МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение высшего образования
«СЕВЕРОКАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Кафедра

инфокоммуникаций

Институт цифрового

развития

ОТЧЁТ

по лабораторной работе №2.12

Дисциплина: «Основы программной инженерии»

Тема: «Декораторы функций в языке Python»

Выполнил:

студент 2 курса

группы Пиж-б-о-21-1

Коныжев Максим

Викторович

Ставрополь 2022

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

1. Был создан репозиторий в Github в который были добавлены правила gitignore для работы IDE PyCharm, была выбрана лицензия MIT, сам репозиторий был клонирован на локальный сервер и был организован в соответствии с моделью ветвления git-flow.

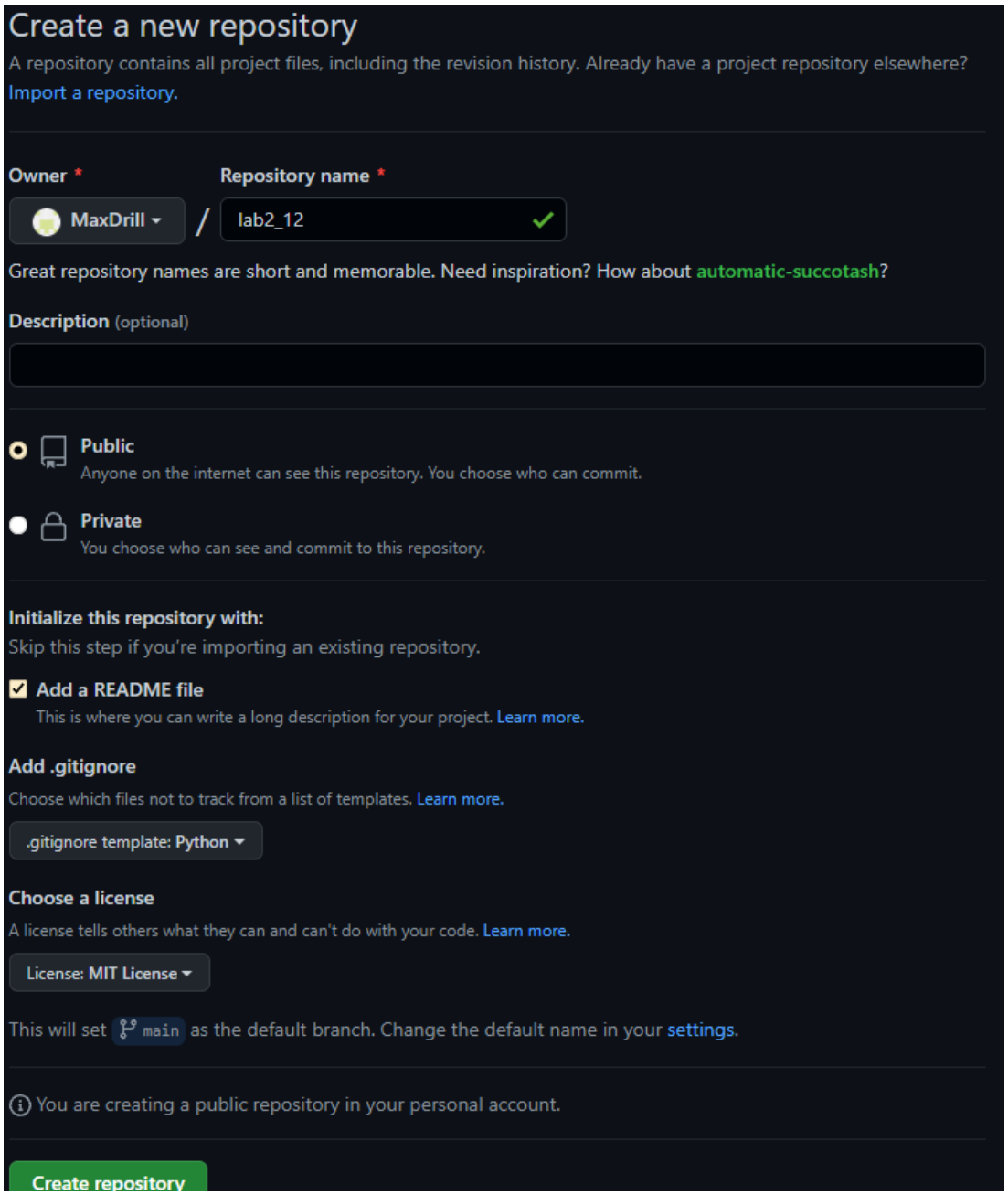

Рисунок 1 – Создание репозитория

```
C:\Users\UESR\gitproj>git clone https://github.com/MaxDrill/lab2_12.git
Cloning into 'lab2_12'...
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.
```

Рисунок 2 – Клонирование репозитория

```
C:\Users\UESR\gitproj\lab2_12>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/UESR/gitproj/lab2_12/.git/hooks]

C:\Users\UESR\gitproj\lab2_12>
```

Рисунок 3 – Организация репозитория с моделью ветвления git-flow

2. Проработайте примеры лабораторной работы.

Пример 1.

```python
#!/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()
```

```
C:\Users\UESR\Desktop\2.1\proj\venv\Scripts\python.exe C:/Users/UES
Функция-обёртка!
Оборачиваемая функция: <function hello_world at 0x014ABE88>
Выполняем обёрнутую функцию...
Hello world!
Выходим из обёртки

Process finished with exit code 0
```

Рисунок 4 – Результат работы примера

Пример 2.

```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()
```

```
C:\Users\UESR\Desktop\2.1\proj\venv\Scripts\python.exe
[*] Время выполнения: 0.9268765449523926 секунд.

Process finished with exit code 0
```

Рисунок 5 – Результат работы примера

3. Индивидуальное задание

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 программы:

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


def sort_list(func):
    """
    Сортировка списка
    """
    def wrapper(*args):
        arg = args[0]
        sort_line = sorted(func(arg))
        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))
```

```
C:\Users\UESR\Desktop\2.1\proj\venv\Scripts\python.
Enter line: 3 5 1 0 9
[0, 1, 3, 5, 9]

Process finished with exit code 0
```

Рисунок 6 – Результат работы программы

Вывод: в результате лабораторной работы были приобретены навыки по работе с замыканиями при написании программ с помощью языка программирования Python версии 3.x.

Контрольные вопросы

1. Что такое замыкание?

Замыкание – это функция, в теле которой присутствуют ссылки на переменные, объявленные вне тела этой функции.

2. Как реализованы замыкания в языке программирования Python?

Замыканием в языке Python называется функция, вложенная в другую функцию и использующая переменные внешней функции.

3. Что подразумевает под собой область видимости Local?

Переменный с областью видимости Local (локальные переменные могут быть использованы только внутри того блока кода, где она была объявлена.

4. Что подразумевает под собой область видимости Enclosing?

Для вложенных функций переменные из функции более высокого уровня имеют данную область видимости.

5. Что подразумевает под собой область видимости Global?

Область видимости Global означает, что данная переменная может быть использована (видна) во всём модуле (файле с расширением .py).

6. Что подразумевает под собой область видимости Build-in?

Это переменный уровня интерпретатора. Для их использования не нужно импортировать модули.