Page No.
Date: / /

Introduction

This document gives coding conventions for the python code comprising the standard library in the main python distribution. Please see the companion informational PEP describing style guidelines for the C code in the C implementation of Python

This document and PEP 257 (Doc string 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 itself.

Many projects have their own coding style guidelines. In the event of any conflicts, such projects - specifying guide take precedence

for that project

Code lay-out.

Indentation

Use 4 space per indentation level.

Continuation lines should align wrapped elements either vertically using python's implicit line joining inside parentheses, brackets, and braces, or using a hanging indent When using a hanging indent the following should be considered; there should be no arguments on the first line and further indentation should be used to clearly distinguish itself as a continuation line:

Correct :

# Aigned with opening delimeter,
foo = long_function_name(var_one,
var_two, var_three, var_four)

Add 4 spaces (an extra level of indential) to distinguish arguemen 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)

Wrong:

# Arguments on first line forbidden when not using vertical alignment
foo = long_function_name(var_one,
var_two, var_three, var_four)

Page No.
Date.: / /

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 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 require that it be written across multiple lines, it's worth noting that the combination of a two character keyword (i.e if), plus a single space, plus an

Page No.
Date.: / /

Opening parenthesis creates a natural 4-space indent for the subsequent lines of the multiline conditional. This can produce a visual conflict with the indent suite of code nested inside the if-statement, which would also naturally be intended to 4 spaces. This PEP takes no exploit position on how (or whether) to further visually distinguish such conditional lines from the nested suite inside the if-statement. Acceptable options 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 distinguish in editors

# supporting syntax highlighting.
if (this_is_one_thing and
    that_is_another_thing):
```

Page No.
Date: / /

Since both conditions are true, we can fronnicate.
do_something().

Add some extra indentation on the conditional continute line.

if (this is one_thing
    and that_is_another_thing):
    do_something().

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

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

my_list = [
    1, 2, 3
]

Page No.
Date: / /

```
result = some_function_that
    takes_arguements(
        'a', 'b', 'c',
        'd', 'e', 'f',
    )
```

Tabs or spaces ?

Spaces are the prefered indentation method.

Tabs should be used solely to remain consistent with code that is already intended with tabs.

Python disallows mixing tabs and spaces for indentation

Maximum line length

Limit all lines to a maximum of 79 characters.

For flowing long blocks of text

Page No.
Date.: / /

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 if the tool places a marker glyph in the final column. When wrapping lines. Some web based tools may not offer dynamic line wrapping at all.

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

Page No.
Date.: / /

The prefered way of wrapping long lines is by using Python's implied line continuation inside parenthesis, brackets and braces. Long lines can be broken over multiple lines by wrapping expressions in parenthesis. These should be used in preference to using a backslash for line continuation

Backslashes may still be appropriate at times. For example, long multiple with-statements could not use implicit continuation before python 3.10, so backslashes were acceptable for that case.

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

Page No.
Date: / /

(see the previous discussion on multiline if-statements for further on the indentation of such multiline with-statements)

Another such case is with assert statements.

Make sure to indent the continued line appropriately.

Should a line break before or After a Binary operator?

For decades the recommended style was to break binary operators. Here, the eye has to do extra work to tell which items are added and which are subtracted.

Wrong

operators sit far away from their operands

Page No.
Date: / /

income = (gross_wages +
taxable_interest +
(dividends_-qualified_-
dividends) -
ira_deduction -
student_loan_interest)

income = (gross_wages).

Following the tradition from mathematically wrong results in more readable code.

correct :

easy to match operators with operands

income = (gross_wages
+ taxable_interest
+ (dividends_-qualified_dividends)
- ira_deduction
- student_loan_interest)