Nick Hertlein

March 18, 2021

IT FDN 110 B: Foundations Of Programming: Python

Assignment09

GitHub URL: https://github.com/nhertlein/IntroToProg-Python-Mod09

OOP Programming & Modules

# Introduction

This document will illustrate the knowledge gained from the ninth lecture and the media portion of the assignment. In addition, I will cover the creation of a Python script that calls in other scripts as modules. The modules were mostly built off existing provided code as part of the assignment. One thing that was different about this assignment is we had to make a “test harness” script which was a great tool to use to run pieces of code to test general functionality before incorporating them into a more complex use. For this task I mostly use the command line or debug, so this was a different way of doing things for me. It always nice to have some additional tools availale!

# Writing the Script

For this week’s assignment I made a project in PyCharm in the Assignment09 folder of the C:\\_PythonClass directory. I made a new DataClasses.py script to use as a baseline to copy in the functionality from the various listing scripts so I could build on it and add new functionality in the steps of the assignment. The first step was to make a header for the file and populate it. The next step was to add an exception that would notify the user this script is not made to be run standalone and must be called from the main.py script if it is tried to be run by itself. This is a small detail, but when you have a lot of scripts of functions sometimes it’s hard to know which one is the main script if naming convention is not clear!

I copied the Person class code from Listing06.py into the DataClasses.py script only modifying the change log. I then made a script TestHarness.py which I used to import the Person class from the DataClasses script and test the functionality. This was necessary because the DataClasses script is set up to only be run from another script so we need to mimic this functionality in order to make sure things work as intended (Figure 1).

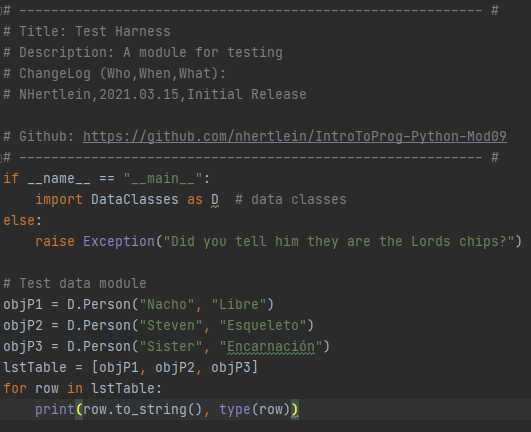


Figure 1. Test Harness code to test Person class functionality

Once I confirmed everything worked properly I moved on to copy in the Employee class code from Listing09.py into the DataClasses script. The only change I made to the code was to update the change log. As with the previous class that was added to the DataClasses script I used the TestHarness script to make sure the functionality worked correctly and I understood how it was working (Figure 2).

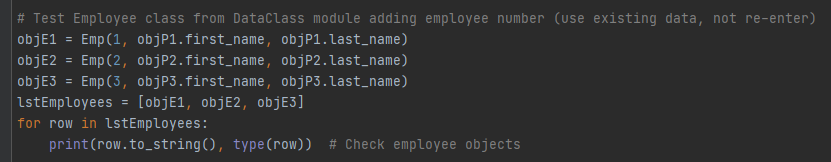


Figure 2. Test Harness code to test Employee class functionality

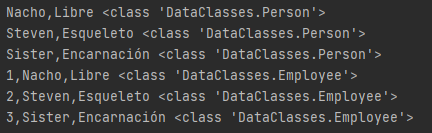


Figure 3. Output from TestHarness script (includes testing of Person class which feeds into it)

The next part of the assignment required making a file ProcessingClass.py and adding in code for the FileProcessor class from Listing07.py. I added a header to the file and I modified the change log for the FileProcessor class. I used the TestHarness script to make sure the functionality worked and I understood it as was done previously (Figure 4).

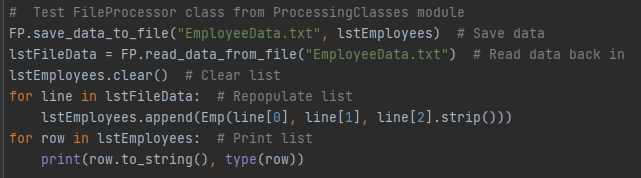


Figure 4. TestHarness code to test FileProcessor class functionality.

The last class to be created as part of the assignment was the EmployeeIO class in the IOClasses.py script. For this another new script IOClasses.py was made and the EmployeeIO class was carried over from Listing11.py. This class utilized the Employee class from DataClasses so it needed to be imported to ensure correct functionality. Like previous files a header was added, and the change log for the EmployeeIO class was modified. I used the TestHarness script to make sure the functionality worked and I understood it as was done previously (Figure 5).

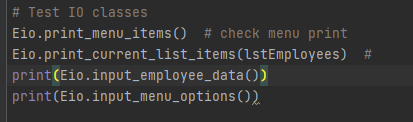


Figure 5. TestHarness code to test EmployeeIO class functionality

Once we knew all the classes and modules were working correctly we could move on to making the Main.py script which is the mothership that calls all the classes and methods at the correct times based on what the user wants to do. I added the header to the file and filled out the change log and moved to import the modules that would be needed to execute the script. The modules are only imported if the script is called directly, otherwise it gives an error message to inform this script is not intended to be imported as a module. The method I used to import the modules was to import the specific class from the module instead of just importing the whole module. This didn’t really make a difference here, but it’s just a way to experiment with importing only the class you need and not the entire module (Figure 6).

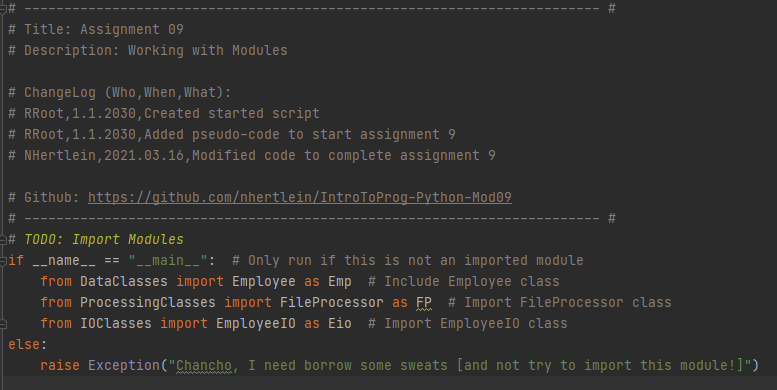


Figure 6. Main script header and importing of modules

The next section of the script defined the variables that would be used as placeholders as well as importing the existing objects from EmployeeData.txt (Figure 7).

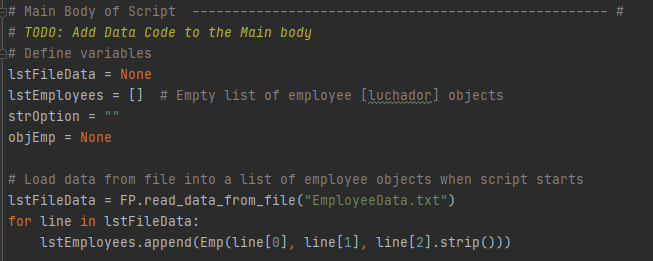


Figure 7. Defining variables to be used in Main script and reading in data

The final part of the program is the main loop which circles to allow the user to make various selections until they decide to exit the program. Due to using classes to execute the tasks the code is pretty straight forward to read (Figure 8).

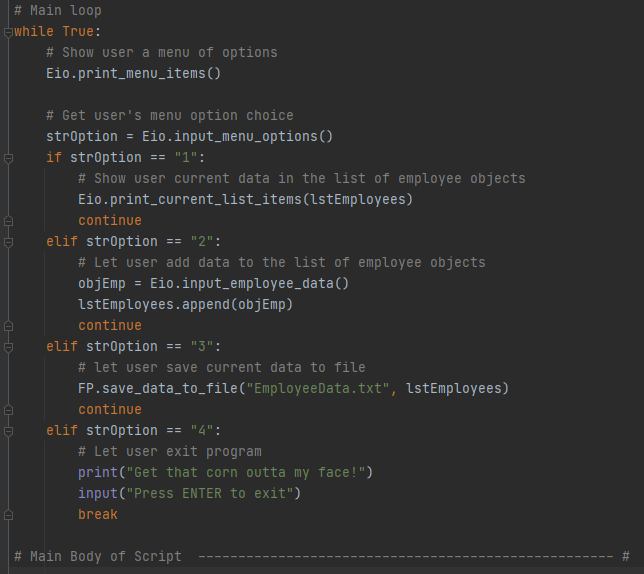


Figure 8. Main loop

# Summary

This module was intended to be an expansion of previous assignments with the addition of using classes in a different way by importing them from other scripts to make them available rather than having them all be part of a single script. I think this helps make the code much more clear and concise. Also it allows you to have a library of common functions so you can use them in future scripts without having to keep copy/pasting the code which is error prone. Another advantage is if you find a bug and fix it, you fix it in every script that references the function automatically without having to trace down all places the code and its mutations live!