

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ
ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение

высшего образования

«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт цифрового развития

Кафедра информационных систем и технологий

Отчет по лабораторной работе №15.

Дисциплина: «Основы программной инженерии»

Выполнил:

Студент группы ПИЖ-б-о-22-1,

направление подготовки: 09.03.04

«Программная инженерия»

ФИО: Джараян Арег Александрович

Проверил:

Воронкин Р. А.

Ставрополь 2023

Тема: Лабораторная работа 2.12 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.

Выполнение работы:

1. Изучил теоретический материал раб


Изучила теоретический материал работы.

2. Создала репозиторий на git.hub.

Репозиторий содержит все файлы проекта, включая историю изменений. У вас уже есть репозиторий проекта в другом месте? [Импортируйте репозиторий.](#)

Обязательные поля отмечены звездочкой (*).



Владелец * Название репозитория *

 мрядская / lab15

✓ lab15.

Отличные названия репозитория короткие и запоминающиеся. Нужно вдохновение? Как насчет [solid-octo-invention?](#)

Описание (необязательно)

- ☒  **Общедоступный**
Этот репозиторий может видеть любой пользователь Интернета. Вы сами выбираете, кто может совершать коммиты.
- ☐  **Приватный**
Вы сами выбираете, кто может просматривать этот репозиторий и фиксировать его в нем.

Инициализируйте этот репозиторий с помощью:

- ☒ **Добавьте файл README**
Здесь вы можете написать подробное описание вашего проекта. [Узнайте больше о README.](#)

[Добавить файл .gitignore](#)

Рисунок 1 – создание репозитория

```
C:\git1\lab14>cd ..  
C:\git1>git clone https://github.com/mryadskaya/lab15.git  
Cloning into 'lab15'...  
remote: Enumerating objects: 5, done.  
remote: Counting objects: 100% (5/5), done.  
remote: Compressing objects: 100% (4/4), done.  
remote: Total 5 (delta 0), reused 0 (delta 0), pack-reused 0  
Receiving objects: 100% (5/5), done.  
C:\git1>cd lab15  
C:\git1\lab15>
```

3. Клонировала репозиторий.

4. Организовать свой репозиторий в соответствии с моделью ветвления git-flow.

```
C:\git1\lab15>git branch develop  
C:\git1\lab15>git push -u origin develop  
Total 0 (delta 0), reused 0 (delta 0), pack-reused 0  
remote:  
remote: Create a pull request for 'develop' on GitHub by visiting:  
remote:      https://github.com/mryadskaya/lab15/pull/new/develop  
remote:  
To https://github.com/mryadskaya/lab15.git  
 * [new branch]      develop -> develop  
branch 'develop' set up to track 'origin/develop'.  
C:\git1\lab15>git checkout develop  
Switched to branch 'develop'  
Your branch is up to date with 'origin/develop'.  
C:\git1\lab15>
```

Рисунок 4 – создание ветки develop

5. Проработал примеры из методички.

```

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

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

```

Рисунок 5 – пример 1

15. 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

—

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

def sum_decorator(start):
    def decorator(func):
        def wrapper(input_string):
            numbers = [int(num) for num in input_string.split()]
            result = func(numbers)
            return result + start
        return wrapper
    return decorator

@sum_decorator(start=5)
def sum_of_numbers(numbers):
    return sum(numbers)

if __name__ == "__main__":
    # Ввод строки целых чисел через пробел
    input_string = input("Введите строку целых чисел через пробел: ")

    # Вызов декорированной функции и вывод результата
    result = sum_of_numbers(input_string)
    print(f"Сумма чисел с учетом начального значения: {result}")
```

начальное значение суммы. Примените декоратор со значением `start=5` к функции и вызовите декорированную функцию. Результат отобразите на экране.

Рисунок 6 – индивидуальное задание

```
C:\Users\ADMIN\PycharmProjects\pythonProject\venv\Scripts\python.exe C:\git1\lab15\Pycharm\индивидуальное.py
Введите строку целых чисел через пробел: 2 1 3 0 6 2 0 1 3
Сумма чисел с учетом начального значения: 23

Process finished with exit code 0
|
```

Рисунок 7 – индивидуальное задание

7. Зафиксировал все изменения в github в ветке develop и слил ветки

```

C:\git1\lab15>git add .

C:\git1\lab15>git status
On branch develop
Your branch is up to date with 'origin/develop'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    new file:   "PyCharm/\320\270\320\275\320\264\320\270\320\262\320\270\320\264\321\203\320\260\320\273\321\214\
0\275\320\276\320\265.py"
    new file:   "PyCharm/\320\277\321\200\320\270\320\274\320\265\321\200.py"

C:\git1\lab15>git commit -m "схранение"
[develop 037440e] схранение
 2 files changed, 42 insertions(+)
 create mode 100644 "PyCharm/\320\270\320\275\320\264\320\270\320\262\320\270\320\264\321\203\320\260\320\273\321\214\
0\275\320\276\320\265.py"
 create mode 100644 "PyCharm/\320\277\321\200\320\270\320\274\320\265\321\200.py"

C:\git1\lab15>git push
Enumerating objects: 6, done.
Counting objects: 100% (6/6), done.
Delta compression using up to 2 threads
Compressing objects: 100% (5/5), done.
Writing objects: 100% (5/5), 1.14 KiB | 166.00 KiB/s, done.
Total 5 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/mryadskaya/lab15.git
 9622067..037440e  develop -> develop

C:\git1\lab15>git checkout main
Switched to branch 'main'
Your branch is up to date with 'origin/main'.

C:\git1\lab15>

```

Рисунок 8 – фиксация изменений в ветку develop

Контрольные вопросы:

1. Что такое декоратор?

Декораторы — один из самых полезных инструментов в Python, однако

новичкам они могут показаться непонятными. Декоратор — это функция, которая

позволяет обернуть другую функцию для расширения её функциональности без

непосредственного изменения её кода

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. Именно поэтому функции являются объектами

первого класса.

3. Каково назначение функций высших порядков?

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?

Декораторы в Python представляют собой способ изменить поведение функции или метода, обернув его в другую функцию. Это мощный механизм, который позволяет добавлять или изменять функциональность функций без изменения их кода. Декораторы часто используются для внесения дополнительной логики, проверок или изменений в функции. Декораторы позволяют модифицировать поведение функций или методов, делая код более модульным и легким для понимания. Они часто используются, например, для логирования, обработки ошибок, кеширования, аутентификации и других аспектов функциональности программы.

5. Какова структура декоратора функций?

```
def decorator_function(original_function):  
  
    def wrapper_function(*args, **kwargs):  
  
        # Дополнительный код, выполняемый перед вызовом оригинальной  
        функции  
  
        result = original_function(*args, **kwargs)  
  
        # Дополнительный код, выполняемый после вызова оригинальной  
        функции  
  
        return result
```

```
return wrapper_function
```

6. Самостоятельно изучить как можно передать параметры

декоратору, а не декорируемой функции?

Использование функций-фабрик декораторов:

```
def decorator_factory(param):
    def decorator_function(original_function):
        def wrapper_function(*args, **kwargs):
            print(f"Дополнительный код с параметром {param} перед вызовом функции")
            result = original_function(*args, **kwargs)
            print(f"Дополнительный код с параметром {param} после вызова функции")
            return result
        return wrapper_function
    return decorator_function

# Использование декоратора с параметром
@decorator_factory(param="some_parameter")
def example_function():
    print("Оригинальная функция")

# Вызов функции, обернутой в декоратор
example_function()
```

2. Использование частичного применения (functools.partial):


```
from functools import partial

def decorator_function(param, original_function, *args, **kwargs):
    print(f"Дополнительный код с параметром {param} перед вызовом функции")
    result = original_function(*args, **kwargs)
    print(f"Дополнительный код с параметром {param} после вызова функции")
    return result

# Создание частичной функции с фиксированным параметром
decorator_with_param = partial(decorator_function, param="some_parameter")

# Использование декоратора с параметром
@decorator_with_param
def example_function():
    print("Оригинальная функция")

# Вызов функции, обернутой в декоратор
example_function()
```