, который данные символы преобразует в символ `"-"` и, кроме того, все подряд идущие дефисы (например, `"–"` или `"---"`) приводит к одному дефису. Полученный результат должен возвращаться в виде строки. Примените декоратор со значением `chars="?!:;, . "` к функции и вызовите декорированную функцию. Результат отобразите на экране.

Рисунок 3. Условия задания



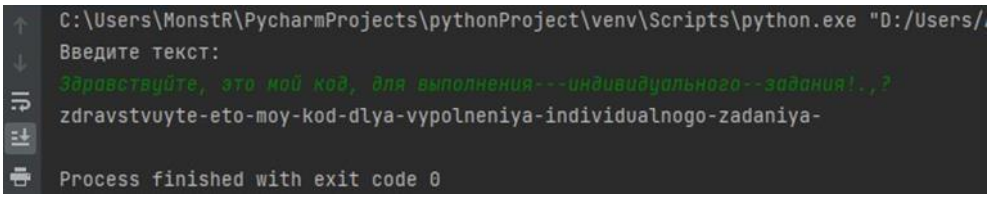
```
if __name__ == '__main__':
    def f(func):
        def g(b, chars=' !?'):
            tmp = ''.join(map(lambda x: x if x not in chars else '-', func(b)
            while '--' in tmp:
                tmp = tmp.replace('--', '-')
            return tmp

        return g


    @f
    def h(t):
        t = {'ё': 'yo', 'а': 'a', 'б': 'b', 'в': 'v', 'г': 'g', 'д': 'd', 'е'
             'з': 'z', 'и': 'i', 'й': 'y', 'к': 'k', 'л': 'l', 'м': 'm', 'н':
             'р': 'r', 'с': 's', 'т': 't', 'у': 'u', 'ф': 'f', 'х': 'h', 'ц':
             'щ': 'shch', 'ъ': '', 'ы': 'y', 'ь': '', 'э': 'e', 'ю': 'yu', 'я
        return ''.join(map(lambda x: t.get(x, x), a.lower()))

    print('Введите текст:')
    a = input()
    print(h(a, chars='?!:;,. '))
```

Рисунок 4. Код



```
C:\Users\MonstR\PycharmProjects\pythonProject\venv\Scripts\python.exe "D:/Users/
Введите текст:
Здравствуйте, это мой код, для выполнения---индивидуального--задания!.,?
zdravstvuyte-eto-moy-kod-dlya-vypolneniya-individualnogo-zadaniya-

Process finished with exit code 0
```

Рисунок 5. Результат выполнения

Ответы на контрольные вопросы:

1. Что такое декоратор?

Декоратор – это функция, которая позволяет обернуть другуюфункцию для расширения её функциональности без непосредственно изменения её кода.

2. Почему функции являются объектами первого класса?

Мы можем сохранять функции в переменные, передавать их в качестве аргументов и возвращать из других функций. Можно даже определить одну функцию внутри другой.

3. Каково назначение функций высших порядков?

Это такие функции, которые могут принимать в качестве аргументов и возвращать другие функции.

4. Как работают декораторы?

Из определения: декоратор - это функция, которая позволяет обернуть другую функцию для расширения её функциональности без непосредственного изменения её кода. На примере кода:

```
def decorator_function(func):
    def wrapper():
        print('Функция-обёртка!')
        print('Оборачиваемая функция: {}'.format(func))
        print('Выполняем обёрнутую функцию...')
        func()
        print('Выходим из обёртки')
    return wrapper
```

5. Какова структура декоратора функций?

Функция декоратор, содержащая в себе декорируемую функцию.

6. Самостоятельно изучить как можно передать параметры декоратору,а не декорируемой функции?

Функциональный вызов func(...), который вернет что-то тоже вызываемое или имя функции, или переменная или экземпляр класса.

Вывод: Приобрела навыки по работе с декораторами функций при написании программ с помощью языка программирования Python версии3.x.