

**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 № 7**

Дисциплина: «Программирование на Python»

Тема: «Декораторы функций в языке Python»

Выполнил: студент 2 курса

группы ИТС-б-о-21-1

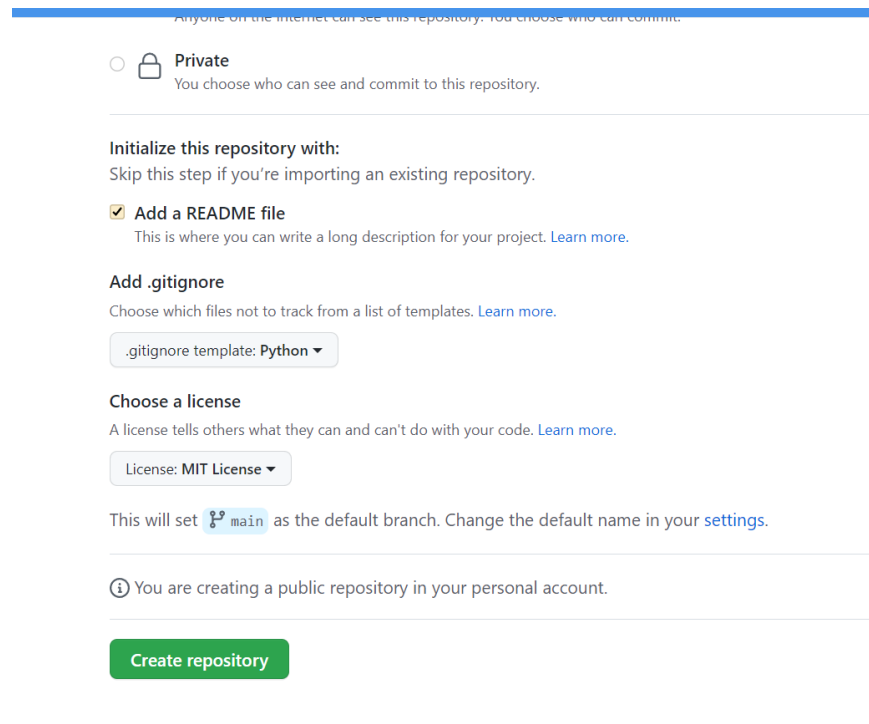
Гайибов Хасан Мамадиерович

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



Anyone on the internet can see this repository. You choose who can commit.

☒ **Public**  
Anyone on the internet can see this repository.

☐ **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](#).

**Create repository**

Рисунок 1. Создание репозитория

2. Выполните клонирование созданного репозитория.

3. Дополните файл .gitignore необходимыми правилами для работы с IDE PyCharm.



```
1 .idea/
2 # Created by https://www.toptal.com/developers/gitignore/api/python,pycharm
3 # Edit at https://www.toptal.com/developers/gitignore?templates=python,pycharm
4
5 ### PyCharm ###
6 # Covers JetBrains IDEs: IntelliJ, RubyMine, PhpStorm, AppCode, PyCharm, Clion, Android Studio, WebStorm and Rider
7 # Reference: https://intellij-support.jetbrains.com/hc/en-us/articles/206544839
8
9 # User-specific stuff
10 .idea/**/workspace.xml
11 .idea/**/tasks.xml
12 .idea/**/usage.statistics.xml
13 .idea/**/dictionaries
```

### Рисунок 3. Дополнение файла .gitignore

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

```
C:\Users\User\Desktop\2 кypc Python\lab 10\lab-10>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/User/Desktop/2 кypc Python\lab 10\lab-10/.git/hooks]
C:\Users\User\Desktop\2 кypc Python\lab 10\lab-10>
```

### Рисунок 4. Организован модель ветвления git flow

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

```
def benchmark(func):
    import time

    def wrapper(*args, **kwargs):
        start = time.time()
        return_value = func(*args, **kwargs)
        end = time.time()
        print('[*] Время выполнения: {} секунд.'.format(end - start))
        return return_value

    return wrapper

@benchmark
def fetch_webpage(url):
    import requests
    webpage = requests.get(url)
    return webpage.text

webpage = fetch_webpage('https://google.com')
print(webpage)
```

### Рисунок 5. Пример лаб работы

## 6. Индивидуальное задание

**Вариант 9.** Объявите функцию, которая принимает строку на кириллице и преобразовывает ее в латиницу, используя следующий словарь для замены русских букв на соответствующее латинское написание:

```
t = {'ё': 'yo', 'а': 'a', 'б': 'b', 'в': 'v', 'г': 'g', 'д': 'd', 'е': 'e', 'ж': 'zh', 'з': 'z', 'и': 'i', 'й': 'y', 'к': 'k', 'л': 'l', 'м': 'm', 'н': 'n', 'о': 'o', 'п': 'p', 'р': 'r', 'с': 's', 'т': 't', 'у': 'u', 'ф': 'f', 'х': 'h', 'ц': 'c', 'ч': 'ch', 'ш': 'sh', 'щ': 'shch', 'ъ': '', 'ы': 'y', 'ь': '', 'э': 'e', 'ю': 'yu', 'я': 'ya'}
```

Функция должна возвращать преобразованную строку. Замены делать без учета регистра (исходную строку перевести в нижний регистр – малые буквы). Определите декоратор с параметром `chars` и начальным значением " !?", который данные символы преобразует в символ "-" и, кроме того, все подряд идущие дефисы (например, "--" или "---" ) приводит к одному дефису. Полученный результат должен возвращаться в виде строки. Примените декоратор со значением `chars="?!;,. "` к функции и вызовите декорированную функцию. Результат отобразите на экране.

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

def pink(func):
    import re

    def pnk(chars=" !?"):
        print(chars)

        h = func(s)
        rx = re.compile('[ !?]*')
        hh = rx.sub('-', h)
        print(hh)
        ch = re.sub(r'[-+]', '-', hh)
        print(ch)

    return pnk

@pink
def wrapper(text):
    p = text.lower()
    t = {'ё': 'yo', 'а': 'a', 'б': 'b', 'в': 'v', 'г': 'g', 'д': 'd', 'е': 'e', 'ж': 'zh', 'з': 'z', 'и': 'i', 'й': 'y', 'к': 'k', 'л': 'l', 'м': 'm', 'н': 'n', 'о': 'o', 'п': 'p', 'р': 'r', 'с': 's', 'т': 't', 'у': 'u', 'ф': 'f', 'х': 'h', 'ц': 'c', 'ч': 'ch', 'ш': 'sh', 'щ': 'shch', 'ъ': '', 'ы': 'y', 'ь': '', 'э': 'e', 'ю': 'yu', 'я': 'ya'}
    return p.translate({ord(key): t[key] for key in t})

if __name__ == "__main__":
    s = 'Б щ --- ! ? - ?Барабанщик сильно занят, Барабанщик барабанит: \
    - Та-ра-ра, та-ра-ра,\
    На прогулку нам пора!'
```

```
x = wrapper(s)
```

## Рисунки 6. Выполненное индивидуальное задание

```
"C:\Users\User\Desktop\2 курс Python\lab 15\2.12\venv\Scripts\python.exe" "C:/Users/User/Desktop/2 курс Python/lab 15/2.12/indiv 1.py"
Б ш --- ! ? - ?Барабанщик сильно занят, Барабанщик барабанит: - Та-ра-ра, та-ра-ра, На прогулку нам пора!
b-shch-----barabanshchik-silno-zanyat,-barabanshchik-barabanit:-----ta-ra-ra,-ta-ra-ra,-----na-progulku-nam-pora-
b-shch-barabanshchik-silno-zanyat,-barabanshchik-barabanit:---ta-ra-ra,-ta-ra-ra,-na-progulku-nam-pora-

Process finished with exit code 0
```

## Рисунок 7. Результат выполненной работы

8. Сделала коммит, выполнил слияние с веткой main, и запустил изменения в уд. репозиторий.

```
C:\Users\User\Desktop\2 курс Python\lab 15\2.12>git add .
C:\Users\User\Desktop\2 курс Python\lab 15\2.12>git commit -m "new"
[main 30b7dca] new
 2 files changed, 61 insertions(+)
 create mode 100644 indiv 1.py
 create mode 100644 primer 1.py
C:\Users\User\Desktop\2 курс Python\lab 15\2.12>git push
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 8 threads
Compressing objects: 100% (4/4), done.
Writing objects: 100% (4/4), 1.18 KiB | 1.18 MiB/s, done.
Total 4 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/aikanyshkaukanbekova/2.12.git
 df86850..30b7dca main -> main
C:\Users\User\Desktop\2 курс Python\lab 15\2.12>
```

## Рисунок 8. Сохранения

### Ответы на 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?

Потому что с ними можно работать как с переменными, могут быть переданы как аргумент процедуры, могут быть возвращены как результат выполнения процедуры, могут быть включены в другие структуры данных.

#### 3. Каково назначение функций высших порядков?

Основной задачей функций высших порядков является возможность принимать в качестве аргументов и возвращать другие функции.

#### **4. Как работают декораторы?**

Они берут декорируемую функцию в качестве аргумента и позволяет совершать с ней какие-либо действия до и после того, что сделает эта функция, не изменяя её.

#### **5. Какова структура декоратора функций?**

Функция `decorator` принимает в качестве аргумента функцию `func`, внутри функции `decorator` другая функция `wrapper`. В конце декоратора происходит возвращение функции `wrapper`.

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

Достаточно обернуть функцию декоратор в другую функцию, которая будет принимать аргументы. И сделать вывод функций `wrapper` и `decorator`.