Henry Chisolm

August 31, 2021

Document Write-up:

Assignment 06

**Modify an existing script by adding more functions to organize the code**

**Introduction:**

The purpose of this assignment is to take an existing script, which in this case currently loads data from a file into a Python List Dictionary object and add more functions to organize the code and run the program that compiles a “To do list” in a text file. In this case the added functions are both performing processing and input/output functions. Specifically, processing functions of adding data to lists, removing data from lists and writing data to a file are being asked to be coded. Also, I/O or input out put functions of inputting a new task and priority and inputting a task to be removed are being asked to be coded. The outcome is a script that can add/edit/ delete and store data to a “ToDoList.txt”.

**Scope of Assignment:**

The assignment asks us to add additional code in the form functions in preestablished classes to run a program that edits, compiles and stores a to do list. The script provided already has programmed classes Programming and IO (input/output) with functions fully fleshed out and then those needing code. Where code needs to be added in this script is (1) in the processing class, (2) in the IO class and finally (3) in the main body of the script. This write up will analyze the individual functions that need code added and the script necessary to run the program. All existing code will not be analyzed in this assignment.

**Class: Processor:** precoded with a read data from file function

**(1) add\_data\_to\_list:** (Processor.add\_data\_to\_list)

In this script we already had the function named which would add data to list with the values or parameters of task, priority and list of rows.

**Graphical user interface, text, application, Word

Description automatically generated(added script)**

In this case we have the parameter list\_of\_rows that we set equal to a dictionary entry. The dictionary entry is organized with the Keys of tasks and priority or also the way that we want the data organized in our to do list. We then open a lstTable, which was a preloaded variable, and .append or allow for this variable to be added to from in the manner of list\_of\_rows. In this case using the information stored from task and priority.

**(2) remove\_data\_from\_list** (Processor.remove\_data\_from\_list)

In this case the function remove\_data\_from\_list has the parameters of task and list\_of\_rows. We set up a statement for the rows in the lstTable that was established as an initialized variable in the precoded script. So with this statement if there is a task that has been keyed from a task already in the ToDoList.txt it’s goal will be to remove that row from our Table and the .txt file. We accomplish this by employing the .remove command attached to our lstTable or the value that we are storing our dictionary entries. Finally the function prints the task that we will enter along with a message of “has been removed from the list”.

**Graphical user interface, text

Description automatically generated with medium confidence(added script)**

**(3) write\_data\_to\_file** (Processor.write\_data\_to\_file)

In this case wea re asking the function write\_data\_to\_file to do as the name suggests and write the data that we enter to a file which we have preciously opened. In this case we accomplish this by setting a variable file that opens a file with the “w” or write function. We then use a for argument with a for rows in our predefined variable of list\_of\_rows which contains our dictionary entries.

Graphical user interface, text, application, website

Description automatically generated**(added script)**

We then use the command .write to our variable file to write what is entered later to a table. In this case the set up gives us a dictionary entry with rows for tasks and priority. Finally we use the command file.close() to close out the file after this command.

**Class: IO** The purpose of this class is to have functions that take input from the user and can be later utilized with other functions to returns some kind of output.

**(4) input\_new\_task\_and\_priority** (IO.input\_new\_task\_and\_priority)

As the name suggests this function collects input by the user about new tasks and their priority for the “ToDoList.txt”. This was an easier function to program in that we set a task and a priority to an input that then asked the question ”Enter a task” and subsequently “ enter priority”. We then run the function with the return task and priority.

**(5) input\_task\_to\_remove** (IO.input\_task\_to\_remove)

As the name of this function suggests this functions takes the input of a task that we enter and then removes it from our to do list. It accomplishes asking for input in the statement “enter task to remove”. This is done in the classic function structure where we define task as the input that we want to receive and then we use the command return task. To run what we have just defined.

**Problems that I had in this section of the code (aka the main body of the script):**

I had to add the script of:

A picture containing text

Description automatically generated

To load the “ToDoList.txt” file. The objFile had already been initialized and setting it to open strFileName (which had been set to open a “ToDoList.txt” file or the list that we had been making. The “a” appends the list. This was the only way I could get the menu to run and the list to be open in this program which is code outside of what it appears that the assignment wanted us to add.

**Main Body of Script:** The main script is already programmed to load data from out “ToDoList.txt and display a Menu of choices to user. We are asked to provide the code through the fnctions we have programmed to:

(6) 1) Add a new Task: If Menu option 1 is chosen. It is preloaded with an if statement. From that we take the variables task, priority and make them equal to the IO class function IO.input\_new\_task\_and\_priority. This will run our function which takes input from the user in two separate sections of task then priority. We also define the list\_of\_rows with our dictionary entries and set ups with the keys of task and priority.

Text

Description automatically generated

We then run the command Processor.add\_data\_to\_list with the parameters of task, priority and list\_of\_rows. This runs the function that will add data to the list

Graphical user interface, text, application

Description automatically generated(image of our script running)

(7) 2) Remove an existing Task: If Menu option 2 is chosen. In this case we set a value of task to the command IO.input\_task\_to\_remove followed by the command Processor.remove\_data\_from\_list. Which in this case asks from data from the user or rater inout and looks for that inout to remove, and that input is utilized and run in the command. Of the Processor class. Honestly, this script I ran, I don’t think ran super successfully.

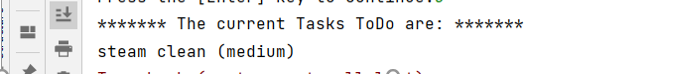
Graphical user interface, text

Description automatically generated

So then the command of IO.input is asking for an enter to continue.

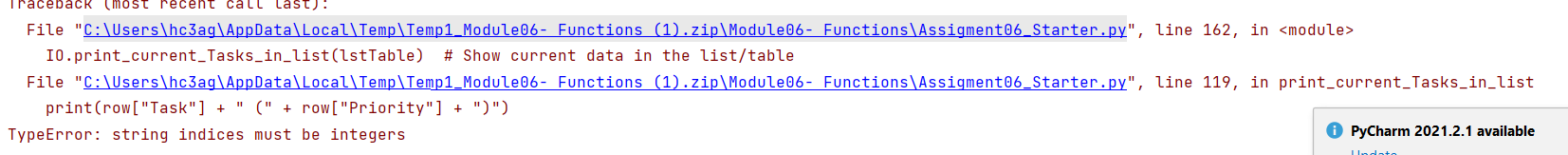
(8) 3) Save Data to File: In this case if menu option 3 is chosen, a preloaded script asking for a y input is run followed by the code Processor.write\_data\_from\_file with the parameters of strFileName and lstTable. The StrFileName is the carriable that opens our “ToDoList.txt” and lstTable is how we store our dictionary entry.

(9) 4) Reload Data from File: In this case we are just employing the Processir.read\_data\_from\_file with the parameters of filename and list\_of\_names. What this serves to do is reload what has already been entered into the program.



This involves preloaded tasks.

**Problems I ran into:** When executing the script and running the script and running menu option 5 Exit the program I ran into:



Errors from the precoded bits. I wasn’t quite sure how to fix this message.

**Summary:**

In this Assignment We used a preexisting script and adapted functions to finish writing a script that would start a “ToDoList.txt” file which served as a to do list that would then accept user input in the form of task and then priority. Store that input where it could be recalled, edited, appended and viewed.