

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ  
РОССИЙСКОЙ ФЕДЕРАЦИИ  
Федеральное государственное автономное  
образовательное учреждение высшего образования  
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Кафедра инфокоммуникаций

**Отчет по лабораторной работе №7**  
**Декораторы функций в языке Python**  
**По дисциплине «Технологии программирования и алгоритмизация»**

Выполнил студент группы ИВТ-б-о-20-1

Бобров Н. В. « » \_\_\_\_\_ 20\_\_ г.

Подпись студента \_\_\_\_\_

Работа защищена « » \_\_\_\_\_ 20\_\_ г.

Проверил Воронкин Р. А. \_\_\_\_\_

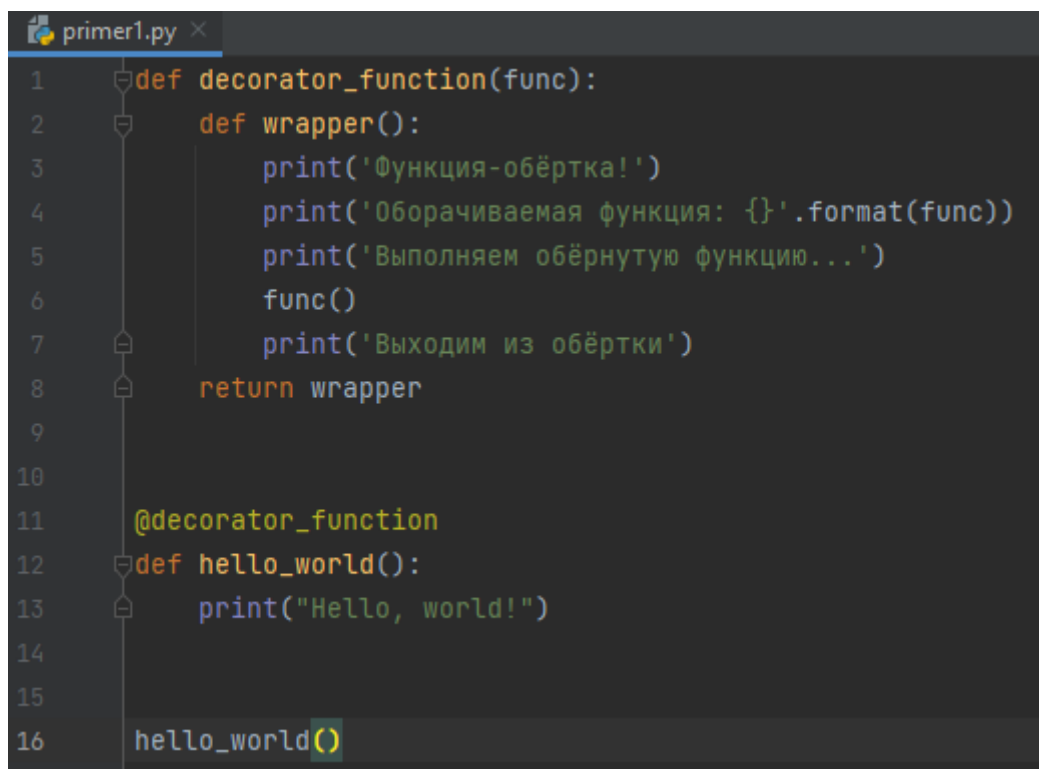
(подпись)

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

### Ход работы:

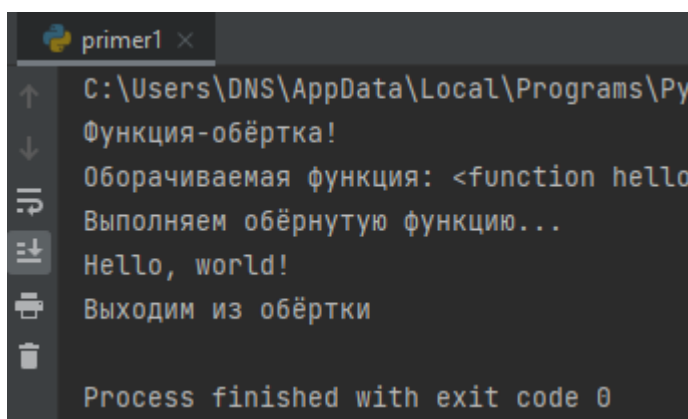
GitHub: <https://github.com/nbobrov8/laba14>

1. Создал общедоступный репозиторий, клонировал его локальный сервер.
2. Изучил теоретический материал и проработал примеры.



```
1 def decorator_function(func):
2     def wrapper():
3         print('Функция-обёртка!')
4         print('Оборачиваемая функция: {}'.format(func))
5         print('Выполняем обёрнутую функцию...')
6         func()
7         print('Выходим из обёртки')
8     return wrapper
9
10
11 @decorator_function
12 def hello_world():
13     print("Hello, world!")
14
15
16 hello_world()
```

Рисунок 1 – Пример кода с декораторами



```
primer1 x
C:\Users\DNS\AppData\Local\Programs\Python\Python38\python.exe primer1.py
Функция-обёртка!
Оборачиваемая функция: <function hello_world at 0x000001F0...>
Выполняем обёрнутую функцию...
Hello, world!
Выходим из обёртки
Process finished with exit code 0
```

Рисунок 2 – Результат работы кода

3. Проработал второй пример с использованием декоратора.

```

primer2.py x
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     return wrapper
15
16
17     @benchmark
18     def fetch_webpage(url):
19         import requests
20         webpage = requests.get(url)
21         return webpage.text
22
23
24  ▶  if __name__ == '__main__':
25     webpage = fetch_webpage('https://google.com')
26     print(webpage)

```

Рисунок 3 – Код второго примера

4. Приступил к выполнению индивидуального задания.

### Индивидуальное задание. Вариант 1.

Условие: 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`.

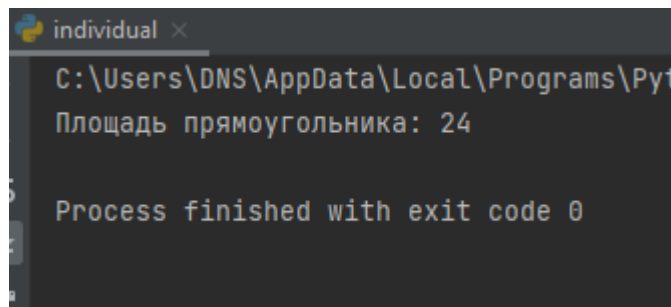
1. Объявил внешнюю функцию `func_show`, в ней вызвал функцию обертку, где обратился к оригинальной функции для получения результата.
2. Объявил декоратор с именем `get_sq`.
3. Сделал вызов декоративной функции.

```

13 def func_show(func):
14     def wrapper(width, height):
15         res = func(width, height) # обратимся к оригинальной функции для получения результата
16         print(f"Площадь прямоугольника: {res}")
17         return res
18     return wrapper # результат работы декоратора - вызов нашей функции-обработчика
19
20
21 @func_show
22 def get_sq(width, height):
23     return width*height
24
25
26 if __name__ == '__main__':
27     # вызов декорированной функции
28     get_sq(width=4, height=6)

```

Рисунок 4 – Код выполненного результата



```

individual x
C:\Users\DNS\AppData\Local\Programs\Python\Python39\python.exe
Площадь прямоугольника: 24

Process finished with exit code 0

```

Рисунок 5 – Результат выполнения кода

### Контрольные 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?

Потому что с ними можно работать как с переменными, могут быть переданы как аргумент процедуры, могут быть возвращены как результат выполнения процедуры, могут быть включены в другие структуры данных.

#### 3. Каково назначение функций высших порядков?

Основной задачей функций высших порядков является возможность принимать в качестве аргументов и возвращать другие функции.

#### 4. Как работают декораторы?

Они берут декорируемую функцию в качестве аргумента и позволяет

совершать с ней какие-либо действия до и после того, что сделает эта функция, не изменяя её.

#### 5. Какова структура декоратора функций?

Функция `decorator` принимает в качестве аргумента функцию `func`, внутри функции `decorator` другая функция `wrapper`. В конце декоратора происходит возвращение функции `wrapper`.

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

Достаточно обернуть функцию декоратор в другую функцию, которая будет принимать аргументы. И сделать вывод функций `wrapper` и `decorator`.

**Вывод:** в ходе выполнения лабораторной работы приобрел навыки по работе с декораторами функций при написании программ с помощью языка программирования Python версии 3.x.