Udacity

Software Engineer practice part 1

Software Engineer : efficient , Robust , maintainable codes

In this code:

1. clean / modular code
2. Improve Code efficiency
3. Effective documentation
4. Version Control

Production code : The code that can be run on server at production/service level

Clean code : Readable , Concise and Simple

Program should be broken to functom and midules ,

You can enpsulate a code with a module , module is a file containing lots of ficntoncs and other moduls

Code Refactor :

Term for restructuring the internl code without changing the external output the code

Why ?

Better for development in lon run

Reuse and maintainability

Become a better programmer , later on we can start writing the codes clean from the scratch

Tips :

***Use 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.*

DON’T REPEAT YOURSEL

***Be consistent but clearly differentiate****- E.g. age\_list and age is easier to differentiate than ages and age. , words for variables and verbs for functiosn*

***Avoid abbreviations and especially single letters***

***Long names != descriptive names : not more than necessary charactors***

***USE THE WITESPACE properly***

***Separate sections with blank lines (break )***

***Try to limit each line to 79 charactors***

***Spaghetti code ; codes common amongst data scientist***

***Abstract out logic to improve readability***

#### *Minimize the number of entities (functions, classes, modules, etc.) ; unnecessary amount of functions and modules*

#### *Functions should do one thing ; if your function name includes ‘And’ 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. In case you need to do something which is useful for yor program or smooth the program logic but necessary it could apply to other programs as well . In that case we can use arbitrary variables and they are readable enough.*

#### 

#### *Try to use as few arguments as possible ; no more than 3 parameters !!!*

#### *Efficient Code ; make a clean and modular*

#### *Reducing the time to run and less memory use ;*

#### *Codes that should work every someday , they don’t need to be optimized but for online deployment the code should be optimized!*

#### *For loop operations , use the vector operations ; numpy and pandas !!*

#### *Know your data structure and find the best function , google !!!*

#### *Masking is way faster than np.where !!!!*

#### *Masking in numpy ?*

#### *Documentation*

#### *Line level ; in-line comments ; if they are done good , you don’t need to read the code , however , many of them in the code is the indication of bad code and the code needs refactoring ; it is not a hard rule , the code should be readable as clean and few comments as possible !*

#### *Function or module ; string at top of functions & moduls*

#### *Triple quote ; doc’s string , purpose of the function or module , Arguments and Returns !!!*

#### *Read me*

Whether it's an application or a package, your project should absolutely come with a README file . At a minimum, this should explain what it does, list its dependencies, and provide sufficiently detailed instructions on how to use it. You want to make it as simple as possible for others to understand the purpose of your project, and quickly get something working.

#### *Git , version Control ,*

#### *Git log*

# Scenario #3

Let's walk through the git commands that go along with each step in the scenario you just observed in the video above.

#### Step 1: Andrew commits his changes to the documentation branch, switches to the development branch, and pulls down the latest changes from the cloud on this development branch, including the change I merged previously for the friends group feature.

##### Commit changes on documentation branch

git commit -m "standardized all docstrings in process.py"

##### Switch to develop branch

git checkout develop

##### Pull latest changes on develop down

git pull

#### Step 2: Then, Andrew merges his documentation branch on the develop branch on his local repository, and then pushes his changes up to update the develop branch on the remote repository.

##### Merge documentation branch to develop

git merge --no-ff documentation

##### Push changes up to remote repository

git push origin develop

#### Step 3: After the team reviewed both of your work, they merge the updates from the development branch to the master branch. Now they push the changes to the master branch on the remote repository. These changes are now in production.

##### Merge develop to master

git merge --no-ff develop

##### Push changes up to remote repository

git push origin master

### Resources

There's a great article on a successful git branching strategy that you should really read [**here**](http://nvie.com/posts/a-successful-git-branching-model/).

### Note on Merge Conflicts

For the most part, git makes merging changes between branches really simple. However, there are some cases where git will be confused on how to combine two changes, and asks you for help. This is called a merge conflict.

Mostly commonly, this happens when two branches modify the same file.

For example, in this situation, let’s say I deleted a line that Andrew modified on his branch. Git wouldn’t know whether to delete the line or modify it. Here, you need to tell git which change to take, and some tools even allow you to edit the change manually. If it isn’t straightforward, you may have to consult with the developer of the other branch to handle a merge conflict

The function of branch :

1. For your branch you can have different remote repo by

git remote add [name\_of\_your\_remote] [name\_of\_your\_new\_branch]

1. You can push your changes to a remote branch (a branch of the origin):

git push <REMOTE\_NAME> <LOCAL\_BRANCH\_NAME>:<REMOTE\_BRANCH\_NAME>

ex: git push origin stage1:development ; This will create branch named “**development**” on remote git repository and push data from local branch “**stage1**”

Unit test :

Should be repeatable and shouldn’t be poor way of testing so that we have to do things manually. Imagine you have hundreds of functions to test and checking them all cannot be done manually. Right? After all you have to decide to put your test program to into the test.

Still it does not tell you if it gives you the correct answer and we have to check the results manually. Test program runs automatically but the results are still being verified manually.

For instance the program runs some inputs and print the results and comparing the results should be done by yourself .

What to do ? Solution is to use assert for all the unit test ,

Problem? Basically unit test should tell us if the results match and if it fails which input does not give the expected results . Assert can give us those info however if it fails one of the input , the program stops running and does not go through the rest of unit tests . right ?

Solution: python has lots of great tools for unit test

Pytest ; pip install …

Note :

* Test file should be starting by test\_YourFileName , pytest recognizes the test files in that format
* For each input the test function should be defuined separately with an assert for each of them ,
  + def test\_YourFunction\_input(input) :
    - assert (yourfunction(input) == expected\_result)
* So this way the pytest could understand which one to test ,
* Check pytest configuration and other capabilities
* When you run pytest it gives you some dots and F , ..F…, meaning that the 4th one failed
* It is wise to have assert for each function otherwise you won’t know which one has been failed ,
* Again the test file should start with ‘test\_’ and the test function should also start with ‘test\_’

# Test Driven Development :

Writing a test function for your code with different inputs that verifies your implementation. If the test passes the implementation is done.

Writing test function before even starting writing your function .

Benefits:

1. It helps verifying your implementation
2. Later on in refactoring, still we can get the same test done to make sure of refactoring
3. Later on in production , making sure that the function is repeatable regardless of hardware and other external factors

LOGGING :

Logging Is very important in software development and also data science . Imagine you put yor program on run over night and tomorrow it crashes , so , good logging shows why it crashed .

# Log Messages

Logging is the process of recording messages to describe events that have occurred while running your software. Let's take a look at a few examples, and learn tips for writing good log messages.

#### *Tip: Be professional and clear*

Bad: Hmmm... this isn't working???

Bad: idk.... :(

Good: Couldn't parse file.

#### *Tip: 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.

#### *Tip: 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

#### *Tip: Provide any useful information*

Bad: Failed to read location data

Good: Failed to read location data: store\_id 8324971

Code Review :

Catching the errors

Knowledge share and productive code

#### *Is the code clean and modular?*

#### *Is the code efficient?*

#### *Is documentation effective?*

#### *Is the code well tested?*

#### *Is the logging effective?*

As you may have noticed, with code reviews you are now dealing with people, not just computers. So it's important to be thoughtful of their ideas and efforts. You are in a team and there will be differences in preferences. The goal of code review isn't to make all code follow your personal preferences, but a standard of quality for the whole team.

#### *Tip: Use a code linter , it s highly configurable and it is useful for following the coding standards*

#### 