Code Style Guide

Attached are some of the best practices, tips, and tricks for writing great Python code that's reusable and maintainable. I've perused some of the best sources on the Internet for Python documentation guidelines and hope that this guide adds value and makes your Python code that much more clean. Please let me know how I can enhance and improve this guide and whether it's valuable. Please contact Yingquan at yli12313@seas.upenn.edu if there's any questions, concerns or just general inquiry. - Best, Y (Thursday, 9/9/21)

Version 1.0	Date: 8/1/21
Version 1.1	Date: 9/9/21
Version 1.2	Date: 12/6/21

TL;DR

If you read nothing else, please read this section.

The following material includes some tips and best practices for documenting Python code. Please read all the material and make your own judgment on what is useful for you and what is not useful. Each person has his or her own coding style and standards and we want people to have the freedom to do things their own way! At a minimum however, all scripts written for the lab should have the two things listed below:

1. A docstring at the top of the script with the following information:
    a. Author
    b. Date
    c. A short sentence describing what the script's overall purpose is.

2. For each function that's written, include a docstring that describes what the function is doing and at a high level and also document the important pieces of logic inside a function, should it become very confusing. Once a piece of code is reusable, the team will want to document it well and use it for the future.

Please see the coding sample as a example of what a well-documented script looks like: **documented_script_example_20210909.py**

Guiding Philosophy to Writing Good Python Code

- Please keep some of these following first principles in mind as you are developing Python scripts, functions, projects, etc. Python code should always be written with the mindset that: "Simple is better than complex" and "Sparse is better than dense."

```
>>> import this
```

```
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
>>>
```

With the above poem in mind, attached below are some of the things to be mindful of when writing Python scripts. Only write good documentation when you have a nice segment of code that you feel is reusable for the future. If the code is for Exploratory Data Analysis (EDA), there does not need to be excellent documentation as the code by nature is exploratory and will not be reused again.

1) Variable Names and Function 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 names: `long_function_name(), some_function_with_arguments(), etc`.
Variable names: `test_data, train_data, final_modeling_results, etc`.

2) File Naming Conventions

- Files/Modules should be names with descriptive words separated by '_' characters.

File names: my_module.py, game_function_code.py, auxiliary_code.py, etc.

3) Spaces vs. Tabs

- Spaces are the preferred method of indentation; *4 spaces* are used for each indentation level.

```
# 4 spaces per each indentation Level
def long_function_name(
|1|2|3|4|5|6|7|8|var_one, var_two, var_three,
|1|2|3|4|5|6|7|8|var_four):
|1|2|3|4|print(var_one)
```

4) Function Docstrings

- A *docstring* is the first statement in a package, module (a.k.a file), class or function that adds a description of the object's functionality in question.
- Functions must have docstrings unless they are: not externally visible, very short or obvious.

Function with Docstring (Example)

```
def get_spreadsheet_cols(file_loc, print_cols=False):
    """Gets and prints the spreadsheet's header columns

    Parameters
    ----------
    file_loc : str
        The file location of the spreadsheet
    print_cols : bool, optional
        A flag used to print the columns to the console (default is
        False)

    Returns
    -------
    list
        a list of strings used that are the header columns
    """

    file_data = pd.read_excel(file_loc)
    col_headers = list(file_data.columns.values)

    if print_cols:
        print("\n".join(col_headers))

    return col_headers
```

Docstring in the Google Format (Example)

```
"""Gets and prints the spreadsheet's header columns

Args:
    file_loc (str): The file location of the spreadsheet
    print_cols (bool): A flag used to print the columns to the console
        (default is False)

Returns:
    list: a list of strings representing the header columns
"""
```

Docstring in the Numpy/Scipy Format (Example)

```
"""Gets and prints the spreadsheet's header columns

Parameters
----------
file_loc : str
    The file location of the spreadsheet
print_cols : bool, optional
    A flag used to print the columns to the console (default is False)

Returns
-------
list
    a list of strings representing the header columns
"""
```

5) File Docstrings

- Files should start with a docstring describing the contents and usage of the module.
- The most important elements of a file docstring: author, date and description.

File Docstring in the Google Format (Template)

```
"""A one line summary of the module or program, terminated by a
period.

Leave one blank line.  The rest of this docstring should contain an
overall description of the module or program.  Optionally, it may also
contain a brief description of exported classes and functions and/or
usage examples.

  Typical usage example:
```

```
  foo = ClassFoo()
  bar = foo.FunctionBar()
"""
```

File Docstring (Example)

```
"""
File name: statistics_operations.py
Author: Frank Ocean
Date created: 4/20/2013

Summary: Provides NumberList and FrequencyDistribution classes for
statistics.

Description: NumberList holds a sequence of numbers, and defines
several statistical operations (mean, stdev, etc.)
FrequencyDistribution holds a mapping from items (not necessarily
numbers) to counts, and defines operations such as Shannon entropy and
frequency normalization.
"""
```

6) Organizing a Python Project

- Python projects should be organized with a folder structure in place. Attached below is an example folder structure.

```
project_root/
|
├── project/  # Project source code
├── docs/     # Important documentation
├── tests/    # Personal tests for the project
├── README    # Tells the public what the project is about
├── HOW_TO_CONTRIBUTE    # Tells others how to contribute (if needed)
├── CODE_OF_CONDUCT      # Project code of conduct (if needed)
├── examples.py          # An example script (if needed)
```

7) Exception Handling

- Use exception handling to catch errors and debug code. Exception handling is a good way to test code as well as maintain code that's reusable for the future.

```
try:
   You do your operations here;
   ......................
except:
```

```
   If there is any exception, then execute this block.
   ......................
else:
   If there is no exception then execute this block.
```

8) Length of a Line of Code

- The Python Style Guide PEP 8 recommends that the length of a line of code ***does not exceed*** 79 characters.

Sources of Python Style Guides

1. PEP 8: https://www.python.org/dev/peps/pep-0008/#type-variable-names
2. Google Python Style Guide: https://google.github.io/styleguide/pyguide.html
3. The Hitchhiker's Guide to Python: https://docs.python-guide.org/writing/style/
4. Real Python: https://realpython.com/documenting-python-code/

Sources of Information on Automated Documentation Generation

1. https://medium.com/blueriders/python-autogenerated-documentation-3-tools-that-will-help-document-your-project-c6d7623814ef
2. https://saddlebackcss.github.io/tutorials/python/basic/2016/01/14/python-basics-4
3. https://pdoc3.github.io/pdoc/
4. https://www.sphinx-doc.org/en/master/
5. https://medium.com/@peterkong/comparison-of-python-documentation-generators-660203ca3804