PEP8 -- Style Guide for Python Code.

Introduction:
The PEP is an abbreviation from of python Enterprise proposal writing whe with proper logic is a key factor of programming, but many other important factors can affect the code's quality. The developer's coding style makes the code much reliable, and every developer should keep in mind that python strictly follows the way of order and format of the string.

PEP is a downent that provides various quinelieur to write. The readable in python. PEP 3 describes how the developer can write beautiful code. It was officially written in 2001 by Gudio Van Rossum Barry Warsaw, and Nick coghlan. The main aim of PEP is to enchance the readability and consistency of code.

* code Lay-out

Indentation

continuation lines should align wrapped elements either vertically using Python's implicit line joining incide pooleonthers, brackets and braces, or during a hanging indent. When using a hanging the tollowing should be considered; there should be



no agrangements on the first line and further indentation should be used to clearly distinguish # Aligned with opening delimiter. foo: long - function- name (var. one, var-two, var-three. var four) # Add 4 spaces (an extra level of indentation) to distinguish arguments from the rest det long-tunction-name (var-one var-two, var-thrée, var-four): print (var. one) # Hanging indents should add a level for = long - punction - name (var one, var-two, var-three, var-tour) The 4-space rule is optional for continuation lines. # Hanging indents * may * be indented to Other than 4 spaces. 100 = long - function - name (var-one, var-two, var-three, var- tour) when the conditional part of an it statement is long enough to require that it be written across multiple lines its worth noting that the combination two character keyword (i.e. if), plus a single

plus on opening parenthesis creates a natural
plus on opening parenthesis creates a natural
4- space indent for the sub-sequent lines of the
multiline conditional. This can produce a visual
multiline conditional. This can produce a visual
conflict with the indented suite of code nested
conflict with the indented suite of code nested
incide the it-statement, which would also
incide the it-statement, which would also
takes no explicit position on how for whether
takes no explicit position on how for whether
to tuther visually distinguish such conditional
lines from the nested suite inside the it
statement. Acceptable options in this situation
include, but are not limited to:

Add a comment, which will provide some distinction in editors

supposting syntax highlighting.

it (this - is - one - thing and

that - is-another - thing):

since both conditions are

true, we can trobnicate. do-something ()

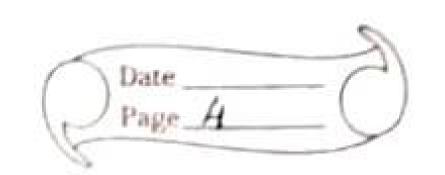
Add some extra indentation on the conditional continuation line it (this - is - one - thing)

and that is - another - thing):

do something ()

(Also see the discussion of whether to break before or after binary operators bleow.)

Vikass FYIT. 99



Tabs or Spaces?

spaces are the preferred indentation method. Tabs
should be used solely to remain consistent with
code that is already indented with tabs

Python disallows mixing tabs and spaces for indentation

* Maximum line length

for to lowing long blocks of text with terrer shruttural restrictions (docstrings or comments), the line length should be limited to 72 characters.

limiting the required editor window with makes it possible to have several files open side by side and works well when using code review tools that present the two versions in adjacent columns.

The default wrapping in most tools 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 to 80, even it the tool places a marker glyph in the final column when wrapping lines.

in two ways; the operations tend to get scattific across different columns on the screen, and each operator is moved away from its operand and on to the pervious lines. Here, the eye has to do extra work to tell which items are added and which are subtracted:

wrong:

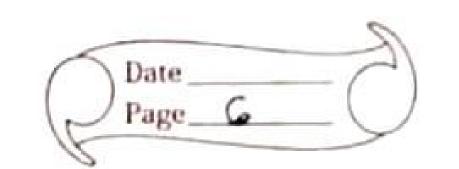
operations sit four away from their oprators incom = (gross-wagy +

taxable-wages+

(dividend. quabfied-dividende)-

ira deduction -

Student - loan - interest).



In python code, it is permissible to break before or after a binary operator, as long as the convention is consistent locally, for new code knuth's style is suggested.

* Blank Lines.

Surround top-level function and class definitions with two blank lines. Method definitions inside a class a surrounded by a single blank line.

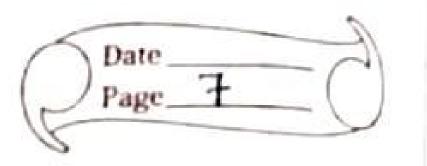
Extra blank lines may be used (spasingly) to sparate groups of related functions. Blank lines may be omitted between a bunch of selated one-liners (e.g. a set of dummy implementations).

Use blank lines in functions, sparingly, to indicates logical sections. Pythom accepts the control from feed character as whitespace; Many tools treat there character as page spratons; so you may use them to separate pages of related sections of your file. Note, some editors and web-based code viewers may not recognize control-Las form feed and will show another glup in its place.

* Source file Encoding.

Code in the crose pyinon distribution should always use UTF-8, and should not have an enwaing declaration.

Vikass FYIT. 99



All identifiers in the python standard and library MUST use ASCII - only identifires, and should use non-ASII characters, spaningly, preferable only to denote places and human names. It is using non-ASCII characters as data, avoid noisy unicode characters like zalgo and byte order marks.

All identifizes in python standard library MUSI USC ASCII - only identifiers, and SHOULD use English words wherever teasible (in many cases abbreviations and technical terms are used which aren't English. Open source projects with a global audience

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

* Imports.

Imports should usually be on separate lines:

correct:

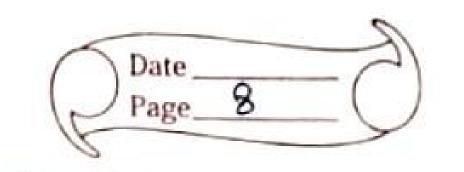
import os

import sys

It's okay to say this though:

correct:

from subprocess import Popen,



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 1. Standard libary imports.

2. Allated third party imports.

3. Local application/Library specific imports.

you should put a blank line between each growing 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 sys-tem is incorrectly configured (such as when a directory inside a package ends up on sys. path):

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

However, explicity relative imports are an acceptable alternative to absolute imports, especially when dealing with complex package layouts where using absolute imports would be uncressarily respose:

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:

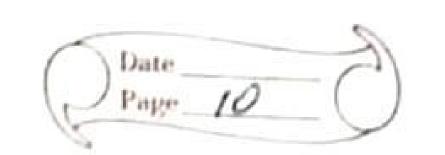
from my class import my class.

It his spelling causes local name clashes, then spell them explicity:

import myclass import too. bar. yourdass and use "myclass. Myclass" and "too. bar. yourdass. Your dass".

wildcard imports (from < module > import *) should be avoided, as they make it unclear which names are present in the namespale, conturing both readers and many automated tools:

There is one defensible use case for a wildcard import, which is to republish an internal interface as part of a public API (for example, overwriting as pure python implementation of an interface with the definitions from an optional accelerator module and exactly which definitions will be over written isn't known in advance).



when republishing names this way, the guidelines below regarding public and internal interfaces

Module Level Dunder Names.

module level "dunders" (i.e. momes with two leading and two traiting underscroes) such as - all -, - author -, version - etc. should be placed after the module document but before any import statements except from - future - imports. Python mandates that future imports must appear in the module before any other code except docstrings:

This is the example module.
This is module does stuff.

from - - - tuhure - - import barry as fluft

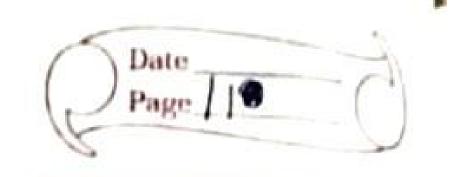
-- all'-- = ['a', b', 'c'] -- version -- = '0.1'

-- author -- = 'cardinal Biggles' import 0s

import sys

string anotes

In python, single-quoted strings and doublequoted strings are the same. This pep does not make a recommendation for this pick a rule



and stick to it. when a string contains single - or double quote characters, however, use the other one to avoid backslashes in the string.

for triple - quoted strings, always use double quote characters to be consistent with the docstring convention in PEP 257.

* Whitespace in Expressions and statements.

Pet Peeves

Avoid extrancous whitespace in the tollowing situation.
Immediately inside parentheses, brackets or bracks:

correct:

spam (ham [1], {eggs: 24)

Between a trailing comma and a following close parenthusis:

correct:

POO = (0),)

Immediately before a comma, semicolon, or colon.

it x = = H: print (x, Y); x, Y = Y, x

| However, in a slice the colon acts like abinary |
|--|
| operator, and should have equal amounts on |
| ethir side (treating it as the operator with the |
| lowest priority). In an extended slice, both |
| clons must have the same |
| amount of spacing applied. Exceptions: when a |
| slice parameter is omitted, the space is |
| omitted. |
| |
| # correct: |
| ham [1:9], ham [1:9:3], ham [:9:3], |
| ham [1:3], ham [1:9:] |
| ham [lover::upper], |
| ham [lover: Step] |
| ham [lower+ offset: upper + offset] |
| ham [: upper - fn(x): step - fn(x)], ham [:! Step - fn(x)] |
| ham [:! Step - fn(x)] |
| ham [lower + OHset : upper + OHset] |
| Immediately betwee the open parentheris that starts |
| Immediately before the open parentheris that starts an indexing or slicing: |
| an maching. |
| # correct: |
| # correct: dct (key] = 1st (index) |
| |
| More than one space around an assignment |
| More than one space around an assignment (or other) operator to align it with another: |
| |
| |
| |
| |
| |

| Vikass FYIT. 99 |
|---|
| # correct: |
| $\begin{array}{c} x = 1 \\ Y = 2 \end{array}$ |
| 1 long. variable = 3 * Other Recommendations |
| Avoid trailing whitespace anywhere. Belause the |
| back stah tollowed by a space and a newline does not count as a line continuation |
| mooker. some editors presever it and many projects (like (python itself) have precommit hooks that reject it. |
| # correct: |
| Submitted $t = 1$ x = x * 2 - 1 |
| hypot2 = $x * * + y * y$ C = $(a+b) * (a-b)$ |
| function annotations should use the normal rule for whoms and always have spaces around the |
| below for more about function annotation.): |
| det munge (input: Anystr): |
| |

Scanned by TapScanner

```
det munge () -> PosInt ....
Don't me spaces around the = sign when und
to indicate a keyword argument, or when used
to indicate a défault value for an un annotée
function parrameter:
# correct :
det complex (real.image = 0.0):
se turn magic (r= real, i= imag)
when combining an argument annotation
with a default value, However do use spaces
a nound the = sign
# correct:
 det munge (sep: Anystr = Nome):
 det nunge (input: Anystr, sep:
 Anystr = None, limit = 1000): ...
 tempound statements (multiple statements on the
 same line) are generally discouraged:
 # correct:
 it . too = = 'blah':
   do-blah-thing()
 dome- ()
  do-+wo ()
```

while sometimes it's okay to put an it / for/what a small body on the same line, never do this for multiclause statements. Also aviode floding such long lines!

* When to use Trailing Commas

Trailing commos are usually optional, except they are mandatory when making a tuple of one element. For clarity, it is recommended to surround the latter in (technically redundant parentheres:

correct: fIIES: ('setup. cfg'.)

worng

FILES = 'Setup.cfq'.

when trailing comman are redundant, they are often helpful when a version control system is used, when a list of value, a regular or imported items is expected to be extended over time. The pattern is to put each value (etc. on a line by itself, always adding a trailing commo, and add the close parenthesis /bracket brace on the next line. However it does not make sense to have a trailing comman the same

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!

Ensures that your comments are clear and easily understandable to other speakers of the language you are writting in.

Pythem codens 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.

* 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 sparated by a line containing a single #.

| | * 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 took distracting it they state the obvious. Don't do this: |
| | X = X + 1 # In (rement × But sometimes, this is useful: |
| | (ompensate for border |
| * | Documentation Strings |
| | lon ventions for writing good documentation Strings (a. k. a. "docstrings") are immortalized in PEP 257. |
| | Write docstrings for all public modules fur Home classes and methods. Do estrings are not necessary for non-public method |
| | |

| | C.ake |
|---|--|
| | " "Return a foobag |
| | optional plotz says to probricate the bizbaz first. |
| | for one liner docstrings, please keep the closing "" on the same line: |
| | " "Return an ex-parmot."" |
| * | Naming Conventions |
| | The naming conventions of pythion's library are a bit of amex, so well never get this completely consistent nevertheless, here are the currently recommended naming standards. New modules and palkages (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 visible to the user as public parts of the API should tollow conventions that reflects usage rather than implementation |
| | *** |
| | |