Lyndsey Holmes

September 10, 2020

Intro to Programming (Python)

Assignment 09

Github Link:

Data Processing with Classes

# Introduction

A project can contain multiple modules that can be called within the same file or called from other files. In order to keep an organized project, it is best practice to have a main function that calls other classes and the classes’ methods. For this project the modules are split between file processing, data classes for setting up the objects and attributes, the input and output module (receiving data from the user), and the main function module. The main function primarily calls the modules. As shown in Figure 11, the main module is the most simplified. The rest of the functions contain the majority of the work. That keeps the code easy to use when it is broken in pieces.

## Starter Script

Initially provided was a starter script to include full modules and pseudo-code. The completed files are DataClasses.py shown in Figures 1-3, Data Classes, Part 1-3, IO Classes shown in Figures 6 and 7, IO Classes, Part 1-2, Test Harness shown in Figures 8-10, Test Harness, Part 1-3. Processing classes required updates to include the Database Processing script, shown in Figure 5: Processing Class, Part 2. The main module required the most work but included well defined pseudo-code.

### Data Classes

The data classes file’s purposed is to create the objects. The object is the person and the attributes include the first and last name and the employee id. The data classes file has two classes. The first class is person and it is inherited by the class employee. So the employee class contains person and employee id but the class person contains the first and last name of the person. Table 1 is a representation of how each class looks in memory.

|  |  |  |
| --- | --- | --- |
| Employee | Person | First Name |
| Last Name |
| Employee ID | |

Table 1: Representation of Classes in Memory

Figure 3 of Data Classes allows the data to be printed as a string rather than a location in memory. If those lines are commented, and the user attempts to print the name or employee id by calling the object, then Python will provide the product type and location in memory.

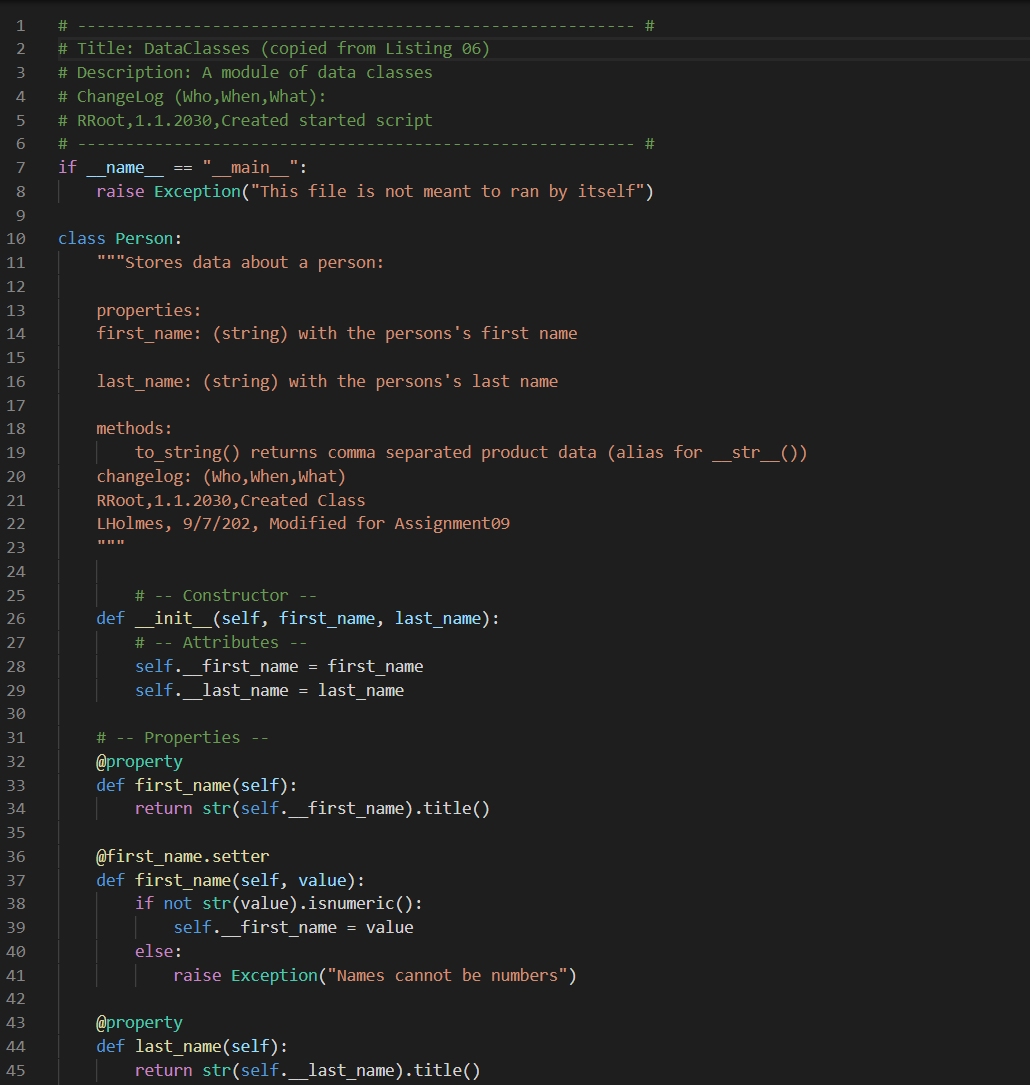


Figure 1: Data Classes, Part 1

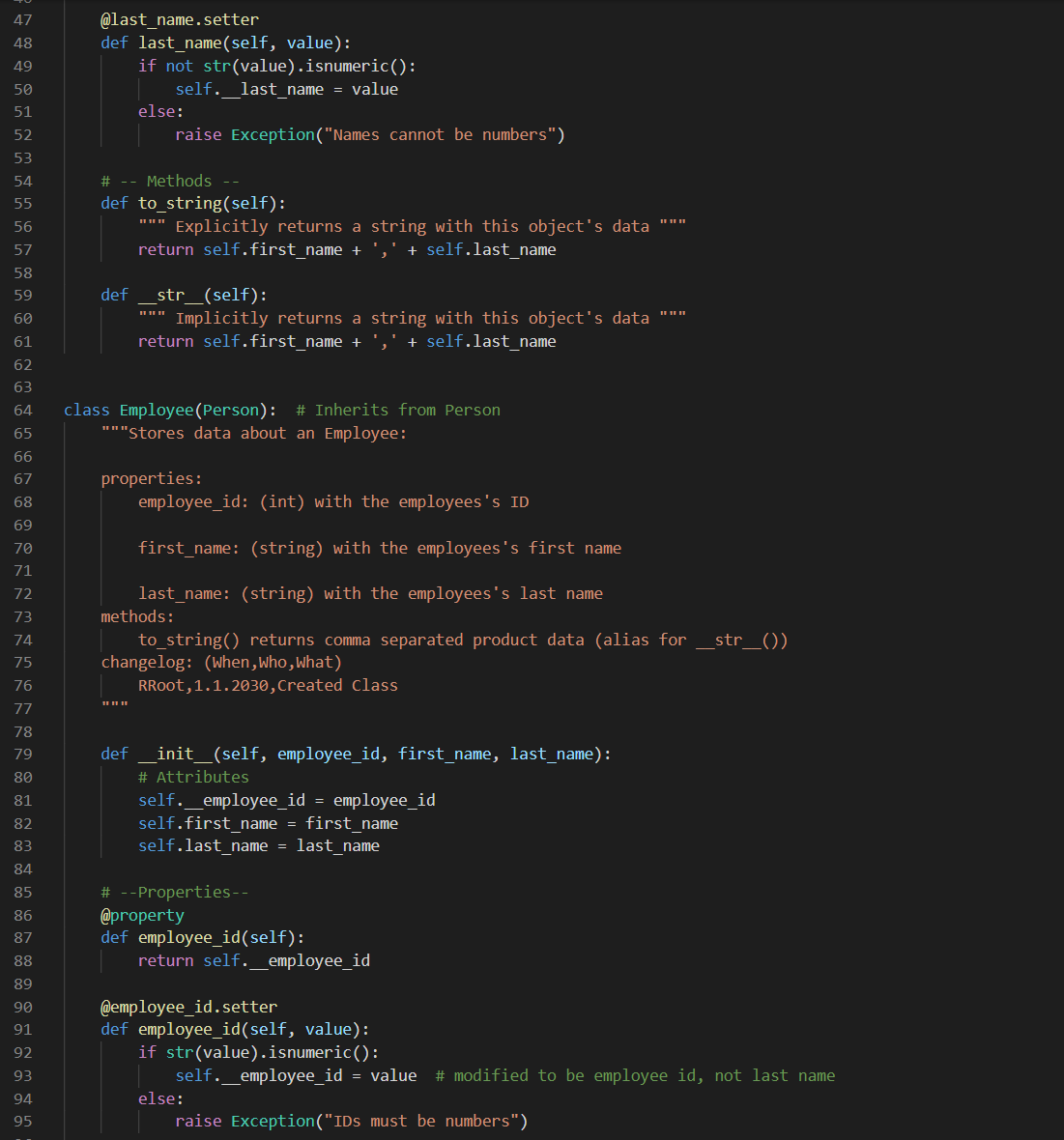


Figure 2: Data Classes, Part 2

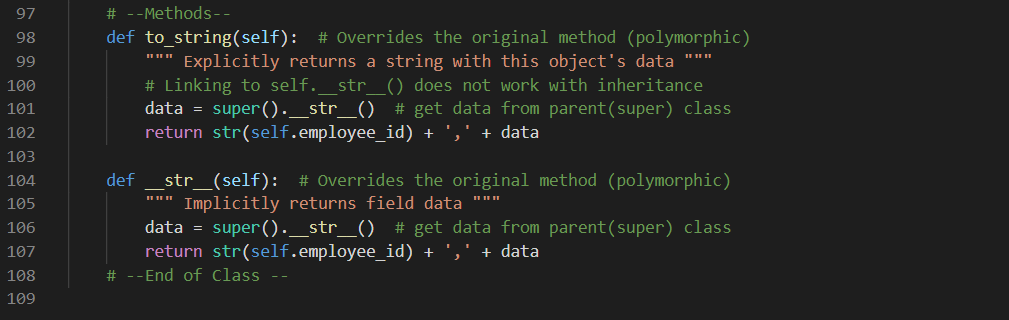


Figure 3: Data Classes, Part 3

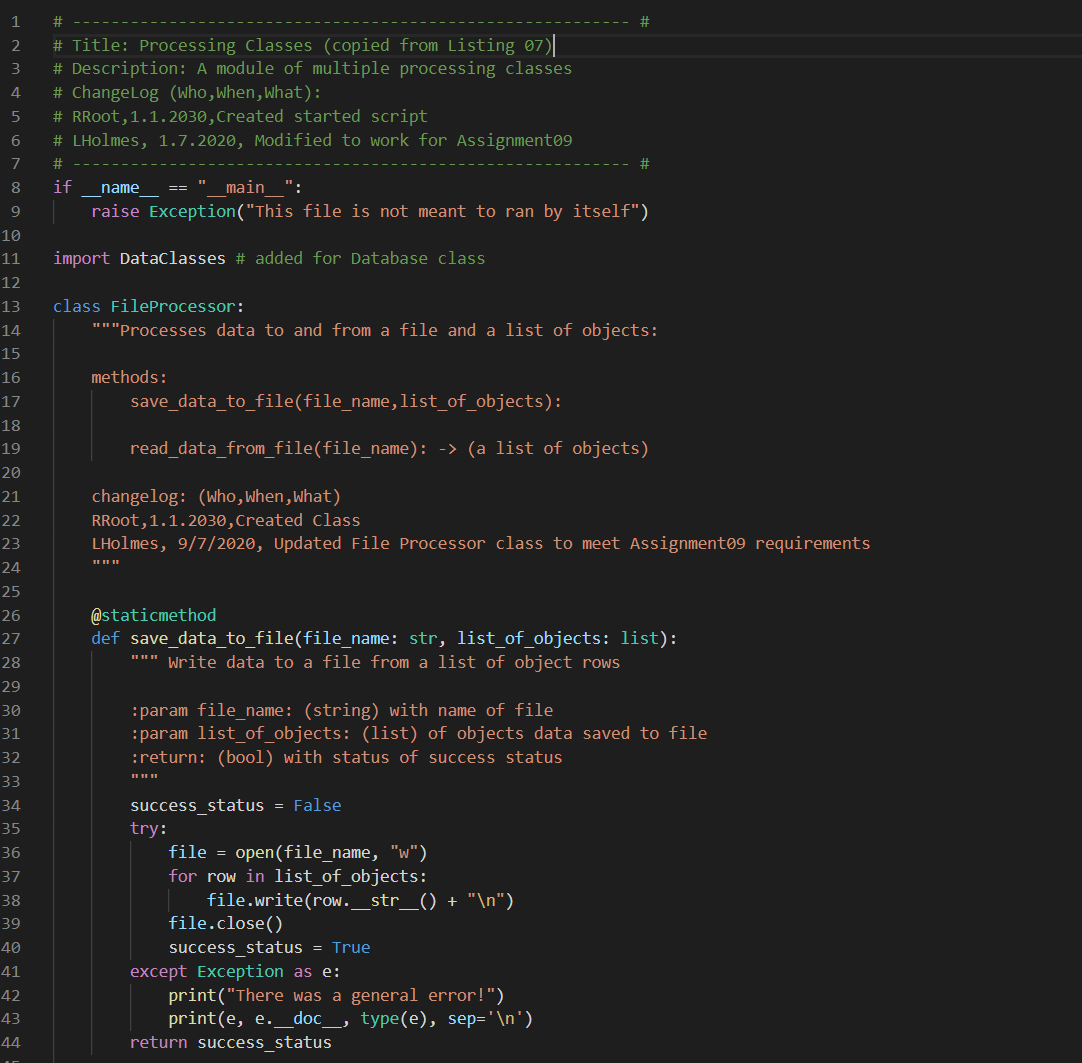


Figure 4: Processing Classes, Part 1

### Processing Class

The purpose of the processing class file is to read data from and save data to a file. Additionally, the Data Processor Class processes the data that is read in from the file into the objects. Figure 4 shows the The Data Processor Class calls the “read\_data\_from\_file.” Once the data is read from the file, the Data Processor class continues by stripping the comma, spaces and end lines from each row of the file. Then the data is applied to the object’s attributes. As shown in figure 5, the data is saved to the object employee, which is appended to a list accumulator called “list\_of\_objects.” The list of objects can be accessed by any classes’ method by calling the database method.

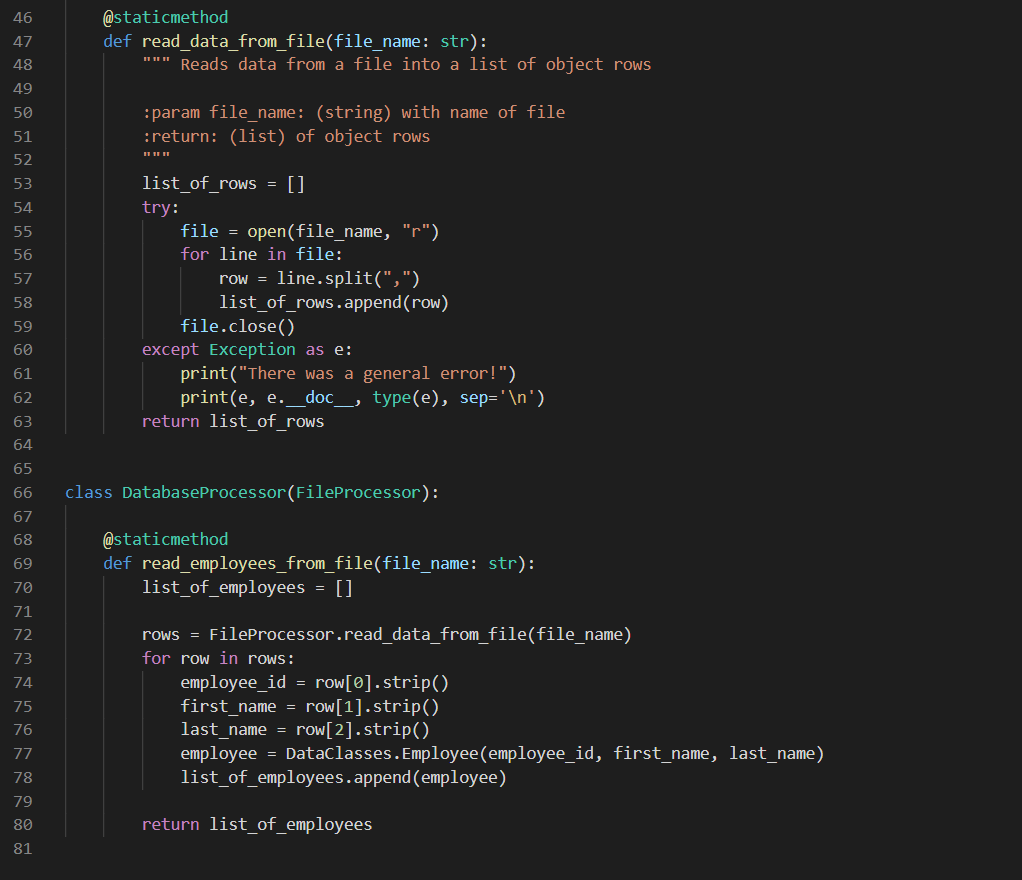


Figure 5: Processing Classes, Part 2

### IO Classes

As shown in figure 6 and 7, there is a separate module for the input/output functions. The script requires input from a user to provide details as to which function of the menu they would like to access. This module also includes the output data when the script first is run (such as the menu options). Two of the methods of IO Classes include try-except statements. Input menu options and input employee data both include try-except statements since they receive data from a user. Instead of crashing the script with an unclear exception error, the script throws an exception with more details as to what went wrong with the input. The method to print current list items requires some manipulation of the data so that the information is printed in a string of the data rather than the location in memory.

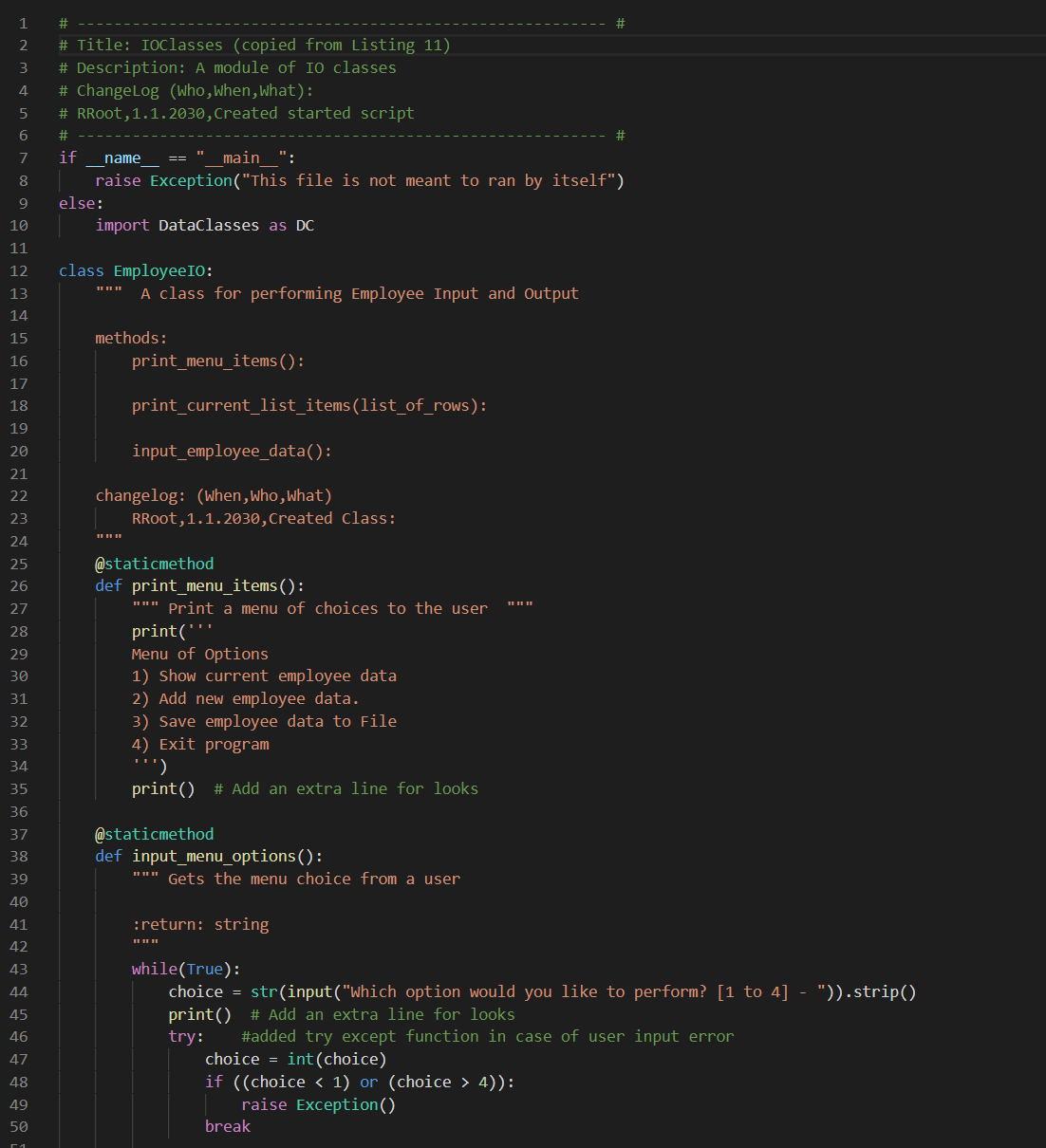


Figure 6: IO Classes, Part 1



Figure 7: IO Classes, Part 2

### Test Harness

Figures 8-10 shows that the other classes are tested. Test Harness tests instances of all the classes and tests all the functions and methods. It doesn’t need to test Main because it tests everything main uses. Test cases need to be checked so that it can be verified that a user will not be able to cause an error in the script. If not all possible options are tested, then the script has a bug.

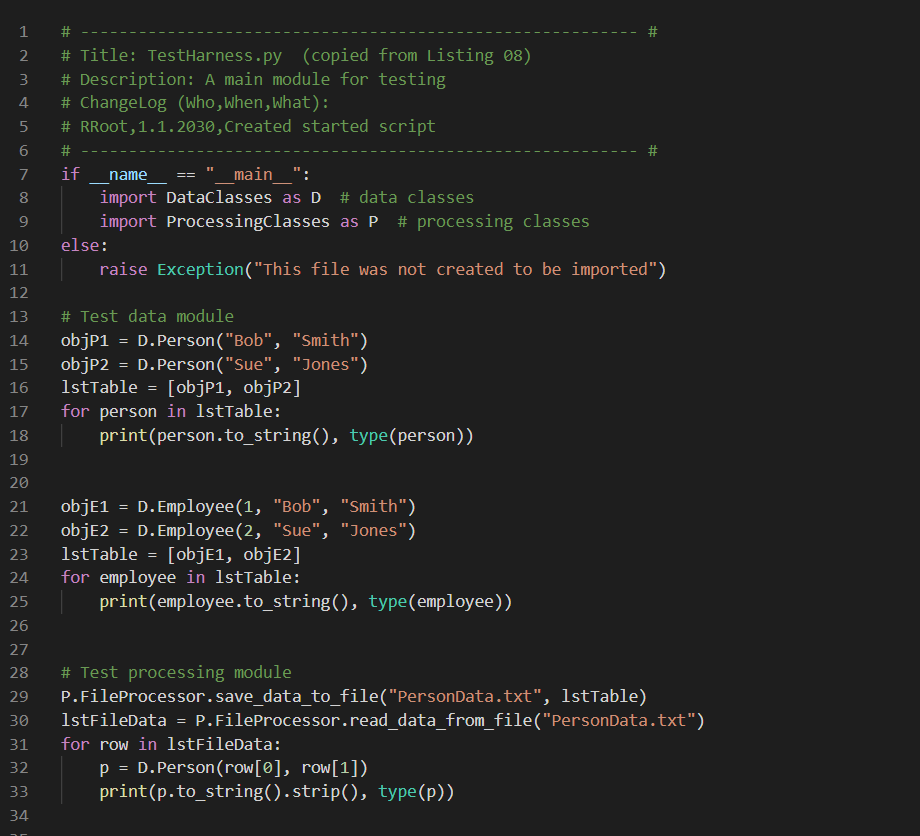


Figure 8: Test Harness, Part 1

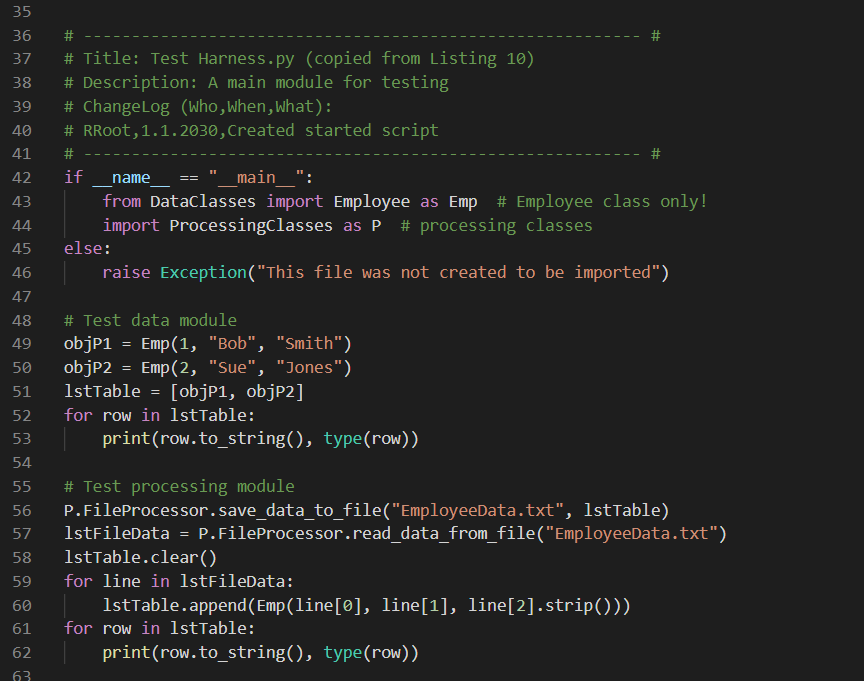
****

Figure 9: Test Harness, Part 2

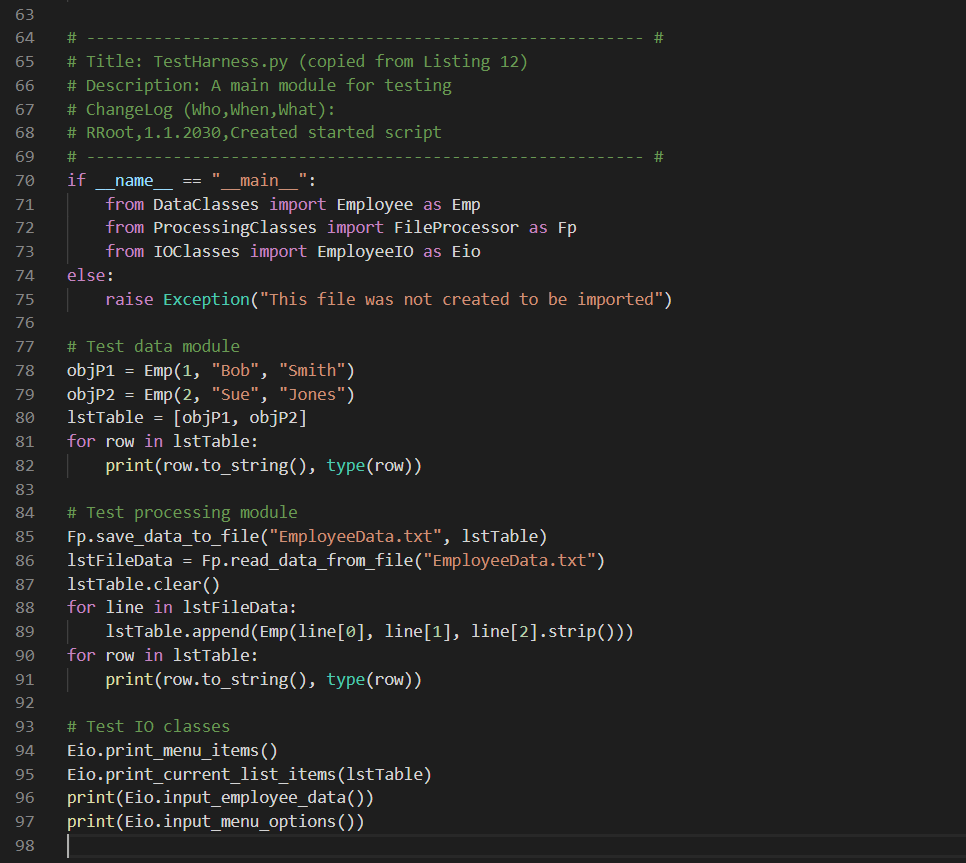
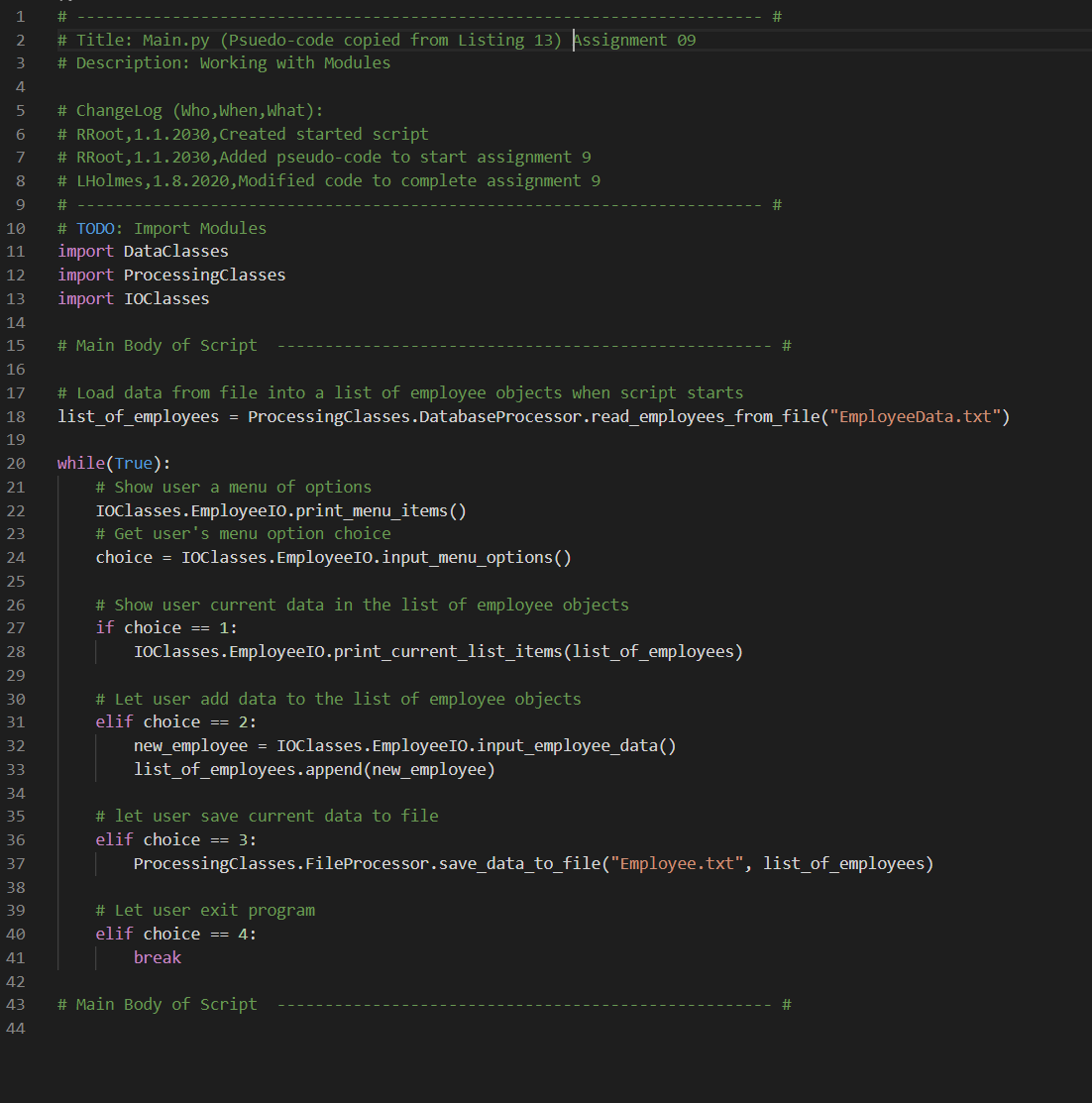


Figure 10: Test Harness, Part 3

### Main function

The main function is the glue that pieces all the methods of the classes together. Its purpose is to provide the architecture of the project. As shown in Figure 11, the start of the script calls the “DataClasses”, “ProcessingClasses” and “IOClasses” modules. The initial step is to call the database processing script from Processing classes. That step reads in the data and fills data into objects which is appended to a list. The next step is looping through the conversation with the user. When the user provides instruction, the input is tested by the “IOClasses” script and then returned to the main function. Once the user would like to save the data, the “ProcessingClasses” module is called again to save the data to a file. Generally, it is good practice to save the new data to a different file than from where the file is read. In this case however, the ‘read from’ and ‘save to’ file is the same file.

Figure 11: Main Module

## Running the script

The first item to appear when running the script is the menu item. It provides the user with the option to read the data in the file, add a new employee, save the data or exit the script, as shown in Figure 12. The main script continues to output the menu and question about option preference to the user because it is set to infinite loop until the user wishes to exit the script. Figure 12 shows what happens once a new employee was added. Figure 13 shows that the user wants to verify the new employee was added and save and exit. Figure 14 shows the text file after the new name was added.

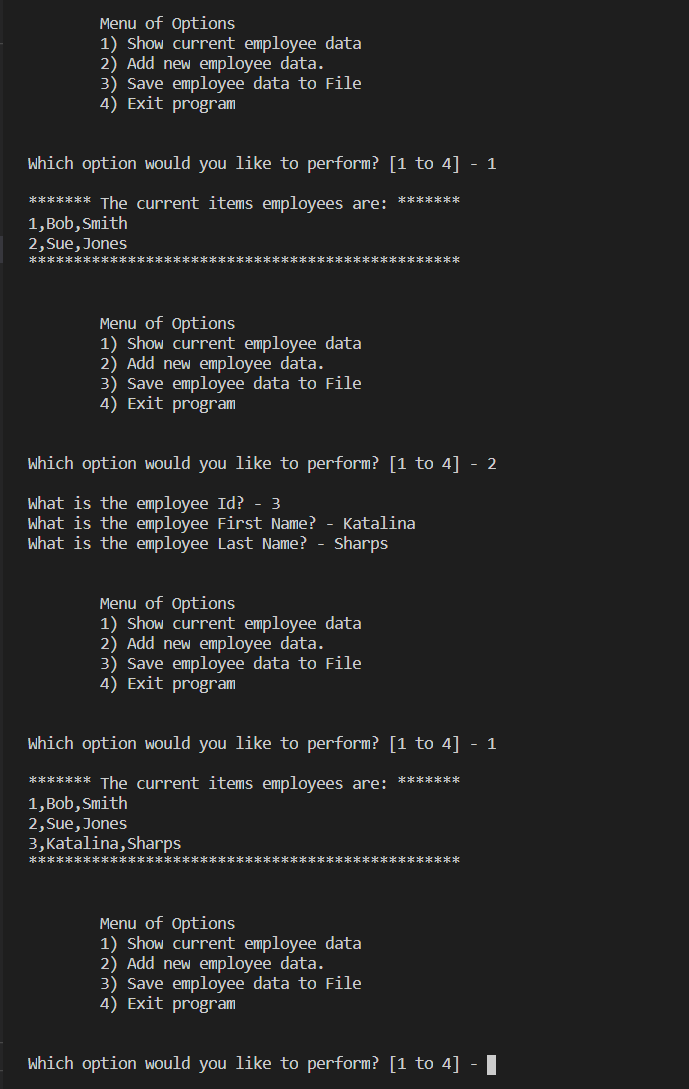


Figure 12: Running the Script, Part 1

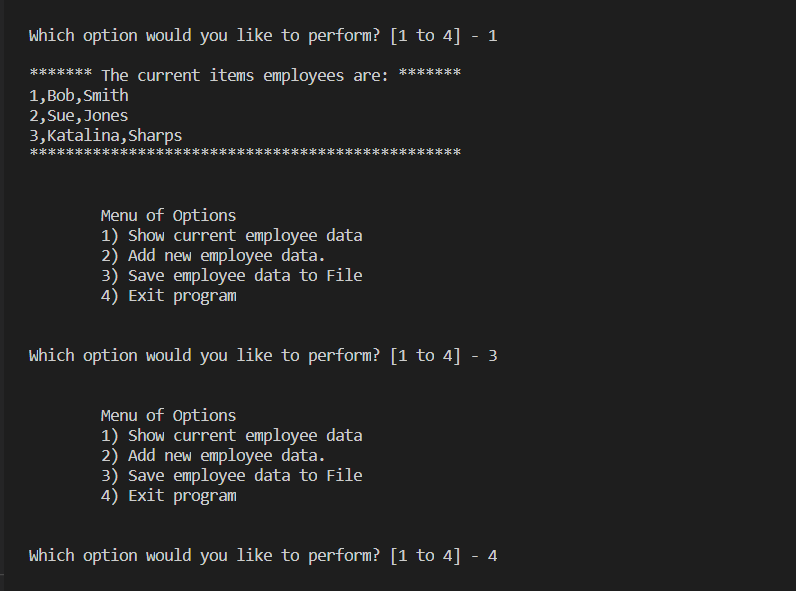


Figure 13: Running the Script, Part 2

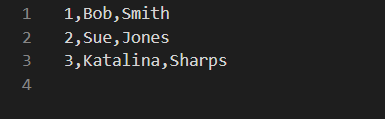


Figure 14: The Text File After Save

# Summary

In summary, breaking up the classes and methods in different modules is an organized way to separate the types (or purposes) of the methods. It would not make sense to have the input and output methods together with the data processing. If there eventually is a bug in the script, another developer would easily be able to find which script in the project to test. Also, classes can be inherited by other classes to break up methods into smaller projects. For example, an employee has a first name, last name, and employee id but the person doesn’t have to have an employee id. So it is best to split up the class person (with first name and last name) and employee class can inherit person and add an employee id.