

**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ**

**Федеральное государственное автономное
образовательное учреждение высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ
УНИВЕРСИТЕТ»**

**Кафедра инфокоммуникаций
«Декораторы функций в Python»**

**Отчет по лабораторной работе № 2.12
по дисциплине «Программирование на Python»**

Выполнил студент группы ИВТ-б-о-21-1

Сумин Никита Сергеевич.

«3 » ноября 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.


Порядок выполнения работы:

1. Создал общедоступный репозиторий на GitHub, в котором 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.](#)

Owner *

 BrandooDi ▾

Repository name *

/ mywork2.12 ✓

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

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:

Skip this step if you're importing an existing repository.

☒

Add a README file

This is where you can write a long description for your project. [Learn more.](#)

Add .gitignore


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

.gitignore template: Python ▾

Choose a license

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

License: MIT License ▾

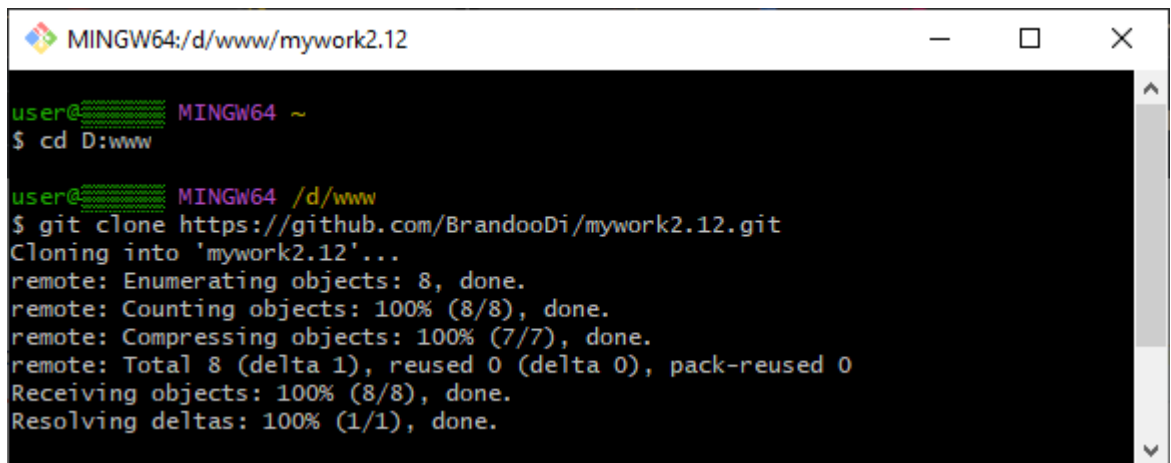
This will set  main as the default branch. Change the default name in your [settings](#).

 You are creating a public repository in your personal account.

Create repository

Рисунок 1 - Создание репозитория

2. Выполните клонирование созданного репозитория.

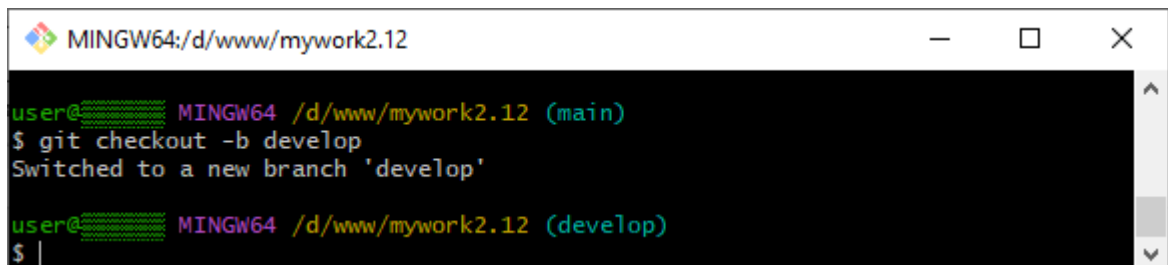


```
MINGW64:/d/www/mywork2.12
user@MINGW64 ~
$ cd D:\www

user@MINGW64 /d/www
$ git clone https://github.com/BrandooDi/mywork2.12.git
Cloning into 'mywork2.12'...
remote: Enumerating objects: 8, done.
remote: Counting objects: 100% (8/8), done.
remote: Compressing objects: 100% (7/7), done.
remote: Total 8 (delta 1), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (8/8), done.
Resolving deltas: 100% (1/1), done.
```

Рисунок 2 - Клонирование репозитория

3. Организуйте свой репозиторий в соответствии с моделью ветвления git-flow.



```
MINGW64:/d/www/mywork2.12
user@MINGW64 /d/www/mywork2.12 (main)
$ git checkout -b develop
Switched to a new branch 'develop'

user@MINGW64 /d/www/mywork2.12 (develop)
$ |
```

Рисунок 3 - Ветвление по модели git-flow 4.

Создайте проект PyCharm в папке репозитория.

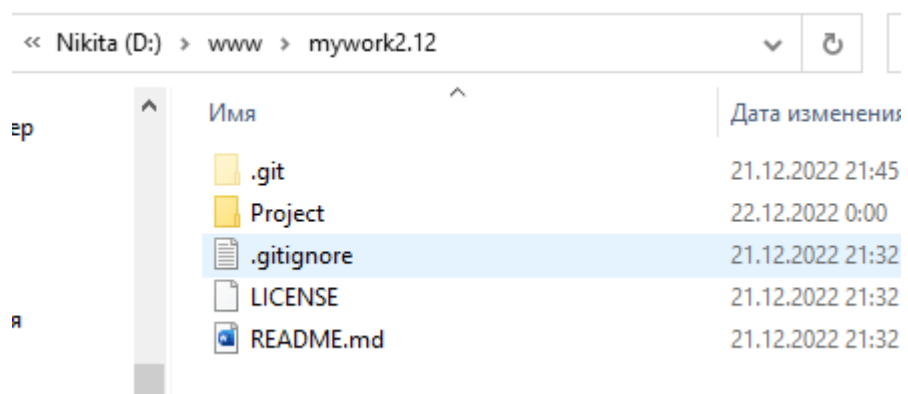


Рисунок 4 - Проект PyCharm

5. Проработайте примеры лабораторной работы. Создайте для него отдельный модуль языка Python. Зафиксируйте изменения в репозитории.

Функции как объекты первого класса можно определять внутри других функций.

```
D:\www\mywork2.12\Project\venv\Scripts\python.exe D:/www/mywork2.12/Project/pr1.py
Hello world!

Process finished with exit code 0
```

Рисунок 5 - Результат выполнения примера 1

Функции можно передавать в качестве аргумента и возвращать их из других функций.

```
D:\www\mywork2.12\Project\venv\Scripts\python.exe D:/www/mywork2.12/Project/pr2.py
Получена функция <function hello_world at 0x00000256073C3E20> в качестве аргумента
Hello world!

Process finished with exit code 0
```

Рисунок 6 - Результат выполнения примера 2

Выражение `@decorator_function` вызывает `decorator_function()` с `hello_world` в качестве аргумента и присваивает имени `hello_world` возвращаемую функцию.

```
D:\www\mywork2.12\Project\venv\Scripts\python.exe D:/www/mywork2.12/Project/pr3.py
Функция-обёртка!
Оборачиваемая функция: <function hello_world at 0x000001F662725630>
Выполняем обёрнутую функцию...
Hello world!
Выходим из обёртки

Process finished with exit code 0
```

Рисунок 7 - Результат выполнения примера 3

Создаём декоратор, замеряющий время выполнения функции. Далее мы используем его на функции, которая делает GET-запрос к главной странице Google. Чтобы измерить скорость, мы сначала сохраняем время перед выполнением обёрнутой функции, выполняем её, снова сохраняем текущее время и вычитаем из него начальное.

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?

Функция-декоратор является функцией высшего порядка, так как принимает функцию в качестве аргумента, а также возвращает функцию. Внутри функции-декоратора определяется другая функция, обёртка, так сказать, которая обёртывает функцию-аргумент и затем изменяет её поведение. Декоратор возвращает эту обёртку.

5. Какова структура декоратора функций? def

```
decorator_function(func):  
    def wrapper():  
    func()    return  
    wrapper  
    @decorator_func  
    tion def  
    function():
```

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

Чтобы получить декоратор, в который можно передать аргументы, нужно из функции с параметрами вернуть функциональный объект, который может действовать как декоратор. Обычно декоратор создает и возвращает внутреннюю функцию-обертку, следовательно полный пример даст внутреннюю функцию внутри внутренней функции.

Вывод: были приобретены навыки по работе с декораторами функций при написании программ с помощью языка программирования Python версии 3.x.