Hyojin Kwak

November 23, 2020

Foundations of Programming

Assignment06

<https://github.com/hihojin/IntroToProg-Python-Mod06>

**Starter.Py with Function and Class**

**Introduction**

This week, we learned about how to define and call function along with class, which contains functions. As script is getting longer, I learned it is more useful to code with function and class so that the main body of codes look not messier. Starter.py is a modified script from the last week’s HomeInventory.py assignment, scripting with dictionaries and adding the row of data in dictionaries to the table of lists. Below paragraphs will show a step-by-step information of how I wrote codes to modify the script.

**Creating the Program**

First, I copied and pasted the codes of ‘Starter.py’ script from a file in canvas and pasted to ‘TextEdit.txt’ so that I can paste the codes right to PyCharm by keeping the original code format. I changed the changelog with my name, date, and the task in the script header as guided.

Of many of other important things to remember for this assignment it was critical to copy the right code in into each function and to correctly return the variables that contains the right values. As we have studied in class on returning value by reference, it was also important to keep in mind to use the name of global variables when calling functions in main body of codes rather to keep using the reference name.

In step 1 where to load any data when the program starts, class Processor.read data from file was used with returning value of list of row ==lstTable and file name == strFileName. The same code from the last time was used by opening the ‘TodoFile.txt’ file with read mode, split the task and priority values with comma then use the values in dictionary format then to append the dictionary of row into the lstTable.

Text

Description automatically generated

*Figure 1. Adding a new task with class IO and class Processor*

IO.input new task and priority() asks user to input the task and priority and processor.add data to list function then adds these in one dictionary of row and appended into the lstTable. lstTable and strStatus were return values and IO.input press to continue(strStatus) will show ‘Success’ if tasks were added properly into the lstTable.

Text

Description automatically generated

*Figure 2. Removing a task with class IO and class Processor*

IO.input task to remove() asks the user which task they would like to remove and return the inputted value in a variable called removing. With return value of lstTable and strStatus from processor.remove data from list function, and by applying ‘removing’ value into the function it successfully removes the specific task that the user wanted to remove.

Text

Description automatically generated

*Figure 3. Saving data to file with class IO and class Processor*

Option 3 saves task and priority to the ‘ToDoFile.txt’ by asking and giving user the option whether to save the data or not. If the user writes ‘y’, the program indicates that the user wants to save data to the text file, runs the code in processor.write data to file, which has the code below:

Graphical user interface, text, application, website

Description automatically generated

*Figure 4. Saving data to file with class Processor*

Option 4 is to reload data from the text file by asking if the user wants to reload the file or not. If the user chooses to reload by putting in ‘y’, code in processor.read data from file runs and it brings the dictionary rows of task and priority values.

Text

Description automatically generated

*Figure 5. Reading data from text file with class Processor*

The last option is to exit the program and the codes are simply to use print(“Goodbye”) with break.

**Getting the Output**

Graphical user interface, text, application

Description automatically generated

*Figure 6. Data saved in ‘ToDoList.txt’*

*Text

Description automatically generated*

Text

Description automatically generated

Text

Description automatically generated

*Figure 7,8,9. Output in PyCharm*

Text

Description automatically generated

*Figure 10. Output in the command shell*

As is shown above, the program ran smoothly this time and the main body of codes looks less messy. However I still get an error message when text file is not created before running the script. Next time, I will try to put try-except code for reading data from the text file section so that the program could run without having the same error.

**Summary**

It was confusing at the first time trying to organize codes in each class and function, but in the end, it was easier to read codes and to troubleshoot errors. I think I need to practice more to get familiarize utilizing classes and function. Learning these two concepts was very helpful and will continue using them.