**Coding Conventions for Python**

Let us follow some basic conventions so that our code is easily understood by others. Most of these coventions follow the PEP 8 -- Style Guide for Python Code given at http://legacy.python.org/dev/peps/pep-0008/

**1. Code Layout:**

**(i) Indentation:**

Use 4 spaces per indentation level.

For functions with many arguments, continuation lines should align wrapped elements vertically using Python's implicit line joining inside parentheses, brackets and braces, e.g.,

Yes:

# Aligned with opening delimiter.

foo = long\_function\_name(var\_one, var\_two,

var\_three, var\_four)

**(ii) Tabs or Spaces?**

Spaces are the preferred indentation method.

Python 3 disallows mixing the use of tabs and spaces for indentation.

Python 2 code indented with a mixture of tabs and spaces should be converted to using spaces exclusively.

When invoking the Python 2 command line interpreter with the -t option, it issues warnings about code that illegally mixes tabs and spaces. When using -tt these warnings become errors. These options are highly recommended!

**(iii) Imports**

Do not pollute namespaces. So use variable names for your package, e.g.:

import numpy as np

import pandas as pd

Imports should usually be on separate lines, e.g.:

Yes: import os

import sys

No: import sys, os

It's okay to say this though:

from subprocess import Popen, PIPE

Imports are always put at the top of the file, just after any module comments and docstrings, and before module globals and constants.

Imports should be grouped in the following order:

standard library imports

related third party imports

local application/library specific imports

You should put a blank line between each group of imports.

**2. Whitespace in Expressions and Statements**

**(i) Pet Peeves**

Avoid extraneous whitespace in the following situations:

Immediately inside parentheses, brackets or braces.

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

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

Immediately before a comma, semicolon, or colon:

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

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

Immediately before the open parenthesis that starts the argument list of a function call:

Yes: spam(1)

No: spam (1)

Immediately before the open parenthesis that starts an indexing or slicing:

Yes: dict['key'] = list[index]

No: dict ['key'] = list [index]

More than one space around an assignment (or other) operator to align it with another.

Yes:

x = 1

y = 2

long\_variable = 3

No:

x = 1

y = 2

long\_variable = 3

**(ii) Other Recommendations**

Always surround binary operators with a single space on either side.

If operators with different priorities are used, consider adding whitespace around the operators with the lowest priority(ies). Use your own judgment; however, never use more than one space, and always have the same amount of whitespace on both sides of a binary operator.

Don't use spaces around the = sign when used to indicate a keyword argument or a default parameter value.

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)

Compound statements (multiple statements on the same line) are generally discouraged.

**3. Comments**

**(i) Always make a priority of keeping the comments up-to-date when the code changes!**

Comments should be complete sentences. If a comment is a phrase or sentence, its first word should be capitalized, unless it is an identifier that begins with a lower case letter (never alter the case of identifiers!).

If a comment is short, the period at the end can be omitted. Block comments generally consist of one or more paragraphs built out of complete sentences, and each sentence should end in a period.

You should use two spaces after a sentence-ending period.

When writing English, Strunk and White apply.

Python coders from non-English speaking countries: please write your comments in English, unless you are 120% sure that the code will never be read by people who don't speak your language.

**(ii)Block Comments**

Block comments generally apply to some (or all) code that follows them, and are indented to the same level as that code. Each line of a block comment starts with a # and a single space (unless it is indented text inside the comment).

Paragraphs inside a block comment are separated by a line containing a single #.

**(iii)Inline Comments**

Use inline comments sparingly.

An inline comment is a comment on the same line as a statement. Inline comments should be separated by at least two spaces from the statement. They should start with a # and a single space.

Inline comments are unnecessary and in fact distracting if they state the obvious. Don't do this:

x = x + 1 # Increment x

But sometimes, this is useful:

x = x + 1 # Compensate for border

**4. Version Bookkeeping**

If you have to have Subversion, CVS, or RCS crud in your source file, do it as follows.

\_\_version\_\_ = "$Revision: e7bf114ee31a $"

# $Source$

These lines should be included after the module's docstring, before any other code, separated by a blank line above and below.

**5. Naming Conventions**

**(i)Names to Avoid**

Never use the characters 'l' (lowercase letter el), 'O' (uppercase letter oh), or 'I' (uppercase letter eye) as single character variable names.

In some fonts, these characters are indistinguishable from the numerals one and zero. When tempted to use 'l', use 'L' instead.

**(ii)Package and Module Names**

Modules should have short, all-lowercase names. Underscores can be used in the module name if it improves readability. Python packages should also have short, all-lowercase names, although the use of underscores is discouraged.

**(iii)Class Names**

Class names should normally use the CapWords convention.

**(iv)Exception Names**

Because exceptions should be classes, the class naming convention applies here. However, you should use the suffix "Error" on your exception names (if the exception actually is an error).

**(v)Global Variable Names**

(Let's hope that these variables are meant for use inside one module only.) The conventions are about the same as those for functions.

Modules that are designed for use via from M import \* should use the \_\_all\_\_ mechanism to prevent exporting globals, or use the older convention of prefixing such globals with an underscore (which you might want to do to indicate these globals are "module non-public").

**(vi)Function Names**

Function names should be lowercase, with words separated by underscores as necessary to improve readability.

mixedCase is allowed only in contexts where that's already the prevailing style (e.g. threading.py), to retain backwards compatibility.

**(vii)Function and method arguments**

Always use self for the first argument to instance methods.

Always use cls for the first argument to class methods.

If a function argument's name clashes with a reserved keyword, it is generally better to append a single trailing underscore rather than use an abbreviation or spelling corruption. Thus class\_ is better than clss. (Perhaps better is to avoid such clashes by using a synonym.)

**(viii)Method Names and Instance Variables**

Use the function naming rules: lowercase with words separated by underscores as necessary to improve readability.

Use one leading underscore only for non-public methods and instance variables.

**(ix)Constants**

Constants are usually defined on a module level and written in all capital letters with underscores separating words. Examples include MAX\_OVERFLOW and TOTAL.