Code Review & Coding Style

CPE101 Winter 2019
@ Cal Poly SLO
By
Toshi

Learning Objectives

1. Learn to read programs.
2. Learn to distinguish between bad code and good code.
3. Learn to write good clean code.
4. Learn a standard coding style that a programmer community uses.

Code Review

- It is a good practice to have somebody else review your code just like you would ask somebody else to proof read your writings.
- Mentorship
- Peer Review
- Code review can help you to:
    - Find discrepancies between requirements and actual implementations
    - Find bugs
    - Improve readability of your code
    - Optimize your code

Comments

- Write comments in your program to help others as well as yourself to understand your program.
- Comments in Python
    - #
        - Everything after # till the end of the line is ignored by Python interpreter.
        - Do not write comments that wraps around into multiple lines on your screen for the sake of readability.
    - ‘’’ (Three consecutive single/double quotation marks)
        - Characters after three consecutive quotation marks are considered comments and ignored by Python interpreter until it encounters another three consecutive quotation marks, which marks the end of the comment.
        - This kind of comments are called docstrings.

Coding Style

- Establishing and following a coding style guide helps your program written in a consistent style and in turn readable.
- Program written in a consistent style is generally easier to read.
- Why readability matters?
  - In real world, if your program has real users, it goes through series of maintenances and changes throughout its life until it is no longer used.
  - Many real world programs are written and maintained by multiple people.
  - An easy-to-read program is easier to understand and maintain than a difficult-to-read program.

Python coding style

Python has its recommended coding style guide https://www.python.org/dev/peps/pep-0008/

Google also has published its python style guide http://google.github.io/styleguide/pyguide.html

- Indentation
  - Use 4 spaces per indentation level. (no tabs)
- Maximum line length
  - Limit all lines to a maximum of 79 characters.
  - use indentations to improve readability if code does not fit in one line.
- Blank lines
  - Surround top-level function and class definitions with two blank lines.
- Imports
  - One module per line
    - Bad: import os, sys

Examples

```
# When your code does not fit in one line, use new lines and indentation

 # Aligned with opening delimiter.
 foo = long_function_name(var_one, var_two,
                          var_three, var_four)

 # More indentation included to distinguish this from the rest.
 def long_function_name(
         var_one, var_two, var_three,
         var_four):
     print(var_one)
```

Examples

```
# align the start of each line with ( and start each line with an operator.

income = (gross_wages

          + taxable_interest

          + (dividends - qualified_dividends)

          - ira_deduction

          - student_loan_interest)
```

Names

Use descriptive (self explanatory) names for your variables, classes, and functions.

- Variable names
  - Use lower case with _
  - E.g. state, address, zip, gender, first_name, last_name, birth_date
- Class names
  - Use CapWords (CamelCase) style
    - E.g. Point, Person, GradePointAverage, SortUtility, RuntimeError
- Function names
  - Use lower case with _
- Constant names
  - Use all capital letters
  - E.g. TIME_ZONE, NUM_WORDS_LIMIT

Comments

- Function header
  - Write docstring just below the function definition describing what function does and listing and describing arguments and return values if any.
  - Docstring style: https://sphinxcontrib-napoleon.readthedocs.io/en/latest/example_google.html
- Block comments
  - A block comment describes a block of code that follows it.
- In-line comments
  - You do not need to add a comment to every line. If you use very descriptive variable names and function names, your code is already self-explanatory.
  - But, if you find a particular code in you program tricky to understand, do not hesitate to add some comments, explaining what the code is trying to do.

File Header Docstring Example

```
# -*- coding: utf-8 -*-
"""Example Google style docstrings.

This module demonstrates documentation as specified by the `Google Python
Style Guide`_. Docstrings may extend over multiple lines. Sections are created
with a section header and a colon followed by a block of indented text.

Example:
    Examples can be given using either the ``Example`` or ``Examples``
    sections.
Attributes:
    module_level_variable1 (int): Module level variables may be documented in
        either the ``Attributes`` section of the module docstring, or in an
        inline docstring immediately following the variable.


"""
```

Function Header Docstring Example

```python
def function_with_types_in_docstring(param1, param2):
    """Example function with types documented in the docstring.

    `PEP 484`_ type annotations are supported. If attribute, parameter, and
    return types are annotated according to `PEP 484`_, they do not need to be
    included in the docstring:

    Args:
        param1 (int): The first parameter.
        param2 (str): The second parameter.

    Returns:
        bool: The return value. True for success, False otherwise.

    .. _PEP 484:
        https://www.python.org/dev/peps/pep-0484/

    """
```

Block and inline Comments Examples

```
# calculate the average score of an exam

exam_scores = [56, 78, 98, 65, 73]

num_scores = len(exam_scores)

num_scores = float(num_scores) # converting to float to get fractional part

avg = sum(exam_scores) / num_scores
```

Number of Lines in Function Body

- Watch out for functions with long function bodies.
- Each function or method should do one thing and only one thing.
- If your function start growing larger than 20 lines, you should take a closer look at that function or method.
- In my personal experiences, functions with if statements branching based on the value of an argument passed as a switch are often trying to do more than one thing.