

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное
образовательное учреждение высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Кафедра инфокоммуникаций

Отчет по лабораторной работе №2.12

Декораторы функций в языке Python

по дисциплине «Технологии программирования и алгоритмизация»

Выполнил

студент группы ИВТ-б-о-20-1

Дыбов Д.В. « » _____ 20__ г.

Работа защищена « » _____ 20__ г.

Проверил Воронкин Р.А. _____

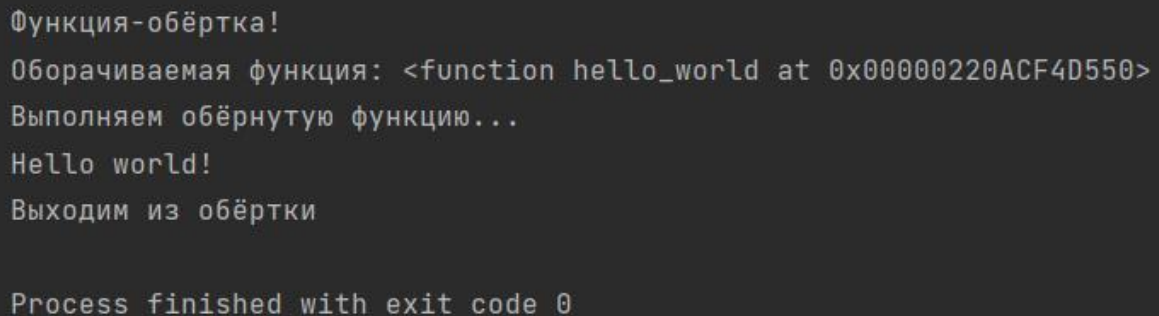
(подпись)

Ставрополь 2021

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

Ход работы

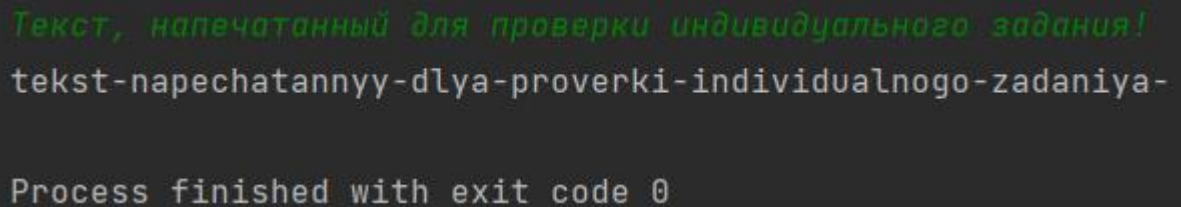
1. Создал новый репозиторий для лабораторной работы №2.12;
2. Клонировал созданный репозиторий на компьютер;
3. Создал новый PyCharm проект в папке репозитория;
4. Проработал пример;
5. Проверил пример на работоспособность:



```
Функция-обёртка!  
Обрабатываемая функция: <function hello_world at 0x00000220ACF4D550>  
Выполняем обёрнутую функцию...  
Hello world!  
Выходим из обёртки  
  
Process finished with exit code 0
```

Рисунок 1 – Результат выполнения примера

6. Выполнил индивидуальное задание;
7. Проверил индивидуальное задание на работоспособность:



```
Текст, напечатанный для проверки индивидуального задания!  
tekst-napechatannyy-dlya-proverki-individualnogo-zadaniya-  
  
Process finished with exit code 0
```

Рисунок 2 – Результат выполнения индивидуального задания

12. С помощью сайта проверил пример на наличие ошибок;
13. Результат не выдал ошибок;

Python code

```
1 #!/usr/bin/env python3
2 # -*- coding: utf-8 -*-
3
4
5 def decorator_function(func):
6     def wrapper():
7         print('Функция-обёртка!')
8         print('Оборачиваемая функция: {}'.format(func))
9         print('Выполняем обёрнутую функцию...')
10        func()
11        print('Выходим из обёртки')
12    return wrapper
13
```

No syntax errors detected :)

Рисунок 3 – Проверка кода на наличие ошибок

14. С помощью сайта проверил индивидуальное задание на наличие ошибок;
15. Результат не выдал ошибок;

Python code

```
1 #!/usr/bin/env python3
2 # -*- coding: utf-8 -*-
3
4
5 def to_lat(func):
6     def simv(text, chars=' !?'):
7         tmp = ''.join(map(lambda x: x if x not in chars else '-', func(text)))
8         while '--' in tmp:
9             tmp = tmp.replace('--', '-')
10        return tmp
11    return simv
12
13
```

No syntax errors detected :)

Рисунок 4 – Проверка индивидуального задания на наличие ошибок

20. Отправил все изменения на репозиторий.

Контрольные вопросы

1. Что такое декоратор?

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

2. Почему функции являются объектами первого класса?

В Python всё является объектом, а не только объекты, которые вы создаёте из классов. В этом смысле Python полностью соответствует идеям объектно-ориентированного программирования. Это значит, что в Python всё это – объекты:

- числа;
- строки;

- классы;
- функции.

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.

3. Каково назначение функций высших порядков?

В Python используются некоторые концепции из функциональных языков вроде Haskell и OCaml. Пропустим формальное определение функционального языка и перейдём к двум его характеристикам, свойственным Python:

- функции являются объектами первого класса;
- следовательно, язык поддерживает функции высших порядков.

Функциональному программированию присущи и другие свойства вроде отсутствия побочных эффектов, но мы здесь не за этим. Лучше сконцентрируемся на другом – функциях высших порядков. Что есть функция высшего порядка?

4. Как работают декораторы?

Внутри функции-декоратора определяется другая функция, обёртка, так сказать, которая обёртывает функцию-аргумент и затем изменяет её поведение.

Декоратор возвращает эту обёртку. Просто добавив `@decorator` перед определением функции, её поведение меняется. Однако выражение с `@` является всего лишь синтаксическим сахаром. Иными словами, выражение `@decorator` вызывает функцию-декоратор с функцией в качестве аргумента.

5. Какова структура декоратора функций?

В начале объявляется функция в которой будет использоваться декорируемая функция. Далее объявляется декорируемая функция перед

которой используется конструкция `@decorator`, где “decorator” имя функции, использующей декорируемую функцию.

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

Для этого после объявления декоратора в скобках нужно указать необходимые параметры: `@decorator(p1)`

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