

**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

Пуценко И. А. « » 2022г.

Подпись студента _____

Работа защищена « » _____ 20__ г.

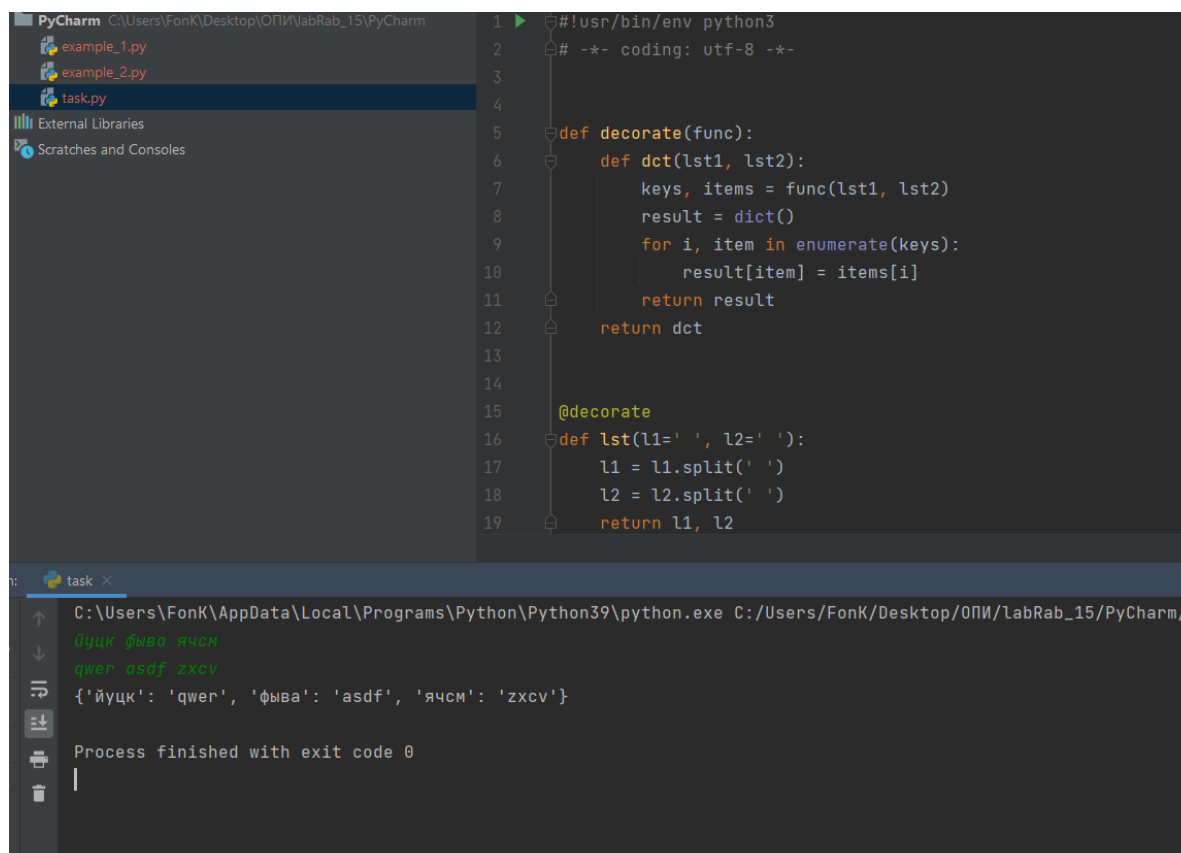
Проверил Воронкин Р.А. _____

(подпись)

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



The screenshot displays the PyCharm IDE interface. The left sidebar shows a project structure with files `example_1.py`, `example_2.py`, and `task.py`. The main editor window shows the code in `task.py`, which defines a `decorate` function and a `lst` function decorated with `@decorate`. The `lst` function takes two strings, splits them by spaces, and returns a dictionary mapping the first string's words to the second string's words. The bottom console shows the execution of `lst('йуцк', 'фыва')`, resulting in the dictionary `{'йуцк': 'qwer', 'фыва': 'asdf', 'ячсм': 'zxcv'}`. The process finished with exit code 0.

```
1 #!/usr/bin/env python3
2 # -*- coding: utf-8 -*-
3
4
5 def decorate(func):
6     def dct(lst1, lst2):
7         keys, items = func(lst1, lst2)
8         result = dict()
9         for i, item in enumerate(keys):
10             result[item] = items[i]
11         return result
12     return dct
13
14
15 @decorate
16 def lst(l1=' ', l2=' '):
17     l1 = l1.split(' ')
18     l2 = l2.split(' ')
19     return l1, l2
```

task

C:\Users\FonK\AppData\Local\Programs\Python\Python39\python.exe C:/Users/FonK/Desktop/ОПИ/LabRab_15/PyCharm/task.py

йуцк фыва ячсм
qwer asdf zxcv
{'йуцк': 'qwer', 'фыва': 'asdf', 'ячсм': 'zxcv'}

Process finished with exit code 0

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 называются 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?

```
#!/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?

```
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.