本文给出主Python版本标准库的编码约定。CPython的C代码风格参见[PEP7](https://www.python.org/dev/peps/pep-0007/)。

本文和[PEP 257 文档字符串标准](https://www.python.org/dev/peps/pep-0257/)改编自Guido最初的《Python Style Guide》, 并增加了Barry的[GNU Mailman Coding Style Guide](http://barry.warsaw.us/software/STYLEGUIDE.txt)的部分内容。

本文会随着语言改变等而改变。

许多项目都有自己的编码风格指南，冲突时自己的指南为准。

**一致性考虑**

Guido的关键点之一是：代码更多是用来读而不是写。本指南旨在改善Python代码的可读性，即[PEP 20](https://www.python.org/dev/peps/pep-0020/)所说的“可读性计数"(Readability counts)。

风格指南强调一致性。项目、模块或函数保持一致都很重要。

最重要的是知道何时不一致, 有时风格指南并不适用。当有疑惑时运用你的最佳判断，参考其他例子并多问！

特别注意：不要因为遵守本PEP而破坏向后兼容性！

部分可以违背指南情况：

* 遵循指南会降低可读性。
* 与周围其他代码不一致。
* 代码在引入指南完成，暂时没有理由修改。
* 旧版本兼容。

[回到顶部](http://www.open-open.com/lib/view/open1433813937629.html#_labelTop)

**代码布局**

**缩进**

每级缩进用4个空格。

括号中使用垂直隐式缩进或使用悬挂缩进。后者应该注意第一行要没有参数，后续行要有缩进。

* Yes

[?](http://www.open-open.com/lib/view/open1433813937629.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | # 对准左括号  foo = long\_function\_name(var\_one, var\_two,                           var\_three, var\_four)    # 不对准左括号，但加多一层缩进，以和后面内容区别。  def long\_function\_name(          var\_one, var\_two, var\_three,          var\_four):      print(var\_one)    # 悬挂缩进必须加多一层缩进.  foo = long\_function\_name(      var\_one, var\_two,      var\_three, var\_four) |

* No

[?](http://www.open-open.com/lib/view/open1433813937629.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | # 不使用垂直对齐时，第一行不能有参数。  foo = long\_function\_name(var\_one, var\_two,      var\_three, var\_four)    # 参数的缩进和后续内容缩进不能区别。  def long\_function\_name(      var\_one, var\_two, var\_three,      var\_four):      print(var\_one) |

4个空格的规则是对续行可选的。

[?](http://www.open-open.com/lib/view/open1433813937629.html)

|  |  |
| --- | --- |
| 1  2  3  4 | # 悬挂缩进不一定是4个空格  foo = long\_function\_name(    var\_one, var\_two,    var\_three, var\_four) |

if语句跨行时，两个字符关键字(比如if)加上一个空格，再加上左括号构成了很好的缩进。后续行暂时没有规定，至少有如下三种格式，建议使用第3种。

[?](http://www.open-open.com/lib/view/open1433813937629.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | # 没有额外缩进，不是很好看，个人不推荐.  if (this\_is\_one\_thing and      that\_is\_another\_thing):      do\_something()    # 添加注释  if (this\_is\_one\_thing and      that\_is\_another\_thing):      # Since both conditions are true, we can frobnicate.      do\_something()    # 额外添加缩进,推荐。  # Add some extra indentation on the conditional continuation line.  if (this\_is\_one\_thing          and that\_is\_another\_thing):      do\_something() |

右边括号也可以另起一行。有两种格式，建议第2种。

[?](http://www.open-open.com/lib/view/open1433813937629.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | # 右括号不回退，个人不推荐  my\_list = [      1, 2, 3,      4, 5, 6,      ]  result = some\_function\_that\_takes\_arguments(      'a', 'b', 'c',      'd', 'e', 'f',      )    # 右括号回退  my\_list = [      1, 2, 3,      4, 5, 6,  ]  result = some\_function\_that\_takes\_arguments(      'a', 'b', 'c',      'd', 'e', 'f',  ) |

**空格或Tab?**

* 空格是首选的缩进方法。
* Tab仅仅在已经使用tab缩进的代码中为了保持一致性而使用。
* Python 3中不允许混合使用Tab和空格缩进。
* Python 2的包含空格与Tab和空格缩进的应该全部转为空格缩进。

Python2命令行解释器使用-t选项时有非法混合Tab和空格的情况会告警。当使用-tt警告提升为错误。强烈推荐这些选项！另外个人推荐pep8和autopep8模块。

**最大行宽**

限制所有行的最大行宽为79字符。

文本长块，比如文档字符串或注释，行长度应限制为72个字符。

多数工具默认的续行功能会破坏代码结构，使它更难理解，不推荐使用。但是超过80个字符加以提醒是必要的。一些工具可能根本不具备动态换行功能。

一些团队强烈希望更长的行宽。如果能达成一致，可以从从80提高到100个字符(最多99个字符)增加了标称线的长度，不过依旧建议文档字符串和注释保持在72的长度。

Python标准库比较保守，限制行宽79个字符(文档字符串/注释72）。

续行的首选方法是使用小括号、中括号和大括号反斜线仍可能在适当的时候。其次是反斜杠。比如with语句中：

[?](http://www.open-open.com/lib/view/open1433813937629.html)

|  |  |
| --- | --- |
| 1  2  3 | with open('/path/to/some/file/you/want/to/read') as file\_1, \       open('/path/to/some/file/being/written', 'w') as file\_2:      file\_2.write(file\_1.read()) |

类似的还有assert。

注意续行要尽量不影响可读性。比如通常在二元运算符之后续行：

[?](http://www.open-open.com/lib/view/open1433813937629.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | 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 -- values are %s, %s" %                               (width, height))          Blob.\_\_init\_\_(self, width, height,                        color, emphasis, highlight) |

**空行**

* 两行空行分割顶层函数和类的定义。
* 类的方法定义用单个空行分割。
* 额外的空行可以必要的时候用于分割不同的函数组，但是要尽量节约使用。
* 额外的空行可以必要的时候在函数中用于分割不同的逻辑块，但是要尽量节约使用。
* Python接 contol-L作为空白符；许多工具视它为分页符，这些要因编辑器而异。

**源文件编码**

在核心Python发布的代码应该总是使用UTF-8(ASCII在Python 2)。

ASCII文件(Python 2)或UTF-8(Python 3)不应有编码声明。

标准库中非默认的编码应仅用于测试或当注释或文档字符串,比如包含非ASCII字符的作者姓名，尽量使用\x , \u , \U , or \N。

Python 3.0及以后版本，[PEP 3131](https://www.python.org/dev/peps/pep-3131/)可供参考，部分内容如下：在Python标准库必须使用ASCII标识符，并尽量只使用英文字母。此外字符串和注释也必须用ASCII。唯一的例外是：（a）测试非ASCII的功能，和（b）作者的名字不是拉丁字母。

**导入**

* 导入在单独行

Yes：

[?](http://www.open-open.com/lib/view/open1433813937629.html)

|  |  |
| --- | --- |
| 1  2  3 | import os  import sys  from subprocess import Popen, PIPE |

No:

import sys, os

* 导入始终在文件的顶部，在模块注释和文档字符串之后，在模块全局变量和常量之前。

导入顺序如下：标准库进口,相关的第三方库，本地库。各组的导入之间要有空行。

相关的all放在导入之后。

* 推荐绝对路径导入，因为它们通常更可读，而且往往是表现更好的（或至少提供更好的错误消息。

[?](http://www.open-open.com/lib/view/open1433813937629.html)

|  |  |
| --- | --- |
| 1  2  3 | import mypkg.sibling  from mypkg import sibling  from mypkg.sibling import example |

在绝对路径比较长的情况下，也可以使用相对路径：

[?](http://www.open-open.com/lib/view/open1433813937629.html)

|  |  |
| --- | --- |
| 1  2 | from . import sibling  from .sibling import example |

Python 3中已经禁止隐式的相对导入。

* 导入类的方法：

[?](http://www.open-open.com/lib/view/open1433813937629.html)

|  |  |
| --- | --- |
| 1  2 | from myclass import MyClass  from foo.bar.yourclass import YourClass |

如果和本地名字有冲突：

[?](http://www.open-open.com/lib/view/open1433813937629.html)

|  |  |
| --- | --- |
| 1  2 | import myclass  import foo.bar.yourclass |

* 禁止使用通配符导入。

通配符导入(from <module> import \*)应该避免，因为它不清楚命名空间有哪些名称存，混淆读者和许多自动化的工具。唯一的例外是重新发布对外的API时可以考虑使用。

[回到顶部](http://www.open-open.com/lib/view/open1433813937629.html#_labelTop)

**字符串引用**

Python中单引号字符串和双引号字符串都是相同的。注意尽量避免在字符串中的反斜杠以提高可读性。

根据[PEP 257](https://www.python.org/dev/peps/pep-0257/), 三个引号都使用双引号。

[回到顶部](http://www.open-open.com/lib/view/open1433813937629.html#_labelTop)

**表达式和语句中的空格**

**强制要求**

* 括号里边避免空格

[?](http://www.open-open.com/lib/view/open1433813937629.html)

|  |  |
| --- | --- |
| 1  2  3  4  5 | # 括号里边避免空格  # Yes  spam(ham[1], {eggs: 2})  # No  spam( ham[ 1 ], { eggs: 2 } ) |

* 逗号，冒号，分号之前避免空格

[?](http://www.open-open.com/lib/view/open1433813937629.html)

|  |  |
| --- | --- |
| 1  2  3  4  5 | # 逗号，冒号，分号之前避免空格  # Yes  if x == 4: print x, y; x, y = y, x  # No  if x == 4 : print x , y ; x , y = y , x |

* 索引操作中的冒号当作操作符处理前后要有同样的空格(一个空格或者没有空格，个人建议是没有。

[?](http://www.open-open.com/lib/view/open1433813937629.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | # Yes  ham[1:9], ham[1:9:3], ham[:9:3], ham[1::3], ham[1:9:]  ham[lower:upper], ham[lower:upper:], ham[lower::step]  ham[lower+offset : upper+offset]  ham[: upper\_fn(x) : step\_fn(x)], ham[:: step\_fn(x)]  ham[lower + offset : upper + offset]  # No  ham[lower + offset:upper + offset]  ham[1: 9], ham[1 :9], ham[1:9 :3]  ham[lower : : upper]  ham[ : upper] |

函数调用的左括号之前不能有空格

[?](http://www.open-open.com/lib/view/open1433813937629.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | # Yes  spam(1)  dct['key'] = lst[index]    # No  spam (1)  dct ['key'] = lst [index] |

* 赋值等操作符前后不能因为对齐而添加多个空格

[?](http://www.open-open.com/lib/view/open1433813937629.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | # Yes  x = 1  y = 2  long\_variable = 3    # No  x             = 1  y             = 2  long\_variable = 3 |

**其他建议**

* 二元运算符两边放置一个空格:

涉及 =、符合操作符 ( += , -=等)、比较( == , < , > , != , <> , <= , >= , in , not in , is , is not )、布尔( and , or , not )。

* 优先级高的运算符或操作符的前后不建议有空格。

[?](http://www.open-open.com/lib/view/open1433813937629.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | # Yes  i = i + 1  submitted += 1  x = x\*2 - 1  hypot2 = x\*x + y\*y  c = (a+b) \* (a-b)    # No  i=i+1  submitted +=1  x = x \* 2 - 1  hypot2 = x \* x + y \* y  c = (a + b) \* (a - b) |

* 关键字参数和默认值参数的前后不要加空格

[?](http://www.open-open.com/lib/view/open1433813937629.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | # 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) |

* 函数注释中，=前后要有空格，冒号和"->"的前面无空格，后面有空格。

[?](http://www.open-open.com/lib/view/open1433813937629.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | # Yes  def munge(input: AnyStr):  def munge(sep: AnyStr = None):  def munge() -> AnyStr:  def munge(input: AnyStr, sep: AnyStr = None, limit=1000):    # No  def munge(input: AnyStr=None):  def munge(input:AnyStr):  def munge(input: AnyStr)->PosInt: |

* 通常不推荐复合语句(Compound statements: 多条语句写在同一行)。

[?](http://www.open-open.com/lib/view/open1433813937629.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | # Yes  if foo == 'blah':      do\_blah\_thing()  do\_one()  do\_two()  do\_three()    # No  if foo == 'blah': do\_blah\_thing()  do\_one(); do\_two(); do\_three() |

* 尽管有时可以在if/for/while 的同一行跟一小段代码，但绝不要跟多个子句，并尽量避免换行。

[?](http://www.open-open.com/lib/view/open1433813937629.html)

|  |  |
| --- | --- |
| 1  2  3  4 | # No  if foo == 'blah': do\_blah\_thing()  for x in lst: total += x  while t < 10: t = delay() |

更不是：

[?](http://www.open-open.com/lib/view/open1433813937629.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | # No  if foo == 'blah': do\_blah\_thing()  else: do\_non\_blah\_thing()    try: something()  finally: cleanup()    do\_one(); do\_two(); do\_three(long, argument,                               list, like, this)    if foo == 'blah': one(); two(); three() |

[回到顶部](http://www.open-open.com/lib/view/open1433813937629.html#_labelTop)

**注释**

与代码自相矛盾的注释比没注释更差。修改代码时要优先更新注释！

注释是完整的句子。如果注释是断句，首字母应该大写，除非它是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，可以省略末尾的句号。注释块通常由一个或多个段落组成。段落由完整的句子构成且每个句子应该以点号(后面要有两个空格)结束，并注意断词和空格。

非英语国家的程序员请用英语书写你的注释，除非你120%确信代码永远不会被不懂你的语言的人阅读。

**注释块**

注释块通常应用在代码前，并和这些代码有同样的缩进。每行以 '# '(除非它是注释内的缩进文本，注意#后面有空格)。

注释块内的段落用仅包含单个 '#' 的行分割。

**行内注释**

慎用行内注释(Inline Comments) 节俭使用行内注释。 行内注释是和语句在同一行，至少用两个空格和语句分开。行内注释不是必需的，重复罗嗦会使人分心。不要这样做：

[?](http://www.open-open.com/lib/view/open1433813937629.html)

|  |  |
| --- | --- |
| 1 | x = x + 1 # Increment x |

但有时很有必要:

[?](http://www.open-open.com/lib/view/open1433813937629.html)

|  |  |
| --- | --- |
| 1 | x = x + 1 # Compensate for border |

**文档字符串**

文档字符串的标准参见：[PEP 257](https://www.python.org/dev/peps/pep-0257/)。

* 为所有公共模块、函数、类和方法书写文档字符串。非公开方法不一定有文档字符串，建议有注释(出现在 def 行之后)来描述这个方法做什么。
* 更多参考：[PEP 257 文档字符串约定](https://www.python.org/dev/peps/pep-0257/)。注意结尾的 """ 应该单独成行，例如：

[?](http://www.open-open.com/lib/view/open1433813937629.html)

|  |  |
| --- | --- |
| 1  2  3 | """Return a foobang  Optional plotz says to frobnicate the bizbaz first.  """ |

* 单行的文档字符串，结尾的 """ 在同一行。

[回到顶部](http://www.open-open.com/lib/view/open1433813937629.html#_labelTop)

**版本标签**

版本注记 (Version Bookkeeping)

如果你必须在源文件中包含git、Subversion、CVS或RCS crud信息，放置在模块的文档字符串之后，任何其他代码之前，上下各用一个空行：

[?](http://www.open-open.com/lib/view/open1433813937629.html)

|  |  |
| --- | --- |
| 1 | \_\_version\_\_ = "$Revision$"# $Source$ |

[回到顶部](http://www.open-open.com/lib/view/open1433813937629.html#_labelTop)

**命名约定**

Python库的命名约定有点混乱，不可能完全一致。但依然有些普遍推荐的命名规范的。新的模块和包 (包括第三方的框架) 应该遵循这些标准。对不同风格的已有的库，建议保持内部的一致性。

**最重要的原则**

用户可见的API命名应遵循使用约定而不是实现。

**描述：命名风格**

有多种命名风格：

* b(单个小写字母)
* B(单个大写字母)
* lowercase(小写串)
* lower\_case\_with\_underscores(带下划线的小写)
* UPPERCASE(大写串)
* UPPER\_CASE\_WITH\_UNDERSCORES(带下划线的大写串)
* CapitalizedWords(首字母大写的单词串或驼峰缩写）

注意: 使用大写缩写时，缩写使用大写字母更好。故 HTTPServerError 比 HttpServerError 更好。

* mixedCase(混合大小写，第一个单词是小写)
* Capitalized\_Words\_With\_Underscores（带下划线，首字母大写，丑陋）

还有一种风格使用短前缀分组名字。这在Python中不常用， 但出于完整性提一下。例如，os.stat()返回的元组有st\_mode, st\_size, st\_mtime等等这样的名字(与POSIX系统调用结构体一致)。

X11库的所有公开函数以X开头, Python中通常认为是不必要的，因为属性和方法名有对象作前缀，而函数名有模块名为前缀。

下面讲述首尾有下划线的情况:

* \_single\_leading\_underscore:(单前置下划线): 弱内部使用标志。 例如"from M import " 不会导入以下划线开头的对象。
* single\_trailing\_underscore\_(单后置下划线): 用于避免与 Python关键词的冲突。 例如：

[?](http://www.open-open.com/lib/view/open1433813937629.html)

|  |  |
| --- | --- |
| 1 | Tkinter.Toplevel(master, class\_='ClassName') |

* \_\_double\_leading\_underscore(双前置下划线): 当用于命名类属性，会触发名字重整。 (在类FooBar中，\_\_boo变成 \_FooBar\_\_boo)。
* \_\_double\_leading\_and\_trailing\_underscore\_\_(双前后下划线)：用户名字空间的魔法对象或属性。例如:\_\_init\_\_ , \_\_import\_\_ or \_\_file\_\_，不要自己发明这样的名字。

**命名约定规范**

* 避免采用的名字

决不要用字符'l'(小写字母el)，'O'(大写字母oh)，或 'I'(大写字母eye) 作为单个字符的变量名。一些字体中，这些字符不能与数字1和0区别。用'L' 代替'l'时。

* 包和模块名

模块名要简短，全部用小写字母，可使用下划线以提高可读性。包名和模块名类似，但不推荐使用下划线。

模块名对应到文件名，有些文件系统不区分大小写且截短长名字，在 Unix上不是问题，但当把代码迁移到 Mac、Windows 或 DOS 上时，就可能是个问题。当然随着系统的演进，这个问题已经不是经常出现。

另外有些模块底层用C或C++ 书写，并有对应的高层Python模块，C/C++模块名有一个前置下划线 (如：\_socket)。

* 类名

遵循CapWord。

接口需要文档化并且可以调用时，可能使用函数的命名规则。

注意大部分内置的名字是单个单词（或两个），CapWord只适用于异常名称和内置的常量。

* 异常名

如果确实是错误，需要在类名添加后缀 "Error"。

* 全局变量名

变量尽量只用于模块内部，约定类似函数。

对设计为通过 "from M import " 来使用的模块，应采用 \_\_all\_\_ 机制来防止导入全局变量；或者为全局变量加一个前置下划线。

* 函数名

函数名应该为小写，必要时可用下划线分隔单词以增加可读性。 mixedCase(混合大小写)仅被允许用于兼容性考虑(如: threading.py)。

* 函数和方法的参数

实例方法第一个参数是 'self'。

类方法第一个参数是 'cls'。

如果函数的参数名与保留关键字冲突，通常在参数名后加一个下划线。

* 方法名和实例变量

同函数命名规则。

非公开方法和实例变量增加一个前置下划线。

为避免与子类命名冲突，采用两个前置下划线来触发重整。类Foo属性名为\_\_a， 不能以 Foo.\_\_a访问。(执著的用户还是可以通过Foo.\_Foo\_\_a。) 通常双前置下划线仅被用来避免与基类的属性发生命名冲突。

* 常量

常量通常在模块级定义,由大写字母用下划线分隔组成。比如括MAX\_OVERFLOW和TOTAL。

* 继承设计

考虑类的方法和实例变量(统称为属性)是否公开。如果有疑问，选择不公开；把其改为公开比把公开属性改为非公开要容易。

公开属性可供所有人使用，并通常向后兼容。非公开属性不给第三方使用、可变甚至被移除。

这里不使用术语"private"， Python中没有属性是真正私有的。

另一类属性是子类API(在其他语言中通常称为 "protected")。 一些类被设计为基类，可以扩展和修改。

谨记这些Python指南：

1. 公开属性应该没有前导下划线。
2. 如果公开属性名和保留关键字冲突，可以添加后置下划线
3. 简单的公开数据属性，最好只公开属性名，没有复杂的访问/修改方法，python的Property提供了很好的封装方法。 d.如果不希望子类使用的属性，考虑用两个前置下划线(没有后置下划线)命名。

**公共和内部接口**

任何向后兼容的保证只适用于公共接口。

文档化的接口通常是公共的，除非明说明是临时的或为内部接口、其他所有接口默认是内部的。

为了更好地支持内省，模块要在\_\_all\_\_属性列出公共API。

内部接口要有前置下划线。

如果命名空间(包、模块或类)是内部的，里面的接口也是内部的。

导入名称应视为实现细节。其他模块不能间接访名字，除非在模块的API文档中明确记载，如os.path中或包的\_\_init\_\_暴露了子模块。

[回到顶部](http://www.open-open.com/lib/view/open1433813937629.html#_labelTop)

**编程建议**

* 考虑多种Python实现(PyPy, Jython, IronPython,Pyrex, Psyco, 等等)。

例如，CPython对a+=b或a=a+b等语句有高效的实现，但在Jython中运行很慢，尽量改用**.join()。**

* None比较用'is'或'is not'，不要用等号。

注意"if x is not None" 与"if x" 的区别。

* 用"is not"代替"not ... is"。前者的可读性更好。

# Yes

if foo is not None

# No

if not foo is None

* 使用基于类的异常。

比较排序操作最好是实现所有六个操作，而不是代码中实现比较逻辑。functools.total\_ordering()装饰符可以生成缺失的比较方法。

[?](http://www.open-open.com/lib/view/open1433813937629.html)

|  |  |
| --- | --- |
| 1 | \_\_eq\_\_，\_\_ne\_\_，\_\_lt\_\_，\_\_lt\_\_，\_\_gt\_\_，\_\_\_\_） |

[PEP207 比较标准](https://www.python.org/dev/peps/pep-0207/)表明反射规则由Python完成。因此解释器可能会交换参数的位置，比如替换y > x为x < y，所以有必要实现这5种方法。

* 使用函数定义def代替lambda赋值给标识符：

[?](http://www.open-open.com/lib/view/open1433813937629.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | # Yes  def f(x):      return 2\*x    # No  f = lambda x: 2\*x |

前者更适合回调和字符串表示。

* 异常类继承自Exception，而不是BaseException。

源于异常，而不是BaseException例外。从BaseException直接继承的例外情况追赶他们几乎总是错误的事情做保留。

要设计基于层次的异常，捕捉到需要的异常，而不是异常引发的位置。能回答：“出了什么问题？”，而不是仅仅指出“问题发生”(更多参考：[PEP3151 重构OS和IO异常层次](https://www.python.org/dev/peps/pep-3151)）

* 适当使用异常链。在Python3中"raise X from Y"明确表示更换且保留了原来的traceback。

替换内部异常(在Python2: "raise X"或"raise X from None")时，确保相关细节转移到新的异常（如转换KeyError为AttributeError保存属性名，或在新的异常中嵌入原始异常)。

* Python2中用" raise ValueError('message')"代替"raise ValueError, 'message'"

后者不兼容Python3语法。前者续行方便。

* 捕获异常时尽量指明具体异常，而不是空"except:"子句。比如：

[?](http://www.open-open.com/lib/view/open1433813937629.html)

|  |  |
| --- | --- |
| 1  2  3  4  5 | # Yes  try:      import platform\_specific\_module  except ImportError:      platform\_specific\_module = None |

空"except:"子句(相当于except Exception)会捕捉SystemExit和KeyboardInterrupt异常，难以用Control-C中断程序，并可掩盖其他问题。如果 你捕捉信号错误之外所有的异常，使用"except Exception"。

空"except:"子句适用的情况两种情况：

a, 打印出或记录了traceback，至少让用户将知道已发生错误。 b, 代码需要做一些清理工作，并用 raise转发了异常。这样try...finally可以捕捉到它。

* Python 2.6以后建议用as显示绑定异常名：

[?](http://www.open-open.com/lib/view/open1433813937629.html)

|  |  |
| --- | --- |
| 1  2  3  4  5 | # Yes  try:      process\_data()  except Exception as exc:      raise DataProcessingFailedError(str(exc)) |

这样才能兼容Python3语法并避免歧义。

* 捕捉操作系统错误时，建议使用Python 3.3引入显式异常层次，支持内省errno值。
* 此外所有try/except子句的代码要尽可的少，以免屏蔽其他的错误。

[?](http://www.open-open.com/lib/view/open1433813937629.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | # Yes  try:      value = collection[key]  except KeyError:      return key\_not\_found(key)  else:      return handle\_value(value)    # No  try:      # 太泛了!      return handle\_value(collection[key])  except KeyError:      # 会捕捉到handle\_value()中的KeyError      return key\_not\_found(key) |

* 本地资源建议使用with语句，以确保即时清理。当然try / finally语句也是可以接受的。
* 上下文管理器在做获取和释放资源之外的事情时，应通过独立的函数或方法。例如：

[?](http://www.open-open.com/lib/view/open1433813937629.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | # Yes  with conn.begin\_transaction():      do\_stuff\_in\_transaction(conn)    # No  with conn:      do\_stuff\_in\_transaction(conn) |

后者指明enter和exit方法。

* 函数或者方法在没有返回时要明确返回None。

[?](http://www.open-open.com/lib/view/open1433813937629.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | # Yesdef foo(x):      if x >= 0:          return math.sqrt(x)      else:          return Nonedef bar(x):      if x < 0:          return None      return math.sqrt(x)# Nodef foo(x):      if x >= 0:          return math.sqrt(x)def bar(x):      if x < 0:          return      return math.sqrt(x) |

* 使用字符串方法而不是string模块。

python 2.0以后字符串方法总是更快，且Unicode字符串相同的API。

* 使用使用 **.startswith()和**.endswith()代替字符串切片来检查前缀和后缀。and

startswith()和endswith更简洁，利于减少错误。例如：

[?](http://www.open-open.com/lib/view/open1433813937629.html)

|  |  |
| --- | --- |
| 1  2  3  4  5 | # Yes  if foo.startswith('bar'):    # No  if foo[:3] == 'bar': |

* 使用isinstance()代替对象类型的比较：

[?](http://www.open-open.com/lib/view/open1433813937629.html)

|  |  |
| --- | --- |
| 1  2  3  4  5 | # Yes  if isinstance(obj, int):    # No  if type(obj) is type(1): |

检查是否是字符串时，注意Python 2中str和unicode有公共的基类:

if isinstance(obj, basestring): 在 Python 2.2 中，types 模块为此定义了 StringTypes 类型，例如：

[?](http://www.open-open.com/lib/view/open1433813937629.html)

|  |  |
| --- | --- |
| 1  2 | # Yes  if isinstance(obj, basestring): |

Python3中Unicode和basestring的不再存在(只有str)和字节对象不再是字符串(是整数序列)

* 对序列(字符串、列表 、元组), 空序列为false:

[?](http://www.open-open.com/lib/view/open1433813937629.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | # Yes  if not seq:     pass  if seq:     pass    # No  if len(seq):     pass  if not len(seq):     pass |

* 字符串后面不要有大量拖尾空格。
* 不要用 == 进行布尔比较

[?](http://www.open-open.com/lib/view/open1433813937629.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | # Yes  if greeting::     pass    # No  if greeting == True     pass  if greeting is True: # Worse     pass |

* Python标准库不使用的功能注释，这块有待用户去发现和体验有用的注释风格。下面有些第三方的建议(略)。