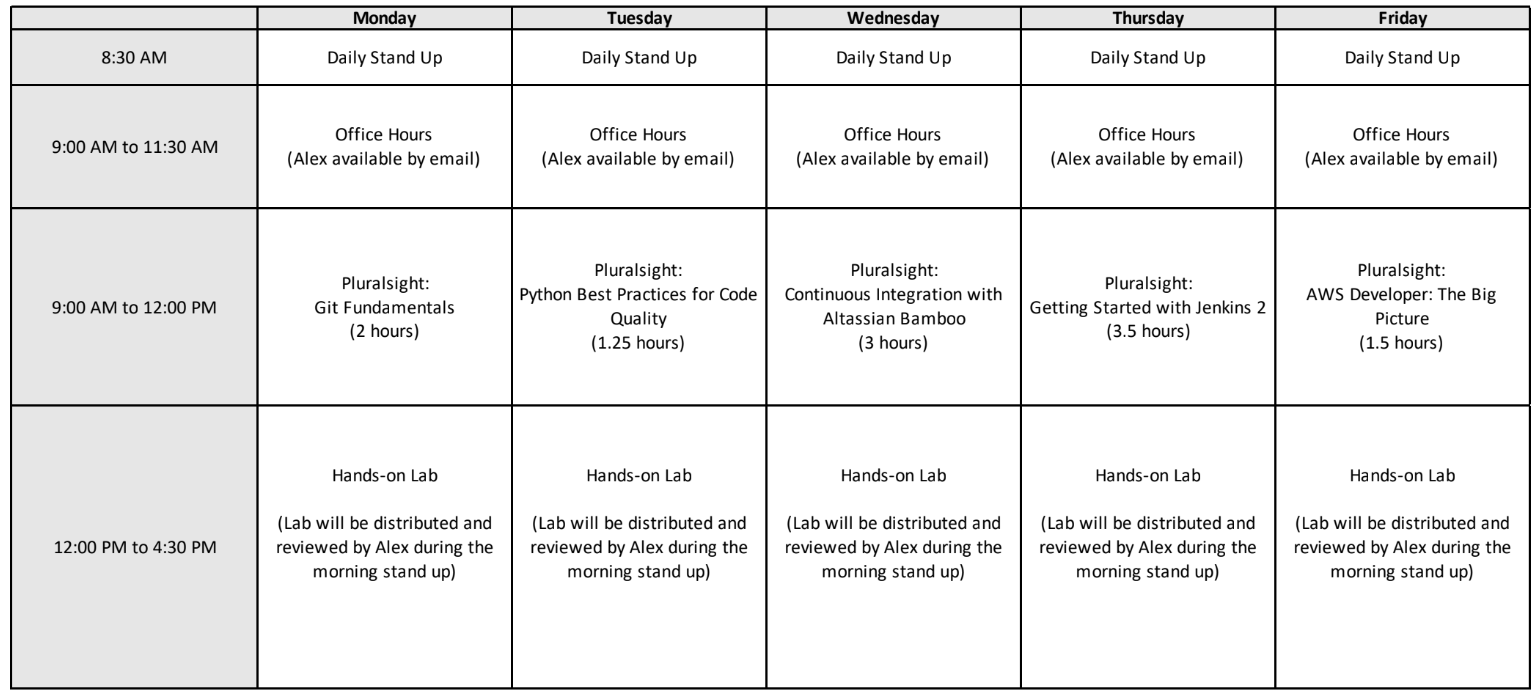
Use the class cheat sheet and materials for reference: <https://bitbucket.org/lmcohort2/materials/src/master/cheatsheet.md>



## Finish Yesterdays self-paced assignments

## Finish Reading The Phoenix Project

## Coding Practice

Remember, the goal here is practice. Try to write easy to read, functional code verified with tests. Do not rush to complete all the problems or even a single problem, just focus on learning. The below "Katas" are in order of complexity.

1. Fizz Buzz

Write a function that takes in a number and returns "Fizz" if that number is evenly divisible by 3 and "Buzz" if it is easily divisible by 5 and "FizzBuzz!" if it is divisible by both 3 and 5.

|  |
| --- |
| def fizzOrBuzz(number):  return "FizzBuzz!" |

2. Bowling

<http://codingdojo.org/kata/Bowling/>

|  |
| --- |
| import sys  def calculateScore(bowling\_frames\_string):  return 0  if \_\_name\_\_ == "\_\_main\_\_":  # You can call this script with an argument  # script.py "5/ X X X X X X X X X X X"  print(calculateScore(sys.argv[1])) |

3. Arg parser

In this challenge, you implement your own arguments parser instead of using the argparse library like in class. You are only allowed to make use of the [sys](https://docs.python.org/3/library/sys.html) library (sys.argv) when retrieving arguments.

<http://codingdojo.org/kata/Args/>

The most approachable way to implement this is to call a function that returns a dictionary containing the values of flags. For example, if you executing "myprogram -v" then the arguments dictionary would be {"verbose": true} For making the code unittestable, you should take in a parameter that is a list of arguments and have the \_\_main\_\_ section pass in sys.argv.

|  |
| --- |
| import sys  def parseArgs(args\_list):  return {}  if \_\_name\_\_ == "\_\_main\_\_":  args = parseArgs(sys.argv) |

|  |
| --- |
| myprogram.py -v -L 3  {  "verbose": True,  "list": [3],  }  if args["verbose"]:  log.debug("I am verbose output") |