**PEP -8 guidelines (summary for limited use cases)**

* Consistency with this style guide is important.
  + Consistency ***within a project*** is more important.
    - Consistency ***within one module or function*** is the most important.

**Indentation**

- Use 4 spaces per indentation level.

**Continuation lines**

***# Aligned with opening delimiter.***

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

var\_three, var\_four)

***# More indentation included to distinguish this from the rest.***

def long\_function\_name(

var\_one, var\_two, var\_three,

var\_four):

print(var\_one)

***# Hanging indents should add a level.***

foo = long\_function\_name(

var\_one, var\_two,

var\_three, var\_four)

The closing brace/bracket/parenthesis on multi-line constructs may either line up under the first non-whitespace character of the last line of list, as in:

my\_list = [

1, 2, 3,

4, 5, 6,

]

result = some\_function\_that\_takes\_arguments(

'a', 'b', 'c',

'd', 'e', 'f',

)

***Maximum Line Length***

Limit all lines to a maximum of 79 characters.

* The preferred way of wrapping long lines is by using Python’s implied line continuation inside parentheses, brackets and braces. Long lines can be broken over multiple lines by wrapping expressions in parentheses.

Following the tradition from mathematics usually results in more readable code:

**Breaking the line**

***# Yes: easy to match operators with operands***

income = (gross\_wages

+ taxable\_interest

+ (dividends - qualified\_dividends)

- ira\_deduction

- student\_loan\_interest)

**Source File Encoding**

* Code in the core Python distribution should always use UTF-8 (or ASCII in Python 2).
* Files using ASCII (in Python 2) or UTF-8 (in Python 3) should not have an encoding declaration.

**Imports**

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

Yes:

*import os*

*import sys*

No:

*import os, sys*

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.

**Use of whitespace**

Avoid trailing whitespace anywhere.

Always surround these binary operators with a single space on either side: assignment (**=**), augmented assignment (**+=**, **-=** etc.), comparisons (**==**, **<**, **>**, **!=**, **<>**, **<=**, **>=**, **in**, **not in**, **is**, **is not**), Booleans (**and**, **or**, **not**).

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)

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)

While sometimes it’s okay to put an if/for/while with a small body on the same line, never do this for multi-clause statements. Also avoid folding such long lines!

Rather not:

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

for x in lst: total += x

while t < 10: t = delay()

**Comments**

Comments that contradict the code are worse than no comments. 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!).

**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).

**Documentation Strings**

Write docstrings for all public modules, **functions**, classes, and **methods**. Docstrings are not necessary for non-public methods, but you should have a comment that describes what the method does. This comment should appear after the def line.

The """ that ends a multiline docstring should be on a line by itself, e.g.:

*"""Return a foobang*

*Optional plotz says to frobnicate the bizbaz first.*

*"""*

For one liner docstrings, please keep the closing """ on the same line.

**Naming conventions**

single\_trailing\_underscore\_: used by convention to avoid conflicts with Python keyword, e.g.:

*Tkinter.Toplevel(master, class\_='ClassName')*

***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.

***Function Names***

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

***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.

**Programming recommendations**

Use ***is not*** operator rather than ***not ... is***. While both expressions are functionally identical, the former is more readable and preferred.

E.g.:

*if foo* ***is not*** *None:*

Always use a def statement instead of an assignment statement that binds a lambda expression directly to an identifier.

Yes:

*def f(x): return 2\*x*

No:

*f = lambda x: 2\*x*

Don’t compare boolean values to True or False using ==:

Yes: *if greeting:*

No: *if greeting == True:*

Worse: *if greeting is True:*