What is the difference between a static method and class method in Python?

Python 中静态方法和类方法的区别是什么？

问题原文：

Python has two decorators @staticmethod and @classmethod. Could someone one explain the difference between the two or point me to a source where it is documented? When should the staticmethod decorator be used over the classmethod decorator?

问题译文：

Python 有两种装饰器静态方法装饰器@staticmethod 和类方法装饰器@classmethod。谁能解释一下两者的区别或者给我指点点文档资料？什么时候该用静态方法装饰器或者类方法装饰器？

Answers：

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

可能一点例子代码会有帮助：注意调用函数 foo、 class_foo、static_foo 的写法上的区别

```
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.

下面是对象实例调用方法的通常做法。对象实例 A，隐式传递给方法的第一个参数。（译者加：也就是将实例 a 对象传递给 foo 方法的 self 参数）。

```
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.

类方法的方式，类的对象实例作为隐式的参数传递给第一个参数而不是 self。（译者加：也就是将类 A 传递给对象实例的 class_foo 方法的 cls 参数）

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

You can even 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.

你甚至可以通过类调用 class_foo 方法。其实，如果你定义类方法内容，可能你就是想通过类调用它而不是通过对象实例。A.foo(1)会引发类型错误异常（TypeError）。

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

With staticmethods, neither self (the object instance) nor cls (the class) is implicitly passed as the first argument.

而静态方法，既不需要 self（对象实例）也不需要 cls（类）作为隐藏参数被传递。

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

foo is just a function, but when you call a.foo you don't just get the function, you get a "curried" 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:

foo 只是个函数，但是当你调用 a.foo 时你不是简单的调用函数，实际上你调用了一个加工过的函数版本：绑定对象实例 a 作为第一个参数的函数。foo 需要 2 个参数，而 a.foo 只需要一个函数（译者加：对象实例 a 自动传递给了第一个参数 self 嘛）

a 绑定到了 foo。这就是为什么下面的终端输出"bound"的原因。

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

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

而 a.class_foo，对象实例 a 是没有绑定到 foo 的，而是类 A 绑定到 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.

这里，静态方法，虽然是一个方法，但是 a.static_foo 只是一个没有绑定任何参数的完好的函数。static_foo 需要 1 个参数，同样 a.static_foo 也只需要一个参数。

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