

Kyle Odland
June 14, 2020
Foundations of Programming: Python
Assignment 09

Linking Modules

Introduction

The ninth assignment for Foundations of Programming: Python was to examine provided DataClasses, ProcessingClasses, and IOClasses modules, then determine what the task of the Main module would be based on pseudo-code, then use the three provided modules to add functionality to the pseudo-code.

Coding

Using a test harness to test the three provided modules gave a good idea of the methods and the structure of the program. The provided pseudo-code for the Main module gave an outline of what task the script was meant to accomplish. Any existing employee data in a file would be read into the program as a list of Employee objects. The program would then give the user a list of choices of how to interact with that data - print existing data, add new employee data, save the data to a file, and exit the program. The set-up for this code was very similar to Assignment08, with a while loop that presents the options, and if statements for each user choice with the code necessary to perform that function.

To read the data from the file, I used the `read_data_from_file` method in the FileProcessor class in the ProcessingClasses module. To print the menu of options for the user and get the user's choice, I used the `print_menu_items` and `input_menu_options` methods from the EmployeeIO class in the IOClasses module. To use these modules I had to import them at the beginning of the Main module. Both of these modules themselves import the DataClasses module.

If the user chooses 1, to show current employee data, the code calls `Eio.print_current_list_items`, from the IOClasses module. If the user chooses 2, to add new employee data, the code creates a new object by calling `Eio.input_employee_data` from IOClasses, and appending that object to the list of employee objects. If the user chooses 3, to save to a file, the code calls `Fp.save_data_to_file`, from the ProcessingClasses module. Finally, if the user chooses 4 to exit, the code breaks out of the while loop and ends the program. Listing 1 shows the main body of the script that accomplishes these tasks.

```

# Load data from file into a list of employee objects when script starts
lstEmployeeData = Fp.read_data_from_file(strFileName)

while True:
    # Show user a menu of options
    Eio.print_menu_items()
    # Get user's menu option choice
    strChoice = Eio.input_menu_options() # get menu option
    # Show user current data in the list of employee objects
    if strChoice == "1":
        Eio.print_current_list_items(lstEmployeeData)
    # Let user add data to the list of employee objects
    elif strChoice == "2":
        objEmp = Eio.input_employee_data() # create new employee object
        lstEmployeeData.append(objEmp) # append to existing list of employee objects
    # let user save current data to file
    elif strChoice == "3":
        Fp.save_data_to_file(strFileName, lstEmployeeData)
        print("Data Saved")
    # Let user exit program
    elif strChoice == "4":
        print("Ending Program")
        break
    else:
        print("Make a choice from 1 to 4")
        continue

```

Listing 1: Main body of script for Main.py

One issue that I had that I had to debug was with the modules that were provided. When I tried to use the `Eio.print_current_list_items` method, I got back an attribute error that a row in my `lstEmployeeData` did not have an `employee_id` attribute. Digging into the provided modules, the `print_current_list_items` is expecting a list of objects, and then prints the `employee_id`, `first_name`, and `last_name` of each of those objects. In the `ProcessingClasses` module however, the `read_data_from_file` method is looking at each line in the file and turning it into a list of strings. This gets returned in my code as `lstEmployeeData`.

This list of strings getting used as a list of objects is what was causing the exception. In the `PC.read_data_from_file` method, the pseudo-code specifically says that the method is meant to return a list of object rows. In order for the `ProcessingClasses` module to create an employee object, I added code to import the `DataClasses` module. In the `PC.read_data_from_file` method, the script reads a line of string from the text file and I use those values to initialize the `DC.Employee` method to create an employee object. I append that object to a list, so the `read_data_from_file` method returns a list of employee objects as intended. I had to make a

similar change to the `PC.save_data_to_file` method, to take a list of objects and turn that into strings to write to the text file.

Running the Code

Figure 1 shows an example of the Assignment 09 Main.py code running in PyCharm, where the user adds a new employee object to the list.

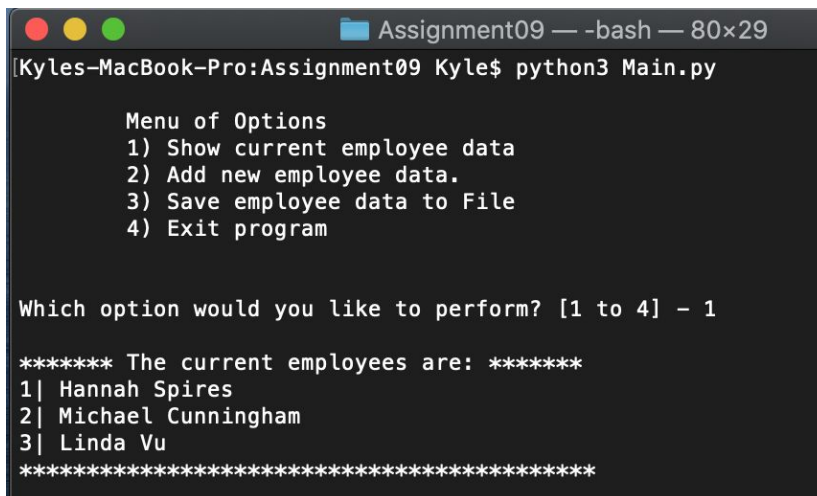
```
Menu of Options
1) Show current employee data
2) Add new employee data.
3) Save employee data to File
4) Exit program

Which option would you like to perform? [1 to 4] - 2

What is the employee Id? - 3
What is the employee First Name? - Linda
What is the employee Last Name? - Vu
```

Figure 1: Adding product object to the list in PyCharm

Figure 2 shows the program displaying the current employee data, while running the code in the macOS terminal.



```
Assignment09 — -bash — 80x29
[Kyle-MacBook-Pro:Assignment09 Kyle$ python3 Main.py

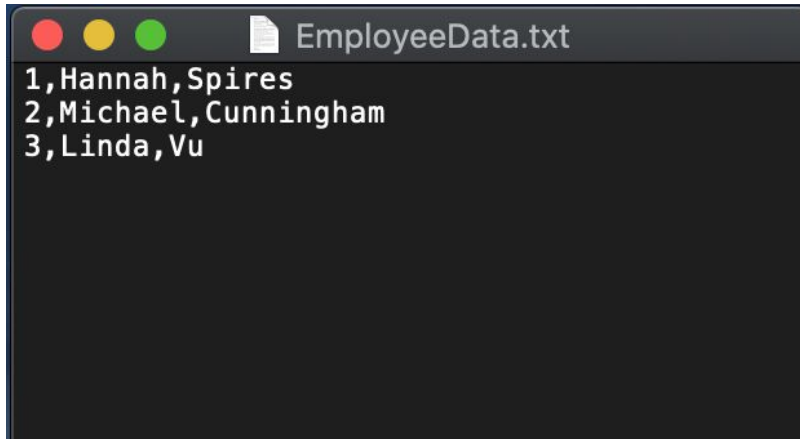
Menu of Options
1) Show current employee data
2) Add new employee data.
3) Save employee data to File
4) Exit program

Which option would you like to perform? [1 to 4] - 1

***** The current employees are: *****
1| Hannah Spires
2| Michael Cunningham
3| Linda Vu
*****
```

Figure 2: Displaying current employee data in macOS terminal

Running the code as above gives the output txt file, shown in Figure 3.



```
1,Hannah,Spires
2,Michael,Cunningham
3,Linda,Vu
```

Figure 3: Output text file with a list of employee data

Summary

This assignment was very similar to the program from Assignment08, with modifications to link modules together, instead of having all of the necessary classes contained in a single program. This allows the programmer to accomplish a separation of concerns, where each module is performing specific tasks to create data, process data, and present data.