

12Министерство 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»**

Выполнил:  
Кожуховский Виктор Андреевич  
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 Роман Александрович

---

(подпись)

Отчет защищен с оценкой \_\_\_\_\_ Дата защиты \_\_\_\_\_

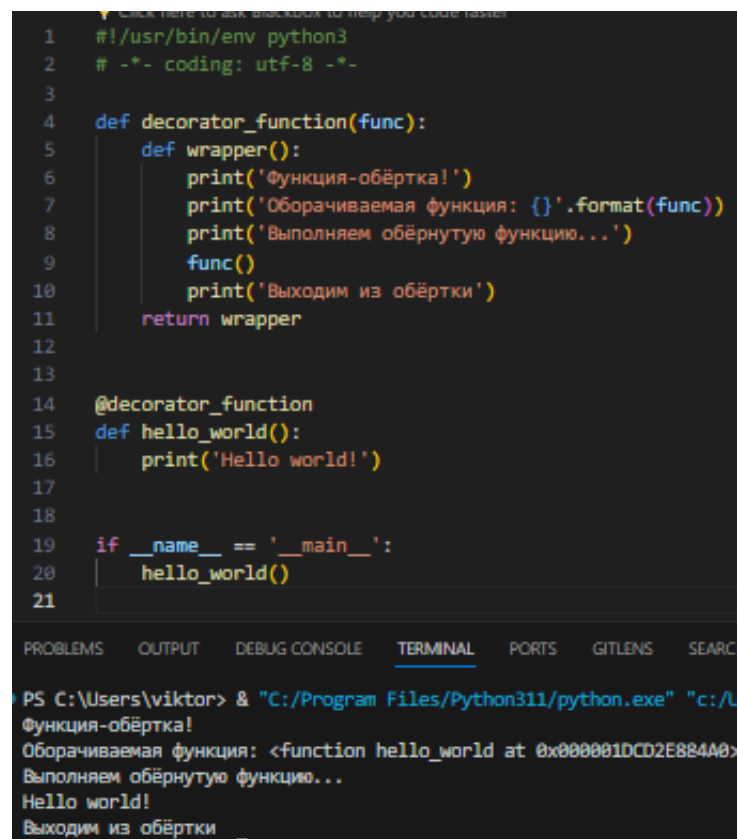
Ставрополь, 2023 г.

**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.

### Методика и порядок выполнения работы

1. Изучил теоретический материал работы.
2. Создал общедоступный репозиторий на GitHub, в котором использована лицензия MIT и язык программирования Python.
3. Выполнил клонирование созданного репозитория.
4. Дополнил файл .gitignore необходимыми правилами для работы с IDE PyCharm.
5. Организовал свой репозиторий в соответствии с моделью ветвления git-flow.
6. Создал проект в папке репозитория.
7. Проработал примеры лабораторной работы.



```
1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3
4  def decorator_function(func):
5      def wrapper():
6          print('Функция-обёртка!')
7          print('Оборачиваемая функция: {}'.format(func))
8          print('Выполняем обёрнутую функцию...')
9          func()
10         print('Выходим из обёртки')
11     return wrapper
12
13
14 @decorator_function
15 def hello_world():
16     print('Hello world!')
17
18
19 if __name__ == '__main__':
20     hello_world()
21
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS GITLENS SEARCH

```
PS C:\Users\viktor> & "C:/Program Files/Python311/python.exe" "c:/l
Функция-обёртка!
Оборачиваемая функция: <function hello_world at 0x000001DCD2E884A0>
Выполняем обёрнутую функцию...
Hello world!
Выходим из обёртки
```

Рисунок 1. Код примера 1 и его выполнение

```
1 #!/usr/bin/env python3
2 # -*- coding: utf-8 -*-
3
4 def benchmark(func):
5     import time
6
7     def wrapper(*args, **kwargs):
8         start = time.time()
9         return_value = func(*args, **kwargs)
10        end = time.time()
11        print('[*] Время выполнения: {} секунд.'.format(end-start))
12        return return_value
13    return wrapper
14
15
16 @benchmark
17 def fetch_webpage(url):
18     import requests
19     webpage = requests.get(url)
20     return webpage.text
21
22
23 if __name__ == '__main__':
24     webpage = fetch_webpage('https://google.com')
25     print(webpage)
26
```

PS C:\Users\viktor> & "C:/Program Files/Python311/python.exe" "c:/Users/viktor/

[\*] Время выполнения: 1.1314055919647217 секунд.

<!doctype html><html itemscope="" itemtype="http://schema.org/WebPage" lang="ru

; &#1072; &#1094; &#1080; &#1080; &#1074; &#1080; &#1085; &#1090; &#1077; &#1088; &#1085

9; , &#1082; &#1072; &#1088; &#1090; &#1080; &#1085; &#1082; &#1080; , &#1074; &#1080; &#10

1086; &#1077; ; " name="description"><meta content="noodp" name="robots"><meta con

/google\_color\_128dp.png" itemprop="image"><title>Google</title><scrip

72358,206,4884,1132070,870537,327208,380746,16114,28684,22430,1362,12318,17581,

4,2,39761,5679,1020,31122,4568,6259,23418,1252,33064,2,2,1,6960,16865,10962,233

13496,29964,7597,5215444,2,225,141,1103,117,5993379,2803214,3311,936,4,30566,19

12799,8409,3322,4681,897,5799,1965,9,9822,1676,1517,3608,819,7459,3117,2048,383

471,4432,836,2958,279,1044,39,4438,2222,415,2424,2371,1251,108,853,149,2383,664

1,1073,1083,4378,811,709,768,3,4,2,2,2,161,95,24,1507,1084,92,630,250,591,283,2

2,823,881,4,85,94,225,1,1,49,560,265,8,1139,165,102,323,7,208,6,561,364,1637,24

,352,84,1622,832,1732,3,21732191,218,1624,3,2378,5,274',kBL:'lXp7',kOPI:8997844

this;})();(function(){google.sn='webhp';google.kHL='ru';})();(function(){

Рисунок 2. Код примера 2 и его выполнение

## 8. Выполнил индивидуальное задание.

Объявите функцию, которая возвращает переданную ей строку в нижнем регистре (с малыми буквами). Определите декоратор для этой функции, который имеет один параметр tag, определяющий строку с названием тега (начальное значение параметра tag равно h1). Этот декоратор должен заключать возвращенную функцией строку в тег tag и возвращать результат. Пример заключения строки "python" в тег h1: <h1>python</h1> . Примените декоратор со значением tag="div" к функции и вызовите декорированную функцию. Результат отобразите на экране.

```
1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3
4  def tag_decorator(tag):
5      def decorator(func):
6          def wrapper(text):
7              return f"<{tag}>{func(text)}</{tag}>"
8          return wrapper
9      return decorator
10
11
12  @tag_decorator(tag="div")
13  def to_lowercase(input_string):
14      return input_string.lower()
15
16
17  if __name__ == '__main__':
18      print(to_lowercase(input("Введите слово для обертки: ")))
19
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  GITLENS  SEARCH ERROR
PS C:\Users\viktor> & "C:/Program Files/Python311/python.exe" "c:/Users/v
Введите слово для обертки: слОВО
<div>слово</div>
```

Рисунок 3. Код индивидуального задания и его выполнение

9. Зафиксировал сделанные изменения в репозитории.
10. Добавил отчет по лабораторной работе в формате PDF в папку doc репозитория. Зафиксировал изменения.
11. Выполнил слияние ветки для разработки с веткой master/main.
12. Отправил сделанные изменения на сервер GitHub.

### Вопросы для защиты работы

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 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?

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

4. Как работают декораторы?

Декораторы позволяют изменять или расширять поведение функций или классов без изменения их кода. Они "оборачивают" исходную функцию, предоставляя дополнительную функциональность до и/или после основной функции, не изменяя её поведение напрямую.

#### 5. Какова структура декоратора функций?

Структура декоратора функций в Python обычно состоит из трех уровней вложенности:

Внешняя функция, которая является самим декоратором. Она принимает функцию в качестве аргумента и возвращает обернутую функцию (внутреннюю функцию).

Внутренняя функция (обернутая функция), которая выполняет действия перед и/или после вызова декорируемой функции. Она принимает аргументы декорируемой функции `*args` и `**kwargs`, вызывает декорируемую функцию и возвращает ее результат.

Сама декорируемая функция, которая выполняет основную работу.

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

Чтобы передать параметры декоратору, нам нужно использовать двусмысленность — первый вызов декоратора должен принимать аргументы, и внутри него должна быть определена и возвращена внутренняя функция, которая является самим декоратором.