Python面向对象编程

一．什么是类什么是实例?

1．类是Python面向对象程序设计的主要工具，通过这种程序设计方法，我们可以把代码的冗余度降到最低，并且通过定制现有的代码来编写新的程序，而不是在原处修改，它提供了代码的定制和复用的机制．

2．类是实例的工厂，类的属性提供了行为，所有从类产生的实例都继承了该类的属性

3．实例代表程序领域中的具体元素，实例属性记录数据，每个特定对象的数据都不同

二．为什么使用类?

1．多重实例：
类是产生对象的工厂，每次调用一个类，就会产生一个具有独立命名空间的新对象，每个由类产生的对象都可以读取类的属性，并获得自己的命名空间来存储数据，这个数据对于每个对象来说都是不同的

2．继承：
我们可以在类的外部重新定义其属性，来扩充这个类

3．组合：
类是一些组件的集合，这些组件以团队的形式共同工作，每个组件都定义了自己的行为以及和其他组件之间的关系

三．属性搜索继承

1．在Python中，关于类属性 类方法的操作都可以使用 object.attribute 表达式(类方法：类中的函数 类属性:类中定义的变量的赋值；这个表达式会在python中启动搜索，来寻找attribute首次出现的位置，搜索的规则为从下到上 从左到右；我们称这个搜索顺序为继承，因为树中位置较低的对象继承了树中位置较高对象拥有的属性．

四．编写类树

1．每个class语句会生成一个新的类对象
2．每次类调用时，就会生成一个新的实例对象
3．实例自动连接到创建了这些实例的类
4．类连接到其超类的方式是，将超类列在类头部的括号内，其从左到右的顺序决定了类树中的次序．

In [2]:
```
#示例代码1.属性搜索继承
class c2:
    x = 'c2 x'
    z = 'c2 z'

class c3:
    w = 'c3 w'
    z = 'c3 z'

class c1(c2, c3):
    x = 'c1 x'
    y = 'c1 y'

i1 = c1()
i2 = c1()

print(i1.w)
#在执行i2.w的时候，会以 i2 c1 c2 c3 的顺序进行类树搜索，找到首个w之后就停止搜索，在此例中直到搜索c3时才找到w;
#因为w只出现在了c3中，我们可以称之为i2从c3继承了属性
```

```
c3 w
```

In [3]:
```
print i1.x
print i2.x
#都会在c1中找到x并停止搜索 因为c1比c2位置更底
```

```
c1 x
c1 x
```

In [4]:
```
print i1.z
print i2.z
#都会在c2中找到z，因为c2比c3更靠左
```

```
c2 z
c2 z
```

In [7]:
```
i1.name = 'this is i1'
i2.name = 'this is i2'

print i1.name
print i2.name
#i2.name 会在实例对象中找到name，不需要爬树
```

```
this is i1
this is i2
```

本例中使用的是多重继承, 在Python中如果class语句中的小括号内有一个以上的超类, 它会以从下往上 从左到右的次序决定超类搜索的顺序

In [8]:
```
#示例代码2.类代码例子

class c1:
    def __init__(self, value):
        self.name = value
    def getName(self):
        print self.name

x = c1('python-training')
x.getName()
```

```
python-training
```

五. 构造函数**init**

每次从类产生实例的时候, Python会自动调用名为**init**的方法, 在实例调用的过程中传入的参数会被**init**所捕获, 第一个参数self, 代表调用了这个类的实例.

六. 经典类与新式类

继承object类的是新式类; 不继承object类的是经典类

In [9]:
```
#示例代码3. 新式类

class A(object):
    def foo(self):
        print 'called class a foo'

class B(A):
    pass

class C(A):
    def foo(self):
        print 'called class c foo'

class D(B, C):
    pass

x = D()
x.foo()

#如果A是新式类, 当调用D的实例的foo()方法时, Python会按照广度优先的方法去搜索foo(), 路径是B-C-A, 执行的是C中的 foo()
```

```
called class c foo
```

In [10]:
```
#示例代码4.经典类

class A():
    def foo(self):
        print 'called class a foo'

class B(A):
    pass

class C(A):
    def foo(self):
        print 'called class c foo'

class D(B, C):
    pass

x = D()
x.foo()

#如果A是经典类, 当调用D的实例的foo()方法时, Python会按照深度优先的方法去搜索foo(), 路径是B-A-C, 执行的是A中的foo()
```

```
called class a foo
```