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»

Вариант 3

Выполнил:
Болуров Ислам Расулович
2 курс, группа ИВТ-б-о-22-1,
09.03.01 «Информатика и вычислительная техника»,
направленность (профиль) «Программное обеспечение средств вычислительной техники и автоматизированных систем», очная форма обучения

_____
(подпись)

Руководитель практики:
Воронкин Р. А.

_____
(подпись)

Отчет защищен с оценкой _____ Дата защиты_____

Ставрополь, 2023

Тема: Декораторы функций в языке Python

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

Пример 1. Раз мы знаем, как работают функции высших порядков, теперь мы можем понять, как работают декораторы. Сначала посмотрим на пример декоратора:

Листинг:

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

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!')


if __name__ == "__main__":
    hello_world()
```

```
C:\Users\User\PycharmProjects\Python_laba_15\venv\Scripts\python.exe
Функция-обёртка!
Оборачиваемая функция: <function hello_world at 0x0000029643979F80>
Выполняем обёрнутую функцию...
Hello world!
Выходим из обёртки

Process finished with exit code 0
```

Рисунок 1. Результат программы

Пример 2. Используем аргументы и возвращаем значения.

Листинг:

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

def benchmark(func):
    import time
```

```
    def wrapper(*args, **kwargs):
        start = time.time()
        return_value = func(*args, **kwargs)
        end = time.time()
        print('[*] Время выполнения: {} секунд.'.format(end-start))
        return return_value
    return wrapper


@benchmark
def fetch_webpage(url):
    import requests
    webpage = requests.get(url)
    return webpage.text[:100]


if __name__ == "__main__":
    webpage = fetch_webpage('https://google.com')
    print(webpage)
```

Индивидуальное задание. Объявите функцию, которая вычисляет периметр многоугольника и возвращает вычисленное значение. Длины сторон многоугольника передаются в виде коллекции (списка или кортежа). Определите декоратор для этой функции, который выводит на экран сообщение: «Периметр фигуры равен = ». Примените декоратор к функции и вызовите декорированную функцию.

Листинг:

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

def perimeter_decorator(func):
    def wrapper():
        sides = []
        n = int(input("Введите количество сторон многоугольника: "))
        for i in range(n):
            side = float(input(f"Введите длину стороны {i + 1}: "))
            sides.append(side)
        result = func(sides)
        print("Периметр фигуры равен =", result)

    return wrapper
```

```
C:\Users\User\PycharmProjects\Python_laba_15\venv\Scripts\python
Введите количество сторон многоугольника: 5
Введите длину стороны 1: 1
Введите длину стороны 2: 5
Введите длину стороны 3: 8
Введите длину стороны 4: 9
Введите длину стороны 5: 10
Периметр фигуры равен = 33.0

Process finished with exit code 0
```

Рисунок 3. Результат программы

Вывод: в ходе выполнения данной лабораторной работы были приобретены навыки по взаимодействию с декораторами функций при написании программ с помощью языка программирования Python версии 3.x.

Ответы на контрольные вопросы

1. Декоратор - это функция, которая принимает одну функцию в качестве аргумента и возвращает другую функцию. Он позволяет изменить поведение функции, добавив к ней дополнительный функционал, без изменения ее исходного кода.

2. Функции в Python являются объектами первого класса, потому что они могут быть присвоены переменным, переданы другим функциям в качестве аргументов, возвращены из другой функции в качестве результата, а также созданы и изменены динамически во время выполнения программы.

3. Функции высших порядков предназначены для работы с другими функциями как с данными. Они могут принимать функции в качестве аргументов, возвращать функции в качестве результатов и использовать функции как переменные. Это позволяет строить абстракции и обобщать поведение функций, делая код более гибким и переиспользуемым.

4. Декораторы работают путем обертывания (wrapping) декорируемой функции внутри декоратора. Когда декорируемая функция вызывается, декоратор выполняет дополнительный код перед или после вызова декорируемой функции. Это позволяет изменять поведение функции без изменения кода самой функции.

5. Структура декоратора функций в Python выглядит следующим образом:

```
def decorator(func):
    def wrapper(*args, **kwargs):
        # Код, выполняемый перед вызовом функции
        result = func(*args, **kwargs)
        # Код, выполняемый после вызова функции
        return result
```

```
    return wrapper
```

6. Чтобы передать параметры декоратору, а не декорируемой функции, можно добавить дополнительный уровень обертывания в декораторе. Например:

```
def decorator_with_parameters(param1, param2):
    def decorator(func):
        def wrapper(*args, **kwargs):
            # Использование параметров декоратора
            print("param1 =", param1)
            print("param2 =", param2)
            result = func(*args, **kwargs)
            return result
        return wrapper
    return decorator

@decorator_with_parameters("param1_value", "param2_value")
def decorated_function():
    # Код декорируемой функции
    pass
```

В этом примере декоратор `decorator_with_parameters` принимает параметры `param1` и `param2`, которые передаются при использовании декоратора `@decorator_with_parameters("param1_value", "param2_value")`. Эти параметры могут быть использованы внутри декоратора `wrapper`, который в свою очередь обертывает исходную функцию `decorated_function`.