Python面向对象编程-封装

封装的意义在于保护或者防止代码被被外部在无意中破坏，在面向对象程序设计中一个类被看作是一个中心的元素并且和使用它的对象结合的很密切，所以保护它不被其它的函数意外的修改就显得格外重要，通过封装可以实现要访问该类的代码和数据必须要通过严格的接口控制.

In [2]:
```
#示例代码1.
class student:
    score = 0

s = student()
s.score = 1000
print s.score

s = student()
s.score = 9999
print s.score

# 在绑定属性时，如果我们直接把属性暴露出去，虽然写起来很简单，但是，没办法检查参数导致可以把成绩随便改
```

```
1000
9999
```

我们需要在类中把一个变量定义的属性或方法定义为私有，Python中的表现形式为__foo，通过在类中定义getter和setter接口来获取和设置该变量的值.

private 权限限制最小的访问修饰符，我们将其称之为私有的，被它修饰的属性及方法，只能被类本身访问连子类也不能访问.

Python中的变量的命名存在以下四种情形：
主要存在四种情形
1. object # public
2. __object__ # special, python system use, user should not define like it
3. __object # private (name mangling during runtime)
4. _object # 服从Python的编码规范认为它是私有的

类的私有方法与属性特点：
 1. 不能被实例访问
 2. 不能被子类访问
 3. 只能在本类中使用
 4. 不能被其他模块所引入

In [10]:
```
#示例代码2.

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

    def getName(self):
        print 'My name is %s' %self.name

    def __getAge(self):   #
        print 'My age is %s' %self.age

    def sayHi(self):
        self.getName()
        self.__getAge()

x = c1("lily", 20)
x.getName()
x.sayHi()
x.__getAge() #实例也不能使用
```

```
My name is lily
My name is lily
My age is 20

---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-10-339374e54704> in <module>()
     19 x.getName()
     20 x.sayHi()
---> 21 x.__getAge() #实例也不能使用

AttributeError: c1 instance has no attribute '__getAge'
```

In [9]:
```
#示例代码3.
class c2(c1):
    def say(self):
        self.getName()
        self.__getAge()
y = c2('lucy', 22)
y.getName()
y.say() #子类中也不能使用
```

```
My name is lucy
My name is lucy

---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-9-cd29760bd337> in <module>()
      6 y = c2('lucy', 22)
      7 y.getName()
----> 8 y.say()

<ipython-input-9-cd29760bd337> in say(self)
      3     def say(self):
      4         self.getName()
----> 5         self.__getAge()
      6 y = c2('lucy', 22)
      7 y.getName()

AttributeError: c2 instance has no attribute '_c2__getAge'
```

类的保护方法与属性的特点：
1. 意思是只有类对象和子类对象自己能访问到这些变量.
2. 根据python的约定，应该将其视作private，而不要在外部(如类的示例)使用它们(如果你非要使用也没办法)，良好的编程习惯是不要在外部使用它.
3. 保护属性与方法不能被模块导入的语法所使用

In [6]:
```
#示例代码

class c3(object):
    _a = 100
    def _foo(self):
        print(c3._a + 100)
        return 'xxx'
    def bar(self):
        return 'yyy'

x = c3()
print(x._foo())  ##保护属性可以在本类使用
print(x.bar())
print(x._a)      ##保护属性可以在类的外部所使用


class c4(c3):
    def f1(self):
        print(self._a + 1)  ##保护属性可以在子类中被使用

x = c4()
x.f1()
print(x._a)
```

```
200
xxx
yyy
100
101
100
```

综上类的保护属性与方法可以被 本类使用 也可以被子类使用 也可以在类的外部被类的实例所使用, 但根据Python的约定, 保护属性与方法尽可能不要在外部使用.

封装的优点:

良好的封装能够减少耦合. 类内部的结构可以自由修改. 可以对成员变量进行更精确的控制. 隐藏信息, 实现细节.

为了限制score的范围, 可以通过一个set_score()方法来设置成绩, 再通过一个get_score()来获取成绩, 这样在set_score()方法里,就可以检查参数:

In [13]:
```
#示例代码4.

class Student(object):
    def get_score(self):
         return self._score

    def set_score(self, value):
        if not isinstance(value, int):
            raise ValueError('score must be an integer!')
        if value < 0 or value > 100:
            raise ValueError('score must between 0 ~ 100!')
        self._score = value

s = Student()
s.set_score(60)
print s.get_score()

s.set_score(9999)
```

```
60
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-13-8a1d69d27573> in <module>()
     16 print s.get_score()
     17
---> 18 s.set_score(9999)

<ipython-input-13-8a1d69d27573> in set_score(self, value)
      9             raise ValueError('score must be an integer!')
     10         if value < 0 or value > 100:
---> 11             raise ValueError('score must between 0 ~ 100!')
     12         self._score = value
     13

ValueError: score must between 0 ~ 100!
```

上面的调用方法又略显复杂, 有没有既能检查参数, 又可以用类似属性这样简单的方式来访问类的变量? 对于追求完美的Python程序员来说, 这是必须要做到的.

In [29]:
```
#示例代码5.

class Student(object):

    @property
    def score(self):
        return self._score

    @score.setter
    def score(self, value):
        if not isinstance(value, int):
            raise ValueError('score must be an integer!')
        if value < 0 or value > 100:
            raise ValueError('score must between 0 ~ 100!')
        self._score = value

x = Student()
x.score = 60
print x.score
x.score = 9999
#@property下修饰的函数的函数名与score.setter下的函数名必须都是一致的
```

```
60

---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-29-fba981adcd81> in <module>()
     18 x.score = 60
     19 print x.score
---> 20 x.score = 9999

<ipython-input-29-fba981adcd81> in score(self, value)
     12             raise ValueError('score must be an integer!')
     13         if value < 0 or value > 100:
---> 14             raise ValueError('score must between 0 ~ 100!')
     15         self._score = value
     16

ValueError: score must between 0 ~ 100!
```

In [ ]: