Clean and Modular Code

Production Code

> software running on production servers to handle live users and data of the intended audience.

Production Quality Code

> code that meets expectations in **reliability, efficiency**, etc., for production.

Clean Code

> readable, simple, and concise.

Modular Code

> logically broken up into functions and modules.

Module

> a file. Modules allow code to be reused by encapsulating them into files that can be imported into other files.

Making your code Modular makes it easier to

- ☐ Reuse your code
- ☐ Write less code
- ☐ Read your code
- ☐ Collaborate on code

Refactoring Code

> **restructuring your code to improve its internal structure, without changing its external functionality.** This gives you a chance to clean and modularize your program after you've got it working.program after you've got it working.

Writing Clean Code: Meaningful Names

Be descriptive and imply type

> E.g. for booleans, you can prefix with `is_` or `has_` to make it clear it is a condition. You can also use part of speech to imply types, like verbs for functions and nouns for variables.

Be consistent but clearly differentiate

> E.g. `age_list` and `age` is easier to differentiate than `ages` and `age`.

Avoid abbreviations and especially single letters

> (Exception: counters and common math variables) Choosing when these exceptions can be made can be determined based on the audience for your code. If you work with other data scientists, certain variables may be common knowledge. While if you work with full stack engineers, it might be necessary to provide more descriptive names in these cases as well.

Writing Clean Code: Meaningful Names (cont)

Long names != descriptive names

> You should be descriptive, but only with relevant information. E.g. good functions names describe what they do well without including details about implementation or highly specific uses.

Try testing how effective your names are by asking a fellow programmer to guess the purpose of a function or variable based on its name, without looking at your code. Coming up with meaningful names often requires effort to get right.

Writing Clean Code: Nice Whitespace

Organize your code with consistent indentation

> the standard is to use 4 spaces for each indent. You can make this a default in your text editor.

Separate sections with blank lines to keep your code well organized and readable.

Try to limit your lines to around 79 characters, which is the guideline given in the PEP 8 style guide.

> In many good text editors, there is a setting to display a subtle line that indicates where the 79 character limit is.

For more guidelines, check out the code layout section of PEP 8

Writing Modular Code

DRY (Don't Repeat Yourself)

> Don't repeat yourself! Modularization allows you to reuse parts of your code. Generalize and consolidate repeated code in functions or loops.

Abstract out logic to improve readability

> Abstracting out code into a function not only makes it less repetitive, but also improves readability with descriptive function names. Although your code can become more readable when you abstract out logic into functions, it is possible to over-engineer this and have way too many modules, so use your judgement.

Minimize the number of entities (functions, classes, modules, etc.)

> There are tradeoffs to having function calls instead of inline logic. If you have broken up your code into an unnecessary amount of functions and modules, you'll have to jump around everywhere if you want to view the implementation details for something that may be too small to be worth it. Creating more modules doesn't necessarily result in effective modularization.

Writing Modular Code (cont)

Functions should do one thing

> Each function you write should be focused on doing one thing. If a function is doing multiple things, it becomes more difficult to generalize and reuse. Generally, if there's an "and" in your function name, consider refactoring.

Arbitrary variable names can be more effective in certain functions

> Arbitrary variable names in general functions can actually make the code more readable.

Try to use fewer than three arguments per function

> Try to use no more than three arguments when possible. This is not a hard rule and there are times it is more appropriate to use many parameters. But in many cases, it's more effective to use fewer arguments. Remember we are modularizing to simplify our code and make it more efficient to work with. If your function has a lot of parameters, you may want to rethink how you are splitting this up.

Efficient Code

- ☐ Execute faster
- ☐ Take up less space in memory/storage

The project you're working on would determine which of these is more important to optimize for your company or product. When we are performing lots of different transformations on large amounts of data, this can make orders of magnitudes of difference in performance.
E.g. Sets faster than lists in python

Documentation

additional text or illustrated information that comes with or is embedded in the code of software.

Helpful for clarifying complex parts of code, making your code easier to navigate, and quickly conveying how and why different components of your program are used.

Several types of documentation can be added at different levels of your program:

- ☐ **In-line Comments** - line level
- ☐ **Docstrings** - module and function level
- ☐ **Project Documentation** - project level

Use version control

Version control, also known as revision control or source control, is the management of changes to documents, computer programs, large websites, and other collections of information. Each revision is associated with a timestamp and the person making the change. The most famous version control system is **Git**

Testing

Testing your code is essential before deployment. It helps you catch errors and faulty conclusions before they make any major impact.

Test driven development

> a development process where you write tests for tasks before you even write the code to implement those tasks.

Unit Test

> a type of test that covers a “unit” of code, usually a single function, independently from the rest of the program.

Log Messages

Logging is the process of recording messages to describe events that have occurred while running your software.

Be professional and clear

```
Bad: Hmmm... this isn't working???
Bad: idk.... :(
Good: Couldn't parse file.
```

Be concise and use normal capitalization

```
Bad: Start Product Recommendation Process
Bad: We have completed the steps necessary and
will now proceed with the recommendation process
for the records in our product database.
Good: Generating product recommendations.
```

Choose the appropriate level for logging

> DEBUG - level you would use for anything that happens in the program.
> ERROR - level to record any error that occurs
> INFO - level to record all actions that are user-driven or system specific, such as regularly scheduled operations

Provide any useful information

```
Bad: Failed to read location data
Good: Failed to read location data: store_id
8324971
```

Code Reviews

Code reviews benefit everyone in a team to promote best programming practices and prepare code for production.

Questions to Ask Yourself When Conducting a Code Review

Is the code clean and modular?

- ☐ Can I understand the code easily?
- ☐ Does it use meaningful names and whitespace?
- ☐ Is there duplicated code?
- ☐ Can you provide another layer of abstraction?
- ☐ Is each function and module necessary?
- ☐ Is each function or module too long?

Is the code efficient?

- ☐ Are there loops or other steps we can vectorize?
- ☐ Can we use better data structures to optimize any steps?
- ☐ Can we shorten the number of calculations needed for any steps?
- ☐ Can we use generators or multiprocessing to optimize any steps?

Is documentation effective?

- ☐ Are in-line comments concise and meaningful?
- ☐ Is there complex code that's missing documentation?
- ☐ Do function use effective docstrings?
- ☐ Is the necessary project documentation provided?

Is the code well tested?

- ☐ Does the code high test coverage?
- ☐ Do tests check for interesting cases?
- ☐ Are the tests readable?
- ☐ Can the tests be made more efficient?

Is the logging effective?

- ☐ Are log messages clear, concise, and professional?
- ☐ Do they include all relevant and useful information?
- ☐ Do they use the appropriate logging level?

Code Review
Code Review Best Practices

Conducting a Code Review

Use a code linter

This can save you lots of time from code review. Using a Python code linter like pylint can automatically check for coding standards and PEP 8 guidelines for you.

Conducting a Code Review (cont)

Explain issues and make suggestions

- ☐ `BAD: Make model evaluation code its own module - too repetitive.`
- ☐ `BETTER: Make the model evaluation code its own module. This will simplify models.py to be less repetitive and focus primarily on building models.`
- ☐ `GOOD: How about we consider making the model evaluation code its own module? This would simplify models.py to only include code for building models. Organizing these evaluations methods into separate functions would also allow us to reuse them with different models without repeating code.`

Keep your comments objective

- ☐ `BAD: I wouldn't groupby genre twice like you did here... Just compute it once and use that for your aggregations.`
- ☐ `BAD: You create this groupby dataframe twice here. Just compute it once, save it as groupby_genre and then use that to get your average prices and views.`
- ☐ `GOOD: Can we group by genre at the beginning of the function and then save that as a groupby object? We could then reference that object to get the average prices and views without computing groupby twice.`

Provide code examples

By **Ilyes KAANICH** (ilyes64)
cheatography.com/ilyes64/
www.linkedin.com/in/ilyes-kaanich/

Published 11th August, 2020.
Last updated 11th August, 2020.