

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 №15
дисциплины «Основы программной инженерии»

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

(подпись)

Проверил Воронкин Роман Александрович

(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 работы.

1. Создание нового репозитория с лицензией MIT.

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 * SashkaHacker / **Repository name *** laba10
✔ laba10 is available.

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

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.](#)

ⓘ You are creating a public repository in your personal account.

Create repository

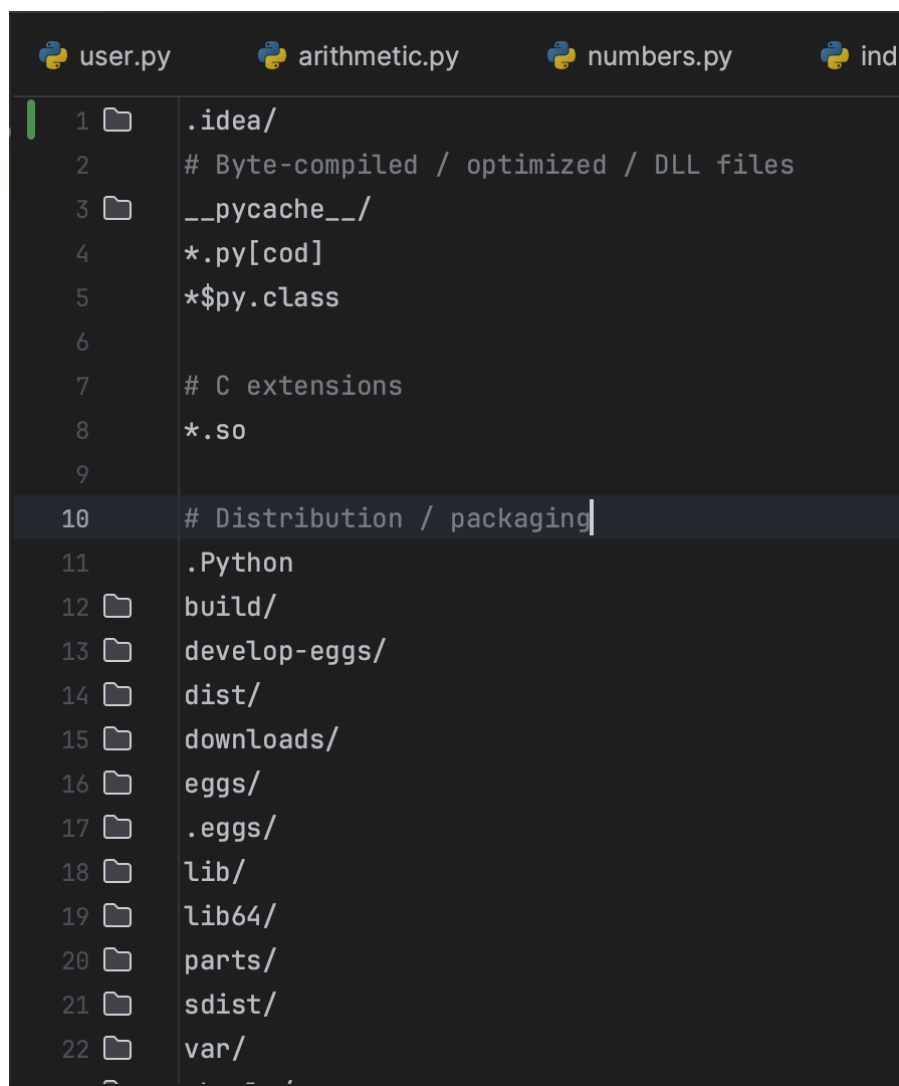
Рисунок 1 – Создание репозитория

2. Клонировал репозиторий на рабочий ПК.

```
Sashka@DESKTOP-U4RPSBI MINGW64 ~/Documents/GitHub
$ git clone https://github.com/SashkaHacker/lab10.git
Cloning into 'lab10'...
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 необходимыми инструкциями.

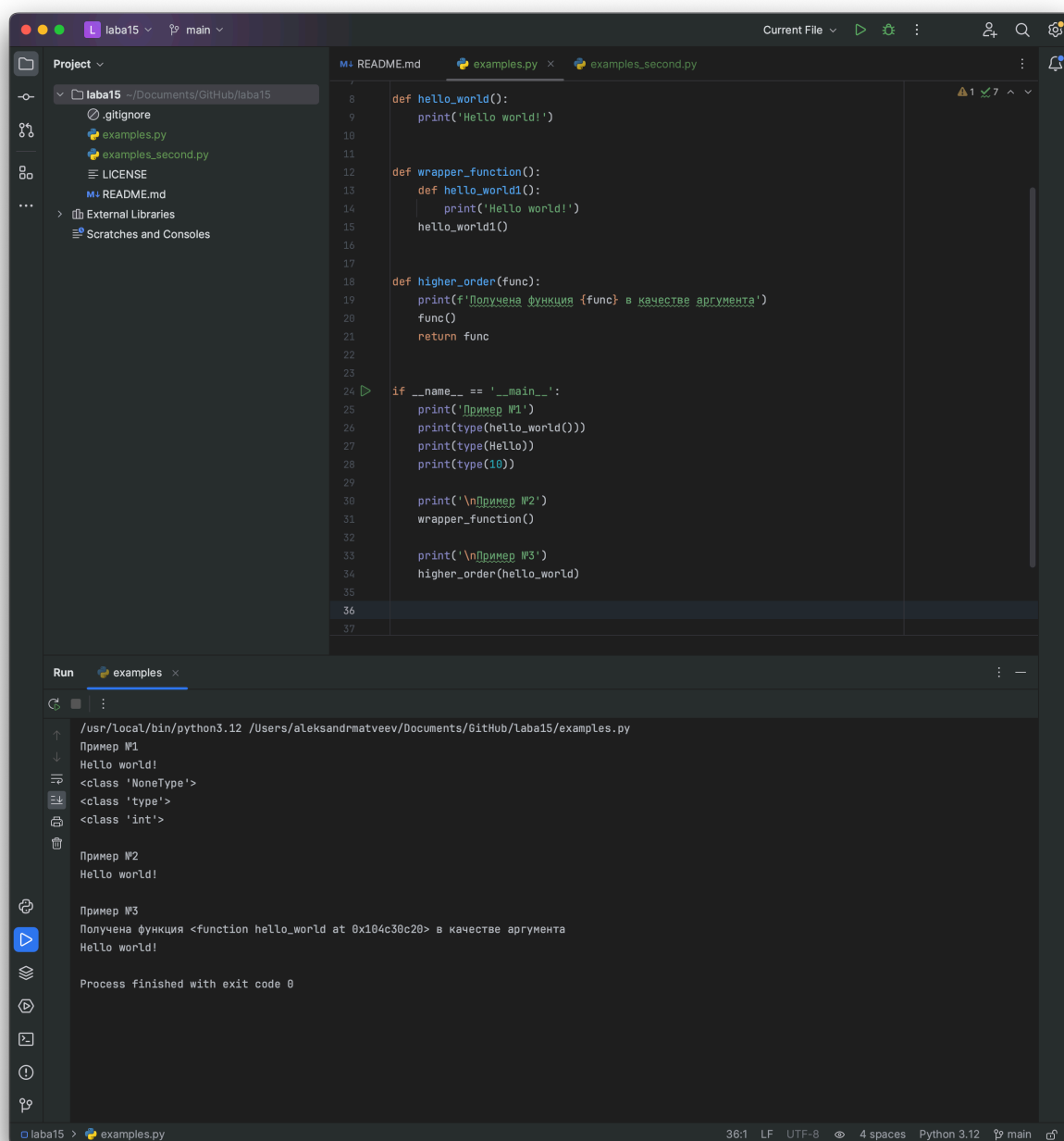


The image shows a code editor window with a dark theme. At the top, there are four tabs: 'user.py', 'arithmetic.py', 'numbers.py', and 'ind'. The active tab is 'user.py'. The editor displays the content of a '.gitignore' file. The text is as follows:

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

Рисунок 3 – Файл .gitignore

4. Проработка примеров из лабораторных задач.



The screenshot shows a code editor with a project named 'laba15'. The file explorer on the left lists files: .gitignore, examples.py, examples_second.py, LICENSE, README.md, External Libraries, and Scratches and Consoles. The main editor displays the content of 'examples.py'.

```
8 def hello_world():
9     print('Hello world!')
10
11
12 def wrapper_function():
13     def hello_world1():
14         print('Hello world!')
15     hello_world1()
16
17
18 def higher_order(func):
19     print(f'Получена функция {func} в качестве аргумента')
20     func()
21     return func
22
23
24 if __name__ == '__main__':
25     print('Пример №1')
26     print(type(hello_world()))
27     print(type>Hello))
28     print(type(10))
29
30     print('\nПример №2')
31     wrapper_function()
32
33     print('\nПример №3')
34     higher_order(hello_world)
35
36
37
```

The 'Run' panel at the bottom shows the execution output for 'examples.py' using Python 3.12. The output is as follows:

```
/usr/local/bin/python3.12 /Users/aleksandrmatveev/Documents/GitHub/Laba15/examples.py
Пример №1
Hello world!
<class 'NoneType'>
<class 'type'>
<class 'int'>

Пример №2
Hello world!

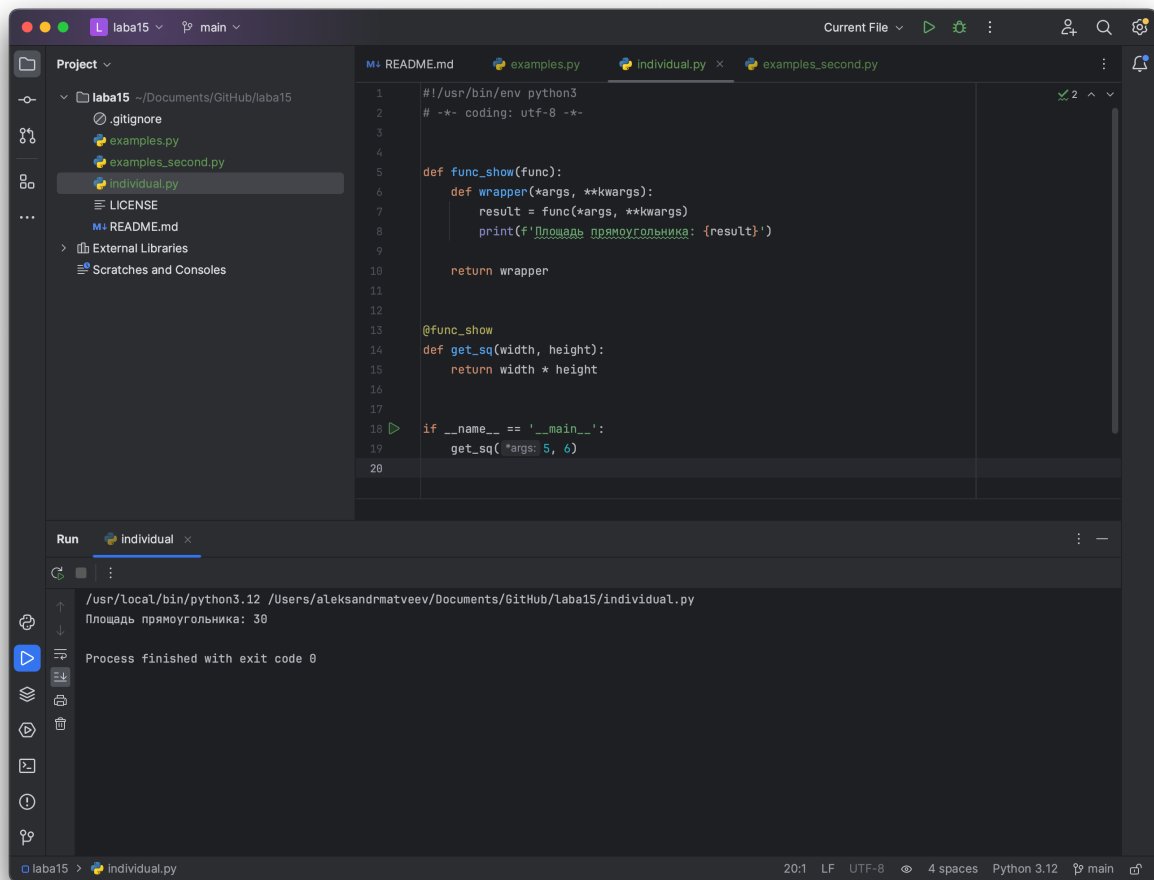
Пример №3
Получена функция <function hello_world at 0x104c30c20> в качестве аргумента
Hello world!

Process finished with exit code 0
```

The status bar at the bottom indicates the file is 'examples.py' in the 'laba15' project, using Python 3.12, with 36 lines of code, LF line endings, UTF-8 encoding, and 4 spaces for indentation.

Рисунок 4 – Примеры часть №1

результат на экране в виде строки (без кавычек): "Площадь прямоугольника: <значение>". Вызовите декорированную функцию `get_sq`.



The screenshot shows a code editor with a project named 'laba15'. The file explorer on the left shows the project structure. The main editor displays a Python file named 'individual.py' with the following code:

```
1 #!/usr/bin/env python3
2 # -*- coding: utf-8 -*-
3
4
5 def func_show(func):
6     def wrapper(*args, **kwargs):
7         result = func(*args, **kwargs)
8         print(f'Площадь прямоугольника: {result}')
9
10    return wrapper
11
12
13 @func_show
14 def get_sq(width, height):
15     return width * height
16
17
18 if __name__ == '__main__':
19     get_sq(*args, 5, 6)
20
```

The bottom panel shows the output of the script:

```
Run individual
/usr/local/bin/python3.12 /Users/aleksandrmatveev/Documents/GitHub/laba15/individual.py
Площадь прямоугольника: 30
Process finished with exit code 0
```

Рисунок 6 – Индивидуальное задание

Контрольные вопросы:

1. Декораторы в Python - это функции, которые позволяют модифицировать поведение других функций, не изменяя их самих. Они представляют собой обертки вокруг функций, добавляющие дополнительный функционал или изменяющие поведение их вызова. Декораторы обертывают функцию, изменяя ее поведение.
2. Функции в Python являются объектами первого класса, потому что они могут быть присвоены переменным, храниться в структурах данных, передаваться в качестве аргументов другим функциям и даже

возвращаться в качестве значений из других функций. Это позволяет писать очень компактные и гибкие программы.

3. Функции высших порядков в Python — это функции, которые принимают другие функции в качестве аргументов или возвращают функции как результат. Это позволяет создавать более модульный и масштабируемый код, поскольку функции могут быть комбинированы и переиспользованы.
4. Декораторы работают путем обертывания функции, изменяя ее поведение.
5. Структура декоратора функций в Python включает в себя определение декоратора, который затем прикрепляется к функции. Они указываются над функцией, а перед ними ставится знак `@`.
6. Чтобы передать параметры декоратору, а не декорируемой функции, можно использовать дополнительный уровень вложенности в декораторе. Это будет функция, которая принимает параметры декоратора и возвращает декоратор.