

Министерство науки и высшего образования Российской Федерации  
Федеральное государственное автономное образовательное учреждение  
высшего образования  
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт цифрового развития  
Кафедра инфокоммуникаций

**ОТЧЕТ**  
**ПО ЛАБОРАТОРНОЙ РАБОТЕ №2.12**  
**дисциплины «Основы программной инженерии»**

Выполнил:  
Соколов Михаил Романович  
2 курс, группа ПИЖ-б-о-22-1,  
09.03.04 «Программная инженерия»,  
направленность (профиль) «Разработка  
и сопровождение программного  
обеспечения», очная форма обучения

---

(подпись)

Руководитель практики:  
Богданов С.С., ассистент кафедры  
инфокоммуникаций

---

(подпись)

Отчет защищен с оценкой \_\_\_\_\_ Дата защиты \_\_\_\_\_

Ставрополь, 2023 г.

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, в котором будет использована лицензия MIT и .gitignore файл для языка 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 (\*).*

**Repository template**

**No template** ▾

Start your repository with a template repository's contents.

**Owner \*** **Repository name \***

PoTtaTto ▾ / fundamentals\_of\_software

✔ fundamentals\_of\_software\_engineering\_lab2\_12 is available.

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

**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 репозитория с заданными настройками

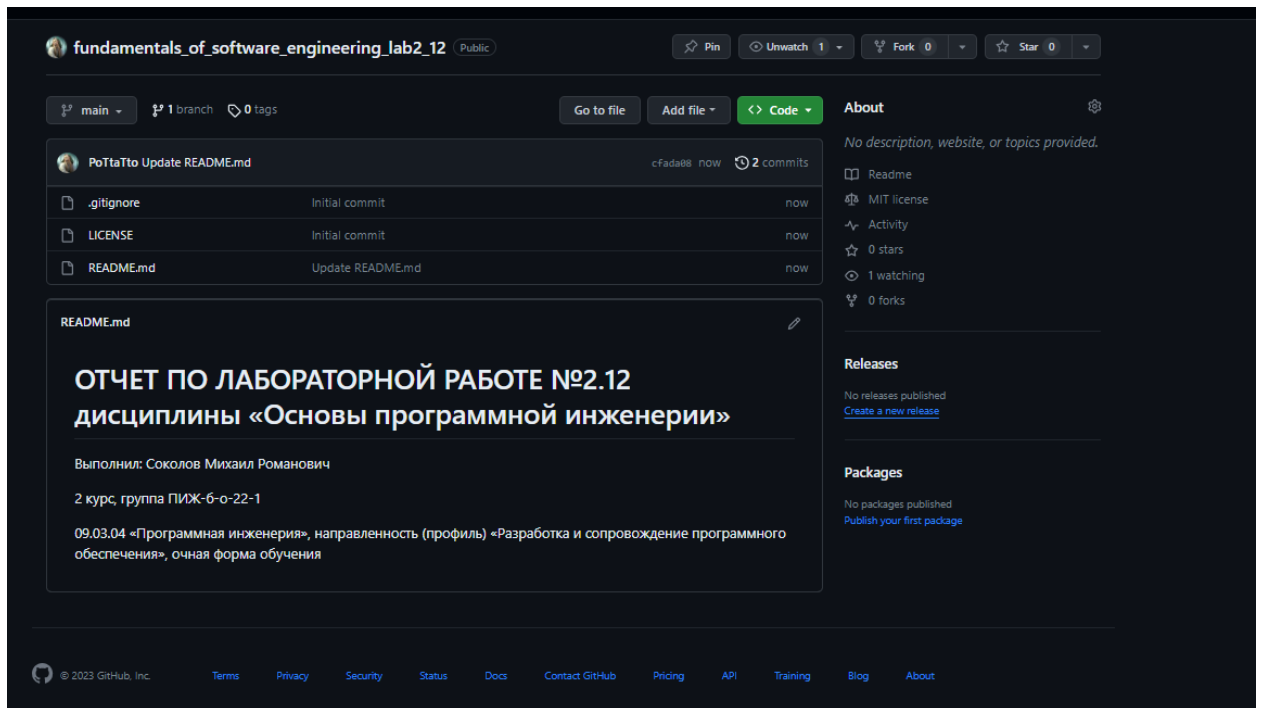


Рисунок 2 – Созданный репозиторий

```
PS C:\Study\СКФУ\Основы программной инженерии\Лабораторная работа 2.12> git clone https://github.com/PoTtaTto/fundamentals_of_software_engineering_lab2_12
Cloning into 'fundamentals_of_software_engineering_lab2_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.
PS C:\Study\СКФУ\Основы программной инженерии\Лабораторная работа 2.12> |
```

Рисунок 3 – Клонирование репозитория

```
PS C:\Study\СКФУ\Основы программной инженерии\Лабораторная работа 2.12> cd fundamentals_of_software_engineering_lab2_12
PS C:\Study\СКФУ\Основы программной инженерии\Лабораторная работа 2.12\fundamentals_of_software_engineering_lab2_12> git checkout -b develop
Switched to a new branch 'develop'
PS C:\Study\СКФУ\Основы программной инженерии\Лабораторная работа 2.12\fundamentals_of_software_engineering_lab2_12>
```

Рисунок 4 – Создание ветки develop, где будут происходить изменения проекта до его полного релиза

```
.gitignore x
1 # Byte-compiled / optimized / DLL files
2 __pycache__/
3 *.py[cod]
4 *$py.class
5
6 # C extensions
7 *.so
8
9 # Distribution / packaging
10 .Python
11 build/
12 develop-eggs/
13 dist/
14 downloads/
15 eggs/
16 .eggs/
17 lib/
18 lib64/
19 parts/
20 sdist/
21 var/
22 wheels/
23 share/python-wheels/
```

Рисунок 5 – Часть .gitignore файла, созданного GitHub

2. Проработаем примеры лабораторной работы, фиксируя изменения.  
Создадим для каждого примера отдельный модуль языка Python:

```
lab_task1.py x
1 #!/usr/bin/env python3
2 # -*- coding: utf-8 -*-
3
4 usage
5 def higher_order(func):
6     print(f'Получена функция {func} в качестве аргумента')
7     return func
8
9 usage
10 @higher_order
11 def hello_world():
12     print('hello world')
13
14 if __name__ == '__main__':
15     hello_world()
```

Run lab\_task1 x

\lab\_task1.py"

Получена функция <function hello\_world at 0x00000164E922E9E0> в качестве аргумента

hello world

Process finished with exit code 0

Рисунок 6 – Код и его выполнение (1)

```
lab_task2.py x
1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3
4  1 usage
5  def decorator_function(func):
6      def wrapper():
7          print('Функция-обёртка!')
8          print('Оборачиваемая функция: {')
9          print('Выполняем обёрнутую функ')
10         func()
11         print('Выходим из обёртки')
12
13     return wrapper
14
15  1 usage
16  @decorator_function
17  def hello_world():
18      print('hello world')
19
20  if __name__ == '__main__':
21      hello_world()
```

Run lab\_task2 x

```
:\env\Scripts\python.exe "C:\Study\СКФУ\Основы программной инженерии\Лабораторная работа 2
.12\fundamentals_of_software_engineering_lab2_12
\lab_task2.py"
Функция-обёртка!
Оборачиваемая функция: <function hello_world at
0x00000222DF748040>
Выполняем обёрнутую функцию...
hello world
Выходим из обёртки

Process finished with exit code 0
```

Рисунок 7 – Код и его выполнение (2)

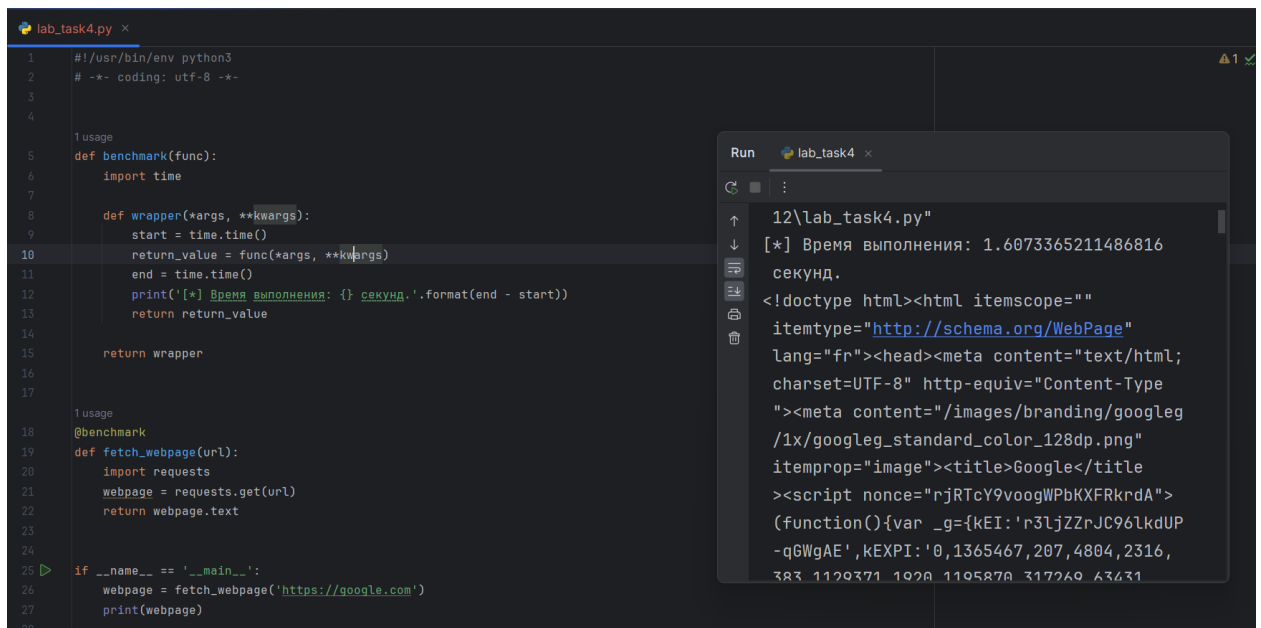
```
lab_task3.py x
1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3
4  1 usage
5  def benchmark(func):
6      import time
7
8      def wrapper():
9          start = time.time()
10         func()
11         end = time.time()
12         print('[*] Время выполнения: {} секунд.'.format(end - start))
13
14     return wrapper
15
16  1 usage
17  @benchmark
18  def fetch_webpage():
19      import requests
20      webpage = requests.get('https://google.com')
21
22  if __name__ == '__main__':
23      fetch_webpage()
24
```

Run lab\_task3 x

```
:\env\Scripts\python.exe "C:\Study\СКФУ\Основы программной инженерии\Лабораторная работа
2.12
\fundamentals_of_software_engineering_
lab2_12\lab_task3.py"
[*] Время выполнения: 2
.1451380252838135 секунд.

Process finished with exit code 0
```

Рисунок 8 – Код и его выполнение (3)

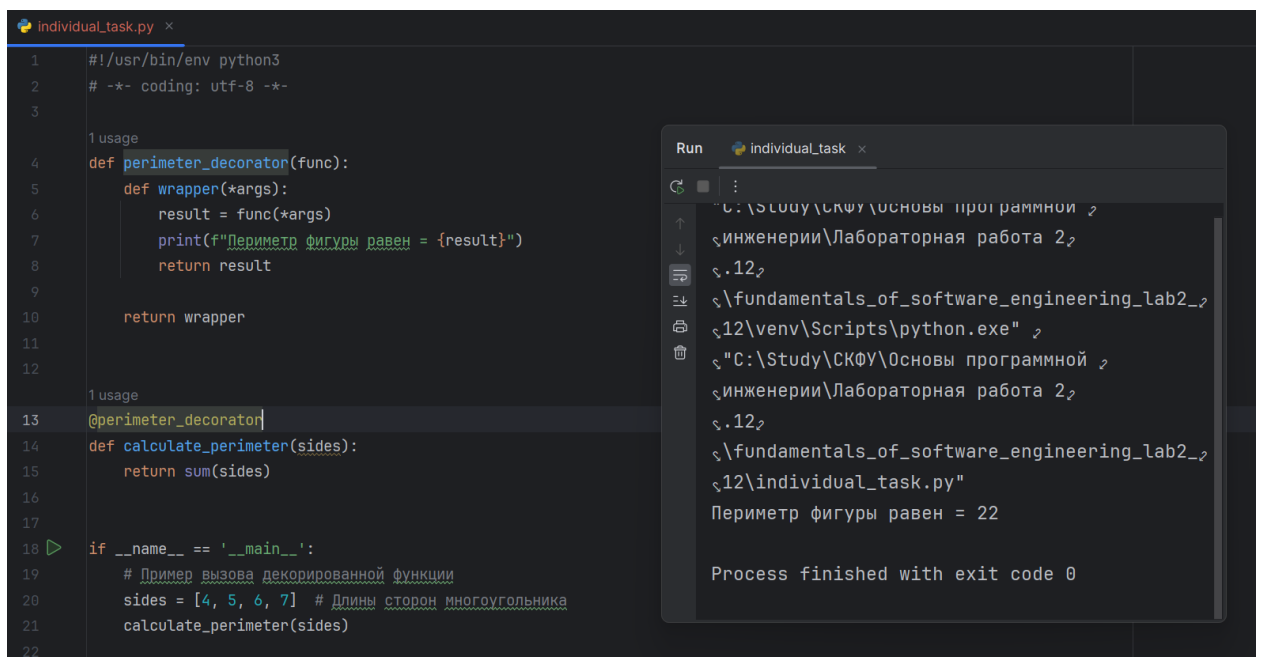


```
1 #!/usr/bin/env python3
2 # -*- coding: utf-8 -*-
3
4
5 1 usage
6 def benchmark(func):
7     import time
8
9     def wrapper(*args, **kwargs):
10         start = time.time()
11         return_value = func(*args, **kwargs)
12         end = time.time()
13         print('[*] Время выполнения: {} секунд.'.format(end - start))
14         return return_value
15
16     return wrapper
17
18 1 usage
19 @benchmark
20 def fetch_webpage(url):
21     import requests
22     webpage = requests.get(url)
23     return webpage.text
24
25 if __name__ == '__main__':
26     webpage = fetch_webpage('https://google.com/')
27     print(webpage)
```

```
Run lab_task4
12\lab_task4.py
[*] Время выполнения: 1.6073365211486816
секунд.
<!doctype html><html itemscope=""
itemtype="http://schema.org/WebPage"
lang="fr"><head><meta content="text/html;
charset=UTF-8" http-equiv="Content-Type
"><meta content="/images/branding/googleg
/1x/googleg_standard_color_128dp.png"
itemprop="image"><title>Google</title
><script nonce="rjRTcY9voogWPbKXFRkdA">
(function(){var _g={kEI:'r3ljZZrJC96lkdUP
-qGWgAE',kEXPI:'0,1365467,207,4804,2316,
383,1120371,1020,1105870,317260,43631
```

Рисунок 9 – Код и его выполнение (4)

3. Выполним индивидуальное задание (вариант №7). Объявите функцию, которая вычисляет периметр многоугольника и возвращает вычисленное значение. Длины сторон многоугольника передаются в виде коллекции (списка или кортежа). Определите декоратор для этой функции, который выводит на экран сообщение: «Периметр фигуры равен = ». Примените декоратор к функции и вызовите декорированную функцию:



```
1 #!/usr/bin/env python3
2 # -*- coding: utf-8 -*-
3
4 1 usage
5 def perimeter_decorator(func):
6     def wrapper(*args):
7         result = func(*args)
8         print(f"Периметр фигуры равен = {result}")
9         return result
10
11     return wrapper
12
13 1 usage
14 @perimeter_decorator
15 def calculate_perimeter(sides):
16     return sum(sides)
17
18 if __name__ == '__main__':
19     # Пример вызова декорированной функции
20     sides = [4, 5, 6, 7] # Длины сторон многоугольника
21     calculate_perimeter(sides)
22
```

```
Run individual_task
C:\Study\СКФУ\Основы программной
инженерии\Лабораторная работа 2
.12
\fundamentals_of_software_engineering_lab2_
12\venv\Scripts\python.exe
"C:\Study\СКФУ\Основы программной
инженерии\Лабораторная работа 2
.12
\fundamentals_of_software_engineering_lab2_
12\individual_task.py"
Периметр фигуры равен = 22
Process finished with exit code 0
```

Рисунок 10 – Код индивидуального задания и его выполнение (1)

```

individual_task.py x
1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3
4  1 usage
5  def perimeter_decorator(func):
6      def wrapper(*args):
7          result = func(*args)
8          print(f"Периметр фигуры равен = {result}")
9          return result
10
11         return wrapper
12
13  1 usage
14  @perimeter_decorator
15  def calculate_perimeter(sides):
16      return sum(sides)
17
18  if __name__ == '__main__':
19      # Пример вызова декорированной функции
20      sides = [100, 90, 80, 70, 60, 120] # Длины сторон многоугольн
21      calculate_perimeter(sides)
  
```

```

Run individual_task x
C:\Study\СКФУ\Основы программной инженерии\Лабораторная работа 2
.12
\fundamentals_of_software_engineering_lab2_
12\venv\Scripts\python.exe"
"C:\Study\СКФУ\Основы программной инженерии\Лабораторная работа 2
.12
\fundamentals_of_software_engineering_lab2_
12\individual_task.py"
Периметр фигуры равен = 520
Process finished with exit code 0
  
```

Рисунок 11 – Код индивидуального задания и его выполнение (2)

4. Выполним merge веток main/develop и отправим изменения на удаленный репозиторий:

```

62ce23c (HEAD -> develop) individual_task.py is added
8e4f57c lab_task4.py is added
d978d76 lab_task3.py is added
60f95df lab_task2.py is added
06e6b22 lab_task1.py is added
cfada08 (origin/main, origin/HEAD, main) Update README.md
7a9e038 Initial commit
PS C:\Study\СКФУ\Основы программной инженерии\Лабораторная работа 2.12\fundamentals_of_software_engineering_lab2_12>
  
```

Рисунок 12 – Коммиты проекта

```

main
Switched to branch 'main'
Your branch is up to date with 'origin/main'.
PS C:\Study\СКФУ\Основы программной инженерии\Лабораторная работа 2.12\fundamentals_of_software_engineering_lab2_12> git merge dev
elop
Updating cfada08..62ce23c
Fast-forward
 .gitignore      | 2 +-
 individual_task.py | 21 ++++++
 lab_task1.py     | 15 ++++++
 lab_task2.py     | 21 ++++++
 lab_task3.py     | 23 ++++++
 lab_task4.py     | 27 ++++++
 6 files changed, 108 insertions(+), 1 deletion(-)
 create mode 100644 individual_task.py
 create mode 100644 lab_task1.py
 create mode 100644 lab_task2.py
 create mode 100644 lab_task3.py
 create mode 100644 lab_task4.py
PS C:\Study\СКФУ\Основы программной инженерии\Лабораторная работа 2.12\fundamentals_of_software_engineering_lab2_12>
  
```

Рисунок 13 – Merge веток main/develop

```
PS C:\Study\СКФУ\Основы программной инженерии\Лабораторная работа 2.12\fundamentals_of_software_engineering_lab2_12> git push origin main
Enumerating objects: 18, done.
Counting objects: 100% (18/18), done.
Delta compression using up to 12 threads
Compressing objects: 100% (16/16), done.
Writing objects: 100% (16/16), 2.44 KiB | 2.44 MiB/s, done.
Total 16 (delta 6), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (6/6), completed with 1 local object.
To https://github.com/PoTtaTto/fundamentals\_of\_software\_engineering\_lab2\_12
   cfada08..62ce23c  main -> main
PS C:\Study\СКФУ\Основы программной инженерии\Лабораторная работа 2.12\fundamentals_of_software_engineering_lab2_12>
```

Рисунок 14 – Отправка изменений на удаленный репозиторий



Ответы на контрольные вопросы:

1. Что такое декоратор?

Декораторы в Python - это специальные функции, которые позволяют модифицировать поведение других функций или методов. Они позволяют добавлять функциональность к существующим функциям, не изменяя их исходный код.

2. Почему функции являются объектами первого класса?

Функции в Python считаются объектами первого класса, потому что их можно присваивать переменным, передавать в функции как аргументы и возвращать из других функций. Они имеют те же свойства, что и другие объекты данных.

3. Каково назначение функций высших порядков?

Функции высших порядков в Python - это функции, которые принимают другие функции в качестве аргументов или возвращают функции в качестве результатов. Они позволяют абстрагироваться от конкретной реализации алгоритма и передавать функции в качестве данных.

4. Как работают декораторы?

Декораторы работают, оборачивая другие функции вокруг себя. Они принимают функцию в качестве аргумента, внутри себя определяют функцию-обертку, которая выполняет дополнительные действия перед или после вызова переданной функции, и возвращают эту новую функцию. При вызове декорированной функции, она будет выполняться через эту функцию-обертку.

5. Какова структура декоратора функций?

Структура декоратора включает в себя определение функции-декоратора, которая принимает функцию в качестве аргумента, внутри себя создает новую функцию (чаще всего используется замыкание), возвращает эту новую функцию и применяет ее к другой функции с использованием символа @.

6. Как можно передать параметры декоратору, а не декорируемой функции?

Декораторы могут принимать параметры, если обернуть их в дополнительную функцию. Это позволяет передавать аргументы в декоратор при его применении к функции. Например:

```
1 usage
def my_decorator_with_params(param):
    def decorator(func):
        def wrapper(*args, **kwargs):
            print(f"Decorator received parameter: {param}")
            return func(*args, **kwargs)
        return wrapper
    return decorator

1 usage
@my_decorator_with_params("my_param_value")
def my_function():
    print("Function executed")

my_function()
```

Рисунок 15 – Пример декоратора с параметрами

В этом примере `my_decorator_with_params` принимает аргумент `param` и возвращает декоратор `decorator`, который в свою очередь оборачивает функцию `my_function`