

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

Кафедра инфокоммуникаций

Дисциплина: «Языки программирования»

Отчет по лабораторной работе №10

Декораторы функции в языке Python

Выполнил студент группы ИТС-б-о-21-1

Усиков Данил

«__»_____20__г.

Подпись студента_____

Проверил: Доцент, к.т.н, доцент

кафедры инфокоммуникаций

Воронкин Р.А.

(подпись)

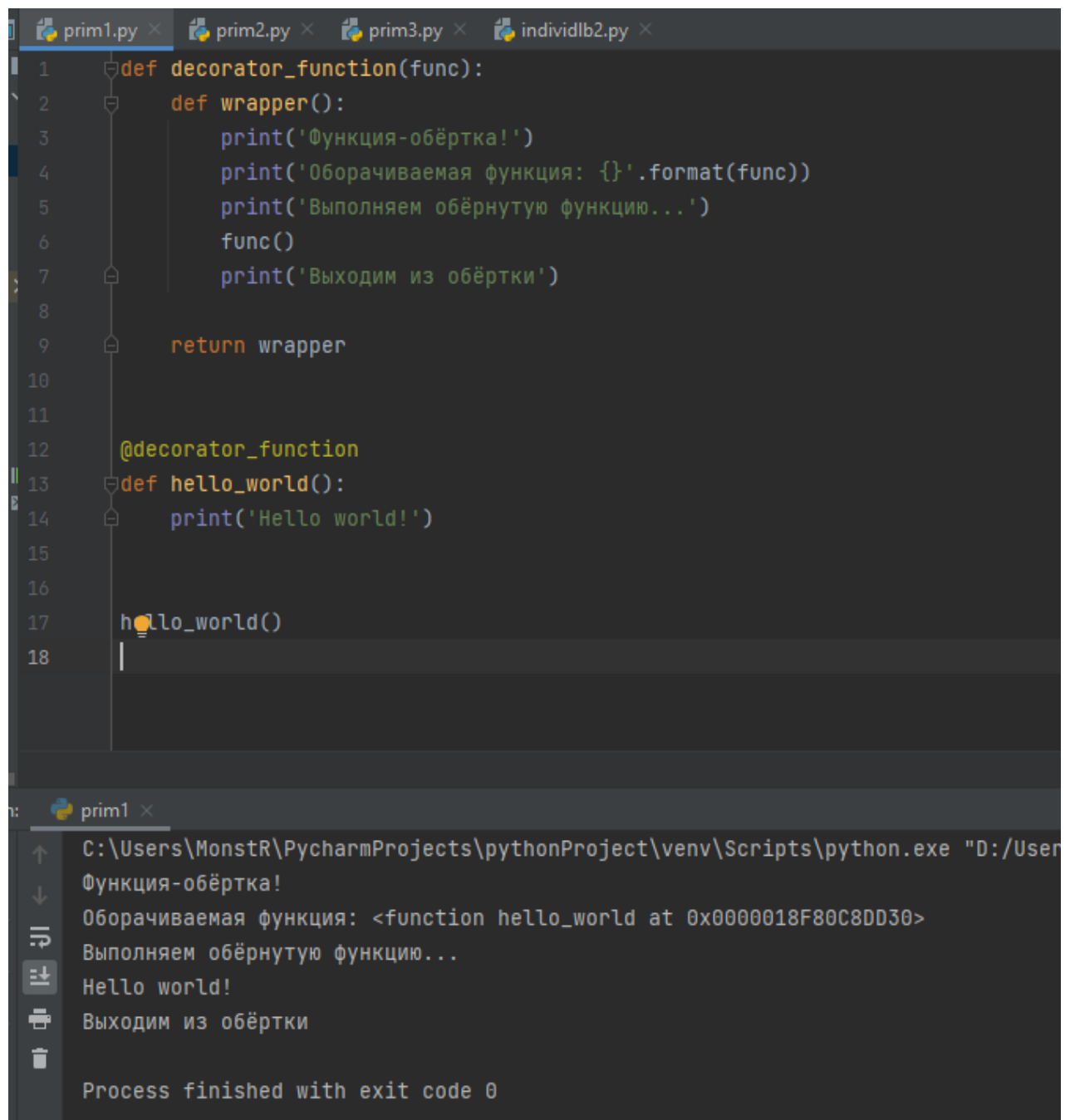
Ставрополь 2022

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

Ссылка на репозиторий - <https://github.com/danilusikov0913/YPlr2#yplr2>

Ход работы:

Пример №1:



```
1 def decorator_function(func):
2     def wrapper():
3         print('Функция-обёртка!')
4         print('Оборачиваемая функция: {}'.format(func))
5         print('Выполняем обёрнутую функцию...')
6         func()
7         print('Выходим из обёртки')
8
9     return wrapper
10
11
12 @decorator_function
13 def hello_world():
14     print('Hello world!')
15
16
17 hello_world()
18
```

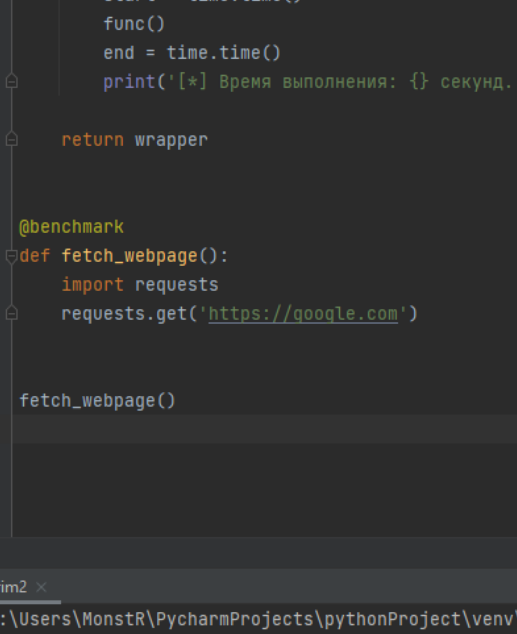
Output:

```
С:\Users\MonstR\PycharmProjects\pythonProject\venv\Scripts\python.exe "D:/User
Функция-обёртка!
Оборачиваемая функция: <function hello_world at 0x0000018F80C8DD30>
Выполняем обёрнутую функцию...
Hello world!
Выходим из обёртки
Process finished with exit code 0
```

Рисунок 1. Код и результат выполнения программой примера 1

Для следующих примеров, чтобы установить библиотеку requests, вводим в строку `pip install requests`.

Пример №2:



The screenshot displays the PyCharm IDE interface. The top toolbar shows icons for running (a green play button) and debugging (a blue bug icon). Below the toolbar, the 'Run' menu is open, showing options like 'Run with Coverage', 'Run with Profiler', and 'Run with Debugger'. The main editor window shows a Python file named 'prim2.py' with the following code:

```

1  def wrapper():
2      start = time.time()
3      func()
4      end = time.time()
5      print('[*] Время выполнения: {} секунд.'.format(end - start))
6
7      return wrapper
8
9  @benchmark
10 def fetch_webpage():
11     import requests
12     requests.get('https://google.com')
13
14     fetch_webpage()

```

The bottom status bar shows the output of the execution:

```

C:\Users\Monstr\P\PycharmProjects\pythonProject\venv\Scripts\python.exe
[*] Время выполнения: 0.9621412754058838 секунд.
Process finished with exit code 0

```

Рисунок 2. Код и результат выполнения программой примера 2

Пример №3:

[illegible]

Рисунок 3. Код и результат выполнения программой примера 3

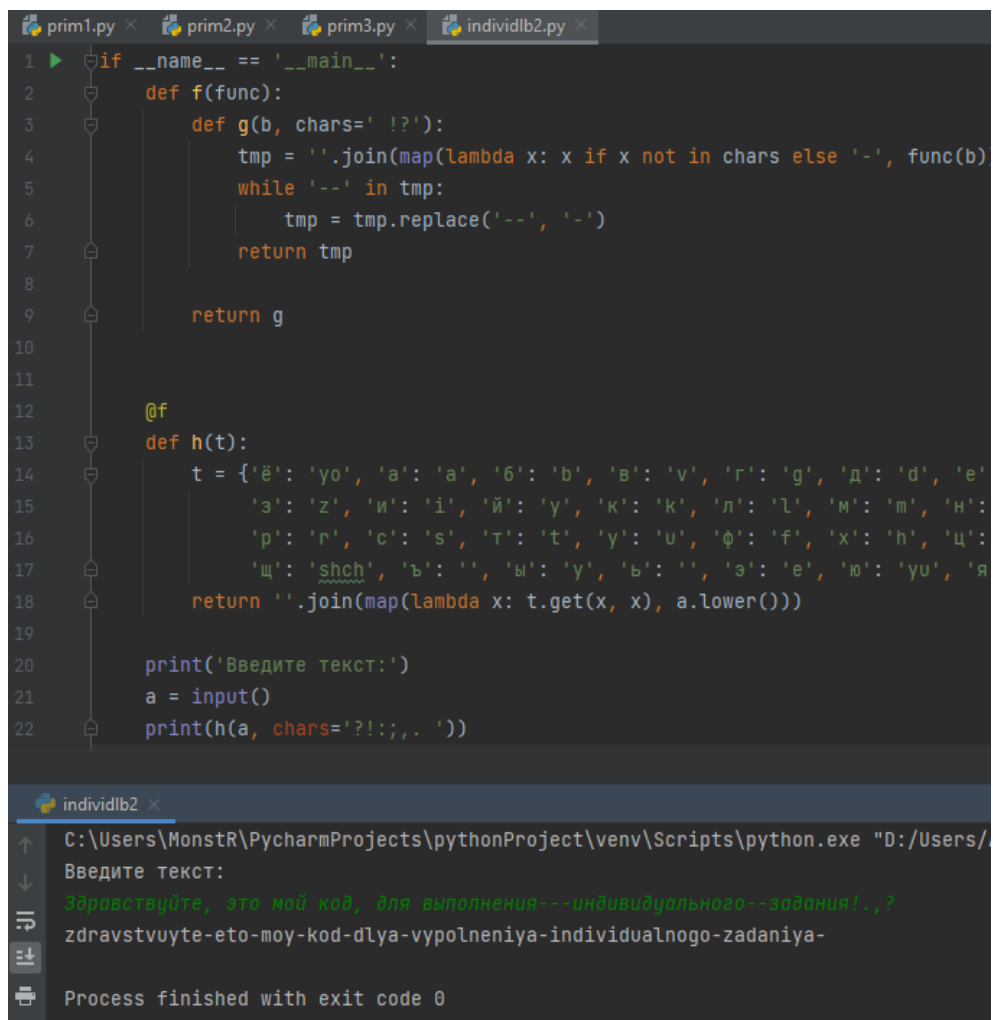
Вариант 19(9)

Индивидуальное задание:

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.



```
1  if __name__ == '__main__':
2      def f(func):
3          def g(b, chars=' !?'):
4              tmp = ''.join(map(lambda x: x if x not in chars else '-', func(b)))
5              while '--' in tmp:
6                  tmp = tmp.replace('--', '-')
7              return tmp
8
9          return g
10
11
12  @f
13  def h(t):
14      t = {'ё': 'yo', 'а': 'a', 'б': 'b', 'в': 'v', 'г': 'g', 'д': 'd', 'е':
15          'э': 'z', 'и': 'i', 'й': 'y', 'к': 'k', 'л': 'l', 'м': 'm', 'н':
16          'п': 'r', 'с': 's', 'т': 't', 'у': 'u', 'ф': 'f', 'х': 'h', 'ц':
17          'щ': 'shch', 'ъ': '', 'ы': 'y', 'ь': '', 'э': 'e', 'ю': 'yu', 'я':
18          return ''.join(map(lambda x: t.get(x, x), a.lower()))
19
20  print('Введите текст:')
21  a = input()
22  print(h(a, chars='?!:;,._'))
```

individl2

C:\Users\MonstR\PycharmProjects\pythonProject\venv\Scripts\python.exe "D:/Users/

Введите текст:

Здравствуйте, это мой код, для выполнения индивидуального задания!.,?

zdravstvuyte-eto-moy-kod-dlya-vypolneniya-individualnogo-zadaniya-

Process finished with exit code 0

Рисунок 4. Код и результат выполнения программой индивидуального задания

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?

Мы можем сохранять функции в переменные, передавать их в качестве аргументов и возвращать из других функций. Можно даже определить одну функцию внутри другой.

3. Каково назначение функций высших порядков?

Это такие функции, которые могут принимать в качестве аргументов и возвращать другие функции.

4. Как работают декораторы?

Из определения: декоратор — это функция, которая позволяет обернуть другую функцию для расширения её функциональности без непосредственного изменения её кода. На примере кода:

```
def decorator_function(func):  
    def wrapper():  
        print('Функция-обёртка!')  
        print('Оборачиваемая функция: {}'.format(func))  
        print('Выполняем обёрнутую функцию...')  
        func()  
        print('Выходим из обёртки')  
    return wrapper
```

5. Какова структура декоратора функций?

Функция декоратор, содержащая в себе декорируемую функцию.

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

Функциональный вызов `func(...)`, который вернет что-то тоже вызываемое или имя функции, или переменная или экземпляр класса

Вывод: приобрела навыки по работе с декораторами функций при написании программ с помощью языка программирования Python версии 3.x.