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:
Чернова Софья Андреевна,
2 курс, группа ПИЖ-б-о-20-1,
Проверил:
Доцент кафедры инфокоммуникаций,
Воронкин Р.А.

Ставрополь, 2021 г

1. Ход работы:

1.1 Пример 1 (рис. 1)

```
>>> 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

1.2 Пример 2 (рис. 2)

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

Рисунок 2 – присвоение

1.3 Пример 3 (рис. 3)

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

Рисунок 3 – применение декоратора

1.4 Пример 4 (рис. 4)

```
>>> 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

1.5 Пример 5 (рис. 5)

```
>>> 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

1.6 Индивидуальное задание (рис. 6, 7)

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

def decorator_setup(start=0):
    def decorator_function(func):
        def wrapper(args):
            result = func(args)
            return result + start

        return wrapper

    return decorator_function


@decorator_setup(start=5)
def ind(data):
    dig = list(map(int, data.split()))
    return sum(dig)


def main():
    string = input("Enter the numbers:\n")
    result = ind(string)
    print(result)


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

Рисунок 6 – код программы

```
Enter the numbers:
5 3 8 12 9
42

Process finished with exit code 0
```

Рисунок 7 – результат работы программы

2. 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.

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. Самостоятельно изучить 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
```