17-Feb-22

PAGE No.
DATE / /

PEP - 8

* INTRODUCTION :-

PEP is an abbreviation form of python enterprises proposal writing code with proper logic is a key factor of programming, but many other important factors can affect the code's quality PEP-8 is a document that provides various guideline to write readable in python. It describes how developer can write beautiful codes.
The main aim of PEP is to enhance the readability and consistency of codes.

* CODE LAYOUT :-

INDENTATION :

Uses 4 spaces per indentation level. continuation lines should align wrapped element either vertically using python's implicit line joining inside parenthesis, brackets and braces, or using a hanging indent.
When using a hanging indent following should be considered ; there should be no arguments on first line and further indentation should be used to clearly distinguish itself as a continuation line.
Example :

```
foo = function_name (var1, var2, var3, var4)
```

When the conditional part of an if-statement is long enough to require that it be written a cross

multiple lines, its worth nothing that the combination of two character Keyboard, plus a single space, plus an opening parenthesis create a natural 4-space indent for subsequent lines of multiline conditional.

The closing brace / bracket / parenthesis on multiline constructs may either line up under the first non-Whitespace character of last line of list or it may be lined up under the first character of line that starts multiline constructe.

⟹ TABS OR SPACES :

Spaces are 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 :

Limits all lines to a maximum of 79 characters. limiting the required editor window width makes it possible to have several files open by side, and works well when using code review tools that present the two version in adjacent columns. The limits are chosen to avoid wrapping in editors with the window width set to 80, even if tool places a markers in final column when wrapping lines.

The preferred 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. Backlashes may still be appropriate at times.

⟹ LINE BREAK BEFORE OR AFTER BINARY OPERATOR :

The recommended style was to break after binary operators. But this can hurt readibility in two ways : the operators tend to get scattered across different across different columns on screen, and each operator is moved away from its operand and onto previous line.
In python code, it is permissible to break before or after a binary operator, as long as convention is consistents locally.

⟹ BLANK LINES :

~~Sow~~
Surround top-level function and class definitions with two blank lines.
Method definitions inside a class are surrounded by a single blank line. Extra blank lines may be used to separate groups of related function Blank lines many be omitted between a bunch of related one-liners.

PAGE No.
DATE / /

⇒ IMPORTS :

Imports should usually be on seprate lines They are always put at top of file, just after any module comment and docstrings and before module globals and constants.
Imports should be grouped in following order:
1) Standard library imports
2) Related third party imports
3) local application / library specific imports Absolute Absolute import are recommended, as they are usually more reabie and trend to be better behaved if the import system is incorrectly configued
Standard library code should avoid complex package layouts and always we absolute imports.

⇒ MODULE LEVEL DUNDER NAMES :

Module level "dunder" (names with two leading and two trailing underscores) such as --- all --, -- author --, -- version --, etc should be placed after module docstring but before any import statements excepts from -- future -- imports. must appear in module before any other code except docstrings.

✳ STRING QUOTES :

In python, single-quoted strings and double-quoted

Strings are same. This PEP does not make a recommendation for this.
When a string contains single or double quote characters, use the other one to avoid backslashes in the string. It improves readability

* WHITESPACES IN EXPRESSIONS & STATEMENTS :-

⇒ PET PEEVES
Avoid extraneous whitespace as follow.
- Immediately inside parentheses, brackets or braces
- Between a trailing comma and a following close parenthesis
- Immediately before a comma, semicolon, or colon:
- However, in a slice the colon acts like a binary operator, and should have equal amount on either side.
- Immediately the open parenthesis that starts the argument list of a function call:
- Immediately before the open parenthesis that starts an indexing or slicing.
- More than one space around as assignment (or other) operator to align it with another:

→ ~~OTHER~~ OTHER RECOMMENDATIONS
- Avoid trailing whitespace anywhere. Because it's usually invisible, it can be confusing some editors don't preserve it and many project have pre-commit hooks that reject it.

- Always surround these binary operators with a single space on either side : assignment (=), augmented assignment (+=, -= etc).
- If operator with different priorities are used, consider adding whitespace around the operators with the low priority.
- Function annotations should use the normal rules for colons and always have spaces around the → arrow if present.
- Don't use spaces around the = sign when used to indicate a keyword argument, or when used to indicate a default value.

* WHEN TO USE TRAILING COMMAS

Trailing commas are usually optional, except they are mandatory when making a tuple of one element ~~a tuple of one~~

When trailing commas are redundant they are often helpful when a version control system is used when a list of value, arguments or imported items is expected to be extended over time.

* COMMENTS

⇒ BLOCK COMMENTS

Block comment generally apply to some code that follows them, and are indented to the same level as that code.

PAGE No.
DATE / /

→ INLINE COMMENTS.

An inline comments is a comment on the same line as a statement. Inline comments should be separated by at least Start with the statement.

⇒ DOCUMENTATION STRINGS.

Docstrings are not necessary for non-public method but you should have a comment the describes what the method does.

—* NAMING CONVENTIONS

⇒ OVERRIDING PRINCIPLE

Name that are visible to the user as public parts of the API should follow conventions that reflect usage rather than implementation.

⇒ NAMING STYLES

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

⇒ NAMING CONVENTIONS.

1) Name to Avoid

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

2) ASCII compatibility
Identifier used in the standard library must be ASCII compatible as described in the PEP3131

3) Package and Module Name.
Modules should have short, all-lowercase name Underscores can be used in the module name if it improves readability.

4) Class Name
class name should normally use the capWords convention.
The naming convention for function may be used instead in cases where the ~~unf~~ interface is documented and used primarily as a callable.

5) Type Variable Names
Names of type variable introduced in PEP 484 should normally use capwords preferring short names : T, AnyStr. Num.

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

7) Global variable Name
(Let's hope that these variable are meant

for use inside one module only). The conventions are about the same as those for Functions.

8.) Function and variable Name:
Function names should be lowercase, with words separated by underscores as necessary to improve readability.
Variable name follow the same convention as Function name.

9.) Function and Method Arguments
Always use self for the First argument to instance methods.
Always use cls for the First argument to class methods.

10.) Method Name 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.

11.) Constants
Constants are usually defined on a module level and written in all capital letters with underscores separating words.

12.) Designing for Inheritance

Always decide whether a class's method and instance variable should be public or non-public. If in doubt, choose non-public, it's easier to make it public later than to make a public attribute nonpublic.

Public attributes are those that you expect unrelated clients of your class to use, with you commitment to avoid backwards incompatible changes. Non-public attributes are those that are not intended to be used by third parties, you make no guarantee that non-public attributes won't change or ever be removed

⟹ PUBLIC AND INTERNAL INTERFACES

Any backwards compatibility guarantees apply only to public interfaces. Accordingly, it is important that users be able to clearly distinguish between public and internal interfaces.

Documented interfaces are considered public, unless the documentation explicitly declares them to be provisional or internal interfaces exempt from the usual backwards compatibility guarantee.

All undocumented interface should be assumed to be internal.

* PROGRAMMING RECOMMENDATIONS

code should be written in a way that does not distvantage other implementations of python

Comparions to singletons like None should always be done with is or is not, never the equality operators.
Use is not operator rather than not ... is while both expression are functionally identical the former is more readable and preferred.
Always use def statement instead of an assignment statement that binds a lamba expression directly to an identifier.

→ Function Annotations
Function Annotations should be use PEP 484
The experimentation with annotation style that was recommended previously in the PEP is no longer encouraged.

⇒ Variable Annotations
PEP 526 introduced variable annotation. The style recommendations for the them are similar to those on function annotations described.