Python 编程规范 v2

执行

- 本规范使用 pylint 及对应的配置文件来进行检测，关于 pylint 的安装和配置见：http://blog.csdn.net/lanphaday/article/details/6089902

编码

- 所有的 Python 脚本文件都应在文件头标上如下标识或其兼容格式的标识：

[python] view plaincopy

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

- 设置编辑器，默认保存为 utf-8 格式。

注释

- 业界普遍认同 Python 的注释分为两种的概念，一种是由 # 开头的“真正的”注释，另一种是 docstrings。前者表明为何选择当前实现以及这种实现的原理和难点，后者表明如何使用这个包、模块、类、函数（方法），甚至包括使用示例和单元测试。
- 坚持适当注释原则。对不存在技术难点的代码坚持不注释，对存在技术难点的代码必须注释。但与注释不同，推荐对每一个包、模块、类、函数（方法）写 docstrings，除非代码一目了然，非常简单。

格式

缩进

- Python 依赖缩进来确定代码块的层次，行首空白符主要有两种：tab 和空格，但严禁两者混用。
- 公司内部使用 2 个空格的 tab 进行缩进。

空格

- 空格在 Python 代码中是有意义的，因为 Python 的语法依赖于缩进，在行首的空格称为前导空格。在这一节不讨论前导空格相关的内容，只讨论非前导空格。非前导空格在 Python 代码中没有意义，但适当地加入非前导空格可以增进代码的可读性。
- 在二元算术、逻辑运算符前后加空格，如：

[python] view plaincopy

```
1.  a = b + c
```

- “:”用在行尾时前后皆不加空格，如分枝、循环、函数和类定义语言；用在非行尾时两端加空格，如 dict 对象的定义：

[python] view plaincopy

```
d = {'key' : 'value'}
```

- 括号（含圆括号、方括号和花括号）前后不加空格，如：

[html] view plaincopy

```
do_something(arg1, arg2)
```

而不是

[python] view plaincopy

```
do_something( arg1, arg2 )
```

- 逗号后面加一个空格，前面不加空格；

空行

- 适当的空行有利于增加代码的可读性，加空行可以参考如下几个准则：
    - 在类、函数的定义间加空行；
    - 在 import 不同种类的模块间加空行；
    - 在函数中的逻辑段落间加空行，即把相关的代码紧凑写在一起，作为一个逻辑段落，段落间以空行分隔；

断行

- 尽管现在的宽屏显示器已经可以单屏显示超过 256 列字符，但本规范仍然坚持行的最大长度不得超过 78 个字符的标准。折叠长行的方法有以下几种方法：
    - 为长变量名换一个短名，如：

[python] view plaincopy

```
this._is.a.very.long.variable_name = this._is.another.long.variable_name
```

应改为：

[python] view plaincopy

```
variable_name1 = this._is.a.very.long.variable_name
variable_name2 = this._is.another.variable_name
variable_name1 = variable_name2s
```

    - 在括号（包括圆括号、方括号和花括号）内换行，如：

[python] view plaincopy

```
class Edit(Widget):
  def __init__(self, parent, width,
      font = FONT, color = BLACK, pos = POS, style = 0): # 注意：多一层缩进
    pass
```

或：

[python] view plaincopy

```
very_very_very_long_variable_name = Edit(parent,
  width,
  font,
  color,
  pos) # 注意：多一层缩进
do_sth_with(very_very_very_long_variable_name)
```

- 如果行长到连第一个括号内的参数都放不下，则每个元素都单独占一行：

[python] view plaincopy

```
very_very_very_long_variable_name = ui.widgets.Edit(
  panrent,
  width,
  font,
  color,
  pos) # 注意：多一层缩进
do_sth_with(very_very_very_long_variable_name)
```

- 在长行加入续行符强行断行，断行的位置应在操作符前，且换行后多一个缩进，以使维护人员看代码的时候看到代码行首即可判定这里存在换行，如：

[html] view plaincopy

```
if color == WHITE or color == BLACK \
    or color == BLUE: # 注意 or 操作符在新行的行首而不是旧行的行尾，上一行的续行符不可省略
  do_something(color);
else:
  do_something(DEFAULT_COLOR);
```

命名

- 一致的命名可以给开发人员减少许多麻烦，而恰如其分的命名则可以大幅提高代码的可读性，降低维护成本。

常量

- 常量名所有字母大写，由下划线连接各个单词，如：

[python] view plaincopy

```
WHITE = 0xffffffff
THIS_IS_A_CONSTANT = 1
```

变量

- 变量名全部小写，由下划线连接各个单词，如：

[python] view plaincopy

```
color = WHITE
this_is_a_variable = 1
```

- 不论是类成员变量还是全局变量，均不使用 m 或 g 前缀。私有类成员使用单一下划线前缀标识，多定义公开成员，少定义私有成员。

- 变量名不应带有类型信息，因为 Python 是动态类型语言。如 iValue、names_list、dict_obj 等都是不好的命名。

函数

- 函数名的命名规则与变量名相同。

类

- 类名单词首字母大写，不使用下划线连接单词，也不加入 C、T 等前缀。如：

[python] view plaincopy

```
class ThisIsAClass(object):
  passs
```

模块

- 模块名全部小写，对于包内使用的模块，可以加一个下划线前缀，如：

[python] view plaincopy

```
module.py
_internal_module.py
```

包

- 包的命名规范与模块相同。

缩写

- 命名应当尽量使用全拼写的单词，缩写的情况有如下两种：
    - 常用的缩写，如 XML、ID 等，在命名时也应只大写首字母，如：

[python] view plaincopy

```python
class XmlParser(object):pass
```

    - 命名中含有长单词，对某个单词进行缩写。这时应使用约定成俗的缩写方式，如去除元音、包含辅音的首字符等方式，例如：
        - function 缩写为 fn
        - text 缩写为 txt
        - object 缩写为 obj
        - count 缩写为 cnt
        - number 缩写为 num，等。

特定命名方式

- 主要是指 __xxx__ 形式的系统保留字命名法。项目中也可以使用这种命名，它的意义在于这种形式的变量是只读的，这种形式的类成员函数尽量不要重载。如：

[python] view plaincopy

```python
class Base(object):
  def __init__(self, id, parent = None):
    self.__id__ = id
    self.__parent__ = parent
  def __message__(self, msgid):
    # …略
```

其中 __id__、__parent__ 和 __message__ 都采用了系统保留字命名法。

语句

import

- import 语句有以下几个原则需要遵守：

- import 的次序，先 import Python 内置模块，再 import 第三方模块，最后 import 自己开发的项目中的其它模块；这几种模块中用空行分隔开来。
- 一条 import 语句 import 一个模块。
- 当从模块中 import 多个对象且超过一行时，使用如下断行法（此语法 py2.5 以上版本才支持）：

[python] view plaincopy

```
from module import (obj1, obj2, obj3, obj4,
    obj5, obj6)
```

- 不要使用 from module import *，除非是 import 常量定义模块或其它你确保不会出现命名空间冲突的模块。

赋值

- 对于赋值语句，主要是不要做无谓的对齐，如：

[python] view plaincopy

```
a        = 1                # 这是一个行注释
variable = 2                # 另一个行注释
fn       = callback_function  # 还是行注释
```

没有必要做这种对齐，原因有两点：一是这种对齐会打乱编程时的注意力，大脑要同时处理两件事（编程和对齐）；二是以后阅读和维护都很困难，因为人眼的横向视野很窄，把三个字段看成一行很困难，而且维护时要增加一个更长的变量名也会破坏对齐。直接这样写为佳：

[python] view plaincopy

```
a = 1 # 这是一个行注释
variable = 2 # 另一个行注释
fn = callback_function # 还是行注释
```

分枝和循环

- 对于分枝和循环，有如下几点需要注意的：
  - 不要写成一行，如：

[python] view plaincopy

```
if not flg: pass
```

和

[python] view plaincopy

```
1.  for i in xrange(10): print i
```

都不是好代码，应写成

[python] view plaincopy

```
1.  if not flg:
2.    pass
3.  for i in xrange(10):
4.    print i
```

注：本文档中出现写成一行的例子是因为排版的原因，不得作为编码中不断行的依据。

- 条件表达式的编写应该足够 pythonic，如以下形式的条件表达式是拙劣的：

[python] view plaincopy

```
1.  if len(alist) != 0: do_something()
2.  if alist != []: do_something()
3.  if s != "": do_something()
4.  if var != None: do_something()
5.  if var != False: do_something()
```

上面的语句应该写成：

[python] view plaincopy

```
1.  if seq: do_somethin() # 注意，这里命名也更改了
2.  if var: do_something()
```

- 用得着的时候多使用循环语句的 else 分句，以简化代码。

已有代码

- 对于项目中已有的代码，可能因为历史遗留原因不符合本规范，应当看作可以容忍的特例，允许存在；但不应在新的代码中延续旧的风格。
- 对于第三方模块，可能不符合本规范，也应看作可以容忍的特例，允许存在；但不应在新的代码中使用第三方模块的风格。
- tab 与空格混用的缩进是"'不可容忍'"的，在运行项目时应使用 -t 或 -tt 选项排查这种可能性存在。出现混用的情况时，如果是公司开发的基础类库代码，应当通知类库维护人员修改；第三方模块则可以通过提交 patch 等方式敦促开发者修正问题。

已有风格

- 开发人员往往在加入项目之前已经形成自有的编码风格，加入项目后应以本规范为准编写代码。特别是匈牙利命名法，因为带有类型信息，并不适合 Python 编程，不应在 Python 项目中应用。