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

Институт цифрового развития
Кафедра инфокоммуникаций

ОТЧЕТ
ПО ЛАБОРАТОРНОЙ РАБОТЕ №2.12
дисциплины «Основы программной инженерии»

Выполнил:
Баратов Семен Григорьевич
2 курс, группа ПИЖ-б-о-22-1,
09.03.04 «Программная инженерия»,
направленность (профиль) «Разработка и сопровождение программного обеспечения», очная форма обучения

_____
(подпись)

Преподаватель:
Воронкин Р.А., канд. тех. наук, доцент,
доцент кафедры инфокоммуникаций

_____
(подпись)

Отчет защищен с оценкой _____ Дата защиты_____

Ставрополь, 2023 г.

Тема: Декораторы функций в языке Python.

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

Результаты выполнения

1. Создали репозиторий с лицензией MIT, добавили в .gitignore необходимые правила для работы с IDE PyCharm, клонировали репозиторий, организовали репозиторий в соответствии с моделью git-flow.

```
Last login: Tue Oct 24 20:29:02 on ttys000
itssyoma@MacBook-Air-Sema Основы программной инженерии % git clone https://github.com/itssyoma/megarepo_21.git
Cloning into 'megarepo_21'...
remote: Enumerating objects: 4, done.
remote: Counting objects: 100% (4/4), done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 4 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (4/4), done.
itssyoma@MacBook-Air-Sema Основы программной инженерии % cd megarepo_21
itssyoma@MacBook-Air-Sema megarepo_21 % git checkout -b develop
Switched to a new branch 'develop'
itssyoma@MacBook-Air-Sema megarepo_21 % git branch release
itssyoma@MacBook-Air-Sema megarepo_21 % git branch develop
fatal: a branch named 'develop' already exists
itssyoma@MacBook-Air-Sema megarepo_21 % git branch hotfix
itssyoma@MacBook-Air-Sema megarepo_21 % git branch feature
itssyoma@MacBook-Air-Sema megarepo_21 %
```

Рисунок 1 – Работа с репозиторием в командной строке.

2. Проработали пример.

example.py > ...

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


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


@benchmark
def fetch_webpage():
    import requests
    webpage = requests.get('https://google.com')


if __name__ == "__main__":
    fetch_webpage()

```

ПРОБЛЕМЫ ВЫХОДНЫЕ ДАННЫЕ КОНСОЛЬ ОТЛАДКИ ТЕРМИНАЛ ПОРТЫ

```
/usr/local/bin/python3 "/Users/itssyoma/Yandex.Disk.localized/Учеба/Основы пр
itssyoma@MacBook-Air-Sema megarepo_212 % /usr/local/bin/python3 "/Users/itssy
repo_212/example.py"
[*] Время выполнения: 0.47729921340942383 секунд.
```

Рисунок 2 – Код и результат работы программы.

3. Выполнили индивидуальное задание (вариант 1). Объявите функцию с именем get_sq, которая вычисляет площадь прямоугольника по двум

параметрам: width и height – ширина и высота прямоугольника и возвращает результат. Определите декоратор для этой функции с именем (внешней функции) func_show, который отображает результат на экране в виде строки (без кавычек): "Площадь прямоугольника: <значение>". Вызовите декорированную функцию get_sq.

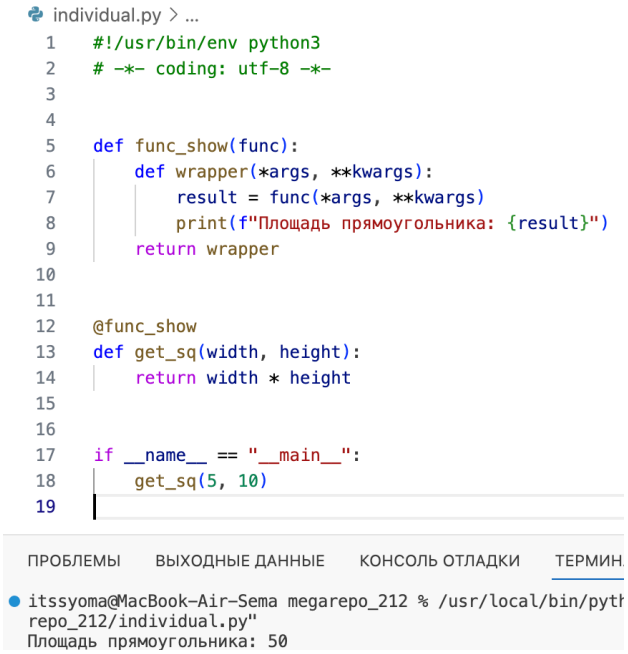

Рисунок 3 – Код и результат работы программы.

Ответы на контрольные вопросы

1. Что такое декоратор?

Декоратор - это функция, которая принимает другую функцию в качестве аргумента, добавляет к ней какое-то дополнительное поведение и возвращает измененную функцию или заменяет её на другую.

2. Почему функции являются объектами первого класса?

Функции в Python являются объектами первого класса, потому что они могут быть переданы как аргументы другим функциям, возвращены как значения из других функций, сохранены в переменных и структурах данных.

3. Каково назначение функций высших порядков?

Функции высших порядков - это функции, которые либо принимают другие функции в качестве аргументов, либо возвращают функции в качестве результатов.

4. Как работают декораторы?

Декораторы работают путем обертывания (wrapping) одной функции вокруг другой. Когда декорированная функция вызывается, она фактически вызывает обернутую функцию с дополнительным поведением.

5. Какова структура декоратора функций?

Структура декоратора функций в Python обычно выглядит следующим образом:

```
def decorator(func):
    def wrapper(*args, **kwargs):
        # Дополнительное поведение перед вызовом функции
        result = func(*args, **kwargs)
        # Дополнительное поведение после вызова функции
        return result
    return wrapper

@decorator
def some_function():
    # Тело функции
    pass
```

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

Для передачи параметров декоратору, а не декорируемой функции, можно определить декоратор как функцию с параметрами и использовать еще один уровень обертки:

```
def parametrized_decorator(param):
    def decorator(func):
        def wrapper(*args, **kwargs):
            # Дополнительное поведение перед вызовом функции, используя
параметр param
            result = func(*args, **kwargs)
            # Дополнительное поведение после вызова функции
            return result
        return wrapper
    return decorator

@parametrized_decorator("some_param")
def some_function():
    # Тело функции
    pass
```