МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

ИНСТИТУТ ЦИФРОВОГО РАЗВИТИЯ

Отчет по лабораторной работе №15
«Декораторы функций в языке Python»

по дисциплине «Основы программной инженерии»

Выполнила:
Первых Дарья Александровна,2 курс, группа ПИЖ-б-о-20-1,
Проверил:
Доцент кафедры инфокоммуникаций,
Воронкин Р.А.

Ставрополь, 2021 г.

ВЫПОЛНЕНИЕ

```
def hello_world():
    print('Hello world!')
```

Рисунок 1 – Пример работы с областью видимости local

```
>>> def hello_world():
...     print('Hello world!')
...
>>> type(hello_world)
<class 'function'>
>>> class Hello:
...     pass
...
>>> type(Hello)
<class 'type'>
>>> type(10)
<class 'int'>
>>>
```

Рисунок 2 – Пример работы с областью видимости enclosing

```
>>> hello = hello_world
>>> hello()
```

Рисунок 3 – Пример работы с функцией

```
>>> def wrapper_function():
...     def hello_world():
...         print('Hello world!')
...     hello_world()
...
>>> wrapper_function()
Hello world!
```

Рисунок 4 – Пример работы с функцией mul5

```
>>> def higher_order(func):
...     print('Получена функция {} в качестве аргумента'.format(func))
...     func()
...     return func
...
>>> higher_order(hello_world)
Получена функция <function hello_world at 0x000001DA1C2EA170> в качестве аргумента
Hello world!
<function hello_world at 0x000001DA1C2EA170>
```

Рисунок 5 – Пример замыкания

```
>>> 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
```

Рисунок 6 – Пример функции с использованием локальных и глобальных переменных

```
>>> @decorator_function
... def hello_world():
...     print('Hello world!')
...
>>> hello_world()
Функция-обёртка!
Оборачиваемая функция: <function hello_world at 0x000001DA1C2EA4D0>
Выполняем обёрнутую функцию...
Hello world!
Выходим из обёртки
```

Рисунок 7 – Пример работы с замыканием, как средством для построения иерархических данных

```
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
```

fetch_webpage()

main

```
C:\Users\podar\study\anaconda\envs\LR15\python.exe "C:/Users/podar/study/PyCharm
[*] Время выполнения: 1.0455541610717773 секунд.
```

Рисунок 8 – Пример задания

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

@benchmark
def fetch_webpage():
    import requests
    webpage = requests.get('https://google.com')
```

```
main
C:\Users\podar\study\anaconda\envs\LR15\python.exe "C:/Users/podar/study/PyC
[*] Время выполнения: 0.9437682628631592 секунд.
```

Рисунок 9 – Пример задания

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 = <число>». Примените декоратор к функции и вызовите декорированную функцию.

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


def show_perimeter(func):
    def wrapped(*args):
        print(f"Периметр фигуры равен = {func(args)}")

    return wrapped


@show_perimeter
def count_perimeter(sides):
```

```
main
C:\Users\podar\study\anaconda\envs\LR15\python.exe "C:/Users
7 21 3
Периметр фигуры равен = 31
```

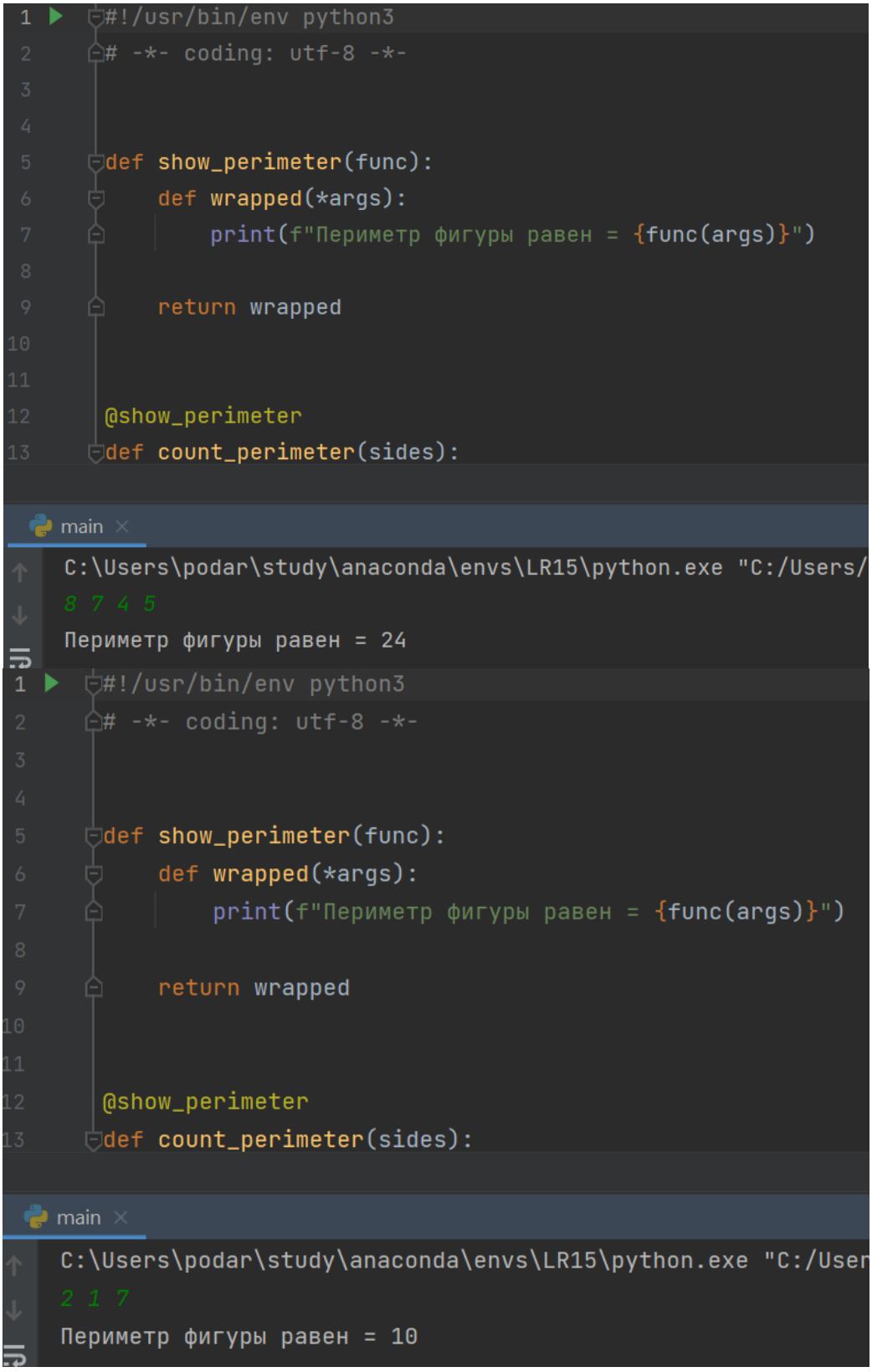


Рисунок 10 – Индивидуальное задание

Вопросы

1. **Что такое декоратор?**

   Декоратор – это функция, которая позволяет обернуть другую функцию для расширения её функциональности без непосредственного изменения её кода.

2. **Почему функции являются объектами первого класса?**

   Объектами первого класса в контексте конкретного языка программирования называется 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?**

   Функции высших порядков – это такие функции, которые могут принимать в качестве аргументов и возвращать другие функции.

4. **Как работают декораторы?**

   Декоратор – это функция, которая позволяет обернуть другую функцию для расширения её функциональности без непосредственного изменения её кода. Внутри декораторы мы определяем другую функцию, обёртку, так сказать, которая обёртывает функцию-аргумент и затем изменяет её поведение. Мы создаём декоратор, замеряющий время выполнения функции. Далее мы используем его функции, которая делает GET- запрос к главной странице. Чтобы измерить скорость, мы сначала сохраняем время перед выполнением обёрнутой функции, выполняем её снова сохраняем текущее время и вычитаем из него начальное.
   Выражение @decorator_function вызывает decorator_function() с hello_world в качестве аргумента и присваивает имени hello_world возвращаемую функцию.

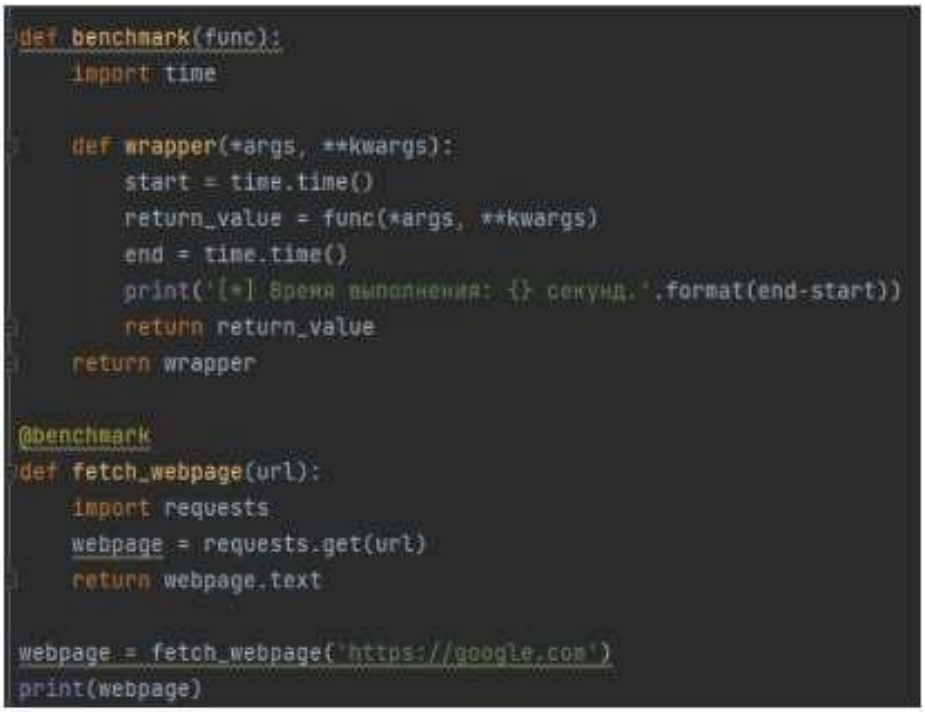
```python
def benchmark(func):
    import time

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

@benchmark
def fetch_webpage():
    import requests
    webpage = requests.get('https://google.com')

fetch_webpage()
```

5. **Какова структура декоратора функций?**



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

```python
import functools


def decoration(*args):
    def dec(func):
        @functools.wraps(func)
        def decor():
            func()
            print(*args)
        return decor
    return dec


@decoration('This is *args')
def func_ex():
    print('Look at that')

if __name__ == '__main__':
    func_ex()
```

```
Look at that
This is *args

Process finished with exit code 0
```