

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

Институт цифрового развития
Кафедра инфокоммуникаций

«Декораторы функций в языке Python»

ОТЧЕТ
по лабораторной работе №15
дисциплины
«Основы программной инженерии»

Выполнил:

Сотников Андрей Александрович
2 курс, группа ПИЖ-б-о-21-1,
09.03.04 «Программная
инженерия», направленность
(профиль) «Разработка и
сопровождение программного
обеспечения», очная форма
обучения

(подпись)

Проверил:

(подпись)

Отчет защищен с оценкой _____ Дата защиты _____

Ставрополь, 2022 г.

Пример:

```

1 #!/usr/bin/env python3
2 # -*- coding: utf-8 -*-
3
4 new =
5
6 def benchmark(func):
7     import time
8
9     new =
10
11     def wrapper(*args, **kwargs):
12         start = time.time()
13         return_value = func(*args, **kwargs)
14         end = time.time()
15         print(f'[+] Done {name:15s} | {func.__name__} | {end - start}')
16         return return_value
17
18     return wrapper
19
20
21 new =
22
23 @benchmark
24 def fetch_webpage(url):
25     import requests
26     webpage = requests.get(url)
27     return webpage.text
28
29
30 if __name__ == '__main__':
31     webpage = fetch_webpage('https://google.com')
32     print(webpage)
33
34
35 fetch_webpage()
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100

```

Рисунок 1 – Код и результат работы примера

Индивидуальное задание: Объявите функцию, которая принимает строку, удаляет из нее все подряд идущие пробелы и переводит ее в нижний регистр – малые буквы. Результат (строка) возвращается функцией. Определите декоратор, который строку, возвращенную функцией, переводит в азбуку Морзе, используя следующий словарь для замены русских букв и символа пробела на соответствующие последовательности из точек и тире:

```

morze = {'a': '.-', 'б': '-. .', 'в': '---', 'г': '---.', 'д': '---.', 'е':
'.', 'ё': '.', 'ж': '---.', 'з': '---.', 'и': '...', 'й': '---.', 'к': '---.',
'л': '---.', 'м': '---', 'н': '---.', 'о': '---', 'п': '---.', 'р': '---.', 'с':
'...', 'т': '-', 'у': '---', 'ф': '---.', 'х': '---', 'ц': '---.', 'ч': '---.',
'ш': '---', 'щ': '---.', 'ъ': '---.', 'ы': '---.', 'ь': '---.', 'э':
'---.', 'ю': '---.', 'я': '---.', ' ': '---'}

```

Преобразованная строка возвращается декоратором. Примените декоратор к функции и вызовите декорированную функцию. Результат работы отобразите на экране.

```
Ex.py × Individual.py ×
1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3
4
5  AndrejMirroX *
6  def decorator_func(func):
7      morze = {'a': '.-', 'б': '-...-', 'в': '.--', 'г': '--.', 'д': '-..', 'е':
8              '.-', 'ё': '.', 'ж': '...-', 'з': '--..', 'и': '..-', 'й': '---', 'к': '-.-',
9              'л': '-..', 'м': '--', 'н': '-.', 'о': '---', 'п': '-.-', 'с':
10             '...', 'т': '-', 'у': '..-', 'ф': '...-', 'х': '....', 'ц': '-.-.', 'ч': '---',
11             'ш': '- - -', 'щ': '- - .-', 'ъ': '- - .--', 'ы': '- - .--', 'ь': '- - .-', 'э': '..-',
12             'ю': '..--', 'я': '-.-.', ' ': '-----'}
13
14      AndrejMirroX *
15      def wrapper(text):
16          text = func(text)
17          text = ' '.join([morze.get(ch, '') for ch in text])
18
19          return text
20
21      return wrapper
22
23  AndrejMirroX
24  @decorator_func
25  def rus_lat(text):
26      morze = {'a': '.-', 'б': '-...-', 'в': '.--', 'г': '--.', 'д': '-..', 'е':
27              '.-', 'ё': '.', 'ж': '...-', 'з': '--..', 'и': '..-', 'й': '---', 'к': '-.-',
28              'л': '-..', 'м': '--', 'н': '-.', 'о': '---', 'п': '-.-', 'с':
29              '...', 'т': '-', 'у': '..-', 'ф': '...-', 'х': '....', 'ц': '-.-.', 'ч': '---',
30              'ш': '- - -', 'щ': '- - .-', 'ъ': '- - .--', 'ы': '- - .--', 'ь': '- - .-', 'э': '..-',
31              'ю': '..--', 'я': '-.-.', ' ': '-----'}
32
33      text = ' '.join(text.split())
34      text = text.lower()
35
36      return text
37
38  if __name__ == '__main__':
39      txt = "Это тест программы"
40      print(rus_lat(txt))
41
42  rus_lat()
43
44  Individual ×
45  C:\Users\soetni\AppData\Local\Programs\Python\Python39\python.exe C:\labor-15\PyCharm\Individual.py
46  .. ..
47
48  Process finished with exit code 0
```

Рисунок 2 – Код и результат программы 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?

Тот факт, что всё является объектами, открывает перед нами множество возможностей. Мы можем сохранять функции в переменные, передавать их в качестве аргументов и возвращать из других функций. Можно даже определить одну функцию внутри другой. Иными словами, функции — это объекты первого класса. Из определения в Википедии: 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.

3. Каково назначение функций высших порядков?

Функции высших порядков — это такие функции, которые могут принимать в качестве аргументов и возвращать другие функции.

Если вы знакомы с основами высшей математики, то вы уже знаете некоторые математические функции высших порядков порядка вроде дифференциального оператора $\frac{d}{dx}$. Он принимает на входе функцию и возвращает другую функцию, производную от исходной. Функции высших порядков в программировании работают точно так же — они либо принимают функцию(и) на входе и/или возвращают функцию(и).

4. Как работают декораторы?

Декоратор — это функция, которая позволяет обернуть другую функцию для расширения её функциональности без непосредственного изменения её кода.

5. Какова структура декоратора функций?

```
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
```

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

Мы также можем создавать декораторы, которые принимают аргументы. Посмотрим на пример:

```
def benchmark(iters):  
    def actual_decorator(func):  
        import time  
  
        def wrapper(*args, **kwargs):  
            total = 0  
            for i in range(iters):  
                start = time.time()  
                return_value = func(*args, **kwargs)  
                end = time.time()  
                total = total + (end-start)  
            print('[*] Среднее время выполнения: {} секунд.'.format(total/iters))  
            return return_value  
        return wrapper  
    return actual_decorator  
  
@benchmark(iters=10)  
def fetch_webpage(url):  
    import requests  
    webpage = requests.get(url)  
    return webpage.text
```

```
webpage = fetch_webpage('https://google.com')
print(webpage)
```

Здесь мы модифицировали наш старый декоратор таким образом, чтобы он выполнял декорируемую функцию `iters` раз, а затем выводил среднее время выполнения. Однако чтобы добиться этого, пришлось воспользоваться природой функций в Python.

Функция `benchmark()` на первый взгляд может показаться декоратором, но на самом деле таковым не является. Это обычная функция, которая принимает аргумент `iters`, а затем возвращает декоратор. В свою очередь, он декорирует функцию `fetch_webpage()`. Поэтому мы использовали не выражение `@benchmark`, а `@benchmark(iters=10)` — это означает, что тут вызывается функция `benchmark()` (функция со скобками после неё обозначает вызов функции), после чего она возвращает сам декоратор.

Да, это может быть действительно сложно уместить в голове, поэтому держите правило:

Декоратор принимает функцию в качестве аргумента и возвращает функцию.

В нашем примере `benchmark()` не удовлетворяет этому условию, так как она не принимает функцию в качестве аргумента. В то время как функция `actual_decorator()`, которая возвращается `benchmark()`, является декоратором.