Modules

# Introduction

We have been asked to describe and demonstrate the use of Python modules. The program reads an existing list from a file and can print the list, add employees, or save the information back to the file. When adding employees, the program asks for an ID number and a first and last name. The data is stored as attributes in a class object. Each object is then added to a list to create a table of data which can be printed or saved. This is similar to the many of the programs developed for assignments, but uses modules and classes to organize the script similar to Module 8.

# Writing the script

To begin with, we were given starting templates to follow in developing the code. The code was broken down into files, or modules, for data, processing, and presentation, along with the main body of the script. While developing the final code, many of the functions were copied to another file to be worked individually then combined to generate the complete script.

A benefit to this method is that multiple people can work on different parts of the code at the same time. Since each module can be a separate file, there are no issues with having to allow multiple people to write to a shared file.

## Test Harness

For this assignment, we are creating and testing the code incrementally. The module files are written to throw an error if they are run directly rather than called from another script as shown in Figure 1.

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

Figure 1: Module files

### Test One: Person Class

The first step is to create and test the Person class in the DataClasses module. We are given the code for the Person class and the testing file, although I changed the import call for Listing06 to DataClasses as shown in Figure 2.

if \_\_name\_\_ == **"\_\_main\_\_"**:  
 import DataClasses as D *# data classes* import Listing07 as P *# processing classes*else:  
 raise Exception(**"This file was not created to be imported"**)

Figure 2: Import DataClasses to test

The Person class defines an object with first and last name attributes, both of which are set as non-numeric and formatted with Title case. Together, they are returned to the main function as a string.

The script places the Person objects in a table, then prints out the first and last name as a string and the class for each object (see Figure 3 lines 1 and 2). In addition, the PersonData.txt file is created with the object attributes as shown in Figure 4. The file is read and used to regenerate the objects, which are then printed on screen (see Figure 3 lines 3 and 4).

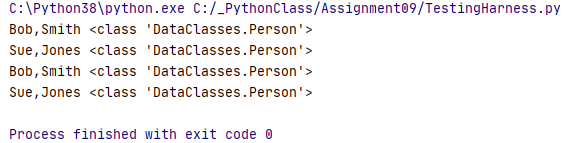


Figure 3: Test one print results

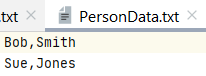


Figure 4: PersonData.txt file

### Test Two: Employee Class

The next step is to create and test the Employee class in the DataClasses module. This time, the import call only requests the Employee class, rather than the entire DataClasses module as shown in Figure 5.

if \_\_name\_\_ == **"\_\_main\_\_"**:  
 from DataClasses import Employee as Emp *# Employee class only!* import Listing07 as P *# processing classes*else:  
 raise Exception(**"This file was not created to be imported"**)

Figure 5: Import Employee class to test

The Employee class defines an object with an employee ID and first and last name attributes. The ID is set to numeric only, but the name formats are controlled by the parent class, Person. Together, they are combined and returned to the main function as a string.

The script places the Employee objects in a table, then prints out the employee ID, first and last name as a string and the class for each object (see Figure 6 lines 1 and 2). In addition, the EmployeeData.txt file is created with the object attributes as shown in Figure 7. The file is read and used to regenerate the objects, which are then printed on screen (see Figure 6 lines 3 and 4).

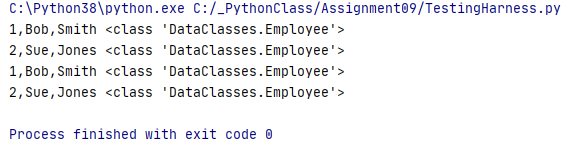


Figure 6: Test two print results

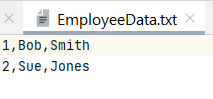


Figure 7: EmployeeData.txt file

I am getting an error (see Figure 8) but it is not currently affecting the running of the script.

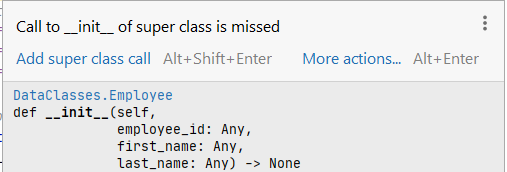


Figure 8: Employee class error

### Test Three: FileProcessor Class

The next step is to create and test the FileProcessor class in the ProcessingClasses module. Since the given code had already been saved in the assignment folder, I used Refactor to rename Listing07 to ProcessingClasses. Surprisingly, it changed the import call in the TestingHarness file as well, as shown in Figure 9.

if \_\_name\_\_ == **"\_\_main\_\_"**:  
 from DataClasses import Employee as Emp *# Employee class only!* import ProcessingClasses as P *# processing classes*else:  
 raise Exception(**"This file was not created to be imported"**)

Figure 9: Import FileProcessor to test

This class contains static methods for reading and writing data from a file. These are basically the same functions used in prior weeks’ assignments, just contained and imported from a separate file (see Figure 10).

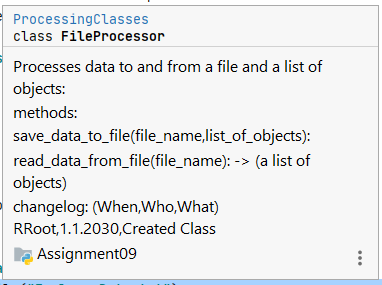


Figure 10: FileProcessor methods

### Test Four: EmployeeIO Class

The next step is to create and test the EmployeeIO class in the IOClasses module. This time, the import call only requests the EmployeeIO class, rather than the entire IOClasses module as shown in Figure 11.

if \_\_name\_\_ == **"\_\_main\_\_"**:  
 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"**)

Figure 11: Import EmployeeIO to test

The script runs through the existing tests for the data and processing modules then moves to the IO classes. The intent is that the menu will print, followed by the list of employees. Data for a new employee will be requested and then then menu option will be requested. Running the code throws the error shown in Figure 12 after the menu is printed.

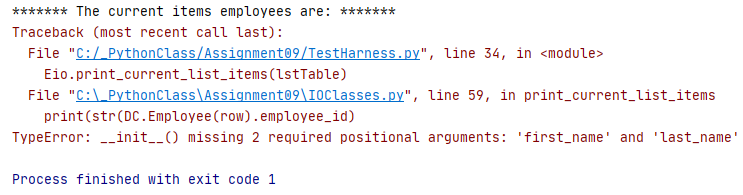


Figure 12: Test four error

Going to line 59 in IOClasses, I can see that each of the print rows is throwing the error shown in Figure 13.

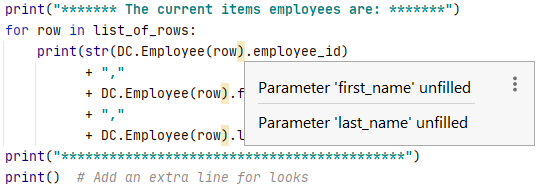


Figure 13: IOClasses error

This could be fallout from the error in Figure 8 - class Employee inherits first and last name from the Person class, but they are also defined within the Employee class. I revised the code to refer back to the parent class for initialization as shown in Figure 14.

def \_\_init\_\_(self, employee\_id, first\_name, last\_name):  
 *# Attributes* super(Employee, self).\_\_init\_\_(first\_name, last\_name)  
 self.\_\_employee\_id = employee\_id  
 *# self.first\_name = first\_name  
 # self.last\_name = last\_name*

Figure 14: Employee class code fix

Additionally, each line of the print statement is calling the Employee class, which requires three arguments, but is only passing one. Replacing the method code as shown in Figure 15 cleared the error and allowed the script to finish properly (see Figure 16).

@staticmethod  
def print\_current\_list\_items(list\_of\_rows: list):  
 *""" Print the current items in the list of Employee rows  
  
 :param list\_of\_rows: (list) of rows you want to display  
 """* print(**"\*\*\*\*\*\*\* The current items employees are: \*\*\*\*\*\*\*"**)  
 for row in list\_of\_rows:  
 print(str(row.employee\_id)  
 + **","** + row.first\_name  
 + **","** + row.last\_name)  
 print(**"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"**)  
 print() *# Add an extra line for looks*

Figure 15: EmployeeIO code fix

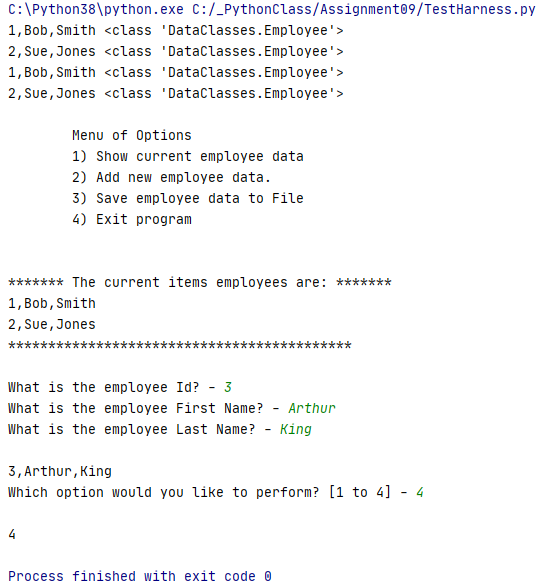


Figure 16: Test four successful

## Main Body

The main body of the script is similar to much of our earlier work: create a menu for the user to choose from then use a branching structure inside of a loop to do the work (see Figure 17).

*# Main Body of Script ---------------------------------------------------- #  
# TODO: Add Data Code to the Main body  
# Load data from file into a list of employee objects when script starts  
# Show user a menu of options  
# Get user's menu option choice  
 # Show user current data in the list of employee objects  
 # Let user add data to the list of employee objects  
 # let user save current data to file  
 # Let user exit program  
  
# Main Body of Script ---------------------------------------------------- #*

Figure 17: Main body pseudo-code

The branched code was copied directly from previous assignments, except that I added a loop to deal with incorrect entries as seen in Figure 18.

*# Get user's menu option choice*strChoice = Eio.input\_menu\_options(**"Which option would you like to perform? [1 to 4] - "**) *# message*while strChoice not in str(**"1, 2, 3, 4"**):  
 strChoice = Eio.input\_menu\_options(**"Entry should be 1, 2, 3, or 4 - "**) *# message*

Figure 18: Code to enforce choices

# Running the script

To see whether the code functioned as intended, I first ran it in PyCharm. The data from EmployeeData.txt is loaded and can be seen by entering option 1 as shown in Figure 19.

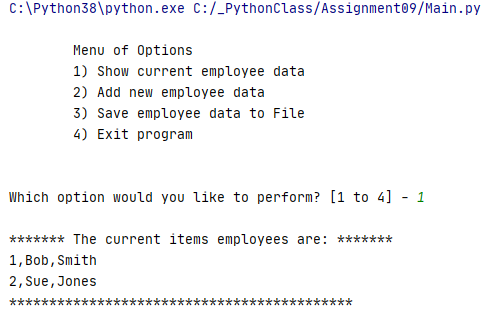


Figure 19: Show current data - PyCharm

I then added an employee using option 2 and showed the current list again (see Figure 20).

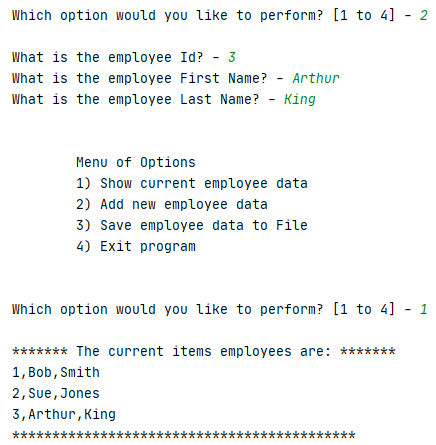


Figure 20: Adding an employee - PyCharm

Then I entered option 4 to exit the program without saving the data.

Finally, I ran the program from the command window. I added an employee and saved the data to the file as shown in Figure 21.

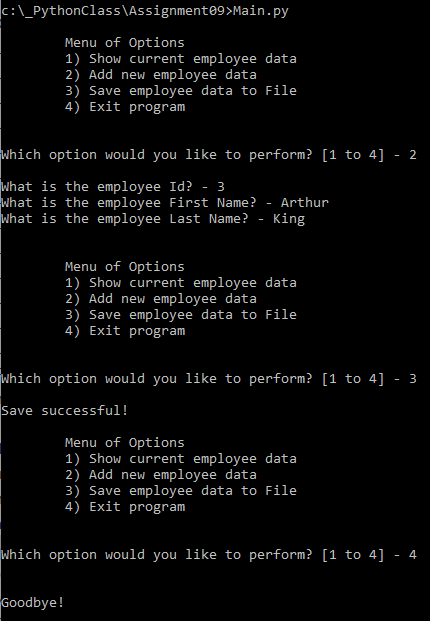


Figure 21: Run program – cmd

The new employee is saved to the file as shown in Figure 22.

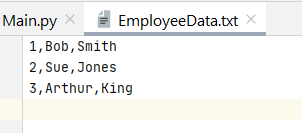


Figure 22: Employee data file

# Summary

Using the textbook, existing class codes, and the Module 9 documentation, I created the program using classes, objects and modules, and successfully ran it in PyCharm and the OS command window.