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 №15
дисциплины «Программирование на Python»

Выполнил:
Середа Кирилл Витальевич
1 курс, группа ИВТ-б-о-22-1,
09.03.01 «Информатика и
вычислительная техника», очная
форма обучения

_____
(подпись)

Руководитель практики:
Воронкин Роман Александрович

_____
(подпись)

Отчет защищен с оценкой _____ Дата защиты_____

Ставрополь, 2023 г.

Тема: Декораторы функций в языке Python

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

Ход выполнения:

1) Проработал пример 1

```
Функция-обёртка!
Оборачиваемая функция: <function hello_world at 0x7fd4f01dd280>
Выполняем обёрнутую функцию...
Hello world!
Выходим из обёртки
```

Рисунок 1 – Результат работы программы

2) Проработал пример 2

```
[*] Время выполнения: 1.308683156967163 секунд.
<!doctype html><html itemscope="" itemtype="http://schema.org/WebPage" lang="ru"><head><meta content
```

Рисунок 2 – Результат работы программы

3) Индивидуальное задание 1 – Вариант 1: 1 Объявите функцию с именем get_sq , которая вычисляет площадь прямоугольника по двум параметрам: width и height – ширина и высота прямоугольника и возвращает результат. Определите декоратор для этой функции с именем (внешней функции) func_show , который отображает результат на экране в виде строки (без кавычек): "Площадь прямоугольника: <значение>". Вызовите декорированную функцию get_sq .

```
Введите ширину прямоугольника: 12
Введите высоту прямоугольника: 3
Площадь прямоугольника: 36.0
```

Рисунок 3 – Результат работы программы

Код программы:

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

def func_show(inner_func):
    def wrapper(*args, **kwargs):
        result = inner_func(*args, **kwargs)
        print(f"Площадь прямоугольника: {result}")
        return result
    return wrapper
```

```
@func_show
def get_sq(width, height):
    return width * height

if __name__ == "__main__":
    try:
        width = float(input("Введите ширину прямоугольника: "))
        height = float(input("Введите высоту прямоугольника: "))

        get_sq(width, height)

    except ValueError:
        print("Ошибка: Введите числовые значения для ширины и высоты.")
```

Ответы на вопросы:

1. Что такое декоратор? Декоратор — это функция, которая позволяет обернуть другую функцию для расширения её функциональности без непосредственного изменения её кода. Декораторы можно рассматривать как практику метапрограммирования, когда программы могут работать с другими программами как со своими данными.

2. Почему функции являются объектами первого класса? Объектами первого класса в контексте конкретного языка программирования называются элементы, с которыми можно делать всё то же, что и с любым другим объектом: передавать как параметр, возвращать из функции и присваивать переменной. С функцией все это делать можно, поэтому ее и можно назвать объектом первого класса.

3. Каково назначение функций высших порядков? Функции высших порядков — это такие функции, которые могут принимать в качестве аргументов и возвращать другие функции.

4. Как работают декораторы? Пример:

```
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('Привет, мир!')
```

Здесь decorator_function() является функцией-декоратором. Она является функцией высшего порядка, так как принимает функцию в качестве

аргумента и возвращает функцию. Внутри decorator_function() определена другая функция, которая обёртывает функцию-аргумент и затем изменяет её поведение. Декоратор возвращает эту обёртку.

5. Какова структура декоратора?

```
def decorator(func):
    def wrapper(*args, **kwargs):
        # Код до вызова целевой функции
        result = func(*args, **kwargs)  # Вызов целевой функции
        # Код после вызова целевой функции
        return result
    return wrapper
```

6. Самостоятельно изучить как можно передать параметры декоратору, а не декорируемой функции? В Python можно передавать параметры декоратору, добавляя еще один уровень вложенности.

```
7. def decorator_with_parameters(param1, param2):
       def actual_decorator(func):
           def wrapper(*args, **kwargs):
               print(f"Decorator parameters: {param1}, {param2}")
               result = func(*args, **kwargs)
               return result
           return wrapper
       return actual_decorator
```

Вывод: В результате выполнения работы были приобретены навыки по работе с декораторами функций при написании программ с использованием языка программирования Python версии 3.x.