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

Халимендик Я. Д. « » 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.

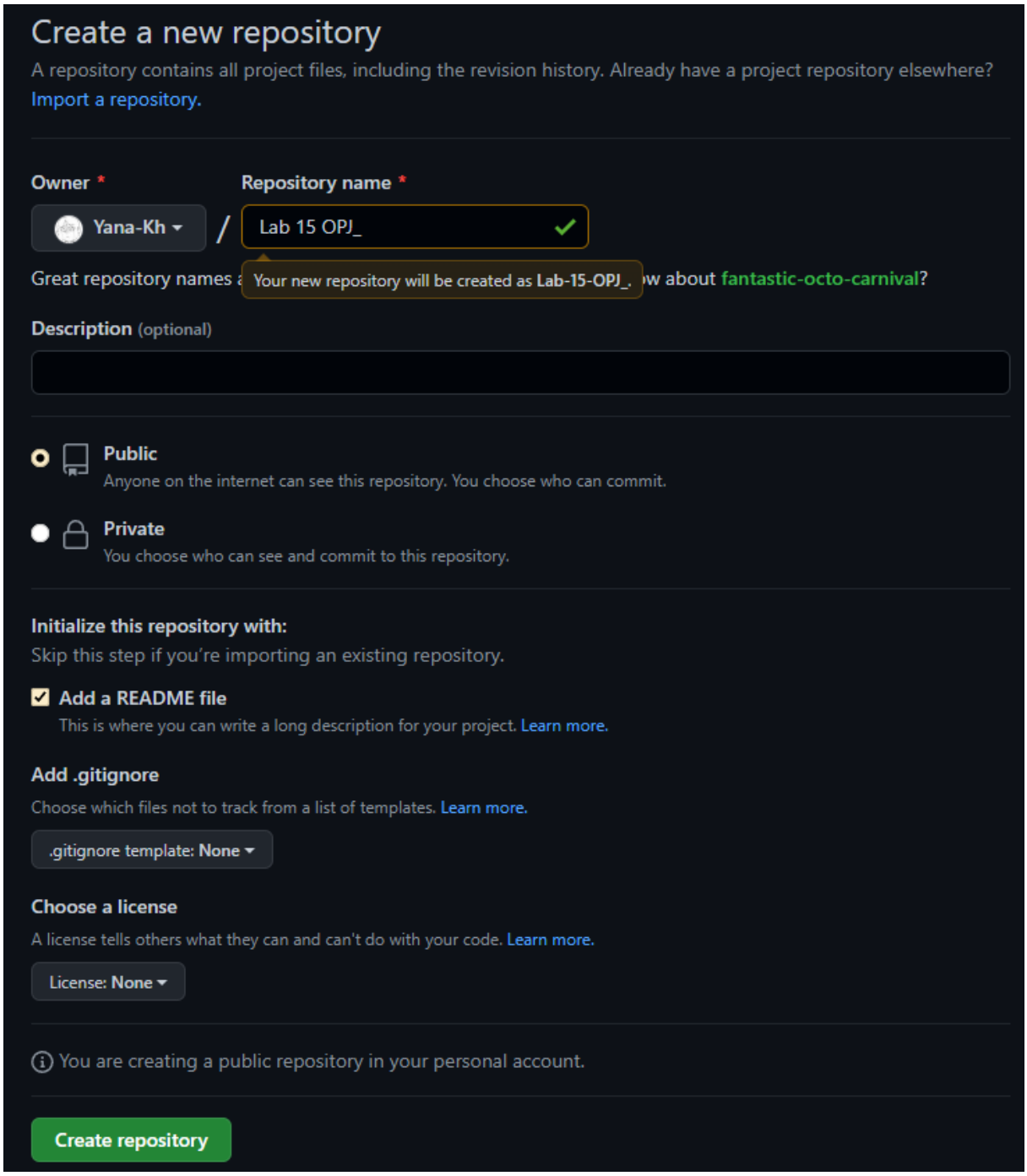


Рисунок 1 – Создание репозитория

3. Выполните клонирование созданного репозитория.

```
C:\Users\ynakh>cd C:\Users\ynakh\OneDrive\Рабочий стол\Git

C:\Users\ynakh\OneDrive\Рабочий стол\Git>git clone https://github.com/Yana-Kh/Lab-15-OPJ.git
Cloning into 'Lab-15-OPJ'...
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 5 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (5/5), done.

C:\Users\ynakh\OneDrive\Рабочий стол\Git>
```

Рисунок 2 – Клонирование репозитория

4. Дополните файл .gitignore необходимыми правилами для работы с IDE PyCharm.

```
C:\Users\ynakh\OneDrive\Рабочий стол\Git\Lab-15-OPJ>git add .

C:\Users\ynakh\OneDrive\Рабочий стол\Git\Lab-15-OPJ>git status
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   .gitignore


C:\Users\ynakh\OneDrive\Рабочий стол\Git\Lab-15-OPJ>
```

Рисунок 3 – Дополнение файла .gitignore

5. Организуйте свой репозиторий в соответствие с моделью ветвления git-flow.

```
C:\Users\ynakh\OneDrive\Рабочий стол\Git\Lab-15-OPJ>git flow init

Which branch should be used for bringing forth production releases?
   - main
Branch name for production releases: [main]
Branch name for "next release" development: [develop]

How to name your supporting branch prefixes?
Feature branches? [feature/]
Bugfix branches? [bugfix/]
Release branches? [release/]
Hotfix branches? [hotfix/]
Support branches? [support/]
Version tag prefix? []
Hooks and filters directory? [C:/Users/ynakh/OneDrive/Рабочий стол/Git/Lab-15-OPJ/.git/hooks]

C:\Users\ynakh\OneDrive\Рабочий стол\Git\Lab-15-OPJ>
```

Рисунок 4 – организация репозитория в соответствии с моделью git-flow

6. Создайте проект PyCharm в папке репозитория.

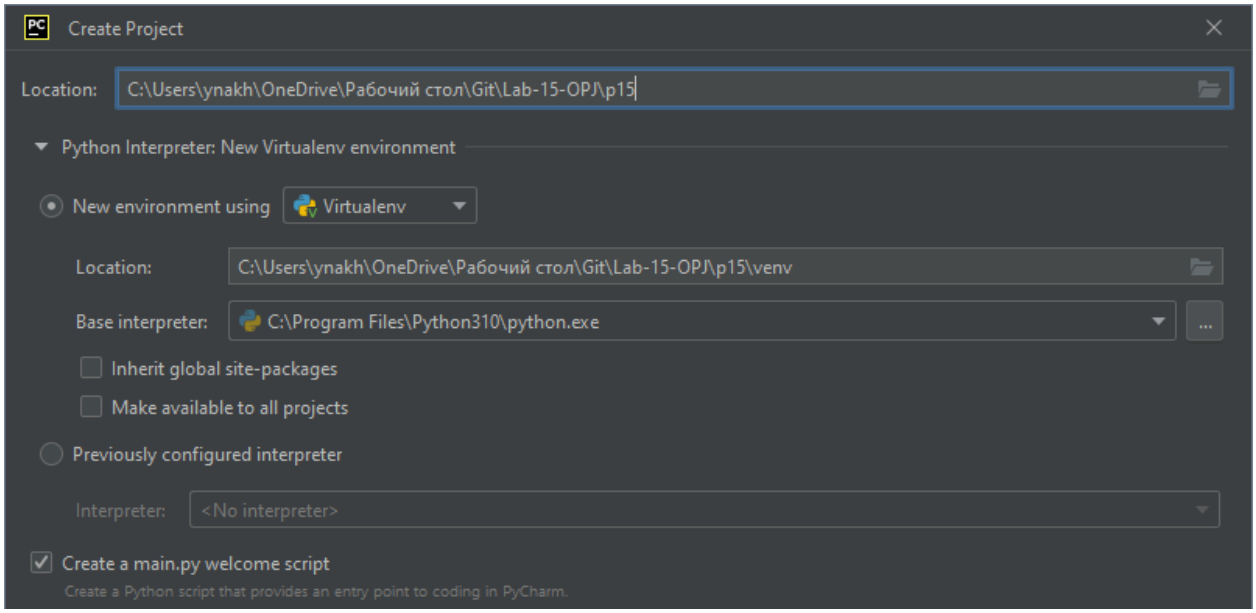


Рисунок 5 – Создание проекта PyCharm в папке репозитория

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:

```python
#!/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.

Вариант 32(2):

2. На вход программы поступает строка из целых чисел, записанных через пробел. Напишите функцию `get_list`, которая преобразовывает эту строку в список из целых чисел и возвращает его. Определите декоратор для этой функции, который сортирует список чисел, полученный из вызываемой в нем функции. Результат сортировки должен возвращаться при вызове декоратора. Вызовите декорированную функцию get_list и отобразите полученный отсортированный список на экране.

Код:

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

def sort_list(func):
    """
    Сортировка списка
    """
    def wrapper(not_s_l):
        sort_line = sorted(func(not_s_l))
        return sort_line
    return wrapper


@sort_list
def get_list(line):
    """
    Создание списка из строки цифр разделенных пробелами
    """
    new_line = [int(i) for i in line.split()]
    return new_line


if __name__ == '__main__':
    s = input("Enter line: ")
    print(get_list(s))
```

```
Enter line: 4 0 9 1 8
[0, 1, 4, 8, 9]

Process finished with exit code 0
```

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 называется элементы, с которыми можно делать всё то же, что и с любым другим объектом: передавать, как параметр, возвращать из функции и присваивать переменной.

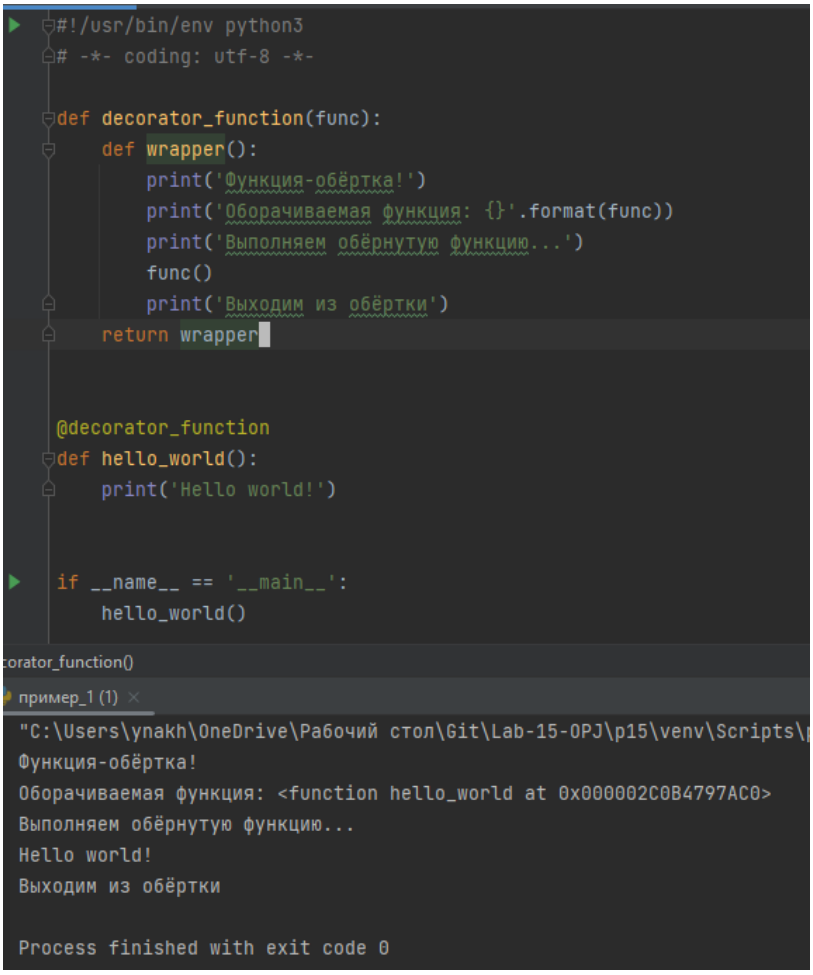
3. Каково назначение функций высших порядков?

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

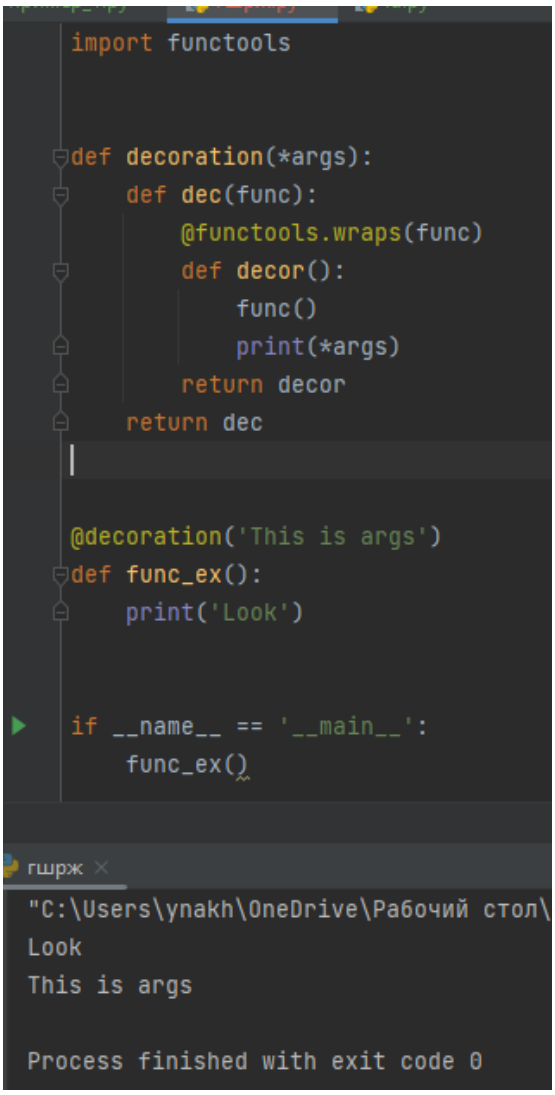
4. Как работают декораторы?

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?



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



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