

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»

Выполнила:

Кувшин Ирина Анатольевна
2 курс, группа ПИЖ-б-о-21-1,
011.03.04 «Программная инженерия»,
направленность (профиль) «Разработка
и сопровождение программного
обеспечения», очная форма обучения

(подпись)

Проверил:

(подпись)

Отчет защищен с оценкой _____ Дата защиты _____

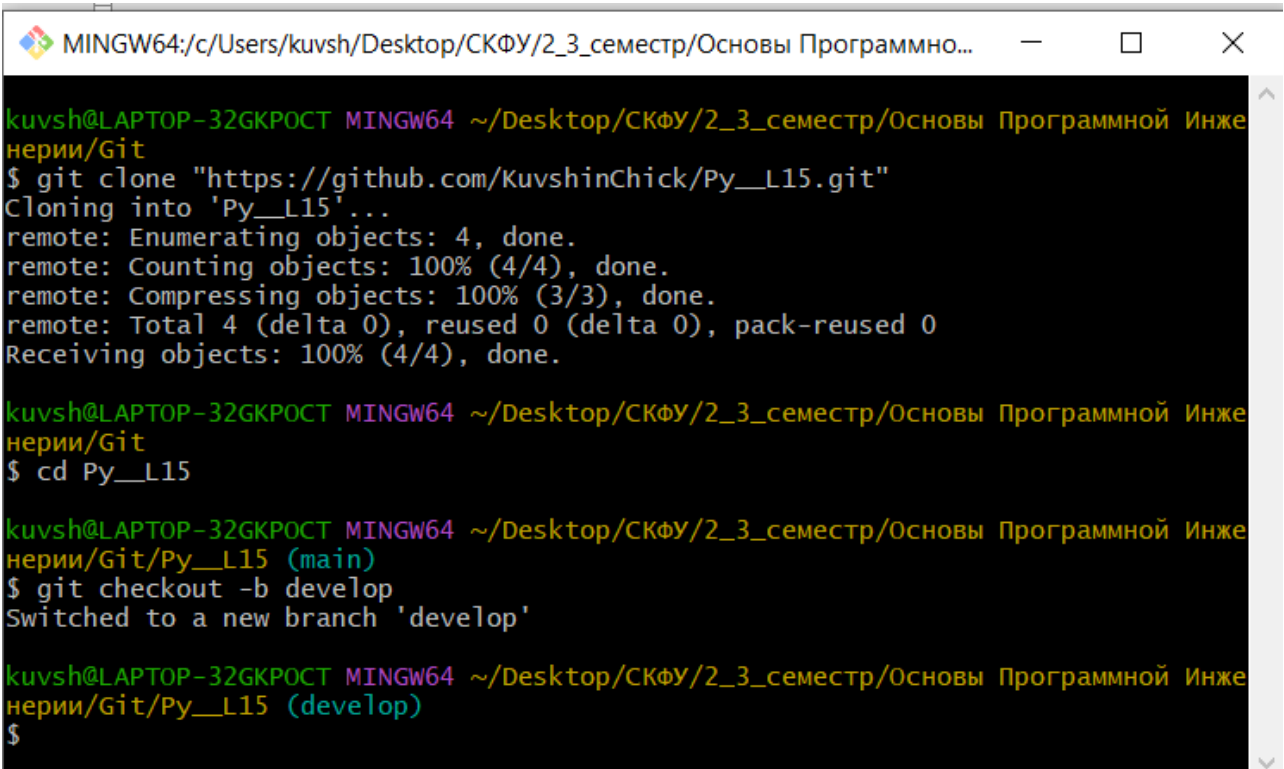
Ставрополь, 2022 г.

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/KuvshinChick/Py__L15.git

Ход работы:

1. Изучить теоретический материал работы.
2. Создать общедоступный репозиторий на GitHub, в котором будет использована лицензия MIT и язык программирования Python.
3. Выполните клонирование созданного репозитория.
4. Дополните файл .gitignore необходимыми правилами для работы с IDE PyCharm.
5. Организуйте свой репозиторий в соответствии с моделью ветвления git-flow.



```
MINGW64:/c/Users/kuvsh/Desktop/СКФУ/2_3_семестр/Основы Программно...
kuvsh@LAPTOP-32GKPOCT MINGW64 ~/Desktop/СКФУ/2_3_семестр/Основы Программно...
$ git clone "https://github.com/KuvshinChick/Py__L15.git"
Cloning into 'Py__L15'...
remote: Enumerating objects: 4, done.
remote: Counting objects: 100% (4/4), done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 4 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (4/4), done.

kuvsh@LAPTOP-32GKPOCT MINGW64 ~/Desktop/СКФУ/2_3_семестр/Основы Программно...
$ cd Py__L15

kuvsh@LAPTOP-32GKPOCT MINGW64 ~/Desktop/СКФУ/2_3_семестр/Основы Программно...
$ git checkout -b develop
Switched to a new branch 'develop'

kuvsh@LAPTOP-32GKPOCT MINGW64 ~/Desktop/СКФУ/2_3_семестр/Основы Программно...
$
```

Рисунок 15.1 – Клонирование репозитория и создание ветки develop

```
MINGW64:/c:/Users/kuvsh/Desktop/СКФУ/2_3_семестр/Основы Программной Инженерии/Git/Py_L15
kuvsh@LAPTOP-32GKP0CT MINGW64 ~/Desktop/СКФУ/2_3_семестр/Основы Программной Инженерии/Git/Py_L15 (develop)
$ git status
On branch develop
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   README.md

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        .gitignore

no changes added to commit (use "git add" and/or "git commit -a")

kuvsh@LAPTOP-32GKP0CT MINGW64 ~/Desktop/СКФУ/2_3_семестр/Основы Программной Инженерии/Git/Py_L15 (develop)
$ git add .

kuvsh@LAPTOP-32GKP0CT MINGW64 ~/Desktop/СКФУ/2_3_семестр/Основы Программной Инженерии/Git/Py_L15 (develop)
$ git commit -m "modified .gitignore & readme"
[develop 5ad5c26] modified .gitignore & readme
2 files changed, 132 insertions(+), 1 deletion(-)
create mode 100644 .gitignore
```

Рисунок 15.2 – Обновление .gitignore и readme

6. Создайте проект PyCharm в папке репозитория.
7. Проработать примеры лабораторной работы.

8. Выполнить индивидуальное задание.

5. Вводится строка целых чисел через пробел. Напишите функцию, которая преобразовывает эту строку в список чисел и возвращает их сумму. Определите декоратор для этой функции, который имеет один параметр `start` – начальное значение суммы. Примените декоратор со значением `start=5` к функции и вызовите декорированную функцию. Результат отобразите на экране.

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

# sum_start - функция, принимающая начальную сумму(start)
# возвращает декоратор
def sum_start(start):
    # декоратор
    def decorator(func):
        # Функция - обёртка
        def wrapped(*args, **kwargs):
            # Получаем сумму чисел в листе из ф-и num_sum()
            n_s = func(*args, **kwargs)
            # Возвращаем сумму с начальным счетчиком
            return n_s + start

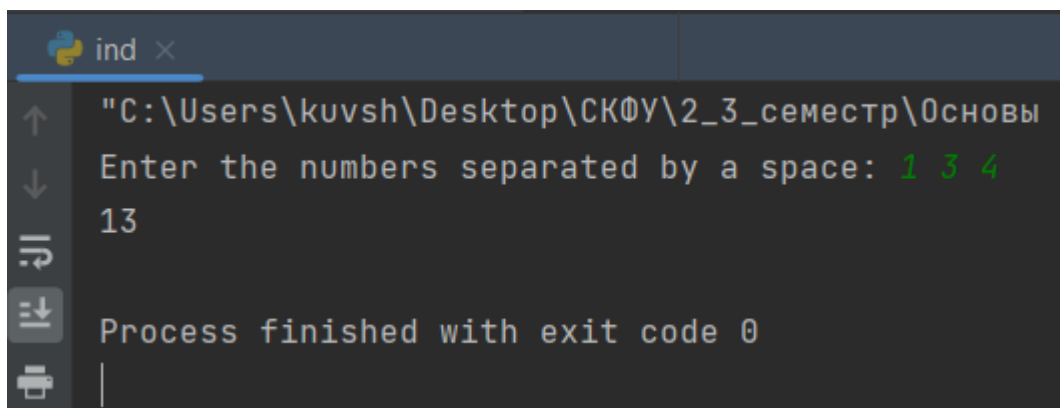
        return wrapped

    return decorator

# Вызывается функция sum_start(), возвращающая декоратор
@sum_start(5)
def num_sum(s):
    num_list = [int(x) for x in s.split()]
    return sum(num_list)

if __name__ == '__main__':
    st = input("Enter the numbers separated by a space: ")
    print(num_sum(st))
```

Рисунок 15.5 – Код программы



```
ind x
"C:\Users\kuvsh\Desktop\СКФУ\2_3_семестр\Основы
Enter the numbers separated by a space: 1 3 4
13
Process finished with exit code 0
```

Рисунок 15.6 – Результат работы программы

9. Зафиксировать изменения в репозитории.

10. Добавьте отчет по лабораторной работе в формате PDF в папку doc репозитория. Зафиксируйте изменения.

11. Выполните слияние ветки для разработки с веткой master/main.
12. Отправьте сделанные изменения на сервер GitHub.
13. Отправьте адрес репозитория GitHub на электронный адрес преподавателя.

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.