

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 №2.12
дисциплины «Основы программной инженерии»

Выполнил:
Юрьев Илья Евгеньевич
2 курс, группа ПИЖ-б-о-22-1,
09.03.04 «Программная инженерия»,
направленность (профиль) «Разработка
и сопровождение программного
обеспечения», очная форма обучения

(подпись)

Руководитель практики:
Богданов С.С., ассистент кафедры
инфокоммуникаций

(подпись)

Отчет защищен с оценкой _____ Дата защиты _____

Ставрополь, 2023 г.

Тема: 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.

Ход выполнения работы:

1. Создать общедоступный репозиторий на GitHub, в котором будет использована лицензия MIT и язык программирования Python:

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 ().*

Repository template

No template ▾

Start your repository with a template repository's contents.


Owner * **Repository name ***


 daxstrong ▾ / lr2_12

✔ lr2_12 is available.

Great repository names are short and memorable. Need inspiration? How about [fuzzy-enigma](#) ?

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.

Рисунок 1 – Создание репозитория с заданными настройками

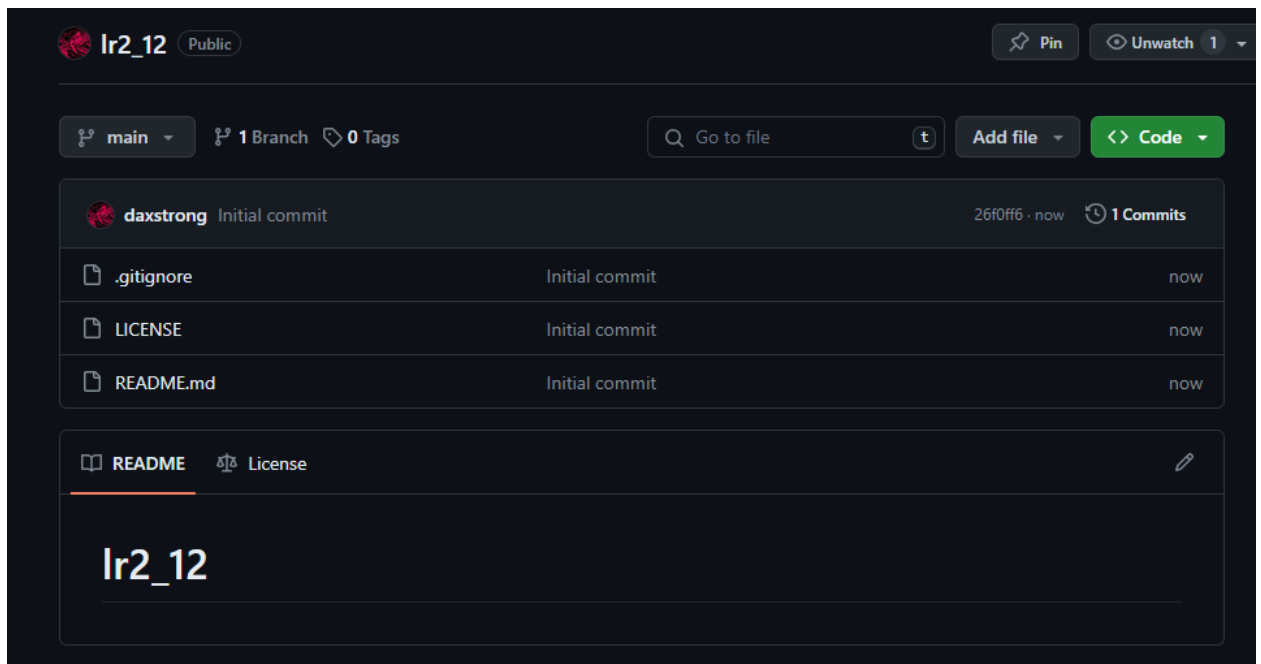


Рисунок 2 – Созданный репозиторий

```
ilyay@DESKTOP-FF1JT6S MINGW64 ~/OneDrive/Рабочий стол/Основы программной инженерии
$ git clone https://github.com/daxstrong/lr2_12.git
Cloning into 'lr2_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 3 – Клонирование репозитория

```
ilyay@DESKTOP-FF1JT6S MINGW64 ~/OneDrive/Рабочий стол/Основы программной инженерии/lr2_12 (main)
$ git checkout -b develop
Switched to a new branch 'develop'
```

Рисунок 4 – Создание ветки develop

2. Проработать примеры лабораторной работы, оформляя код согласно PEP-8:

```
1 ▶ #!/usr/bin/env python3
2 # -*- coding: utf-8 -*-
3
4 def benchmark(func):
5     import time
6
7     def wrapper(*args, **kwargs):
8         start = time.time()
9         return_value = func(*args, **kwargs)
10        end = time.time()
11        print('[*] Время выполнения: {} секунд.'.format(end - start))
12        return return_value
13
14    return wrapper
15
16
17    @benchmark
18    def fetch_webpage(url):
19        import requests
20        webpage = requests.get(url)
21        return webpage.text
22
23
24 ▶ if __name__ == '__main__':
25     webpage = fetch_webpage('https://google.com')
26     print(webpage)
```

Рисунок 5 – Пример №1

[illegible]

Рисунок 6 – Вывод программы (Пример №1)

3. Выполним 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.

```
1  ▶  #!/usr/bin/env python3
2      # -*- coding: utf-8 -*-
3
4      def sort_list_decorator(func):
5          def wrapper(string_input):
6              numbers = func(string_input)
7              sorted_numbers = sorted(numbers)
8              return sorted_numbers
9
10         return wrapper
11
12
13     @sort_list_decorator
14     def get_list(string_input):
15         return list(map(int, string_input.split()))
16
17
18  ▶  if __name__ == "__main__":
19       input_string = input("Введите строку из целых чисел, разделенных пробелами: ")
20       sorted_result = get_list(input_string)
21       print("Отсортированный список:", sorted_result)
22
```

Рисунок 7 – Решение индивидуального задания

```
C:\Users\ilyay\AppData\Local\Microsoft\WindowsApps\python3.11.exe "C:/Users
Введите строку из целых чисел, разделенных пробелами: 3 1 66 11
Отсортированный список: [1, 3, 11, 66]
```

Рисунок 8 – Вывод программы (Индивидуальное задание)

Объяснение кода:

Код включает декоратор `sort_list_decorator`, который сортирует список, возвращаемый функцией `get_list`. Декоратор принимает строку ввода,

преобразует ее в список целых чисел с помощью `get_list`, сортирует этот список и возвращает результат.

4. Зафиксируем сделанные изменения, сольем ветки и отправим на удаленный репозиторий:

```
ilyay@DESKTOP-FF1JT6S MINGW64 ~/OneDrive/Рабочий стол/Основы программной инженерии/lr2_12 (develop)
$ git log --oneline
f843498 (HEAD -> develop) Финальные изменения
26f0ff6 (origin/main, origin/HEAD, main) Initial commit
```

Рисунок 15 – Коммиты ветки `develop` во время выполнения лабораторной Работы

```
ilyay@DESKTOP-FF1JT6S MINGW64 ~/OneDrive/Рабочий стол/Основы программной инженерии/lr2_12 (develop)
$ git checkout main
Switched to branch 'main'
Your branch is up to date with 'origin/main'.

ilyay@DESKTOP-FF1JT6S MINGW64 ~/OneDrive/Рабочий стол/Основы программной инженерии/lr2_12 (main)
$ git merge develop
Updating 26f0ff6..f843498
Fast-forward
 .idea/.gitignore           | 8 ++++++
 .idea/inspectionProfiles/profiles_settings.xml | 6 +++++
 .idea/lr2_12.iml          | 8 ++++++
 .idea/misc.xml             | 4 ++++
 .idea/modules.xml          | 8 ++++++
 .idea/vcs.xml              | 6 +++++
 ex1.py                     | 26 ++++++
 individual.py              | 21 ++++++
8 files changed, 87 insertions(+)
create mode 100644 .idea/.gitignore
create mode 100644 .idea/inspectionProfiles/profiles_settings.xml
create mode 100644 .idea/lr2_12.iml
create mode 100644 .idea/misc.xml
create mode 100644 .idea/modules.xml
create mode 100644 .idea/vcs.xml
create mode 100644 ex1.py
create mode 100644 individual.py
```

Рисунок 16 – Слияние веток `main` и `develop`

```

ilyay@DESKTOP-FF1JT6S MINGW64 ~/OneDrive/Рабочий стол/Основы программной инженерии/lr2_12 (main)
$ git push origin main
Enumerating objects: 13, done.
Counting objects: 100% (13/13), done.
Delta compression using up to 12 threads
Compressing objects: 100% (11/11), done.
Writing objects: 100% (12/12), 2.28 KiB | 2.28 MiB/s, done.
Total 12 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/daxstrong/lr2_12.git
26f0ff6..f843498 main -> main

```

Рисунок 17 – Отправка изменений на удаленный репозиторий

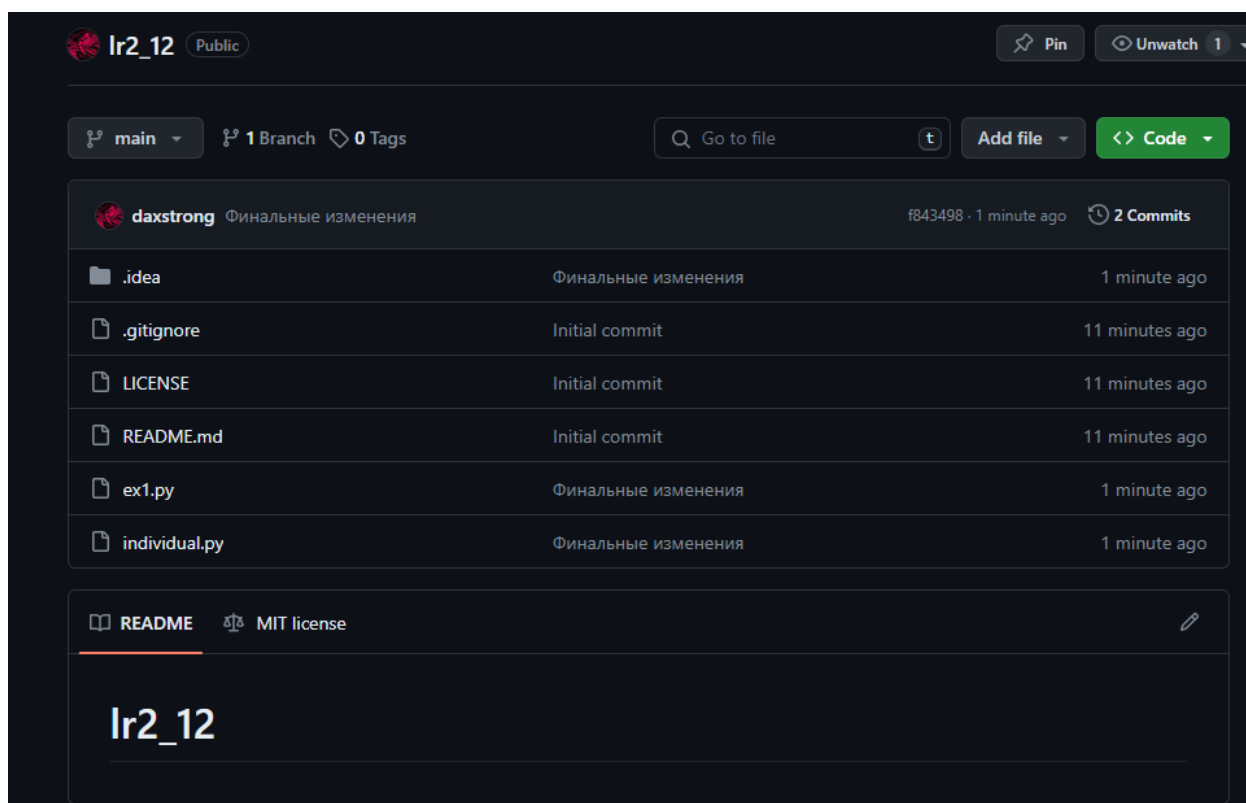


Рисунок 18 – Изменения удаленного репозитория

Ответы на контрольные вопросы:

1. Что такое декоратор?

Декоратор – это функция в Python, которая позволяет изменить поведение другой функции без изменения её кода. Он используется для добавления функциональности к существующей функции, обертывая её вокруг другой функции.

2. Почему функции являются объектами первого класса?

Функции в Python являются объектами первого класса, потому что они могут быть присвоены переменным, переданы как аргументы в функции, возвращены из другой функции и имеют те же свойства, что и другие типы данных в Python.

3. Каково назначение функций высших порядков?

Функции высших порядков – это функции, которые могут принимать другие функции в качестве аргументов или возвращать их как результат. Они позволяют абстрагировать действия и работать с функциями как с данными.

4. Как работают декораторы?

Декораторы работают путем обертывания одной функции внутри другой. Это позволяет изменять поведение декорируемой функции, не изменяя её код.

5. Какова структура декоратора функций?

Декоратор функции – это функция, которая принимает другую функцию в качестве аргумента, обычно использует внутреннюю функцию (wrapper), которая оборачивает оригинальную функцию и возвращает эту внутреннюю функцию. Пример:

```
def my_decorator(func):  
    def wrapper(*args, **kwargs):  
        # Дополнительный код до выполнения функции  
        result = func(*args, **kwargs)  
        # Дополнительный код после выполнения функции  
        return result  
    return wrapper
```

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

В Python можно передать параметры декоратору, используя дополнительную обертку. Можно создать функцию-декоратор, которая принимает аргументы и возвращает другую функцию, которая уже будет декоратором. Например:

```
def decorator_with_args(arg1, arg2):
    def decorator(func):
        def wrapper(*args, **kwargs):
            print(f"Arguments passed to decorator: {arg1}, {arg2}")
            return func(*args, **kwargs)
        return wrapper
    return decorator

@decorator_with_args("Hello", "World")
def my_function(x, y):
    return x + y

result = my_function(3, 5)
print(f"Result: {result}")
```