ФЕДЕРАЛЬНОЕ АГЕНСТВО ЖЕЛЕЗНОДОРОЖНОГО ТРАНСПОРТА

Федеральное государственное бюджетное образовательное учреждение высшего образования

«ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ПУТЕЙ СООБЩЕНИЯ Императора Александра I»

Кафедра «Информационные и вычислительные системы»

Дисциплина «Программирование на языках высокого уровня (Python)»

ОТЧЁТ

ПО ЛАБОРАТОРНОЙ РАБОТЕ № 6

Выполнил студент Шефнер А.
Факультет: АИТ
Группа: ИВБ-211

Проверил: Баталов Д.И.

Санкт-Петербург

2023

Оценочный лист результатов ЛР № 6

Ф.И.О. студента _____Шефнер Альберт_____

Группа _____ИВБ-211_____

№ п/п	Материалы необходимые для оценки знаний, умений и навыков	Показатель оценивания	Критерии Оценивания	Шкала оценивания	Оценка
1	Лабораторная работа№	Соответствие методике выполнения	Соответствует	7	
			Не соответствует	0	
		Срок выполнения	Выполнена в срок	2	
			Выполнена с опозданием на 2 недели	0	
		оформление	Соответствует требованиям	1 0	
			Не соответствует		
	ИТОГО количество баллов			10	

Доцент кафедры

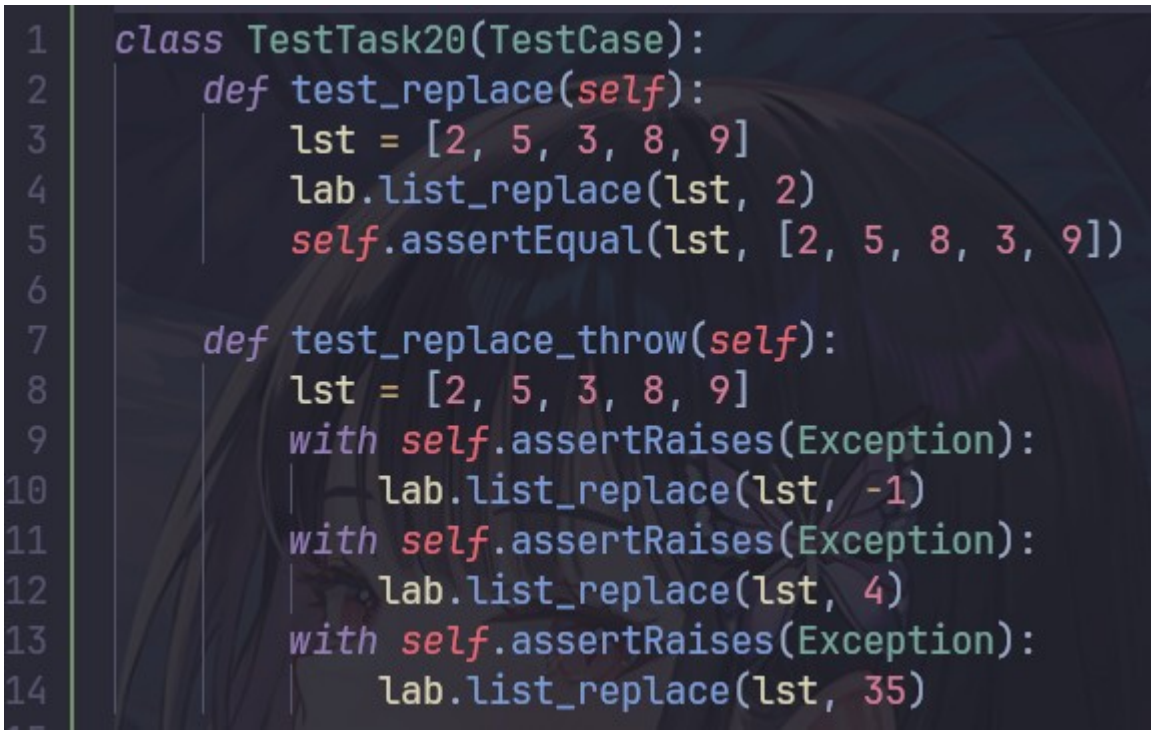
«Информационные и вычислительные

системы» Баталов Д.И. «__»

_____2023 г.

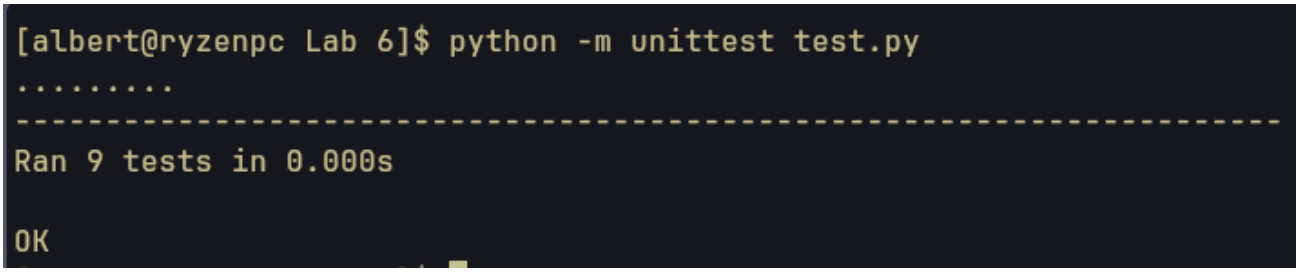
Тестирование

Для тестов используется модуль встроенной библиотеки unittest. Формат тестов везде одинаковы: сначала вызов функции или процедуры, потом сверка её результата с ожидаемым с помощью assert. Если assert имеет два параметра, тестируемое значение всегда слева, ожидаемое всегда справа. Вот пример одного из тестов:



```
class TestTask20(TestCase):
    def test_replace(self):
        lst = [2, 5, 3, 8, 9]
        lab.list_replace(lst, 2)
        self.assertEqual(lst, [2, 5, 8, 3, 9])

    def test_replace_throw(self):
        lst = [2, 5, 3, 8, 9]
        with self.assertRaises(Exception):
            lab.list_replace(lst, -1)
        with self.assertRaises(Exception):
            lab.list_replace(lst, 4)
        with self.assertRaises(Exception):
            lab.list_replace(lst, 35)
```

Программа успешно проходит все приведённые ниже тесты.



```
[albert@ryzenpc Lab 6]$ python -m unittest test.py
.........
----------------------------------------------------------------------
Ran 9 tests in 0.000s

OK
```

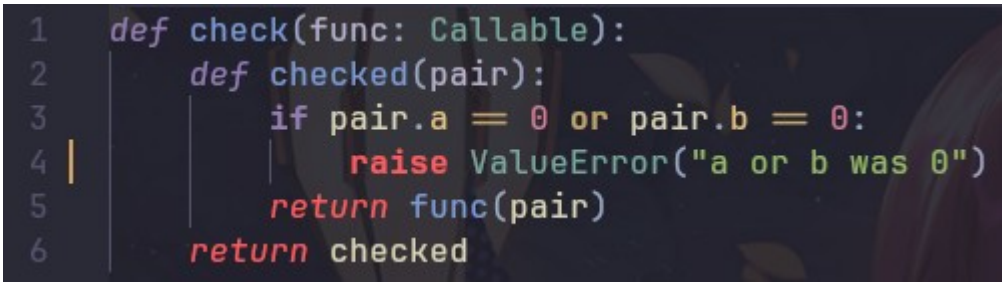
Более подробные результаты:

```
[albert@ryzenpc Lab 6]$ python -m unittest test.py -v
test_in_list (test.TestFunctions.test_in_list) ... ok
test_in_list_error (test.TestFunctions.test_in_list_error) ... ok
test_str_upper (test.TestFunctions.test_str_upper) ... ok
test_str_upper_error (test.TestFunctions.test_str_upper_error) ... ok
test_pair_error (test.TestPair.test_pair_error) ... ok
test_pair_operations (test.TestPair.test_pair_operations) ... ok
test_russian_p_error (test.TestRussianPError.test_russian_p_error) ... ok
test_safe_pair (test.TestSafePair.test_safe_pair) ... ok
test_safe_pair_with_zero (test.TestSafePair.test_safe_pair_with_zero) ... ok

----------------------------------------------------------------------
Ran 9 tests in 0.000s

OK
```

1. Напишите класс, реализующий все арифметические операции над двумя значениями (a и b). В случае, если одно из значений при вызове операции равно нулю, генерируется исключение.

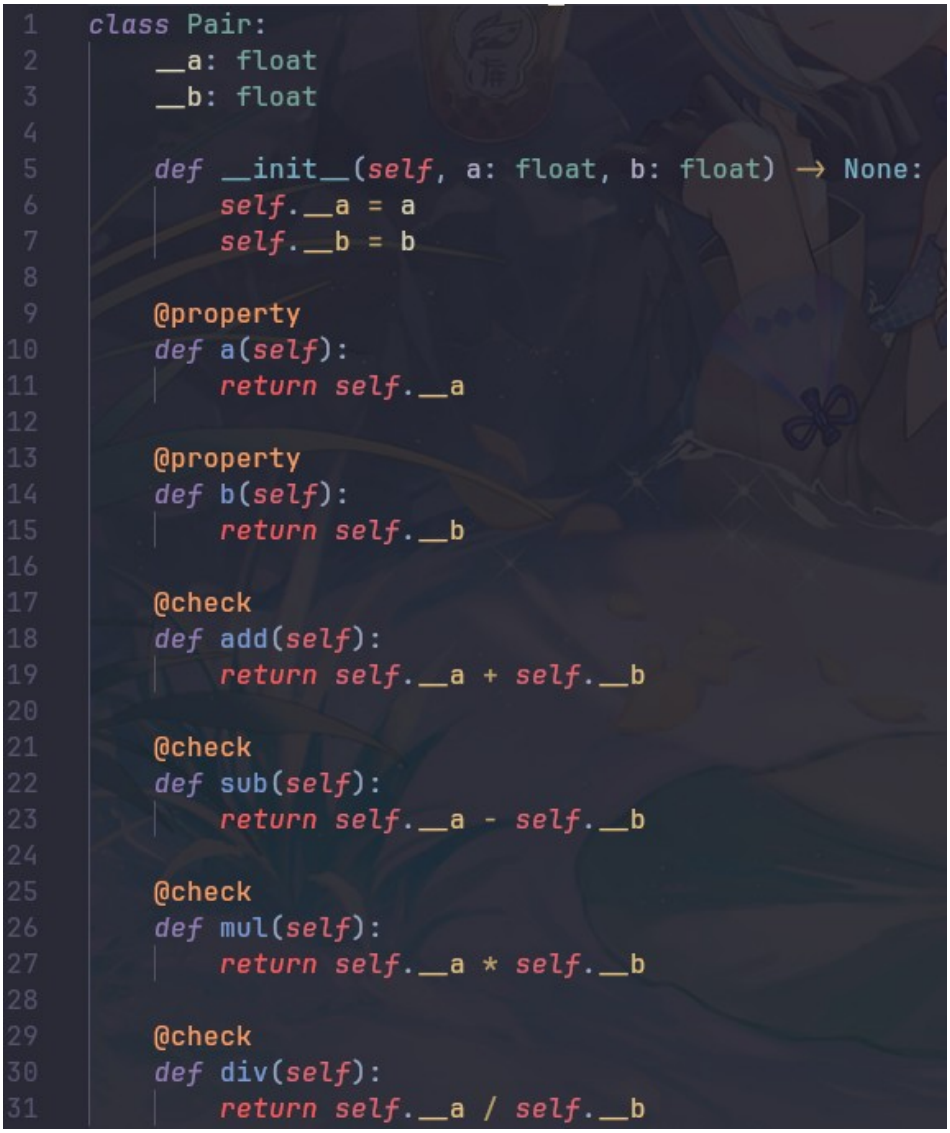
Решение:

Декоратор, реализующий проверку:



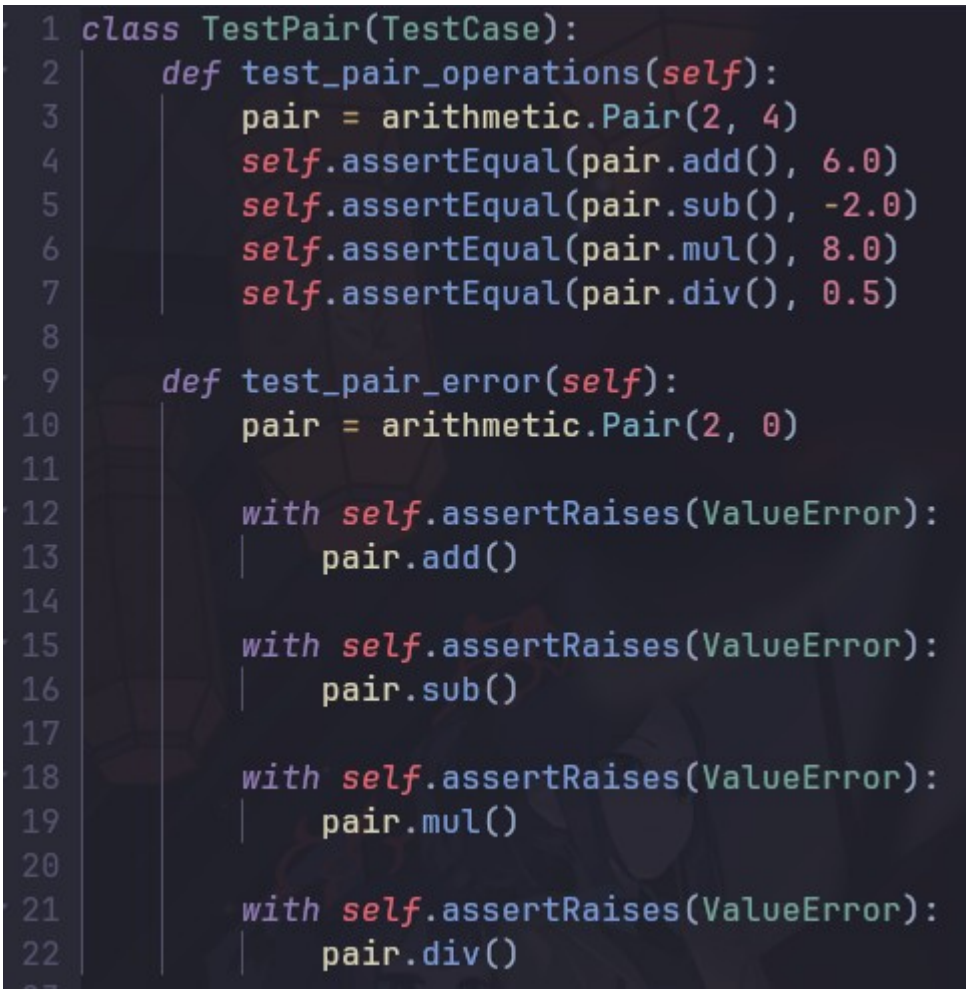
```python
def check(func: Callable):
    def checked(pair):
        if pair.a == 0 or pair.b == 0:
            raise ValueError("a or b was 0")
        return func(pair)
    return checked
```

Класс:



```python
class Pair:
    __a: float
    __b: float

    def __init__(self, a: float, b: float) -> None:
        self.__a = a
        self.__b = b

    @property
    def a(self):
        return self.__a

    @property
    def b(self):
        return self.__b

    @check
    def add(self):
        return self.__a + self.__b

    @check
    def sub(self):
        return self.__a - self.__b

    @check
    def mul(self):
        return self.__a * self.__b

    @check
    def div(self):
        return self.__a / self.__b
```

Тесты:

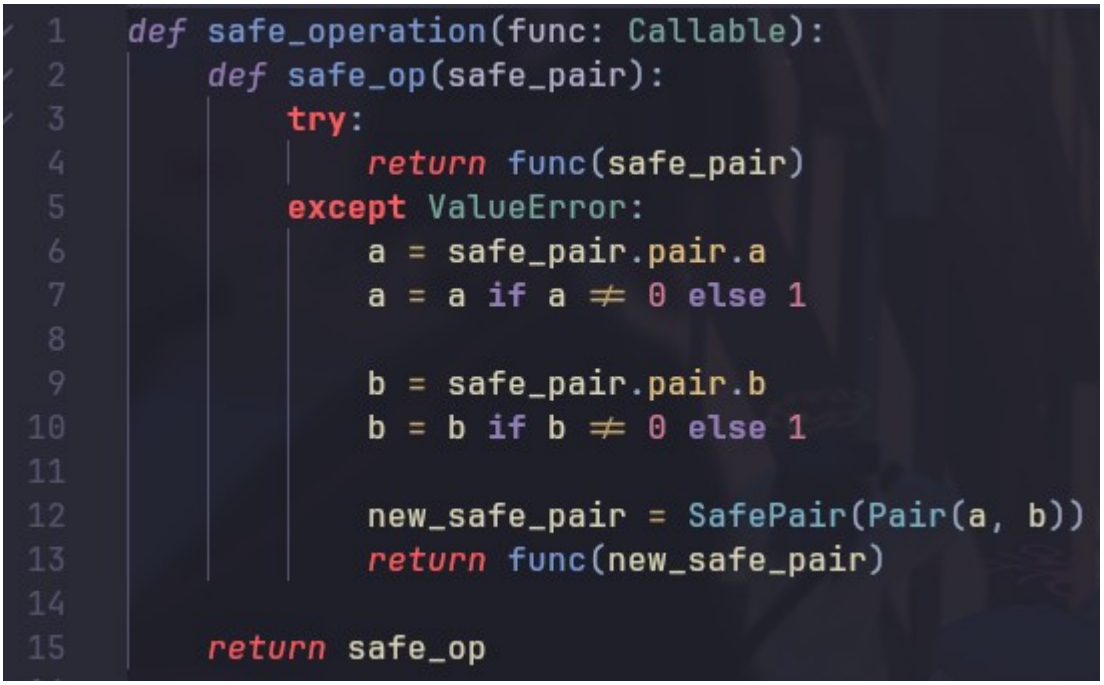


```python
class TestPair(TestCase):
    def test_pair_operations(self):
        pair = arithmetic.Pair(2, 4)
        self.assertEqual(pair.add(), 6.0)
        self.assertEqual(pair.sub(), -2.0)
        self.assertEqual(pair.mul(), 8.0)
        self.assertEqual(pair.div(), 0.5)

    def test_pair_error(self):
        pair = arithmetic.Pair(2, 0)

        with self.assertRaises(ValueError):
            pair.add()

        with self.assertRaises(ValueError):
            pair.sub()

        with self.assertRaises(ValueError):
            pair.mul()

        with self.assertRaises(ValueError):
            pair.div()
```

2. Напишите класс, реализующий такие арифметические действия, как деление и умножение. Если одно из значений при вызове операции равно нулю, генерируется исключение, значение ноль меняется на 1 и вычисление операции продолжается.

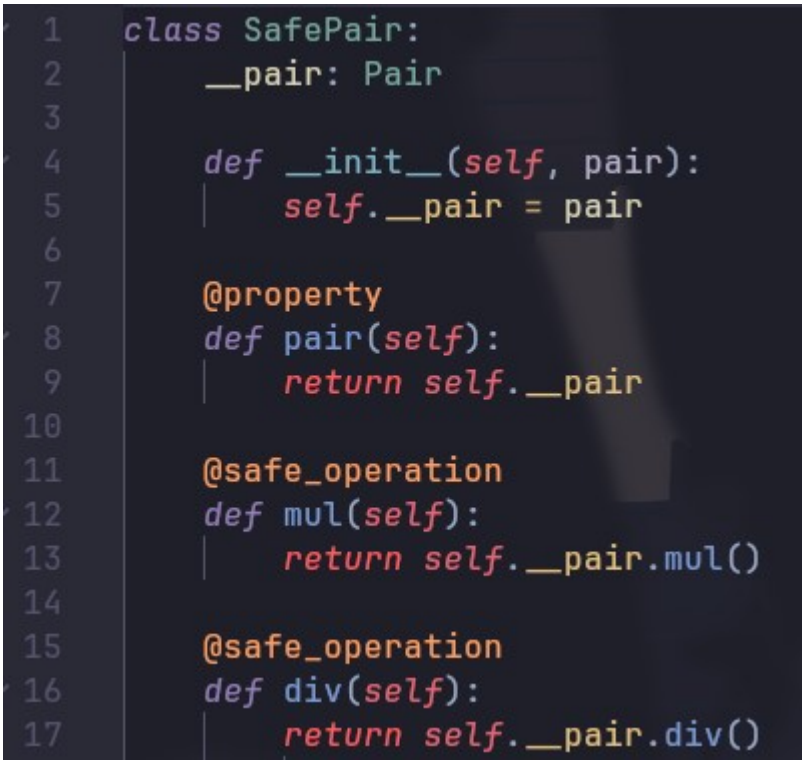
Решение:

Декоратор, оборачивающий метод в try/except блок для безопасного использования класса Pair



```python
def safe_operation(func: Callable):
    def safe_op(safe_pair):
        try:
            return func(safe_pair)
        except ValueError:
            a = safe_pair.pair.a
            a = a if a != 0 else 1

            b = safe_pair.pair.b
            b = b if b != 0 else 1

            new_safe_pair = SafePair(Pair(a, b))
            return func(new_safe_pair)

    return safe_op
```

Класс:



```python
class SafePair:
    __pair: Pair

    def __init__(self, pair):
        self.__pair = pair

    @property
    def pair(self):
        return self.__pair

    @safe_operation
    def mul(self):
        return self.__pair.mul()

    @safe_operation
    def div(self):
        return self.__pair.div()
```

Тесты:

```python
class TestSafePair(TestCase):
    def test_safe_pair(self):
        pair = arithmetic.Pair(2, 4)
        safe_pair = arithmetic.SafePair(pair)
        self.assertEqual(pair.mul(), safe_pair.mul())
        self.assertEqual(pair.div(), safe_pair.div())

    def test_safe_pair_with_zero(self):
        pair = arithmetic.Pair(0, 5)
        safe_pair = arithmetic.SafePair(pair)
        self.assertEqual(safe_pair.mul(), 5.0)
        self.assertEqual(safe_pair.div(), 0.2)
```

3. Напишите функцию, возводящую строку в верхний регистр. Добавьте проверку на то, что на вход функции подается не пустая строка.

Решение:

```python
def str_upper(s: str) -> str:
    if not isinstance(s, str):
        raise ValueError("s must be str.")

    if s == "":
        raise ValueError("s must not be empty string.")

    return s.upper()
```

Тесты:

```python
    def test_str_upper(self):
        s = "Xd with Str"
        result = functions.str_upper(s)
        self.assertEqual(result, "XD WITH STR")

    def test_str_upper_error(self):
        with self.assertRaises(ValueError):
            functions.str_upper("")

        with self.assertRaises(ValueError):
            functions.str_upper(5) # type: ignore
```

4. Напишите функцию, проверяющую вхождение задаваемого элемента в список. Добавьте проверку на то, что список не пустой.

Решение:

```python
def in_list(lst: list, elem) -> bool:
    if not isinstance(lst, list):
        raise ValueError("lst must be List.")

    if len(lst) == 0:
        raise ValueError("lst must not be empty list.")

    return elem in lst
```

Тесты:

```python
    def test_in_list(self):
        lst = [2, 5, 6]
        self.assertTrue(functions.in_list(lst, 2))
        self.assertFalse(functions.in_list(lst, 7))

    def test_in_list_error(self):
        with self.assertRaises(ValueError):
            functions.in_list([], 5)

        with self.assertRaises(ValueError):
            functions.in_list(5, 3) # type: ignore
```

5. Реализуйте собственный класс исключения, которое будет генерироваться каждый раз, когда в строке, которая является аргументом для функции, присутствует символ «п».

Решение:

```python
class RussianPError(Exception):
    def __init__(self, message: str) -> None:
        super().__init__(message)


def str_lower(s: str):
    if 'п' in s:
        raise RussianPError("'п' must not be in s")

    return s.lower()
```

Тесты:

```python
class TestRussianPError(TestCase):
    def test_russian_p_error(self):
        self.assertEqual(perror.str_lower("Str"), "str")

        with self.assertRaises(perror.RussianPError):
            perror.str_lower("привет")
```