Python classmethod()

programiz.com/python-programming/methods/built-in/classmethod

Join our newsletter for the latest updates.

The classmethod() method returns a class method for the given function.

The syntax of `classmethod()` method is:

```
classmethod(function)
```

`classmethod()` is considered un-Pythonic so in newer Python versions, you can use the `@classmethod` decorator for classmethod definition.

The syntax is:

```
@classmethod
def func(cls, args...)
```

classmethod() Parameters

`classmethod()` method takes a single parameter:

function - Function that needs to be converted into a class method

Return value from classmethod()

`classmethod()` method returns a class method for the given function.

What is a class method?

A class method is a method that is bound to a class rather than its object. It doesn't require creation of a class instance, much like staticmethod.

The difference between a static method and a class method is:

- Static method knows nothing about the class and just deals with the parameters
- Class method works with the class since its parameter is always the class itself.

The class method can be called both by the class and its object.

```
Class.classmethod()
Or even
Class().classmethod()
```

But no matter what, the class method is always attached to a class with the first argument as the class itself *cls*.

```
def classMethod(cls, args...)
```

Example 1: Create class method using classmethod()

```
class Person:
    age = 25

    def printAge(cls):
        print('The age is:', cls.age)

# create printAge class method
Person.printAge = classmethod(Person.printAge)

Person.printAge()
```

Output

```
The age is: 25
```

Here, we have a class `Person`, with a member variable *age* assigned to 25.

We also have a function `printAge` that takes a single parameter *cls* and not `self` we usually take.

cls accepts the class `Person` as a parameter rather than Person's object/instance.

Now, we pass the method `Person.printAge` as an argument to the function `classmethod`. This converts the method to a class method so that it accepts the first parameter as a class (i.e. Person).

In the final line, we call `printAge` without creating a Person object like we do for static methods. This prints the class variable *age*.

When do you use class method?

1. Factory methods

Factory methods are those methods that return a class object (like constructor) for different use cases.

It is similar to function overloading in C++. Since, Python doesn't have anything as such, class methods and static methods are used.

Example 2: Create factory method using class method

```
from datetime import date

# random Person
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    @classmethod
    def fromBirthYear(cls, name, birthYear):
        return cls(name, date.today().year - birthYear)

    def display(self):
        print(self.name + "'s age is: " + str(self.age))

person = Person('Adam', 19)
person.display()

person1 = Person.fromBirthYear('John',  1985)
person1.display()
```

Output

```
Adam's age is: 19
John's age is: 31
```

Here, we have two class instance creator, a constructor and a `fromBirthYear` method.

The constructor takes normal parameters *name* and *age*. While, `fromBirthYear` takes *class*, *name* and *birthYear*, calculates the current age by subtracting it with the current year and returns the class instance.

The fromBirthYear method takes Person class (not Person object) as the first parameter *cls* and returns the constructor by calling `cls(name, date.today().year - birthYear)`, which is equivalent to `Person(name, date.today().year - birthYear)`

Before the method, we see `@classmethod`. This is called a decorator for converting `fromBirthYear` to a class method as `classmethod()`.

---

2. Correct instance creation in inheritance

Whenever you derive a class from implementing a factory method as a class method, it ensures correct instance creation of the derived class.

You can create a static method for the above example but the object it creates, will always be hardcoded as Base class.

But, when you use a class method, it creates the correct instance of the derived class.

Example 3: How the class method works for the inheritance?

```
from datetime import date

# random Person
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    @staticmethod
    def fromFathersAge(name, fatherAge, fatherPersonAgeDiff):
        return Person(name, date.today().year - fatherAge + fatherPersonAgeDiff)

    @classmethod
    def fromBirthYear(cls, name, birthYear):
        return cls(name, date.today().year - birthYear)

    def display(self):
        print(self.name + "'s age is: " + str(self.age))

class Man(Person):
    sex = 'Male'

man = Man.fromBirthYear('John', 1985)
print(isinstance(man, Man))

man1 = Man.fromFathersAge('John', 1965, 20)
print(isinstance(man1, Man))
```

Output

```
True
False
```

Here, using a static method to create a class instance wants us to hardcode the instance type during creation.

This clearly causes a problem when inheriting `Person` to `Man`.

`fromFathersAge` method doesn't return a `Man` object but its base class `Person`'s object.

This violates OOP paradigm. Using a class method as `fromBirthYear` can ensure the OOP-ness of the code since it takes the first parameter as the class itself and calls its factory method.