**Тема занятия №21: Объектно-ориентированное программирование**

Согласно модели данных Python, язык предлагает три вида методов: статические, класса и экземпляра класса. Давайте посмотрим, что же происходит за кулисами каждого из видов методов. Понимание принципов их работы поможет в создании красивого и эффективного кода. Начнём с самого простого примера, в котором демонстрируются все три вида методов.

**class ToyClass:**

**def instancemethod(self):**

**return 'instance method called', self**

**@classmethod**

**def classmethod(cls):**

**return 'class method called', cls**

**@staticmethod**

**def staticmethod():**

**return 'static method called'**

Ранее было сказано, с определенным допущением классы можно рассматривать как модули, содержащие переменные со значениями и функции. Только здесь переменные называются полями или свойствами, а функции – методами. Вместе поля и методы называются атрибутами.

Однако в случае классов, когда метод применяется к объекту, этот экземпляр передается в метод в качестве первого аргумента:

**>>> class A:**

**... def meth(self):**

**... print('meth')**

**...**

**>>> a = A()**

**>>> a.meth()**

**meth**

**>>> A.meth(a)**

**meth**

Вызов a.meth() на самом деле преобразуется к A.meth(a), то есть мы идем к "модулю A" и в его пространстве имен ищем атрибут meth. Там оказывается, что meth это функция, принимающая один обязательный аргумент. Тогда ничего не мешает сделать так:

**>>> b = 10**

**>>> A.meth(b)**

**meth**

В таком "модульном формате" вызова методов передавать объект-экземпляр именно класса A совсем не обязательно. Однако нельзя сделать так:

**>>> b = 10**

**>>> b.meth()**

**Traceback (most recent call last):**

**File "<stdin>", line 1, in <module>**

**AttributeError: 'int' object**

**has no attribute 'meth'**

Если объект передается методу в нотации через точку, то этот метод должен быть описан в том классе, которому принадлежит объект, или в родительских классах. В данном случае у класса int нет метода meth(). Объект b классу A не принадлежит. Поэтому интерпретатор никогда не найдет метод meth().

Что делать, если возникает необходимость в методе, который не принимал бы объект данного класса в качестве аргумента? Да, мы можем объявить метод вообще без параметров и вызывать его только через класс:

**>>> class A:**

**... def meth():**

**... print('meth')**

**...**

**>>> A.meth()**

**meth**

**>>> a = A()**

**>>> a.meth()**

**Traceback (most recent call last):**

**File "<stdin>", line 1, in <module>**

**TypeError: meth() takes 0 positional**

**arguments but 1 was given**

Получается странная ситуация. Ведь meth() вызывается не только через класса, но и через порожденные от него объекты. Однако в последнем случае всегда будет возникать ошибка. То есть имеется потенциально ошибочный код. Кроме того, может понадобиться метод с параметрами, но которому не надо передавать экземпляр данного класса.

В ряде языков программирования, например в Java, для таких ситуаций предназначены статические методы. При описании этих методов в их заголовке ставится ключевое слово static. Такие методы могут вызываться через объекты данного класса, но сам объект в качестве аргумента в них не передается.

В Python острой необходимости в статических методах нет, так как код может находиться за пределами класса, и программа не начинает выполняться из класса. Если нам нужна просто какая-нибудь функция, мы можем определить ее в основной ветке. В Java это не так. Там, не считая импортов, весь код находится внутри классов. Поэтому методы, не принимающие объект данного класса и играющие роль обычных функций, необходимы. Статические методы решают эту проблему.

Однако в Python тоже можно реализовать подобное, то есть статические методы, с помощью декоратора @staticmethod:

**>>> class A:**

**... @staticmethod**

**... def meth():**

**... print('meth')**

**...**

**>>> a = A()**

**>>> a.meth()**

**meth**

**>>> A.meth()**

**meth**

**Пример с параметром:**

**>>> class A:**

**... @staticmethod**

**... def meth(value):**

**... print(value)**

**...**

**>>> a = A()**

**>>> a.meth(1)**

**1**

**>>> A.meth('hello')**

**Hello**

Методы класса

Методы класса принимают класс в качестве параметра, который принято обозначать как cls. Он указывает на класс ToyClass, а не на объект этого класса. При декларации методов этого вида используется декоратор classmethod.

Методы класса привязаны к самому классу, а не его экземпляру. Они могут менять состояние класса, что отразится на всех объектах этого класса, но не могут менять конкретный объект.

Встроенный пример метода класса — dict.fromkeys()— возвращает новый словарь с переданными элементами в качестве ключей.

**dict.fromkeys('AEIOU') # <- вызывается при помощи класса dict**

**{'A': None, 'E': None, 'I': None, 'O': None, 'U': None}**

Статические методы

Статические методы декларируются при помощи декоратора staticmethod. Им не нужен определённый первый аргумент (ни self, ни cls).

Их можно воспринимать как методы, которые “не знают, к какому классу относятся”.

Таким образом, статические методы прикреплены к классу лишь для удобства и не могут менять состояние ни класса, ни его экземпляра.

Статические методы в Python – по-сути обычные функции, помещенные в класс для удобства и находящиеся в пространстве имен этого класса. Это может быть какой-то вспомогательный код. Вообще, если в теле метода не используется self, то есть ссылка на конкретный объект, следует задуматься, чтобы сделать метод статическим. Если такой метод необходим только для обеспечения внутренних механизмов работы класса, то возможно его не только надо объявить статическим, но и скрыть от доступа из вне.

Пусть у нас будет класс "Цилиндр". При создании объектов от этого класса у них заводятся поля высота и диаметр, а также площадь поверхности. Вычисление площади можно поместить в отдельную статическую функцию. Она вроде и относится к цилиндрам, но, с другой стороны, само вычисление объекта не требует и может быть использовано где угодно.

**from math import pi**

**class Cylinder:**

**@staticmethod**

**def make\_area(d, h):**

**circle = pi \* d \*\* 2 / 4**

**side = pi \* d \* h**

**return round(circle\*2 + side, 2)**

**def \_\_init\_\_(self, di, hi):**

**self.dia = diameter**

**self.h = high**

**self.area = self.make\_area(di, hi)**

**a = Cylinder(1, 2)**

**print(a.area)**

**print(a.make\_area(2, 2))**

В примере вызов make\_area() за пределами класса возможен в том числе через экземпляр. При этом понятно, в данном случае свойство area самого объекта a не меняется. Мы просто вызываем функцию, находящуюся в пространстве имен класса.

Рассмотрим следующий пример:

**from datetime import date**

**class Person:**

**def \_\_init\_\_(self, name, age):**

**self.name = name**

**self.age = age**

**@classmethod**

**def from\_birth\_year(cls, name, year):**

**return cls(name, date.today().year - year)**

**@staticmethod**

**def is\_adult(age):**

**return age > 18**

**person1 = Person('Sarah', 25)**

**person2 = Person.from\_birth\_year('Roark', 1994)**

**>>> person1.name, person1.age**

**Sarah 25**

**>>> person2.name, person2.age**

**Roark 24**

**>>> Person.is\_adult(25)**

**True**

Рассмотрим следующий пример:

**class Class1(object):**

**@staticmethod**

**def sum1(x, y): # Статический метод**

**return x + y**

**def sum2(self, x, y): # Обычный метод в классе**

**return x + y**

**def sum3(self, x, y):**

**return Class1.sum1(x, y) # Вызов из метода класса**

**print Class1.sum1(10, 20) # Вызываем статический метод**

**c1 = Class1()**

**print c1.sum2(15, 6) # Вызываем метод класса**

**print c1.sum1(50, 12) # Вызываем статический метод**

**# через экземпляр класса**

**print c1.sum3(23, 5) # Вызываем статический метод**

**# внутри класса**

Когда использовать каждый из методов?

Выбор того, какой из методов использовать, может показаться достаточно сложным. Тем не менее с опытом этот выбор делать гораздо проще.

Чаще всего метод класса используется тогда, когда нужен генерирующий метод, возвращающий объект класса. Как видим, метод класса from\_birth\_year используется для создания объекта класса Person по году рождения, а не возрасту.

Статические методы в основном используются как вспомогательные функции и работают с данными, которые им передаются.

Запомнить следующее:

Методы экземпляра класса получают доступ к объекту класса через параметр self и к классу через self.\_\_class\_\_.

Методы класса не могут получить доступ к определённому объекту класса, но имеют доступ к самому классу через cls.

Статические методы работают как обычные функции, но принадлежат области имён класса. Они не имеют доступа ни к самому классу, ни к его экземплярам.