python 编程规范

PEP8 是 Python 官网指导推荐的编程规范，OpenStack 等项目都遵循它。上元也要求严格遵循该规范。本文中附着了 PEP8 原文，同时附带一篇中文翻译稿。推荐使用 PyChram 编辑 python 代码，它可自动提示那些不满足编程规范的代码, PyChram 已经默认导入 PEP8 规范。如果已经习惯用 Eclipse的 PyDev ，可以用文中附件--pep8 检测工具检测自己的代码是否符合 PEP8 规范。

目录

1. 代码编排

1.1. 缩进

4 个空格的缩进（编辑器都可以完成此功能），不使用 Tap，更不能混合使用 Tap 和空格。

以下是带括号的一些缩进原则。

Yes:

#和括号开始的部分对齐

```
foo = long_function_name(var_one, var_two,
var_three, var_four)
```

No:

```
# 禁止对齐下一层的代码
foo = long_function_name(var_one, var_two,
var_three, var_four)
```

No:

```
# 需要进一层的缩进，区分下一层的代码
def long_function_name(
var_one, var_two, var_three,
var_four):
print(var_one)
```

如果 if语句占用多行，推荐不需要特殊的缩进

```
#不需要特殊的缩进
if (this
and that):
do_something()
```

在闭合的括号中，后面的括号对齐变量名：

```
my_list = [
1, 2, 3,
4, 5, 6,
]
result = some_function_that_takes_arguments(
'a', 'b', 'c',
'd', 'e', 'f',
)
```

1.2. 每行最大长度 79

换行可以使用反斜杠，最好使用圆括号。换行点要在操作符的后边敲回车。

较长代码行折行的首选方法是在圆括号、方括号和花括号内使用 Python 的隐式续行方式。通过圆括号内的表达式的折行来把较长的代码行折成多行。同时确保适当的续行缩进。二元运算符的首选的折行处是在运算符之后，而不是之前。

```
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)
```

1.3. 空行规则

类和顶层函数定义之间空两行；

类中的方法定义之间空一行；

函数内逻辑无关段落之间空一行；

其他地方尽量不要再空行。

```
import xxx


class Rectangle(Blob):
```

```
def __init__(self, width, height,
color='black', emphasis=None, highlight=0):
... ...

def abc(self):
... ...
```

1.4.编码方式

Python 文件必须使用 UTF-8 格式。IDE 文件编码方式设置成 UTF-8。

python 文件第一行写上，这样才能写中文注释

```
# -*- coding: utf-8 -*-
```

在代码中的字符串应该使用 ，\u, \或者\N 来包含非ASCII 数据。

1.5. Import

1.5.1.不要在一句 import 中多个库

Yes:

```
import os
import sys
```

No:

```
import sys, os
```

1.5.2. import 库顺序

按标准、第三方，上元外部模块，上元内部模块顺序依次排放，之间空一行。

```
import sys

Import os
```

```
from twisted.application import service

from ozcommon import configruration
```

1.5.3.使用绝对路径而不是相对路径

Yes：

```
import mypkg.sibling
from mypkg import sibling
from mypkg.sibling import example
```

No ：

```
from .import sibling
from .sibling import example
```

1.5.4. import所有(from import *)应该避免使用

2. 空格的使用

总体原则，避免不必要的空格。

2.1.各种右括号前不要加空格

Yes:

```
spam(ham[1], {eggs: 2}, [])
```

No:

```
spam( ham[ 1 ], { eggs: 2 }, [ ] )
```

2.2.逗号、冒号、分号前不要加空格，在他们后面加

Yes

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

No

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

```
x , y = y , x
```

2.3. 参数列表,索引或切片的左括号前不应加空格

```
Yes
spam(1)
dict['key'] = list[index]
No
spam (1)
dict ['key'] = list [index]
```

2.4. 在二元操作符两边都加上一个空格

比如赋值(=),比较(==, , !=, <>, <=, >=, in, not in, is, is not),布尔(and, or, not)

```
Yes
x == 1
y = 1
No
x<1
x                = 1
y                = 2
long_variable = 3
```

2.5. 在算术运算符前后加空格

```
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)
```

2.6.不要在一个关键字参数或者一个缺省参数值的 = 符号前后加一个空格

```
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)
```

2.7.通常不推荐使用复合语句（一行代码中有多条语句）

```
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()
```

2.8.通常情况下，一行代码包括一个小的 if/for/while块，是可以的。但是多子句绝不可以。同样，需要避免折叠类似的长代码行！

```
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()
```

3. 注释

不好理解的注释不如没有注释。注释要和代码保持与时俱进！注释应该是一条完整的句子。代码注释请用中文，除非语法特殊字符或者变量名等。

3.1. 块注释

在一段代码前增加的注释。在‘#’后加一空格。段落之间以只有‘#’的行间隔。比如：

```
    # 这是块注释段落 1
    #
    # 这是块注释段落 2
    #
    # 这是块注释段落 3
```

3.2. 行注释

在一句代码后加注释。

如果语句显而易见，那么内嵌注释是不必要的，比如：

```
x = x + 1   # 加 1
```

而应该是有意义的注释。格式要求：语句后面要加 2 个空格，在加# 号，之后再加一个空格，之后再写注释。

```
    x = x + 1    #边界弥补
```

避免无谓的注释

3.3. 文档描述

为所有的共有模块、函数、类、方法写文档描述；非共有的没有必要，但是可以写注释（在 def 的下一行）。

注意最重要的是，’’’作为多行的文档字符串的结束，应该单独一行，并且之前有一个空行

’’返回卷对象
该卷对象为你输入 src_vol卷的克隆
’’’

4. 命名规范

4.1. 命名风格

有如下命名规范：
lowercase 全小写字母
lower_case_with_underscores 使用下划线分隔的小写字母
UPPERCASE 大写字母
UPPER_CASE_WITH_UNDERSCORES 使用下划线分隔的大写字母

CapitalizedWords 多个单词无分隔符连接，每个单词首字母大写。大驼峰命名法。

名称前后下划线有特殊意义：
单下划线开始指弱’内部使用’。例如：from M import * 不会导入以下划线开始的对象。
_single_leading_underscore
单下划线结束用来避免与 Python 关键字冲突
Tkinter.Toplevel(master, class_=’ClassName’)
双下划线开始，该方法不会被子类继承。

```
class A(object):
def __init__(self):
self.__private()
self.public()
```

```
def __private(self):
print 'A.__private()'

def public(self):
print 'A.public()'


class B(A):
def __private(self):
    print 'B.__private()'

    def public(self):
    print 'B.public()'


    b = B()
```

上述代码正确的答案是：

A.__private()

B.public()

4.2. 避免使用的名字

永远不要使用‘1’（小写的L），‘0’（大写的 o）和 I（大写的 i）作为单字变量名。

在某些字体中，这些字很难和数字的 0 和 1 区分。当打算用’l’的时候,用’L’来代替。

4.3. 包和模块名称

模块应该用简短的，全小写的名字。如果能增强可读性的话，可以使用下划线。 Python 的包也要用全小写的，短名称，但是不建议用