python语言编码规范

# 命名规则

## 文件的命名

* 文件名全小写,可使用下划线

## 类的命名（适用于python）

* 总是使用首字母大写单词串。
* 内部类可以使用额外的前导下划线

例如：

MyClass

## 包命名（适用于python）

* 应该是简短的、小写的名字。
* 如果下划线可以改善可读性可以加入

例如：

mypackage

## 方法和函数参数的命名（适用于python）

* 函数名应该为小写，可以用下划线风格单词以增加可读性。
* 混合大小写仅被允许用于这种风格已经占据优势的时候，以便保持向后兼容。
* 函数和方法的参数总使用“self”作为实例方法的第一个参数。

## 变量的命名

* 所有字母都用小写
* 使用下划线 '\_' 作为单词的分隔。
* 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。
* 常量名所有字母大写，由下划线连接各个单词如MAX\_OVERFLOW，TOTAL。

例如：

this\_is\_a\_variable= 1

# 排版规则

## 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

## 缩进

* 用4个空格来缩进代码。
* 不要使用TAB，用空格，大多数编辑器可以用空格代替TAB。TAB应固定8个空格，因为大多数编辑器都是这么设置的。
* 虽然没有规定缩进的层次，但是4至5层是合适的。如果缩进的层次太多，你可能需要考虑是否进行代码重构了。

## 空行和空格

* 顶级定义之间空两行, 方法定义之间空一行。
* 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. 不要在逗号, 分号, 冒号前面加空格, 但应该在它们后面加(除了在行尾)。
* 不要用空格来垂直对齐多行间的标记。

## 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.

## 导入格式

* 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。
* 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。

# 编码要求

## 异常（适用于python）

异常必须遵守特定条件:

* 像这样触发异常: raise MyException("Error message") 或者 raise MyException . 不要使用两个参数的形式( raise MyException, "Error message" )或者过时的字符串异常( raise "Error message" ).
* 模块或包应该定义自己的特定域的异常基类, 这个基类应该从内建的Exception类继承. 模块的异常基类应该叫做”Error”.
* 永远不要使用except语句来捕获所有异常, 也不要捕获Exception或者StandardError， 除非你打算重新触发该异常, 或者你已经在当前线程的最外层(记得还是要打印一条错误消息).
* 尽量减少try/except块中的代码量. try块的体积越大, 期望之外的异常就越容易被触发. 这种情况下, try/except块将隐藏真正的错误.
* 使用finally子句来执行那些无论try块中有没有异常都应该被执行的代码. 这对于清理资源常常很有用, 例如关闭文件.
* 当捕获异常时, 使用as 而不要用逗号

## 嵌套/局部/内部类或函数（适用于python）

* 类可以定义在方法, 函数或者类中. 函数可以定义在方法或函数中. 封闭区间中定义的变量对嵌套函数是只读的，推荐使用

## 列表推导（适用于python）

* 可以在简单情况下使用，列表推导与生成器表达式提供了一种简洁高效的方式来创建列表和迭代器, 而不必借助map(), filter(), 或者lambda.
* 适用于简单情况. 每个部分应该单独置于一行。
* 映射表达式, for语句, 过滤器表达式. 禁止多重for语句或过滤器表达式. 复杂情况下还是使用循环。

## 默认迭代器和操作符（适用于python）

* 如果类型支持, 就使用默认迭代器和操作符. 比如列表, 字典及文件等. 容器类型, 像字典和列表, 定义了默认的迭代器和关系测试操作符(in和not in) 。
* 优先考虑这些方法, 而不是那些返回列表的方法. 当然，这样遍历容器时，你将不能修改容器。

## 生成器（适用于python）

* 所谓生成器函数, 就是每当它执行一次生成(yield)语句, 它就返回一个迭代器, 这个迭代器生成一个值. 生成值后, 生成器函数的运行状态将被挂起, 直到下一次生成.
* 鼓励使用. 注意在生成器函数的文档字符串中使用”Yields:”而不是”Returns:”

## Lambda函数（适用于python）

* 与语句相反, lambda在一个表达式中定义匿名函数. 常用于为 map() 和 filter() 之类的高阶函数定义回调函数或者操作符。
* 适用于单行函数. 如果代码超过60-80个字符, 最好还是定义成常规(嵌套)函数.
* 对于常见的操作符，例如乘法操作符，使用 operator 模块中的函数以代替lambda函数.

## 条件表达式（适用于python）

* 条件表达式是对于if语句的一种更为简短的句法规则。
* 适用于单行函数. 在其他情况下，推荐使用完整的if语句

## 默认参数值（适用于python）

* 可以在函数参数列表的最后指定变量的值, 如果调用foo时只带一个参数, 则b被设为0. 如果带两个参数, 则b的值等于第二个参数。
* 不要在函数或方法定义中使用可变对象作为默认值.

例如：

**def** foo(a, b=None):

**if** b **is** None:

b = []

## True/False的求值（适用于python）

* Python在布尔上下文中会将某些值求值为false. 按简单的直觉来讲, 就是所有的”空”值都被认为是false. 因此0， None, [], {}, “” 都被认为是false.
* 尽可能使用隐式的false

## 函数与方法装饰器（适用于python）

* 用于函数及方法的装饰器 (也就是@标记). 最常见的装饰器是@classmethod 和@staticmethod, 用于将常规函数转换成类方法或静态方法.
* 避免装饰器自身对外界的依赖(即不要依赖于文件, socket, 数据库连接等), 因为装饰器运行时这些资源可能不可用(由 pydoc 或其它工具导入).
* 应该保证一个用有效参数调用的装饰器在所有情况下都是成功的.

## 线程（适用于python）

* 虽然Python的内建类型例如字典看上去拥有原子操作, 但是在某些情形下它们仍然不是原子的
* 优先使用Queue模块的 Queue 数据类型作为线程间的数据通信方式.

## TODO注释（适用于python）

* 为临时代码使用TODO注释, 它是一种短期解决方案
* 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.