Python编码规范（参考Goole）

## 目录: 分号

## 行长度

## 括号

## 缩进

## 空白行

异常

## 注释

## -文档字符串

## -模块

## -函数和方法

## -类

## -块注释和行注释

## Class（类）

## 字符串

## 导入格式

## 语句

## 命名

## -应该避免的名称

## -命名约定

## 分号

不要在行尾加分号, 也不要用分号将两条命令放在同一行。

## 行长度

每行不超过80个字符

以下情况除外：

1. 长的导入模块语句
2. 注释里的URL

不要使用反斜杠连接行。

Python会将 [圆括号, 中括号和花括号中的行隐式的连接起来](http://docs.python.org/2/reference/lexical_analysis.html" \l "implicit-line-joining) , 你可以利用这个特点. 如果需要, 你可以在表达式外围增加一对额外的圆括号。

推荐: foo\_bar(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')

如果一个文本字符串在一行放不下, 可以使用圆括号来实现隐式行连接:

x = ('这是一个非常长非常长非常长非常长 '

'非常长非常长非常长非常长非常长非常长的字符串')

在注释中，如果必要，将长的URL放在一行上。

Yes: # See details at

# http://www.example.com/us/developer/documentation/api/content/v2.0/csv\_file\_name\_extension\_full\_specification.html

No: # See details at

# http://www.example.com/us/developer/documentation/api/content/\

# v2.0/csv\_file\_name\_extension\_full\_specification.html

注意上面例子中的元素缩进; 你可以在本文的 :ref:`缩进 <indentation>`部分找到解释.

## 括号

宁缺毋滥的使用括号

除非是用于实现行连接, 否则不要在返回语句或条件语句中使用括号. 不过在元组两边使用括号是可以的.

## 缩进

用4个空格来缩进代码

绝对不要用tab, 也不要tab和空格混用. 对于行连接的情况, 你应该要么垂直对齐换行的元素(见 :ref:`行长度 <line\_length>` 部分的示例), 或者使用4空格的悬挂式缩进(这时第一行不应该有参数):

Yes: # 与起始变量对齐

foo = long\_function\_name(var\_one, var\_two,

var\_three, var\_four)

# 字典中与起始值对齐

foo = {

long\_dictionary\_key: value1 +

value2,

...

}

# 4 个空格缩进，第一行不需要

foo = long\_function\_name(

var\_one, var\_two, var\_three,

var\_four)

# 字典中 4 个空格缩进

foo = {

long\_dictionary\_key:

long\_dictionary\_value,

...

}

No: # 第一行有空格是禁止的

foo = long\_function\_name(var\_one, var\_two,

var\_three, var\_four)

# 2 个空格是禁止的

foo = long\_function\_name(

var\_one, var\_two, var\_three,

var\_four)

# 字典中没有处理缩进

foo = {

long\_dictionary\_key:

long\_dictionary\_value,

...

}

## 空白行

用两个空白行分隔顶层函数和类定义。  
类中的方法定义用一个空行分隔。  
可以使用额外的空白行（节省空间）来分隔相关功能组。在一堆相关的单行程序（例如一组虚拟执行程序）之间可能会省略空白行。  
在函数中使用空行来节省逻辑部分。  
恰当的使用空白行可以提高代码的可读性

## 异常

不要轻易使用 try/except

except 后面需要指定捕捉的异常，裸露的 except 会捕捉所有异常，意味着会隐藏潜在的问题

可以有多个 except 语句，捕捉多种异常，分别做异常处理

使用 finally 子句来处理一些收尾操作

try/except 里的内容不要太多，只在可能抛出异常的地方使用，如：

# badtry:    user = User()    user.name = "leon"    user.age = int(age) # 可能抛出异常

## 注释

确保对模块, 函数, 方法和行内注释使用正确的风格

**文档字符串**

Python有一种独一无二的的注释方式: 使用文档字符串. 文档字符串是包, 模块, 类或函数里的第一个语句. 这些字符串可以通过对象的\_\_doc\_\_成员被自动提取, 并且被pydoc所用. (你可以在你的模块上运行pydoc试一把, 看看它长什么样). 我们对文档字符串的惯例是使用三重双引号"""( [PEP-257](http://www.python.org/dev/peps/pep-0257/) ). 一个文档字符串应该这样组织: 首先是一行以句号, 问号或惊叹号结尾的概述(或者该文档字符串单纯只有一行). 接着是一个空行. 接着是文档字符串剩下的部分, 它应该与文档字符串的第一行的第一个引号对齐. 下面有更多文档字符串的格式化规范.

**模块**

每个文件应该包含一个许可样板. 根据项目使用的许可(例如, Apache 2.0, BSD, LGPL, GPL), 选择合适的样板.

**函数和方法**

下文所指的函数,包括函数, 方法, 以及生成器.

一个函数必须要有文档字符串, 除非它满足以下条件:

外部不可见

非常短小

简单明了

文档字符串应该包含函数做什么, 以及输入和输出的详细描述. 通常, 不应该描述"怎么做", 除非是一些复杂的算法. 文档字符串应该提供足够的信息, 当别人编写代码调用该函数时, 他不需要看一行代码, 只要看文档字符串就可以了. 对于复杂的代码, 在代码旁边加注释会比使用文档字符串更有意义.

关于函数的几个方面应该在特定的小节中进行描述记录， 这几个方面如下文所述. 每节应该以一个标题行开始. 标题行以冒号结尾. 除标题行外, 节的其他内容应被缩进2个空格.

**类**

类应该在其定义下有一个用于描述该类的文档字符串. 如果你的类有公共属性(Attributes), 那么文档中应该有一个属性(Attributes)段. 并且应该遵守和函数参数相同的格式.

class SampleClass(object):

"""Summary of class here.

Longer class information....

Longer class information....

Attributes:

likes\_spam: A boolean indicating if we like SPAM or not.

eggs: An integer count of the eggs we have laid.

"""

def \_\_init\_\_(self, likes\_spam=False):

"""Inits SampleClass with blah."""

self.likes\_spam = likes\_spam

self.eggs = 0

def public\_method(self):

"""Performs operation blah."""

**块注释和行注释**

最需要写注释的是代码中那些技巧性的部分. 如果你在下次 [代码审查](http://en.wikipedia.org/wiki/Code_review) 的时候必须解释一下, 那么你应该现在就给它写注释. 对于复杂的操作, 应该在其操作开始前写上若干行注释. 对于不是一目了然的代码, 应在其行尾添加注释.

# We use a weighted dictionary search to find out where i is in

# the array. We extrapolate position based on the largest num

# in the array and the array size and then do binary search to

# get the exact number.

if i & (i-1) == 0: # true iff i is a power of 2

为了提高可读性, 注释应该至少离开代码2个空格.

另一方面, 绝不要描述代码. 假设阅读代码的人比你更懂Python, 他只是不知道你的代码要做什么.

# BAD COMMENT: Now go through the b array and make sure whenever i occurs

# the next element is i+1

## Class（类）

显示的写明父类，如果不是继承自别的类，就继承自 object 类

使用 super 调用父类的方法

支持多继承，即同时有多个父类，建议使用 Mixin

## 字符串

Yes: x = a + b

x = '%s, %s!' % (imperative, expletive)

x = '{}, {}!'.format(imperative, expletive)

x = 'name: %s; score: %d' % (name, n)

x = 'name: {}; score: {}'.format(name, n)

No: x = '%s%s' % (a, b) # use + in this case

x = '{}{}'.format(a, b) # use + in this case

x = imperative + ', ' + expletive + '!'

x = 'name: ' + name + '; score: ' + str(n)

避免在循环中用+和+=操作符来累加字符串. 由于字符串是不可变的, 这样做会创建不必要的临时对象, 并且导致二次方而不是线性的运行时间. 作为替代方案, 你可以将每个子串加入列表, 然后在循环结束后用 .join 连接列表. (也可以将每个子串写入一个 cStringIO.StringIO 缓存中.)

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)

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>'

在同一个文件中, 保持使用字符串引号的一致性. 使用单引号'或者双引号"之一用以引用字符串, 并在同一文件中沿用. 在字符串内可以使用另外一种引号, 以避免在字符串中使用. PyLint已经加入了这一检查.

Yes:

Python('Why are you hiding your eyes?')

Gollum("I'm scared of lint errors.")

Narrator('"Good!" thought a happy Python reviewer.')

No:

Python("Why are you hiding your eyes?")

Gollum('The lint. It burns. It burns us.')

Gollum("Always the great lint. Watching. Watching.")

为多行字符串使用三重双引号"""而非三重单引号'''. 当且仅当项目中使用单引号'来引用字符串时, 才可能会使用三重'''为非文档字符串的多行字符串来标识引用. 文档字符串必须使用三重双引号""". 不过要注意, 通常用隐式行连接更清晰, 因为多行字符串与程序其他部分的缩进方式不一致.

Yes:

print ("This is much nicer.\n"

"Do it this way.\n")

No:

print """This is pretty ugly.

Don't do this.

"""

## 导入格式

每个导入应该独占一行

Yes: import os

import sys

No: import os, sys

导入总应该放在文件顶部, 位于模块注释和文档字符串之后, 模块全局变量和常量之前. 导入应该按照从最通用到最不通用的顺序分组:

1. 标准库导入
2. 第三方库导入
3. 应用程序指定导入

每种分组中, 应该根据每个模块的完整包路径按字典序排序, 忽略大小写.

import foo

from foo import bar

from foo.bar import baz

from foo.bar import Quux

from Foob import ar

## 语句

通常每个语句应该独占一行

不过, 如果测试结果与测试语句在一行放得下, 你也可以将它们放在同一行. 如果是if语句, 只有在没有else时才能这样做. 特别地, 绝不要对 try/except 这样做, 因为try和except不能放在同一行.

Yes:

if foo: bar(foo)

No:

if foo: bar(foo)

else: baz(foo)

try: bar(foo)

except ValueError: baz(foo)

try:

bar(foo)

except ValueError: baz(foo)

## 命名

**应该避免的名称**

单字符名称, 除了计数器和迭代器.

包/模块名中的连字符(-)

双下划线开头并结尾的名称(Python保留, 例如\_\_init\_\_)

**命名约定**

1.所谓"内部(Internal)"表示仅模块内可用, 或者, 在类内是保护或私有的.

2.用单下划线(\_)开头表示模块变量或函数是protected的(使用import \* from时不会包含).

3.用双下划线(\_\_)开头的实例变量或方法表示类内私有.

4.将相关的类和顶级函数放在同一个模块里. 不像Java, 没必要限制一个类一个模块.

5.对类名使用大写字母开头的单词(如CapWords, 即Pascal风格), 但是模块名应该用小写加下划线的方式(如lower\_with\_under.py). 尽管已经有很多现存的模块使用类似于CapWords.py这样的命名, 但现在已经不鼓励这样做, 因为如果模块名碰巧和类名一致, 这会让人困扰.