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:
Логвинов Иван Васильевич
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:

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-


def benchmark(func):
    import time

    def wrapper(*args, **kwargs):
        start = time.time()
        return_value = func(*args, **kwargs)
        end = time.time()
        print('[*] Время выполнения: {} секунд.'.format(end - start))
        return return_value

    return wrapper


@benchmark
def fetch_webpage(url):
    import requests
    webpage = requests.get(url)
    return webpage.text


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

Рисунок 1 – Код примера

Индивидуальное задание: 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.

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

def decorator_func(func):

    def wrapper(text, chars=' !?'):
        text = ''.join(['-' if i in chars else i for i in text])
        while '--' in text:
            text = text.replace('--', '-')
        return func(text)

    return wrapper


@decorator_func
def rus_lat(text):

    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'}

    text = ''.join([t[i] if i != '-' else '-' for i in text])
    return text


if __name__ == '__main__':
    txt = "Текст     ???? который .... нужно !!::заменить".lower()
    print(rus_lat(txt, chars='?!:;,. '))
```

```
C:\Users\Иван\AppData\Local\Programs\
tekst-kotoryy-nuzhno-zamenit

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 . 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.