Методы класса и статические методы

Статические методы

Когда мы реализовали класс Drob, нам нужна была функция, которая считает НОД для 2 чисел, чтобы сокращать дробь.

Это полезная функция. Хочется хранить ее внутри класса.

Но это не метод 1 экземпляра класса (одной дроби), он общий для всех дробей.

Этот метод не зависит ни от одной переменной ни в дроби, ни во всем классе дробей.

Такие методы называют статическими методами класса и пишут с `@staticmethod` перед определением метода.
Если статических методов в классе несколько, то у каждого нужно написать `@staticmethod`
Добавим в класс Drob статический метод nod.

Статический метод не относится к конкретной дроби. Он для всех дробей. У него **нет self**.

Статический метод не может написать self.переменная или self.функция, потому что у него нет self

Обращение к статической функции:

- изнутри объекта класса: self или название класса
- снаружи: название класса

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

    @staticmethod
    def nod(x, y):      # нет self
        res = x % y
        while res > 1:
            x, y = y, res
            res = x % y
        return y

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

```
        k = Drob.nod(self.a, self.b)        # можно self.nod
        self.a //= k
        self.b //= k

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

    # реализация функций __eq__, __lt__, __add__, __sub__, __mul__,

# конец класса
d1 = Drob(4, 6)
print(d1)               # 2/3

z = Drob.nod(123, 21)   # класс.статический_метод
print(z)                # 3
```

Еще один пример. В классе Segment1 отрезков по оси X можно написать функцию пересечения отрезка с другим отрезком как функцию экземпляра класса crossed_with или как функцию не относящуюся ни к одному экземпляру (общую для всех экземпляров) is_crossed.

```
class Segment1(object):
    """Класс Segment1 описывает отрезки на оси X"""

    def __init__(self, start=0, finish=0):
        # Эта функция вызывается, когда мы создаем новый объект класса.
        # self - это название переменной, которая указвает на сам объект.
        self.start = start      # переменная объекта
        self.finish = finish

    def __str__(self):
        return '{} {}'.format(self.start, self.finish)

    def crossed_with(self, other):      # метод экземпляра класса
        """ Пересекается этот отрезок self с другим отрезком other? """
        if self.finish < other.start or other.finish < self.start:
            return False
        return True

    @staticmethod
    def is_crossed(seg1, seg2):         # статический  метод класса (нет self)
        """ Пересекаются ли отрезки seg1 и seg2? """
        if seg1.finish < seg2.start or seg2.finish < seg1.start:
            return False
        return True

# вызываем метод экземпляра класса и статический метод класса:
s1 = Segment1(2, 5)
s2 = Segment1(3, 7)

print(s1.crossed_with(s2))          # метод экземпляра класса
print(Segment1.is_crossed(s1, s2))  # статический метод класса
```

Переменные класса

Пусть мы делаем класс окружность. В каждой окружности есть радиус, (x, y) - координаты центра окружности. У окружности можно посчитать радиус и периметр. Для их вычсления нужно число пи (3.14...)

Вопрос - к какой именно окружности принадлежит число пи? Ни к какой. Оно характеризует все окружности сразу.

Значит, число пи должно храниться не в переменных экземпляра класса (1 окружность), а в переменных всего класса (объект класса).

Переменные класса пишутся или вне методов в классе, или внутри метода как *класс.переменная*

Посчитаем, сколько экземпляров класса Drob создали за время работы программы. Можно создать глобальную переменну counter. Но лучше сделать ее переменной класса.

```
class Segment1(object):
    """Класс Segment1 описывает отрезки на оси X"""

    counter = 0 # еще не было создано ни одной окружности

    def __init__(self, start=0, finish=0):
        # Эта функция вызывается, когда мы создаем новый объект класса.
        # self - это название переменной, которая указвает на сам объект.
        self.start = start      # переменная объекта
        self.finish = finish
        Segment1.counter += 1

    @classmethod
    def how_many(cls):
        return cls.counter

# конец класса
s1 = Segment1(2, 5)
s2 = Segment1(3, 7)

print(Segment1.how_many())          # вызов метода класса по имени класса
print(s1.how_many())                # вызов метода класса по имени класса
```

Переменная `counter` - одна на весь класс. Общая. Каждый экземпляр класса (1 дробь) может в нее писать по ссылке **Segment1.counter**
Чем отличается `@staticmethod` от `@classmethod`?
В методе класса how_many:

- есть аргумент **cls** (класс Drob). Если **self** означает 1 дробь (объект с полями и методами), то **cls**- объект класса (один на класс Drob, описывает этот класс).
- Обращаться к полям и методам класса можно или по имени класса Drob.counter или по переменной cls.counter;
- Вызывается этот метод по имени класса или ссылки на экземпляр класса. Лучше всегда обращайтесь к переменной по имени класса.

Делаем экземпляр класса

Напишем функцию read(line), которая будет из строки делать отрезок.

Хотим, чтобы `Segment1.read('2, 5')` создал и вернул `Segment1(2, 5)`
Функция read всего класса, а не 1 отрезка. Значит, это или staticmethod, или classmethod.

Если функция всего класса возвращает экземпляр класса, делайте ее classmethod(Позже объясним зачем, когда будем изучать наследование классов).

```
class Segment1(object):
    ...
    @classmethod
    def read(cls, line):
        start, finish = map(int, line.split(','))
        t = Segment1(start, finish)
        return t
        # или сразу return Segment1(start, finish) - можно обойтись без переменной
t
```

Если есть ссылка на объект всего класса, можно заменить вызов `Segment1(..)` на `cls(..)`

```
class Segment1(object):
    ...
    @classmethod
    def read(cls, line):
        start, finish = map(int, line.split(','))
        return cls(start, finish)                    # cls - ссылка на весь класс
```

Итого

```
class A(object):
    class_var = 2

    def __init__(self, x):
        self.x = x

    @classmethod
    def foo(cls):
        ...
    @staticmethod
    def bzz():
        ...

a = A()
A.class_var = 33
A.foo()
A.bzz()
```

- Класс - тоже объект. У класса могут быть переменные и методы. Класс A, переменная class_var, метод foo.
- переменная класса пишется вне всех функций внутри класса; class_var
- обращаются к переменной класса по *класс.переменная*; A.class_var
- метод класса не работает с 1 экземпляром класса, у него нет self. `def foo(cls):`
- метод класса работает с объектом класса, на него ссылаются через *cls* ;
- конструктор можно вызвать через _cls(аргументы) `cls(7)`
- к методу класса можно обратиться через имя класса `A.foo()`
- если метод не для 1 дроби, а для всех и ему не нужны переменные класса, то делаем `@staticmethod`

Задачи

Drob

Drob.nod, Drob.read

В класс Drob дописать статический метод nod и метод класса read.

Drob - калькулятор 2 дробей

Реализовать калькуляр дробей, который вычисляет сумму и разность дробей, заданных по формату:

```
1/3
+
1/6
=
```

Нужно напечатать ответ: `1/2`

Drob - сумматор многих дробей

Как в предыдущей задаче. Только дробей может быть не 2, а больше. Только + и -.

Drob - калькулятор без скобок

Напишите калькулятор, который считает дроби.

Нужно 2 стека. Стек дробей stack_drob и стек операций (+, -, *) stack_op.

Пусть у каждой операции будет приоритет.

prior = {'+': 1, '-': 1, '*':2, '=':0} # символ операции : приоритет операции

Положить в стек можно функцию `append` работы с list, а взять функцией `pop()`. Сначала положить в стек операций =. (Кладем кортеж ('=', 0) - символ и приоритет. Такие пары хранятся в стеке операций. Как считать выражение:

- если дробь, положить в стек дробей.
- если операция:
  - если в стеке операция с меньшим приоритетом, положить нашу операцию в стек операций;
  - иначе пока подходящие операции не закончатся,
    - достать операцию из стека операций;
    - достать 2 дроби из стека дробей;
    - посчитать результат "дробь операция дробь" и положить его в стек дробей.

В конце должен быть пустой стек операций и 1 дробь в стеке дробоей - результат работы калькулятора.

Кнаты, сикли, галеоны

У волшебников свои деньги. Это галеоны (galeon), сикли (sicle), кнаты (knut).

- 1 галеон = 17 сиклей
- 1 сикль = 29 кнатов

Написать класс WMoney, который хранит сумму в кнатах, а печатает по формату 1 galeon 5 sicle 10 knut.

В классе написать метод read(str), который из строки '1 galeon 5 sicle 10 knut' делает WMoney(galeon=1, sicle=5, knut=10)

Timer - дописать

Кнаты, сикли, галеоны

Чек в магическом магазине со сдачей.

Автомат по продаже билетов

Переменные класса - список допустимых монет.

Электричка на Марсе

Модифицировать задачу дополнения расписания, при условии, что планета не Земля, и там другое количество минут в 1 часе и часов в сутках (может дробное???