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

ИНСТИТУТ ЦИФРОВОГО РАЗВИТИЯ

Отчёт по лабораторной работе №4.2 по дисциплине:

Основы программной инженерии

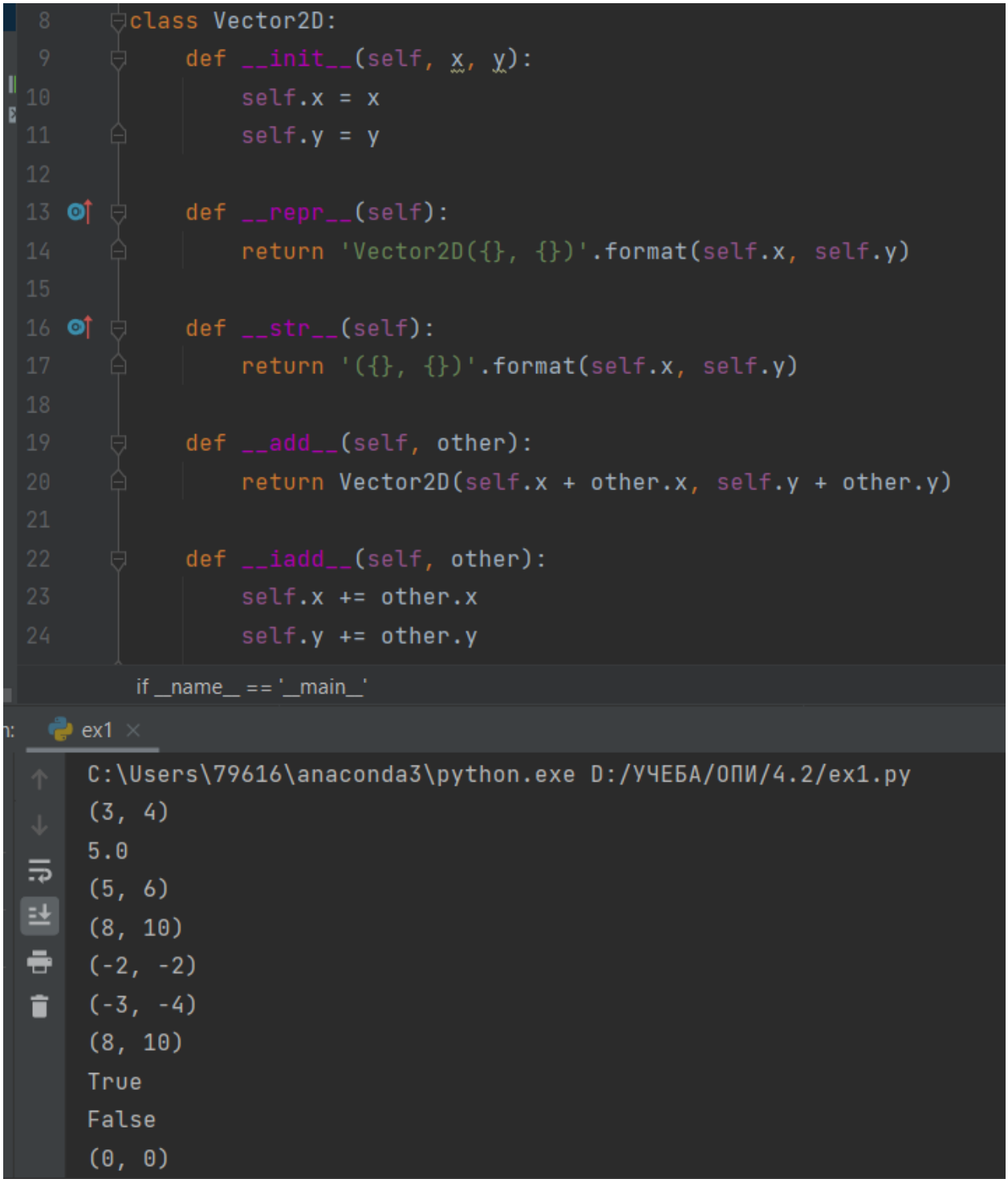
Выполнила:
студент группы ПИЖ-б-о-20-1
Лазарева Дарья Олеговна

Проверил:
доцент кафедры инфокоммуникаций
Романкин Р.А.

Ставрополь, 2022 г.

ВЫПОЛНЕНИЕ:

1. Примеры из методических указаний



```
class Vector2D:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def __repr__(self):
        return 'Vector2D({}, {})'.format(self.x, self.y)

    def __str__(self):
        return '({}, {})'.format(self.x, self.y)

    def __add__(self, other):
        return Vector2D(self.x + other.x, self.y + other.y)

    def __iadd__(self, other):
        self.x += other.x
        self.y += other.y

if __name__ == '__main__'
```

```
ex1
C:\Users\79616\anaconda3\python.exe D:/УЧЕБА/ОПИ/4.2/ex1.py
(3, 4)
5.0
(5, 6)
(8, 10)
(-2, -2)
(-3, -4)
(8, 10)
True
False
(0, 0)
```

```python
class Rational:
    def __init__(self, a=0, b=1):
        a = int(a)
        b = int(b)
        if b == 0:
            raise ValueError("Illegal value of the denominator")
        self.__numerator = a
        self.__denominator = b
        self.__reduce()

    # Сокращение дроби.
    def __reduce(self):
        # Функция для нахождения наибольшего общего делителя
        def gcd(a, b):
```

```
if __name__ == '__main__'
```

ex2

```
C:\Users\79616\anaconda3\python.exe D:/УЧЕБА/ОПИ/4.2/ex2.py
r1 = 3 / 4
r2 = 5 / 6
r1 + r2 = 19 / 12
r1 - r2 = -1 / 12
r1 * r2 = 5 / 8
r1 / r2 = 9 / 10
r1 == r2: False
r1 != r2: True
r1 > r2: False
r1 < r2: True
r1 >= r2: False
r1 <= r2: True
```

Индивидуальное задание №1

Выполнить индивидуальное задание 1 лабораторной работы 4.1, максимально задействовав имеющиеся в Python средства перегрузки операторов.

```python
#!/usr/bin/env python 3
# -*- coding: utf-8 -*-


class Num:

    def __init__(self, first, second):
        self.first = first
        self.second = second
```

```
        if self.first == 0:
            raise ValueError

    def __pow__(self, other):
        a = self.first + self.second
        b = other.first + other.second
        return a ** b


if __name__ == "__main__":
    num1 = Num(3.4, 0)
    num2 = Num(2.7, 0)
    print(f"Результат: {num1 ** num2}")
```

```
C:\Users\79616\anaconda3\python.exe D:/УЧЕБА/ОПИ/4.2/ind1.py
Результат: 27.226579002152647
```

Индивидуальное задание №2

Дополнительно к требуемым в заданиях операциям перегрузить операцию индексирования []. Максимально возможный размер списка задать константой. В отдельном поле size должно храниться максимальное для данного объекта количество элементов списка; реализовать метод size(), возвращающий установленную длину. Если количество элементов списка изменяется во время работы, определить в классе поле count. Первоначальные значения size и count устанавливаются конструктором. В тех задачах, где возможно, реализовать конструктор инициализации строкой.

Создать класс BitString для работы с битовыми строками не более чем из 100 бит. Битовая строка должна быть представлена списком типа int, каждый элемент которого принимает значение 0 или 1. Реальный размер списка задается как аргумент конструктора инициализации. Должны быть реализованы все традиционные операции для работы с битовыми строками: and, or, xor, not. Реализовать сдвиг влево и сдвиг вправо на заданное количество битов.

```
#!/usr/bin/env python 3
# -*- coding: utf-8 -*-


class BitString:

    #Инициализация
    def __init__(self, x):
        self.size = x
        self.x = [0] * self.size

    #Установка значения
    def set(self, x):
        self.x = list(map(int, f'{x:b}'.rjust(self.size, '0')))
```

```
    #Оператор not (~)
    def __invert__(self):
        self.x = [int(not i) for i in self.x]
        return self

    #Оператор or (|)
    def __or__(self, other):
        x = [a | b for a, b in zip(self.x, other.x)]
        return ''.join(map(str, x))

    #Оператор xor (^)
    def __xor__(self, other):
        x = [a ^ b for a, b in zip(self.x, other.x)]
        return ''.join(map(str, x))

    #Оператор and (&)
    def __and__(self, other):
        x = [a & b for a, b in zip(other.x, self.x)]
        return ''.join(map(str, x))

    #Оператор сдвиг влево (<<)
    def __lshift__(self, x):
        del (self.x[0:x])
        self.x += [0] * x
        return self

    #Оператор сдвиг вправо (>>)
    def __rshift__(self, x):
        del (self.x[len(self.x) - x:])
        self.x = [0] * x + self.x
        return self

    #Вывод результата в консоль
    def __str__(self):
        return ''.join(map(str, self.x))


if __name__ == "__main__":
    x = BitString(8)  # Размер списка 1 - 8 бит
    y = BitString(8)  # Размер списка 2 - 8 бит

    x.set(55)  # Первая цифра 00110111
    print(x)
    y.set(27)  # Вторая цифра 00011011
    print(y)

    print(f'{x} and {y} = {x & y}')
    print(f'{x} or {y} = {x | y}')
    print(f'{x} xor {y} = {x ^ y}')
    print(f'{x} not = {~x}')
    print(f'{y} >> 1 = {y >> 1}')
    print(f'{x} << 2 = {x << 2}')
```

```
C:\Users\79616\anaconda3\python.exe D:/УЧЕБА/ОПИ/4.2/ind2.py
00110111
00011011
00110111 and 00011011 = 00010011
00110111 or 00011011 = 00111111
00110111 xor 00011011 = 00101100
00110111 not = 11001000
00011011 >> 1 = 00001101
11001000 << 2 = 00100000
```

ВОПРОСЫ

1. Какие средства существуют в Python для перегрузки операций? Заключение опреатора в двойное подчёркивание «___» с обеих сторон.

2. Какие существуют методы для перегрузки арифметических операцийи операций отношения в языке Python?

__sub___(self, other) - вычитание (x - y).

__mul___(self, other) - умножение (x * y).

__truediv___(self, other) - деление (x / y).

__floordiv___(self, other) - целочисленное деление (x // y).

__mod___(self, other) - остаток от деления (x % y).

__divmod___(self, other) - частное и остаток (divmod(x, y)).

__pow___(self, other[, modulo]) - возведение в степень (x ** y , pow(x, y[,modulo])).

__lshift___(self, other) - битовый сдвиг влево (x << y).

__rshift___(self, other) - битовый сдвиг вправо (x >> y).

__and___(self, other) - битовое И (x & y).

__xor___(self, other) - битовое ИСКЛЮЧАЮЩЕЕ ИЛИ (x ^ y).

__or___(self, other) - битовое ИЛИ (x | y).

__radd___(self, other) ,

__rsub___(self, other) ,

__rmul___(self, other) ,

__rtruediv___(self, other) ,

__rfloordiv___(self, other) ,

__rmod___(self, other) ,

__rdivmod___(self, other) ,

__rpow___(self, other) ,

__rlshift___(self, other) ,

__rrshift___(self, other) ,

__rand___(self, other) ,

__rxor___(self, other) ,

__ror___(self, other) - делают то же самое, что и арифметическиеоператоры, перечисленные выше, но для аргументов, находящихся справа, и только в случае, если для левого операнда не определён соответствующий метод.

3. В каких случаях будут вызваны следующие методы:____add___, __iadd__и_____radd____?

Например, операция x + y будет сначала пытаться вызвать x._add_(y) , и только в том случае, если это не получилось, будет пытаться вызватьy.____radd__(x) . Аналогично для остальных методов.

4. Для каких целей предназначен метод_new_? Чем он отличаетсяот метода_init_?

Он управляет созданием экземпляра. В качестве обязательного аргумента принимает класс (не путать с экземпляром). Должен возвращатьэкземпляр класса для его последующей его передачи методу_init_.

5. Чем отличаются методы____str___и___repr___?

__str___(self) - вызывается функциями str, print и format. Возвращаетстроковое представление объекта.

__repr___(self) - вызывается встроенной функцией repr; возвращает "сырые" данные, использующиеся для внутреннего представления в python.