Python 开发编码规范

这篇文档所给出的编码约定适用于在主要的 Python 发布版本中组成标准库的 Python 代码，请查阅相关的关于在 Python 的 C 实现中 C 代码风格指南的描述。

这篇文档改编自 Guido 最初的《Python 风格指南》一文，并从《Barry's style guide》中添加了部分内容。在有冲突的地方，Guide 的风格规则应该是符合本 PEP 的意图 (译注：指当有冲突时，应以 Guido 风格为准)。这篇 PEP 仍然尚未完成 (实际上，它可能永远都不会完成)。

在这篇风格指导中的一致性是重要的。在一个项目内的一致性更重要。在一个模块或函数内的一致性最重要。但最重要的是：知道何时会不一致——有时只是没有实施风格指导。当出现疑惑时，运用你的最佳判断，看看别的例子，然后决定怎样看起来更好。并且要不耻下问！

打破一条既定规则的两个好理由：

(1) 当应用这个规则是将导致代码可读性下降，即便对某人来说，他已经习惯于按这条规则来阅读代码了。

(2) 为了和周围的代码保持一致而打破规则 (也许是历史原因)，虽然这也是个清除其它混乱的好机会 (真正的 XP 风格)。

代码的布局

缩进

使用 Emacs 的 Python-mode 的默认值： 4 个空格一个缩进层次。对于确实古老的代码，你不希望产生混乱，可以继续使用 8 空格的制表符 (8-space tabs)。Emacs Python-mode 自动发现文件中主要的缩进层次，依此设定缩进参数。

制表符还是空格

永远不要混用制表符和空格。最流行的 Python 缩进方式是仅使用空格，其次是仅使用制表符，混合着制表符和空格缩进的代码将被转换成仅使用空格。(在 Emacs 中，选中整个缓冲区，按 ESC-x 去除制表符。) 调用 Python 命令行解释器时使用 -t 选项，可对代码中不合法得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。

对于新的项目，强烈推荐仅使用空格而不是制表符。许多编辑器拥有使之易于实现的功能(在 Emacs 中，确认 indent-tabs-mode 是 nil) 。

行的最大长度

周围仍然有许多设备被限制在每行 80 字符：而且，窗口限制在 80 个字符。使将多个窗口并排放置成为可能。在这些设备上使用默认的折叠方式看起来有点丑陋。因此，请将所有行限制在最大 79 字符 (Emacs 准确得将行限制为长 80 字符)，对顺序排放的大块文本 (文档字符串或注释)，推荐将长度限制在 72 字符。

折叠长行的首选方法是使用 Pyhon 支持的圆括号，方括号和花括号内的行延续。如果需要，你可以在表达式周围增加一对额外的圆括号，但是有时使用反斜杠看起来更好，确认恰当得缩进了延续的行。

Emacs 的 Python-mode 正确得完成了这些。一些例子：

```
#!Python
    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数和类的定义，　　类内方法的定义用单个空行分割，　　额外的空行可被用于 (保守的)分割相关函数组成的群，　　在一组相关的单句中间可以省略空行。　　(例如： 一组哑元素)。

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。在函数中使用空行时，请谨慎的用于表示一个逻辑段落。　　Python 接受 contol-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，因此在你的文件中，可以用他们来为相关片段分页。

编码

Python 核心发布中的代码必须始终使用　　ASCII 或 Latin-1 编码 (又名 ISO-8859-1)，使用 ASCII 的文件不必有编码 cookie，Latin-1 仅当注释或文档字符串涉及作者名字需要　　Latin-1 时才被使用：

另外使用 \x 转义字符是在字符串中包含非　　ASCII(non-ASCII) 数据的首选方法。

作为 PEP 263 实现代码的测试套件的部分文件是个例外。

导入

通常应该在单独的行中导入 (Imports)，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：

1、标准库的导入 (Imports)

2、相关的主包 (major package)的导入 (即，所有的 email 包在随后导入)

3、特定应用的导入 (imports)

你应该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导入的，　　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

即使用 "MyClass.MyClass" 和 "foo.bar.YourClass.YourClass"

表达式和语句中的空格

Guido 不喜欢在以下地方出现空格：

紧挨着圆括号，方括号和花括号的，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，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" 。

紧贴着函数调用的参数列表前开式括号 (open parenthesis)的，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，如：

"dict ['key'] = list [index]" 。要始终将它写成 "dict['key'] = list[index]" 。

在赋值 (或其它)运算符周围的用于和其它并排的一个以上的空格，如：

```
#!Python
        x= 1
        y= 2
        long_variable = 3
```

要始终将它写成

```
#!Python
        x = 1
        y = 2
        long_variable = 3
```

(不要对以上任意一条和他争论——Guido 养成这样的风格超过 20 年了。)

其它建议

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

按你的看法在算术运算符周围插入空格。始终保持二元运算符两边空格的一致。

一些例子：

```
#!Python
        i = i+1
        submitted = submitted + 1
        x = x*2 - 1
        hypot2 = x*x + y*y
        c = (a+b) * (a-b)
        c = (a + b) * (a - b)
```

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，例如：

```
#!Python
        def complex(real ， imag=0。 0)：
            return magic(r=real ， i=imag)
```

不要将多条语句写在同一行上：

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()

注释

同代码不一致的注释比没注释更差。当代码修改时，始终优先更新注释 ! 注释应该是完整的句子， 如果注释是一个短语或句子， 首字母应该大写， 除非他是一个以小写字母开头的标识符 (永远不要修改标识符的大小写)。

如果注释很短， 最好省略末尾的句号。 注释块通常由一个或多个由完整句子构成的段落组成，每个句子应该以句号结尾。你应该在句末，句号后使用两个空格，以便使 Emacs 的断行和填充工作协调一致。

用英语书写时，断词和空格是可用的。非英语国家的 Python 程序员：请用英语书写你的注释，除非你 120%的确信这些代码不会被不懂你的语言的人阅读。

注释块

注释块通常应用于跟随着一些 (或者全部) 代码并和这些代码有着相同的缩进层次。注释块中每行以‘ # ’和一个空格开始 (除非他是注释内的缩进文本)。注释块内的段落以仅含单个‘ # ’的行分割。注释块上下方最好有一空行包围 (或上方两行下方一行，对一个新函数定义段的注释)。

行内注释

一个行内注释是和语句在同一行的注释， 行内注释应该谨慎适用， 行内注释应该至少用两个空格和语句分开，它们应该以 '#和单个空格开始。

```
x = x+1                 # Increment x
```

如果语意是很明了的， 那么行内注释是不必要的， 事实上是应该被移除的。 不要这样写：

```
x = x+1                 # Increment x
x = x+1                 # Compensate for border
```

但是有时，这样是有益的：

```
x = x+1                 # Compensate for border
```

文档字符串

应该一直遵守编写好的文档字符串的约定 PEP 257 [3] 。为所有公共模块，函数，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。这个注释应该在 "def" 这行后。

PEP 257 描述了好的文档字符串的约定。一定注意，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

如果你要将 RCS 或 CVS 的杂项 (crud) 包含在你的源文件中，按如下做。

#!Python

__version__ = "$Revision ： 1。4 $"

$Source： E：/cvsroot/Python_doc/pep8 。txt， v $

这个行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

Python 库的命名约定有点混乱，所以我们将永远不能使之变得完全一致，不过还是有公认的命名规范的。新的模块和包 (包括第三方的框架)必须符合这些标准，但对已有的库存在不同风格的，保持内部的一致性是首选的。

描述：命名风格

有许多不同的命名风格。以下的有助于辨认正在使用的命名风格，独立于它们的作用。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)

Lowercase（小写）

lower_case_with_underscores（有下划线的小写）

UPPERCASE （大写）

UPPER_CASE_WITH_UNDERSCORES （有下划线的大写）

CapitalizedWords (或 CapWords，CamelCase 这样命名是因为可从字母的大小写分出单词。这有时也被当作 StudlyCaps。

mixedCase (与 CapitalizedWords 的不同在于首字母小写 !)

Capitalized_Words_With_Underscores （有下划线的首字母大写） (丑陋 !)

还有用短的特别前缀将相关的名字聚合在一起的风格。这在 Python 中不常用，但是出于完整性要提一下，例如， os.stat()函数返回一个元组，他的元素传统上说名如 st_mode，st_size， st_mtime 等等。

X11 库的所有公开函数以 X 开头。(在 Python 中，这个风格通常认为是不必要的，因为属性和方法名以对象作前缀，而函数名以模块名作前缀。)

另外，以下用下划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组合)：

_single_leading_underscore(单个下划线作前导) :弱的“ 内部使用 (internal use) ”标志。 (例如，“ from M import * ”不会导入以下划线开头的对象)。

single_trailing_underscore_(单个下划线结尾)： 用于避免与 Python 关键词的冲突， 例如：“ Tkinter.Toplevel(master ， class_='ClassName') ”。

_double_leading_underscore(双下划线)：从 Python 1.4 起为类私有名。

_double_leading_and_trailing_underscore_ ：“ 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魔法行为 (例如：运算符重载)：有时被构造器插入，以便自己使用或为了调试。因此，在未来的版本中， 构造器 (松散得定义为 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。 欲成为构造器的一部分的用户代码可以在下滑线中结合使用短前缀，例如：

_bobo_magic_attr__ 。

说明：命名约定

应避免的名字。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‘ 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 分辨。试着在使用‘ l ’时用‘ L ’代替。

模块名

模块应该是不含下划线的， 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，模块名被选为相当短是重要的，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。

当用 C 或 C++ 编写的扩展模块有一个伴随 Python 模块提供高层 (例如进一步的面向对象)接口时， C/C++ 模块有下划线前导 (如： _socket)。 Python 包应该是不含下划线的，简短的，全小写的名字。

类名

几乎不出意料，类名使用 CapWords 约定。内部使用的类外加一个前导下划线。

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 (扩展)的模块使用 “ error ”(例如： os.error)，而 Python 模块通常用 “ Error ” (例如： xdrlib.Error) 。趋势似乎是倾向使用 CapWords 异常名。

全局变量名

(让我们祈祷这些变量仅在一个模块的内部有意义)

这些约定和在函数中的一样。模块是被设计为通过“ from M import * ”来使用的，必须用一个下划线作全局变量 (及内部函数和类)的前缀防止其被导出 (exporting) 。

函数名

函数名应该为小写，可能用下划线风格单词以增加可读性。 mixedCase 仅被允许用于这种风格已经占优势的上下文 (如： threading.py)，以便保持向后兼容。

方法名和实例变量

这段大体上和函数相同： 通常使用小写单词， 必要时用下划线分隔增加可读性。 仅为不打算作为类的公共界面的内部方法和实例使用一个前导下划线， Python 不强制要求这样：它取决于程序员是否遵守这个约定。

使用两个前导下划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。 (固执的用户还是可以通过 Foo._Foo__a 得到访问权。)

通常双前导下划线仅被用于避免含子类的类中的属性名的名字冲突。

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，除非你实现的本质上只是记录，人们几乎总是更喜欢代之给出一个函数作为类的界面 (Python 2.2 的一些开发者在这点上做得非常漂亮)。

同样， 确定你的属性是否应为私有的。 私有和非私有的区别在于模板将永远不会对原有的类 (导出类)有效，而后者可以。你应该在大脑中就用继承设计好了你的类，私有属性必须

说明：命名约定

应避免的名字。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‘ 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 分辨。试着在使用‘ l ’时用‘ L ’代替。

模块名

模块应该是不含下划线的， 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，模块名被选为相当短是重要的，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。

当用 C 或 C++ 编写的扩展模块有一个伴随 Python 模块提供高层 (例如进一步的面向对象)接口时， C/C++ 模块有下划线前导 (如： _socket)。Python 包应该是不含下划线的，简短的，全小写的名字。

类名

几乎不出意料，类名使用 CapWords 约定。内部使用的类外加一个前导下划线。

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 (扩展) 的模块使用 “ error ”(例如： os.error)，而 Python 模块通常用 “ Error ” (例如： xdrlib.Error) 。趋势似乎是倾向使用 CapWords 异常名。

全局变量名

(让我们祈祷这些变量仅在一个模块的内部有意义)

这些约定和在函数中的一样。模块是被设计为通过“ from M import * ”来使用的，必须用一个下划线作全局变量 (及内部函数和类)的前缀防止其被导出 (exporting) 。

函数名

函数名应该为小写，可能用下划线风格单词以增加可读性。 mixedCase 仅被允许用于这种风格已经占优势的上下文 (如： threading.py)，以便保持向后兼容。

方法名和实例变量

这段大体上和函数相同： 通常使用小写单词， 必要时用下划线分隔增加可读性。 仅为不打算作为类的公共界面的内部方法和实例使用一个前导下划线， Python 不强制要求这样：它取决于程序员是否遵守这个约定。

使用两个前导下划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。 (固执的用户还是可以通过 Foo._Foo__a 得到访问权。)

通常双前导下划线仅被用于避免含子类的类中的属性名的名字冲突。

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，除非你实现的本质上只是记录，人们几乎总是更喜欢代之给出一个函数作为类的界面 (Python 2.2 的一些开发者在这点上做得非常漂亮)。

同样， 确定你的属性是否应为私有的。 私有和非私有的区别在于模板将永远不会对原有的类 (导出类)有效，而后者可以。你应该在大脑中就用继承设计好了你的类，私有属性必须