Geoffrey Sears  
November 6, 2019  
Foundations of Programming, Python  
Assignment 05

# Introduction

The assignment this week was to modify a pre-existing template in order to make a functioning script which could:

* Load a pre-existing text file (ToDoList.txt) into memory for use by the application.
* Add new items to a to do list to be stored into the List of Dictionary nested format.
* Remove items from the list through their associated dictionary key.
* Write the current list in memory to a text file, so it can be used the next time the application is used.
* Provide the user a way to gracefully exit the program.

This week’s assignment proved to be a difficult one for me, but I ended up with a functional script that hit all of the above points.

# Applying My Knowledge

## Data Section

The first step in writing this script was understanding the template which we had been given. The script was divided into Data, Processing and Input/Output sections. A number of variables were declared at the top of the script which I modified to conform with the conventions we have learned to date. In particular:

* I renamed objFile to strFile to represent the filename which identified the text file which would be opened and updated by the script.
* objFile was then assigned to None as a placeholder for future file handling via the open function.
* There was a statement {task,priority} which served as a hint for the structure of the dictionaries we would be populating as rows using the dicRow variable. I commented this statement out as it was not useful.

The remaining variables (lstTable, strMenu, strChoice) were left as written. I will reference them below as we get into later blocks of code.

## Processing Section

The first block of code I inserted into the script was this:

objFile = open(strFile,**"r"**)  
**for** row **in** objFile:  
 lstRow = row.split(**","**)  
 dicRow = {**"Task"**: lstRow[0], **"Priority"**: lstRow[1].strip()}  
 lstTable.append(dicRow)  
objFile.close()

This code reads into the program any data which has been previously written to the “ToDoList.txt” file stored in the same directory as the script.

The open function uses the “r” mode parameter which instructs the program to open the file for reading, but to not make any changes to the file.

The “for row in **objFile**:” loop reads through the opened text file (“ToDoList.txt”) and separates the elements within the file by the comma delimitation. It then assembles the returned values into dictionary statements using the **dicRow** variable. Each iteration of this loop then appends those values to the variable **lstTable**, constructing a list with nested dictionaries for each row.

The end result of this block is that the program is restored to the state that it last left off when previous executions of the program wrote newly entered data to the file.

### While Loop

The main functionality of the program is then built within the While Loop on line 45. This starts with a text block which defines the programs menu:

**while** (**True**):  
 print(**"""  
 Menu of Options  
 1) Show current data  
 2) Add a new item.  
 3) Remove an existing item.  
 4) Save Data to File  
 5) Exit Program  
 """**)  
 strChoice = str(input(**"Which option would you like to perform? [1 to 5] - "**))  
 print() *# adding a new line for looks*

The program continuously iterates through this menu as the user makes their desired selections. The program only terminates when option #5 is executed. Following this supplied content, I composed the following blocks of code for each of the program’s options.

### Option 1 – Show Current Data

**if** (strChoice.strip() == **'1'**):  
print(**"Task"**,**" | "**,**"Priority"**)  
 **for** row **in** lstTable:  
 print(row[**"Task"**],**" | "**,row[**"Priority"**])  
 **continue**

This block takes the existing variable **lstTable** and walks through each of its elements which are stored dictionaries constructed through the initialization loop created at the head of the processing section. The print statement makes use of the dictionaries’ keys to return the “Task” and “Priority” Key-Value pairs. For ease of reading, I separated these elements using a pipe “|”. The continue at the bottom of the block returns the script to the top of the while loop once the for loop has completed.

### Option 2 – Add a new item.

This section allows the user to add new content to **lstTable** through a pair of input statements, and the use of the append method to add the newly entered values to the end of the list.

**elif** (strChoice.strip() == **'2'**):  
 taskEntry = input(**"Enter the name of the task: "**)  
 taskPri = input(**"Enter the priority of the task (1-Low to 5-Highest): "**)  
 dicRow = {**"Task"**: taskEntry, **"Priority"**: taskPri}  
 lstTable.append(dicRow)  
 **continue**

The statement ***dicRow = {“Task”: taskEntry, “Priority”:taskPri}***constructs a properly formatted dictionary to be subsequently appended to the **lstTable**.

### Option 3 – Remove an existing Item

For me, this is where it got hard. To accomplish this step I wrote the following block:

**elif** (strChoice.strip() == **'3'**):  
 *# TODO: Add Code Here* taskDelete = input(**"Enter the name of the task to delete: "**)  
*# objFile = open(strFile)* **for** row **in** range(len(lstTable)):  
 **try**:  
 **while** lstTable[row][**'Task'**] == taskDelete:  
 **del** lstTable[row]  
 **except**:  
 **continue**

I went around and around the bend on this one. Most of the attempts to implement this appeared to be successful until I tested to see if the element that I had entered in the input statement had been removed from the active list of tasks. It had not. Initially I was having difficulty getting the nested dictionary to be correctly evaluated for whether the entered Key **taskDelete** was in the dictionary returned via the for loop.

I eventually gave up on my initial method, and settled on one which I found on the Internet here: <https://www.geeksforgeeks.org/python-removing-dictionary-from-list-of-dictionaries/>

At the time I had settled on this solution I was having difficulty with the fact that the For loop was never matching the dictionary being returned in my conditional statement. I needed to provide the [‘Task’] key label to match the appropriate key within the dictionary returned by the loop to the **taskDelete** variable.

Once I had managed to make that connection, I was able to implement the delete command. But here I ran into another problem. Error control. For reasons I could not figure out, as soon as the element was deleted from **lstTable.** The program would error out with a key error. I elected to suppress this using the try:/except: commands, once I determined that the for loop was successfully deleting the item from the list, and there seemed to be no other detrimental affect of ignoring the error. This was the hardest portion of the script to resolve.

### Option 4 – Save Data to File

This step was kind of old hat:

**elif** (strChoice.strip() == **'4'**):  
 *# TODO: Add Code Here* objFile = open(strFile, **"w"**)  
 **for** row **in** lstTable:  
 objFile.write(row[**"Task"**]+**','**+row[**"Priority"**]+**'\n'**)  
 objFile.close()  
 **continue**

I used the same construction we have used in numerous exercizes before. The open function used “w” mode, which stands for Write. This means that when this is executed the contents of **lstTable**  through the subsequent for loop would be written to the text file, and whatever was in ToDoList.txt would be destroyed. This is okay, because the list being written was constructed from the contents of this file at the start of the Processing section and then modified to suit the purposes of the user.

The **objFile.write(row[“Task”]+’,’+row[“Priority”]+’\n’)** statement makes use of the dictionaries Key-Value subscripts to return the values within the dictionary to be written to the text file in comma-delimited format.

### Option 5 – Exit Program

This step provides the user with the ability to gracefully exit the program.

**elif** (strChoice.strip() == **'5'**):  
 print(**'Thank you for using the ToDo Manager.'**)  
 input(**'Please hit Enter to exit the program.'**)  
 *# TODO: Add Code Here* **break** *# and Exit the program*

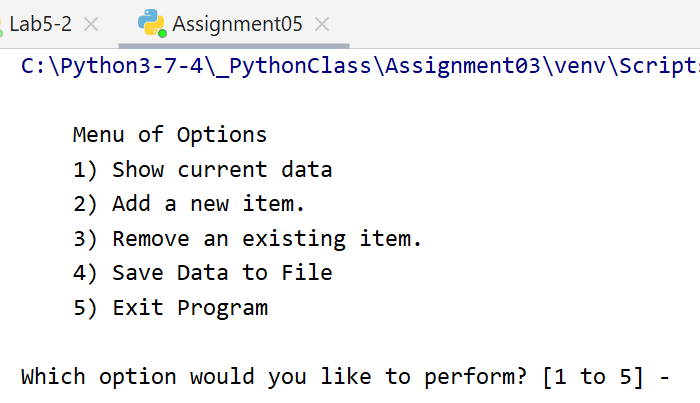
The control is simply matching to the user entering option “5”, and in order to give the user control over when the program closes, particularly if they are executing this from the desktop I included an input statement to pause the program before executing a break to break the While loop at the head of the processing section.

## Does it Work?

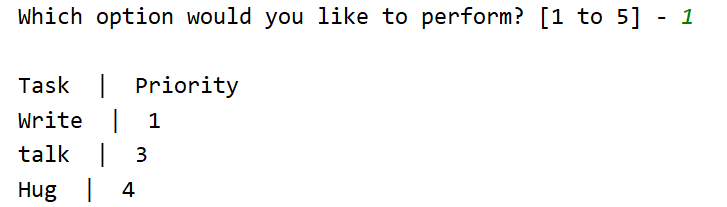
The following screen shots demonstrate the functionality of the program.

### Pycharm

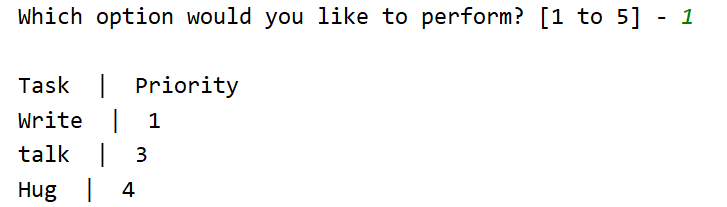
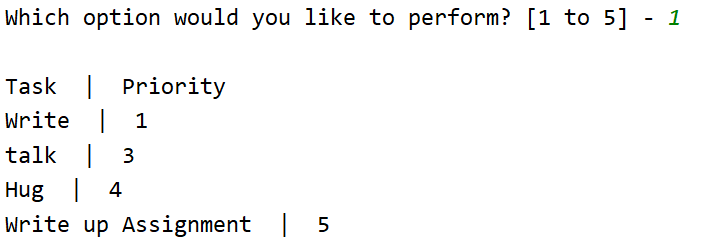
On initial execution the program provides a menu of options:



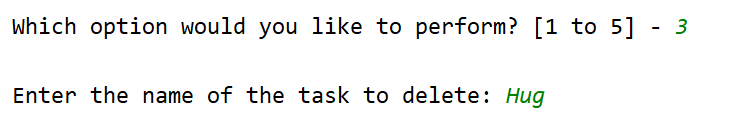
Selecting option #1 will output the contents of the ToDoList.txt file in the form of a table:



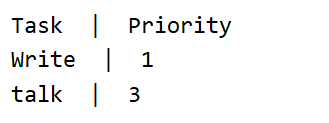
If I choose option #2 I am prompted for inputs to add to the ToDo list, and then can confirm they are taken via option #1 again:

Having done this, I am able to remove items from my list using option #3:

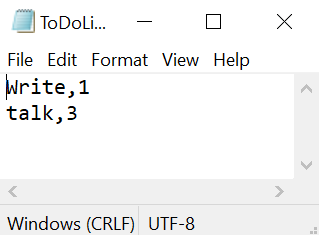


Which results in “Hug” being removed from the list:

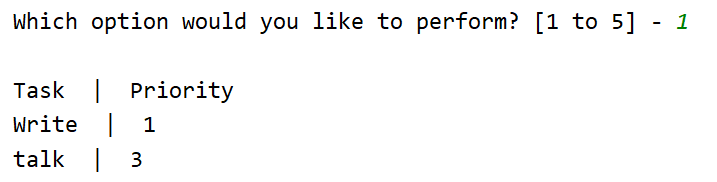


Please note that “Write up Assignment” was also dropped from the list because I had to interrupt the demonstration to remove an extraneous piece of code.

Finally, I was able to write the contents of the list to Text file using option #4:

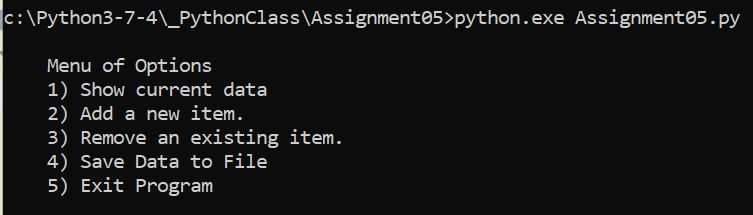


Which after closing the program and reopening it will load the same data into the program and display it using option #1:

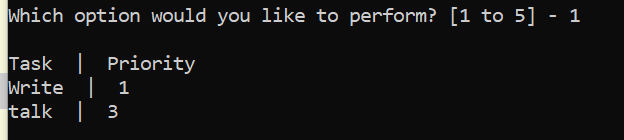


### Via Windows Command Prompt

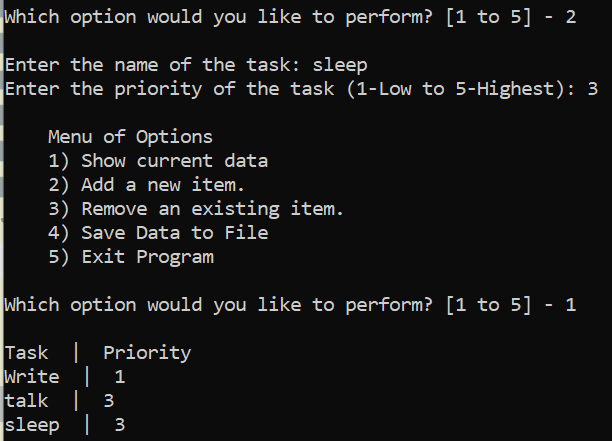
The program is able to execute from the command line as follows:



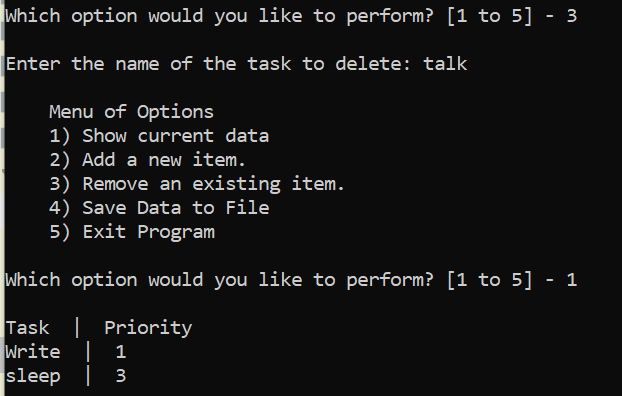
Selecting Option #1 lists the current contents of ToDoList.txt:



Selecting Option #2 allows you to add to the current list:

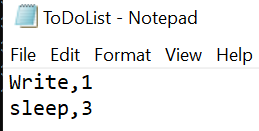


Selecting Option #3 allows you to remove an item from the list:



After selecting option #4 the current list is written to the text file:





# Summary

In this week’s assignment we were asked to take a script template and complete it to provide the following functionality:

* Display the current contents of the To Do list as stored in the ToDoList.txt file contained within the same directory.
* Load the information stored in that file into a nested List, Dictionary structure to permit management of that data within the execution of the program.
* Add new members to the list by collecting input from the user and appending that input to the list stored in memory.
* Remove existing members of the list as stored in memory.
* Provide the user the ability to write their changes to the list to the text file, and
* Provide the user with a graceful way to exit the program.

I was really stuck on this week’s assignment around how to remove items from the list. The methods taught in the classroom on their own did not appear to suffice to the assignment until I made use of the try … except method to provide Error control which permitted me to by pass an error which was occurring in my For Loop which was iterating through the contents of the lstTable list which served as the container for the dictionary content. I don’t know if that was the intended solution, but it was the best one I could come up with.

Last week we covered using the While loop as an overall control method for allowing the program to continually prompt the user for input after executing each requested action around maintaining the HomeInventory list. This week we added the piece of managing list content in a more interactive way using file storage to create continuity between user sessions.