Keith Burdette

February 7th, 2020

Foundations of Programming: Python

Assignment 03

**Using Python Dictionaries**

**INTRODUCTION**

This week, the assignment was to build a ToDo list program that reads a text file with task/priority pairs into memory. The user can then display the data, add to the list or knock items off the list. The user can also save changes back to the original text file and exit the program.

**TODO LIST PROGRAM**

To start, we were given a template with the base program variables and navigation defined with steps laid out to get the program to work. Step 1 was to read data in from an existing ToDo list file. Starting out, there is no ToDoList.txt file. If I try to open the file with python, I would get an error saying “File Not Found.” I can use the os library and the path.exists() function to check whether or not my ToDoList,txt is present. If it is, I create a ReaderFileObj to open the file then use a for loop to process each row in that file. I know the file will have comma separated pairs so the first action I will complete on a row will be to split the pair into a list by the comma. Next I will create a dictionary row with a “task” and a “priority” from our split pair. Outside the for loop, I close the ReaderFileObj. This code block is shown in Figure 1.

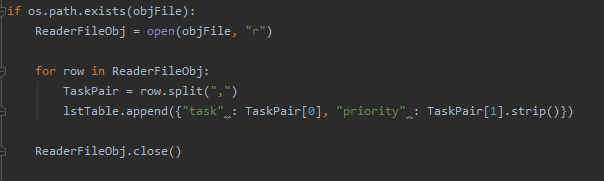


Figure 1 – Read File if Exists

Building the action menu was step 2 which was included with the template. Step 3 was displaying the data from your table. This is accomplished with a for loop. For each row in the table, read the “task” and “priority” items from the dictionary and print them out in an aesthetically pleasing fashion. I separated mine with dash. This loop is shown in figure 2.

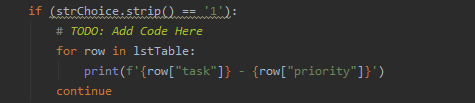


Figure 2 – Print All Items From To Do List

For step 4, it asked to add a task to the list. This required two inputs from the user: the name of the task and its priority. Then I append this pair to my existing list. This is shown in figure 3.

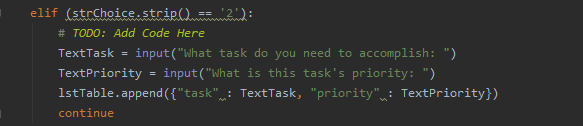


Figure 3 – Ask User For Items to add to List

For step 5, the objective was to remove a completed task from the list. This seemed reasonably simple; take a user input and compare it to all the existing tasks in your list. If you find a match, remove it from the list and print a statement saying the task has been removed. I set a FindFlag that I initially set to false but flip to true once the item is found in the list. If the item is not in the list, print out a statement telling the user that. This is shown in figure 4.

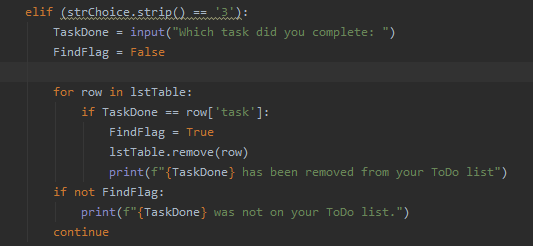


Figure 4 – Remove an item from list

Step 6 asked us to save the task list to the ToDoList.txt file. I opened a new WriteFileObj in write mode so that my file will be overwritten with the changes that I made to the list in memory. This was much simpler than try to parse the initial file again and make changes within it. I used a for loop to go through each row and print each task/priority pair separated by a comma with a carriage return at the end. After the for loop, I close my WriteFileObj and that completes step 6. This is show in figure 5.’

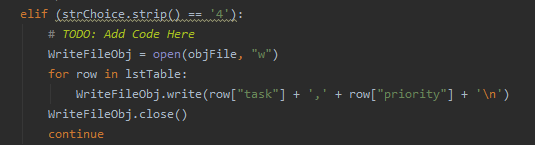


Figure 5 – Write To Do List from Memory to file

Step 7 was to exit the program. With the break statement already present, I didn’t actually need to add any code. I thought it would be courteous to remind the user that any changes made will only be saved if the file got saved. Many programs offer one last would you like to save changes menu but the assignment didn’t explicitly ask for that. I could have just repeated the same save code on exit to make sure any changes were captured.

**TESTING THE PROGRAM**

When first testing the program, I had no data or text file to read. I added a case to my load file code that prints out a statement saying that you have nothing to do if there is no file. The initial run of the file is shown in figure 6.

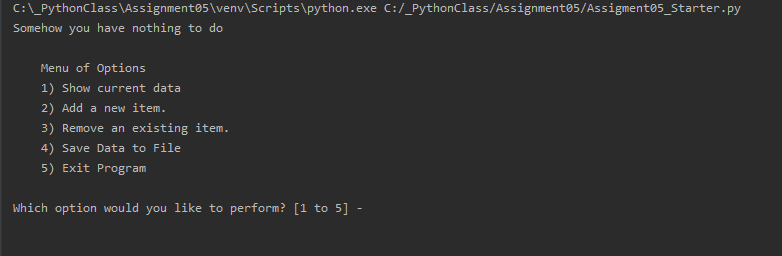


Figure 6 – Program Output on Startup if no file

I decided a similar statement this was also needed in the print out data option too. I test the length of my list and if it is zero, print out the same message. Now onto testing. I’ll add an item in figure 7. Remove it in figure 8 then show my no item message in figure 9.

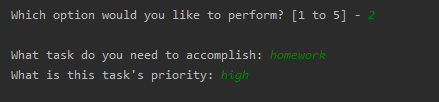


Figure 7 – Adding an item to To Do List

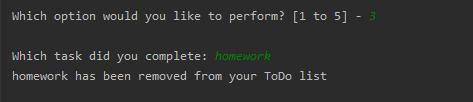


Figure 8 – Removing Item from To Do List

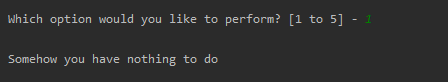


Figure 9 – Print out empty To Do List

I’m clearly not done with my homework yet so I’ll add that back along with several other things I need to do. Then, I’ll save it to a text file which I show in figure 10.

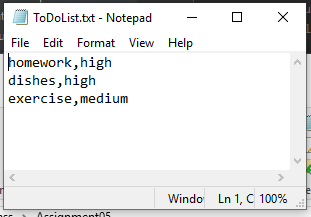


Figure 10 – Text File Output

I exited the program with option 5 and then switched to the command terminal. I’ll first do a print task list to show that option of the program but also show that the data loads from a text file properly, This is shown in figure 11.

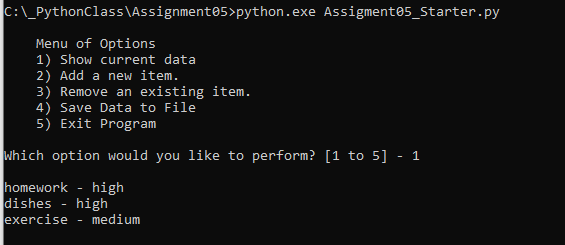


Figure 11 – Command Line data load and print task list

I decided to do some more testing of my code; what happens if I add the same item twice? Since I need to eat often I added it to the list twice then tried to remove it. This is shown in figure 12.

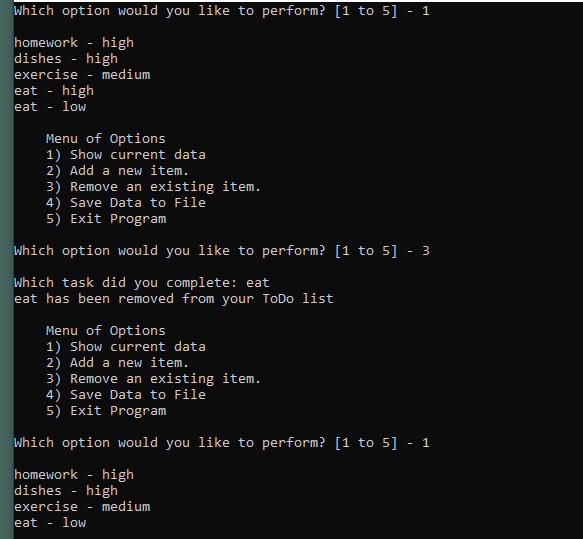


Figure 12 – Add and Remove a double entry – it doesn’t work!

As shown in the figure, there is something wrong with my remove code. I decided to add some print statements to help me figure out what was happening and showed that in figure 13.

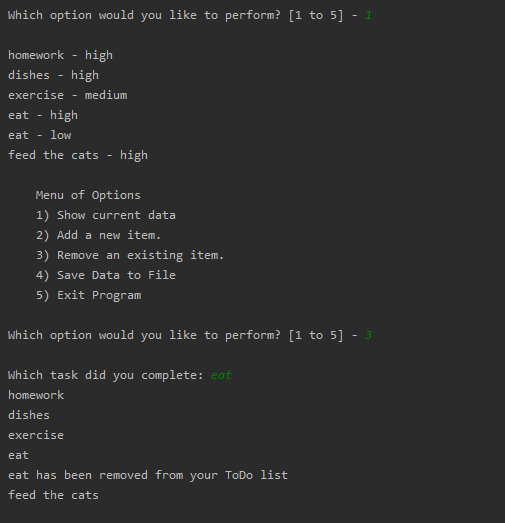


Figure 13 – Print Step by Step of Remove Sequence

It appears my second eat instance is being skipped as I go through the for loop, but why? I added a indexed print out of every item of the list as I go through the for loop. This new code is shown in figure 14. The result of running this code is in figure 15.

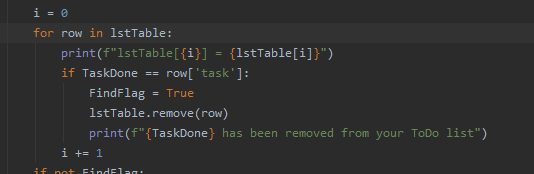


Figure 14 – Code of Remove section showing each line getting printed

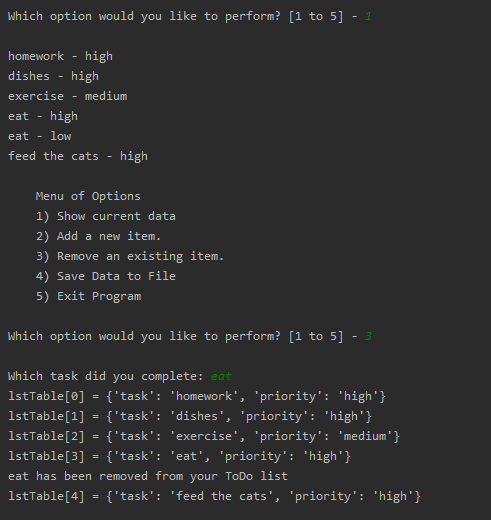


Figure 15 – Result of remove code, second eat gets skipped!

Eat is initially the 4th and 5th item in the list. But once the first instance of eat gets removed, there are only 5 elements left from my initial 6 element list. The second instance of eat gets moved to the 4th position thus it gets skipped in the for loop and feed the cats becomes the 5th and final element of the list. Removing an item in the list while iterating through it will not work. I saved as and set out with a new file to fix it. Figure 16 shows this.

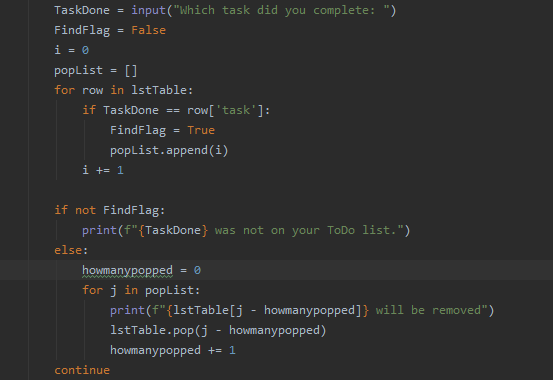


Figure 16 – Revised Remove code with pop function

Since I can’t remove items while I’m iterating, I decided to create a list of the index numbers of the elements I want to remove. Instead of using the remove function, I can use the pop function to remove items by the index number. Of course, we will run into the same issue where the index of element changes as items get removed. If we keep track of how many items we’ve removed, we can use it to adjust the index we need to pop. Figure 17 shows this code removing my double eat example.

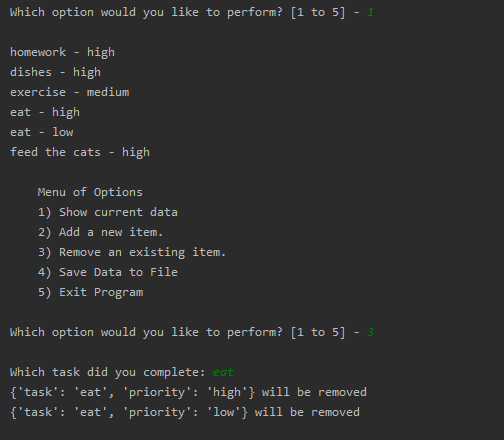


Figure 17 – Double item getting properly removed by code

**SUMMARY**

With this assignment, we were tasked with creating a To Do List using python. We needed to read and write data to/from a file and have it converted into python list. While last week we used two-dimensional lists to accomplish a similar task, this week we needed to use a dictionary to control task and priority attributes of individual list elements. This allowed greater control of the items so they could be standardized and indexed with words instead of numbers.

We also needed to be able to add and remove items from out lists. I ran into issues when trying to iterating through a list and removing items. As items get removed, the list changes and our for loop skips items as the list gets re-indexed. Instead of using the remove function, I decided to use the pop function which can remove records based on their index. Since our list of items to pop was built in ascending order, I was able to use a pop count that allowed me to keep track of the updated index as items got removed. Another method I could have used to solve this problem would be to make the task field unique. If someone entered an item that was already in the ToDo list, I could have appended a number to the item. For example, eat becomes eat2. That way, when trying to remove items, I’m guaranteed to only remove one. I decided to use the modified remove method so I didn’t need to run through the data every time something gets added.