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 №2.12 по дисциплине основы программной инженерии

Выполнил:
Шальнев Владимир Сергеевич,
2 курс, группа ПИЖ-б-о-20-1,

Проверил:
Доцент кафедры
прикладной математики и
компьютерной безопасности,
Воронкин Р.А.

Отчет защищен с оценкой _____ Дата защиты_____

Ставрополь, 2021 г.

ВЫПОЛНЕНИЕ:

```
F:\pythonProject>git clone https://github.com/HAXF13D/laboratory-15
Cloning into 'laboratory-15'...
remote: Enumerating objects: 11, done.
remote: Total 11 (delta 0), reused 0 (delta 0), pack-reused 11
Unpacking objects: 100% (11/11), done.
```

Клонирование репозитория

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

Пример 1

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

Пример 2

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

Пример 3

```
>>> def higher_order(func):
...     print('Получена функция {} в качестве аргумента'.format(func))
...     func()
...     return func
...
>>> higher_order(hello_world)
Получена функция <function hello_world at 0x000001FD84ACE0D0> в качестве аргумента
Hello world!
<function hello_world at 0x000001FD84ACE0D0>
```

Пример 4

```
>>> 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!')
...
>>> hello_world()
Функция-обёртка!
Оборачиваемая функция: <function hello_world at 0x000001FD85285790>
Выполняем обёрнутую функцию...
Hello world!
Выходим из обёртки
```

Пример 5

```
>>> def benchmark(func):
...     import time
...
...     def wrapper():
...             start = time.time()
...             func()
...             end = time.time()
...             print('[*] Время выполнения: {} секунд.'.format(end-start))
...     return wrapper
...
>>> @benchmark
... def fetch_webpage():
...     import requests
...     webpage = requests.get('https://google.com')
...
>>> fetch_webpage()
[*] Время выполнения: 0.5772018432617188 секунд.
```

Пример 6

```
>>> def benchmark(func):
...     import time
...
...     def wrapper(*args, **kwargs):
...             start = time.time()
...             return_value = func(*args, **kwargs)
...             end = time.time()
...             print('[*] Время выполнения: {} секунд.'.format(end-start))
...             return return_value
...     return wrapper
...
>>> @benchmark
... def fetch_webpage(url):
...     import requests
...     webpage = requests.get(url)
...     return webpage.text
...
>>> webpage = fetch_webpage('https://google.com')
[*] Время выполнения: 0.3326547145843506 секунд.
>>> print(webpage)
<!doctype html><html itemscope="" itemtype="http://schema.org/WebPage" lang="r
;&#1077;&#1088;&#1085;&#1077;&#1090;&#1077;: &#1074;&#1077;&#1073; &#1089;&#10
085;&#1086;&#1075;&#1086;&#1077; &#1076;&#1088;&#1091;&#1075;&#1086;&#1077;."
asonal-holidays-2021-6753651837109324-6752733080595605-cst.gif" itemprop="imag
 &#8211; 2021" property="twitter:title"><meta content="&#1050;&#1072;&#1083;&#
itter:description"><meta content="&#1050;&#1072;&#1083;&#1077;&#1085;&#1076;&#
ontent="summary_large_image" property="twitter:card"><meta content="@GoogleDoo
```

Пример 7

```python
#!/usr/bin/env 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 individual_func(data):
    digits = list(map(int, data.split()))
    return sum(digits)


def main():
    string = input(
        "Введите строку целых чисел, разделенных пробелом:\n"
    )
    result = individual_func(string)
    print(result)


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

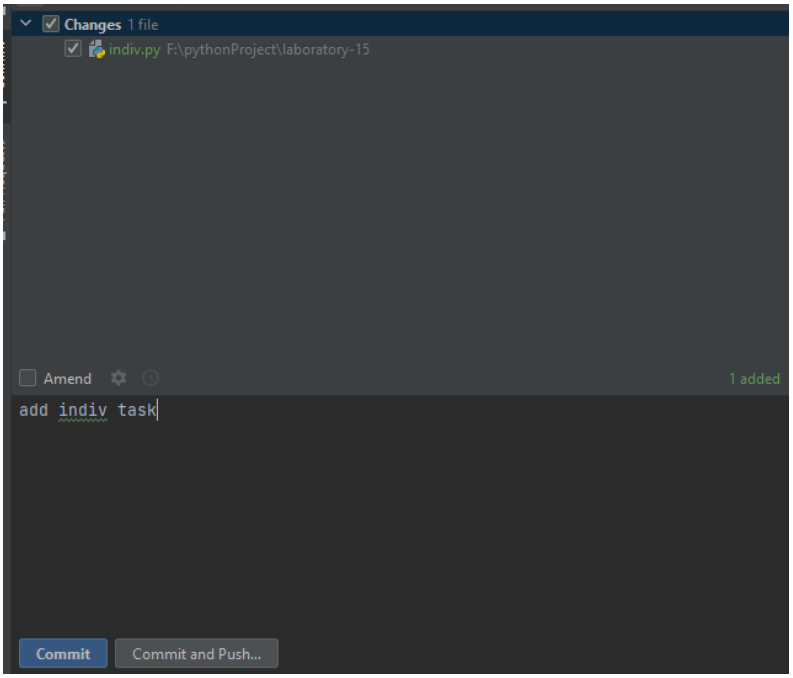
Индивидуальное задание 1

```
Введите строку целых чисел, разделенных пробелом:
1 2 3
11
```

Значение №1

```
Введите строку целых чисел, разделенных пробелом:
1
6
```

Значение №2



Коммит изменений

ОТВЕТЫ НА 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.

4. 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?

Для этого, нам нужно создать конструкция вида:

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

        return wrapper

    return decorator_function
```