

**“杨哈哈”网站**

|  |  |  |
| --- | --- | --- |
| 文件状态：  　[√]草稿  　[ ]正式发布  　[　]正在修改 | **文件标识：** | SE-2018-G16-编码规范 |
| **当前版本：** | 0.1.1 |
| **作者：** | 周德阳，何银超，冯一鸣 |
| **完成日期：** | 2018-5-6 |

**编码规范**

**版 本 历 史**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **版本** | **作者** | **协助者** | **起止日期** | **备注** |
| 0.1.1 | 周德阳 | 周德阳 | 2018/5/5-2018/5/6 | 起草 |

目录

[分号 2](#_Toc516142667)

[行长度 2](#_Toc516142668)

[括号 2](#_Toc516142669)

[缩进 3](#_Toc516142670)

[空行 4](#_Toc516142671)

[空格 4](#_Toc516142672)

[注释 4](#_Toc516142673)

[文档字符串 5](#_Toc516142674)

[模块 5](#_Toc516142675)

[函数和方法 5](#_Toc516142676)

[类 5](#_Toc516142677)

[字符串 7](#_Toc516142678)

[导入格式 9](#_Toc516142679)

# 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：

长的导入模块语句

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 , 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。

# 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.

Yes: if foo:

bar()

while x:

x = bar()

if x and y:

bar()

if not x:

bar()

return foo

for (x, y) in dict.items(): ...

No: if (x):

bar()

if not(x):

bar()

return (foo)

# 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,

...

}

# 空行

顶级定义之间空两行, 方法定义之间空一行

顶级定义之间空两行, 比如函数或者类定义. 方法定义, 类定义与第一个方法之间, 都应该空一行. 函数或方法中, 某些地方要是你觉得合适, 就空一行.

# 空格

按照标准的排版规范来使用标点两边的空格

括号内不要有空格.

按照标准的排版规范来使用标点两边的空格

大部分.py文件不必以#!作为文件的开始. 根据 PEP-394 , 程序的main文件应该以 #!/usr/bin/python2或者 #!/usr/bin/python3开始.

# 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 ). 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 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

类

如果一个类不继承自其它类, 就显式的从object继承. 嵌套类也一样.

Yes: class SampleClass(object):

pass

class OuterClass(object):

class InnerClass(object):

pass

class ChildClass(ParentClass):

"""Explicitly inherits from another class already."""

No: class SampleClass:

pass

class OuterClass:

class InnerClass:

pass

继承自 object 是为了使属性(properties)正常工作, 并且这样可以保护你的代码, 使其不受Python 3000的一个特殊的潜在不兼容性影响. 这样做也定义了一些特殊的方法, 这些方法实现了对象的默认语义, 包括 \_\_new\_\_, \_\_init\_\_, \_\_delattr\_\_, \_\_getattribute\_\_, \_\_setattr\_\_, \_\_hash\_\_, \_\_repr\_\_, and \_\_str\_\_ .

# 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.

"""

文件和sockets

在文件和sockets结束时, 显式的关闭它.

除文件外, sockets或其他类似文件的对象在没有必要的情况下打开, 会有许多副作用, 例如:

它们可能会消耗有限的系统资源, 如文件描述符. 如果这些资源在使用后没有及时归还系统, 那么用于处理这些对象的代码会将资源消耗殆尽.

持有文件将会阻止对于文件的其他诸如移动、删除之类的操作.

仅仅是从逻辑上关闭文件和sockets, 那么它们仍然可能会被其共享的程序在无意中进行读或者写操作. 只有当它们真正被关闭后, 对于它们尝试进行读或者写操作将会跑出异常, 并使得问题快速显现出来.

而且, 幻想当文件对象析构时, 文件和sockets会自动关闭, 试图将文件对象的生命周期和文件的状态绑定在一起的想法, 都是不现实的. 因为有如下原因:

没有任何方法可以确保运行环境会真正的执行文件的析构. 不同的Python实现采用不同的内存管理技术, 比如延时垃圾处理机制. 延时垃圾处理机制可能会导致对象生命周期被任意无限制的延长.

对于文件意外的引用,会导致对于文件的持有时间超出预期(比如对于异常的跟踪, 包含有全局变量等).

推荐使用 "with"语句 以管理文件:

with open("hello.txt") as hello\_file:

for line in hello\_file:

print line

对于不支持使用"with"语句的类似文件的对象,使用 contextlib.closing():

import contextlib

with contextlib.closing(urllib.urlopen("http://www.python.org/")) as front\_page:

for line in front\_page:

print line

Legacy AppEngine 中Python 2.5的代码如使用"with"语句, 需要添加 "from \_\_future\_\_ import with\_statement".

TODO注释

为临时代码使用TODO注释, 它是一种短期解决方案. 不算完美, 但够好了.

TODO注释应该在所有开头处包含"TODO"字符串, 紧跟着是用括号括起来的你的名字, email地址或其它标识符. 然后是一个可选的冒号. 接着必须有一行注释, 解释要做什么. 主要目的是为了有一个统一的TODO格式, 这样添加注释的人就可以搜索到(并可以按需提供更多细节). 写了TODO注释并不保证写的人会亲自解决问题. 当你写了一个TODO, 请注上你的名字.

# TODO(kl@gmail.com): Use a "\*" here for string repetition.

# TODO(Zeke) Change this to use relations.

如果你的TODO是"将来做某事"的形式, 那么请确保你包含了一个指定的日期("2009年11月解决")或者一个特定的事件("等到所有的客户都可以处理XML请求就移除这些代码").

# 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:

标准库导入

第三方库导入

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.