

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

Институт цифрового развития
Кафедра инфокоммуникаций

ОТЧЁТ
ПО ЛАБОРАТОРНОЙ РАБОТЕ №2.12
дисциплины
«Программирование на Python»

Выполнил:
Костукайло Кирилл Николаевич
2 курс, группа ИВТ-б-о-21-1,
09.03.01 «Информатика и
вычислительная техника»,
направленность (профиль)
«Программное обеспечение средств
вычислительной техники и
автоматизированных систем», очная
форма обучения

(подпись)

Проверил:
Кафедры инфокоммуникаций, старший
преподаватель
Воронкин Р.А.

(подпись)

Отчёт защищён с оценкой _____ Дата защиты _____

Ставрополь, 2022 г.

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.

Ход работы:

1. Создал общедоступный репозиторий на GitHub, в котором будет использована лицензия MIT и язык программирования Python, клонировал созданного репозитория.
2. Создал проект PyCharm в папке репозитория и проработал пример лабораторной работы.

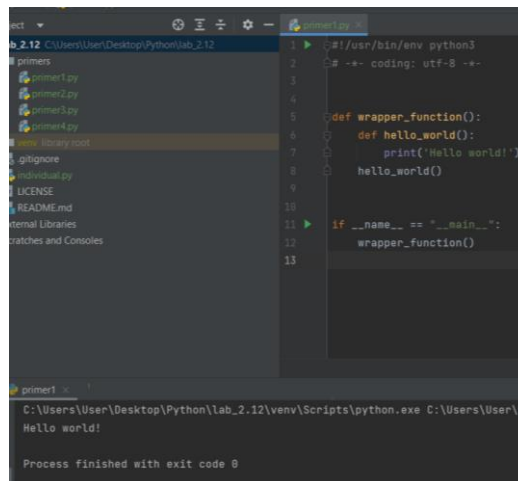


Рисунок 1. Пример 1

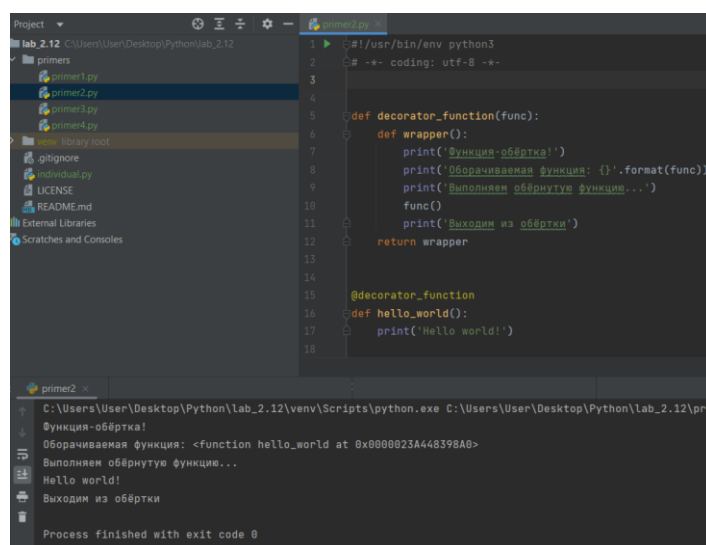
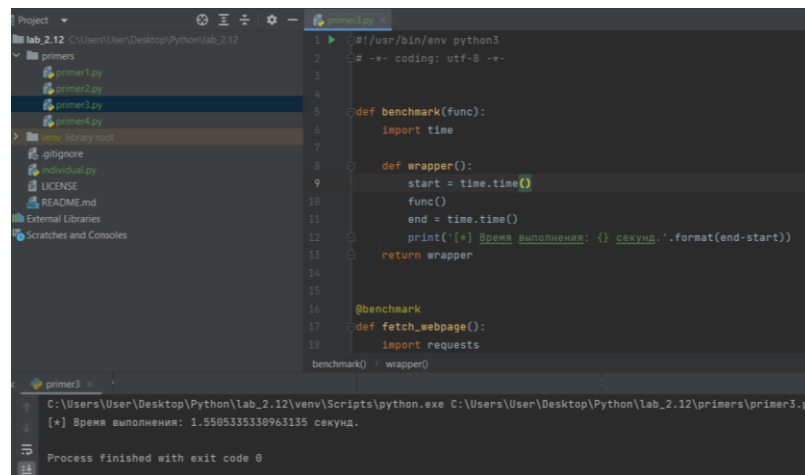


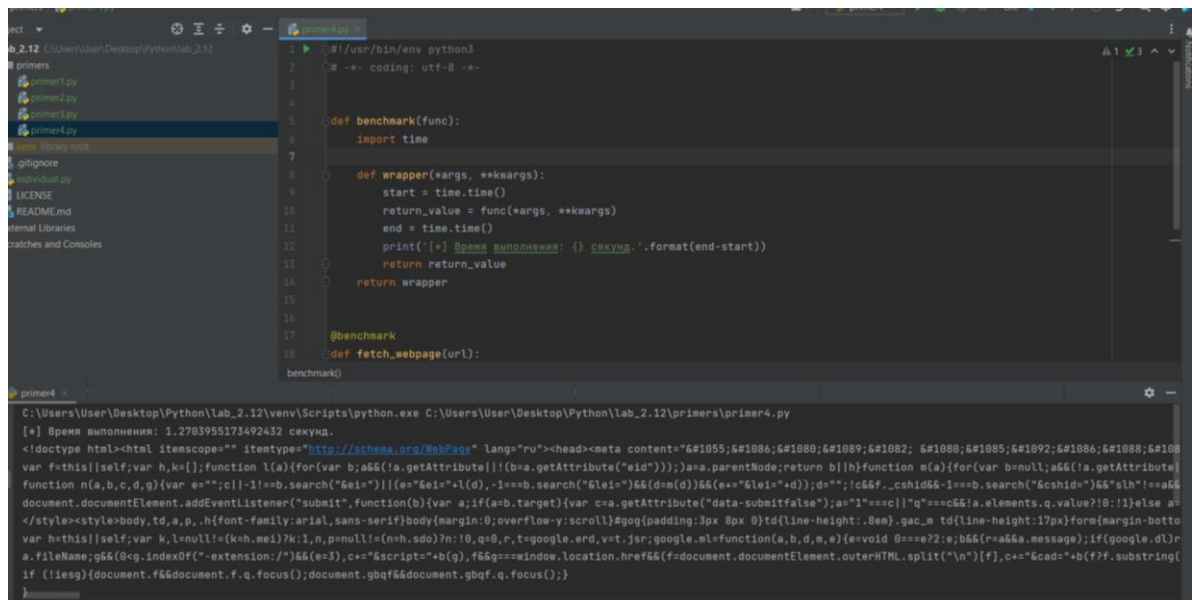
Рисунок 2. Пример 2



```
1 #!/usr/bin/env python3
2 # -*- coding: utf-8 -*-
3
4
5 def benchmark(func):
6     import time
7
8     def wrapper():
9         start = time.time()
10        func()
11        end = time.time()
12        print('[*] Время выполнения: {} секунд.'.format(end-start))
13    return wrapper
14
15
16 @benchmark
17 def fetch_webpage():
18     import requests
19
20 benchmark() > wrapper()
```

Process finished with exit code 0

Рисунок 3. Пример 3



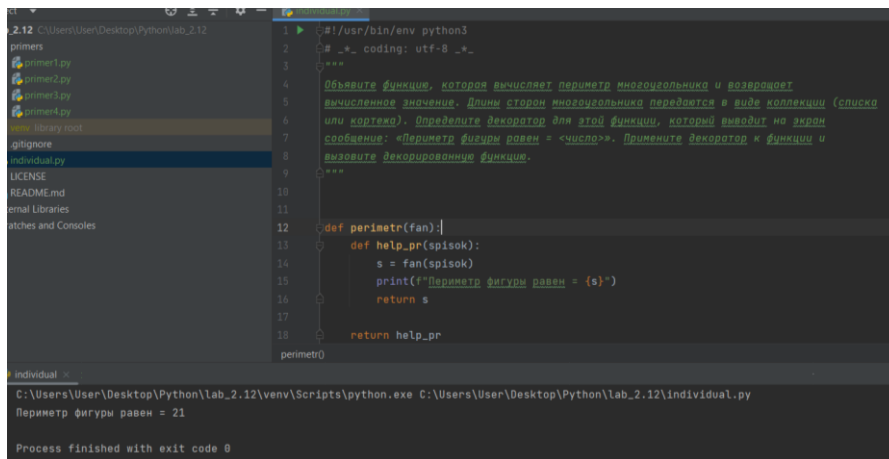
```
1 #!/usr/bin/env python3
2 # -*- coding: utf-8 -*-
3
4
5 def benchmark(func):
6     import time
7
8     def wrapper(*args, **kwargs):
9         start = time.time()
10        return_value = func(*args, **kwargs)
11        end = time.time()
12        print('[*] Время выполнения: {} секунд.'.format(end-start))
13    return return_value
14
15
16 @benchmark
17 def fetch_webpage(url):
```

Process finished with exit code 0

Рисунок 4. Пример 4

Выполнил индивидуальную задачу:

Индивидуальная задача. 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.



```
2.12 C:\Users\User\Desktop\Python\lab_2.12
primers
primer1.py
primer2.py
primer3.py
primer4.py
__init__.py
LICENSE
README.md
venv\Scripts\python.exe
venv\Scripts\python3
venv\Scripts\python3.exe
venv\Scripts\python3.10.exe
venv\Scripts\python3.10.1.exe
venv\Scripts\python3.10.1.exe

1 #!/usr/bin/env python3
2 # -*- coding: utf-8 -*-
3 """
4 объявите функцию, которая вычисляет периметр многоугольника и возвращает
5 вычисленное значение. Длины сторон многоугольника передаются в виде коллекции (списка
6 или кортежа). Определите декоратор для этой функции, который выводит на экран
7 сообщение: «Периметр фигуры равен = <число>». Примените декоратор к функции и
8 вызовите декорированную функцию.
9 """
10
11
12 def perimetr(fan):
13     def help_pr(spisok):
14         s = fan(spisok)
15         print(f"Периметр фигуры равен = {s}")
16         return s
17     return help_pr
18
19 perimetr0
```

C:\Users\User\Desktop\Python\lab_2.12\venv\Scripts\python.exe C:\Users\User\Desktop\Python\lab_2.12\individual.py
Периметр фигуры равен = 21
Process finished with exit code 0

Рисунок 5. Инд. Задание

Ответы на контрольные вопросы:

1. Что такое декоратор?

Декоратор — это функция, которая позволяет обернуть другую функцию для расширения её функциональности без непосредственного изменения её кода. Вот почему декораторы можно рассматривать как практику мета программирования, когда программы могут работать с другими программами как со своими данными.

2. Почему функции являются объектами первого класса?

В Python всё является объектом, а не только объекты, которые вы создаёте из классов. В этом смысле он (Python) полностью соответствует идеям объектно-ориентированного программирования. Это значит, что в Python всё это — объекты:

- числа;
- строки;
- классы (да, даже классы!);
- функции (то, что нас интересует).

Тот факт, что всё является объектами, открывает перед нами множество возможностей. Мы можем 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.

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?

```
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
```

Здесь `decorator_function()` является функцией-декоратором. Как вы могли заметить, она является функцией высшего порядка, так как принимает функцию в качестве аргумента, а также возвращает функцию. Внутри `decorator_function()` мы определили другую функцию, обёртку, так сказать, которая обёртывает функцию-аргумент и затем изменяет её поведение. Декоратор возвращает эту обёртку.

5. Какова структура декоратора функций?

В 4 вопросе пример. Здесь `decorator_function()` является функцией декоратором. Как вы могли заметить, она является функцией высшего порядка, так как принимает функцию в качестве аргумента, а также возвращает функцию. Внутри `decorator_function()` мы определили другую функцию, обёртку, так сказать, которая обёртывает функцию-аргумент и затем изменяет её поведение. Декоратор возвращает эту обёртку.

6. Самостоятельно изучить как можно передать параметры декоратору, а не декорируемой функции? В декоратор можно передать и сам параметр. В этом случае нужно добавить ещё один слой абстракции, то есть — ещё одну функцию-обёртку. Это обязательно, поскольку аргумент передаётся декоратору. Затем, функция, которая вернулась, используется для декорации нужной.