PCS7　Python 编码规范

概述

本文参考自 Python 增进提案仓库 008 号文件的倡议。

原文“PEP 008 《Style Guide for Python Code》”：*http://www.python.org/dev/peps/pep-0008/*

精巧地址：*http://bit.ly/2OQ8Z3*

其中文翻译：*http://wiki.woodpecker.org.cn/moin/PythonCodingRule*

精巧地址：*http://bit.ly/3HURoL*

在这里先简要叙述一下要点。

应用

Python 八荣八耻

以动手实践为荣，以只看不练为耻；

以打印日志为荣，以单步跟踪为耻；

以空格缩进为荣，以制表缩进为耻；

以单元测试为荣，以人工测试为耻；

以模块复用为荣，以复制粘贴为耻；

以多态应用为荣，以分支判断为耻；

以 Pythonic 为荣，以冗余拖沓为耻；

以总结分享为荣，以跪求其解为耻。

这首诗出自于 CPyUG:40912：*http://groups.google.com/group/python-cn/browse_thread/thread/71c74c4e77577365/a4940f98b59bde43*

精巧地址：*http://bit.ly/4jeBor*

它非常简明地指出了学习 Python 的多个注意点。

一致性的建议

愚蠢地使用一致性是无知的妖怪（A Foolish Consistency is the Hobgoblin of Little Minds）。这里的一致性主要是指一个项目内的一致性和一个模块或函式内的一致性，相对于前者而言，后者更为重要。但最重要的是知道何时会不一致。当出现不一致时，运用自己的最佳判断，看看别的例子，然后决定怎样看起来更好。

代码的布局

缩进

建议使用 Emacs 的 Python-mode 默认值：4 个空格一个缩进层次。对于确实古老的代码，若不希望产生混乱，可以继续使用 8 空格的制表符。在 Emacs 的 Python-mode 中会自动发现文件中主要的缩进层次，依此设定缩进参数。如果使用其他的编辑器，如 vim、gedit、ulipad 等，积极建议把 4 个空格作为一个缩进层次。

制表符还是空格

永远不要混用制表符和空格，因为如果混用了，虽然在编辑环境中显示两条语句为同一缩进层次，但因为制表符和空格的不同会导致 Python 解释为两个不同的层次。

最流行的 Python 缩进方式是仅使用空格，其次是仅使用制表符。若一定要混合使用制表符和空格，可以将其转换成仅使用空格。如在 Emacs 中，选中整个缓冲区，按 ESC+X 键去除制表符。或者在调用 Python 命令行解释器时使用-t 选项，可对代码中不合法的混合制表符和空格发出警告，使用-tt 时警告将变成错误，这些选项是被高度推荐的。但是强烈推荐仅使用空格而不是制表符。

行的最大长度

有许多设备被限制为每行 80 字符，窗口也限制为 80 个字符，因此，建议将所有行限制在最大 79 字符（Emacs 准确地将行限制为长 80 字符）。

对顺序摆放的大块文本（文档字符串或注释），推荐将长度限制为 72 字符。折叠长行的首选方法是使用 Pyhon 支持的圆括号、方括号或花括号内的行延续。如果需要，可以在表达式周围增加一对额外的圆括号，但是使用反斜杠看起来会更好，比如下面这个例子：

```
          class Rectangle(Blob):
                def __init__(self, width, height,
                             color='black', emphasis=None, highlight=0):
                      if width == 0 and height == 0 and \
                         color == 'red' and emphasis == 'strong' or \
                         highlight > 100:
                             raise ValueError, "sorry, you lose"
                      if width == 0 and height == 0 and (color == 'red' or
                                                         emphasis is None):
                             raise ValueError, "I don't think so"
                      Blob.__init__(self, width, height,
                                    color, emphasis, highlight)
```

空行

用两行空行分割顶层函式和类的定义，类内方法的定义用单个空行分割。

额外的空行可被用于分割一组相关函式。在一组相关的单句中间可以省略空行。

当空行用于分割方法的定义时，在 class 行和第一个方法定义之间也要有一个空行。

在函式中使用空行时，请谨慎地将它用于表示一个逻辑段落。Python 接受 Control+L（即 ^L）换页符作为空格；Emacs（和一些打印工具）视这个字符为页面分割符，因此在文件中，可以用它们来作为相关片段分页。

导入

通常应该在单独的行中导入，例如：

```
No:  import sys, os
Yes: import sys
       import os
```

但是这样也是可以的：

```
from types import StringType, ListType
```

imports 通常被放置在文件的顶部，仅在模块注释和文档字符串之后，在模块的全局变量和常量之前。imports 应该有顺序地成组安放，顺序依次为：标准库的导入、 相关的主包的导入、特定应用的导入。同时建议在每组导入之间放置一个空行。

对于内部包的导入是不推荐使用相对路径的，而对所有导入都要使用包的绝对路径。

从一个包含类的模块中导入类时，通常可以写成这样：

```
from MyClass import MyClass
from foo.bar.YourClass import YourClass
```

如果这样写导致了本地名字冲突，那么就这样写：

```
import MyClass
import foo.bar.YourClass
```

即可以使用 MyClass.MyClass 和 foo.bar.YourClass.YourClass 这种写法。

空格

建议不要在以下地方出现空格：

- 紧挨着圆括号、方括号和花括号的地方，如 spam(ham[1], { eggs: 2 })，要始终将它写成 spam(ham[1], {eggs: 2})
- 紧贴在逗号、分号或冒号前的地方，如 if x == 4 : print x , y ; x , y = y , x，要始终将它写成 if x == 4: print x, y; x, y = y, x
- 紧贴着函式调用的参数列表前开式括号（open parenthesis ）的地方，如 spam (1)，要始终将它写成 spam(1)
- 紧贴在索引或切片开始的开式括号前的地方，如 dict ['key'] = list [index]，要始终将它写成 dict['key'] = list[index]

在赋值（或其他）运算符周围，用于和其他并排的一个以上的空格，如：

```
x           = 1
y           = 2
long_variable = 3
```

要始终将它写成

```
x = 1
y = 2
long_variable = 3
```

始终在这些二元运算符两边放置一个空格，如赋值（=）、比较（==, <, >, !=, <>, <=,>=, in, not in, is, is not）、布尔运算（and, or, not）。

在算术运算符周围插入空格，始终保持二元运算符两边的空格一致。

不要在用于指定关键字参数或默认参数值的=号周围使用空格。不要将多条语句写在同一行上，如：

```
No:  if foo == 'blah': do_blah_thing()
Yes: if foo == 'blah':
```

```
            do_blah_thing()
No:  do_one(); do_two(); do_three()
Yes: do_one()
     do_two()
     do_three()
```

文档化

为所有公共模块、函式、类和方法编写文档字符串。文档字符串对非公开的方法不是必要的，但你应该有一个描述这个方法做什么的注释。

多行文档字符串结尾的""" 应该单独成行，例如：

```
"""Return a foobang
Optional plotz says to frobnicate the bizbaz first.
"""
```

对单行的文档字符串，结尾的"""在同一行也可以。

版本注记

如果要将 RCS 或 CVS 的杂项包含在你的源文件中，按如下格式操作：

```
__version__ = "$Revision: 1.4 $"
# $Source: E:/cvsroot/python_doc/pep8.txt,v $
```

对于 CVS 的服务器工作标记更应该在代码段中明确出它的使用说明，如在文档最开始的版权声明后应加入如下版本标记：

文件：id

版本：$Revision$

这样的标记在提交给配置管理服务器后，会自动适配成为相应的字符串，如：

文件：$Id: ussp.py,v 1.22 2004/07/21 04:47:41 hd Exp $

版本：$Revision: 1.4 $

这些应该包含在模块的文档字符串之后，所有代码之前，上下用一个空行分割。

命名约定

以下的命名风格是众所周知的。

- b （单个小写字母）
- B （单个大写字母）
- 小写串 如：getname

- 带下画线的小写字符串如：_getname
- 大写串 如：GETNAME
- 带下画线的大写字符串如：_GETNAME
- CapitalizedWords（首字母大写单词串）
- mixedCase（混合大小写单词串）
- Capitalized_Words_With_Underscores（带下画线的首字母大写单词串）

另外，以下用下画线作前导或结尾的特殊形式是被公认的（这些通常可以和任何习惯一起使用）。

- _single_leading_underscore（以一个下画线作前导）：弱的“内部使用（internal use）”标志。例如，“from M import *”不会导入以下画线开头的对象。
- single_trailing_underscore_（以一个下画线结尾）：用于避免与 Python 关键词的冲突，例如“Tkinter.Toplevel(master, class_='ClassName')”。
- __double_leading_underscore （双下画线）：从 Python 1.4 起为类私有名。
- __double_leading_and_trailing_underscore__：特殊的（magic） 对象或属性，存在于用户控制的（user-controlled）名字空间，例如：__init__、__import__ 或 __file__。有时它们被用户定义，用于触发某个特殊行为（magic behavior）（例如：运算符重载）；有时被构造器（infrastructure）插入，以便自己使用或为了调试。因此，在未来的版本中，构造器（也可定义为 Python 解释器和标准库）可能打算建立自己的魔法属性列表，用户代码通常应该限制将这种约定作为己用。成为构造器的一部分的用户代码可以在下画线中结合使用短前缀，例如：__bobo_magic_attr__。

应避免的名字则有：永远不要用字符 l（小写字母 el（就是读音，下同））、O（大写字母 oh），或者 I（大写字母 eye）作为单字符的变量名。在某些字体中，这些字符不能与数字 1 和 0 分开。当想要使用 l 时，用 L 代替它。

模块名：模块应该是不含下画线的、简短的、小写的名字。因为模块名被映射到文件名，有些文件系统大小写不敏感并且截短长名字，模块名被设为相当短是重要的——这在 Unix 上不是问题，但当代码传到 Mac 或 Windows 上就可能是个问题了。Python 包应该是不含下画线的，简短的，全小写的名字。

类名：几乎没有例外，类名总是使用首字母大写单词串（CapWords）的约定。

异常名：如果模块对所有情况定义了单个异常，它通常被叫做“error”或“Error”。似乎内建的模块使用“error”（例如：os.error），而 Python 模块通常用“Error”（例如：xdrlib.Error）。

全局变量名：一般全部大写字母命名。

函式名：函式名应该为小写，可用下画线风格单词以增加可读性。

方法名和实例变量：这大体上和函式相同，通常使用小写单词，必要时用下画线分隔增加可读性。使用一个前导下画线仅用于不打算作为类的公共接口的内部方法和实例变量。Python 不强制要求这样，它取决于程序员是否遵守这个约定。使用两个前导下画线以表示类私有的名字。Python 将这些名字和类名连接在一起：如果类 Foo 有一个属性名为__a，它不能以 Foo.__a 访问。通常，双前导下画线应该只用来避免与类（为可以子类化所设计）中的属性发生名字冲突。

继承的设计：始终要确定一个类中的方法和实例变量是否要被公开。通常，永远不要将数据变量公开，除非你实现的本质只是记录。人们总是更喜欢给类提供一个函式的接口作为替换。同样，确定你的属性是否应为私有的。私有与非公有的区别在于： 前者永远不会被用在一个派生类中，而后者可能会。私有属性必须有两个前导下画线，无后置下画线。非公有属性必须有一个前导下画线，无后置下画线。公共属性没有前导和后置下画线，除非它们与保留字冲突，在此情况下，单个后置下画线比前置或混乱地拼写要好，例如：class_优于 klass。

其他建议

要对像 None 之类的单值进行比较，应该永远用：is 或 is not 来做。当你本意是 if x is not None 时，写成 if x 要小心，例如当你测试一个默认为 None 的变量或参数是否被设置为其他值时。这个其他值可能是一个在布尔上下文中为假的值!

基于类的异常总是好过基于字符串的异常。模块和包应该定义它们自己的域内特定的基异常类，基类应该是内建的 Exception 类的子类。还始终包含一个类的文档字符串。例如：

```
class MessageError(Exception):
      """Base class for errors in the email package."""
```

应当使用字符串方法代替字符串模块，除非必须向后兼容 Python 2.0 以前的版本。字符串方法运行总是非常快，而且和 unicode 字符串共用同样的 API（应用程序接口）

在检查前缀或后缀时避免对字符串进行切片。用 startswith()和 endswith()代替，因为它们是明确的并且错误更少。例如：

```
No:  if foo[:3] == 'bar':
Yes: if foo.startswith('bar'):
```

对象类型的比较应该始终用 isinstance()代替直接比较类型。例如：

```
No:  if type(obj) is type(1):
Yes: if isinstance(obj, int):
```

检查一个对象是否是字符串时，紧记它也可能是 unicode 字符串！在 Python 2.3、str 和 unicode 有公共的基类（basestring），所以你可以这样做：

```
    if isinstance(obj, basestring):
```

对序列（字符串（strings）、列表（lists）、元组（tuples））使用空列表是 false 这个事实，因此 if not seq 或 if seq 比 if len（seq）或 if not len（seq）好。

书写字符串文字时不要有意后置空格。这种后置空格在视觉上是不可辨认的，并且有些编辑器会将它们修整掉。

不要用 == 来比较布尔型的值以确定是 True 或 False。

```
    No:  if greeting == True:
    Yes: if greeting:
    No:  if greeting == True:
    Yes: if greeting:
```