Tiffany Taylor

May 12, 2020

Memory List Manipulation

Assignment05

Memory List Manipulation

# Introduction

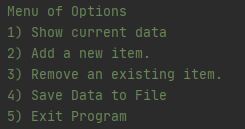
Assignment05 through “The Foundation of Programming” course with the University of Washington asks us to add code to an existing script that will define the input and output of a menu of options for manipulating a list table that will be stored in memory until the information is written to a text file.

# Read the Existing Script

This exercise started with us needing to utilize a start script from the teacher. Before attempting to write the needed code, I read through each line of both comments and code of the existing script to make sense and build a framework in my mind of what I was working with and what already exists. This helped me to understand what still needs to be added using my own analytic thinking and compare it to what was written and being asked.

# The End Result

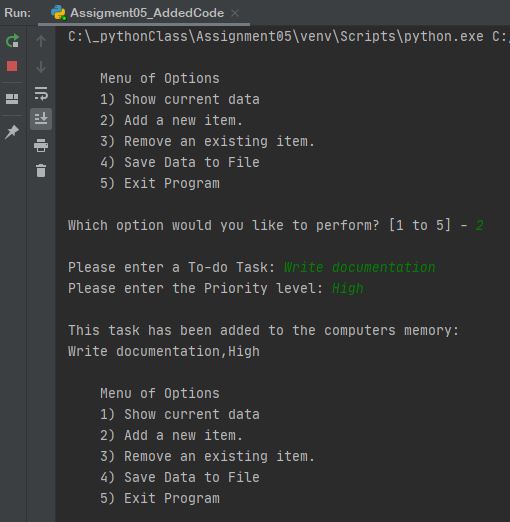
The existing code gave us a pre-designed menu of options for manipulating data in a to-do list table with dictionary rows. This data, which is stored in the memory, could be controlled via the menu of options choice (including writing it to an external file) See Figure 1.



***Figure 1: Pre-designed Menu of Options***

First, we’ll look at some screenshots of the end result we’re trying to achieve to better visualize the choices we make when writing the code. We can look over the code used in the next section.

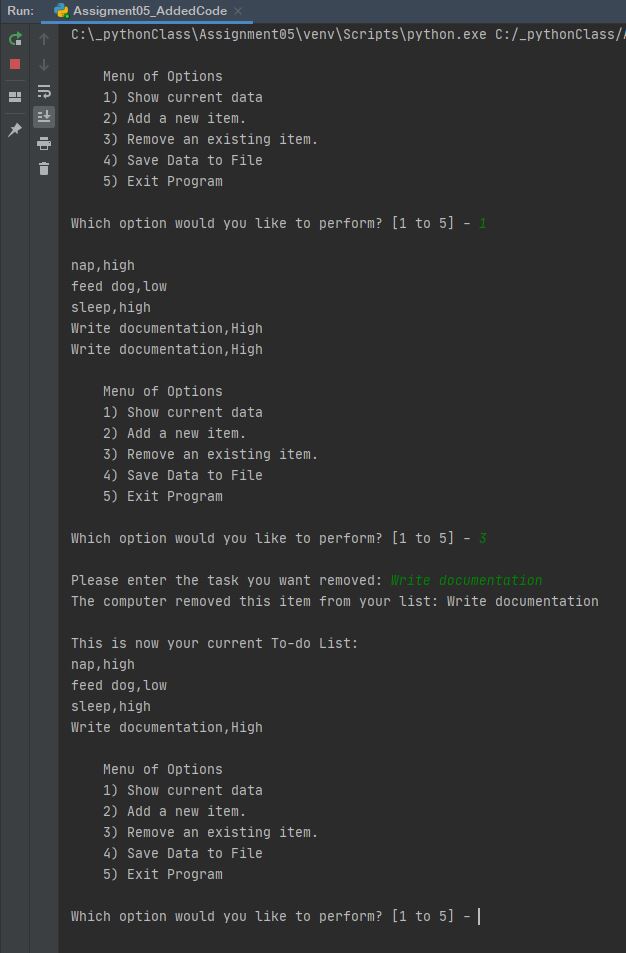
The first option in the menu will not show you anything at the very beginning, so we will start by giving you examples of the result starting with option 2 from the menu: “Add a new item”. When adding an item to the list table (which will be stored in memory, not the file), you want to be sure to collect the task and the priority level from the user since the existing code is asking for a dictionary row that will have two keys: ‘Task’ and ‘Priority’. The users input will become the value of these keys as we create the rows for our table of the to-do list. Figure 2 below will show us a desired result when the user selects “2” from the menu:



***Figure 2: Adding a task to the lstTable variable***

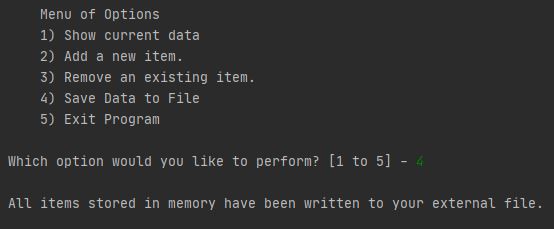
You can see we have 2 questions for the user to collect their data and then we let them know it’s been stored in the computers memory (which is difference from being written on the file – option 4).

Now we can show you the result of what to expect when choosing option 1 (showing the current data stored in the lstTable variable) and now that something exists in the table we can also utilize option 3 (removing something from the table). Figure 3 shows examples of the desired result:



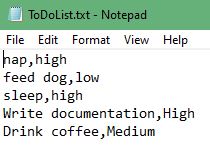
***Figure 3: Utilizing Option 1 and Option 3***

And the last option for manipulating the data is taking the stored data from our lstTable variable and writing it directly to an external file; in this case a text file. In Figure 4 you see the user choosing option 4 and what the system reports back to the user.



***Figure 4: Write data to the text file***

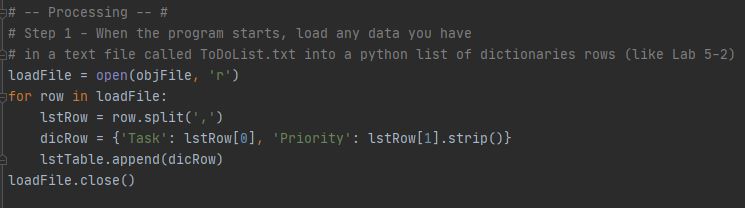
In Figure 5 you can see what the text file now looks like after the user has added several things (option 2) and now written them (option 4) to the text file.



***Figure 5: Text file with data that’s been written via the program***

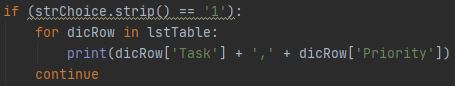
# Create the Logic

Now that you have a better idea of the result you’re looking for, let’s work backwards to specific the code needed to reach those desired outcomes. The first task in the document was asking us to write code that will process when the program starts to load any data on the text file and make them dictionary rows stored in a list, ready for possible data manipulation. Figure 6 shows you how you can open the file, and then read through each row of data in the file and store the rows into a dictionary row (introduction the keys) and then store those new dictionary rows into a list array that will be known as our table of data stored in memory while running the program. This would yield no information the first time the program is ran as the text document would be blank. But in future runs, there may be data on the text file that will now be extrapolated to be worked with.



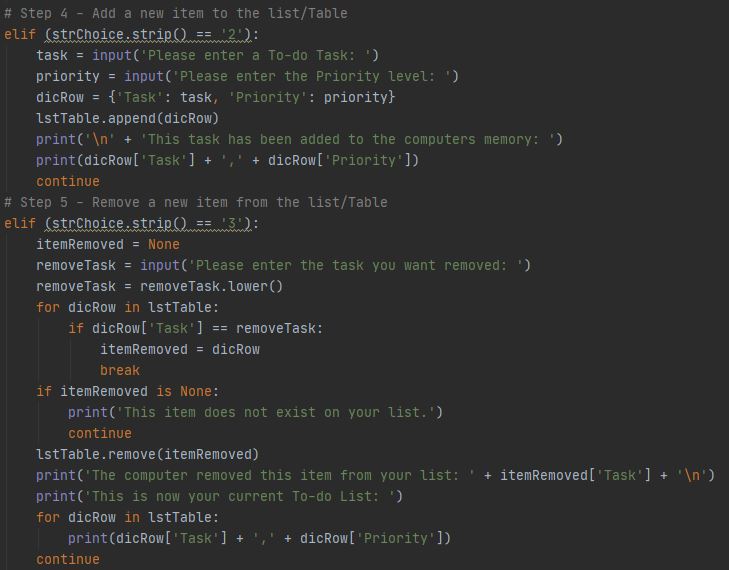
***Figure 6: Processing the code step***

Now we write the code for what happens with each option selection. Menu option 1 asks us to show the current data (in the list table, stored in memory, at that moment). Figure 7 shows us the loop we use to cycle through and present the task value and priority value of each dictionary row that’s currently stored in the table (the result would yield something like the first half of Figure 3.



***Figure 7: Option 1 Code***

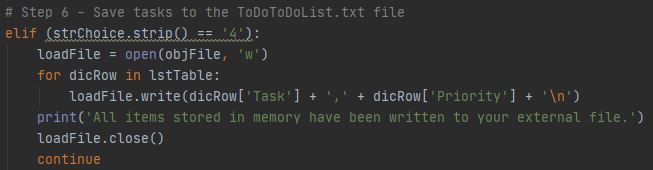
Menu options 2 and 3 were the most difficult code to write. They involve the most manipulation and input/output commands, leaving room for several different kinds of errors. Option 2 is about adding an item to the list and Option 3 is about choosing an item to remove from the stored list. In Figure 8 you’ll see the code for both in one screenshot to show how different the code can be despite it seeming like you might do something similar when simply “add this task” vs. “remove this task”. You’ll see removing a task involves a lot more error catching code to make it work smoothly; whereas adding something will simply just be getting the data and appending it.



***Figure 8: Add and Remove tasks on the lstTable variable***

Let’s look a little closer at the code for Option 3 (above in Figure 8). Here we created a new local variable to work with in various places as the code runs for this option – itemRemoved. We want to be able to reference the item removed in other places of our code. First, we store the users input of what they want to remove in a variable removeTask, we then have to make this lower case because if someone has “sleep” on the list, but typed in “Sleep” when asked what they want to delete, it will tell them (per our code) that their item doesn’t exist since Python is case sensitive. Then we need to look through each row in the table and see if any match the value of removeTask from the user. If not, we let them know. If so, we use the .remove() function to remove the item from the list table and then let the user know what was removed to avoid any confusion. Then, as a nice gesture, I added code to let the user know what the full list now currently looks like after the removal of the item. You can see the result reflected up in Figure 3.

We saw how to write data to an external file in the last documentation document so we won’t go in-depth, but Figure 9 will illustrate the code used when the user selects Option 4. You open the file in write mode, and then loop through each row of data in the table and write the values for keys in each row. Let the user know it’s been done with a print() function and then close the file to avoid any errors.

\

***Figure 9: Write to the external file***

# Summary

In summary, it’s important to know there is a difference in adding data to memory (accessed only while current run of the program is running) and adding data to a file (saved for access at a later date in a separate application). Just because you append or remove things from a list you want on a text document, does not mean the text document is being added to or things removed from. All the changes you make will only be stored, once you connect to the file and .write() to the file. Also, it’s important to remember that common errors can pop up when manipulating lists (like index location, and upper vs. lower case text) etc. that you want to build code in to handle.