Pyclub style guidelines

Pyclub

27 Feb 2018

1 Guidelines

1.1 The rules of pyclub

- Code and comments in English.
- Pythonic, follow PEP8 if possible.
- Python 3 whenever applicable. Assume at least 2.7.
- Python 3 and 2 compatible if applicable (favour libraries that have bindings for both if of equal quality and syntax that works for both).
- Clean code (i.e. short functions, few useful comments... (see the eponymous book https://www.safaribooksonline.com/library/view/clean-code/9780136083238/).
- Try to be crossplatform.
- Sufficiently documented using Sphinx.
- Sufficiently tested.
- Good exception coverage.
- Explicit imports (not starred).
- Follow the agreed upon project organisation.
- Should prefer standard python libraries unless there is a good reason to use something else.
- Should use libraries that have been agreed upon by the group.
- Use code from the pyclub repository whenever applicable.
- Try not to reinvent the wheel.
- Make methods/attributes private unless the user needs to access them.
- If any TODO or similar items are left in the code (ideally not in the repo) add a ":" after to allow most editors to pick up the pattern.
- For complete programs: have a setup.py script and include test data.

1.2 Pyclub PEP8 extract

1.2.1 Pythonic style hints

- Limit returns from different places.
- Use underscore for ignored variable (ideally 2).
- Use one statement per line.
- Use *with* statements when applicable.
- Use list comprehension for simple loops.
- Use '-'*n or [3]*n for repeated items.

1.2.2 Python 2 and 3 compatibility

- Use explicit calls to *object* in class declarations.
- Use *print* as function not as a statement.
- Use explicit calls to *super*.

1.2.3 Exception style rules

- Use a specific exception hierarchy.
- Use explicit messages with data type and content if expected to be small.
- Decide the level of appropriate coverage (as per Kirsty's comments in the last PyClub).

1.2.4 Test coverage

- Preferably using pytest because other libraries either deprecated or not pythonic (if people agree).
- Minimum coverage?
- Provide test datasets if applicable.

1.2.5 PEP8 reminder

- 4 spaces.
- Limit all lines to a maximum of 79 characters.

- 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 (sparingly) to separate groups of related functions. Blank lines may be omitted between a bunch of related one-liners (e.g. a set of dummy implementations).

  Use blank lines in functions, sparingly, to indicate logical sections.

- Code in the core Python distribution should always use UTF-8 (or ASCII in Python 2).

  Files using ASCII (in Python 2) or UTF-8 (in Python 3) should not have an encoding declaration.

- Imports are always put at the top of the file, just after any module comments and docstrings, and before module globals and constants. Imports on different lines Imports should be grouped in the following order:

  standard library imports related third party imports local application/library specific imports

  You should put a blank line between each group of imports.

- Whitespace: Yes:

```
i = i + 1
submitted += 1
x = x*2 - 1
hypot2 = x*x + y*y
c = (a+b) * (a-b)
```

  No:

```
i=i+1
submitted +=1
x = x * 2 - 1
hypot2 = x * x + y * y
c = (a + b) * (a - b)
```

  Don't use spaces around the = sign when used to indicate a keyword argument or a default parameter value.

- Documentation Strings

  Conventions for writing good documentation strings (a.k.a. "docstrings") are immortalized in PEP 257 .

  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 multiline docstring should be on a line by itself, e.g.:

```
"""Return a foobang

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

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

- names:
  - module_name (short, all-lowercase names).
  - ClassName (PascalCase)
  - function_names, method_name, variable_name (snake_case): lowercase with words separated by underscores as necessary to improve readability.
  - MODULE_CONSTANTS or CLASS_VARIABLES are usually written in all capital letters with underscores separating words.
- Derive exceptions from Exception rather than BaseException .
- Use exception chaining appropriately. In Python 3, "raise X from Y" should be used to indicate explicit replacement without losing the original traceback.
- Object type comparisons should always use `isinstance()` instead of comparing types directly.
- For sequences, (strings, lists, tuples), use the fact that empty sequences are false.
- Don't compare boolean values to *True* or *False* using "==".

2 References

- UCL engineering introduction to python
- Numpy for matlab users
- MIT introductory course to python
- Official python turorial
- Official python turorial more advanced
- Clean code and Clean coder by Robert Martin
- Best Practices for Scientific Computing
- Refactoring by Martin Fowler