[**python中的实例方法、静态方法、类方法、类变量和实例变量**](http://www.cnblogs.com/kex1n/p/5979366.html)

class MyTest:  
  
    myname = 'peter'  
  
    # add a instance attribute  
    def \_\_init\_\_(self, name):  
        self.name = name  
  
    # class access class attribute  
    def sayhello(self):  
        print "say hello to %s" % MyTest.myname  
  
    # instance can access class attribute  
    def sayhello\_1(self):  
        print "say hello to %s" % self.myname  
  
    # It's a snap! instance can access instance attribute  
    def sayhello\_2(self):  
        print "say hello to %s" % self.name  
  
    # class can not access instance attribute!!!  
    def sayhello\_3(self):  
        #print "say hello to %s" % MyTest.name  
        pass  
  
if \_\_name\_\_ == '\_\_main\_\_':  
  
    test = MyTest("abc")  
    test.sayhello()  
    test.sayhello\_1()  
    test.sayhello\_2()  
    test.sayhello\_3()  
  
    # class's definition can be changed dynamically  
    MyTest.yourname = "Allen"  
  
    print MyTest.myname  
    print MyTest.yourname

 ---------------------------------------------------------------------------------------------------------------------

class Foo:  
  
    def func(self):  
        print "object method"  
  
    @classmethod  
    def cfunc(cls):  
        print "class method"  
  
    @staticmethod  
    def sfunc(a, b):  
        print "static method:", a, " + ", b, "=", a + b  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
    foo = Foo()

    # instance method can be called by object and class name  
    foo.func()  
    Foo.func(foo)

     # both instance and class can call class method  
    foo.cfunc()  
    Foo.cfunc()

     # both instance and class can call static method   
    Foo.sfunc(10, 20)  
    foo.sfunc(50, 100)

 ----------------------------------------------------------------------------------------------------------------------

注：使用的是Python2.7。

**一、实例方法**

实例方法就是类的实例能够使用的方法。如下：

class Foo:  
    def \_\_init\_\_(self, name):  
        self.name = name  
    def hi(self):  
        print self.name

if \_\_name\_\_ == '\_\_main\_\_':  
    foo01 = Foo('letian')  
    foo01.hi()  
    print type(Foo)  
    print type(foo01)  
    print id(foo01)  
    print id(Foo)

运行结果为:

letian  
<type 'classobj'>  
<type 'instance'>  
40124704  
31323448[code]  
可以看到，Foo的type为classobj（类对象，python中定义的类本身也是对象），foo01的type为instance（实例）。而hi()是实例方法，所以foo01.hi()会输出'letian'。实例方法的第一个参数默认为self，代指实例。self不是一个关键字，而是约定的写法。init()是生成实例时默认调用的实例方法。将Foo的定义改为以下形式：  
class Foo:  
    def \_\_init\_\_(this, name):  
        this.name = name  
    def hi(here):  
        print here.name

运行依然正确。 内置函数id用来查看对象的标识符，下面是其doc内容：

>>> print id.\_\_doc\_\_  
id(object) -> integer

Return the identity of an object.  This is guaranteed to be unique among  
simultaneously existing objects.  (Hint: it's the object's memory address.)

**二、静态方法**

静态方法是一种普通函数，就位于类定义的命名空间中，它不会对任何实例类型进行操作。使用装饰器@staticmethod定义静态方法。类对象和实例都可以调用静态方法：

class Foo:  
    def \_\_init\_\_(self, name):  
        self.name = name  
    def hi(self):  
        print self.name  
    @staticmethod  
    def add(a, b):  
        print a + b

if \_\_name\_\_ == '\_\_main\_\_':  
    foo01 = Foo('letian')  
    foo01.hi()  
    foo01.add(1,2)  
    Foo.add(1, 2)

运行结果如下：

letian  
3  
3

**三、类方法**

类方法是将类本身作为对象进行操作的方法。类方法使用@classmethod装饰器定义，其第一个参数是类，约定写为cls。类对象和实例都可以调用类方法：

class Foo:  
    name = 'letian '  
    @classmethod  
    def hi(cls, x):  
        print cls.name \* x

if \_\_name\_\_ == '\_\_main\_\_':  
    foo01 = Foo()  
    foo01.hi(2)  
    Foo.hi(3)

运行结果如下：

letian letian   
letian letian letian

注意，很多其他的编程语言不允许实例调用类方法。

**四、super**

super用来执行父类中的函数，例如：

class Foo(object):  
    def hi(self):  
        print 'hi,Foo'

class Foo2(Foo):  
    def hi(self):  
        super(Foo2, self).hi()

if \_\_name\_\_ == '\_\_main\_\_':  
    foo2 = Foo2()  
    foo2.hi()

运行结果：

hi,Foo

注意，Foo类必须继承某个类（并且这个继承链开始于object类），否则会报错。如果改成下面的形式：

class Foo:  
    def hi(self):  
        print 'hi,Foo'

class Foo2(Foo):  
    def hi(self):  
        super(Foo2, self).hi()

if \_\_name\_\_ == '\_\_main\_\_':  
    foo2 = Foo2()  
    foo2.hi()

运行时报错如下：

......  
TypeError: must be type, not classobj

关于super，具体请见http://docs.python.org/2/library/functions.html?highlight=super#super以及super.doc。

**五、类变量和实例变量**

类变量定义在类的定义之后，实例变量则是以为self.开头。例如：

class Foo(object):  
    val = 0  
    def \_\_init\_\_(self):  
        self.val = 1

if \_\_name\_\_ == '\_\_main\_\_':  
    foo = Foo()  
    print foo.val  
    print Foo.val

运行结果为：

1  
0

实例也能够访问类变量，如下：

class Foo(object):  
    val = 0  
    def \_\_init\_\_(self):  
        pass  
if \_\_name\_\_ == '\_\_main\_\_':  
    foo = Foo()  
    print foo.val  
    print Foo.val

运行结果如下：

0  
0

另外，可以通过以下方式访问类变量：

class Foo(object):  
    val = 3  
    def \_\_init\_\_(self):  
        print self.\_\_class\_\_.val  
if \_\_name\_\_ == '\_\_main\_\_':  
    foo = Foo()

运行结果：

3

还可以这样：

class Foo(object):  
    val = 3  
    def \_\_init\_\_(self):  
        pass  
    @classmethod  
    def echo(cls):  
        print cls.val  
if \_\_name\_\_ == '\_\_main\_\_':  
    Foo.echo()

运行结果：

3

**六、如何调用父类的构造函数**

子类（派生类）并不会自动调用父类（基类）的init方法，例如：

class Foo(object):  
    def \_\_init\_\_(self):  
        self.val = 1

class Foo2(Foo):  
    def \_\_init\_\_(self):  
        print self.val

if \_\_name\_\_ == '\_\_main\_\_':  
    foo2 = Foo2()

运行时报错。

调用父类的init方法有两种，第一种：

class Foo(object):  
    def \_\_init\_\_(self):  
        self.val = 1

class Foo2(Foo):  
    def \_\_init\_\_(self):  
        Foo.\_\_init\_\_(self)   //类调用实例方法时，需要传入self指代的实例  
        print self.val

if \_\_name\_\_ == '\_\_main\_\_':  
    foo2 = Foo2()

第二种：

class Foo(object):  
    def \_\_init\_\_(self):  
        self.val = 1

class Foo2(Foo):  
    def \_\_init\_\_(self):  
        super(Foo2, self).\_\_init\_\_()    
        print self.val

if \_\_name\_\_ == '\_\_main\_\_':  
    foo2 = Foo2()

这两种方法的运行结果均为：

1

不过这两种方法是有区别的。