实例方法 静态方法与类方法

In [9]:
```
#示例代码1: 实例方法

class c1(object):
    kinds = 'Person'  ##定义一个类属性
    def __init__(self, name="lily"):
        self.name = name

    def test1(self):  #定义一个实例方法
        print('test1 My kinds is %s' %c1.kinds)     #实例方法可以通过类名.属性名访问类属性
        print('test1 My kinds is %s'%self.kinds)    #实例方法可以通过self.属性名访问实例属性
        print('My name is %s' %self.name)
        print('这是实例方法!!')
```

什么是实例方法?

实例方法一般被类所生成的实例调用, 和其他面向对象语言的实例方法使用的时候差不多, 在定义实例方法的时候无论实例方法内部逻辑是否需要使用参数, 形参列表第一个参数都需要定义, 名字为self. 为什么第一个参数一定要写因为这个self代表了类实例化出的各个对象.

实例方法的访问方式:

```
x = c1()     #通过c1类 实例化出一个x对象
x.test1()    #通过实例x访问实例方法 test1
c1.test1(x) #通过类来调用实例方法, 需要传入一个类的实例
```

结论: 实例方法可以被实例调用, 也可以被类调用, 但是被类调用时. 需要传入一个类的实例, 实例方法可以通过self访问实例属性.

In [11]:
```
x = c1()
x.test1()
print('#'* 10)
c1.test1(x)
```

```
test1 My kinds is Person
test1 My kinds is Person
My name is lily
这是实例方法!!
##########
test1 My kinds is Person
test1 My kinds is Person
My name is lily
这是实例方法!!
```

In [16]:
```
#示例代码2: 静态方法

class c1(object):
    kinds = 'Person'  ##定义一个类属性
    def __init__(self, name="lily"):
        self.name = name

    @staticmethod
    def test2():
        print('这是静态方法!!')
        print('test2 My kind is %s' %c1.kinds) #静态方法可以使用 类名.属性名 访问类属性
        print('My name is %s' %self.name ) #此处代码报错, 静态方法不可以使用实例属性.
```

什么是静态方法?
静态方法是调用时跟具体实例没有关联的方法, 形参的第一个参数不是self或cls, 它属于类本身, 而不属于类的实例. 但静态方法虽为类所有, 但它既可以通过类的实例对象来使用, 也可以通过类来使用. 但是推荐使用类名.静态方法名的方式来调用静态方法, 静态方法是个独立的、单纯的函数, 它仅仅托管于某个类的名称空间中, 便于使用和维护.

静态方法的访问方式:

```
x = c1()   #通过c1类 实例化出一个x对象
c1.test2() #通过c1类来调用静态方法
x.test2()  #通过c1类的实例来调用静态方法
```

注意:
静态方法不能访问实例属性

什么情况使用静态方法?
一个方法不需要访问对象状态, 其所需参数都是通过显式参数提供时, 可以使用静态方法.

In [17]:
```
x = c1()
c1.test2()
```

```
这是静态方法!!
test2 My kind is Person

---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-17-e11d7c3cb433> in <module>
      1 x = c1()
----> 2 c1.test2()

<ipython-input-16-fd73c39a57e2> in test2()
     10         print('这是静态方法!!')
     11         print('test2 My kind is %s' %c1.kinds)
---> 12         print('My name is %s' %self.name )
     13

NameError: name 'self' is not defined
```

In [18]:
```
#示例代码3:类方法

class c1(object):
    kinds = 'Person'  ##定义一个类属性
    def __init__(self, name):
        self.name = name

    @classmethod
    def test3(cls):
        print('这是类方法')
        print('My kinds is %s' %cls.kinds) #类方法可以访问类属性, 使用cls.类属性的方式
        print('My name is %s' %self.name)  #此处代码报错, 类方法不可以访问实例属性
```

什么是类方法?
类方法不是绑定到对象上, 而是绑定在类上的方法, 它更关注于从类中调用方法, 而不是从实例中调用方法. 类方法是将类本身作为对象进行操作的方法, 假设有个方法, 且这个方法在逻辑上采用类本身作为对象来调用更合理, 那么这个方法就可以定义为类方法.

类方法的访问方式:

```
x = c1('lily')  #通过c1类 实例化出一个x对象
c1.test3()      #通过c1类来调用类方法
x.test3()       #通过c1类的实例来调用类方法
```

结论：类方法可以被实例调用，也可以被类调用．在其他编程语言如JAVA中，类方法是不可以被实例调用的．

注意：
类方法不能访问实例属性．
类方法可以通过cls访问类属性．

In [28]:
```
x = c1("lily")
x.test3()
```

```
My kinds is Person

---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-28-782b5f13773f> in <module>
      1 x = c1("lily")
----> 2 x.test3()

<ipython-input-18-260c6e9e0416> in test3(cls)
      9     def test3(cls):
     10         print('My kinds is %s' %cls.kinds)
---> 11         print('My name is %s' %self.name)
     12         print('这是类方法')

NameError: name 'self' is not defined
```

In [29]:
```
c1.test3()
```

```
My kinds is Person

---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-29-9292b7b51795> in <module>
----> 1 c1.test3()

<ipython-input-18-260c6e9e0416> in test3(cls)
      9     def test3(cls):
     10         print('My kinds is %s' %cls.kinds)
---> 11         print('My name is %s' %self.name)
     12         print('这是类方法')

NameError: name 'self' is not defined
```

#示例代码4：类代码应用举例
有一个班级类，两个类方法为添加学生数量和返回学生人数，学生类是班级类的子类，每实例化一个学生，班级的人数就加1．

In [34]:
```
class c1:
    student_num = 0
    @classmethod
    def add_member(cls):
        cls.student_num += 1
    @classmethod
    def get_member(cls):
        return "班级的总人数为：%s" %cls.student_num

class Student(c1):
    def __init__(self, name):
        self.name = name
        c1.add_member()

lily = Student("lily")
print(c1.get_member())

lucy = Student("lucy")
print(c1.get_member())
```

```
班级的总人数为：1
班级的总人数为：2
```

In [ ]: