  3.  [Python 编码风格方面的准则](http://www.elias.cn/Python/PythonStyleGuide?from=Develop.PythonStyleGuide#toc24)

  3.1  [分号](http://www.elias.cn/Python/PythonStyleGuide?from=Develop.PythonStyleGuide#toc25)

  3.2  [每行长度](http://www.elias.cn/Python/PythonStyleGuide?from=Develop.PythonStyleGuide#toc26)

  3.3  [圆括号](http://www.elias.cn/Python/PythonStyleGuide?from=Develop.PythonStyleGuide#toc27)

  3.4  [缩进](http://www.elias.cn/Python/PythonStyleGuide?from=Develop.PythonStyleGuide#toc28)

  3.5  [空行](http://www.elias.cn/Python/PythonStyleGuide?from=Develop.PythonStyleGuide#toc29)

  3.6  [空格](http://www.elias.cn/Python/PythonStyleGuide?from=Develop.PythonStyleGuide#toc30)

  3.7  [Python 解释器](http://www.elias.cn/Python/PythonStyleGuide?from=Develop.PythonStyleGuide#toc31)

  3.8  [注释](http://www.elias.cn/Python/PythonStyleGuide?from=Develop.PythonStyleGuide#toc32)

  3.9  [类](http://www.elias.cn/Python/PythonStyleGuide?from=Develop.PythonStyleGuide#toc33)

  3.10  [字符串](http://www.elias.cn/Python/PythonStyleGuide?from=Develop.PythonStyleGuide#toc34)

  3.11  [TODO style](http://www.elias.cn/Python/PythonStyleGuide?from=Develop.PythonStyleGuide" \l "toc35)

  3.12  [import 分组及顺序](http://www.elias.cn/Python/PythonStyleGuide?from=Develop.PythonStyleGuide#toc36)

  3.13  [语句](http://www.elias.cn/Python/PythonStyleGuide?from=Develop.PythonStyleGuide#toc37)

  3.14  [访问控制](http://www.elias.cn/Python/PythonStyleGuide?from=Develop.PythonStyleGuide#toc38)

  3.15  [命名](http://www.elias.cn/Python/PythonStyleGuide?from=Develop.PythonStyleGuide#toc39)

  3.16  [程序入口](http://www.elias.cn/Python/PythonStyleGuide?from=Develop.PythonStyleGuide#toc40)

  3.17  [总结](http://www.elias.cn/Python/PythonStyleGuide?from=Develop.PythonStyleGuide#toc41)

# 3.  Python 编码风格方面的准则

[(Edit)](http://www.elias.cn/Python/PythonStyleGuide?action=edit&s=25&auto=y&from=Python.PythonStyleGuide" \o "edit section:  分号 )

## 3.1  分号

不要用分号作为你的行结束符，也不要利用分号在同一行放两个指令。

[(Edit)](http://www.elias.cn/Python/PythonStyleGuide?action=edit&s=26&auto=y&from=Python.PythonStyleGuide" \o "edit section:  每行长度 )

## 3.2  每行长度

一行最多可以有80个字符。

**例外：** 导入模块的行可以超过80个字符再结束。

确保 Python 隐含的连接行（line joining）放在圆括号、方括号或大括号之间。如果需要的话，你可以在表达式两头放一堆额外的圆括号。

**Yes：**

fooBar(self, width, height, color='black', design=None, x='foo',  
       emphasis=None, highlight=0)  
  
  
**if** ((width == 0) **and** (height == 0)  
    **and** (color == 'red') **and** (emphasis == 'strong')):

[[$[Get Code]]](http://www.elias.cn/Python/PythonStyleGuide?action=sourceblock&num=24)

当表示文本的字符串（literal string）一行放不下的时候，用圆括号把隐含的连接行括起来。

x = ('This will build a very long long '  
     'long long long long long long string')

[[$[Get Code]]](http://www.elias.cn/Python/PythonStyleGuide?action=sourceblock&num=25)

注意上例中连续行中元素的缩进，参考[缩进](http://www.elias.cn/Python/PythonStyleGuide?from=Develop.PythonStyleGuide" \l "Indentation)章节的解释。

[(Edit)](http://www.elias.cn/Python/PythonStyleGuide?action=edit&s=27&auto=y&from=Python.PythonStyleGuide" \o "edit section:  圆括号 )

## 3.3  圆括号

吝啬地使用圆括号。在以下情况下别用：

* 在 return 语句中。
* 在条件判断语句中。除非是用圆括号来暗示两行是连在一起的（参见上一节）。
* 在元组（tuple）周围。除非因为语法而不得不加或是为了显得更清晰。

但在下列情况下是可以用圆括号的：

* 暗示两行是连在一起的。
* 用来括起长表达式中的子表达式（包括子表达式是条件判断语句一部分的情形）。

实际上，在子表达式周围用圆括号比单纯依赖运算符优先级要好。

**Yes：**

**if** foo:  
  
**while** x:  
  
**if** x **and** y:  
  
**if** **not** x:  
  
**if** (x < 3) **and** (**not** y):  
  
**return** foo  
  
**for** x, y **in** dict.items():  
  
x, (y, z) = funcThatReturnsNestedTuples()

[[$[Get Code]]](http://www.elias.cn/Python/PythonStyleGuide?action=sourceblock&num=26)

**No：**

**if** (x):  
  
**while** (x):  
  
**if** **not**(x):  
  
**if** ((x < 3) **and** (**not** y)):  
  
**return** (foo)  
  
**for** (x, y) **in** dict.items():  
  
(x, (y, z)) = funcThatReturnsNestedTuples()

[[$[Get Code]]](http://www.elias.cn/Python/PythonStyleGuide?action=sourceblock&num=27)

[(Edit)](http://www.elias.cn/Python/PythonStyleGuide?action=edit&s=28&auto=y&from=Python.PythonStyleGuide" \o "edit section:  缩进 )

## 3.4  缩进

**注意和**[**PEP8**](http://www.python.org/dev/peps/pep-0008/)**不同，这里遵循作为本文起源的原始 Google Python 编码风格指南。**

用两空格缩进代码块。不要用 tab 或者混用 tab 和空格。如果要暗示两行相连，要么就把被包装的元素纵向对其，就像“每行长度”章节中的例子那样；要么就用**4**个空格（不是两个，这样可以和后面紧跟着的嵌套代码块区分开，避免混淆）作悬挂缩进（hanging indent），在这种情况下，首行不应该放任何参数。

**Yes：**

*# 与起始定界符对齐：*  
foo = longFunctionName(var\_one, var\_two,  
                       var\_three, var\_four)  
  
*# 4空格悬挂缩进，而且首行空着：*  
foo = longFunctionName(  
    var\_one, var\_two, var\_three,  
    var\_four)

[[$[Get Code]]](http://www.elias.cn/Python/PythonStyleGuide?action=sourceblock&num=28)

**No：**

*# 不应该在首行塞东西：*  
foo = longFunctionName(var\_one, var\_two,  
    var\_three, var\_four)  
  
*# 不应该用两空格的悬挂缩进：*  
foo = longFunctionName(  
  var\_one, var\_two, var\_three,  
  var\_four)

[[$[Get Code]]](http://www.elias.cn/Python/PythonStyleGuide?action=sourceblock&num=29)

[(Edit)](http://www.elias.cn/Python/PythonStyleGuide?action=edit&s=29&auto=y&from=Python.PythonStyleGuide" \o "edit section:  空行 )

## 3.5  空行

在顶级定义（可以是函数或者类定义）之间加两个空行。在方法定义之间以及“ class ”那一行与第一个方法之间加一个空行。在[\_\_doc\_\_ string](http://www.elias.cn/Python/PythonStyleGuide?from=Develop.PythonStyleGuide" \l "Doc_strings)和它后面的代码之间加一个空行。在函数和方法内部你认为合适的地方加一个空行。

在文件最后总是加一个空行，这可以避免很多 diff 工具生成“No newline at end of file”信息。

[(Edit)](http://www.elias.cn/Python/PythonStyleGuide?action=edit&s=30&auto=y&from=Python.PythonStyleGuide" \o "edit section:  空格 )

## 3.6  空格

在圆括号、方括号、大括号里面不要加空格。

**Yes：** spam(ham[1], {eggs: 2}, [])

**No：** spam( ham[ 1 ], { eggs: 2 }, [ ] )

在逗号、分号、冒号前面不要加空格。逗号、分号、冒号后面**必须**加空格，除非那是行尾。

**Yes：**

**if** x == 4:  
  
**print** x, y  
  
x, y = y, x

[[$[Get Code]]](http://www.elias.cn/Python/PythonStyleGuide?action=sourceblock&num=30)

**No：**

**if** x == 4 :  
  
**print** x , y  
  
x , y = y , x

[[$[Get Code]]](http://www.elias.cn/Python/PythonStyleGuide?action=sourceblock&num=31)

在表示参数、列表、下标、分块开始的圆括号/方括号前面不要加空格。

**Yes：** spam(1)

**No：** spam (1)

**Yes：** dict['key'] = list[index]

**No：** dict ['key'] = list [index]

在二元运算符两边各家一个空格，包括：赋值（=）、比较（==、<、>、!=、<>、<=、>=、in、not in、is、is not）、以及布尔运算符（and、or、not）。你肯定能判断出是否应该在算术运算符周围加空格，因为在二元运算符两边加空格的原则总是一致的。

**Yes：** x == 1

**No：** x<1

等号（“=”）用于指名参数或默认参数值时，两边不要加空格。

**Yes：** def Complex(real, imag=0.0): return Magic(r=real, i=imag)

**No：** def Complex(real, imag = 0.0): return Magic(r = real, i = imag)

[(Edit)](http://www.elias.cn/Python/PythonStyleGuide?action=edit&s=31&auto=y&from=Python.PythonStyleGuide" \o "edit section:  Python 解释器 )

## 3.7  Python 解释器

模块开头应该是一个“shebang”行，用于指定执行此程序的 Python 解释器：

*#!/usr/bin/python2.5*

[[$[Get Code]]](http://www.elias.cn/Python/PythonStyleGuide?action=sourceblock&num=32)

[Google App Engine 要求使用 Python 2.5](http://code.google.com/appengine/docs/python/purepython.html)。

[(Edit)](http://www.elias.cn/Python/PythonStyleGuide?action=edit&s=32&auto=y&from=Python.PythonStyleGuide" \o "edit section:  注释 )

## 3.8  注释

[(Edit)](http://www.elias.cn/Python/PythonStyleGuide?action=edit&s=33&auto=y&from=Python.PythonStyleGuide" \o "edit section:  Doc strings )

### Doc strings

Python 有一种独特的注释风格称为 \_\_doc\_\_ String。包、模块、类、函数的第一个语句如果是字符串那么就是一个 \_\_doc\_\_ String。这种字符串可以用对象的 \_\_doc\_\_() 成员函数自动提取，而且会被用于 pydoc。（试着在你的模块上运行 pydoc 来看看它是怎么工作的。）我们对 \_\_doc\_\_String 的约定是：用三个双引号把字符串括起来。 \_\_doc\_\_ String 应该这样组织：首先是一个以句号结束的摘要行（实实在在的一行，[不多于80个字符](http://www.elias.cn/Python/PythonStyleGuide?from=Develop.PythonStyleGuide" \l "Line_length)），然后接一个空行，再接 \_\_doc\_\_String 中的其他内容，并且这些文字的起始位置要和首行的第一个双引号在同一列。下面几节是关于 \_\_doc\_\_ String 格式更进一步的原则

[(Edit)](http://www.elias.cn/Python/PythonStyleGuide?action=edit&s=34&auto=y&from=Python.PythonStyleGuide" \o "edit section:  模块 )

### 模块

每个文件开头都应该包含一个带有版权信息和许可声明的块注释。

[(Edit)](http://www.elias.cn/Python/PythonStyleGuide?action=edit&s=35&auto=y&from=Python.PythonStyleGuide" \o "edit section:  版权和许可声明 )

#### 版权和许可声明

*#!/usr/bin/python2.5*  
*#*  
*# Copyright [current year] the Melange authors.*  
*#*  
*# Licensed under the Apache License, Version 2.0 (the "License");*  
*# you may not use this file except in compliance with the License.*  
*# You may obtain a copy of the License at*  
*#*  
*#   http://www.apache.org/licenses/LICENSE-2.0*  
*#*  
*# Unless required by applicable law or agreed to in writing, software*  
*# distributed under the License is distributed on an "AS IS" BASIS,*  
*# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.*  
*# See the License for the specific language governing permissions and*  
*# limitations under the License.*

[[$[Get Code]]](http://www.elias.cn/Python/PythonStyleGuide?action=sourceblock&num=33)

然后接着写描述模块内容的\_\_doc\_\_ String，其中作者信息应该紧跟着许可声明。如果还给出了作者邮件地址，那添加到 \_\_authors\_\_ 列表的整个作者字符串应该就是一个[与 RFC 2821 兼容的](http://www.ietf.org/rfc/rfc2821.txt)电子邮件地址。

[(Edit)](http://www.elias.cn/Python/PythonStyleGuide?action=edit&s=36&auto=y&from=Python.PythonStyleGuide" \o "edit section:  模块头及作者信息)

#### 模块头及作者信息

"""用一行文字概述模块或脚本，用句号结尾。  
  
留一个空行。本 \_\_doc\_\_ string 的其他部分应该包括模块或脚本的全面描述。作为可选项，还可以包括导出的类和函数的简要描述。  
  
  ClassFoo: 一行概述。  
  functionBar(): 一行概述。  
"""  
  
\_\_authors\_\_ = [  
  *# 请按照姓氏字母顺序排列：*  
  '"John Smith" <johnsmith@example.com>',  
  '"Joe Paranoid" <joeisgone@example.com>',  *# 应提供电子邮件地址*  
]

[[$[Get Code]]](http://www.elias.cn/Python/PythonStyleGuide?action=sourceblock&num=34)

如果新的贡献者还没添加到[AUTHORS file](http://soc.googlecode.com/svn/trunk/AUTHORS)中，那应该在该贡献者第一次向版本控制工具提交代码时同时把他/她添加到这个文件里。

[(Edit)](http://www.elias.cn/Python/PythonStyleGuide?action=edit&s=37&auto=y&from=Python.PythonStyleGuide" \o "edit section:  函数和方法 )

### 函数和方法

如果不是用途非常明显而且非常短的话，所有函数和方法都应该有 \_\_doc\_\_ string 。此外，所有外部能访问的函数和方法，无论有多短、有多简单，都应该有 \_\_doc\_\_ string 。 \_\_doc\_\_ string 应该包括函数能做什么、输入数据的具体描述（“Args:”）、输出数据的具体描述（“Returns:”、“Raises:”、或者“Yields:”）。 \_\_doc\_\_ string 应该能提供调用此函数相关的足够信息，而无需让使用者看函数的实现代码。如果参数要求特定的数据类型或者设置了参数默认值，那 \_\_doc\_\_ string 应该明确说明这两点。“Raises:”部分应该列出此函数可能抛出的所有异常。生成器函数的 \_\_doc\_\_ string 应该用“Yields:”而不是Returns:。

函数和方法的 \_\_doc\_\_ string 一般不应该描述实现细节，除非其中涉及非常复杂的算法。在难以理解的代码中使用块注释或行内注释会是更合适的做法。

**def** fetchRows(table, keys):  
  """取出表中的多行内容。  
  
  Args:  
    table: 打开的表。 Table 类的实例。  
    keys: 字符串序列，表示要从表中取出的行的键值。  
  
  Returns:  
    一个字典，映射指定键值与取出的表中对应行的数据：  
  
    {'Serak': ('Rigel VII', 'Preparer'),  
     'Zim': ('Irk', 'Invader'),  
     'Lrrr': ('Omicron Persei 8', 'Emperor')}  
  
    如果 keys 参数中的键值没有出现在字典里，就表示对应行在表中没找到。  
  
  Raises:  
    IOError: 访问 table.Table 对象时发生的错误。  
  """  
  **pass**

[[$[Get Code]]](http://www.elias.cn/Python/PythonStyleGuide?action=sourceblock&num=35)

[(Edit)](http://www.elias.cn/Python/PythonStyleGuide?action=edit&s=38&auto=y&from=Python.PythonStyleGuide" \o "edit section:  类 )

### 类

类应该在描述它的类定义下面放 \_\_doc\_\_ string 。如果你的类有公开属性值，那应该在 \_\_doc\_\_string 的 Attributes: 部分写清楚。

**class** SampleClass(object):  
  
  """这里是类的概述。  
  
  详细的描述信息……  
  详细的描述信息……  
  
  Attributes:  
    likes\_spam: 布尔型，表示我们是否喜欢垃圾邮件。  
    eggs: 整数，数我们下了多少蛋。  
  """  
  
  **def** \_\_init\_\_(self, likes\_spam=False):  
    """拿点什么来初始化 SampleClass 。  
  
    Args:  
      likes\_spam: 初始化指标，表示 SampleClass 实例是否喜欢垃圾邮件（默认是 False）。  
    """  
    self.likes\_spam = likes\_spam  
    self.eggs = 0  
  
  **def** publicMethod(self):  
    """执行一些操作。"""  
    **pass**

[[$[Get Code]]](http://www.elias.cn/Python/PythonStyleGuide?action=sourceblock&num=36)

[(Edit)](http://www.elias.cn/Python/PythonStyleGuide?action=edit&s=39&auto=y&from=Python.PythonStyleGuide" \o "edit section:  块注释及行内注释 )

### 块注释及行内注释

加注释的最后一个位置是在难以理解的代码里面。如果你打算在下一次代码复查（code review）的时候解释这是什么意思，那你应该现在就把它写成注释。在开始进行操作之前，就应该给复杂的操作写几行注释。对不直观的代码则应该在行末写注释。

*# 我们用带权的字典检索来查找 i 在数组中的位置。我们根据数组中最大的数和*  
*# 数组的长度来推断可能的位置，然后做二分法检索找到准确的值。*  
  
**if** i & (i-1) == 0:        *# 当且仅当 i 是 2 的幂时，值为 true*

[[$[Get Code]]](http://www.elias.cn/Python/PythonStyleGuide?action=sourceblock&num=37)

这些注释应该和代码分开才更易读。在块注释值钱应该加一个空行。一般行末注释应该和代码之间至少空两个格。如果连续几行都有行末注释（或者是在一个函数中），**可以**把它们纵向对齐，但这不是必须的。

另一方面，不要重复描述代码做的事。假设正在读代码的人比你更懂 Python （尽管这不是你努力的方向）。 On the other hand, never describe the code. Assume the person reading the code knows Python (though not what you're trying to do) better than you do.

*# \*不好的注释\*：现在要遍历数组 b 并且确保任何时候 i 出现时，下一个元素都是 i+1*

[[$[Get Code]]](http://www.elias.cn/Python/PythonStyleGuide?action=sourceblock&num=38)

[(Edit)](http://www.elias.cn/Python/PythonStyleGuide?action=edit&s=40&auto=y&from=Python.PythonStyleGuide" \o "edit section:  类 )

## 3.9  类

如果不从其他基类继承，那就应该明确地从 object 基类继承。这一条对嵌套类也适用。

**No：**

**class** SampleClass:  
  **pass**  
  
**class** OuterClass:  
  **class** InnerClass:  
    **pass**

[[$[Get Code]]](http://www.elias.cn/Python/PythonStyleGuide?action=sourceblock&num=39)

**Yes：**

**class** SampleClass(object):  
  **pass**  
  
**class** OuterClass(object):  
  **class** InnerClass(object):  
    **pass**  
  
**class** ChildClass(ParentClass):  
  """已经从另一个类显式继承了。"""  
  **pass**

[[$[Get Code]]](http://www.elias.cn/Python/PythonStyleGuide?action=sourceblock&num=40)

从 object 继承是为了让类属性能够正常工作，这会避免我们一旦切换到 Python 3000 时，打破已经习惯了的特有风格。同时这也定义了一些特殊函数，来实现对象（object）的默认语义，包括：\_\_new\_\_、\_\_init\_\_、\_\_delattr\_\_、\_\_getattribute\_\_、\_\_setattr\_\_、\_\_hash\_\_、\_\_repr\_\_、和 \_\_str\_\_。

[(Edit)](http://www.elias.cn/Python/PythonStyleGuide?action=edit&s=41&auto=y&from=Python.PythonStyleGuide" \o "edit section:  字符串 )

## 3.10  字符串

应该用 % 运算符来格式化字符串，即使所有的参数都是字符串。不过你也可以在 + 和 % 之间做出你自己最明智的判断。

**No：**

x = '%s%s' % (a, b)  *# 这种情况下应该用 +*  
x = imperative + ', ' + expletive + '!'  
x = 'name: ' + name + '; score: ' + str(n)

[[$[Get Code]]](http://www.elias.cn/Python/PythonStyleGuide?action=sourceblock&num=41)

**Yes：**

x = a + b  
x = '%s, %s!' % (imperative, expletive)  
x = 'name: %s; score: %d' % (name, n)

[[$[Get Code]]](http://www.elias.cn/Python/PythonStyleGuide?action=sourceblock&num=42)

应该避免在循环中用 + 或 += 来连续拼接字符串。因为字符串是不变型，这会毫无必要地建立很多临时对象，从而二次方级别的运算量而不是线性运算时间。相反，应该把每个子串放到 list 里面，然后在循环结束的时候用 ''.join() 拼接这个列表。

**No：**

employee\_table = '<table>'  
**for** last\_name, first\_name **in** employee\_list:  
  employee\_table += '<tr><td>%s, %s</td></tr>' % (last\_name, first\_name)  
employee\_table += '</table>'

[[$[Get Code]]](http://www.elias.cn/Python/PythonStyleGuide?action=sourceblock&num=43)

**Yes：**

items = ['<table>']  
**for** last\_name, first\_name **in** employee\_list:  
  items.append('<tr><td>%s, %s</td></tr>' % (last\_name, first\_name))  
items.append('</table>')  
employee\_table = ''.join(items)

[[$[Get Code]]](http://www.elias.cn/Python/PythonStyleGuide?action=sourceblock&num=44)

对多行字符串使用 """ 而不是 '''。但是注意，通常使用隐含的行连接（implicit line joining）会更清晰，因为多行字符串不符合程序其他部分的缩进风格。

**No：**

**print** """这种风格非常恶心。  
不要这么用。  
"""

[[$[Get Code]]](http://www.elias.cn/Python/PythonStyleGuide?action=sourceblock&num=45)

**Yes：**

**print** ("这样会好得多。**\n**"  
           "所以应该这样用。**\n**")

[[$[Get Code]]](http://www.elias.cn/Python/PythonStyleGuide?action=sourceblock&num=46)

[(Edit)](http://www.elias.cn/Python/PythonStyleGuide?action=edit&s=42&auto=y&from=Python.PythonStyleGuide" \o "edit section:  TODO style )

## 3.11  TODO style

在代码中使用 TODO 注释是临时的、短期的解决方案，或者说是足够好但不够完美的办法。

TODO 应该包括全部大写的字符串 TODO ，紧接用圆括号括起来的你的用户名：TODO(username) 。其中冒号是可选的。主要目的是希望有一种一致的 TODO 格式，而且可以通过用户名检索。

*# TODO(someuser): 这里应该用 "\*" 来做级联操作。*  
*# TODO(anotheruser) 用 relations 来修改这儿。*

[[$[Get Code]]](http://www.elias.cn/Python/PythonStyleGuide?action=sourceblock&num=47)

如果你的 TODO 是“在未来某个时间做某事”的形式，确保你要么包括非常确定的日期（“Fix by November 2008”）或者非常特殊的事件（“在数据库中的 Foo 实体都加上新的 fubar 属性之后删除这段代码。”）。

[(Edit)](http://www.elias.cn/Python/PythonStyleGuide?action=edit&s=43&auto=y&from=Python.PythonStyleGuide" \o "edit section:  import 分组及顺序 )

## 3.12  import 分组及顺序

应该在不同的行中做 import ：

**Yes：**

**import** sys  
**import** os

[[$[Get Code]]](http://www.elias.cn/Python/PythonStyleGuide?action=sourceblock&num=48)

**No：**

**import** sys, os

[[$[Get Code]]](http://www.elias.cn/Python/PythonStyleGuide?action=sourceblock&num=49)

import 总是放在文件开头的，也即在所有模块注释和 \_\_doc\_\_ string 的后面，在模块全局变量及常量的前面。import 应该按照从最常用到最不常用的顺序分组放置：

* import 标准库
* import 第三方库
* import Google App Engine 相关库
* import Django 框架相关库
* import SoC framework 相关库
* import 基于 SoC 框架的模块
* import 应用程序特有的内容

应该按照字母顺序排序，但所有以 from ... 开头的行都应该靠前，然后是一个空行，再然后是所有以 import ... 开头的行。以 import ... 开头的标准库和第三方库的 import 应该放在最前面，而且和其他分组隔开：

**import** a\_standard  
**import** b\_standard  
**import** a\_third\_party  
**import** b\_third\_party  
  
**from** a\_soc **import** f  
**from** a\_soc **import** g  
  
**import** a\_soc  
**import** b\_soc

[[$[Get Code]]](http://www.elias.cn/Python/PythonStyleGuide?action=sourceblock&num=50)

在 import/from 行中，语句应该按照字母顺序排序：

**from** a **import** f  
**from** a **import** g  
**from** a.b **import** h  
**from** a.d **import** e  
  
**import** a.b  
**import** a.b.c  
**import** a.d.e

[[$[Get Code]]](http://www.elias.cn/Python/PythonStyleGuide?action=sourceblock&num=51)

[(Edit)](http://www.elias.cn/Python/PythonStyleGuide?action=edit&s=44&auto=y&from=Python.PythonStyleGuide" \o "edit section:  语句 )

## 3.13  语句

一般一行只放一个语句。但你可以把测试和测试结果放在一行里面，只要这样做合适。具体来说，你不能这样写 try/except ，因为 try 和 except 不适合这样，你只可以对不带 else 的 if 这么干。

**Yes：**

**if** foo: fuBar(foo)

[[$[Get Code]]](http://www.elias.cn/Python/PythonStyleGuide?action=sourceblock&num=52)

**No：**

**if** foo: fuBar(foo)  
**else**:   fuBaz(foo)  
  
**try**:               fuBar(foo)  
**except** ValueError: fuBaz(foo)  
  
**try**:  
  fuBar(foo)  
**except** ValueError: fuBaz(foo)

[[$[Get Code]]](http://www.elias.cn/Python/PythonStyleGuide?action=sourceblock&num=53)

[(Edit)](http://www.elias.cn/Python/PythonStyleGuide?action=edit&s=45&auto=y&from=Python.PythonStyleGuide" \o "edit section:  访问控制 )

## 3.14  访问控制

如果存取器函数很简单，那你应该用公开的变量来代替它，以避免 Python 中函数调用的额外消耗。在更多功能被加进来时，你可以用 Property 特性来保持语法一致性。

另一方面，如果访问更复杂，或者访问变量的成本较为显著，那你应该用函数调用（遵循[命名准则](http://www.elias.cn/Python/PythonStyleGuide?from=Develop.PythonStyleGuide" \l "Naming)），比如 getFoo() 或者 setFoo()。如果过去的行为允许通过 Property 访问，那就别把新的存取器函数绑定到 Property 上。所有仍然企图用旧方法访问变量的代码应该是非常显眼的，这样调用者会被提醒这种改变是比较复杂的。

[(Edit)](http://www.elias.cn/Python/PythonStyleGuide?action=edit&s=46&auto=y&from=Python.PythonStyleGuide" \o "edit section:  命名 )

## 3.15  命名

[(Edit)](http://www.elias.cn/Python/PythonStyleGuide?action=edit&s=47&auto=y&from=Python.PythonStyleGuide" \o "edit section:  要避免的命名方式)

### 要避免的命名方式

* 使用单个字符命名，除非是计数器或迭代器。
* 在任何包或模块的名字里面使用减号。
* \_\_double\_leading\_and\_trailing\_underscore\_\_ 在变量开头和结尾都使用两个下划线（在 Python 内部有特殊含义）。

[(Edit)](http://www.elias.cn/Python/PythonStyleGuide?action=edit&s=48&auto=y&from=Python.PythonStyleGuide" \o "edit section:  命名约定)

### 命名约定

**注意这里某些命名约定和**[**PEP8**](http://www.python.org/dev/peps/pep-0008/)**不一样，而是遵循“Google Python 编码风格指南”的原始版本，也即本编码风格指南的起源。**

* “Internal”表示模块内部或类中的保护域（protected）和私有域。
* 变量名开头加一个下划线（\_）能对保护模块中的变量及函数提供一些支持（不会被 import \* from 导入）。
* 在实例的变量和方法开头加两个下划线（\_\_）能有效地帮助把该变量或方法变成类的私有内容（using name mangling）。
* 把模块中相关的类和顶级函数放在一起。不像 Java ，这里无需要求自己在每个模块中只放一个类。但要确保放在同一模块中的类和顶级函数是[高内聚的](http://en.wikipedia.org/wiki/Cohesion_computer_science)。
* 对类名使用驼峰式（形如 CapWords），而对模块名使用下划线分隔的小写字母（lower\_with\_under.py）。

[(Edit)](http://www.elias.cn/Python/PythonStyleGuide?action=edit&s=49&auto=y&from=Python.PythonStyleGuide" \o "edit section:  命名样例)

### 命名样例

|  |  |  |
| --- | --- | --- |
| **类别** | **公开的** | **内部的** |
| Packages | lower\_with\_under |  |
| Modules | lower\_with\_under | \_lower\_with\_under |
| Classes | CapWords | \_CapWords |
| Exceptions | CapWords |  |
| Functions | firstLowerCapWords() | \_firstLowerCapWords() |
| Global/Class Constants | CAPS\_WITH\_UNDER | \_CAPS\_WITH\_UNDER |
| Global/Class Variables | lower\_with\_under | \_lower\_with\_under |
| Instance Variables | lower\_with\_under | \_lower\_with\_under (protected) or\_\_lower\_with\_under (private) |
| Method Names \* | firstLowerCapWords() | \_firstLowerCapWords() (protected) or\_\_firstLowerCapWords() (private) |
| Function/Method Parameters | lower\_with\_under |  |
| Local Variables | lower\_with\_under |  |

\* 考虑只在首选项设置中使用[对公开属性的直接访问](http://www.elias.cn/Python/PythonStyleGuide?from=Develop.PythonStyleGuide" \l "Access_control)来作为 getters 和 setters，因为函数调用在 Python 中是比较昂贵的，而 property 之后可以用来把属性访问转化为函数调用而无需改变访问语法。

[(Edit)](http://www.elias.cn/Python/PythonStyleGuide?action=edit&s=50&auto=y&from=Python.PythonStyleGuide" \o "edit section:  程序入口 )

## 3.16  程序入口

所有的 Python 源代码文件都应该是可导入的。在 Python 中，pychecker、 pydoc、 以及单元测试都要求模块是可导入的。你的代码应该总是在执行你的主程序之前检查if \_\_name\_\_ == '\_\_main\_\_': ，这样就不会在模块被导入时执行主程序。大写 main() 是故意要和命名约定的其他部分不一致，也就是说建议写成 Main() 。

**if** \_\_name\_\_ == '\_\_main\_\_':  
  *# 参数解析*  
  main()

[[$[Get Code]]](http://www.elias.cn/Python/PythonStyleGuide?action=sourceblock&num=54)

在模块被导入时，所有顶级缩进代码会被执行。注意不要调用函数、创建对象、或者做其他在文件被 pycheck 或 pydoc 的时候不应该做的操作。

[(Edit)](http://www.elias.cn/Python/PythonStyleGuide?action=edit&s=51&auto=y&from=Python.PythonStyleGuide" \o "edit section:  总结 )

## 3.17  总结

**要一致**

如果你在编写代码，花几分钟看看你周围的代码并且弄清它的风格。如果在 if 语句周围使用了空格，那你也应该这样做。如果注释是用星号组成的小盒子围起来的，那你同样也应该这样写。

编码风格原则的关键在于有一个编程的通用词汇表，这样人们可以人民可以集中到你要说什么而不是你要怎么说。我们在这里提供全局的编码风格规则以便人们知道这些词汇，但局部风格也重要。如果你加入文件中的代码看起来和里面已经有的代码截然不同，那在读者读它的时候就会被破坏节奏。尽量避免这样。