

**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

Разворотников Денис « » 2022г.

Подпись студента_____

Работа защищена « »_____20__г.

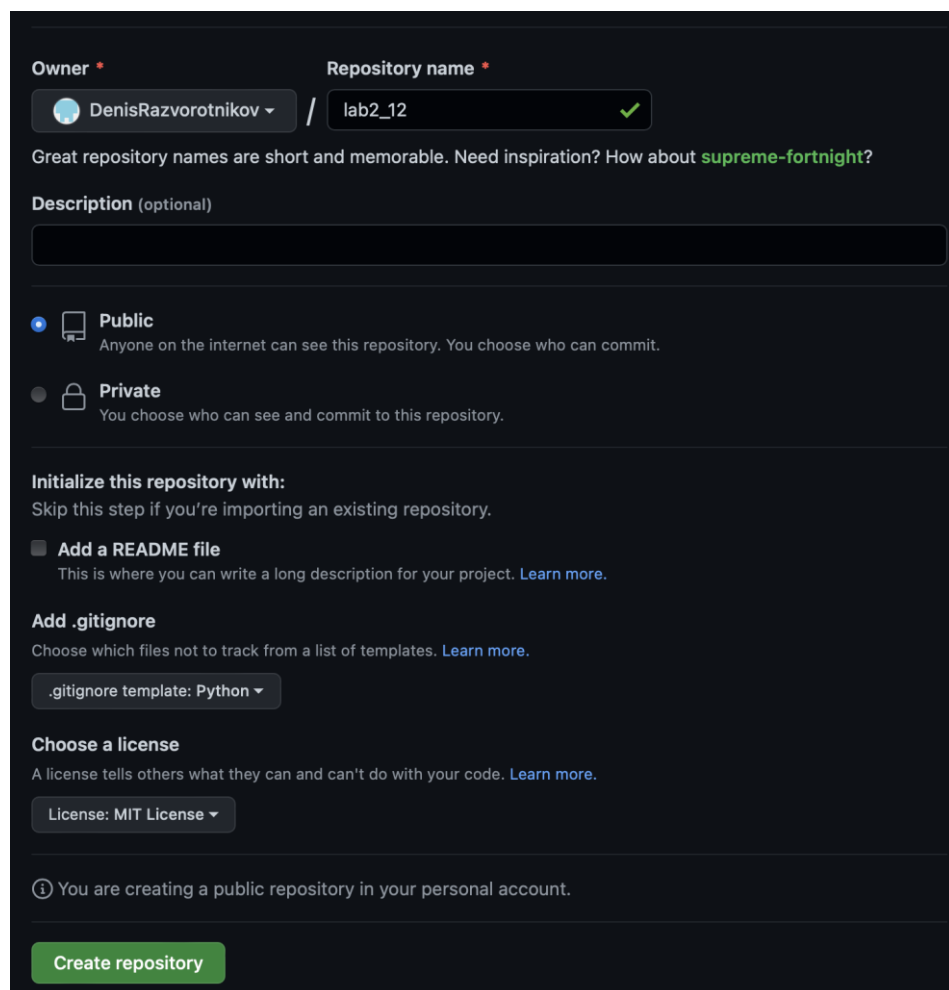
Проверил Воронкин Р.А. _____
(подпись)

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.



The screenshot shows the GitHub 'Create repository' form. At the top, the 'Owner' is set to 'DenisRazvorotnikov' and the 'Repository name' is 'lab2_12', which is marked as valid with a green checkmark. Below this, a message states: 'Great repository names are short and memorable. Need inspiration? How about [supreme-fortnight?](#)'. The 'Description' field is optional and currently empty. The visibility is set to 'Public', with a note: 'Anyone on the internet can see this repository. You choose who can commit.' The 'Private' option is also visible. Under 'Initialize this repository with:', there are three checkboxes: 'Add a README file' (checked), 'Add .gitignore' (checked), and 'Choose a license' (checked). The '.gitignore' template is set to 'Python'. The license is set to 'MIT License'. A note at the bottom states: 'You are creating a public repository in your personal account.' A green 'Create repository' button is at the bottom.

Рисунок 1 – Создание репозитория

2. Выполните клонирование созданного репозитория

```
denisrazvorotnikov@Air-Denis desktop % git clone https://github.com/DenisRazvorotnikov/lab2_12.git
Cloning into 'lab2_12'...
remote: Enumerating objects: 4, done.
remote: Counting objects: 100% (4/4), done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 4 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (4/4), done.
denisrazvorotnikov@Air-Denis desktop %
```

Рисунок 2 – Клонирование репозитория

3. 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()
```

```
/usr/local/bin/python3 /Users/denisrazvorotnikov/Desktop/lab2_12/1.py
Функция-обёртка!
Оборачиваемая функция: <function hello_world at 0x10b322830>
Выполняем обёрнутую функцию...
Hello world!
Выходим из обёртки
Process finished with exit code 0
```

Рисунок 3 – Результат работы программ

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()
```

```
/usr/local/bin/python3 /Users/denisrazvorotnikov/Desktop/lab2_12/2
[*] Время выполнения: 0.6541862487792969 секунд.

Process finished with exit code 0
```

Рисунок 4 – Результат работы программы

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 -*-

def external(start=0):
    def middle(func):
        def inner(*args):
            arg = args[0]
            return func(arg) + start
        return inner
    return middle

@external(start=5)
def sum_line(string):
    return sum(list(map(int, string.split()))))

if __name__ == "__main__":
    numb = input("Please enter numbers with gaps: ")
    print(sum_line(numb))
```

```
/usr/local/bin/python3 /Users/denisrazvorotnikov/Desktop/la
Please enter numbers with gaps: 12 43 43 53 6 3 23 1 4 43
236

Process finished with exit code 0
```

Рисунок 5 – 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.