

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: «Программирование на Python»

Тема: «Декораторы функций в языке Python»

Вариант 8

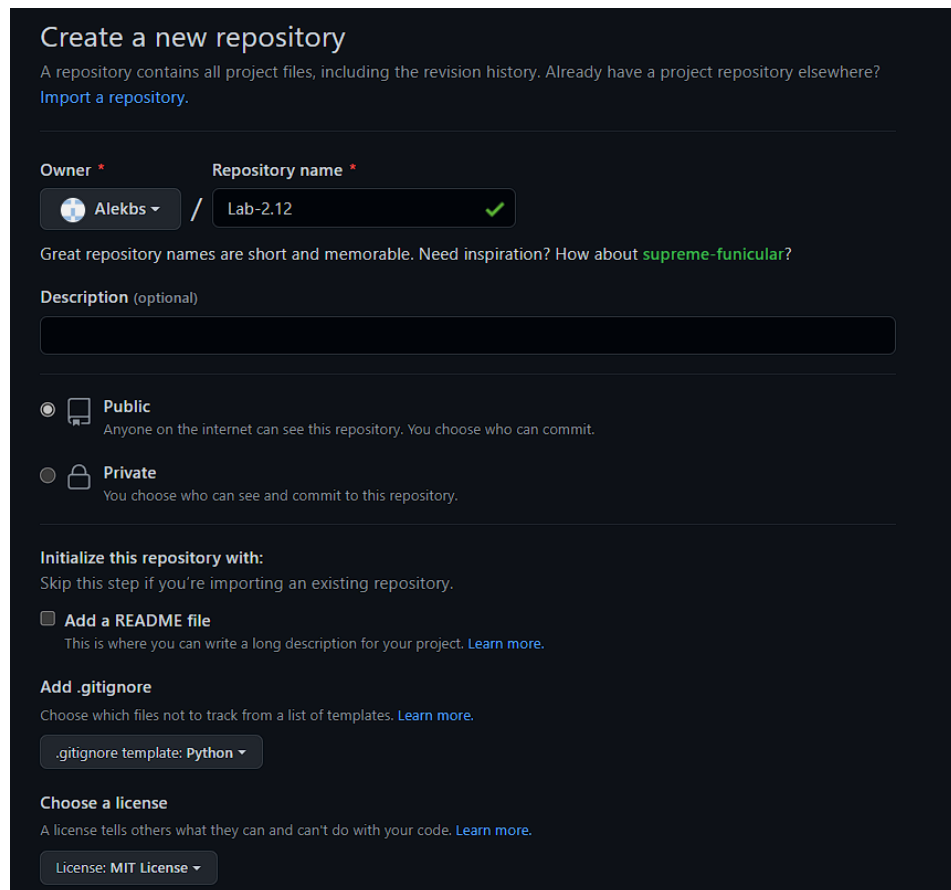
Выполнила: студент 2 курса,
группы ИВТ-б-о-21-1
Богдан Александр
Анатольевич

Ставрополь 2022

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

Практическая часть:

1. Создал общедоступный репозиторий на GitHub.



Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

Owner * Repository name *

Aleks / Lab-2.12 ✓

Great repository names are short and memorable. Need inspiration? How about [supreme-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

Рисунок 1. Создание репозитория

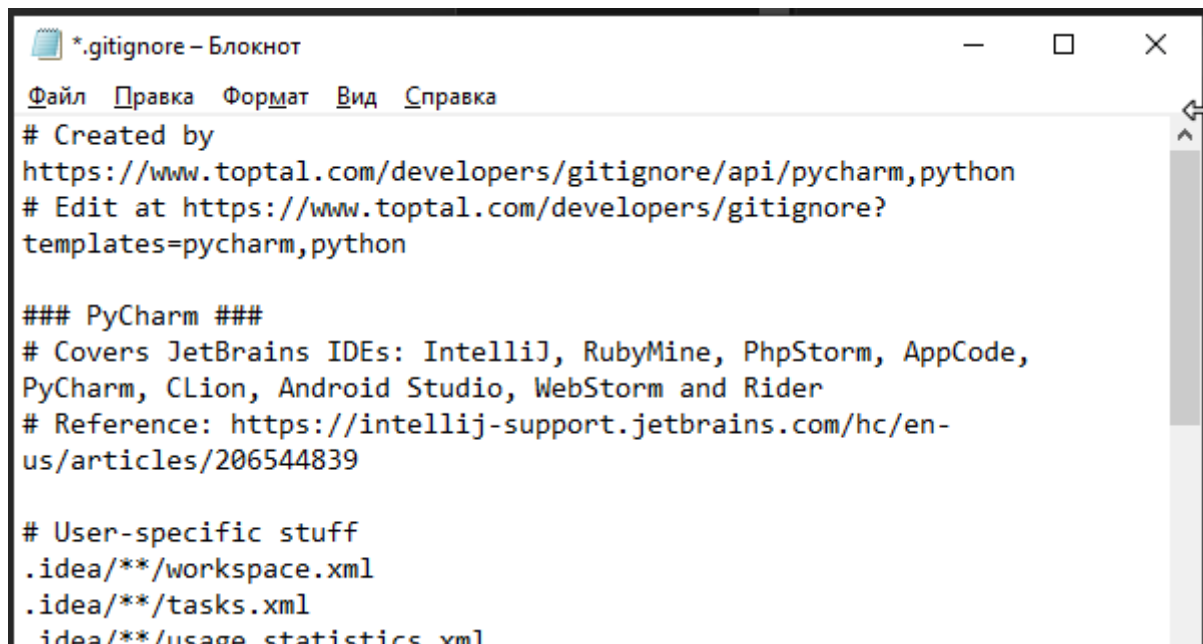
2. Выполнил клонирование созданного репозитория.

```
Microsoft Windows [Version 10.0.19044.2251]
(c) Корпорация Майкрософт (Microsoft Corporation). Все права защищены.

D:\WAR\gi>git clone https://github.com/Alekbs/Lab-2.12
Cloning into 'Lab-2.12'...
remote: Enumerating objects: 4, done.
remote: Counting objects: 100% (4/4), done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 4 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (4/4), done.
```

Рисунок 2. Клонирование репозитория

3. Дополнил файл .gitignore.



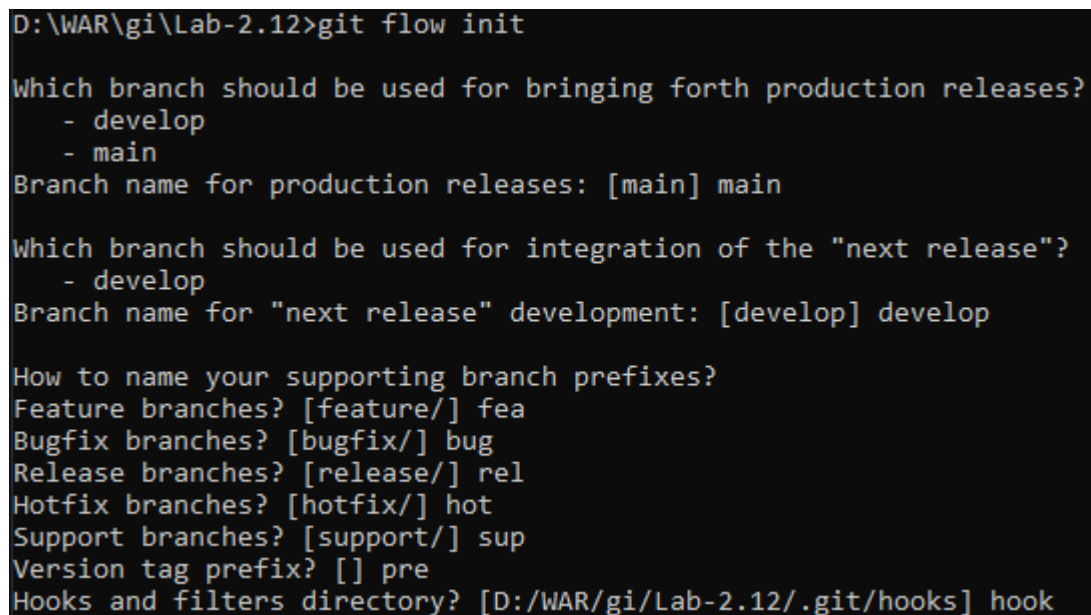
```
*.gitignore – Блокнот
Файл  Правка  Формат  Вид  Справка
# Created by
https://www.toptal.com/developers/gitignore/api/pycharm,python
# Edit at https://www.toptal.com/developers/gitignore?
templates=pycharm,python

### PyCharm ###
# Covers JetBrains IDEs: IntelliJ, RubyMine, PhpStorm, AppCode,
PyCharm, CLion, Android Studio, WebStorm and Rider
# Reference: https://intellij-support.jetbrains.com/hc/en-
us/articles/206544839

# User-specific stuff
.idea/**/workspace.xml
.idea/**/tasks.xml
.idea/**/usage.statistics.xml
```

Рисунок 3. Изменение файла .gitignore

4. Организовал свой репозиторий в соответствии с моделью ветвления git-flow.



```
D:\WAR\gi\Lab-2.12>git flow init

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

Which branch should be used for integration of the "next release"?
  - develop
Branch name for "next release" development: [develop] develop

How to name your supporting branch prefixes?
Feature branches? [feature/] fea
Bugfix branches? [bugfix/] bug
Release branches? [release/] rel
Hotfix branches? [hotfix/] hot
Support branches? [support/] sup
Version tag prefix? [] pre
Hooks and filters directory? [D:/WAR/gi/Lab-2.12/.git/hooks] hook
```

Рисунок 4. Организация репозитория в соответствии с git-flow

5. Проработал примеры лабораторной работы.

```
C:\Users\aleks\AppData\Local\F
Hello world!
```

Рисунок 5. Результат работы примера 1

```
Функция-обёртка!
Оборачиваемая функция: <function hello_world at 0x900002C934C75160>
Выполняем обёрнутую функцию...
Hello world!
Выходим из обёртки

Process finished with exit code 0
```

Рисунок 6. Результат работы примера 2

```
[*] Время выполнения: 1.9278357028961182 секунд.

Process finished with exit code 0
```

Рисунок 7. Результат работы примера 3

```
var a=window.innerWidth,b=window.innerHeight;if(!a||!b){var c=wind
var d=this||self,e=function(a){return a};
var g;var l=function(a,b){this.g=b==h?a:""};l.prototype.toString=
function p(a){google.timers&&google.timers.load&&google.tick&&goog
function _F_installCss(c){}
(function(){google.jl={blt:'none',chnk:0,dw:false,dwu:true,emtn:0,
Process finished with exit code 0
```

Рисунок 8. Результат работы примера 4

6. Выполнил 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.

```
C:\Users\aleks\AppData\Local\Programs  
[1, 2, 3, 4, 8, 9]
```

Рисунок 9. Результат работы 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. Вот почему декораторы можно рассматривать как практику метапрограммирования, когда программы могут работать с другими программами как со своими данными.

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. Каково назначение функций высших порядков?

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?

```
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
```

Здесь `decorator_function()` является функцией-декоратором. Как вы могли заметить, она является функцией высшего порядка, так как принимает функцию в качестве аргумента, а также возвращает функцию. Внутри `decorator_function()` мы определили другую функцию, обёртку, так сказать, которая обёртывает функцию-аргумент и затем изменяет её поведение. Декоратор возвращает эту обёртку. модифицировали её поведение. Однако выражение с `@` является всего лишь синтаксическим сахаром для `hello_world = decorator_function(hello_world)`. Иными словами, выражение `@decorator_function` вызывает `decorator_function()` с `hello_world` в качестве аргумента и присваивает имени `hello_world` возвращаемую функцию.

5. Какова структура декоратора функций?

Пример в 4 вопросе.

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

В декораторе можно передать и сам параметр. В этом случае нужно добавить еще один слой, то есть – еще одну функцию-обертку. Это обязательно, поскольку аргумент передается декоратору. Затем, функция, которая вернулась, используется для декорации нужной.

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