

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

**Отчет по лабораторной работе № 15
«Перегрузка операторов в языке Python»**

по дисциплине «Основы программной инженерии»

Выполнила:

Первых Дарья Александровна,
2 курс, группа ПИЖ-б-о-20-1

Проверил:

Доцент кафедры инфокоммуникаций,
Воронкин Р.А.

Ставрополь, 2022 г.

ВЫПОЛНЕНИЕ

```
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
    return self

def __sub__(self, other):
    return Vector2D(self.x - other.x, self.y - other.y)

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

C:\Users\podar\study\anaconda\envs\15LR\python.exe "C:/Users/podar/
(3, 4)
5.0
(5, 6)
(8, 10)
(-2, -2)
(-3, -4)
(8, 10)
True

Рисунок 1 – Перегрузка операторов

```
def __gt__(self, rhs): # >
    if isinstance(rhs, Rational):
        return self.__float__() > rhs.__float__()
    else:
        return False

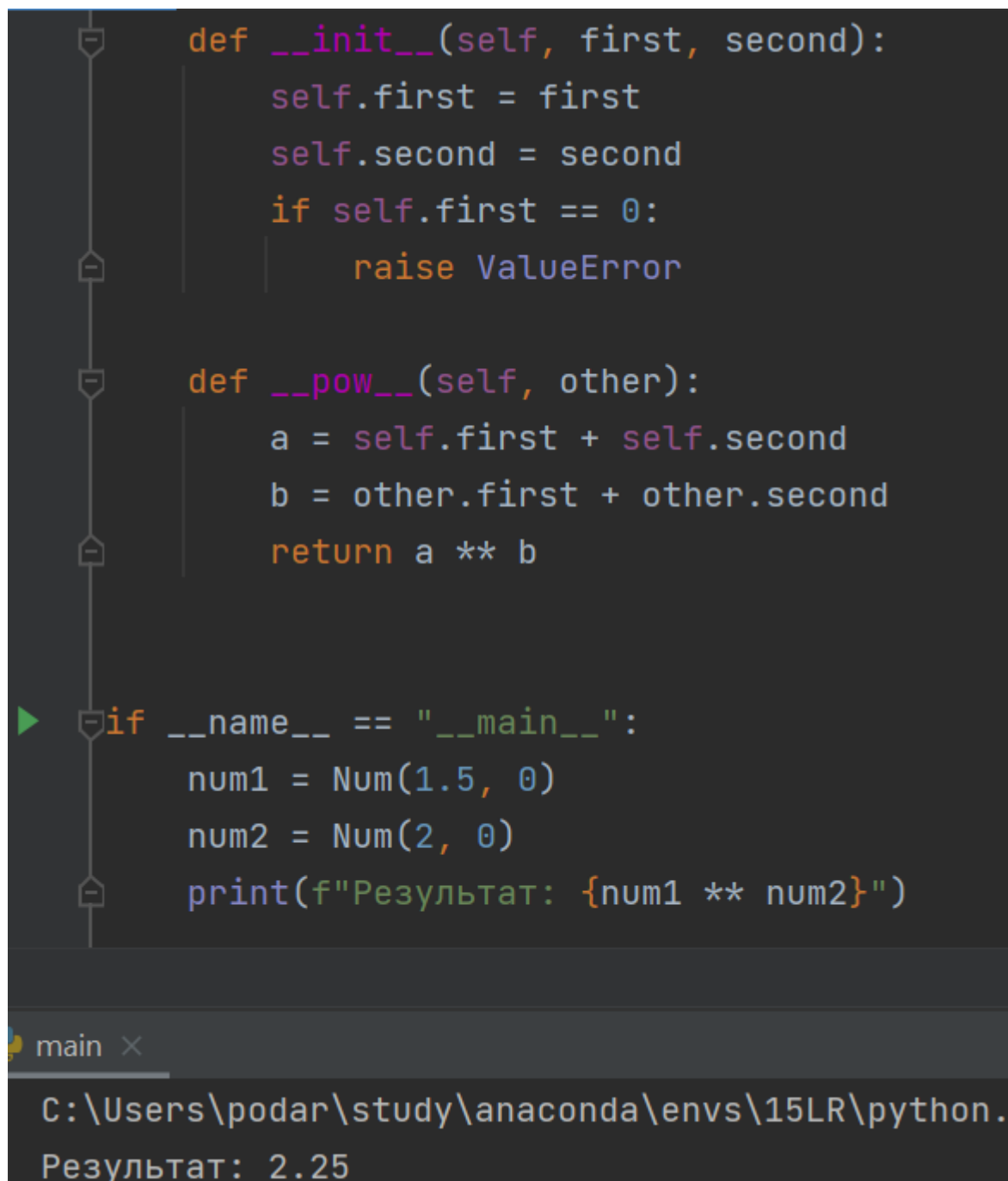
def __lt__(self, rhs): # <
    if isinstance(rhs, Rational):
        return self.__float__() < rhs.__float__()
    else:
        return False

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

C:\Users\podar\study\anaconda\envs\15LR\python.exe "C:/Users/p
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

Рисунок 2 – Перегрузка операторов

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



```
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(1.5, 0)
    num2 = Num(2, 0)
    print(f"Результат: {num1 ** num2}")
```

main x

C:\Users\podar\study\anaconda\envs\15LR\python.
Результат: 2.25

Рисунок 3 – Индивидуальное задание №1

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.

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.

```
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}')
```

```
if __name__ == "__main__"
```

```
main x
```

```
C:\Users\podar\study\anaconda\envs\15LR\python.exe
00110111
00011011
00110111 and 00011011 = 00010011
00110111 or 00011011 = 00111111
00110111 xor 00011011 = 00101100
00110111 not = 11001000
00011011 >> 1 = 00001101
11001000 << 2 = 00100000
```

Рисунок 4 – Индивидуальное задание №2

ВОПРОСЫ

1. Какие средства существуют в Python для перегрузки операций?

Заключение оператора в двойное подчёркивание «__» с обеих сторон.

2. Какие существуют методы для перегрузки арифметических операций и 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 ($x \wedge y$).

`__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 арифметические

операторы, 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 вызвать `y.__radd__(x)`. Аналогично для остальных методов.

4. Для каких целей предназначен метод `__new__`? Чем он отличается от метода `__init__`?

Он управляет созданием экземпляра. В качестве обязательного аргумента принимает класс (не путать с экземпляром). Должен возвращать экземпляр класса для его последующей его передачи методу `__init__`.

5. Чем отличаются методы `__str__` и `__repr__`?

`__str__(self)` - вызывается функциями `str`, `print` и `format`. Возвращает строковое представление объекта.

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