**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 А, и x = A() - экземпляр класса А, 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 х меньше, чем у |
| \_\_le\_\_(self, other) | x <= y | Возвращает True, если х меньше или равно у |
| \_\_eq\_\_(self, other) | x == y | Возвращает True, если х равно у |
| \_\_ne\_\_(self, other) | x != y | Возвращает True, если х НЕ равно у |
| \_\_gt\_\_(self, other) | x > y | Возвращает True, если х больше, чем у |
| \_\_ge\_\_(self, other) | x >= y | Возвращает True, если х больше или равно у |

| **Специальная функция** | **Оператор** | **Коментарий** |
| --- | --- | --- |
| \_\_add\_\_(self, other) | x + y |  |
| \_\_sub\_\_(self, other) | x - y |  |
| \_\_mul\_\_(self, other) | x \* y |  |
| \_\_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 ХУ) 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 х. 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