## 代码布局设计

### 1.1 缩进

A、   使用**四个空格**来进行缩进

B、   换行的时候可以使用反斜杠，最好的方法是使用**圆括号**，在使用反斜杠的时候，在反斜杠的后直接回车，**不能有任何空格存在**。

比较好的做法如下：

|  |
| --- |
| # Aligned with opening delimiter.  foo = long\_function\_name(var\_one, var\_two,               var\_three, var\_four） |

包含更多的缩进表示是剩余部分：

|  |
| --- |
| # More indentation included to distinguish this from the rest.  def long\_function\_name(          var\_one, var\_two, var\_three,          var\_four):      print(var\_one) |

    悬挂缩进应该添加一个级别：

|  |
| --- |
| # Hanging indents should add a level.  foo = long\_function\_name(      var\_one, var\_two,      var\_three, var\_four) |

 比较差的做法如下：（代码同样是可以运行的）

|  |
| --- |
| # Arguments on first line forbidden when not using vertical alignment.—未使用垂直对齐  foo = long\_function\_name(var\_one, var\_two,      var\_three, var\_four)    # Further indentation required as indentation is not distinguishable.（未使用缩进来表示每一层级）  def long\_function\_name(      var\_one, var\_two, var\_three,      var\_four):      print(var\_one) |

对于续行来说，四个空格的缩进是可选的。

 可选的如下：

|  |
| --- |
| # Hanging indents \*may\* be indented to other than 4 spaces.悬挂缩进的时候可以不是四个空格  foo = long\_function\_name(    var\_one, var\_two,    var\_three, var\_four) |

当使用if语句的时候，如果条件恰好的缩进为四个空格空格，那么导致后面的语句的缩进也是四个空格，那么这种情况下是可以接受的，如下所示：

没有额外的缩进：

|  |
| --- |
| # No extra indentation.  if (this\_is\_one\_thing and      that\_is\_another\_thing):      do\_something() |

 添加一个注释来进行分割缩进，做到语法高亮显示：

|  |
| --- |
| # Add a comment, which will provide some distinction in editors  # supporting syntax highlighting.  if (this\_is\_one\_thing and      that\_is\_another\_thing):      # Since both conditions are true, we can frobnicate.      do\_something() |

 在续行中添加额外的缩进：

|  |
| --- |
| # Add some extra indentation on the conditional continuation line.  if (this\_is\_one\_thing          and that\_is\_another\_thing):      do\_something() |

 成对的小括号，中括号在多行的结构中可以写成多行，然后括号在第一个不为空白的位置结束。如下：

|  |
| --- |
| my\_list = [      1, 2, 3,      4, 5, 6,      ]  result = some\_function\_that\_takes\_arguments(      'a', 'b', 'c',      'd', 'e', 'f',      ) |

 或者对齐第一个字符的位置结束，如下：

|  |
| --- |
| my\_list = [      1, 2, 3,      4, 5, 6,  ]  result = some\_function\_that\_takes\_arguments(      'a', 'b', 'c',      'd', 'e', 'f',  ) |

### 1.2 tab和空格的选择

关于tab的空格的选择，在python2中是可以混用的，但是在python3中，只能用一种风格。

### 1.3 最大行长度

行的最大长度为**79**个字符

在书写文档或者是注释的时候，行长度应该控制在72个字符。

反斜杠在有的时候是适用的，例如在参数很长，但是不能隐式的使用多行的时候，如下反斜杠的使用：

|  |
| --- |
| 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()) |

确保在合适的时候将连续的行进行分开，最好的位置是操作符之后，而不是在操作符之前，如下：

|  |
| --- |
| 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("values are %s, %s" %                               (width, height))          Blob.\_\_init\_\_(self, width, height,                        color, emphasis, highlight) |

### 1.4 空行

* T**op level函数和类的定义**的时候，**空两行**。
* **类中方法的定义空一行**。
* 在函数中谨慎使用空行来表示相关的逻辑段。
* **无关的函数之间用一个空行进行分割**。

### 1.5 源文件编码

在源文件中一直使用utf-8编码，在python2中使用ascll编码（存疑？）。

文件，在python2 中使用ascll编码，在python3中使用utf-8编码

### 1.6 导入

Import经常**使用单独的行**，如下：

|  |
| --- |
| import os  import sys |

或者使用如下的方式：

|  |
| --- |
| § from subprocess import Popen, PIPE |

**模块内容顺序：模块说明和docstring—import—globals&constants—其他定义**

**import总是在文件的最上行，在模块的注释和docstring之后，在模块的全局变量之前**。

import可以按照以下顺序进行组织：

         A标准类库import

         B第三方import

         C本地类库import

在每个组导入之后，可以用空行进行分割

把所有\_\_all\_\_相关类型的声明放在import之后

推荐使用绝对导入，可读性强，如下：

|  |
| --- |
| import mypkg.sibling  from mypkg import sibling  from mypkg.sibling import example |

对于复杂的封装布局来说，相对导入也是可以接受的，主要是使用绝对导入的时候路径太长，如下：

|  |
| --- |
| from . import sibling  from .sibling import example |

当导入一个类的时候，可以使用如下的方式：

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

当以上的写法导致本地名称冲突，可以写成如下：

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

并且使用"myclass.MyClass" and"foo.bar.yourclass.YourClass"。

在导入模块的时候，应该避免通配符的存在，如下：

|  |
| --- |
| from <module> import \* |

## 字符串引号

在对于字符串的标示中，使用双引号还是单引号是没有区别的，主要就是两者混合使用从而**避免反斜杠**的出现。

## 在表达式和语句中使用空格

### 3.1 避免使用空格情况

A． **在小括号，中括号，大括号中避免使用空格**

|  |
| --- |
| Yes: spam(ham[1], {eggs: 2})  No:  spam( ham[ 1 ], { eggs: 2 } ) |

B． **在逗号，分号，冒号之前不需要空格**

Yes: if x == 4: print x, y; x, y = y, x

No:  if x == 4 : print x , y ; x , y = y , x

C． 在切片的时候，避免使用空格，在扩展的切片中，必须使用相同的空格个数，如下所示：

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]

D．函数的左括号前不要添加空格：

|  |
| --- |
| Yes: spam(1)  No:  spam (1) |

E． 中括号前不要添加空格

Yes: dct['key'] = lst[index]

No:  dct ['key'] = lst [index]

F． **操作符左右各一个空格**，不要为了追求一致从而添加空格个数

|  |
| --- |
| Yes:  x = 1  y = 2  long\_variable = 3  No:  x             = 1  y             = 2  long\_variable = 3 |

1. 在制定关键字参数或者默认参数值的时候，**不要在=附近加上空格。**   
   推荐：

def **complex**(real, imag=0.0):

return magic(r=real, i=imag)

不推荐：

def **complex**(real, imag = 0.0):

return magic(r = real, i = imag)

### 3.2 其他建议

A． 避免在任何结尾添加空白。

B． 在下列操作符中左右各留空白

|  |
| --- |
| assignment ( = ), augmented assignment ( += , -= etc.), comparisons ( == , < , > , != , <> , <= , >= , in , not in , is , is not ), Booleans ( and , or , not ) |

C． 如果操作符优先级不同，注意在操作符左右留空白，特别是高优先级和低优先级的

|  |
| --- |
| 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) |

D． **在使用函数的时候，赋值和默认值之间不需要空格**

|  |
| --- |
| 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) |

E． 不要将多语句写在同一行

Rather not:

if foo == 'blah': do\_blah\_thing()

for x in lst: total += x

while t < 10: t = delay()

Definitely not:

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()

## 4.注释

**在修改的代码的时候，务必修改注释。**

**注释必须是英文，最好是完整的句子，首字母大写**

### 4.1 块注释

**在一段代码前增加注释，在#后添加一个空格，段落之间只有一个#作为行间隔**

# Description : Module config.

#

# Input : None

#

# Output : None

### 4.2 行注释

在使用行注释的时候，在**代码句子结束之后至少两个空格**，**然后用#开头后跟一个空格**

x = x + 1                 # Increment x

But sometimes, this is useful:

x = x + 1                 # Compensate for border

### 4.3 文档注释

在所有的公共模块，函数，类，方法中加入文档注释，这些注释写在def之后。

在进行多行注释的时候，注意**“”“结束的时候，必须独占一行**，如下：

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

当文档注释是一行的时候，确保开始的““”和“”“在同一行中。

## 命名规范

使用单独的小写字母（b）

使用单独的大写字母（B）

使用小写字母（lowercase）

使用小写字母和下划线（lower\_case\_with\_underscores）

使用大写字母（UPPERCASE）

使用大写字母和下划线（UPPER\_CASE\_WITH\_UPPERCASE）

驼峰式写法（CamelCase）：**在使用缩写的时候，大写优于小写**例如HTTPServer优于HttpServer

首字母大写，然后使用下划线是一种**丑陋的**写法

### 5.1 避免使用的名称

在写变量的时候，尽量避免小写的l和大写字母O和大写字母I，主要原因是容易和数字中1,0相混淆。

### 5.2 包和模块的名称

**模块**尽量使用简短的**全部小写**的名称，如果可以增加可读性那么**可以使用下划线**，**[Python](http://lib.csdn.net/base/python" \o "Python知识库" \t "http://blog.csdn.net/kellyseeme/article/details/_blank)**的**包不推荐使用下划线**，但是在引用其他语言写的扩展包中可以使用下划线来表示区分。

### 5.3 类名称

类名称主要遵循为CapWords约定，表示为首字母大写。模块内部使用的类采用\_CapWords的方式。

### 5.4 异常名称

异常归于类，从而也要遵循类名的规范，主要是在后缀上必须添加“Error“

### 5.5 全局变量名

全局变量只在模块类有效，和function命名相同。全局变量尽量只在模块内有效，类似C语言中的static。实现方法有两种，一是\_\_all\_\_机制;二是前缀一个下划线。

### 5.6 方法名称

方法名称全部为小写，**下划线是可选的**（在增加可读性的基础上使用）。

### 5.7方法变量

类的方法第一个参数总是self

类方法的静态变量总是为crs

**如果一个方法的参数和保留字相冲突，那么在后面添加下划线进行区分**

### 5.8 常量

常量命名**全部使用大写**，可以使用下划线进行分割。

### 5.9 类的属性方法命名

* 类的属性（方法和变量）命名使用全部小写的方式，可以使用下划线。
* 类的属性有3种作用域public、non-public和subclass API，可以理解成C++中的public、private、protected，non-public属性前，前缀一条下划线。
* 类的属性若**与关键字名字冲突，后缀**一下划线，尽量不要使用缩略等其他方式。
* **为避免与子类属性命名冲突，在类的一些属性前，前缀两条下划线**。比如：类Foo中声明\_\_a,访问时，只能通过Foo.\_Foo\_\_a，避免歧义。如果子类也叫Foo，那就无能为力了。
* 数据值应该使用名词作为名字，方法使用谓词（动词加对象）

## 6.编码建议

### 6.1单独比较的时候使用is或者is not，不要使用==进行比较。

### 6.2当实现比较的方法的时候，最好全部实现

\_\_eq\_\_ , \_\_ne\_\_ ,\_\_lt\_\_ , \_\_le\_\_ , \_\_gt\_\_ , \_\_ge\_\_ )，而不要单独实现一个。

### **6.3 使用startswith() and endswith()**代替切片进行序列前缀或后缀的检查。

比如

Yes: if foo.startswith(‘bar’):优于

No: if foo[:3] == ‘bar’:

### 6.4 编码中考虑到其他python实现的效率等问题

比如运算符‘+’在CPython（Python）中效率很高，都是Jython中却非常低，所以应该采用.join()的方式。

### 6.5使用isinstance()比较对象的类型。

比如  
Yes: if isinstance(obj, int): 优于  
No: if type(obj) is type(1):

### 6.6 异常编写

* 使用基于类的异常，每个模块或包都有自己的异常类，此异常继承自Exception。
* 异常中不要使用裸露的except，except后跟具体的exceptions。
* 异常中try的代码尽量少。

try:

value = collection[key]

except KeyError:

return key\_not\_found(key)

else:

return handle\_value(value)

要优于

try:

# Too broad!

return handle\_value(collection[key])

except KeyError:

# Will also catch KeyError raised by handle\_value()

return key\_not\_found(key)

### 6.7 判断序列空或不空

有如下规则  
Yes: if not seq:  
if seq:  
优于  
No: if len(seq)  
if not len(seq)

### 6.8 字符串不要以空格收尾。

### 6.9 二进制数据判断使用 if boolvalue的方式。

## 编码示例

### **7.1变量**

**常量**：大写加下划线

IMG_256USER\_CONSTANT

对于不会发生改变的全局变量，使用大写加下划线。  
**私有变量**: 小写和一个前导下划线

IMG_257\_private\_value

Python 中不存在私有变量一说，若是遇到需要保护的变量，使用小写和一个前导下划线。但这只是程序员之间的一个约定，用于警告说明这是一个私有变量，外部类不要去访问它。但实际上，外部类还是可以访问到这个变量。  
**内置变量**: 小写，两个前导下划线和两个后置下划线

IMG_258\_\_class\_\_

两个前导下划线会导致变量在解释期间被更名。这是为了避免内置变量和其他变量产生冲突。用户定义的变量要严格避免这种风格。以免导致混乱。  
  
7.2函数和方法  
总体而言应该使用，小写和下划线。但有些比较老的库使用的是混合大小写，即首单词小写，之后每个单词第一个字母大写，其余小写。但现在，小写和下划线已成为规范。  
**私有方法** ： 小写和一个前导下划线

IMG_259def \_secrete(self):  
IMG_260    print "don't test me."

这里和私有变量一样，并不是真正的私有访问权限。同时也应该注意一般函数不要使用两个前导下划线(当遇到两个前导下划线时，Python 的名称改编特性将发挥作用)。特殊函数后面会提及。  
**特殊方法**： 小写和两个前导下划线，两个后置下划线

IMG_261def \_\_add\_\_(self, other):  
IMG_262    return int.\_\_add\_\_(other)

这种风格只应用于特殊函数，比如操作符重载等。  
**函数参数**: 小写和下划线，缺省值等号两边无空格

IMG_263def connect(self, user=None):  
IMG_264    self.\_user = user

### 7.2 Should a line break before or after a binary operator? 在二元运算符之前应该换行吗？

# 不推荐: 操作符离操作数太远

income = (gross\_wages +

taxable\_interest +

(dividends - qualified\_dividends) -

ira\_deduction -

student\_loan\_interest)

# 推荐：运算符和操作数很容易进行匹配

income = (gross\_wages

+ taxable\_interest

+ (dividends - qualified\_dividends)

- ira\_deduction

- student\_loan\_interest)