МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ

ФЕДЕРАЦИИ

Федеральное государственное автономное

образовательное учреждение высшего образования

«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Кафедра инфокоммуникаций

«Декораторы функций в языке Python»

Отчет по лабораторной работе № 2.12

по дисциплине «Основы программной инженерии»

Выполнил студент группы ПИЖ-б-о-21-1

Гребенкин Е. А. « » 2022г.

Подпись студента_____

Работа защищена « »_____20__г.

Проверил Воронкин Р.А. _____

(подпись)

Ставрополь 2022

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

Выполнение 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 соответствие с моделью ветвления git-flow.
6. Создайте проект PyCharm в папке репозитория.
7. Проработайте примеры лабораторной работы.

Пример 1.

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



@decorator_function def
hello_world():
print('Hello world')
  if __name__ ==
'__main__':
hello_world()
```

```
Функция-обёртка!
Оборачиваемая функция: <function hello_world at 0x1028587c0>
Выполняем обёрнутую функцию...
Hello world!
Выходим из обёртки
```

Рисунок 6 – Результат работы программы

```
#!/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()
```

```
[*] Время выполнения: 1.0039680004119873 секунд.

Process finished with exit code 0
```

Рисунок 7 – Результат работы программы

8. Выполните индивидуальные задания.

Объявите функцию, которая вычисляет площадь круга и возвращает вычисленное значение.

В качестве аргумента ей передается значение радиуса. Определите декоратор для этой функции, который выводит на экран сообщение:

«Площадь круга равна = <число>». В строке выведите числовое значение с точностью до сотых.

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
 def decorator(func):
def wrapper(*args):
        print(f'Площадь круга равна = {func(*args):1.2f}')
return wrapper



@decorator def
s_circle(r):
import math


    return math.pi * r**2
  if __name__ ==
'__main__':
    s_circle(int(input('enter r: ')))
```

```
Введите строку: Работай программка
<div>работай программка</div>
```

Рисунок 8 – Результат работы программы

Вопросы для защиты работы

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?

Объектами первого класса в контексте конкретного языка программирования называется 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.

4. Как работают декораторы?

Декоратор – это функция, которая позволяет обернуть другую функцию для расширения её функциональности без непосредственного изменения её кода. Внутри декораторы мы определяем другую функцию, обёртку, так сказать, которая обёртывает функцию-аргумент и затем изменяет её поведение.

5. Какова структура декоратора функций?

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

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


@decorator_function
def hello_world():
    print('Hello world!')


if __name__ == '__main__':
    hello_world()
```

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

```python
import functools


def decoration(*args):
    def dec(func):
        @functools.wraps(func)
        def decor():
            func()
            print(*args)
        return decor
    return dec


@decoration('This is args')
def func_ex():
    print('Look')


if __name__ == '__main__':
    func_ex()
```

Вывод: в ходе выполнения практической работы были приобретены навыки по работе декораторами функций при написании программ с помощью языка программирования Python.