

**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ  
РОССИЙСКОЙ  
ФЕДЕРАЦИИ**

**Федеральное государственное автономное  
образовательное учреждение высшего образования  
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»**

**Кафедра инфокоммуникаций  
«Декораторы функций в языке Python»**

**Отчет по лабораторной работе № 2.12  
по дисциплине «Основы программной инженерии»**

Выполнил студент группы ПИЖ-б-о-21-1

Коновалова В.Н. « » 2022г.

Подпись студента \_\_\_\_\_

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

Проверил Воронкин Р.А. \_\_\_\_\_  
(подпись)

Ставрополь 2022

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

Ход работы:

1. Изучить теоретический материал работы.
2. Создать общедоступный репозиторий на GitHub, в котором будет использована лицензия MIT и язык программирования Python.
3. Выполните клонирование созданного репозитория.
4. Дополните файл .gitignore необходимыми правилами для работы с IDE PyCharm.
5. Организуйте свой репозиторий в соответствии с моделью ветвления git-flow.
6. Создайте проект PyCharm в папке репозитория.
7. Проработайте примеры лабораторной работы.

Пример 1.

Код:

```
#!/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()
```

```
Функция-обёртка!  
Оборачиваемая функция: <function hello_world at 0x00000237619B3AC0>  
Выполняем обёрнутую функцию...  
Hello world!  
Выходим из обёртки  
  
Process finished with exit code 0
```

Рисунок 6 – Результат работы программы

Код:

```
#!/usr/bin/env python3  
# -*- coding: utf-8 -*-  
  
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')  
  
if __name__ == '__main__':  
    fetch_webpage()
```

```
[*] Время выполнения: 0.5143680572509766 секунд.  
  
Process finished with exit code 0
```

Рисунок 7 – Результат работы программы

## 8. Выполните индивидуальные задания.

Вводится строка целых чисел через пробел. Напишите функцию, которая преобразовывает эту строку в список чисел и возвращает их сумму. Определите декоратор для этой функции, который имеет один параметр `start` – начальное значение суммы. Примените декоратор со значением `start=5` к функции и вызовите декорированную функцию. Результат отобразите на экране.

Код:

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Вводится строка целых чисел через пробел. Напишите функцию, которая
преобразовывает
эту строку в список чисел и возвращает их сумму. Определите декоратор для
этой функции,
который имеет один параметр start – начальное значение суммы. Примените
декоратор
со значением start=5 к функции и вызовите декорированную функцию. Результат
отобразите на экране.
"""

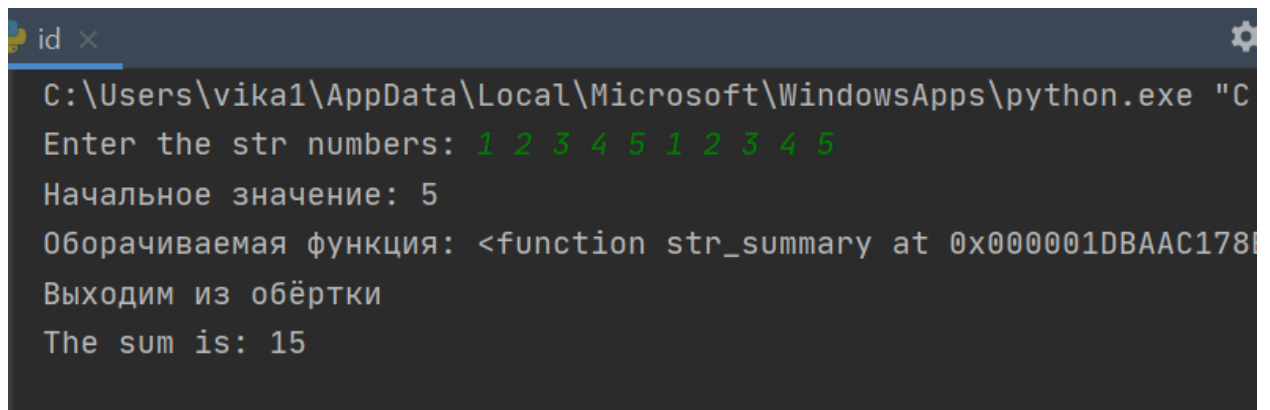
def decorator(start=0):
    def wrapper(func):
        def inner(*args):
            print(f"Начальное значение: {start}")
            print(f"Оборачиваемая функция: {func}")
            string = args[0]
            lst = string.split()
            lst = lst[start:]
            start_string = " ".join(lst)
            return_sum = func(start_string)
            print("Выходим из обёртки")
            return return_sum

        return inner

    return wrapper

@decorator(start=5)
def str_summary(string):
    a = [int(s) for s in string.split()]
    summa = sum(elem for elem in a)
    return summa

if __name__ == "__main__":
    strr = input("Enter the str numbers: ")
    print(f"The sum is: {str_summary(strr)}")
```

A screenshot of a Python IDLE window. The title bar shows 'id' and a close button. The command prompt shows the path 'C:\Users\vika1\AppData\Local\Microsoft\WindowsApps\python.exe "C...'. The program prompts 'Enter the str numbers:' and receives the input '1 2 3 4 5 1 2 3 4 5' in green text. It then displays 'Начальное значение: 5', 'Оборачиваемая функция: <function str\_summary at 0x000001DBAAC178...', 'Выходим из обёртки', and finally 'The sum is: 15'.

```
id ×
C:\Users\vika1\AppData\Local\Microsoft\WindowsApps\python.exe "C...
Enter the str numbers: 1 2 3 4 5 1 2 3 4 5
Начальное значение: 5
Оборачиваемая функция: <function str_summary at 0x000001DBAAC178...
Выходим из обёртки
The sum is: 15
```

Рисунок 8 – Результат работы программы

## 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?

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.

## 5. Какова структура декоратора функций?

```
#!/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()
```

decorator\_function()

пример\_1 (1) ×

"C:\Users\ynakh\OneDrive\Рабочий стол\Git\Lab-15-OPJ\p15\venv\Scripts\  
Функция-обёртка!  
Оборачиваемая функция: <function hello\_world at 0x000002C0B4797AC0>  
Выполняем обёрнутую функцию...  
Hello world!  
Выходим из обёртки

Process finished with exit code 0

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

```
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')

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

гшрж x

"C:\Users\ynakh\OneDrive\Рабочий стол\  
Look  
This is args

Process finished with exit code 0

Вывод: в ходе выполнения практической работы были приобретены навыки по работе декораторами функций при написании программ с помощью языка программирования Python.