**Методы класса (classmethod) и статические методы (staticmethod)**

[Смотреть материал на видео](https://www.youtube.com/watch?v=78PTvj2wYH8&list=PLA0M1Bcd0w8zPwP7t-FgwONhZOHt9rz9E" \t "_blank)

Мы продолжаем тему методов в ООП. До сих пор мы с вами определяли методы просто как функции внутри класса, например:

**class** Vector:

**def** \_\_init\_\_(self, x, y):

        self.x = x

        self.y = y

**def** get\_coord(self):

**return** self.x, self.y

И у каждого такого метода, как правило, первым идет параметр self – ссылка на экземпляр класса, из которого метод был вызван:

v = Vector(10, 20)

coord = v.get\_coord()

**print**(coord)

В данном случае, при вызове метода get\_coord () параметр self будет вести на объект v класса Vector. Об этом мы с вами уже говорили. Также отмечали, что при вызове такого метода напрямую из класса нужно явно указывать первый аргумент self:

coord2 = Vector.get\_coord(v)

Так вот, в Python помимо таких «стандартных» методов можно задавать методы уровня класса и статические методы с помощью встроенных декораторов:

@classmethod и @staticmethod

Давайте я поясню на простом примере, что они значат. Добавим в наш класс Vector два атрибута:

**class** Vector:

    MIN\_COORD = 0

    MAX\_COORD = 100

...

А также метод класса:

    @classmethod

**def** validate(cls, arg):

**return** cls.MIN\_COORD <= arg <= cls.MAX\_COORD

который проверяет, попадает ли значение arg в диапазон [MIN\_COORD; MAX\_COORD]. Обратите внимание, у методов класса (когда мы используем декоратор classmethod) первым параметром идет cls – ссылка на класс, а не self – ссылка на объект класса. Это означает, что данный метод может обращаться только к атрибутам текущего класса, но не к локальным свойствам его экземпляров. Мало того, этот метод можно теперь напрямую вызывать из класса, не передавая ссылку на экземпляр, как это было при вызове обычных методов через класс:

res = Vector.validate(5)

**print**(res)

Здесь пользователь класса Vector может совершенно спокойно вызывать метод validate(), не создавая никаких объектов. Но «платой» за это является ограниченность метода: он может работать только с атрибутами класса, но не объекта, что, в общем то, естественно, так как у него изначально нет ссылки на объект. Во всем остальном этот метод работает абсолютно также, как и любой другой метод, объявленный в классе.

Давайте мы им воспользуемся и вызовем внутри класса для проверки корректности координат x, y:

**def** \_\_init\_\_(self, x, y):

        self.x = self.y = 0

**if** Vector.validate(x) **and** Vector.validate(y):

            self.x = x

            self.y = y

Обратите внимание, мы здесь обращаемся к методу класса через пространство имен Vector. Но также можем прописать и self:

**if** self.validate(x) **and** self.validate(y):

В этом случае интерпретатор Python сам подставит нужный класс в параметр cls данного метода, так как экземпляр содержит информацию о классе, от которого был образован.

Наконец, третий тип методов – статические методы, определяются декоратором @staticmethod. Это методы, которые не имеют доступа ни к атрибутам класса, ни к атрибутам его экземпляров, то есть, некая независимая, самостоятельная функция, объявленная внутри класса. Обычно, это делают для удобства, т.к. их функционал так или иначе связан с тематикой класса.

Например, в нашем классе Vector можно объявить такой статический метод, который бы вычислял квадратичную норму вектора (длину вектора в квадрате):

    @staticmethod

**def** norm2(x, y):

**return** x\*x + y\*y

Здесь нет никаких скрытых параметров, которые бы автоматически заполнялись интерпретатором языка. Только те, что мы прописываем сами. Я указал два параметра x, y, по которым вычисляется квадрат длины радиус-вектора. То есть, это некая вспомогательная, сервисная функция, связанная с векторами, для вычисления квадратичной нормы любого радиус-вектора. Воспользоваться этим методом можно как вне класса:

res = Vector.norm2(5, 6)

Так и внутри класса:

**def** \_\_init\_\_(self, x, y):

        self.x = self.y = 0

**if** self.validate(x) **and** self.validate(y):

            self.x = x

            self.y = y

**print**(Vector.norm2(self.x, self.y))

Либо, также обратиться к этому методу через self:

**print**(self.norm2(self.x, self.y))

Подведем итог различных типов методов в классах. Обычные методы, как правило, вызываются из экземпляров классов и работают с атрибутами экземпляров и атрибутами классов. Методы классов обычно вызываются через класс, реже через его экземпляры и имеют доступ только к атрибутам самого класса, в котором объявлены. Наконец, статические методы – это совершенно изолированные функции, которые работают только с параметрами, прописанными в ней самой и не имеют доступа к атрибутам класса или его экземпляров.

Поэтому, если вам нужен метод, который работает с атрибутами объектов класса, то это обычное определение функций внутри класса с первым параметром self. Если метод работает только с атрибутами класса, то возможно, имеет смысл его определить как метод класса и тогда можно будет вызывать без ссылки на объект этого класса. Третий тип, статические методы часто определяют как вспомогательные, сервисные, связанные с логикой работы самого класса.

Вот общее руководство по выбору этих методов. Надеюсь, из этого занятия вам стало понятно, что такое методы класса и статические методы, а также для чего имеет смысл их использовать.