

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

Институт цифрового развития
Кафедра инфокоммуникаций

ОТЧЁТ

по лабораторной работе №2.12

Дисциплина: «Программирование на Python»
Тема: «Декораторы функций в языке Python»

Выполнил:
Епифанов Алексей Александрович
2 курс, группа ИВТ-б-о-22-1,
09.03.01 «Информатика и
вычислительная техника»,
направленность (профиль)
«Программное обеспечение средств
вычислительной
техники и автоматизированных систем
», очная форма обучения

(подпись)

Руководитель практики:
Воронкин Роман Александрович

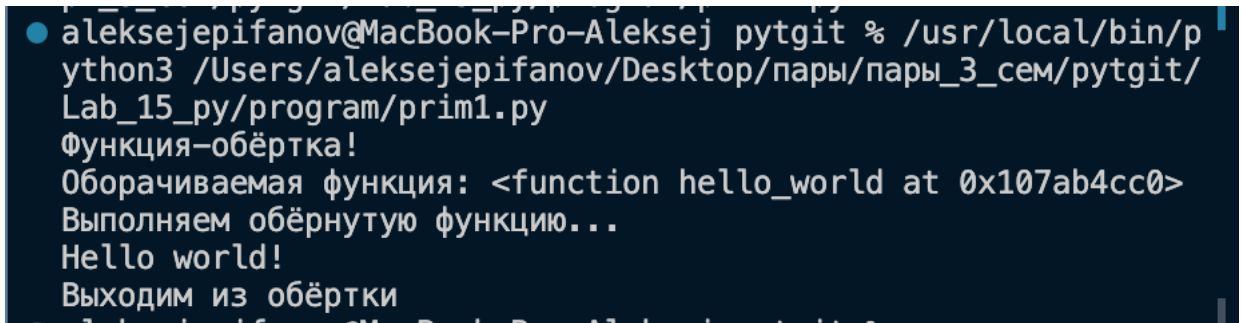
(подпись)

Отчет защищен с оценкой _____ Дата защиты _____
Ставрополь, 2023 г.

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

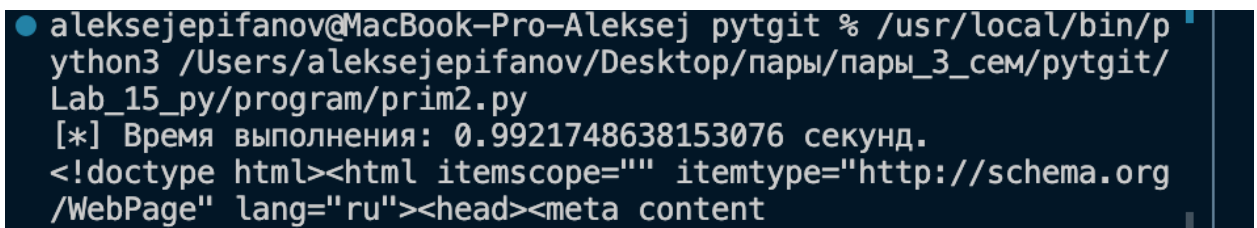
Порядок выполнения работы:

1. Создал новый репозиторий, клонировал его, в нем создал ветку developer и перешел на нее.
2. Проработал примеры лабораторной работы:



```
aleksejepifanov@MacBook-Pro-Aleksej pytgit % /usr/local/bin/python3 /Users/aleksejepifanov/Desktop/пары/пары_3_сем/pytgit/Lab_15_py/program/prim1.py
Функция-обёртка!
Оборачиваемая функция: <function hello_world at 0x107ab4cc0>
Выполняем обёрнутую функцию...
Hello world!
Выходим из обёртки
```

Рисунок 1. Вывод примера 1



```
aleksejepifanov@MacBook-Pro-Aleksej pytgit % /usr/local/bin/python3 /Users/aleksejepifanov/Desktop/пары/пары_3_сем/pytgit/Lab_15_py/program/prim2.py
[*] Время выполнения: 0.9921748638153076 секунд.
<!doctype html><html itemscope="" itemtype="http://schema.org/
/WebPage" lang="ru"><head><meta content
```

Рисунок 2. Вывод примера 2

3. Выполнил индивидуальное задание вариант 10: объявите функцию, которая принимает строку, удаляет из нее все подряд идущие пробелы и переводит ее в нижний регистр – малые буквы. Результат (строка) возвращается функцией. Определите декоратор, который строку, возвращенную функцией, переводит в азбуку Морзе, используя следующий словарь для замены русских букв и символа пробела на соответствующие последовательности из точек и тире. Преобразованная строка возвращается декоратором. Примените декоратор к функции и вызовите декорированную функцию. Результат работы отобразите на экране.

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
def morze_decorator(func):
    """
```

Декоратор функции, возвращающий строку, преобразованную в азбуку морзе.

```

"""
morze = {
    'a': '.-.', 'б': '-...', 'в': '---', 'г': '--.',
    'д': '-..', 'е': '.', 'ё': '.', 'ж': '...-', 'з': '-...',
    'и': '..-', 'й': '----', 'к': '-.-', 'л': '-..', 'м': '--',
    'н': '-.', 'о': '---', 'п': '---', 'р': '-.-', 'с': '...',
    'т': '-', 'у': '-.-', 'ф': '-.-', 'х': '....', 'ц': '-.-',
    'ч': '----', 'ш': '----', 'щ': '---.', 'ъ': '---.', 'ы': '-.-',
    'ь': '-.-', 'э': '-..', 'ю': '-.-', 'я': '-.-', ' ': '----'
}

def wrapper(arg):
    """
    Функция-обертка.
    """
    arg = func(arg)
    return ' '.join([morze[char] for char in arg if char in morze])
return wrapper

@morze_decorator
def process_string(s):
    """
    Переводит строку в нижний регистр и удаляет из нее лишние пробелы.
    """
    s = ' '.join(s.lower().split())
    return s

if __name__ == "__main__":
    input_string = "ПрИвЕт Мир Я Алексей"
    output_string = process_string(input_string)
    print(f"{output_string = }")

```

Рисунок 3. Вывод программы ind.py

Ответы на контрольные вопросы:

1. Что такое декоратор?

Декоратор — это функция, которая позволяет обернуть другую функцию для расширения её функциональности без непосредственного изменения её кода. Декораторы можно рассматривать как практику метапрограммирования, когда программы могут работать с другими программами как со своими данными.

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. С функцией все это делать можно, поэтому ее и можно назвать объектом первого класса.

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?

Пример:

```
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
```

Здесь `decorator_function()` является функцией-декоратором. Она является функцией высшего порядка, так как принимает функцию в качестве аргумента, а также возвращает функцию. Внутри `decorator_function()` определена другая функция, которая обёртывает функцию-аргумент и затем изменяет её поведение. Декоратор возвращает эту обёртку.

Перед функцией остается прописать `@decorator_function`.

Однако выражение с `@` является всего лишь синтаксическим сахаром для `hello_world = decorator_function(hello_world)`.

5. Какова структура декоратора функций?

```
def decorator(func):
    def wrapper(*args, **kwargs):
        # Код до вызова целевой функции
        result = func(*args, **kwargs) # Вызов целевой функции
        # Код после вызова целевой функции
        return result
```

```
return wrapper
```

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

В Python можно передавать параметры декоратору, добавляя еще один уровень вложенности.

Например:

```
def decorator_with_parameters(param1, param2):  
    def actual_decorator(func):  
        def wrapper(*args, **kwargs):  
            print(f"Decorator parameters: {param1 }, {param2}")  
            result = func(*args, **kwargs)  
            return result  
        return wrapper  
    return actual_decorator
```

Вызов декоратора с параметрами будет выглядеть так:

```
@decorator_with_parameters(p1, p2)
```

Вывод: в результате выполнения работы были приобретены навыки по работе с декораторами функций при написании программ с помощью языка программирования Python версии 3.x.