

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ
ФЕДЕРАЦИИ ФГАОУ ВО «СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ
УНИВЕРСИТЕТ»

Институт цифрового развития

Кафедра инфокоммуникаций

Дисциплина: «языки программирования»

ОТЧЕТ

по лабораторной работе №2

Выполнил: студент 2 курса группы ИТС-21-1

Снадный Михаил Сергеевич

Проверил: к.ф.-м.н., доцент кафедры инфокоммуникаций

Воронкин Роман Александрович

Работа защищена с оценкой: _____

Ставрополь, 2022

Тема:

Декораторы функций в языке Python

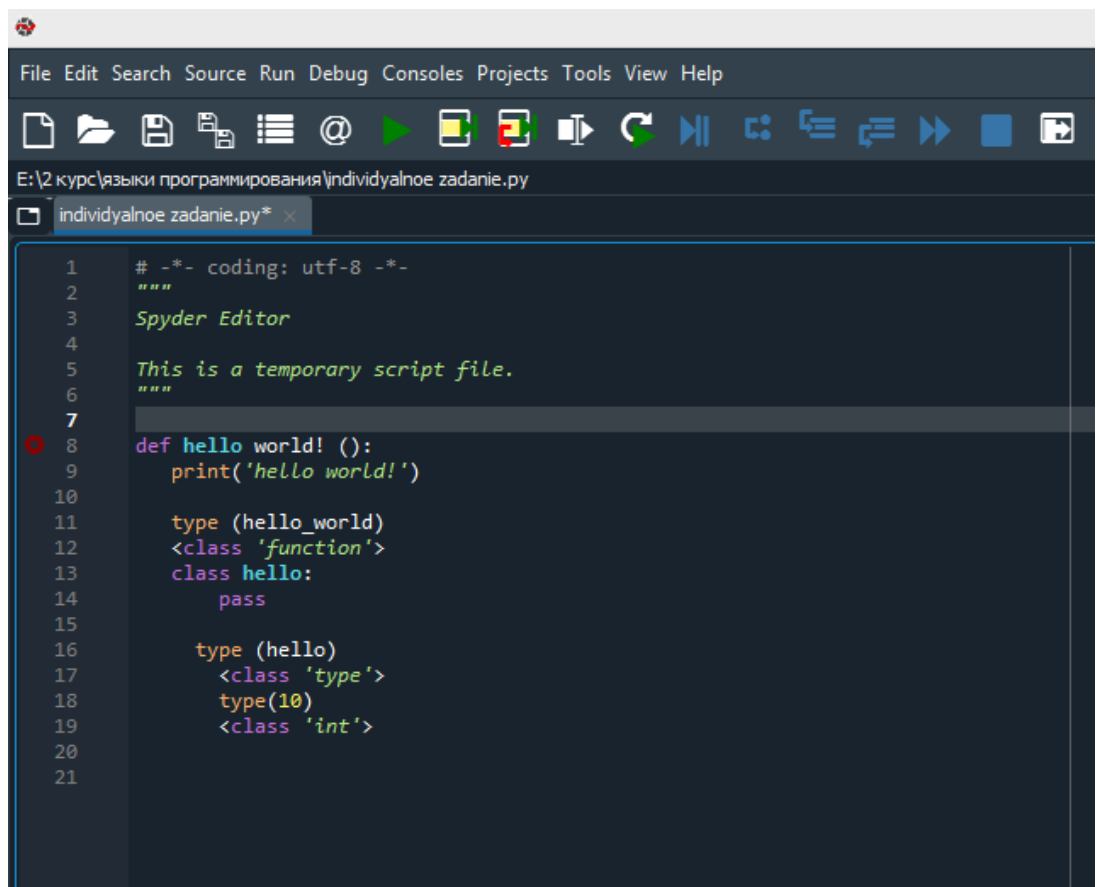
Цель работы:

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

Порядок выполнения работы:

- 1) Создадим общедоступный репозиторий на GitHub (<https://github.com/peach909/12.git>)
- 2) Решим задачи с помощью языка программирования Python3. И отправим их на GitHub.

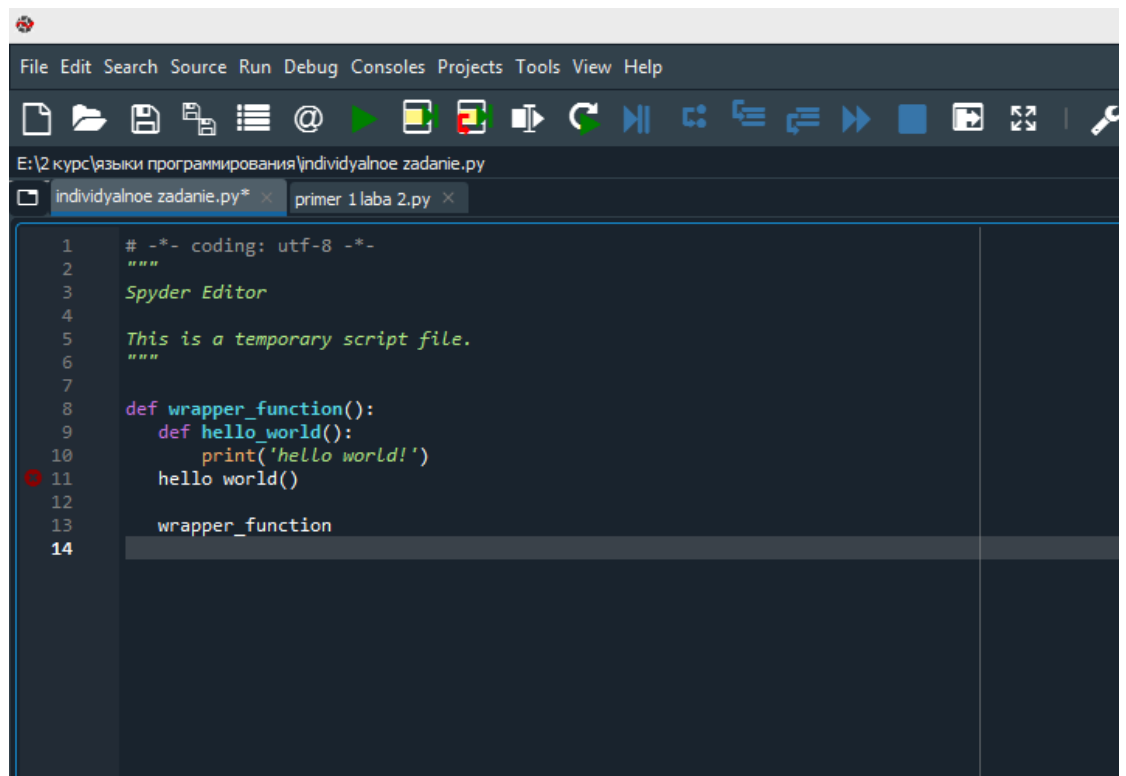
Проработал пример 1



```
1  # -*- coding: utf-8 -*-
2  """
3  Spyder Editor
4
5  This is a temporary script file.
6  """
7
8  def hello world! ():
9      print('hello world!')
10
11     type (hello_world)
12     <class 'function'>
13     class hello:
14         pass
15
16     type (hello)
17     <class 'type'>
18     type(10)
19     <class 'int'>
20
21
```

Рисунок 1. Результат

Проработал пример 2

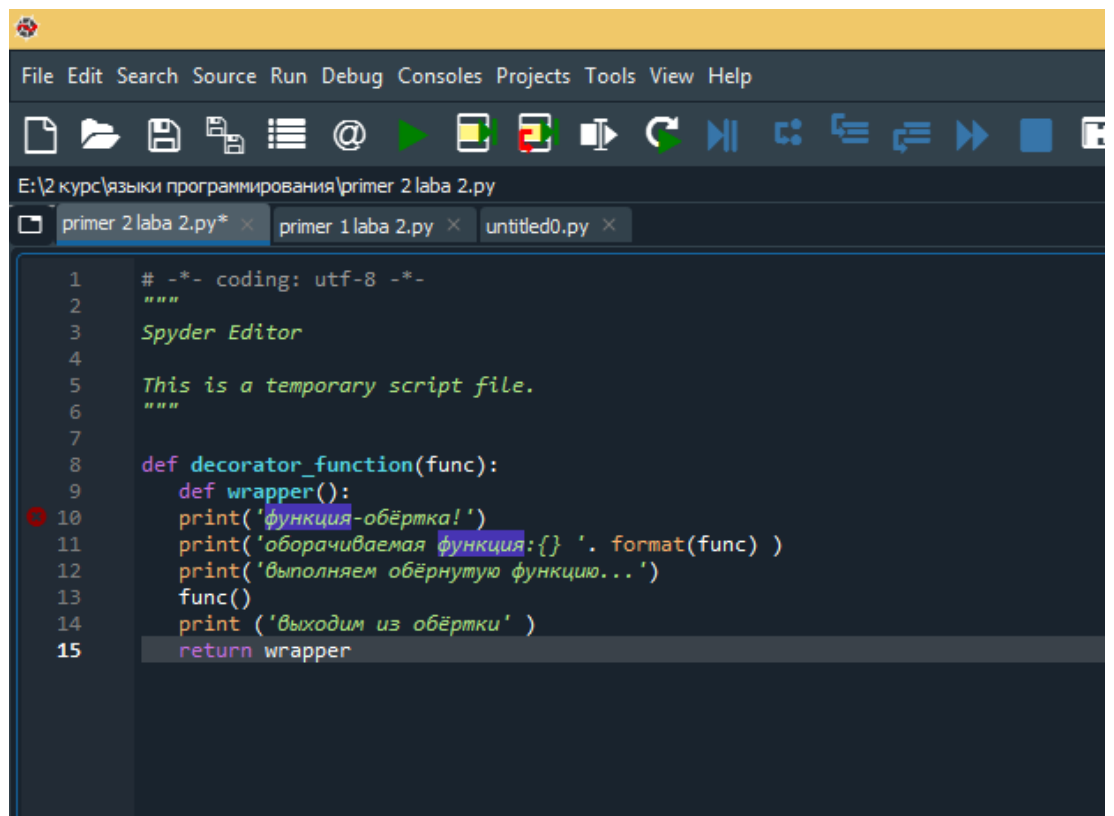


The screenshot shows the Spyder IDE interface. The top menu bar includes File, Edit, Search, Source, Run, Debug, Consoles, Projects, Tools, View, and Help. Below the menu is a toolbar with various icons for file operations and execution. The main editor window displays a Python script with the following content:

```
1  # -*- coding: utf-8 -*-
2  """
3  Spyder Editor
4
5  This is a temporary script file.
6  """
7
8  def wrapper_function():
9      def hello_world():
10         print('hello world!')
11     hello_world()
12
13     wrapper_function
14
```

Рисунок 2. Результат

Проработал пример 3



The screenshot shows the Spyder IDE interface. The top menu bar includes File, Edit, Search, Source, Run, Debug, Consoles, Projects, Tools, View, and Help. Below the menu is a toolbar with various icons for file operations and execution. The main editor window displays a Python script with the following content:

```
1  # -*- coding: utf-8 -*-
2  """
3  Spyder Editor
4
5  This is a temporary script file.
6  """
7
8  def decorator_function(func):
9      def wrapper():
10         print('функция-обёртка!')
11         print('оборачиваемая функция:{}'.format(func) )
12         print('выполняем обёрнутую функцию...')
13         func()
14         print ('выходим из обёртки' )
15     return wrapper
```

Рисунок 3. Результат

Индивидуальное задание

```
1  from functools import wraps
2  >>> def tag(t):
3  ...     def decorator_func(func):
4  ...         @wraps(func)
5  ...         def wrapped(*args, **kwargs):
6  ...             value = func(*args, **kwargs)
7  ...             return f'<{t}>{value}<{t}>'
8  ...             return wrapped
9  ...     return decorator_func
10 ...
11 >>> tag('h1')
12 ... def to_lower(s):
13 ...     return s.lower()
14 ...
15 >>>
16 >>> to_lower('PYTHON')
17 '<h1>python<h1>'
18 >>>
19 >>> tag('div')
20 ... def to_lower(s):
21 ...     return s.lower()
22 ...
23 >>>
24 >>> to_lower('PYTHON')
25 '<div>python<div>'
```

Рисунок 4. Результат

Контрольные вопросы:

1. Что такое декоратор?

Декоратор — это функция, которая позволяет обернуть другую функцию для расширения её

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

В Python всё является объектом, а не только объекты, которые вы создаёте из классов. В этом

смысле он (Python) полностью соответствует идеям объектно-ориентированного

программирования

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?

Просто добавив `@decorator_function` перед определением функции `hello_world()`, мы

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?

```
. def decorator_func(func):  
.     @wraps(func)  
.     def wrapped(*args, **kwargs):  
.         value = func(*args, **kwargs)  
.         return f'<{t}>{value}<{t}>'  
.     return wrapped  
.     return decorator_func
```

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

Вывод:

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