[**python 变量命名规范**](http://greybeard.iteye.com/blog/1355216)

**博客分类：**

* [python](http://greybeard.iteye.com/category/188408)

出自:<http://www.diybl.com/course/3_program/python/20111130/563643.html>   
模块名：   
小写字母，单词之间用\_分割   
ad\_stats.py   
  
包名：   
和模块名一样   
  
类名：   
单词首字母大写   
AdStats   
ConfigUtil   
  
全局变量名（类变量，在java中相当于static变量）：   
大写字母，单词之间用\_分割   
NUMBER   
COLOR\_WRITE   
  
普通变量：   
小写字母，单词之间用\_分割   
this\_is\_a\_var   
  
实例变量：   
以\_开头，其他和普通变量一样   
\_price      
\_instance\_var   
  
私有实例变量（外部访问会报错）：   
以\_\_开头（2个下划线），其他和普通变量一样   
\_\_private\_var   
  
专有变量：   
\_\_开头，\_\_结尾，一般为python的自有变量，不要以这种方式命名   
\_\_doc\_\_   
\_\_class\_\_   
  
普通函数：   
和普通变量一样：   
get\_name()   
count\_number()   
ad\_stat()   
  
私有函数（外部访问会报错）：   
以\_\_开头（2个下划线），其他和普通函数一样   
\_\_get\_name()   
————————————————————————————————————————————————————————————————————  
文件名   
全小写,可使用下划线   
包   
应该是简短的、小写的名字。如果下划线可以改善可读性可以加入。如mypackage。   
模块   
与包的规范同。如mymodule。   
类   
总是使用首字母大写单词串。如MyClass。内部类可以使用额外的前导下划线。   
  
函数&方法   
函数名应该为小写，可以用下划线风格单词以增加可读性。如：myfunction，my\_example\_function。   
\*注意\*：混合大小写仅被允许用于这种风格已经占据优势的时候，以便保持向后兼容。   
函数和方法的参数   
总使用“self”作为实例方法的第一个参数。总使用“cls”作为类方法的第一个参数。   
如果一个函数的参数名称和保留的关键字冲突，通常使用一个后缀下划线好于使用缩写或奇怪的拼写。   
全局变量   
对于from M import \*导入语句，如果想阻止导入模块内的全局变量可以使用旧有的规范，在全局变量上加一个前导的下划线。   
\*注意\*:应避免使用全局变量   
变量   
变量名全部小写，由下划线连接各个单词。如color = WHITE，this\_is\_a\_variable = 1   
\*注意\*：   
1.不论是类成员变量还是全局变量，均不使用 m 或 g 前缀。   
2.私有类成员使用单一下划线前缀标识，多定义公开成员，少定义私有成员。   
3.变量名不应带有类型信息，因为Python是动态类型语言。如 iValue、names\_list、dict\_obj 等都是不好的命名。   
常量   
常量名所有字母大写，由下划线连接各个单词如MAX\_OVERFLOW，TOTAL。   
异常   
以“Error”作为后缀。   
缩写   
命名应当尽量使用全拼写的单词，缩写的情况有如下两种：   
1.常用的缩写，如XML、ID等，在命名时也应只大写首字母，如XmlParser。   
2.命名中含有长单词，对某个单词进行缩写。这时应使用约定成俗的缩写方式。   
例如：   
function 缩写为 fn   
text 缩写为 txt   
object 缩写为 obj   
count 缩写为 cnt   
number 缩写为 num，等。   
前导后缀下划线   
一个前导下划线：表示非公有。   
一个后缀下划线：避免关键字冲突。   
两个前导下划线：当命名一个类属性引起名称冲突时使用。   
两个前导和后缀下划线：“魔”（有特殊用途）对象或者属性，例如\_\_init\_\_或者\_\_file\_\_。绝对不要创造这样的名字，而只是使用它们。   
\*注意\*：关于下划线的使用存在一些争议。   
Python 用下划线作为变量前缀和后缀指定特殊变量。   
  
\_xxx      不能用'from module import \*'导入   
\_\_xxx\_\_ 系统定义名字   
\_\_xxx    类中的私有变量名   
  
核心风格：避免用下划线作为变量名的开始。   
  
因为下划线对解释器有特殊的意义，而且是内建标识符所使用的符号，我们建议程序员避免用下划线作为变量名的开始。一般来讲，变量名\_xxx被看作是“私有的”，在模块或类外不可以使用。当变量是私有的时候，用\_xxx 来表示变量是很好的习惯。因为变量名\_\_xxx\_\_对Python 来说刑厥夂澹杂谄胀ǖ谋淞坑Φ北苊庹庵置绺瘛?br>   
"单下划线" 开始的成员变量叫做保护变量，意思是只有类对象和子类对象自己能访问到这些变量；   
"双下划线" 开始的是私有成员，意思是只有类对象自己能访问，连子类对象也不能访问到这个数据。   
  
以单下划线开头（\_foo）的代表不能直接访问的类属性，需通过类提供的接口进行访问，不能用“from xxx import \*”而导入；以双下划线开头的（\_\_foo）代表类的私有成员；以双下划线开头和结尾的（\_\_foo\_\_）代表python里特殊方法专用的标识，如 \_\_init\_\_（）代表类的构造函数。   
特定命名方式   
主要是指 \_\_xxx\_\_ 形式的系统保留字命名法。项目中也可以使用这种命名，它的意义在于这种形式的变量是只读的，这种形式的类成员函数尽量不要重载。如   
class Base(object):   
def \_\_init\_\_(self, id, parent = None):   
self.\_\_id\_\_ = id   
self.\_\_parent\_\_ = parent   
def \_\_message\_\_(self, msgid):   
# …略   
其中 \_\_id\_\_、\_\_parent\_\_ 和 \_\_message\_\_ 都采用了系统保留字命名法。   
附:Google Python命名规范   
module\_name, package\_name, ClassName, method\_name, ExceptionName, function\_name, GLOBAL\_VAR\_NAME, instance\_var\_name, function\_parameter\_name, local\_var\_name.   
————————————————————————————————————————————————————————  
from：http://hi.baidu.com/kxw102/blog/item/212e9f3859202fe33b87ce4b.html   
理解Python命名机制   
  
引子   
我热情地邀请大家猜测下面这段程序的输出：   
class A(object):   
       def \_\_init\_\_(self):   
              self.\_\_private()   
              self.public()   
       def \_\_private(self):   
              print 'A.\_\_private()'   
       def public(self):   
              print 'A.public()'   
class B(A):   
       def \_\_private(self):   
              print 'B.\_\_private()'   
       def public(self):   
              print 'B.public()'   
b = B()   
  
初探   
正确的答案是：   
A.\_\_private()   
B.public()   
如果您已经猜对了，那么可以不看我这篇博文了。如果你没有猜对或者心里有所疑问，那我的这篇博文正是为您所准备的。   
一切由为什么会输出“A.\_\_private()”开始。但要讲清楚为什么，我们就有必要了解一下Python的命名机制。   
据 Python manual，变量名（标识符）是Python的一种原子元素。当变量名被绑定到一个对象的时候，变量名就指代这个对象，就像人类社会一样，不是吗？当变 量名出现在代码块中，那它就是本地变量；当变量名出现在模块中，它就是全局变量。模块相信大家都有很好的理解，但代码块可能让人费解些。在这里解释一下：   
代码块就是可作为可执行单元的一段Python程序文本；模块、函数体和类定义都是代码块。不仅如此，每一个交互脚本命令也是一个代码块；一个脚本文件也是一个代码块；一个命令行脚本也是一个代码块。   
接 下来谈谈变量的可见性，我们引入一个范围的概念。范围就是变量名在代码块的可见性。如果一个代码块里定义本地变量，那范围就包括这个代码块。如果变量定义 在一个功能代码块里，那范围就扩展到这个功能块里的任一代码块，除非其中定义了同名的另一变量。但定义在类中的变量的范围被限定在类代码块，而不会扩展到 方法代码块中。   
  
迷踪   
据上节的理论，我们可以把代码分为三个代码块：类A的定义、类B的定义和变量b的定义。根据类定义，我们知道代码给类A定义了三个成员变量（Python的函数也是对象，所以成员方法称为成员变量也行得通。）；类B定义了两个成员变量。这可以通过以下代码验证：   
>>> print '\n'.join(dir(A))   
\_A\_\_private   
\_\_init\_\_   
public   
>>> print '\n'.join(dir(B))   
\_A\_\_private   
\_B\_\_private   
\_\_init\_\_   
public   
咦，为什么类A有个名为\_A\_\_private的 Attribute 呢？而且\_\_private消失了！这就要谈谈Python的私有变量轧压了。   
  
探究   
懂 Python的朋友都知道Python把以两个或以上下划线字符开头且没有以两个或以上下划线结尾的变量当作私有变量。私有变量会在代码生成之前被转换为 长格式（变为公有）。转换机制是这样的：在变量前端插入类名，再在前端加入一个下划线字符。这就是所谓的私有变量轧压（Private name mangling）。如类A里的\_\_private标识符将被转换为\_A\_\_private，这就是上一节出现\_A\_\_private和 \_\_private消失的原因了。   
再讲两点题外话：   
一是因为轧压会使标识符变长，当超过255的时候，Python会切断，要注意因此引起的命名冲突。   
二是当类名全部以下划线命名的时候，Python就不再执行轧压。如：   
>>> class \_\_\_\_(object):   
       def \_\_init\_\_(self):   
              self.\_\_method()   
       def \_\_method(self):   
              print '\_\_\_\_.\_\_method()'   
>>> print '\n'.join(dir(\_\_\_\_))   
\_\_class\_\_   
\_\_delattr\_\_   
\_\_dict\_\_   
\_\_doc\_\_   
\_\_getattribute\_\_   
\_\_hash\_\_   
\_\_init\_\_   
\_\_method              # 没被轧压   
\_\_module\_\_   
\_\_new\_\_   
\_\_reduce\_\_   
\_\_reduce\_ex\_\_   
\_\_repr\_\_   
\_\_setattr\_\_   
\_\_str\_\_   
\_\_weakref\_\_   
>>> obj = \_\_\_\_()   
\_\_\_\_.\_\_method()   
>>> obj.\_\_method()      # 可以外部调用   
\_\_\_\_.\_\_method()   
现在我们回过头来看看为什么会输出“A.\_\_private()”吧！   
  
真相   
相信现在聪明的读者已经猜到答案了吧？如果你还没有想到，我给你个提示：真相跟C语言里的宏预处理差不多。   
因为类A定义了一个私有成员函数（变量），所以在代码生成之前先执行私有变量轧压（注意到上一节标红的那行字没有？）。轧压之后，类A的代码就变成这样了：   
class A(object):   
       def \_\_init\_\_(self):   
              self.\_A\_\_private()          # 这行变了   
              self.public()   
       def \_A\_\_private(self):           # 这行也变了   
              print 'A.\_\_private()'   
       def public(self):   
              print 'A.public()'   
是不是有点像C语言里的宏展开啊？   
因为在类B定义的时候没有覆盖\_\_init\_\_方法，所以调用的仍然是A.\_\_init\_\_，即执行了self.\_A\_\_private()，自然输出“A.\_\_private()”了。   
下面的两段代码可以增加说服力，增进理解：   
>>> class C(A):   
       def \_\_init\_\_(self):          # 重写\_\_init\_\_，不再调用self.\_A\_\_private   
              self.\_\_private()       # 这里绑定的是\_C\_private   
              self.public()   
       def \_\_private(self):   
              print 'C.\_\_private()'   
       def public(self):   
              print 'C.public()'   
>>> c = C()   
C.\_\_private()   
C.public()   
############################   
>>> class A(object):   
       def \_\_init\_\_(self):   
              self.\_A\_\_private()   # 调用一个没有定义的函数，Python会把它给我的 ^\_^～   
              self.public()   
       def \_\_private(self):   
              print 'A.\_\_private()'   
       def public(self):   
              print 'A.public()'   
>>>a = A()   
A.\_\_private()   
A.public()