Classmethod () function in Python

In Python, the classmethod () function returns a class method for the function. How is the syntax of classmethod () function, what parameters does it have and how to use it? Invites you to read the track.

In Python, **the classmethod () function** returns a class method for the function. How is the syntax of *classmethod ()* function, what parameters does it have and how to use it? Invites you to read the track.

The classmethod () function syntax in Python

```
classmethod(function)
```

classmethod () is considered un-Pythonic (Python's non-formal language), so in newer versions of Python, you should use *decor class @classmethod* to determine the method

The syntax is as follows:

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

Parameters of classmethod ()

1. *classmethod ()* has only one parameter, *function* - the function that needs to be passed into the classmethod

Value returned from classmethod ()

By definition, *classmethod ()* returns a class method for the given function.

So what is the class method here?

Class method is a method belonging to the whole class. When executed, it does not use any instance of that class, quite similar to staticmethod.

However there are a few differences between *static methods* and *class methods* like:

1. *The static method* does not use anything related to the class or instance of that class but only works with parameters.
2. With the *class method* , the whole class will be passed into the first parameter (cls) of this method.
3. *The method class* can be called from both its class and object.

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

The method class is always attached to the class because it implicitly assigns the class to the first parameter (cls) when calling the function.

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

Example 1: Create class method (class method) using classmethod ()

```
class Nhanvien:
 tuoi = 25

 def printTuoi(cls):
 print('S? tu?i là:', cls.tuoi)

# t?o ph??ng th?c class printTuoi
Nhanvien.printTuoi = classmethod(Nhanvien.printTuoi)

Nhanvien.printTuoi()
```

Here, we have a *Nhanvien* class , with member age variable *(tuoi)* assigned to 25.

We also have a *printTuoi* function *that* takes an *uncommon cls* parameter.

cls accepts the Nhanvien class as a parameter instead of object / instance of *Nhanvien.*

Now, we pass the method *Nhanvien.print.* As the argument to the classmethod function. This converts the method to a class method so that it accepts the first parameter as a class (ie *Nhanvien).*

In the last line, we call *print,* but not create a *Nhanvien* object as in the static method.

Running the above code, the program will return the result:

```
S? tu?i là: 25
```

When to use class method?

1. Factory method

Factory methods are methods that return an object of the class in different ways.

Like loading functions in C ++, but Python is not available so here we use class methods and static methods.

Example 2: Create factory method using class method

```
from datetime import date

# random Nhanvien
```

```
class Nhanvien:
 def __init__(self, ten, tuoi):
 self.ten = ten
 self.tuoi = tuoi

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

 def ketqua(self):
 print("Tu?i c?a " + self.ten + " là: " + str(self.tuoi))

nhanvien = Nhanvien('Alice', 23)
nhanvien.ketqua()

nhanvien1 = Nhanvien.fromBirthYear('Simon', 1990)
nhanvien1.ketqua()
```

The program returns the result:

```
Tu?i c?a Alice là: 23
Tu?i c?a Simon là: 28
```

Here we have two class instances, one created, one class method *fromBirthYear*.

Instance is created with the name parameter *(ten)* and age *(tuoi), fromBirthYear* takes information from class, *ten* and *birthYear* and then calculates the current age using the current year subtracting the *birthYear* and returning the class result instance.

The *fromBirthYear* method *takes* information from the *Nhanvien* class (not the *Nhanvien* object *)* as parameter *cls* and returns a function as *cls (ten, date.today (). Year - birthYear)* equivalent to *Nhanvien (ten, date.today) () .year - birthYear*).
This is thanks to the *@classmethod* decorator . This decorator itself transforms the *fromBirthYear* method of the method class with *classmethod ().*

2. Create the correct instance in Inheritance (inheritance).

Inheritance (inheritance) is the reuse of properties and functions of a class to define a new class. New class created is called subclass (child class or derived class), inherited class is called parent class (base class or parent class).

If you derive a class from the method factory creation using classmethod, it definitely creates the correct object in the subclass.

You can do the same but use staticmethod, the object created is definitely in the parent class.

Example 3: How the class method works in 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))
```

The program returns the result:

```
True
False
```

This example uses a static method to create a class with a hard-fixed data type during creation.

This caused a problem when inheriting *Man* from *Person.*

The method *fromFathersAge* does not return an object in *Man* but returns an object in *Person* class - parent class.

This violates the OOP model. Using the method class *fromBirthYear* can guarantee the OOP model of the code because it takes the first parameter as the class itself is passed into its same method method.

See more:

1. Built-in Python functions
2. Python function parameter
3. Recursive function in Python

Previous article: Function chr () in Python

Next lesson: complex () function in Python