解释一：

```
class X:
  def f(self, a, b):
```

self 是类方法的一个位置参数，它就是类的实例对象自己，当实例调用方法时：
instance = X()
instance.f('a', 'b')
等同于：
X.f(instance, 'a', 'b')
第一个参数是实例自己。

解释二：

Python 要 self 的理由

Python 的类的方法和普通的函数有一个很明显的区别，在类的方法必须有个额外的第一个参数（self ），但在调用这个方法的时候不必为这个参数赋值 （显胜于隐 的引发）。Python 的类的方法的这个特别的参数指代的是对象本身，而按照 Python 的惯例，它用 self 来表示。（当然我们也可以用其他任何名称来代替，只是规范和标准在那建议我们一致使用 self）

为何 Python 给 self 赋值而你不必给 self 赋值？

例子说明：创建了一个类 MyClass，实例化 MyClass 得到了 MyObject 这个对象，然后调用这个对象的方法 MyObject.method(arg1,arg2) ，这个过程中，Python 会自动转为 Myclass.mehod(MyObject,arg1,arg2)

这就是 Python 的 self 的原理了。即使你的类的方法不需要任何参数，但还是得给这个方法定义一个 self 参数，虽然我们在实例化调用的时候不用理会这个参数不用给它赋值

我解释一下 python 的类方法为什么要写一个 self 参数

这是对前面一个 php 程序员问 python 方法为什么要手写一个 self 的回答，当时那个帖非常的热闹，但是下面没有一个回复讲到要点，等我有空，已经找不到原帖了。

原因有多重。首先是 python 中几乎所有的东西的一级对象（一级对象的定义：http://en.wikipedia.org/wiki/First-class_object），method 也不例外，比如你写一个:

```
class X:
  def f(self, a, b):
```

...

那么可以这样引用 f：

X.__dict__['f']

或者

X.f.__func__

现在问题来了，得到 f 以后怎么调用？f 是一个方法，方法必须作用于对象。如果 x 是一个 X 对象，我们可以 x.f(...)，但是如果是以上面的方式得到的 f 呢？怎么 f 让作用于某个对象？最直观的方法就是和参数一起传递进去。

当然的限不同的设计也可以满足上面制。比如不要手写的 self 参数，增加 this 关键字，增加一个调用 f 个格式。这样的设计和 pep20 第二条不符合。

为理解 python 的 self，不妨对比一下 ruby 和 javascript。ruby 的方式是方法不是一级对象，所以绕过了这个问题。javascript 没有类，方法在调用时 this 会绑定到方法所属的对象，函数调用时 this 绑定到 window，函数做构造器调用时 this 绑定到新对象。这么多规则，比 Python 复杂。

self 是类方法的一个位置参数，它就是类的实例对象自己，当实例调用方法时：

instance = X()

instance.f('a', 'b')

等同于：

X.f(instance, 'a', 'b')

第一个参数是实例自己。