**Name:** Kerry Bosworth  
**Date:** Dec 5th, 2019  
**Assignment:** 09  
**GitHub Repo**: <https://github.com/yukkutobu777/IntroToProg-Python-Mod09>  
**GitHub Website:** <https://yukkutobu777.github.io/IntroToProg-Python-Mod09/>

# Inheritance

**Introduction**

This week we moved the classes out of the main script and into their own respective files. Then continued to work with object classes and introduced inheritance.

**Main Module**

The following main module code was added to the scripts that should not be directly executed:

**if** \_\_name\_\_ == **"\_\_main\_\_"**:  
 **raise** Exception(**"This file is not meant to ran by itself"**)

Our IO class had the following because it referenced the DataClasses in a static method and needed a way to find the objects located in another file.

**if** \_\_name\_\_ == **"\_\_main\_\_"**:  
 **raise** Exception(**"This file is not meant to ran by itself"**)  
**else**:  
 **import** DataClasses **as** DC

The main script had the following. It needed a map to all the other files it would need.

**if** \_\_name\_\_ == **"\_\_main\_\_"**:  
 **import** sys  
 **from** DataClasses **import** Employee **as** Emp  
 **from** ProcessingClasses **import** FileProcessor **as** Fp  
 **from** IOClasses **import** EmployeeIO **as** Eio  
**else**:  
 **raise** Exception(**"This file was not created to be imported"**)

**Class Inheritance**

By default classes inherit the base object class. You can define it any of the three ways:

**class** Person:

**class** Person():

**class** Person(object):

Any additional class that are added can use ‘object’ also or you can indicate a parent class.

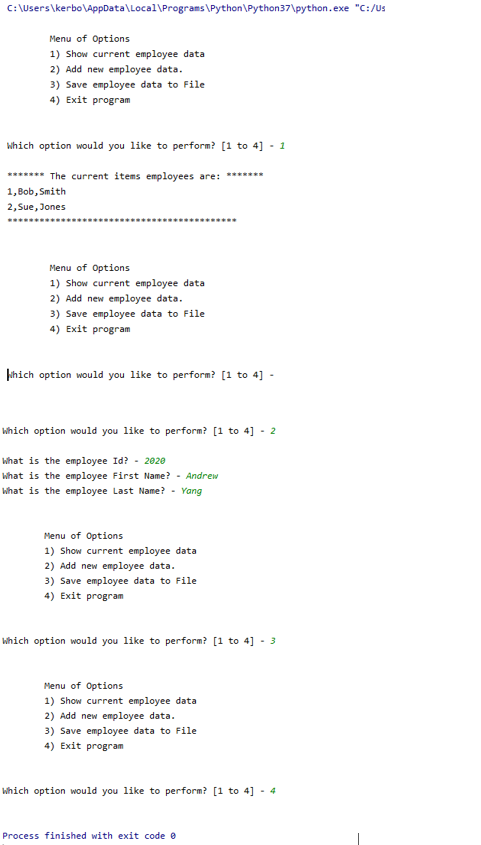
**Programming Assignment**

This week we created a Person (parent) class and Employee (child) class. When defining the employee class, we set it up like this:

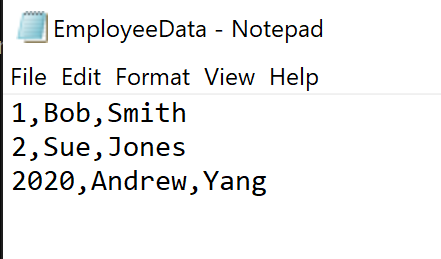
**class** Employee(Person):

In the main program, after listing the classes to import, we declared our variables and then started the main body of the script. First the file was read and the objects were stored into a list table. Then this was passed as an argument for various operations like printing the current list, adding to the list and writing the list to the file.

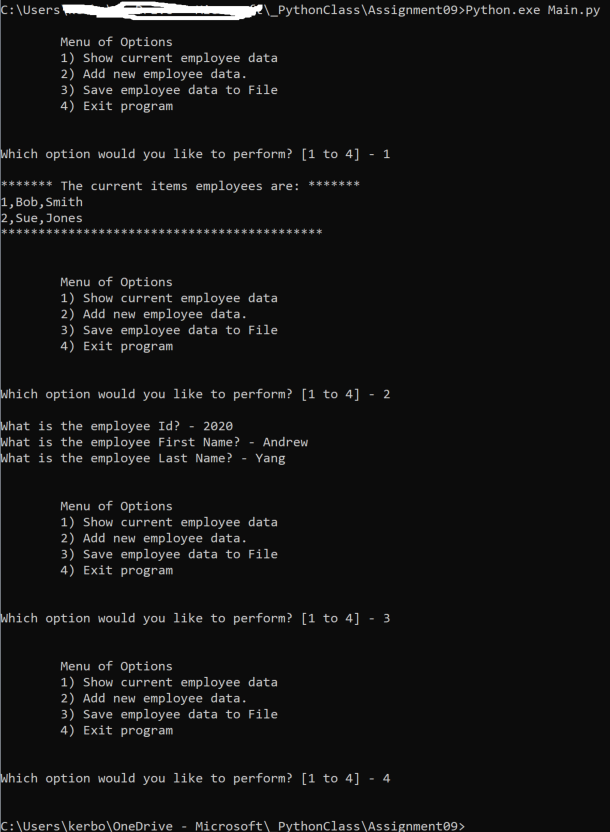
After running through the Test Harness code and then building the main functioning code the final output is below (figure 1 - 4).



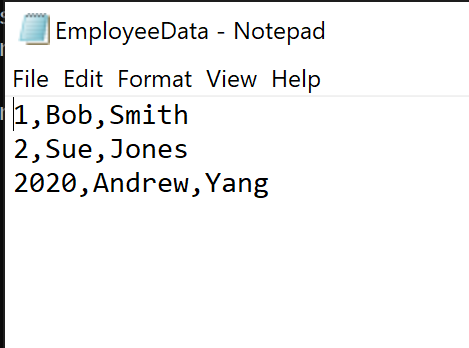
**Figure 1:** Run of program in PyCharm



**Figure 2:** Output file after running in PyCharm



**Figure 3:** Run of program in command line

****

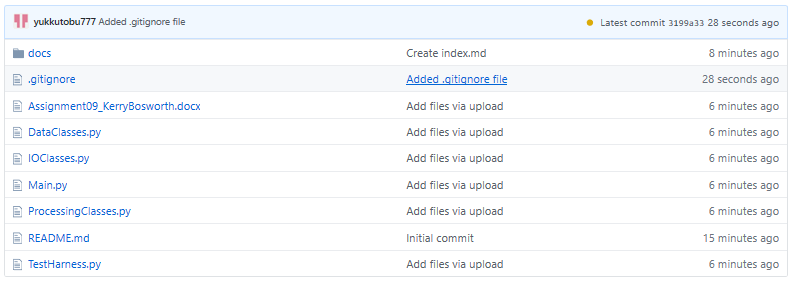
**Figure 4:** Output file after running in command line

**Git Command Line**

To date we have been creating repositories on GitHub. Last week we learned how to update them with GitHub Desktop. This week we learned Git Command line. Here are the steps to get that working.

1. Installed Git (<https://git-scm.com/book/en/v2/Getting-Started-Installing-Git>)
2. Started the Git Bash shell
3. Checked version with:   
   git –version
4. Set global config variables:  
   $ git config --global user.name "Kerry Bosworth"

$ git config --global user.email [<input](mailto:yukkutobu777@gmail.com) my email>

1. Created an IntroToProg-Python-Mod09 on GitHub.
2. Cloned it locally.  
   git clone <https://github.com/yukkutobu777/IntroToProg-Python-Mod09>  
   cd IntroToProg-Python-Mod09
3. Went into new local directory and did some initial configuration.  
   git init  
   vi .gitignore (exclude EmployeeData.txt and \*pycache\*)
4. Checked status, did commit and then pull/push up to the remote repository.  
   git add -A   
   git status  
   git commit -m "Added .gitignore file"  
   git pull origin master  
   git push origin master (asked for my userid/password).
5. Verified new local file was pushed up to repository (See figure 5).  
     
     
   **Figure 5**: List of repository on GitHub.  
     
   I noticed the updates for the Assignment doc were not being picked up. This may be expected for this file type. For now, I will manually put the updated Word doc in the online repository and continue to test on future assignments.

**Summary**

As more complex programs are constructed, Unified Modeling Language is a good tool to lay out the idea, map the code and determine dependencies. There are three different types of diagrams, class (shows parent/child relationship), use case (actor/system explain components) or composition (looks at objects made from classes, composition/aggregation). A combination of a three can be used on a new project as appropriate. I look forward to using those on future assignments.