DEVELOPER ONBOARDING

Author(s)	**Siva Sankar, B Ravikanth**
Reviewer(s)	**Ritesh Srivastava**
Contributor	
Version	**1.1**
Date	**February 15th 2020**

Git

- All developers are expected to be familiar with git commands (and underlying concepts) such as **commit**, **stash**, **branching**, **clone**, **pull** and **push**.
- No file which is generated by a program should be committed - for example csv, json etc.
- Configurations should not be committed, credentials should always be stored separately by each developer in .env files. These .env files are not to be committed.

Resources and Exercises

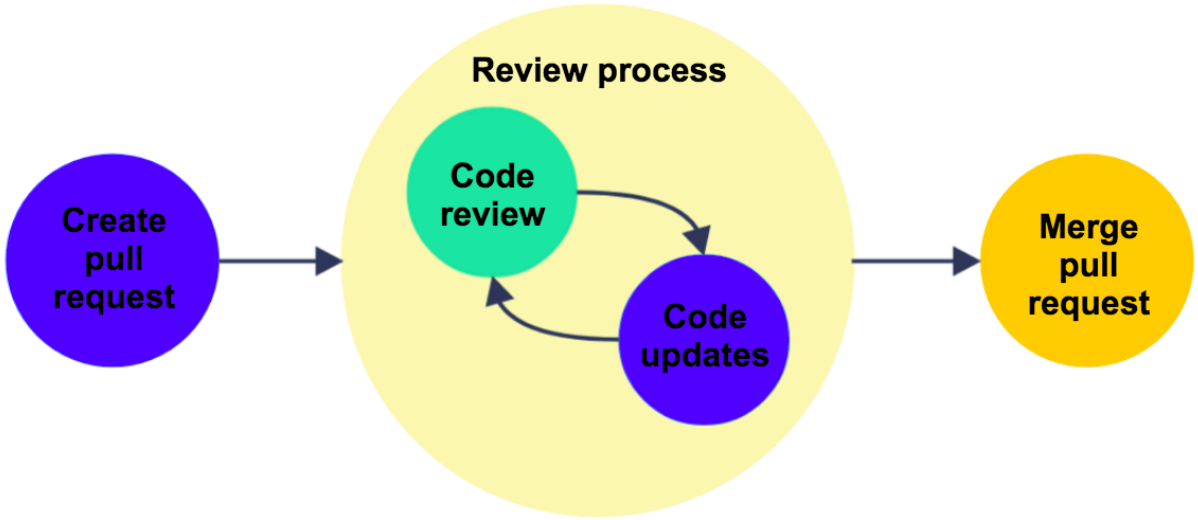
Learn Git Branching Interactively
RealPython - Git Introduction

Standard Process Flow

- Every developer must work on his/her own branch, named after the story ID on JIRA

```
# Get the latest version of the repository
git pull
# 1) Create a new branch using the -b flag
git checkout -b <branch-name>
# 2) Edit add and commit your files on the new branch
git status
git add <file-name>
git commit -m "commit-message"
# 3) Push your branch to the remote repository
git push -u origin <branch-name>
```

- Developers must ensure that they are not committing files that they aren't working on or are being worked on by other developers. Code must be checked into the remote daily EOD
- Once the work on a story or a group of related stories is done, then the following process must be followed to achieve closure.



- Pull request will need to be reviewed and approved by at least one qualified development team member in order to be merged into the master branch.

Code style guidelines

- Python naming and code style conventions must be strictly followed according to PEP-8
- Developers must use code linters to inspect code for style and logical inspections. The recommended linter is flake8.
  Guide to setup the linter in Pycharm.
- Imports must be optimised in code, no unnecessary imports should be a part of the code written.
- In order to ensure consistent code formatting, code must be run through Black tool before check-in. To ensure style consistency within jupyter notebooks, the plugin nb_black must be used by adding the following code to the first cell of every notebook

```
1 %load_ext nb_black
```

- Black and Flake8 can be auto-triggered upon file save in PyCharm using Filewatchers plugin. The set up process is described in this link. The process described here for Black can be replicated for Flake8.
- All the aforementioned libraries and plug-ins must be installed in the virtual environment created for use and must be part of the development requirements.

Object Oriented Programming

All developers are expected to know and employ OOPs concepts such as classes, inheritance, difference between class and instance methods and attributes, static methods. Developers need to write code to create a program that creates a simple BlackJack game. The requirements are as follows-

- The game needs to have one player versus an automated dealer.
- The player can stand or hit.
- The player must be able to pick their betting amount.
- You need to keep track of the player's total money.
- You need to alert the player of wins, losses, or busts, etc...

Gameplay rules are as follows -

To play a hand of Blackjack the following steps must be followed:

1. Create a deck of 52 cards
2. Shuffle the deck
3. Ask the Player for their bet
4. Make sure that the Player's bet does not exceed their available chips
5. Deal two cards to the Dealer and two cards to the Player
6. Show only one of the Dealer's cards, the other remains hidden
7. Show both of the Player's cards
8. Ask the Player if they wish to Hit, and take another card
9. If the Player's hand doesn't Bust (go over 21), ask if they'd like to Hit again.
10. If a Player Stands, play the Dealer's hand. The dealer will always Hit until the Dealer's value meets or exceeds 17
11. Determine the winner and adjust the Player's chips accordingly
12. Ask the Player if they'd like to play again

Playing Cards

A standard deck of playing cards has four suits (Hearts, Diamonds, Spades and Clubs) and thirteen ranks (2 through 10, then the face cards Jack, Queen, King and Ace) for a total of 52 cards per deck. Jacks, Queens and Kings all have a rank of 10. Aces have a rank of either 11 or 1 as needed to reach 21 without busting. As a starting point in your program, you may want to assign variables to store a list of suits, ranks, and then use a dictionary to map ranks to values. Use classes to help you define the Deck and the Player's hand.

As a bonus, expand the game to have multiple players.

Command Line Operations

All developers are expected to be comfortable with using the Linux terminal/shell for simple navigation actions, SSH login, transferring files and git commands. Developers should be able to use command line editors such as vi/nano and understand how environment variables can be set and accessed. Understanding how to use Linux Makefiles is a bonus.

Python developers are expected to know how to parse command line arguments given to their scripts. Developers should focus on using the Click framework to achieve this as well as on the inbuilt argparse module.

Resources
Click Tutorial 1
Click Tutorial 2

Excercise
Write a python program which can be executed via the commandline to take path to a text file as input and write out the counts of vowels in the file. Use inbuilt Python data structures and modules where applicable.

Intermediate and Advanced Python Concepts

- Decorators(Bonus)
- Logging
- Working with JSON files
- Importing environment variables
- Pathlib module to work with paths
- Exception Handling
- Python Design Patterns(Bonus)
- Python Best Practices - The Hitchhiker's Guide to Python

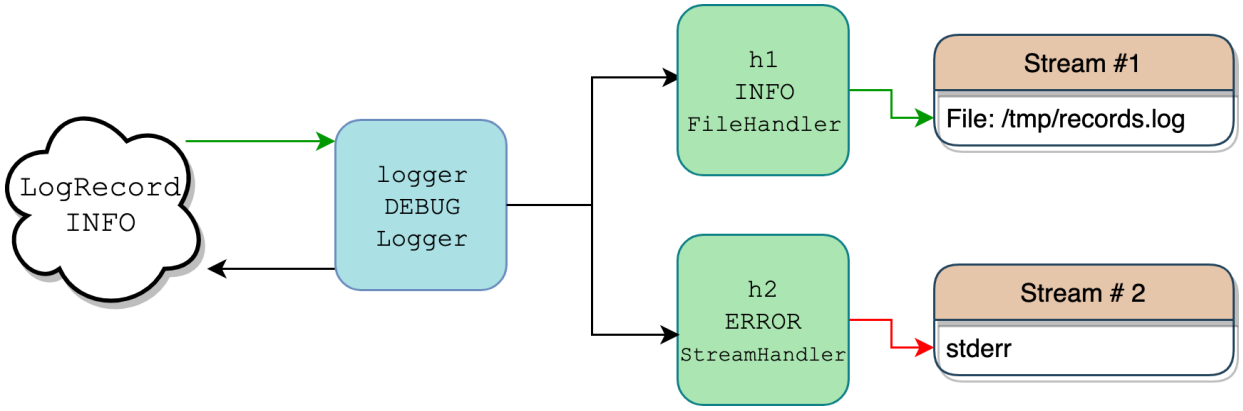
Logging guidelines

Treat logs as event streams

- *Logs* provide visibility into the behavior of a running app.
- In server-based environments they are commonly written to a file on disk (a “logfile”); but this is only an output format.

> Logs are the stream of aggregated, time-ordered events collected from the output streams of all running processes and backing services. Logs in their raw form are typically a text format with one event per line (though backtraces from exceptions may span multiple lines). Logs have no fixed beginning or end, but flow continuously as long as the app is operating.

- Developers should not attempt to write to or manage logfiles. Instead, each running process writes its event stream, unbuffered, to stdout.
- During local development, the developer will view this stream in the foreground of their terminal to observe the app’s behavior.
- In staging or production deploys, each process’ stream will be captured by the execution environment, collated together with all other streams from the app, and routed to one or more final destinations for viewing and long-term archival.



Resources

RealPython - Logging Introduction
The Hitchhiker's Guide to Python

Development guidelines for data scientists

- Notebooks are for exploration and communication only.
- Data munging and model training for production shouldn't be dependent on notebooks.
- Specific scripts should be written to maintain data lineage as various transformations happen from raw data to training set.
- Treat analysis as a DAG. Once exploration is done, the development should follow the below track-
    - All steps from data gathering, data preprocessing, feature generation to model training should be put into individual scripts that can be run in order to reproduce results.
    - This is especially true from a data engineering stand point. It is hard to keep track of data versions when using notebooks to create the data.
    - We usually end up with multiple data files or tables in the database with not very informative ways. The steps needed to create the same data again are usually lost inside notebooks.
    - By using scripts to generate data we need, we ensure that the code that was used to generate the data is checked in.
    - We can create the DAGs using simple tools like airflow or Makefiles.
    - DAGs should be documented using simple markdown files or sphinx documentation.
- Guidelines while using notebooks to share code, reports and results-
    - No hard paths inside the notebooks, use relative paths or read from config files/env files.
    - All files relevant for model development for core components must be stored in AWS S3 buckets.
    - Ensure that all cells are in order of execution.
        - Generally while developing we tend to create cells as and where necessary which usually introduces out of order code execution steps.
        - The final notebook being shared should be reproducible and that means when someone uses run all cells functionality, they should be able to produce desired results without any errors.

    - DRY - If a piece of code is being or will be used multiple times within notebooks or across notebooks, it should be modularized.

        - Better to develop these functions or classes in an IDE with code linting, type checking and debugging capabilities than inside notebooks.
        - By following a specific module structure, your code is much better organized. This template is beneficial.
        - The above template allows users to easily import code from python modules into the notebook.
        - This modularity will also allow us to develop better production scripts and write unit tests.
    - Use nb_black extension to maintain code style consistency and readability
    - Use nb_dime extension to get readable and usable diffs of notebooks.

Django

Quadratyx follows Microservice pattern for developing assets. All developers are required to familiarize with Django and Django REST Framework in order to develop web services.

Resources
Introduction to Django
Introduction to Django REST Framework
What is REST?
Microservice Design Patterns
Requests library guide

Excercise
Write a Django REST API which takes in a string as input and returns a list of phone numbers extracted from the string. Save all extracted phone numbers along with the time at which the service was requested in a MongoDB collection. Create test cases for your API. Create a git repository on Github and commit your code regularly with informative commit messages.

Bonus - Write a Django API which takes in the name of the user and returns predicted gender. Save all the names alongwith the predicted gender and the time at which teh service was requested to a MongoDB collection. Create test cases for your API. Create a git repository on Github and commit your code regularly with informative commit messages.

Database Interaction and Management

Python developers should become familiar with SQLAlchemy and PyMongo modules to interact with various SQL based and NoSQL databases.

Resources

SQLAlchemy + SQLite tutorial

PyMongo Tutorial