Day_flow's Python Language Coding Standards

Last update: 17 October 2023

1. Maximum Line Length: Limit lines to 79 characters for code and 72 for docstrings/comments. You can go up to 88 characters if necessary, but try to keep it readable.
2. Imports: Imports should be organized and grouped as follows:
    - Standard library imports.
    - Related third-party imports.
    - Local application/library specific imports.
3. Whitespace in Expressions and Statements: Use whitespace around operators and after commas, but avoid excessive whitespace.
4. Comments and Documentation: Use clear and concise comments and docstrings to explain your code. Follow PEP 257 for docstring conventions.
5. Naming Conventions: Follow PEP 8's naming conventions for variables, functions, and classes. Variables and functions: lowercase_with_underscores. Constants: UPPERCASE_WITH_UNDERSCORES. Classes: CapWords (CamelCase).
6. Avoid Using Single Letters for Variables: Use descriptive variable names.
7. Function and Method Length: Keep functions and methods small and focused. If a function becomes too long, consider refactoring it into smaller functions.
8. Avoid Hardcoding Values: Use constants or configuration files instead of hardcoding values.
9. Function and Method Length: Keep functions and methods small and focused. If a function becomes too long, consider refactoring it into smaller functions.
10. Avoid Hardcoding Values: Use constants or configuration files instead of hardcoding values.
11. Exception Handling: Use try-except blocks to handle exceptions gracefully.
12. Use List Comprehensions: When appropriate, use list comprehensions to simplify code.
13. Use if __name__ == '__main__':: To allow importing your module without executing code, use this construct to encapsulate the code that should run when the script is executed.

Naming Conventions

The following are the currently recommended naming standards.

Avoid These Names

Never use the characters ‘l’ (lowercase letter el), ‘O’ (uppercase letter oh), or ‘I’ (uppercase letter eye) as single character variable names.

Package and Module Names

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

Name of Class

Class names should normally use the CapWords convention. The naming convention for functions may be used instead in cases where the interface is documented and used primarily as a callable.

Exception Names

The class naming convention applies here. However, you should use the suffix “Error” on your exception names.

Function and Variable Names

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

Variable names follow the same convention as function names.

Function and Method Arguments

Always use self for the first argument to instance methods.

Always use CLS for the first argument to class methods.

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.

Constants

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

Indentation

The guideline suggests using 4 spaces per indentation level.

Tabs or Spaces

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

Python disallows mixing tabs and spaces for indentation.

Commenting convention

Comments should be complete sentences. The first word should be capitalized, unless it is an identifier that begins with a lowercase 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).

Paragraphs inside a block comment are separated 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.

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.
PEP 257 describes good docstring conventions. Note that most importantly, the """ that ends a multi line docstring should be on a line by itself:

```
"""Return a foobang
Optional plotz says to frobnicate the bizbaz first.
"""
```

For one liner docstrings, please keep the closing """ on the same line:
`"""Return an ex-parrot."""`

Functions comments

For all functions outside the Class please add a short header comment describing the purpose of the function and the name of the function.
Example:

```
# Bubble sort
# A sorting function to sort the target array using
def bubble_sort(unsorted_array, length):
      need_swap = True
```

```
      while need_swap:
            need_swap = False
            for i in range(length - 1):
                  if unsorted_array[i] > unsorted_array[i + 1]:
                        need_swap = True
                         unsorted_array[i], unsorted_array[i + 1] = unsorted_array[i +
1],unsorted_array[i]
      return unsorted_array
```

Maximum Line Length

Limit all lines to a maximum of 79 characters.

Imports

The import statement, just like any other statement or keyword in Python should be used and added to the code properly following the best practices. Let's see them one by on –

Multiple Imports

Multiple Imports should usually be on separate lines. For example –

```
import numpy
import pandas
import matplotlib
```

Always on the Top

Imports are always put at the top of the file i.e.

- After any module comments and docstrings.
- Before module globals and constants.

For example –

```
# import the numpy module
```

```
import numpy
```

Import Modules in an Order

A good practice is toimport modules in the following order –

Standard library modules – e.g. sys, os, getopt, re.

Third-party library modules – e.g. ZODB, PIL.Image, etc.

Locally developed modules.

Absolute Imports

Absolute imports are recommended, as they are usually more readable and tend to be better performed if the import system is incorrectly configured. For example –

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

Wildcard imports (from import *) should be avoided

Avoid the Wildcard imports since they make it unclear which names are present in the namespace, confusing both readers and many automated tools.

Whitespace in Expressions and Statements

Avoid unnecessary whitespace as in the following situations –

Between a trailing comma

```
# Correct:
a = (0,)
```

```
# Wrong:
b = (0, )
```

Immediately before a comma, semicolon, or colon –

```
# Correct:
if a == 5: print(a, b); a, b = b, a
```

```
# Wrong:
if a == 5 : print(a , b) ; a , b = b , a
```

Immediately before the open parenthesis that starts the argument list of a function call

```
# Correct:
demo()
```

```
# Wrong:
demo ()
```

Immediately before the open parenthesis that starts an indexing or slicing

```
# Correct:
dct['key'] = lst[index]
```

```
# Wrong:
dct ['key'] = lst [index]
```