What is the difference between @staticmethod and @classmethod in Python?

What is the difference between a function decorated with `@staticmethod` and one decorated with `@classmethod` ?

python

edited Aug 5 '16 at 22:30
Mark Amery
30k 17 138 174

asked Sep 25 '08 at 21:01
Daryl Spitzer
35.9k 54 132 155

18 Answers

Maybe a bit of example code will help: Notice the difference in the call signatures of `foo` , `class_foo` and `static_foo` :

```
class A(object):
    def foo(self,x):
        print "executing foo(%s,%s)"%(self,x)

    @classmethod
    def class_foo(cls,x):
        print "executing class_foo(%s,%s)"%(cls,x)

    @staticmethod
    def static_foo(x):
        print "executing static_foo(%s)"%x

a=A()
```

Below is the usual way an object instance calls a method. The object instance, `a` , is implicitly passed as the first argument.

```
a.foo(1)
# executing foo(<__main__.A object at 0xb7dbef0c>,1)
```

With classmethods, the class of the object instance is implicitly passed as the first argument instead of `self` .

```
a.class_foo(1)
# executing class_foo(<class '__main__.A'>,1)
```

You can also call `class_foo` using the class. In fact, if you define something to be a classmethod, it is probably because you intend to call it from the class rather than from a class instance. `A.foo(1)` would have raised a TypeError, but `A.class_foo(1)` works just fine:

```
A.class_foo(1)
# executing class_foo(<class '__main__.A'>,1)
```

One use people have found for class methods is to create inheritable alternative constructors.

With staticmethods, neither `self` (the object instance) nor `cls` (the class) is implicitly passed as the first argument. They behave like plain functions except that you can call them from an instance or the class:

```
a.static_foo(1)
# executing static_foo(1)

A.static_foo('hi')
# executing static_foo(hi)
```

Staticmethods are used to group functions which have some logical connection with a class to the class.

`foo` is just a function, but when you call `a.foo` you don't just get the function, you get a "partially applied" version of the function with the object instance `a` bound as the first argument to the function. `foo` expects 2 arguments, while `a.foo` only expects 1 argument.

`a` is bound to `foo`. That is what is meant by the term "bound" below:

```
print(a.foo)
# <bound method A.foo of <__main__.A object at 0xb7d52f0c>>
```

With `a.class_foo`, `a` is not bound to `class_foo`, rather the class `A` is bound to `class_foo`.

```
print(a.class_foo)
# <bound method type.class_foo of <class '__main__.A'>>
```

Here, with a staticmethod, even though it is a method, `a.static_foo` just returns a good 'ole function with no arguments bound. `static_foo` expects 1 argument, and `a.static_foo` expects 1 argument too.

```
print(a.static_foo)
# <function static_foo at 0xb7d479cc>
```

And of course the same thing happens when you call `static_foo` with the class `A` instead.

```
print(A.static_foo)
# <function static_foo at 0xb7d479cc>
```

edited Feb 9 '16 at 20:20
Rafay
2,988 5 32 58

answered Nov 3 '09 at 19:13
unutbu
413k 55 791 904

59 I don't understand what's the catch for using staticmethod. we can just use a simple outside-of-class function. – Alcott Sep 19 '11 at 3:08

147 @Alcott: You might want to move a function into a class because it logically belongs with the class. In the Python source code (e.g. multiprocessing,turtle,dist-packages), it is used to "hide" single-underscore "private" functions from the module namespace. Its use, though, is highly concentrated in just a few modules -- perhaps an indication that it is mainly a stylistic thing. Though I could not find any example of this, `@staticmethod` might help organize your code by being overridable by subclasses. Without it you'd have variants of the function floating around in the module namespace. – unutbu Sep 19 '11 at 10:34

2 ... along with some explanation on *where* and *why* to use either instance, class or static methods. You didn't give a single word about it, but neither did the OP asked about it. – MestreLion May 3 '12 at 9:50

45 @Alcott: as unutbu said, static methods are an organization/stylistic feature. Sometimes a module have many classes, and some helper functions are logically tied to a a given class and not to the others, so it makes sense not to "pollute" the module with many "free functions", and it is better to use a static method than relying on the poor style of mixing classes and function defs together in code just to show they are "related" – MestreLion May 3 '12 at 9:55

17 @unutbu: You should however delete that Guido quote: It was from an early draft of python 2.2 release notes, and it was changed in 2.2.3 to: `However, class methods are still useful in other places, for example, to program inheritable alternate constructors.` Even the original quote is very misleading, as he only said that the python distribuition does not use `classmethod`. But just because python itself don't use a given feature, it doesn't mean it's a useless feature. Think about **complex numbers** or even **sqrt**: python itself may not use either, but it's far from being useless to offer – MestreLion May 3 '12 at 10:44

A staticmethod is a method that knows nothing about the class or instance it was called on. It just gets the arguments that were passed, no implicit first argument. It is basically useless in Python -- you can just use a module function instead of a staticmethod.

A classmethod, on the other hand, is a method that gets passed the class it was called on, or the class of the instance it was called on, as first argument. This is useful when you want the method to be a factory for the class: since it gets the actual class it was called on as first argument, you can always instantiate the right class, even when subclasses are involved. Observe for instance how `dict.fromkeys()` , a classmethod, returns an instance of the subclass when called on a subclass:

```
>>> class DictSubclass(dict):
...     def __repr__(self):
...         return "DictSubclass"
...
>>> dict.fromkeys("abc")
{'a': None, 'c': None, 'b': None}
>>> DictSubclass.fromkeys("abc")
DictSubclass
>>>
```

edited Jul 27 '10 at 0:09   answered Sep 25 '08 at 21:05
Thomas Wouters
67k 15 110 105

516 A staticmethod isn't useless - it's a way of putting a function into a class (because it logically belongs there), while indicating that it does not require access to the class. – Tony Meyer Sep 26 '08 at 10:10

99 Hence only 'basically' useless. Such organization, as well as dependency injection, are valid uses of staticmethods, but since modules, not classes like in Java, are the basic elements of code organization in Python, their use and usefulness is rare. – Thomas Wouters Sep 26 '08 at 13:40

29 What's logical about defining a method inside a class, when it has nothing to do with either the class or its instances? – Ben James Mar 12 '10 at 9:11

80 Perhaps for the inheritance sake? Static methods can be inherited and overridden just like instance methods and class methods and the lookup works as expected (unlike in Java). Static methods are not really resolved statically whether called on the class or instance, so the only difference between class and static methods is the implicit first argument. – haridsv Apr 8 '10 at 1:32

59 They also create a cleaner namespace, and makes it easier to understand the function have something to do with the class. – Imbrondir Aug 22 '11 at 11:58

Basically `@classmethod` makes a method whose first argument is the class it's called from (rather than the class instance), `@staticmethod` does not have any implicit arguments.

edited Oct 8 '12 at 2:07   answered Sep 25 '08 at 21:07
Tadeck   Terence Simpson
65.9k 13 99 156   **1,732** 11 15

Official python docs:

@classmethod

> A class method receives the class as implicit first argument, just like an instance method receives the instance. To declare a class method, use this idiom:
>
> ```
> class C:
>     @classmethod
>     def f(cls, arg1, arg2, ...): ...
> ```
>
> The `@classmethod` form is a function *decorator* – see the description of function definitions in *Function definitions* for details.
>
> It can be called either on the class (such as `C.f()`) or on an instance (such as `C().f()`). The instance is ignored except for its class. If a class method is called for a derived class, the derived class object is passed as the implied first argument.
>
> Class methods are different than C++ or Java static methods. If you want those, see `staticmethod()` in this section.

@staticmethod

> A static method does not receive an implicit first argument. To declare a static method, use this idiom:

```
class C:
    @staticmethod
    def f(arg1, arg2, ...): ...
```

The `@staticmethod` form is a function *decorator* – see the description of function definitions in *Function definitions* for details.

It can be called either on the class (such as `C.f()`) or on an instance (such as `C().f()`). The instance is ignored except for its class.

Static methods in Python are similar to those found in Java or C++. For a more advanced concept, see `classmethod()` in this section.

edited Dec 11 '12 at 9:56
me_and
9,108 2 38 78

answered Nov 3 '09 at 19:23
Chris B.
1,634 11 9

Here is a short article on this question

> @staticmethod function is nothing more than a function defined inside a class. It is callable without instantiating the class first. It's definition is immutable via inheritance.
>
> @classmethod function also callable without instantiating the class, but its definition follows Sub class, not Parent class, via inheritance. That's because the first argument for @classmethod function must always be cls (class).

answered Nov 3 '09 at 19:02
Tom Neyland
4,756 2 24 47

1 So does that mean that by using a staticmethod I am always bound to the Parent class and with the classmethod I am bound the class that I declare the classmethod in (in this case the sub class)? – Mohan Gulati Nov 3 '09 at 19:06

5 No. By using a staticmethod you aren't bound at all; there is no implicit first parameter. By using classmethod, you get as implicit first parameter the class you called the method on (if you called it directly on a class), or the class of the instance you called the method on (if you called it on an instance). – Matt Anderson Nov 3 '09 at 19:18

4 +1: a brief, simple, but solid explanation. Could be expanded a bit to show that, by having a class as a first argument, class methods have direct access to other class attributes and methods, while static methods do not (they would need to hardcode MyClass.attr for that) – MestreLion May 3 '12 at 10:01

@decorators were added in python 2.4 If you're using python < 2.4 you can use the classmethod() and staticmethod() function.

For example, if you want to create a factory method (A function returning an instance of a different implementation of a class depending on what argument it gets) you can do something like:

```
class Cluster(object):

    def _is_cluster_for(cls, name):
        """
        see if this class is the cluster with this name
        this is a classmethod
        """
        return cls.__name__ == name
    _is_cluster_for = classmethod(_is_cluster_for)

    #static method
    def getCluster(name):
        """
        static factory method, should be in Cluster class
        returns a cluster object for the given name
        """
        for cls in Cluster.__subclasses__():
            if cls._is_cluster_for(name):
                return cls()
    getCluster = staticmethod(getCluster)
```

Also observe that this is a good example for using a classmethod and a static method, The static method clearly belongs to the class, since it uses the class Cluster internally. The classmethod only needs information about the class, and no instance of the object.

Another benefit of making the `_is_cluster_for` method a classmethod is so a subclass can decide to change it's implementation, maybe because it is pretty generic and can handle more than one type of cluster, so just checking the name of the class would not be enough.

edited Feb 20 at 15:34 answered Feb 24 '12 at 9:32

Jens Timmerman
3,316 1 18 31

> **What is the difference between @staticmethod and @classmethod in Python?**

You may have seen Python code like this pseudocode, which demonstrates the signatures of the various method types and provides a docstring to explain each:

```
class Foo(object):

    def a_normal_method(self, arg_1, kwarg_2=None):
        '''
        Return a value that is a function of the instance with its
        attributes, and other arguments such as arg_1 and kwarg2
        '''

    @staticmethod
    def a_static_method(arg_0):
        '''
        Return a value that is a function of arg_0. It does not know the
        instance or class it is called from.
        '''

    @classmethod
    def a_class_method(cls, arg1):
        '''
        Return a value that is a function of the class and other arguments.
        respects subclassing, it is called with the class it is called from.
        '''
```

The Normal Method

First I'll explain the `normal_method` . This may be better called an "**instance method**". When an instance method is used, it is used as a partial function (as opposed to a total function, defined for all values when viewed in source code) that is, when used, the first of the arguments is predefined as the instance of the object, with all of its given attributes. It has the instance of the object bound to it, and it must be called from an instance of the object. Typically, it will access various attributes of the instance.

For example, this is an instance of a string:

```
', '
```

if we use the instance method, `join` on this string, to join another iterable, it quite obviously is a function of the instance, in addition to being a function of the iterable list, `['a', 'b', 'c']` :

```
>>>', '.join(['a', 'b', 'c'])
'a, b, c'
```

Static Method

The static method does *not* take the instance as an argument. Yes it is very similar to a module level function. However, a module level function must live in the module and be specially imported to other places where it is used. If it is attached to the object, however, it will follow the object conveniently through importing and inheritance as well.

An example is the `str.maketrans` static method, moved from the `string` module in Python 3. It makes a translation table suitable for consumption by `str.translate` . It does seem rather silly when used from an instance of a string, as demonstrated below, but importing the function from the `string` module is rather clumsy, and it's nice to be able to call it from the class, as in `str.maketrans`

```
# demonstrate same function whether called from instance or not:
>>> ', '.maketrans('ABC', 'abc')
{65: 97, 66: 98, 67: 99}
>>> str.maketrans('ABC', 'abc')
{65: 97, 66: 98, 67: 99}
```

In python 2, you have to import this function from the increasingly deprecated string module:

```
>>> import string
>>> 'ABCDEFG'.translate(string.maketrans('ABC', 'abc'))
'abcDEFG'
```

Class Method

A class method is a similar to a static method in that it takes an implicit first argument, but instead of taking the instance, it takes the class. Frequently these are used as alternative constructors for better semantic usage and it will support inheritance.

The most canonical example of a builtin classmethod is `dict.fromkeys`. It is used as an alternative constructor of dict, (well suited for when you know what your keys are and want a default value for them.)

```
>>> dict.fromkeys(['a', 'b', 'c'])
{'c': None, 'b': None, 'a': None}
```

When we subclass dict, we can use the same constructor, which creates an instance of the subclass.

```
>>> class MyDict(dict): 'A dict subclass, use to demo classmethods'
>>> md = MyDict.fromkeys(['a', 'b', 'c'])
>>> md
{'a': None, 'c': None, 'b': None}
>>> type(md)
<class '__main__.MyDict'>
```

See the pandas source code for other similar examples of alternative constructors, and see also the official Python documentation on `classmethod` and `staticmethod`.

edited Feb 18 '16 at 3:54

answered Jan 23 '15 at 20:01

Aaron Hall ♦
73.8k 21 174 167

I think a better question is "When would you use @classmethod vs @staticmethod?"

@classmethod allows you easy access to private members that are associated to the class definition. this is a great way to do singletons, or factory classes that control the number of instances of the created objects exist.

@staticmethod provides marginal performance gains, but I have yet to see a productive use of a static method within a class that couldn't be achieved as a standalone function outside the class.

answered May 19 '15 at 15:27

Nathan Tregillus
2,733 1 22 38

`@staticmethod` just disables the default function as method descriptor. classmethod wraps your function in a container callable that passes a reference to the owning class as first argument:

```
>>> class C(object):
...  pass
...
>>> def f():
...  pass
...
>>> staticmethod(f).__get__(None, C)
<function f at 0x5c1cf0>
>>> classmethod(f).__get__(None, C)
<bound method type.f of <class '__main__.C'>>
```

As a matter of fact, `classmethod` has a runtime overhead but makes it possible to access the owning class. Alternatively I recommend using a metaclass and putting the class methods on that metaclass:

```
>>> class CMeta(type):
...  def foo(cls):
...   print cls
...
>>> class C(object):
...  __metaclass__ = CMeta
...
>>> C.foo()
<class '__main__.C'>
```

answered Sep 25 '08 at 21:24

Armin Ronacher
23.7k 9 58 64

3 What's the problem with a metaclass? – Armin Ronacher Sep 26 '08 at 3:20

8 What are the advantages to using a metaclass for this? – Daryl Spitzer Jan 18 '09 at 17:51

One possible downside of a metaclass for this that immediately occurs to me is that you can't call the classmethod directly on an instance. `c = C(); c.foo()` raises AttributeError, you'd have to do `type(c).foo()` . This might also be considered a feature - I can't think of why you'd want to though. – Aaron Hall ♦ Jun 14 '16 at 21:57

To decide whether to use @staticmethod or @classmethod you have to look inside your method. **If your method accesses other variables/methods in your class then use @classmethod**. On the other hand if your method does not touch any other parts of the class then use @staticmethod.

```
class Apple:

    _counter = 0

    @staticmethod
    def about_apple():
        print('Apple is good for you.')

        # note you can still access other member of the class
        # but you have to use the class instance
        # which is not very nice, because you have repeat yourself
        #
        # For example:
        # @staticmethod
        #    print('Number of apples have been juiced: %s' % Apple._counter)
        # @classmethod
        #    it is especially useful when you move this code to other classes,
        #       you don't have to rename Apple class name
        #    print('Number of apples have been juiced: %s' % cls._counter)

    @classmethod
    def make_apple_juice(cls, number_of_apples):
        print('Make juice:')
        for i in range(number_of_apples):
            cls._juice_this(i)

    @classmethod
    def _juice_this(cls, apple):
        print('Juicing %d...' % apple)
        cls._counter += 1
```

edited Jan 10 at 13:27 answered Apr 22 '16 at 15:40

Du D.
1,541 7 24

Pity that your static method example actually **does** access a class variable. You could improve your example, maybe. – Cilyan Jul 1 '16 at 17:25

@classmethod means: when this method is called, we pass the class as the first argument instead of the instance of that class (as we normally do with methods). This means you can use the class and its properties inside that method rather than a particular instance.

@staticmethod means: when this method is called, we don't pass an instance of the class to it (as we normally do with methods). This means you can put a function inside a class but you can't access the instance of that class (this is useful when your method does not use the instance).

answered Dec 13 '15 at 19:37

Tushar.PUCSD
625 9 13

Here is one good link for this topic, and summary it as following.

`@staticmethod` function is nothing more than a function defined inside a class. It is callable without instantiating the class first. It’s definition is immutable via inheritance.

- Python does not have to instantiate a bound-method for object.
- It eases the readability of the code, and it does not depend on the state of object itself;

`@classmethod` function also callable without instantiating the class, but its definition follows Sub class, not Parent class, via inheritance, can be overridden by subclass. That’s because the first argument for `@classmethod` function must always be *cls* (class).

- *Factory methods*, that are used to create an instance for a class using for example some sort of pre-processing.
- *Static methods calling static methods*: if you split a static methods in several static methods, you shouldn't hard-code the class name but use class methods

answered Nov 16 '15 at 2:00

zangw
14k 6 48 71

Static Methods:

- Simple functions with no self argument.
- Work on class attributes; not on instance attributes.
- Can be called through both class and instance.
- The built-in function staticmethod()is used to create them.

Benefits of Static Methods:

- It localizes the function name in the classscope
- It moves the function code closer to where it is used

```
@staticmethod
def some_static_method(*args, **kwds):
pass
```

Class Methods:

- Functions that have first argument as classname.
- Can be called through both class and instance.
- These are created with classmethod in-built function.

```
@classmethod
def some_class_method(cls, *args, **kwds):
pass
```

edited Oct 3 '16 at 10:47

answered Oct 3 '16 at 10:41

Preya
703 5 17

Another consideration with respect to staticmethod vs classmethod comes up with inheritance. Say you have the following class:

```
class Foo(object):
    @staticmethod
    def bar():
        return "In Foo"
```

And you then want to override `bar()` in a child class:

```
class Foo2(Foo):
    @staticmethod
    def bar():
        return "In Foo2"
```

This works, but note that now the `bar()` implementation in the child class (`Foo2`) can no longer take advantage of anything specific to that class. For example, say `Foo2` had a method called `magic()` that you want to use in the `Foo2` implementation of `bar()` :

```
class Foo2(Foo):
    @staticmethod
    def bar():
        return "In Foo2"
    @staticmethod
    def magic():
        return "Something useful you'd like to use in bar, but now can't"
```

The workaround here would be to call `Foo2.magic()` in `bar()` , but then you're repeating yourself (if the name of `Foo2` changes, you'll have to remember to update that `bar()` method).

To me, this is a slight violation of the open/closed principle, since a decision made in `Foo` is impacting your ability to refactor common code in a derived class (ie it's less open to extension). If `bar()` were a `classmethod` we'd be fine:

```
class Foo(object):
    @classmethod
    def bar(cls):
        return "In Foo"

class Foo2(Foo):
    @classmethod
    def bar(cls):
        return "In Foo2 " + cls.magic()
    @classmethod
    def magic(cls):
        return "MAGIC"

print Foo2().bar()
```

Gives: `In Foo2 MAGIC`

edited Oct 4 '16 at 16:09

answered Sep 29 '16 at 17:02

Adam Parkin
5,223 8 30 65

In Python, a classmethod receives a class as the implicit first argument. The class of the object instance is implicitly passed as the first argument. This can be useful when one wants the method to be a factory of the class as it gets the actual class (which called the method) as the first argument, one can instantiate the right class, even if subclasses are also concerned.

A staticmethod is just a function defined inside a class. It does not know anything about the class or instance it was called on and only gets the arguments that were passed without any implicit first argument. Example:

```
class Test(object):
    def foo(self, a):
        print "testing (%s,%s)"%(self,a)

    @classmethod
    def foo_classmethod(cls, a):
        print "testing foo_classmethod(%s,%s)"%(cls,a)

    @staticmethod
    def foo_staticmethod(a):
        print "testing foo_staticmethod(%s)"%a

test = Test()
```

staticmethods are used to group functions which have some logical connection with a class to the class.

answered Jul 6 '16 at 8:53

Green Harry
81 1 4

I will try to explain the basic difference using an example.

```
class A(object):
    x = 0

    def say_hi(self):
        pass

    @staticmethod
    def say_hi_static():
        pass

    @classmethod
    def say_hi_class(cls):
        pass

    def run_self(self):
        self.x += 1
        print self.x # outputs 1
        self.say_hi()
        self.say_hi_static()
        self.say_hi_class()

    @staticmethod
    def run_static():
        print A.x  # outputs 0
        # A.say_hi() #  wrong
        A.say_hi_static()
        A.say_hi_class()

    @classmethod
    def run_class(cls):
        print cls.x # outputs 0
        # cls.say_hi() #  wrong
        cls.say_hi_static()
        cls.say_hi_class()
```

1 - we can directly call static and classmethods without initializing

```
# A.run_self() #  wrong
A.run_static()
A.run_class()
```

2- Static method cannot call self method but can call other static and classmethod

3- Static method belong to class and will not use object at all.

4- Class method are not bound to an object but to a class.

answered Sep 20 '16 at 9:03
Rizwan Mumtaz
1,596 1 13 21

```
#!/usr/bin/python
#coding:utf-8

class Demo(object):
    def __init__(self,x):
        self.x = x

    @classmethod
    def addone(self, x):
        return x+1

    @staticmethod
    def addtwo(x):
        return x+2

    def addthree(self, x):
        return x+3

def main():
    print Demo.addone(2)
    print Demo.addtwo(2)

    #print Demo.addthree(2) #Error
    demo = Demo(2)
    print demo.addthree(2)


if __name__ == '__main__':
    main()
```

answered Nov 18 '13 at 5:53
Zuckonit
379 2 14

A quick hack-up ofotherwise identical methods in iPython reveals that `@staticmethod` yields marginal performance gains (in the nanoseconds), but otherwise it seems to serve no function. Also, any performance gains will probably be wiped out by the additional work of processing the method through `staticmethod()` during compilation (which happens prior to any code execution when you run a script).

For the sake of code readability I'd avoid `@staticmethod` unless your method will be used for loads of work, where the nanoseconds count.

edited Dec 17 '12 at 19:10
Keith Pinson
3,825 5 32 70

answered Dec 17 '12 at 18:51
Cathal Garvey
7

5 "Otherwise seems to serve no function": not strictly true. See above discussion. – Keith Pinson Dec 17 '12 at 19:46

---

protected by Jon Clements ♦ Jan 7 '13 at 9:36

Thank you for your interest in this question. Because it has attracted low-quality or spam answers that had to be removed, posting an answer now requires 10 reputation on this site (the association bonus does not count).

Would you like to answer one of these unanswered questions instead?