Correct:

Aligned with opening delimiter: foo = long -
function_name (var_one, var_two,
var_three, var_four)

Adds spaces (an extra level of indentation) to distinguish
the arguments from the rest.

def long_function_name (
    var_one, var_two,
    var_three, var_four)

wrong:

Arguments on first line forbidden when not using
vertical alignment:

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

further indentation required as indentation is not
distinguishable

def long_function_name (
    var_one, var_two, var_three,
    var_four):
print (var_one)

The 4-Space rule is optional for continuation lines.

Optional:

hanging indents * may * be indented to other than 4 Spaces.

```
foo = long + function_name(
var_one, var_two,
    var_three, var_four)
```

When the conditional part of an if Statement is long enough to required that it be written across multiple lines, it's worth noting that the Combination of a two character keyword (i.e), plus a single Space, plus an Opening parent hesis creates a nature 4-Space indent for the Subsequent line of the multiline conditional. This can produces a visual Conflict with the indented Suite of code nested inside the if Statement, which would also naturally be indented to 4 Spaces. The pep take no Explict position on how (or whether) to futur visually distinguish Such Conditional line from the nested Suite inside the if - Statement. Acceptable Option in this Situation include but are not limited to:

# No extra indentation.
if (this_is_one_thing and
    that_is_another_thing):
    do_something()

Add a comment, which will provide
some distinction in editors

# Supporting syntax highlighting.
if (this_is_one_thing and
        that_is_another_thing):

    # Since both conditions are true,
    we can fasbicate.

    do_something()

Add some extra indentation on the conditional
continuation line.

if (this_is_one_thing
        and
    that_is_another_thing):
        do_something()

The closing brace/bracket/parenthesis of
non-whitespace characters may either line up under the first
of last as in:

```
my list = [
    1, 2, 3
    4, 6, 6
]

result =
some function that takes argument

[
    'a', 'b', 'c'
    'd', 'e', 'f'
]
```

or it may be lined up under the first character of the line starts the multi-line construct, as in:

```
my list = [
    1, 2, 3,
    4, 3, 6
]

result =
some function that take a argument
[
    'a', 'b', 'c',
    'd', 'e', 'f',
```

Page No.
Date

Tabs or Spaces ?

Spaces are the preferred indentation method

Tabs should be used solely to remain consistent with code that is already indent of with tabs

Python disallows mixing tabs and spaces for indentation.

Maximum line length :

Limit all line to Maximum of 79 characters.

For flowing long block of text with fewer structural restrictions (docstring or comments) the line length should be limited to 72 characters.

Limiting the required editor window width makes it possible to have several files open side by side, and work well when using code review tools that present the two version in adjacent columns

The default wrapping in most tool disrupts the visual structure of the code, making it more difficult to understand. The limits are chosen to avoid wrapping in

Editors with the window width set
Even if the tool places a marker
in the final column when wrapping
some web based tools may
offer dynamic line wrapping at all

Some team strongly prefer a longer line
length. For code maintained
exclusively or predominantly by a team
that can reach aggreement on these
issues, it is okay to increase the
long length limit (up to 99 characters)
provided that comments and docstrings
are still wrapped at 72 characters.

The python standard library is conservative
and requires limiting lines to 79
characters (and docstrings/comments
to 72).

The preferred way of wrapping long lines
is by using python implied line
continuation inside parentheses, brackets
and braces. Long line can be broken
over multiple lines by wrapping
expressions in parentheses. These should
be used in preference to using a
back slash for line continuation.

Page No.
Date

Backslashes may still be appropriate at times. for Example, long multiple with-statement could not be used before python 3.10, so backslashes were acceptable for the case:

```
with open('/path/to/some/file/you/want/to/read') as file_1:
    with open('/path/to/some/file/being/written', 'w') as file_2:
        file_2.write(file_1.read())
```

(See the previous discussion on multiline if statement for further thoughts on the indentation of such multiline with statements.)

Another such case is with assert statements. Make sure to indent the continued line appropriately.

- Source file Encoding :

Code in the Core python distribution should always use UTF-8, and should not have an Encoding declaration.

In the Standard library, non-UTF-8 encodings should be used only for test purpose. use non-ASCII characters sparingly, preferably only to denote places and human names. If using non-ASCII characters as data, avoid mosiy unicode characters like Zalgo and byte order marks.

All identifiers in the python Standard library must use ASCII-only identifiers, and should use English words wherever feasible (in many cases, abbreviations and technical terms are used which aren't English).

Open Source projects with a global audience are encouraged to adopt a similar policy.

Page No.
Date

Imports :

Imports should usually be on separate lines:

# Correct:
import os
import sys

# wrong:
import sys, os

its okay to say this though:

# Correct:
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.

2. Related third party imports.

3. Local application / library specific imports

You should put a blank line between each group of imports

- Absolute imports are recommended, as they are usually more readable and tend to be better behaved (or at least give better error messages) if the import system is incorrectly configured (such as a directory ending up on sys.path):

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

Example

- However, explicit relative imports are an acceptable alternative to absolute imports, especially when dealing with complex package layouts where using absolute imports would be unnecessarily verbose

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

Standard library code should avoid complex package layouts and always use absolute imports.

When importing a class from a class-containing module, it's usually okay to spell this

```
from myclass import myclass
from foo.bar.yourclass import yourclass
```

If this spelling causes local name clashes, then spell them explicitly:

```
import myclass
import foo.bar.yourclass
```

and use "myclass.myclass" and "foo.bar.yourclass.yourclass"

Comments:

Comments that contradict that code are worse than no comments. The priority of keeping the comments always make a code to date when the code changes.

Comments should be complete sentence. The 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 generally consistence of one more paragraphs built out of complete sentence with each sentence ending in a period:

You should use two spaces after a sentence-ending period in multi-sentence comments, except after the final sentence

Ensure that your comments are clearly and easily understandable to other speakers of the language you are writing in.

Page No.
Date

Block Comments:

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

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

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

```
x = x + 1
    Increment x
```

But sometime, this is useful:

```
x = x + 1
Compensate for border
```

Document 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.

PEP 257 describes good docstring conventions. Note that the most important part of a docstring is that it ends with """. Should be on a line by itself.

```
"""Return a foobang.

Optional plotz says to frobnicate the bizbaz first.
"""
```

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

```
"""return an ex-parrot."""
```

Page No.
Date

Naming Conventions :

The naming conventions of python's library are a bit of a mess, so we'll never get this completely consistent -- nevertheless, here are the currently recommended naming standards. New modules and packages (including third party frameworks) should be written to these standards, but where an existing library has a different style, internal consistency is preferred.

Overriding principle :

Names that are visable to the user as public parts of that API should follow conventions that reflect usage rather than implementation

Descriptive : Naming Styles

There are lot of different naming styles. It helps to be able to recognized what naming style is being used, independently from what they are used for.

The following naming style are commonly existing wished.

- b (Single lower case letter)
- B (Single Uppercase letter)
- lowercase
- lower_case_with_underscores
- UpperCase
- Capitalized words (or Cap words, or Compl
  So named because bumpy look of it
  letters. This is also Somet
  known as Studly Caps
- Mixed case (differs from Capitalized w
  by initial lower case character!)
- Capitalized_words_with_underscores (ug

Page No.
Date

Prescriptive : Naming Conventions

Name to Avoid

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

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

Package and module Names:

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

When an Extension module written in C or C++ has an accompanying python module that provides a higher level

Types variable Names:

Names of type variable introduced in pep 484 should normally use CapWords preferring short names: T, AnyStr, Num. It is recommended to add suffixes _co or _contra to the variables used to declare covariant or contravariant behaviour correspondingly:

```
from typing import TypeVar

VT_co = TypeVar('VT_co', covariant=True)

KT_contra = TypeVar('KT_contra',
contravariant = True)
```

CapWords convention

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).

Page No.
Date

global variable name: Modules that are designed for use via from M import * should use the - all - mechanism to prevent exporting globals, or fixing such globals with an underscore (which you might want to do to indicate these globals are "module non-public").

Function and Variable Names:

function name should be lowercase, with words seperated by underscores as necessary to improve readability

Variable names follow only in contexts where that's already the prevailing style (e.g. threading.py to retain backwards compatibility).

Mixed cased is allowed only in context where that already prevails.

Because Exceptions should be classes, class naming convention applies or

- function and Method Arguments:

Always use self for the first argument to instance methods.

Always use cls for the first arguments to class methods.

If a function argument's name clashes with 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.

- 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.

To avoid name clashes with subclasses, use two leading underscores to invoke python's name mangling rules.

Page No.
Date

Constants :

Constants are usally defined on a module level and written all capital letters with under scores separating words. Example include MAX_OVERFLOW and TOTAL.

Designing for inheritance :

Always Decided where a class's methods and instance variable (collectively; "attributes") should be public or non public. if in doubt, chose non public. its easier to make it public later than to make a public attribute a non-public.

Blank lines : Surround top level function and class definition with two blank lines.

Method definations inside a class are surrounded by a single blank line.

Extra blank lines may be used (sparingly) to separate groups of related functions.

- function annotations

with the acceptance of pep 484, the style rules for functions annotations have changed.

- function annotation should use pep 84 syn (There are some formatting recommend for annotations and perious selections).

- The entire community with annotations style that was recommended previously in this pep is no are now engouged. for ex. ample Making up a large third party library or application ha easy it was add those annotatio thing, and observing whether their presence increases code and

- for code that wants to make a differ used of function annotation. it is recommendable to put a comment of the form.

Variable Annotations:

Pep 526 introduce variable annotations. The style recommendations for them are similar to those on function annotations described above.

Annotations for module level variables, class and instance variables, and local variable should have a single space after the colon.

There should be no space before the colon.

If an assignment has a right hand side, then the equality sign should have exactly one space on both sides.

Also beware of writing if x when you really mean if x is not None - e.g. when testing whether a variable or argument that defaults to None was set to some other value. The other value might have a type (such as Container) that could be false in a boolean context.

fyIT-123

Sakshi Kaypte

Page No.
Date

PEP 8:

Introduction: This document gives coding conventions for the python code comprising the standard library in the main python distribution. Please see the companion of python. This document and PEP 257 (Docstring conventions) were adapted from Guido's original python style guide essay with some additions from Barry's style guide. This style guide evolves over time as additional conventions are identified and past conventions are rendered obsolete by changes in the language it self. Many projects have their own coding style guidelines. In the event of any conflicts, such project specific guide take precedence for that project.

Code lay-out Indentation: Use 4 spaces per indentation level. Continuation lines should align wrapped element either vertically using python implicit line joining inside parentheses, brackets and braces, or using a hanging indent. when using a hanging indent the following should be considerated; there should be no arguments on the first line and further indentation should be used to clearly distinguish itself as a continuation line.

Jaksh Gupta

fyIT-123

Page No.
Date

Should a line break before or After a binary Operator?

for decades the recommended Style was to break after binary operators. But this Cost hurt readability in 2 ways: the Operators tends to gets scattered across different columns on the Screen and a Each operator is moved away from its operand and into the previous line. Here the Eye has to do Extra work to tell which item are added and which Substracted.

To Solve this readability problem mathematicians and their publishers follow the opposite Convention Donalmed Knuth explains the traditional rule in his Computer and Types Setting Senders break after binary Op erations and relationship, displayed formula always break binary operations"

following the tradition from mathematics usually ars Suits is more addorables