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 31

Выполнил:
Репкин Александр Павлович
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., канд. техн. наук,
доцент кафедры инфокоммуникаций

_____
(подпись)

Отчет защищен с оценкой _____ Дата защиты_____

Ставрополь, 2023 г.

Тема: Декораторы функций в языке Python.

Цель: приобрести навыки работы с декораторами функций при написании программ с помощью языка программирования Python версии 3.x.

Порядок выполнения работы:

1. Выполнены примеры лабораторной работы. В первом примере был показан способ создания и применения декоратора. Второй пример создавался с аналогичной целью, однако в нём также показана возможность импортирования как в декораторе, так и в декорируемой функции.

```
Функция-обёртка!
Оборачиваемая функция: <function hello_world at 0x00000226819C60D0>
Выполняем обёрнутую функцию...
Hello world!
Выходим из обёртки
```

Рисунок 1. Полученный результат выполнения первого примера.

```python
#!/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()
```

Рисунок 2. Код первого примера.

```
[*] Время выполнения: 0.6949737071990967 секунд.
<!doctype html><html itemscope="" itemtype="http://schema.org/WebPage" lang="ru"><head><meta content="&#1055;&#1086;&#1080;&#1089;&#1082; &#1080;&#1085;&#1092;&#1086;&#1088;&#1084;&#1072;&#1094;&#1080;&#1080; &#1074; &#1080;&#1085;&#1090;&#1077;&#1088;&#1085;&#1077;&#1090;&#1077;: &#1074;&#1077;&#1073; &#1089;&#1090;&#1088;&#1072;&#1085;&#1080;&#1094;&#1099;, &#1082;&#1072;&#1088;&#1090;&#1080;&#1085;&#1082;&#1080;, &#1074;&#1080;&#1076;&#1077;&#1086; &#1080; &#1084;&#1085;&#1086;&#1075;&#1086;&#1077; &#1076;&#1088;&#1091;&#1075;&#1086;&#1077;." name="description"><
```

Рисунок 3. Полученный результат выполнения второго примера.

```python
#!/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


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

Рисунок 4. Код второго примера.

2. Выполнено индивидуальное задание. В нём требовалось объявить функцию с именем get_sq, вычисляющую площадь прямоугольника по параметрам width и height, после чего возвращает результат. Также, требовалось определить декоратор для этой функции с именем func_show, отображающий результат на экране: "Площадь прямоугольника: <значение>". Этот декоратор применялся к get_sq.

```
Good day, user! Please, enter height: 5.2
Great, now we need width: 4
Rectangle's square:  20.8
```

Рисунок 5. Полученный результат индивидуального задания.

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

def func_show(func):
    """Декоратор, выводящий «Площадь прямоугольника:<значение>»"""

    def wrapper(width, height):
        print("Rectangle's square: ", func(width, height))

    return wrapper


@ func_show
def get_sq(width, height):
    """Вычисление площади прямоугольника."""

    return width * height


if __name__ == "__main__":
    height = float(input("Good day, user! Please, enter height: "))
    width = float(input("Great, now we need width: "))
    get_sq(width, height)
```

Рисунок 6. Полученный код индивидуального задания.

3. Ответы на вопросы.

1) Что такое декоратор?

Ответ: декоратор – функция, которая позволяет обернуть другую функцию для расширения её функциональности без непосредственного изменения её кода. Вот почему декораторы можно рассматривать как практику метапрограммирования, когда программы могут работать с другими программами как со своими данными.

2) Почему функции являются объектами первого класса?

Ответ: в Python всё является объектами, благодаря этому можно сохранять функции в переменные, передавать их в качестве аргументов и возвращать из других функций. Можно даже определить одну функцию внутри другой. Иными словами, функции – это объекты первого класса (Объектами первого класса называются элементы, с которыми можно делать всё то же, что и с любым другим объектом: передавать как параметр, возвращать из функции и присваивать переменной).

3) Каково назначение функций высших порядков?

Ответ: функции высших порядков могут принимать в качестве аргументов (и возвращать) другие функции.

4) Как работают декораторы?

Ответ: декоратор — функция, позволяющая обернуть другую функцию для расширения её функциональности без непосредственного изменения её кода.

5) Какова структура декоратора функций?

Ответ: пример структуры декоратора:

```
def simple_decorator(func):
    def wrapper():
        ...
    return wrapper.
```

6) Как можно передать параметры декоратору, а не декорируемой функции?

Ответ: для передачи параметров декоратору, а не декорируемой функции, можно создать функцию-обёртку для декоратора, принимающую необходимые параметры и возвращающую сам декоратор.

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