

**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ  
РОССИЙСКОЙ ФЕДЕРАЦИИ  
Федеральное государственное автономное  
образовательное учреждение высшего образования  
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»**

**Кафедра инфокоммуникаций**

**Отчет по лабораторной работе № 2.12  
по дисциплине «Основы программной инженерии»**

Выполнил студент группы ПИЖ-б-о-22-1

Душин Александр Владимирович.

Подпись студента \_\_\_\_\_

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

Проверил Воронкин Р.А. \_\_\_\_\_

(подпись)

Ставрополь 2023

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 с использованием лицензии 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 (\*).*


Owner \*

Repository name \*

 MrPlatynum

/


ProgrammEngineering15

 ProgrammEngineering15 is available.


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

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.

---

Create repository

Рисунок 1 – Создание общедоступного репозитория на GitHub с заданными настройками

```
Alexander@DESKTOP-IUJLQQ3 MINGW64 ~/Documents
$ git clone https://github.com/MrPlatynum/ProgrammEngineering15.git
Cloning into 'ProgrammEngineering15'...
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 2 – Клонирование созданного репозитория на локальный компьютер

```
env.bak/
venv.bak/

# Spyder project settings
.spyderproject
.spyproject

# Rope project settings
.ropeproject

# mkdocs documentation
/site

# mypy
.mypy_cache/
.dmypy.json
dmypy.json

# Pyre type checker
.pyre/

# pytype static type analyzer
.pytype/

# Cython debug symbols
cython_debug/

# PyCharm
# JetBrains specific template is maintained in a separate JetBrains.gitignore that can
# be found at https://github.com/github/gitignore/blob/main/Global/JetBrains.gitignore
# and can be added to the global gitignore or merged into this file. For a more nuclear
# option (not recommended) you can uncomment the following to ignore the entire idea folder.
.idea/
```

Рисунок 3 – файл .gitignore

```
Alexander@DESKTOP-IUJLQQ3 MINGW64 ~/Documents/ProgrammEngineering15 (main)
$ git checkout -b develop
Switched to a new branch 'develop'

Alexander@DESKTOP-IUJLQQ3 MINGW64 ~/Documents/ProgrammEngineering15 (develop)
$ |
```

Рисунок 4 – организация репозитория в соответствии с моделью ветвления git flow

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)
27

```

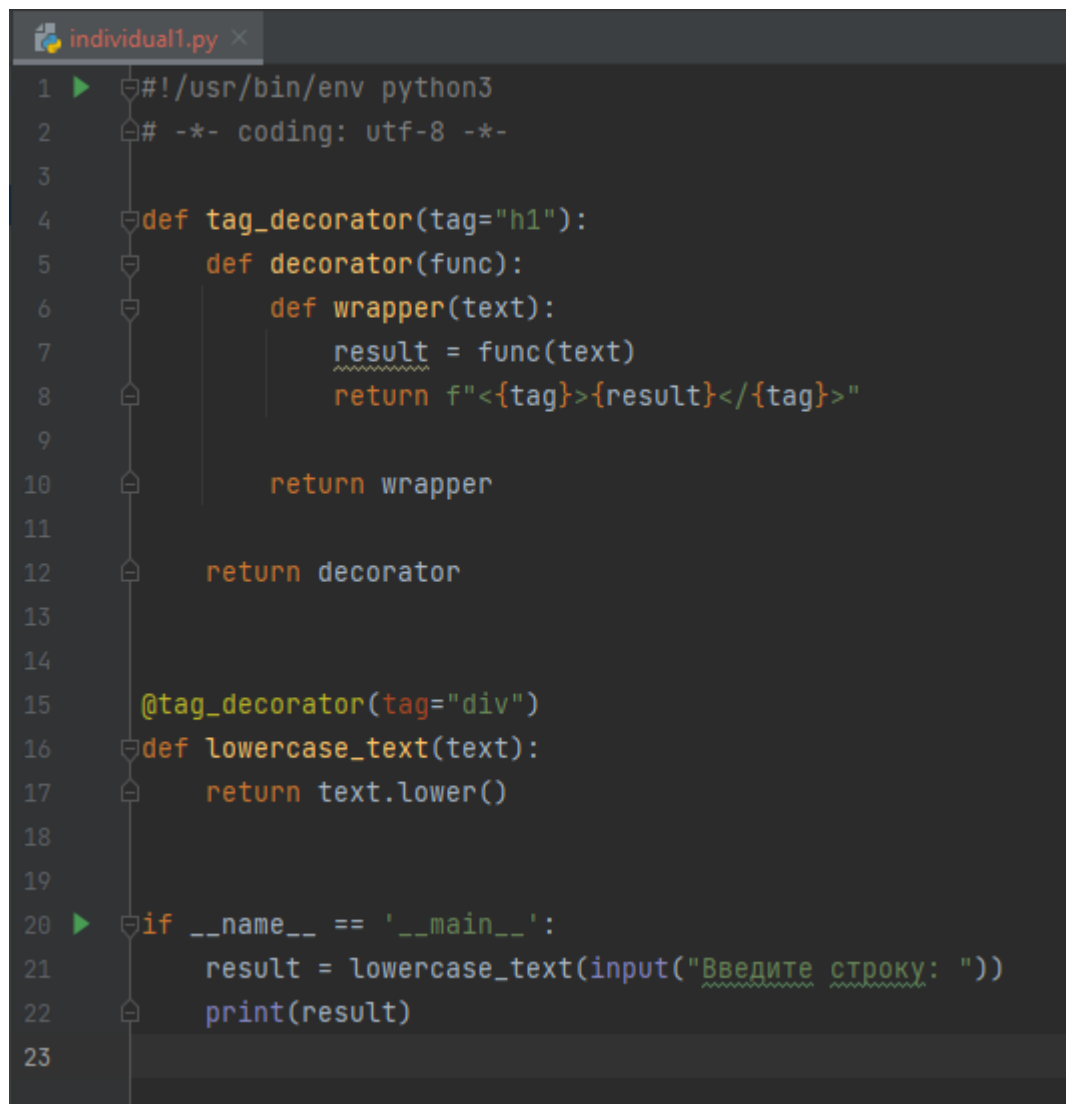
Рисунок 5 – Пример 1

[illegible]

### Рисунок 6 – Вывод программы

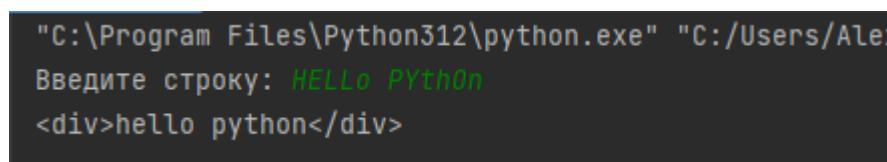
3. Объявите функцию, которая возвращает переданную ей строку в нижнем регистре (с малыми буквами). Определите декоратор для этой функции,

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.



```
individual1.py x
1  ▶  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3
4  def tag_decorator(tag="h1"):
5      def decorator(func):
6          def wrapper(text):
7              result = func(text)
8              return f"<{tag}>{result}</{tag}>"
9
10         return wrapper
11
12     return decorator
13
14
15     @tag_decorator(tag="div")
16     def lowercase_text(text):
17         return text.lower()
18
19
20  ▶  if __name__ == '__main__':
21      result = lowercase_text(input("Введите строку: "))
22      print(result)
23
```

Рисунок 7 – Решение индивидуального задания



```
"C:\Program Files\Python312\python.exe" "C:/Users/Alex
Введите строку: HELLO PYTHON
<div>hello python</div>
```

Рисунок 8 – Вывод программы

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

```
Alexander@DESKTOP-IUJLQQ3 MINGW64 ~/Documents/ProgrammEngineering15 (develop)
$ git log --oneline
758feb3 (HEAD -> develop) Финальные изменения
8d2f567 (origin/main, origin/HEAD, main) Initial commit
```

Рисунок 9 – Коммиты ветки develop во время выполнения лабораторной работы

```
Alexander@DESKTOP-IUJLQQ3 MINGW64 ~/Documents/ProgrammEngineering15 (develop)
$ git checkout main
Switched to branch 'main'
M       .gitignore
Your branch is up to date with 'origin/main'.

Alexander@DESKTOP-IUJLQQ3 MINGW64 ~/Documents/ProgrammEngineering15 (main)
$ git merge develop
Updating 8d2f567..758feb3
Fast-forward
 example1.py   | 26 ++++++
 individual1.py | 22 ++++++
 2 files changed, 48 insertions(+)
 create mode 100644 example1.py
 create mode 100644 individual1.py
```

Рисунок 10 – Слияние ветки develop в ветку main

```
Alexander@DESKTOP-IUJLQQ3 MINGW64 ~/Documents/ProgrammEngineering15 (main)
$ git push origin main
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 12 threads
Compressing objects: 100% (4/4), done.
Writing objects: 100% (4/4), 1012 bytes | 1012.00 KiB/s, done.
Total 4 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/MrPlatynum/ProgrammEngineering15.git
 8d2f567..758feb3  main -> main
```

Рисунок 11 – Отправка на удаленный репозиторий

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}")
```