Перегрузка операторов

Мы можем легко найти минимальное число, сумму чисел или отсортировать последовательность:

```
>>> a = [3, 7, -1, 10]
>>> min(a)
-1
>>> sorted(a)
[-1, 3, 7, 10]
>>> sum(a)
19
```

Хочется, чтобы эти функции работали и для классов. Для этого нужно, чтобы классы умели сравнивать < (для min и sorted), и складывать через + (для sum).

Мы научились, чтобы `print` могла печатать экземпляры класса. Для этого мы в классе пишем функцию **`__str__(self)`**. Если есть класс A, и `x = A()` - экземпляр класса A, то при `print(x)`вызывается `str(x)`, которая вызывает `x.__str__()`
Можно переопределить в классе функции, чтобы работали операторы:

Специальная функция	Оператор	Значение
__lt__(self, other)	x < y	Возвращает True, если x меньше, чем y
__le__(self, other)	x <= y	Возвращает True, если x меньше или рав
__eq__(self, other)	x == y	Возвращает True, если x равно y
__ne__(self, other)	x != y	Возвращает True, если x НЕ равно y
__gt__(self, other)	x > y	Возвращает True, если x больше, чем y
__ge__(self, other)	x >= y	Возвращает True, если x больше или рав

Специальная функция	Оператор	Коментари
__add__(self, other)	x + y	
__sub__(self, other)	x - y	
__mul__(self, other)	x * y	

Специальная функция	Оператор	Коментари
__div__(self, other)	x / y	
__floordiv__(self, other)	x // y	
__mod__(self, other)	x % y	
__pow__(self, other)	x ** y	

Вспомним умножение строки на число 'hi'*3. Можно написать 3*'hi', получим такой же результат.

Для того, чтобы написать функцию число * строку, нужно переопределить для строки метод __rmul__ .

```
some_object + other
```
Вызывает __add__()

```
other + some_object
```
Вызывает __radd__(). У нее первый операнд other, а второй self.

Ее можно реализовать как:

```
def __radd__(self, other):
    return __add__(other, self)
```

Пример с точкой на плоскости XY

В классе Point (точка на плоскости XY) переопределим функции для == и для <

```
class Point(object):
    def __init__(self, x=0, y=0):
        self.x = x
        self.y = y

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

    def __eq__(self, other):
        return self.x == other.x and self.y == other.y

    def __lt__(self, other):
        """ Меньше та точка, у которой меньше x. При одинаковых x, та, у которой
меньше y."""
        if self.x == other.x:
            return self.y < other.y
        return self.x < other.x
```

```
    # другие функции класса: move, dir, dist...

# Тестируем функции класса:
def test():
    p0 = Point(3, 5)
    p1 = Point(3, 5)
    p2 = Point(-1, 7)
    p3 = Point(3, 1.17)

    print('p0=', p0)       # 3 5
    print('p1=', p1)       # 3 5
    print('p2=', p2)       # -1 7
    print('p3=', p3)       # 3 1.17

    print('p0 == p1', p0 == p1) # True
    print('p1 == p2', p1 == p2) # False

    print('p0 != p1', p0 != p1) # False
    print('p1 != p2', p1 != p2) # True

    print('p2 < p1', p2 < p1)   # True
    print('p1 < p2', p1 < p2)   # False

    print('p3 < p1', p3 < p1)   # True
    print('p1 < p3', p1 < p3)   # False

    a = [p0, p1, p2, p3]
    pmin = min(a)
    print('pmin =', pmin)       # -1 7

    b = sorted(a)
    print(b)    # [Point(-1, 7), Point(3, 1.17), Point(3, 5), Point(3, 5)]

test()
```

Задачи

Напишите класс Drob, который представляет дроби в виде *целых* числителя и знаменателя.

```
class Drob(object):
    """ Дробь вида a/b"""
    def __init__(self, a=0, b=1):
        self.a = a
        self.b = b
        self.normalize()

    def normalize(self):
        """ Приводит дробь вида 4/6 к 2/3"""

    def __str__(self):
        return '{}/{}'.format(self.a, self.b)
```

```
    # реализуйте функции
    # __eq__
    # __lt__
    # __add__
    # __sub__
    # __mul__
    # __div__
    # и проверьте каждую функцию
```

Задачи старые

min 1D отрезков

Допишите класс Segment1, чтобы можно было найти наименьший отрезок из данных.

Даны отрезки по 1 отрезку на строку. Напечатать самый короткий отрезок. Если несколько отрезков такой длины, то взять из них самый левый.

sort 1D отрезков

Допишите класс Segment1, чтобы можно было найти наименьший отрезок из данных.

Даны отрезки по 1 отрезку на строку. Напечатать самый короткий отрезок. Если несколько отрезков такой длины, то взять из них самый левый.

Обед-1 - сумма

Студент покупает обед в столовой. На вход дано что купил студент в формате: название rrr.kk rub По 1 товару на строку. Напечатать список товаров и сколько они стоят, отсортировать от дорогих к дешевым. Напечатать сколько стоит весь обед в формате rrr.kk

Для этого реализовать класс Rub и функции:

```
class Rub(object):
    """ Класс для работы с рублями и копейками."""
    def __init__(self, rub=0, kop=0):
        self.rub = rub
        self.kop = kop
        self.normalize()

    def __str__(self):
        # тут нужно написать код

    def __lt__(self, other):
        # тут нужно написать код
```

```
    def __add__(self, other):
        res = Rub()
        # тут нужно написать код
        return res

class Goods(object):
    """ Класс описания товара: название и цена"""
    def __init__(self, name='', rub=0, kop=0):
        self.name = name
        self.price = Rub(rub, kop)
```

Пример:

```
Input:
rice 10.50
tea 6.30
cake 10.12
salad 20.00
Output:
salad 20.00 rub
rice 10.50 rub
cake 10.12 rub
tea 6.30 rub
-----
total 46.92 rub
```

Обед-2 - сдача

В последней задаче после печати total нужно спросить сколько дал денег покупатель:

```
t = input('tender:')
```

Потом напечатать сколько надо дать сдачи (change)

Пример:

```
# Input:
rice 10.50
tea 6.30
cake 10.12
salad 20.00
#Output:
salad 20.00 rub
rice 10.50 rub
cake 10.12 rub
tea 6.30 rub
-----
total 46.92 rub
# Input:
tender 100
# Output:
tender 100.00 rub
change 53.08 rub
```

Обед-3 - цена за 1 кг и вес

Некоторые товары могут быть заданы как цена за 1 кг и вес в кг.

Их нужно напечатать в чеке в следующем формате:

```
# Input
rice 43.00 0.5
apple 63 0.127
# Output
rice 43.00x0.5 = 21.50
apple 63.00x0.127 = 8.00
```

При округлении части копеек лишнее - отбросить.

63.00x0.127 = 8.001, но мы округлили (отбросили лишнее) до 8.00

1.239 округляем до 1.23