

**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

Шайдеров Дмитрий Викторович.

«3 » ноября 2022г.

Подпись студента \_\_\_\_\_

Работа защищена «    » \_\_\_\_\_ 20\_\_ г.

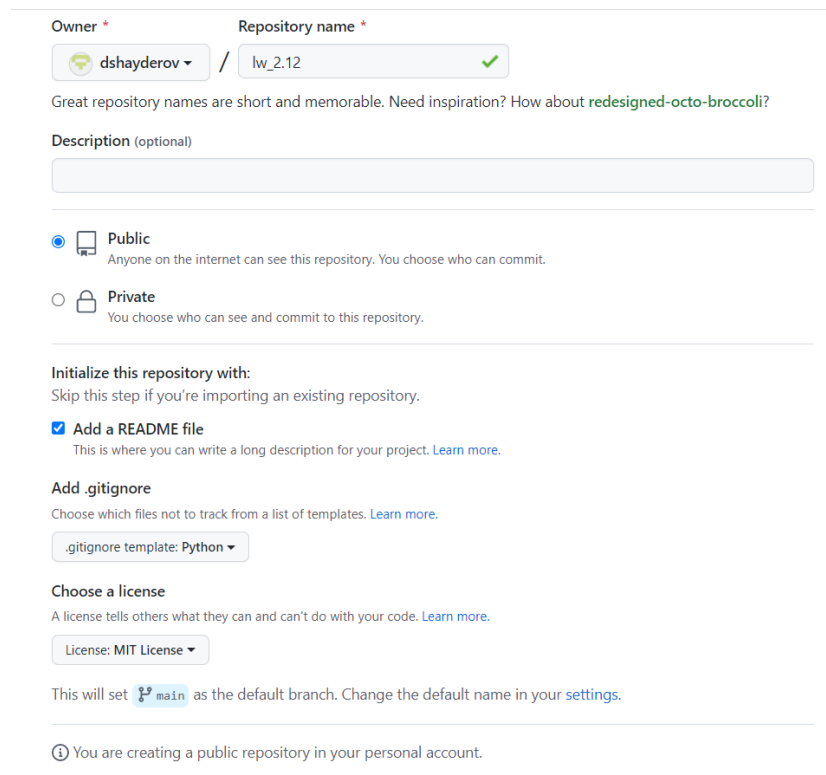
Проверил Воронкин Р.А. \_\_\_\_\_  
(подпись)

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.



Owner \* / Repository name \*

dshayderov / lw\_2.12 ✓

Great repository names are short and memorable. Need inspiration? How about [redesigned-octo-broccoli](#)?

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.

Рисунок 1 - Создание репозитория

2. Выполните клонирование созданного репозитория.

```
Asus@LAPTOP-09A994CG MINGW64 /c/my projects/3
$ git clone https://github.com/dshayderov/lw_2.12.git
Cloning into 'lw_2.12'...
remote: Enumerating objects: 11, done.
remote: Counting objects: 100% (11/11), done.
remote: Compressing objects: 100% (10/10), done.
remote: Total 11 (delta 2), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (11/11), done.
Resolving deltas: 100% (2/2), done.
```

Рисунок 2 - Клонирование репозитория

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

```

Asus@LAPTOP-09A994CG MINGW64 /c/my projects/3/lw_2.12 (main)
$ git checkout -b develop
Switched to a new branch 'develop'

Asus@LAPTOP-09A994CG MINGW64 /c/my projects/3/lw_2.12 (develop)
$ |

```

Рисунок 3 - Ветвление по модели git-flow

4. Создайте проект PyCharm в папке репозитория.

Этот компьютер > OS (C:) > My projects > 3 > lw_2.12 >				
Имя	Дата изменения	Тип	Размер	
Project	10.11.2022 22:12	Папка с файлами		
.gitignore	10.11.2022 22:09	Файл "GITIGNORE"	2 КБ	
LICENSE	10.11.2022 22:09	Файл	2 КБ	
README.md	10.11.2022 22:09	Файл "MD"	1 КБ	

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.

```

primer_1 x
"C:\My projects\3\lw_2.12\Project\venv\Scripts\python.exe" "C:/My projects/3/lw_2.12/Project/hello_world.py"
Hello world!

```

Рисунок 5 - Результат выполнения примера 1

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

```

primer_2 x
"C:\My projects\3\lw_2.12\Project\venv\Scripts\python.exe" "C:/My projects/3/lw_2.12/Project/hello_world.py"
Получена функция <function hello_world at 0x0000022A1EADF040> в качестве аргумента
Hello world!

```

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.

```
primer_3 x
"C:\My projects\3\lw_2.12\Project\venv\Scripts\python.exe" "C:/My pro
Функция-обёртка!
Оборачиваемая функция: <function hello_world at 0x00000189E800D0D0>
Выполняем обёрнутую функцию...
Hello world!
Выходим из обёртки
```

Рисунок 7 - Результат выполнения примера 3

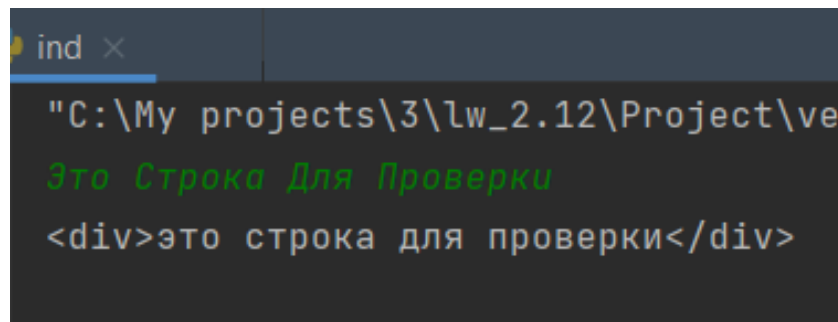
Создаём декоратор, замеряющий время выполнения функции. Далее мы используем его на функции, которая делает GET-запрос к главной странице Google. Чтобы измерить скорость, мы сначала сохраняем время перед выполнением обёрнутой функции, выполняем её, снова сохраняем текущее время и вычитаем из него начальное.

```
primer_4 x
"C:\My projects\3\lw_2.12\Project\venv\Scripts\python.exe" "C:/My projects/3/lw_2.12/Project/primer_4.py"
[*] Время выполнения: 1.602891206741333 секунд.
<!doctype html><html itemscope="" itemtype="http://schema.org/WebPage" lang="ru"><head><meta content="#105
var f=this||self;var h,k=[];function l(a){for(var b;a&&!(a.getAttribute||(b=a.getAttribute("eid"))));)a=a.p
function n(a,b,c,d,g){var e="";c||-1!=b.search("&ei=")||e="&ei="+l(d),-1==b.search("&lei=")&&(d=m(d))&&
document.documentElement.addEventListener("submit",function(b){var a;if(a=b.target){var c=a.getAttribute("c
</style><style>body,td,a,p,.h{font-family:arial,sans-serif}body{margin:0;overflow-y:scroll}#gog{padding:3px
var h=this||self;var k,l=null!==(k=h.mei)?k:1,n,p=null!==(n=h.sdo)?n:!0,q=0,r,t=google.erd,v=t.jsr;google.ml=
a.fileName;g&&(0<g.indexOf("-extension:/")&&(e=3),c+="&script="+b(g),f&&g==window.location.href&&(f=document
if (!iesg){document.f&&document.f.q.focus();document.gbqf&&document.gbqf.q.focus();}
```

Рисунок 8 - Результат выполнения примера 4

#### 6. Решите индивидуальное задание. (Вариант 26/6)

Объявите функцию, которая возвращает переданную ей строку в нижнем регистре (с малыми буквами). Определите декоратор для этой функции, который имеет один параметр `tag`, определяющий строку с названием тега (начальное значение параметра `tag` равно `h1`). Этот декоратор должен заключать возвращенную функцией строку в тег `tag` и возвращать результат. Пример заключения строки "python" в тег `h1`: `<h1>python</h1>` . Примените декоратор со значением `tag="div"` к функции и вызовите декорированную функцию. Результат отобразите на экране.



```
ind x
"C:\My projects\3\lw_2.12\Project\ve
Это Строка Для Проверки
<div>это строка для проверки</div>
```

Рисунок 7 - Результат выполнения индивидуального задания

## Контрольные вопросы:

### 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?

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_function
```

```
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.