

Министерство 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 «Основы программной инженерии»

Выполнила:
Ламская Ксения Вячеславовна
2 курс, группа ПИЖ-б-о-22-1,
09.03.04 «Программная инженерия»,
направленность (профиль) «Разработка и
сопровождение программного
обеспечения», очная форма обучения

(подпись)

Доцент кафедры инфокоммуникаций
Воронкин Роман Александрович

(подпись)

Отчет защищен с оценкой _____ Дата защиты _____

Ставрополь, 2023 г.

Тема: Лабораторная работа 2.12. 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.


Порядок выполнения работы

1. Создание репозитория GitHub.

Create a new repository



A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

Required fields are marked with an asterisk ().*

Owner *	Repository name *
 ksenia-lamskaya ▾	/ 15laba
	✓ 15laba is available.

Great repository names are short and memorable. Need inspiration? How about [upgraded-octo-garbanzo](#) ?

Description (optional)

- ☒  **Public**
Anyone on the internet can see this repository. You choose who can commit.
- ☐  **Private**
You choose who can see and commit to this repository.

Initialize this repository with:

- ☒ **Add a README file**
This is where you can write a long description for your project. [Learn more about READMEs.](#)

Add .gitignore

.gitignore template: Python ▾


Choose which files not to track from a list of templates. [Learn more about ignoring files.](#)

Choose a license

License: MIT License ▾

A license tells others what they can and can't do with your code. [Learn more about licenses.](#)

This will set  **main** as the default branch. Change the default name in your [settings](#).

 You are creating a public repository in your personal account.

Create repository

Рисунок 1 – Создание репозитория

2. Проработайте пример из методички.

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

class Hello:
    pass

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

def wrapper_function():
    def hello_world1():
        print('Hello world!')
    hello_world1()

def higher_order(func):
    print(f'Получена функция {func} в качестве аргумента')
    func()
    return func

if __name__ == '__main__':
    print('Пример №1')
    print(type(hello_world()))
    print(type(Hello))
    print(type(10))

    print('\nПример №2')
    wrapper_function()

    print('\nПример №3')
    higher_order(hello_world)
```

Рисунок 2.1 – Пример кода

```
Пример №1
Hello world!
<class 'NoneType'>
<class 'type'>
<class 'int'>

Пример №2
Hello world!

Пример №3
Получена функция <function hello_world at 0x000001F379A504A0> в качестве аргумента
Hello world!
```

Рисунок 2.2 – Вывод программы

```
#!/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')

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__':
    print('Пример №4')
    hello_world()

    print('\nПример №5')
    print('Done!')
    # fetch_webpage()

    print('\nПример №6')
```

Рисунок 2.3 – Код программы

```
Пример №4
Функция-обёртка!
Оборачиваемая функция: <function hello_world at 0x0000019B824A9EE0>
Выполняем обёрнутую функцию...
Hello world!
Выходим из обёртки

Пример №5
Done!

Пример №6
```

Рисунок 2.4 – Вывод программы

3. Используя замыкания функций, объявите внутреннюю функцию, которая принимает в качестве аргумента коллекцию (список или кортеж) и возвращает или минимальное значение, или максимальное, в зависимости от значения параметра `type` внешней функции. Если `type` равен «`max`», то возвращается максимальное значение, иначе – минимальное. По умолчанию `type` должно принимать значение «`max`». Вызовите внутреннюю функцию замыкания и отобразите на экране результат ее работы.

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

def cyrillic_to_latin(word):
    symb = {'ё': 'yo', 'а': 'a', 'б': 'b', 'в': 'v', 'г': 'g', 'д': 'd', 'е': 'e',
            'ж': 'zh', 'з': 'z', 'и': 'i', 'й': 'y', 'к': 'k', 'л': 'l', 'м': 'm',
            'н': 'n', 'о': 'o', 'п': 'p', 'р': 'r', 'с': 's', 'т': 't', 'у': 'u',
            'ф': 'f', 'х': 'h', 'ц': 'c', 'ч': 'ch', 'ш': 'sh', 'щ': 'shch',
            'ъ': '', 'ы': 'y', 'ь': '', 'э': 'e', 'ю': 'yu', 'я': 'ya'}
    word = word.lower()
    result = ''

    for char in word:
        result += symb.get(char, char)
    return result

def replace_chars(chars): #декоратор
    def decorator(func):
        def wrapper(word):
            for char in chars:
                word = word.replace(char, '-')
            word = '-'.join(filter(None, word.split('-')))
            return func(word)
        return wrapper
    return decorator

# Применяем декоратор со значениями chars="?!:;, . "
@replace_chars("?!:;, . ")
def decorated_cyrillic_to_latin(word):
    return cyrillic_to_latin(word)

result = decorated_cyrillic_to_latin(input('Введите фразу: '))
print(result)
```

Рисунок 3.1 – Код программы

```
Введите фразу: Привет Мир!!!!
privet-mir
```

Рисунок 3.2 – Вывод программы

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?

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?

В Python декораторы представляют собой способ изменения поведения функции или метода без изменения их кода. Они работают, оборачивая (или декорируя) функцию в другую функцию. Декораторы принимают функцию, выполняют некоторый код перед или после вызова этой функции, и возвращают новую функцию (или тот же объект функции).

5. Какова структура декоратора функций?

```
def my_decorator(func):  
    def wrapper(*args, **kwargs)
```

```
result = func(*args, **kwargs)
return result
return wrapper
@my_decorator
def decorated_function(arg1, arg2):
return result
```

Здесь `my_decorator` - это функция, которая принимает другую функцию `func` в качестве аргумента и возвращает новую функцию `wrapper`. `wrapper` обычно принимает `*args` и `**kwargs`, чтобы передать аргументы оригинальной функции.

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

Чтобы передать параметры декоратору, вы можете определить дополнительную функцию, которая возвращает сам декоратор. Когда вы применяете декоратор к функции с использованием `@my_decorator_with_args("Аргумент1", "Аргумент2")`, эти аргументы передаются декоратору.