“Stylish” Code (Python)

WWCode-Silicon Valley

Michelle Ho

Resources

PEP8 - Style Guide:

https://www.python.org/dev/peps/pep-0008/

https://docs.python-guide.org/writing/style/

https://realpython.com/python-code-quality/

https://realpython.com/python-pep8/

Why

- Do what it is supposed to do
- Minimize code bugs/defects/problems
- Maintainability
  - Read more than Write code
    - Write once
    - Read many more times in future
    - Want to remember what/why we wrote….
    - If not remember, at least be able to figure out after reading

How?

- Official style guide - Python Enhancement Proposal (PEP)
  - PEP8 - https://pep8.org/
    - Standard conventions
    - Focuses on code style
  - PEP27 - https://www.python.org/dev/peps/pep-0257/
    - Python Docstrings conventions
- Linters - Code Analysis Tools
  - Logical Lint
    - Finds code errors, potentially unintended errors
    - Bad code patterns
  - Stylistic Lint
    - Code not conforming to style conventions

Linters

Many options (Standalone/Combination Linters)

- Pycodestyle
- Pylint
- Flake8 (combo of Pyflakes, pycodestyle, McCabe)
- Pylama (combo of Pyflakes, pycodestyle, pydocstyle, McCabe, etc)
- Pyflakes
- Bandit
- https://realpython.com/python-code-quality/

Most recommended?

- Flake8
  - Detects logical issues
  - Detects stylistic issues
  - Fewer False positives
- Pylint
  - Detects logical issues
  - Detects stylistic issues
  - Checks for errors
  - Tries to enforce coding standard

When to Lint

- As you write
  - IDE Code editors - Lint plugins
  - Highlights issues as you write
- As you commit/publish code
  - Git Hooks
  - Run linters during pre-commit stage.
    - Will gate commits if pre-commit hook does not return with exit code 0.
- As you run tests/CI pipelines
  - Run linters as do builds
  - Good if already have existing code base (with errors)
    - Monitors/fails if new checkins increase the number of lint errors
    - Don’t have to rewrite whole code base first

PEP8 - Style Conventions

Naming Conventions

- Meaningful names
  - "Explicit is better than Implicit"
  - Don't use generic names (x, y, z)
  - 3 letter character variables or less will be flagged

- Variables, functions, methods, modules, and packages
  - All lowercase word, or words
  - No camel casing ---> used in other languages like C
  - If more than 1 word, separate with "_"
  - Ex: "get_user_name", "calculated_distance", "user_id"

- Class Names
  - Start with Uppercase
  - CamelCased
  - Word, or words. If multiple words, do not separate with "_"
  - Ex: "UserProfile", "MyClass"

- Constants
  - All uppercase letters
  - Word/Words. Words separated by "_"
  - Ex: "LOGIN_ID", "MY_CONSTANT"

Code Layout - Blank Lines

- Top Level Functions/Classes
  - Separate with 2 lines.
  - Clear line breaks show how these are standalone entities with specific functionality
- Method definitions INSIDE a class
  - Separate with 1 line.
- Within a function
  - Use blank lines sparingly
  - Use blank lines to separate out clear steps inside function for better readability
  - Blank line right before the return statement
    - Clarity on what is being returned

Code Layout - Max Line Length

- Extremely long lines/code wraparound
  - Hard to read
  - Often code with multiple windows of code
- PEP8 - 79 characters MAX
- If extremely long line:
  - Python assumes line continuation for code wrapped between parentheses, brackets, or braces

    ```
    def function(arg_one, arg_two,
                 arg_three, arg_four):
        return arg_one
    ```

  - Use backslashes to break line

    ```
    from mypkg import example1, \
        example2, example3
    ```

Code Layout - Max Line Length

- If extremely long line with binary operators (+, -, *, etc)
  - Break line so the binary operator is with the second operand
    - Makes it clear what the being operated on

```
total = (first_variable
         + second_variable
         - third_variable)
```

Code Layout - Indentation

- Be consistent
- Spaces vs. Tabs
    - Do not do a combination of the two.
    - Python3 will mark as an error if use space and tabs for indentations
- Python prefers SPACES over tabs
- Use 4 spaces for indentation
- Configure your text editor to take care of this
    - Format tabs as 4 spaces
    - Ex: in vi editor,
        - set tabstop=4
        - set expandtab

Code Layout - Indentation after Line Breaks

- Long lines
  - If break long lines into multiple lines (max 79 character limitation)
    - Indent following lines for better readability
    - Align indented block with opening delimiters
- Conditional statements vs execution code
  - Add extra indentation to distinguish between the two.
  - `x = 5`
  - `if (x > 3 and`
  - `        x < 10):`
  - `    print(x)`

Code Layout - Hanging Indents

- Every line in a code block, except the first one, is indented.
    - Hanging indents shows line continuation
    - Must not have any arguments on first line
    - var = function(
    -     arg_one, arg_two,
    -     arg_three, arg_four)
- To separate function arguments and function body, double indent line continuation.

```
def function(
        arg_one, arg_two,
        arg_three, arg_four):

    return arg_one
```

Comments

- Document code so can understand by people other than you
- Limit comment line lengths to 72 characters
- Update as necessary
- Use complete sentences.
- Don’t state the obvious. Have meaningful comments

Block Comments

- Document small sections of code
- Help understand purpose and functionality of code block
- Start block comment to same indentation level as code it describes
- Start each line of comment with single “#” and a space
- If more than one paragraph of comments
    - Separate paragraphs with a single line, with just “#”

Inline Comments

- Explain single statement of a piece of code.
- Explain/remind why one specific line of code is necessary
- Use sparingly
- Write in same line that it applies to
- Don’t explain the obvious “This is a variable”
- Format:
    a. Two spaces after end of the code line
    b. followed by “#” and a space
    c. Followed by single line comment

Document Strings

- Documentation Strings
- Surrounded by triple double quotes, or triple single quotes
- First line of any function, class, method, or module
- If multi line docstring
    - Ending triple quotes on its own separate line
- If single line docstring
    - Ending triple quotes on the same line

YES - Whitespacing

- Improves readability
- Be consistent
- Single space on both sides of
  - Binary operators
  - Comparisons
  - Booleans
  - EXCEPT: when assigning default value to a function argument…. No spacing then
- More than one operator in a statement or in an "if" statement
  - Space only around operator of lowest priority
  - `y = x**2 + 5`
  - `z = (x+y) * (x-y)`

NO - Whitespacing

- Trailing whitespacing
  - End of a line
  - End of code file
  - Hard to detect, invisible, can be prone to errors
- Immediately inside of parentheses, braces, brackets
- Before comma, semicolon, or colon
- Before opening parenthesis of a function call
- Before open bracket of list indexing/slicing
- Before trailing comma and closing parenthesis
- Align assignment operators

Logical Code Recommendations

- Don’t compare True/False with equivalency operator
    - My_bool = 6 > 5
    - If my_bool == True:
- Use length check for emptiness
    - No - “if not len(my_list):”
    - No - “if len(my_list) == 0:”
    - Yes - “if not myList:”
- Use “is not”:
    - “If x is not None:”
- Don’t use “if x” if you meant “if x is not None”
- User ‘.startswith’ or ‘.endswith’ instead of slicing (variable[0], variable[-1])

And way more….

- USE LINTERS
    - Pycodestyle
    - Pylint
    - Flake8
- Code review
- Keep Coding