Trevor Huynh

Nov 15, 2022

IT FDN 110 A

Assignment 05

https://github.com/trevwin/UW-IntroToProg-Python/

**Assignment 05: To Do Script**

**Introduction**

In this week, the list and dictionary class type was explored along with their properties. This week’s assignment expands on last week’s assignment by combining dictionary and lists to allow for read/write/removal of data relating to tasks and priority levels in a text file.

**Initial Plan / Pseudo-Code**

The initial plan / pseudo-code was created using the comments from the Assignment05.py starter file. The starter file has already been setup with commented code and the outer while loop and inner if/elif statements for user selection.

|  |
| --- |
| GLOBAL (step 1)   * Declare Variables * Load/read from file, display to user. If no file – return error message |
| STEP 2: Display Menu |
| STEP 3: Show current items in table   * Return to menu |
| STEP 4: Add / Append new task   * Return to menu |
| STEP 5: Remove item (task) from list   * Return to menu |
| STEP 6: Save tasks to file (write)   * Return to menu |
| STEP 7: Exit   * Exit script |

**Step 1: Load Data From ToDoList.txt File**

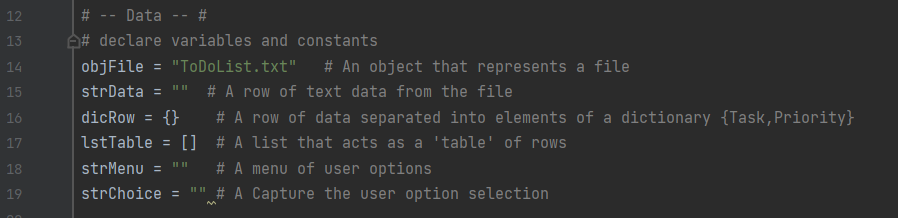
After general declaration of global variables, the first step is to load existing data from the *ToDoList.txt* file. For this step, 2 additional variables were initialized: *objFile2* and *lstRow*.

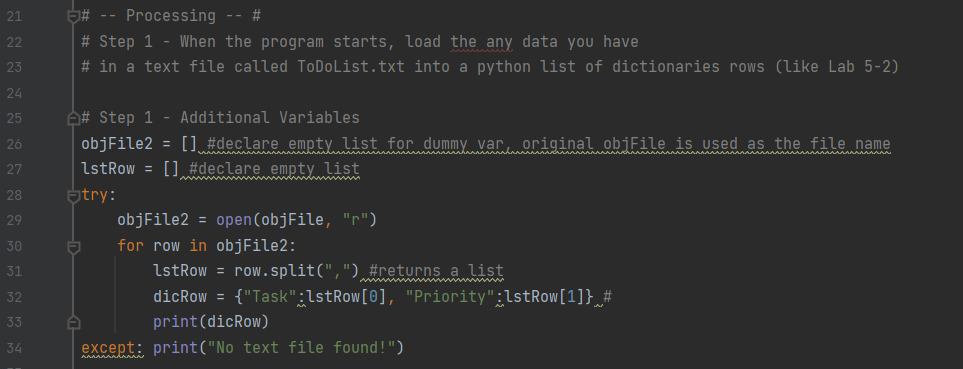
The variable *objFile2* is essentially a dummy variable to store the read contents of the text file as the usual variable for this, *objFile*, was already assigned as text file name in the global variables. The variable *lstRow*, stores the split values in the text file. Once this operation is completed, variable *dicRow* stores the indexed values in *lstRow* in a dictionary data type and assigns a key to it. The keys in this assignment are task and priority.

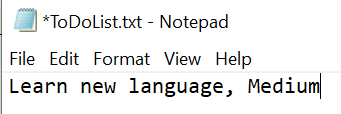
In addition to this, a try/except block is used to get around an error that is anticipated to occur when no text file is found. This is required as this section of the code will always run.

Verification of this section was then run with the following cases:

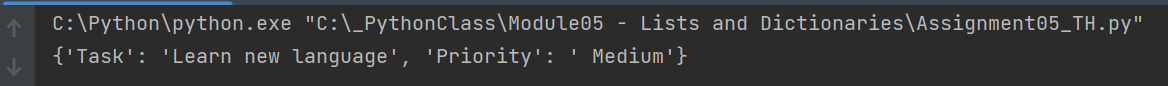
1. With a text file
2. Without a text file

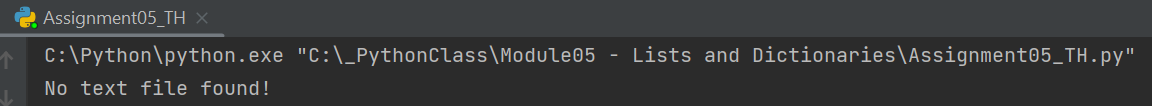
*Figure 1: Global Declaration of Variables and Constants*

*Figure 2: Step 1 Code*



*Figure 3: Text File Contents*

*Figure 4: Output of Step 1 with Pre-loaded Text File*

*Figure 5: Output of Step 1 with Missing Text File*

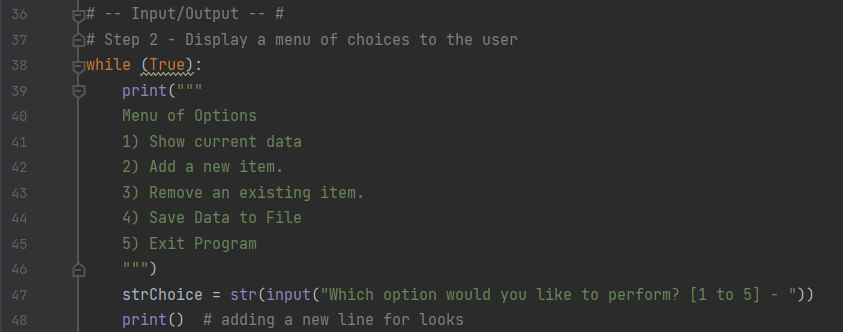
**Step 2 / 3: Display Menu to User, Show Current Items in the Table**

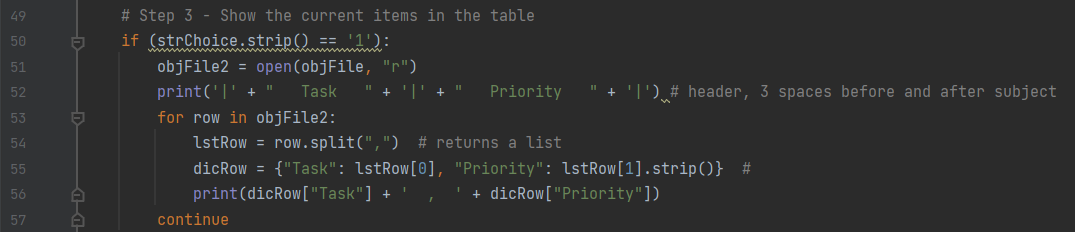
In this section of the code the goal is to display the menu to the user (step 2) and allow the user to initiate showing current items in the table (step 3).

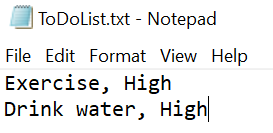
Step 2 has been already completed in the starter file and was not changed.

Step 3 will be similar to step 1 with the difference being that it will be more user friendly (easy to read). It is assumed that the text file exists and there is data in there. The same variables are used as in step 1 as new data will be stored in them. A for loop is used to split the data and return a list, the values in the list are then assigned to dictionary variable *dicRow* using indexing / assigning a key, and finally the values in the dictionary are printed out in a table via key reference.

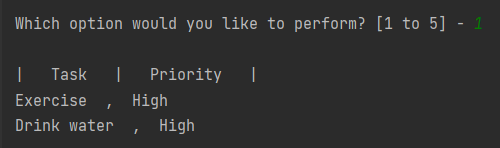
Verification of this section was completed using a new text file with 2 rows of data.

*Figure 6: Step 2 Existing Menu Code in Starter File*

*Figure 7: Step 3 Reading Current Items in Table*



*Figure 8: Text File Data for Step 2/3 Verification*



*Figure 9: Output Data for Step 2/3*

**Step 4: Add new Item to List / Table**

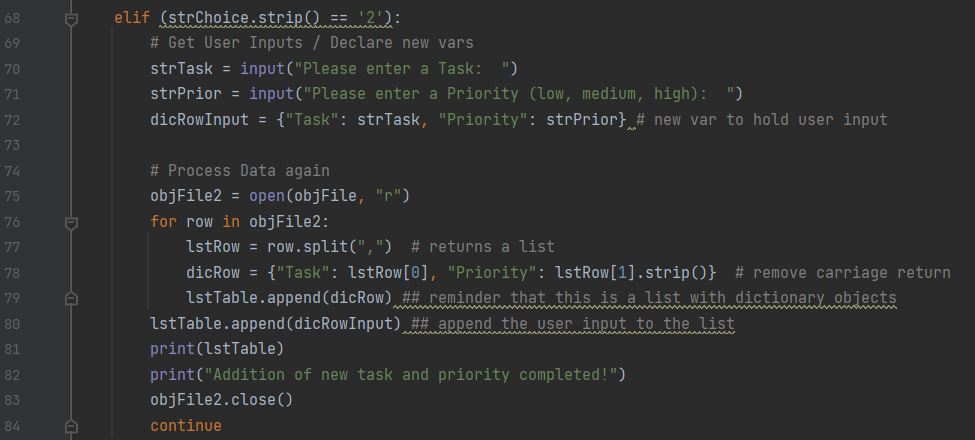
In this section of the code the goal is to essentially append new items (Task / Priorities) to the existing table.

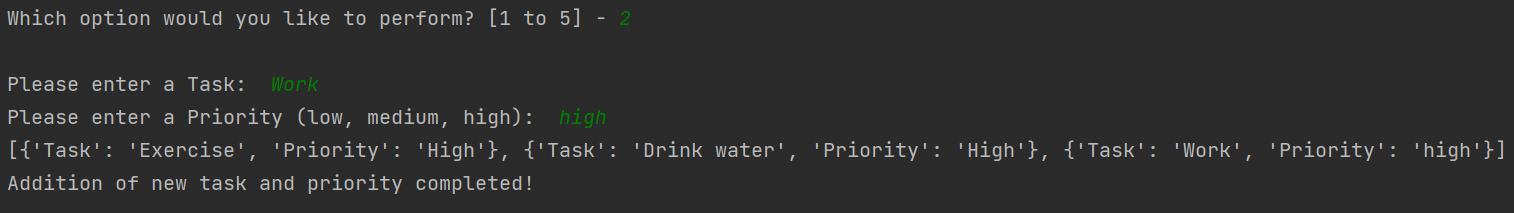
The setup will require appending user inputs for both keys into the existing text file. It will be similar to the previous sections as it retains the data processing steps.

There are 3 new variables created specific to step 4: *strTask, strPrior, dicRowInput*. The 2 user input variables are used to store the user input data specific to the respective dictionary key. Another dictionary is created with only the user inputs.

The same processing step is used to read the data from the text file, split the data and place into a list, create a dictionary using the list indexes, and then appending the dictionary data into a list table. At this point in the code, we have a list with the dictionary objects, *lstTable*, that was read from the text file. As this is inside the for loop, another append operation is applied with the new user input dictionary, *dicRowInput*.

Verification of this section was through printing the final stored values in *lstTable* after the user inputs were appended.

*Figure 10: Step 4 Code*

*Figure 11: Output Verification of Step 4 Code*

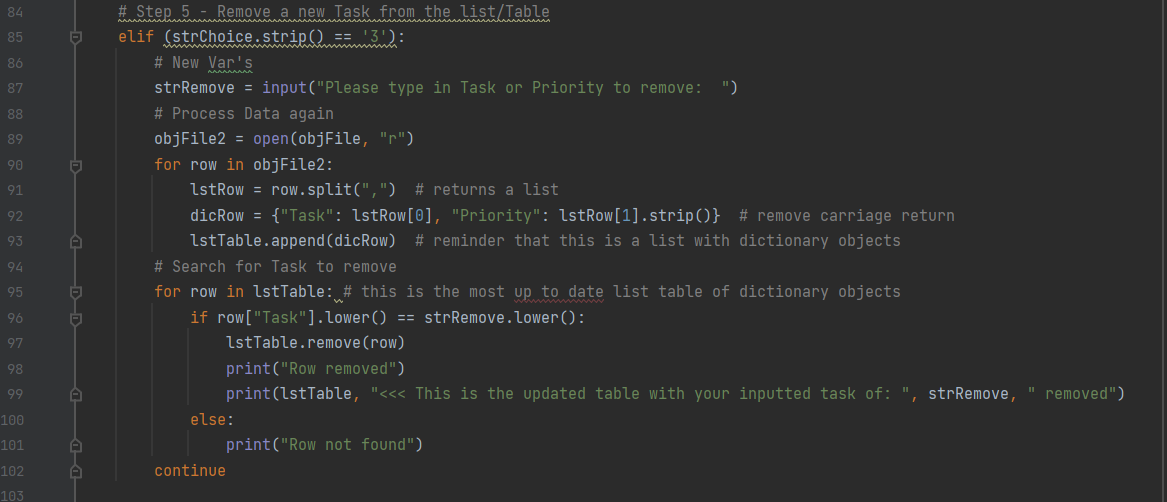
**Step 5 - Remove a new Item From the list/Table**

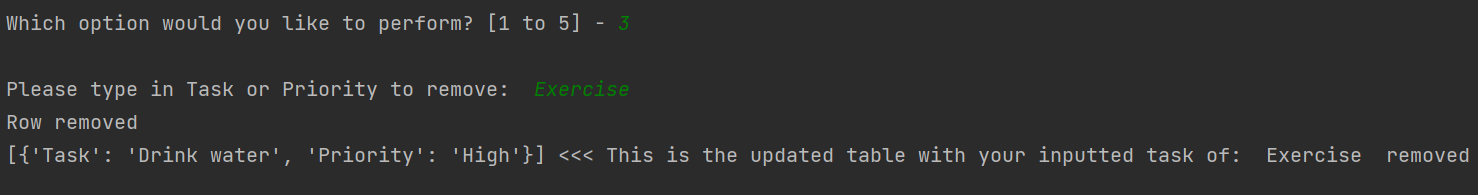
In this section of the code the goal is to remove a task that matches the user input.

The setup will be similar to the previous sections as it retains the data processing steps. At the end of the processing step, an updated list variable with dictionary values is created and stored in memory.

The only new variable created is the user input variable for task removal: *strRemove*. With this, a for loop is utilized where each task is searched for a match to the user input. The *lower( )* function is used on both the existing tasks within the text file and the user input to ensure no mismatches with upper/lowercase letters. In addition, the removal of the specific row where the match to user input occurs utilizes the built in *remove( )*function.

Verification of this section was through printing the final stored values in *lstTable* after the user inputs were appended.

*Figure 12: Step 5 Code*

*Figure 13: Output Verification of Step 5 Code*

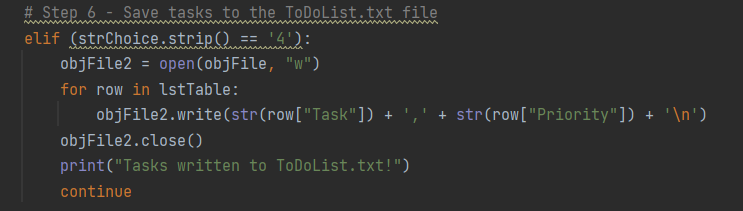
**Step 6 - Save tasks to the ToDoList.txt file**

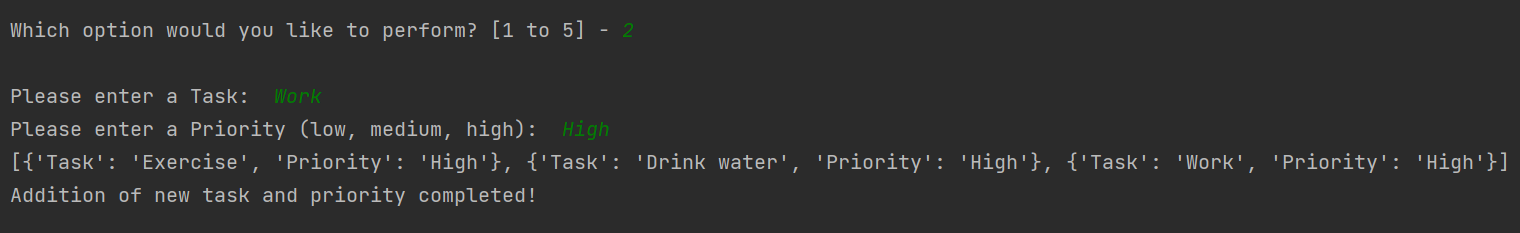
In this section of the code the goal is to write the data to the text file.

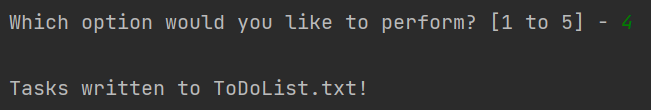
The setup will be similar to the previous sections as it retains the data processing steps. At the end of the processing step, an updated list variable with dictionary values is created and stored in memory.

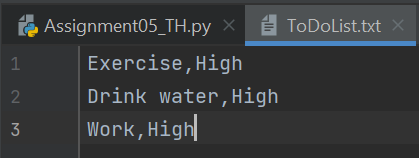
The write function is utilized in a for loop to write each key in the list of dictionary values, lstTable. This also includes a carriage return to allow the write operation to occur on each row.

Verification of this section was through printing the final stored values in *lstTable* after the user inputs were appended.

*Figure 14: Step 6 Code*

*Figure 15: Append Step to Verify Step 6 Write Operation*

*Figure 16: Step 6 Selection*



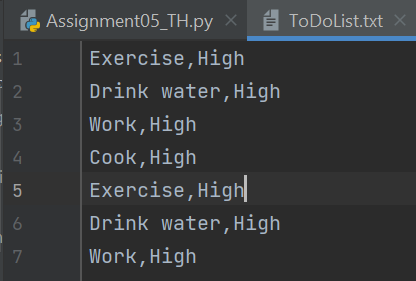
*Figure 17: Step 6 Written Text File*

**Reference Issue**

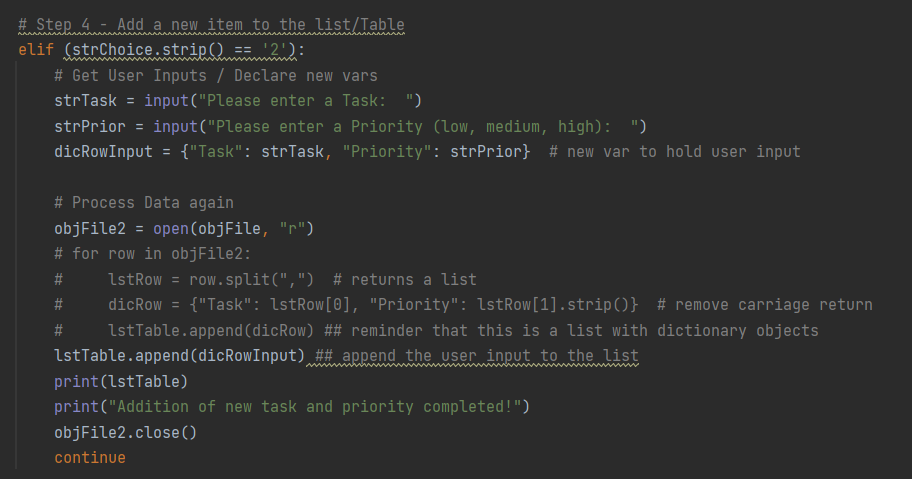
An issue came up while testing repeated append and writes. This issue was narrowed down to performing 2 operations in a row where new tasks were added. This unfortunately cascaded into other parts of the code. The issue was traced back to repeating the data processing part for each step (see sections above). This is essentially a duplicate code. As the list with dictionary values in it, *lstTable*, is storing data, some portions of the code do not use *lstTable*, while others do, and when there is no consistency in using certain variables, it can create cascading duplicate lists.

The changes that were made included keeping a common up to date variable (*lstTable*) for all dictionary values. In addition, the repeated data processing steps to read from the text files were removed. There is no reason to repeatedly do this when the initial data processing is completed during the variable initialization. This is also the reason why viewing current data was impacted; the “view current data” step essentially just read from the text file as opposed to reading what is currently stored in memory.

Once the changes were made, the script was tested again for multiple append/remove operations in a row to verify.



*Figure 18: Duplicate Issue*



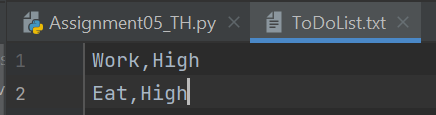
*Figure 19: Example of Duplicate Code in Step 4. The commented-out code was causing the duplicate issues.*

**Step 7 - Exit program**

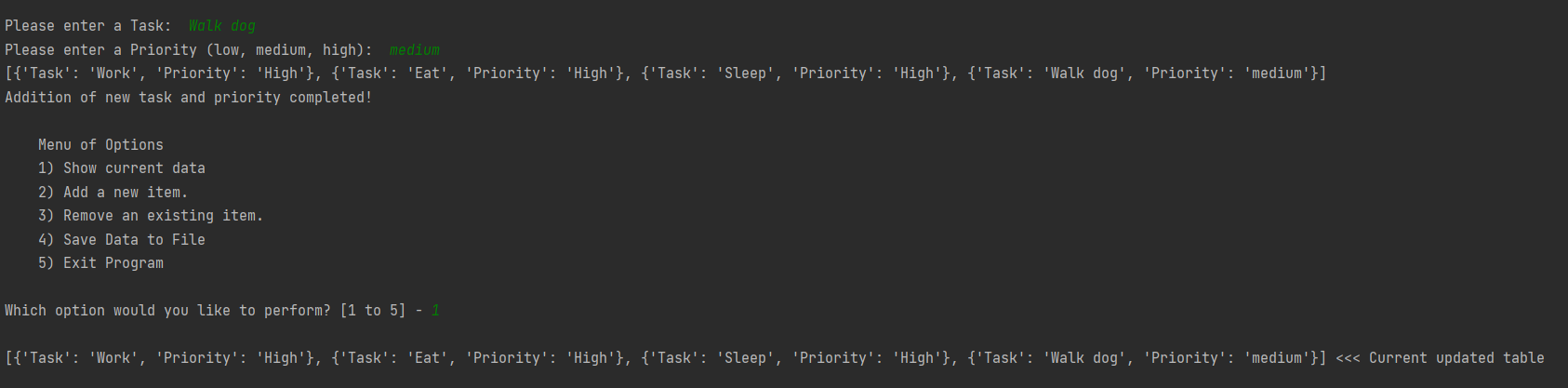
Once the major reference/duplicate issue was resolved, the exit portion of the script used a simple *break* function to break out of the main while loop.

**Final Verification (PyCharm / Windows CMD)**

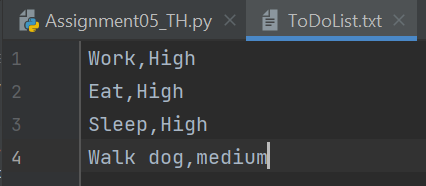
Final verification was completed in PyCharm and Windows CMD. It involved running the script in both environments. For running the script in Windows CMD, the directory was changed from the default one to the assignment05 folder using the “cd” command in Windows CMD. If this is not done the text file cannot be located.



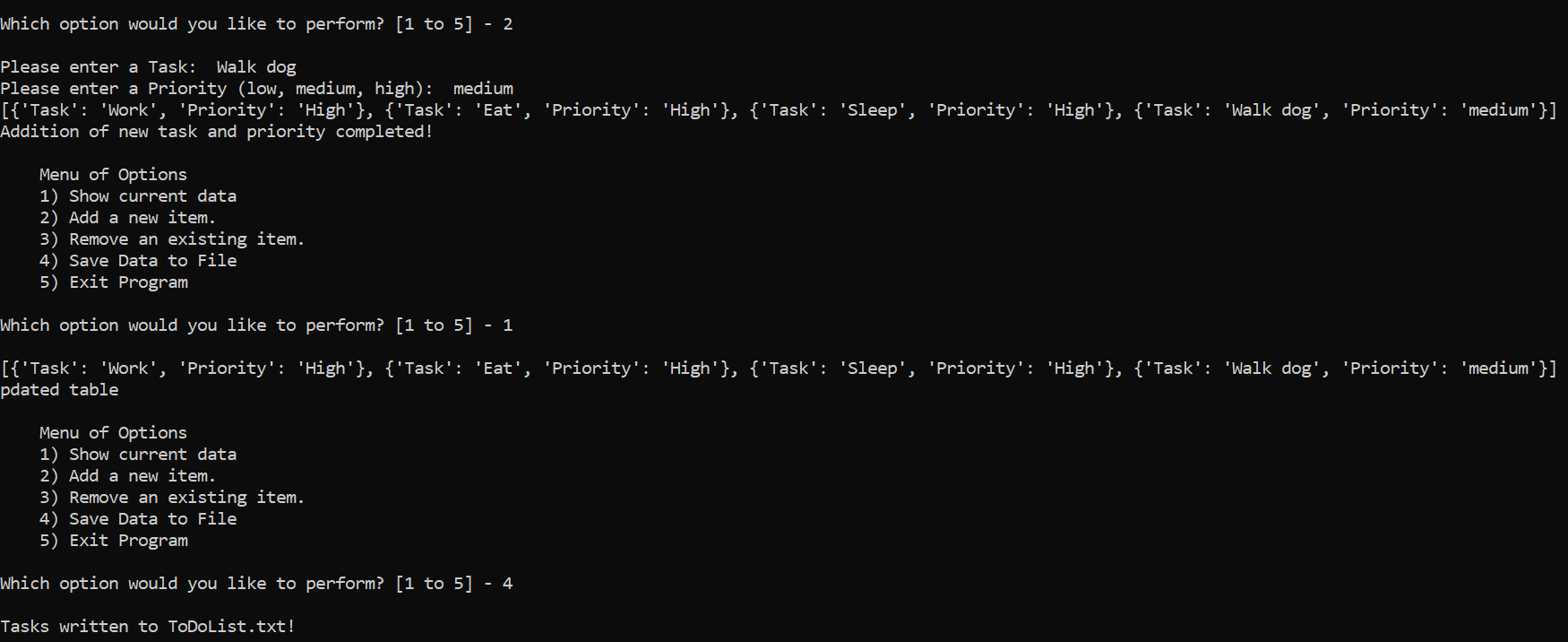
*Figure 20: Pre-loaded Text File for Verification*

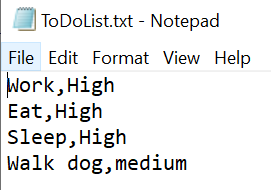


*Figure 21: Testing Multiple Appends*



*Figure 22: Text File After Multiple Appends / Remove*

*Figure 23: Windows CMD Output*



*Figure 24: Text file after running script in Windows CMD*

**Summary**

This assignment expanded on the previous assignment by incorporating list and dictionary class types into a script that reads/writes/removes data from a text file. It also incorporated the concept of separation of concerns which is a structured way to program. This included formally declaring and initializing variables before the main body of the script. As the scripts get more and more complicated, this concept is important as in this assignment, multiple duplication issues arose due to tracking of updated variables.