МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»
ИНСТИТУТ ЦИФРОВОГО РАЗВИТИЯ

Отчет о лабораторной работе №15 по дисциплине «Основы программной инженерии»

Выполнил:
Мамонтов Д.В.,
2 курс, группа ПИЖ-б-о-20-1,
Проверил:
Доцент кафедры инфокоммуникаций,
Воронкин Р.А.

Ставрополь, 2022 г

ХОД РАБОТЫ

```
>>> def hello_world():
...     print('Hello world!')
...
>>> type(hello_world)
<class 'function'>
>>> class Hello:
...     pass
...
>>> type(Hello)
<class 'type'>
>>> type(10)
<class 'int'>
```

Рисунок 1 – изменение типа переменной

```
>>> hello = hello_world
>>> hello()
Hello world!
```

Рисунок 2 – присвоение

```
>>> def wrapper_function():
...     def hello_woeld():
...             print('Hello world!')
...     hello_world()
...
>>> wrapper_function()
Hello world!
```

Рисунок 3 – применение декоратора

```
>>> def higher_order(func):
...     print('Получена функция {} в качестве аргкмента'.format(func))
...     func()
...     return func
...
>>> higher_order(hello_world)
Получена функция <function hello_world at 0x0000025A2492A830> в качестве аргкмента
Hello world!
<function hello_world at 0x0000025A2492A830>
```

Рисунок 4 – функция как значение аргумента

```
>>> def decorator_function(func):
...     def wrapper():
...             print('The wrapper!')
...             print('The wrapped function is: {}'.format(func))
...             print('Making wrapped function...')
...             func()
...             print('Exit')
...     return wrapper
...
>>> @decorator_function
... def hello_world():
...     print('Hello world!')
...
>>> hello_world()
The wrapper!
The wrapped function is: <function hello_world at 0x0000025A2492A680>
Making wrapped function...
Hello world!
Exit
```

Рисунок 5 – применение декоратора

ИНДИВИДУАЛЬНОЕ ЗАДАНИЕ

Задание:

Вводятся два списка (каждый с новой строки) из слов, записанных через пробел. Имеется функция, которая преобразовывает эти две строки в два списка слов и возвращает эти списки. Определите декоратор для этой функции, который из этих двух списков формирует словарь, в котором ключами являются слова из первого списка, а значениями – соответствующие элементы из второго списка. Полученный словарь должен возвращаться при вызове декоратора. Примените декоратор к первой функции и вызовите ее. Результат (словарь) отобразите на экране.

Код:

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

def start_func(text1, text2):
    return text1.split(), text2.split()


def decorated(func):
    def decorated_m(text1, text2):
        data = func(text1, text2)
        return dict(zip(*data))
    return decorated_m


start_func = decorated(start_func)
x = input()
y = input()
print(start_func(x, y))
```

```
Петя Олег Антон Даниил
Художник Музыкант Спортсмен Никто
{'Петя': 'Художник', 'Олег': 'Музыкант', 'Антон': 'Спортсмен', 'Даниил': 'Никто'}
```

Рисунок 1 – результат работы программы

КОНТРОЛЬНЫЕ ВОПРОСЫ

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?

Используя конструкцию @decorator def function(), мы делаем конструкцию вида function=decorator(function), а это значит, что значению нашей функции будет соответствовать значение функции, которую вернул декоратор.

5. Какова структура декоратора функций?

```
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
```

6. Как можно передать параметры декоратору, а не декорируемой функции?

```
def decorator_setup(start=0): def decorator_function(func):


        def wrapper(args):
            result = func(args) return result + start


        return wrapper


    return decorator_function
```