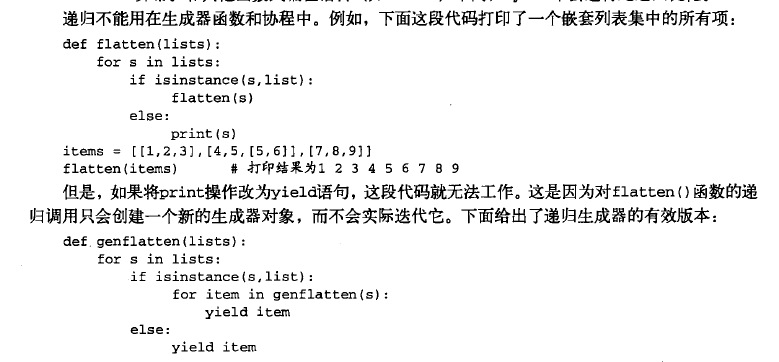
**递归（recursion）：**

递归在python 中有深度限制，默认值是1000，尽管可以调用sys.setrecursionlimit()修改值[sys.getrecursionlimit()]，但仍然受限与主机操作系统使用的限制，超出深度后就会引发RuntimeError的异常,下面是递归在print和generator 的应用：



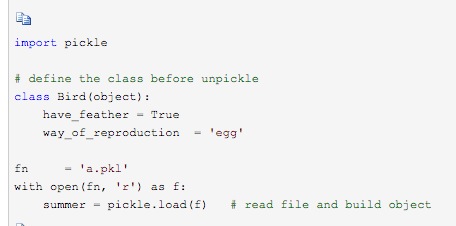
并且，递归函数与装饰器结合，所有的内部调用的函数都**通过装饰后**的。

**Pickle module（保存对象）：**

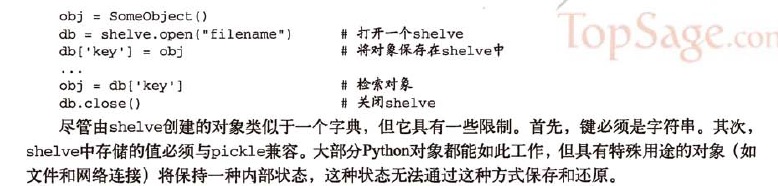
我们可以直接将某个对象所对应位置的数据抓取下来，转换成文本流 (这个过程叫做serialize)，然后将文本流存入到文件中。例如如下实例：



重载实例：



另一个包shelve类似于pickle,但其将对象保存在一个类似于字典的数据库里：



还有一个cPickle包，作用与pickle几乎相同，但是基于c语言编写的，速度是pickle 的1000倍作用。

如果文件过大，则用pickle 加载对象时可能不能正确还原对象关系。解决方法是用Pickler(file[,protocol]),Unpickler(file) ,创建序列化对象。

**XOR（异或）：**

For any number x, x^0=x,x^x=0,then y^x^x=y，

一、交换两个整数的值而不必用第三个参数  
a = 9;  
b = 11;  
  
a=a^b; 1001^1011=0010  
b=b^a; 1011^0010=1001  
a=a^b;  0010^1001=1011  
  
a = 11;  
b = 9;

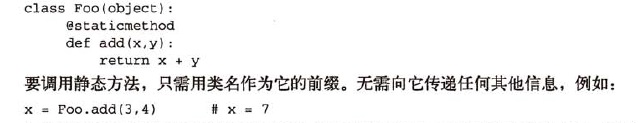
二、奇偶判断

^a操作就是将a中的每一位按位逐一进行异或，例如a=4'b1010，则b=1^0^1^0=0，由此可以判断a中为1的位数是奇数还是偶数，是一个便捷的操作。

**类中的方法：实例方法，静态方法，类方法**

**实例方法**：即第一个参数是self

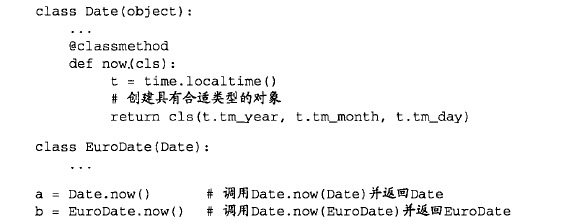
**静态方法：**一种普通函数，位于类的命名空间内，不会对任何实例类型操作。定义静态方法可以使用@staticmethod装饰器，如：



一般地，如果我们需要采取不同地方法创建新实例，由于\_\_init\_\_只有一个，我们可以使用静态方法来创建其相关地属性。

**类方法：**将类本身作为对象进行操作地方法，第一个参数必须是cls.需要@classmethod,f访问时即class.name() [与实例方法类似，只是将对象名字改为类名字访问该函数（方法）]

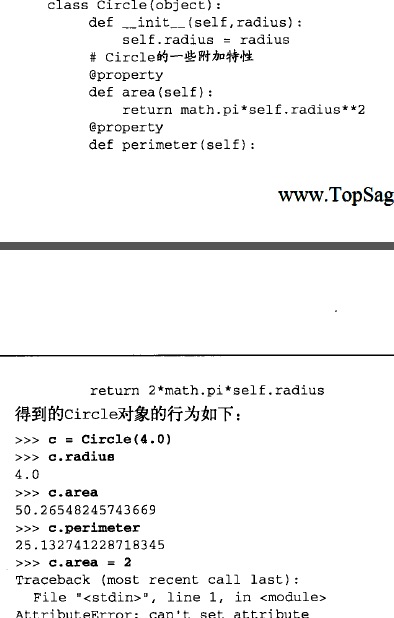
类方法也会随着继承机制继承下来，子类也可以用同类方式访问类方法。



**注意：python中对象实例也可以访问静态方法和类方法，这与其他面向对象语言（如smalltalk and Ruby ）不一样。**

**类中的特性（property）：**

将类的一种特殊的属性归为特性－property,访问这些特性将会计算其对应的值，如：



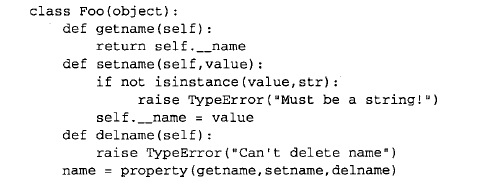
其中area 和perimeter 就是特性，我们可以访问，但不能修改其值。

同样，下面是两种方法使用特性：

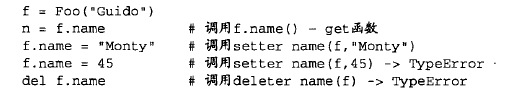
1:使用装饰器：



2:没使用装饰器：



两种方法等效，执行结果如下：

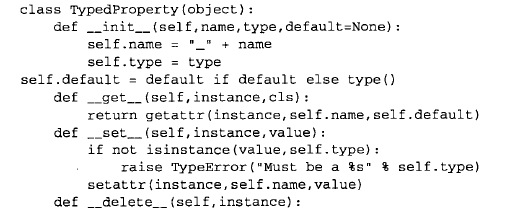


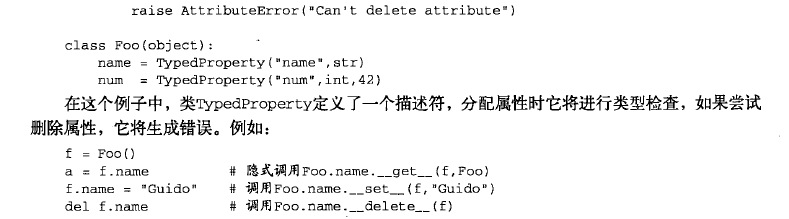
f 是实例，name是类的特性，可以通过f.name 对\_\_name进行赋值和访问。Del f.name进行删除。

**描述符（discriptor）：**

描述符是property的升级版，允许你为重复的property逻辑编写单独的类来处理。(加入上述例子中我们有很多像name特性的类属性，我们就需要为每一个特性设置get,set del函数，这样太麻烦)

描述符，用一句话来说，**就是将某种特殊类型的类的实例指派给另一个类的属性(注意：这里是类属性，而不是对象属性,即只能在类级别上进行实例化)**。而这种特殊类型的类就是实现了\_\_get\_\_，\_\_set\_\_,\_\_delete\_\_的新式类(即继承object)





其他关于描述符的reference:

<http://www.geekfan.net/7862/>

<http://www.jb51.net/article/62987.htm>

**对象的属性绑定：**

关于实例对象的属性（attribute），我们可以在\_\_dict\_\_ 访问该字典，例如实例a,类A:

a.\_\_dict\_\_ => 得到一个字典，这个字典包含a实例的属性名称和相应的值，该字典的key是属性的名称，必须为字符串，value是相应的值，如：{‘name’:’fui’}

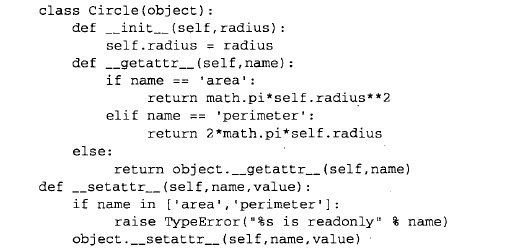
=>a.name=’fui’ ,

**注意：**可以任何时候给实例添加新属性，并且这些修改始终能反应到\_\_dict\_\_属性中。

A.\_\_dict\_\_ => 返回类所有的方法（特殊方法和一般方法）的字典，同前者一样，

A.\_\_dict\_\_.keys( ) => 只返回名称

自定义的实现自己访问属性的函数的类：



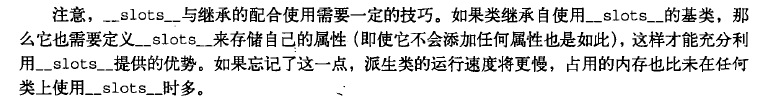
**\_\_Slots\_\_的作用：**

1: 实例上的属性将会被严格限制在\_\_slots\_\_指定的属性名称，如：

\_\_slots\_\_=(‘name’,’age’),这样可以有效防止用户向实例任意添加新属性

2:主要可以优化性能，此时不再用字典存贮实例数据，

另外注意：

****

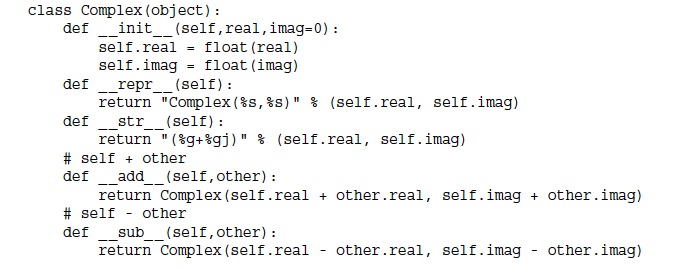
**\_\_repr\_\_(self)和\_\_str\_\_(self)：**

两者都是创建对象的简单的字符串表示，不同点是：

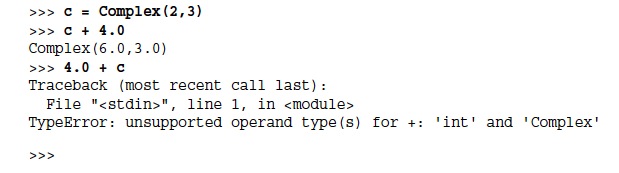
前者可在交换式解释器检查变量看到输出值(可以重新定义\_\_repr\_\_(self) 来决定我们想要显示的内容)，

后者一般都显示在print 后面的语句中，且其更简单，若为定义\_\_str\_\_(self),则调用\_\_repr\_\_(self)显示print.

例如：

****

在交互式解释器运行时**：**

****

**元类（metaclass）：**

是创建类的类，type 本身就是一个元类，可以动态地创建一个类，我们也可以自定义一个元类，来定义我们要怎么创建类。

\_\_metaclass\_\_=something, python 就会用元类创建类了

reference: http://blog.jobbole.com/21351/

**抽象基类（abstractclass）：**

**对象的\_\_dict\_\_ 和\_\_getattr\_\_：**

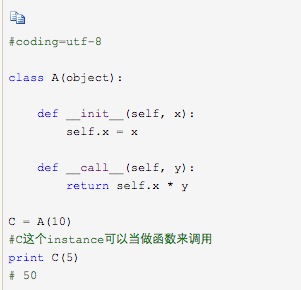
对象的属性一般存储在\_\_dict\_\_的字典中，键为属性名，对应的值为属性本身

可以看到，Python中的属性是分层定义的，比如这里分为object/bird/chicken/summer这四层。当我们需要调用某个属性的时候，Python会一层层向上遍历，直到找到那个属性。(某个属性可能出现再不同的层被重复定义，Python向上的过程中，会选取先遇到的那一个，也就是比较低层的属性定义)。

我们可以用\_\_getattr\_\_(self, name)来查询即时生成的属性。当我们查询一个属性时，如果通过\_\_dict\_\_方法无法找到该属性，那么Python会调用对象的\_\_getattr\_\_方法，来即时生成该属性。

每个特性需要有自己的处理函数，而\_\_getattr\_\_可以将所有的即时生成属性放在同一个函数中处理。\_\_getattr\_\_可以根据函数名区别处理不同的属性。

**对象\_\_call\_\_ 方法：**

****

**threading 包：**

threading.Thread对象： 我们已经介绍了该对象的start()和run(),OOP的话，通过修改Thread类的run()方法来定义线程所要执行的命令。并调用start()方法来运行线程

此外：

* join()方法，调用该方法的线程将等待直到该Thread对象完成，再恢复运行。这与进程间调用wait()函数相类似。

下面的对象用于处理多线程同步。对象一旦被建立，可以被多个线程共享，并根据情况阻塞某些进程。请与[Linux多线程与同步](http://www.cnblogs.com/vamei/archive/2012/10/09/2715393.html)中的同步工具参照阅读。

threading.Lock对象: mutex, 有acquire()和release()方法。

threading.Condition对象: condition variable，建立该对象时，会包含一个Lock对象 (因为condition variable总是和mutex一起使用)。可以对Condition对象调用acquire()和release()方法，以控制潜在的Lock对象。此外:

* wait()方法，相当于cond\_wait()
* notify\_all()，相当与cond\_broadcast()
* nofify()，与notify\_all()功能类似，但只唤醒一个等待的线程，而不是全部

threading.Semaphore对象: semaphore，也就是计数锁(semaphore传统意义上是一种进程间同步工具，见[Linux进程间通信](http://www.cnblogs.com/vamei/archive/2012/10/10/2715398.html))。创建对象的时候，可以传递一个整数作为计数上限 (sema = threading.Semaphore(5))。它与Lock类似，也有Lock的两个方法。

threading.Event对象: 与threading.Condition相类似，相当于没有潜在的Lock保护的condition variable。对象有True和False两个状态。可以多个线程使用wait()等待，直到某个线程调用该对象的set()方法，将对象设置为True。线程可以调用对象的clear()方法来重置对象为False状态

**multiprocessing 包:**

multiprocessing包是Python中的多进程管理包。与threading.Thread类似，它可以利用multiprocessing.Process对象来创建一个进程。该进程可以运行在Python程序内部编写的函数。该Process对象与Thread对象的用法相同，也有start(), run(), join()的方法。此外multiprocessing包中也有Lock/Event/Semaphore/Condition类 (这些对象可以像多线程那样，通过参数传递给各个进程)，用以同步进程，其用法与threading包中的同名类一致。

但在使用这些共享API的时候，我们要注意以下几点:

* 在UNIX平台上，当某个进程终结之后，该进程需要被其父进程调用wait，否则进程成为[僵尸进程](http://www.cnblogs.com/vamei/archive/2012/09/20/2694466.html)(Zombie)。所以，有必要对每个Process对象调用join()方法 (实际上等同于wait)。对于多线程来说，由于只有一个进程，所以不存在此必要性。
* multiprocessing提供了threading包中没有的IPC(比如Pipe和Queue)，效率上更高。应优先考虑Pipe和Queue，避免使用Lock/Event/Semaphore/Condition等同步方式 (因为它们占据的不是用户进程的资源)。
* 多进程应该避免共享资源。在多线程中，我们可以比较容易地共享资源，比如[使用全局变量或者传递参数](http://www.cnblogs.com/vamei/archive/2012/10/11/2720042.html)。在多进程情况下，由于每个进程有自己独立的内存空间，以上方法并不合适。此时我们可以通过[共享内存](http://www.cnblogs.com/vamei/archive/2012/10/10/2715398.html)和Manager的方法来共享资源。但这样做提高了程序的复杂度，并因为同步的需要而降低了程序的效率。

IPC:interprocess communication

**Pipe和Queue**

1) Pipe可以是单向(half-duplex)，也可以是双向(duplex)。我们通过mutiprocessing.Pipe(duplex=False)创建单向管道 (默认为双向)。一个进程从PIPE一端输入对象，然后被PIPE另一端的进程接收，单向管道只允许管道一端的进程输入，而双向管道则允许从两端输入。

2)Queue与Pipe相类似，都是先进先出的结构。但Queue允许多个进程放入，多个进程从队列取出对象。Queue使用mutiprocessing.Queue(maxsize)创建，maxsize表示队列中可以存放对象的最大数量。

**字符串格式化:**

%s    字符串 (采用str()的显示)

%r    字符串 (采用repr()的显示)

%c    单个字符

%b    二进制整数

%d    十进制整数

%i    十进制整数

%o    八进制整数

%x    十六进制整数

%e    指数 (基底写为e)

%E    指数 (基底写为E)

%f    浮点数

%F    浮点数，与上相同

%g    指数(e)或浮点数 (根据显示长度)

%G    指数(E)或浮点数 (根据显示长度)

%%    字符"%"

%[(name)][flags][width].[precision]typecode

(name)为命名

flags可以有+,-,' '或0。+表示右对齐。-表示左对齐。' '为一个空格，表示在正数的左侧填充一个空格，从而与负数对齐。0表示使用0填充。

width表示显示宽度

precision表示小数点后精度

我们还可以用词典来传递真实值。如下：

print("I'm %(name)s. I'm %(age)d year old" % {'name':'Vamei', 'age':99})

可以看到，我们对两个格式符进行了命名。命名使用()括起来。每个命名对应词典的一个key。

**删除list一些元素:**

del a[index1:index2+1]

or a[index1:index2+1]=[]

**python中关于文件的一些操作：**

file object = open(file\_name [, access\_mode][, buffering])

注意： file\_name 必须是字符串，

access\_mode :



file object 的一些attributes:

****

file.close();

fileObject.read([count]); 在这里，被传递的参数是要从已打开文件中**读取的字节**计数

**OS 模块中重命名和删除文件：**

os.rename(current\_file\_name, new\_file\_name)；

os.remove(file\_name)

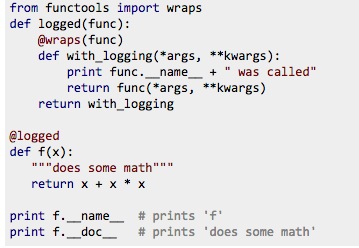
**查询class的方法：**

dir(), 例如dir(list) or dir([])

**关于装饰器的一个例子：**

Python装饰器（decorator）在实现的时候，有一些细节需要被注意。例如，被装饰后的函数其实已经是另外一个函数了（函数名等函数属性会发生改变）

Python的functools包中提供了一个叫wraps的decorator来消除这样的副作用。写一个decorator的时候，最好在实现之前加上functools的wrap，它能保留原有函数的名称和docstring，例如：



**关于交换dict中key和value 的值：**

python3

[?](http://www.jb51.net/article/70542.htm)

|  |  |
| --- | --- |
| 1  2  3  4  5 | >>> d1={'a':1,'b':2}  >>> {value:key for key,value in d1.iteritems()}  {1: 'a', 2: 'b'}  >>> {value:key for key,value in d1.iteritems()}[2]  'b' |

python2.7

[?](http://www.jb51.net/article/70542.htm)

|  |  |
| --- | --- |
| 1  2  3 | >>> d1={'a':1,'b':2}  >>> dict((value,key) for key,value in d1.iteritems())  {1: 'a', 2: 'b'} |

如果有重复的key,用list 来存储相应的value.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | >>> d1={'a':1,'b':2,'c':1}  >>> d=defaultdict(list)  >>> for k,v in d1.iteritems():  ...   d[v].append(k)  ...  >>> d  defaultdict(<type 'list'>, {1: ['a', 'c'], 2: ['b']}) |

**关于dict 中合并两个字典的两种方法比较：**

现有两个字典dict如下：

|  |  |
| --- | --- |
| 1  2 | dict1={1:[1,11,111],2:[2,22,222]}  dict2={3:[3,33,333],4:[4,44,444]} |

合并两个字典得到类似：

|  |  |
| --- | --- |
| 1 | {1:[1,11,111],2:[2,22,222],3:[3,33,333],4:[4,44,444]} |

方法1：

|  |  |
| --- | --- |
| 1 | dictMerged1=dict(dict1.items()+dict2.items()) |

方法2：

|  |  |
| --- | --- |
| 1 | dictMerged2=dict(dict1, \*\*dict2) |

方法2等同于：

|  |  |
| --- | --- |
| 1  2 | dictMerged=dict1.copy()  dictMerged.update(dict2) |

或者：

|  |  |
| --- | --- |
| 1  2 | dictMerged=dict(dict1)＃dict(dict1) 类似于浅拷贝  dictMerged.update(dict2) |

**关于bit representation 的 方法：**

如果是正数，直接bin(i) 即可，

但如果是负数，我们想知道其32位的表示方法，

可以用bin( ( (1<<32) – 1) & i ), 其实这种方法使用与i为任意的数，pos,neg,or zero.

**关于遍历列表的同时也要删除某些元素的时候：**

1: 可以用filter 函数，如：

l = [1,2,3,4]

l = filter(lambda x:x !=4,l)

2:一般能用filter的都可以被列表解析所代替

l = [1,2,3,4]

l = [ i for i in l if i !=4]//同样产生一个新序列，复值给l

上述两种方法都可以安全地删除某些元素

**关于max 的选择用法：**

max([1,2,3],[22,34],[23,2],lambda x:x[0])

注意和sort()方法不一样，sort 对象是一个列表或字典，而max是多个列表和对象。