Cindy Li

Nov 19, 2021

IT FDN 100

Assignment 6

<https://github.com/cindy-x-li/IntroToProg-Python-Mod06>

# The ToDo List Script with Functions

## I. Introduction

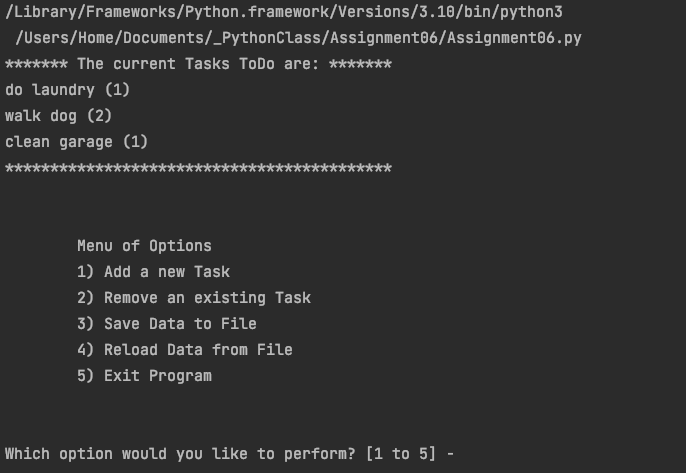
The sixth assignment is to modify a programming template of the completed ToDo List script previously completed in the fifth assignment. However, the provided code is not complete and only uses a few functions, so the goal is to complete the functions and use them to organize the code in the main script.

## II. Code Walkthrough

Although I completed the fifth assignment, using semi-completed functions was very confusing in the beginning. Not only had I have to take the time to understand what every function does, but also figure out which function to call, what order to call them and from which class to call them from in main.

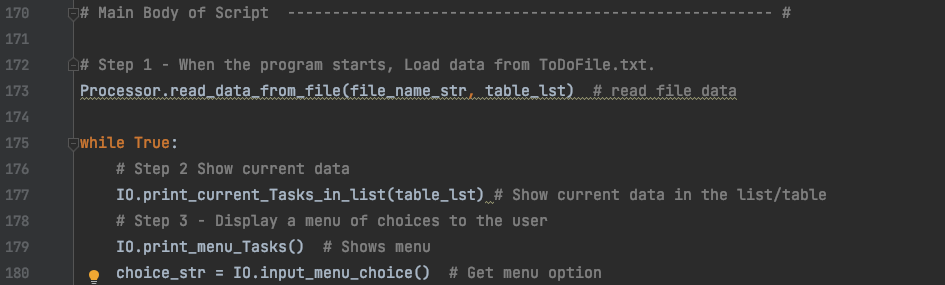
### 1. Steps 1-3:

When the user starts the program, s/he will see Figure 2.1.1, where the current tasks from the file is already loaded into a table to review, the menu is present and there is a prompt for the user to enter a selection from the menu.



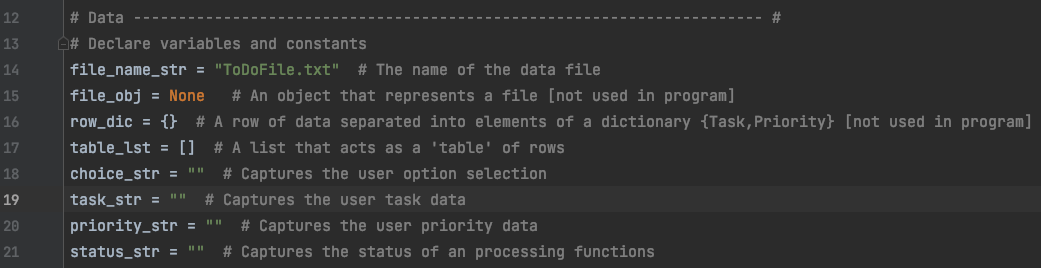
***Figure 2.1.1: Initial screen of the user interface***

The code for the initial screen, as seen in Figure 2.1.1, is already completed in the template (Figure 2.1.2). Reading through this script gave an insight into the structure of the code and how I would then write code into these functions. All of the program’s functions were divided into two classes: Processor and IO, and depending on the action that needs to be performed, functions were called from one or the other.



***Figure 2.1.2: The start of the main body of the script***

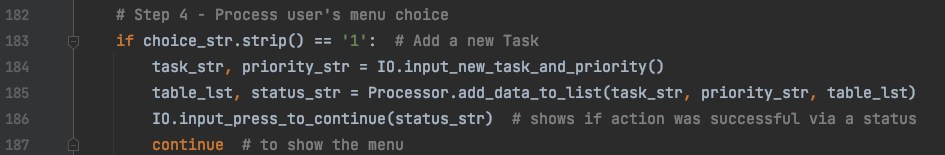
I also used the provided variables and constants in the data section to help guide my code as well (Figure 2.1.3). I added additional variables (lines 19-21) and chose not to use certain variables such as file\_obj and row\_dic (lines 15-16).



***Figure 2.1.3: Data Section.***

### 2. Step 4 – Adding a new Task

Writing functions for this script starts in step 4, where the user adds additional task to the list (Figure 2.2.1).



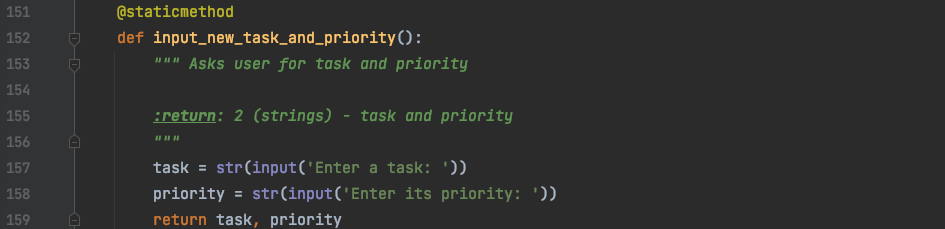
***Figure 2.2.1: Processing user’s choice in main***

To complete this task, the following has to be completed:

1. Request user to enter a task and priority (IO)
2. Add data to a list (Processor)
3. Press enter to continue (IO)

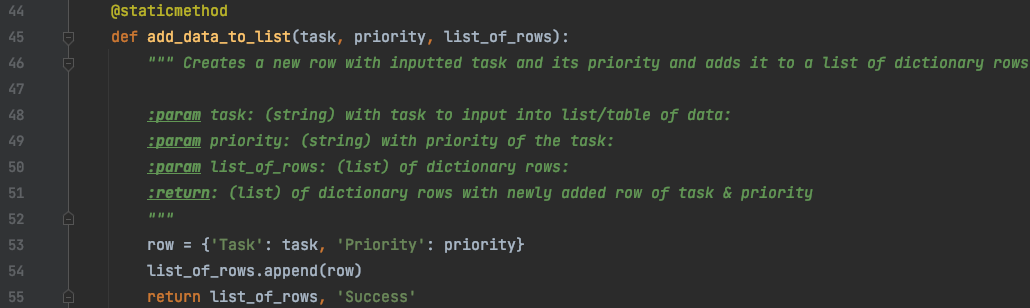
Using this logic, I found the functions in each class that could accomplish this tasks and completed the function if needed.

For part 1, in the function IO.input\_new\_task\_and\_priority, I coded lines 157-158 to request user to inputs and in line 159, I returned two strings (Figure 2.2.2). These two strings are then captured in the variables task\_str and priority\_str (Figure 2.2.1, lines 184)



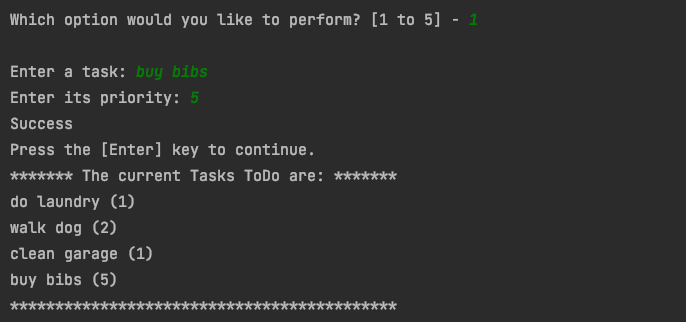
***Figure 2.2.2: Function for inputting a new row into the table***

Then I passed these variables into the next function Processor.add\_data\_to\_function, along with the current list/table of data (Figure 2.2.3, lines 53-55). This function returned the modified list of data and the string “Success” to tell me if the function ran. I decided to incorporate this message into the UI so the user can also see if their input worked or not. This IO code can be seen in Figure 2.2.1, line 186.



***Figure 2.2.3: Function for adding data to table***

As seen in Figure 2.2.4, the user can add a task and its priority to the table.

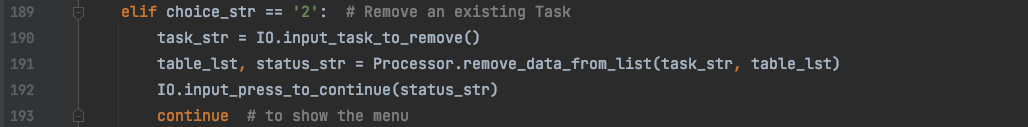
******

***Figure 2.2.4: User adds task and priority to table.***

### 3. Step 4 – Removing an existing Task

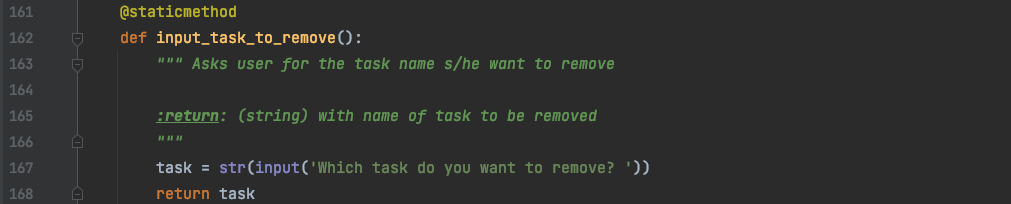
For removing an existing task, I wrote out the following parts and coded it in Figure 2.3.1:

1. Request user for the task to remove (IO)
2. Check if task is in table (Processor)
3. Print out if task was removed or if it cannot be found. (IO)



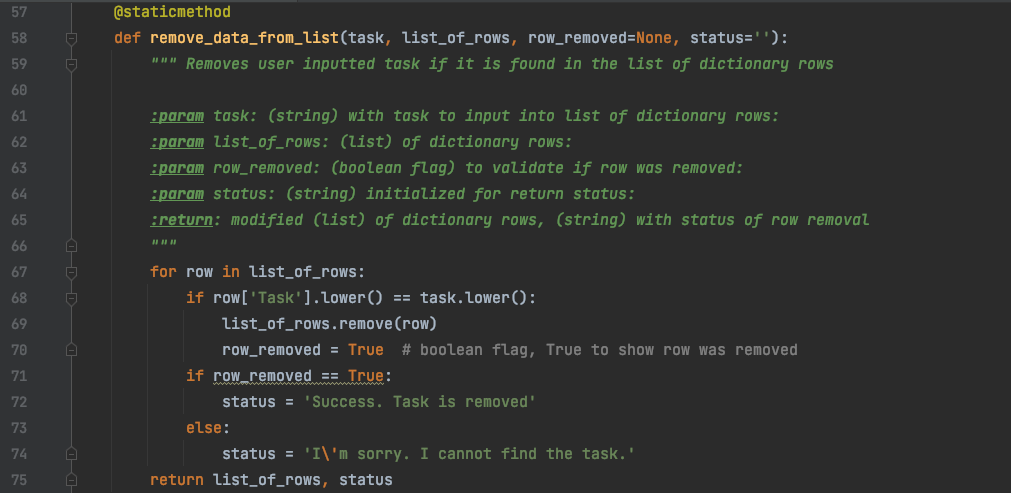
***Figure 2.3.1: Removing an existing task***

Whatever task the user enters, the function in Figure 3.2, lines 167-168, returns the task.



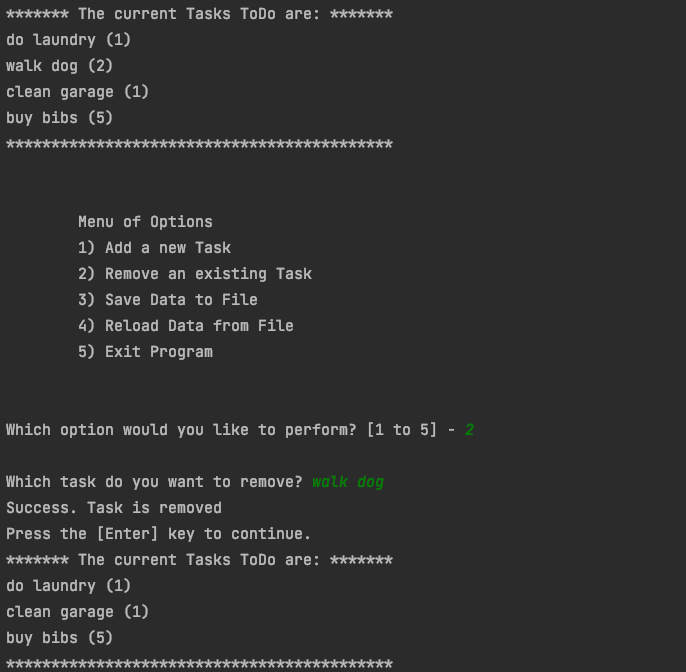
***Figure 2.3.2: Function to request name of task to be removed***

This task is captured in task\_str variable and used by Processor.remove\_data\_from\_list to check if it exists in the table (Figure 3.3, lines 67-69). Then I used a boolean flag check if it was in the table and the user is informed if the requested row was removed or not by returning a status (lines 70-75). The status is printed to the user via the IO.input\_press\_to\_continue by passing in the captured variable, status\_str.



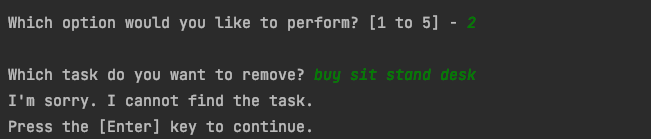
***Figure 2.3.3: Function to remove desired row via task name from list***

Figure 2.3.4 shows the UI for removing the task, “walk dog.”



***Figure 2.3.4: Removing a task***

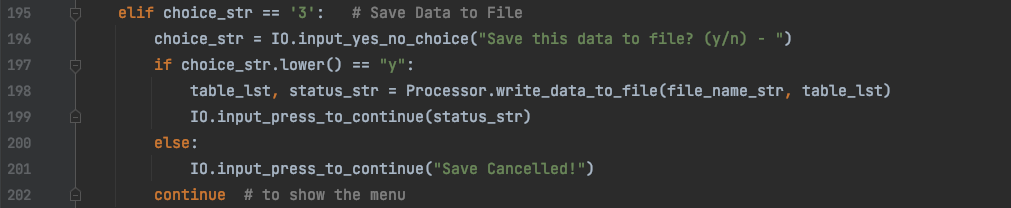
When the user enters a task that does not exist, the program informs the user it cannot be found in the list (Figure 3.5).



***Figure 2.3.5: When a task is not found***

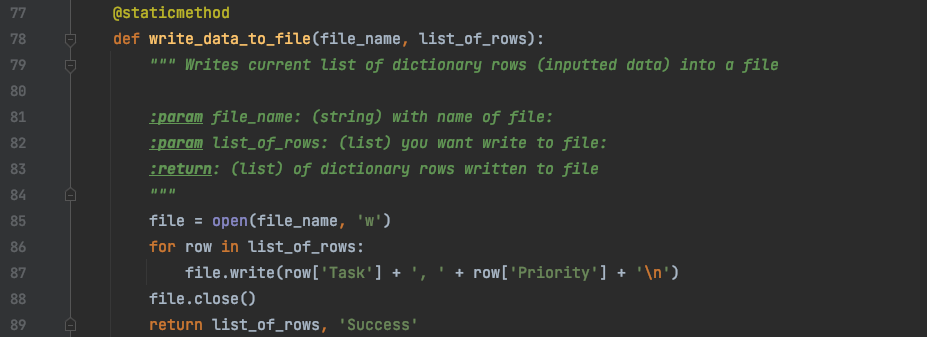
### 4. Step 4 – Saving Data to File

For saving a file, I only had to write 1 line of code, on line 198 to complete this task (Figure 2.4.1). I called the Processor.write\_data\_to\_file and provided the variables: name of the file and the table\_list. Then I captured the modified list of data and the status. The status tells the user and I if the function ran.

******

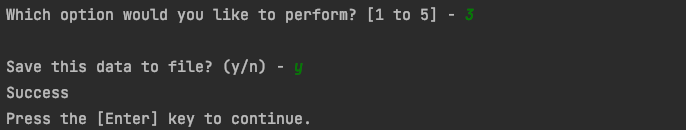
***Figure 2.4.1: Items from the ToDo List***

Figure 2.4.2 demonstrates the code is identical to that of the fifth assignment, with some simplication to variable names.

******

***Figure 2.4.2: Script for adding new items to the list/table***

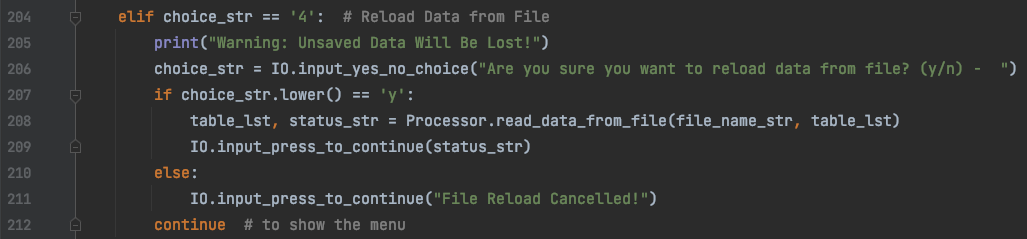
The user is notified when this action is complete (Figure 2.4.3) and prompted to continue with the program.



***Figure 2.4.3: The user is notified data is saved.***

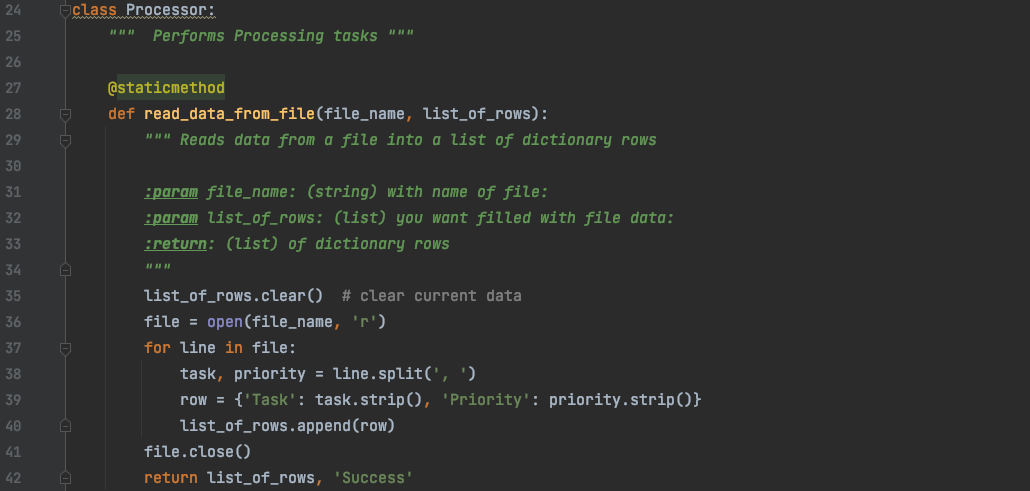
### 5. Step 4 – Reload Data to File

The reload data to file option was the only difference between the fifth and sixth assignments. The concept to understand is when the user is adding items in the program prior to saving, s/he is appending items to a list of data, the variable table\_lst, and that data only exist in that variable until it is saved in the file, by using the write data function in the processor class. As a result, the global variable table\_lst and local variable file contain two separate lists of data prior to writing data to the file. Therefore, using the function Processor.read\_data\_from\_file will reload the data from the file on line 208 and lose any newly added rows of data to table\_lst (Figure 2.5.1).



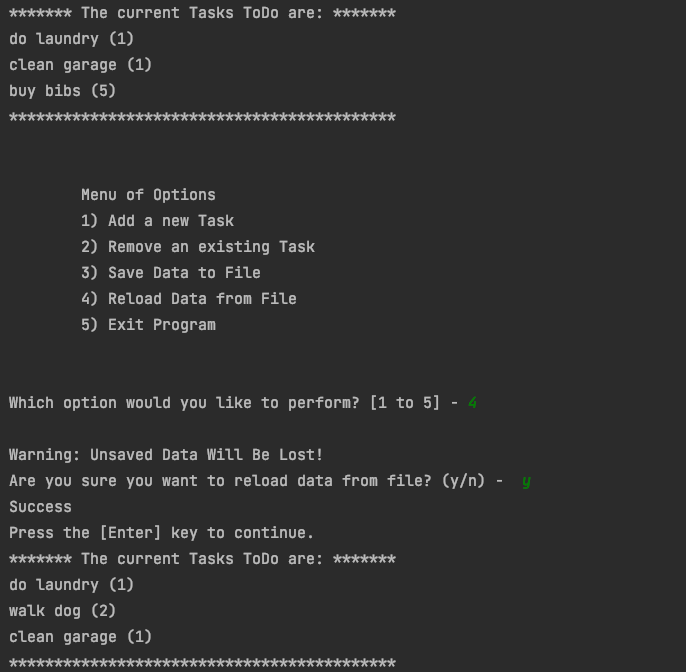
***Figure 2.5.1: Script for reloading data from file***

Figure 2.5.2 demonstrates the code that was included in the provided template.



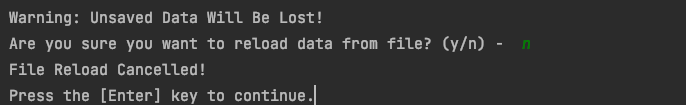
***Figure 2.5.2: Function for reading data from the file***

Figure 2.5.3 is the UI for the script in Figure 2.5.1. The UI shows how the tasks in the “Tasks ToDo” changed after reloading data from the file. The changes made prior, such as deleted task “walk dog” is added back into the table and the added task “buy bibs” is no longer in the table. The table has returned to the original list that was loaded from the file (as seen first in Figure 2.2.1).



***Figure 2.5.3: Reloading data from file***

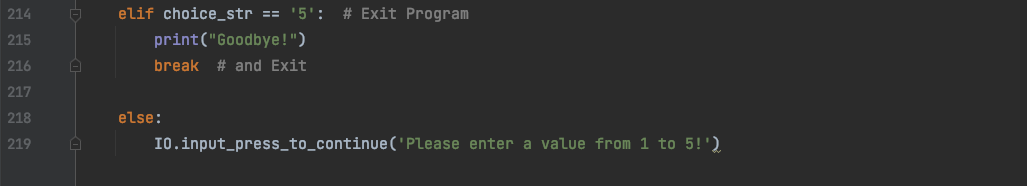
If the user did not want to reload the file after selecting option 4, s/he also has the ability to cancel the selection and retain the modified list (Figure 2.5.4).



***Figure 2.5.4: File reload cancelled***

### 6. Step 4 – Exiting Program & Other Choices Outside the Menu

Figure 2.6.1 shows the script that was already included in the template that enables the user to exit the program by selecting option 5. Figure 2.6.2 demonstrates the UI.

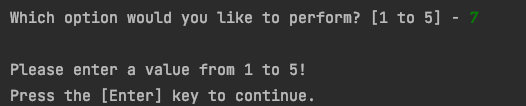
******

***Figure 2.6.1: Demonstrating how to exit the program***

### Macintosh HD:private:var:folders:yw:13gbt77n1hs35zwm545dn46h0000gn:T:TemporaryItems:Screen Shot 2021-11-19 at 7.45.02 PM.png

***Figure 2.6.2: Demonstrating how to exit the program***

I added lines 218-219 in Figure 2.6.1 in case the user enters an option outside of the ones provided in the menu and the user will be prompted to enter a value from 1 to 5 (Figure 2.6.3).



***Figure 2.6.3: Demonstrating how to exit the program***

## III. The Text File

Figure 3.1 and 3.2 shows how the script modified the ToDoFile.txt by adding an additional task, “buy bibs, 5”



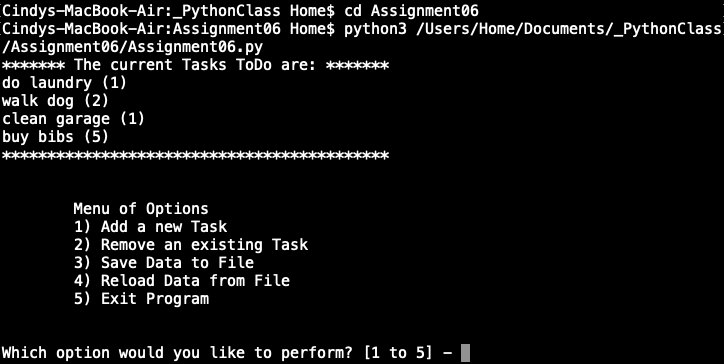
***Figure 3.1: Original text file***



***Figure 3.2: The modified text file***

## VI. Terminal

Figure 4.1 shows the program running in the terminal once it is in the same folder as the text file.



***Figure 4.1: Script running in terminal***

## V: Summary

There are two key takeaways from this assignment. The first takeaway is recognizing how the tools provided by Pycharm’s IDE made this assignment much easier. I could split the screen to have multiple tabs and/or the same tabs open, which makes is easier to write longer lines of code. The second takeaway is appreciating the reusability of functions. For example, the function IO.input\_press\_to\_continue(status\_str) was used in every if/elif statement and it could easily be called upon.

After completing this lab, I still have the following lingering questions about functions:

1. From my perspective, all the statements that handle the lines of code for working with the file occurred in the functions, so file\_obj was never called on in the main function and as a result was never used. I did not want to call on file\_obj in the functions and create a shadowing situation where file\_obj was to be used later on in main. However I wonder if I could have moved some statements from the functions in the main script so file\_obj can be used in main. What are some simple guidelines that can help me determine what goes in a function and what remains in the main script?
2. Is it preferred to pass and use the tuple row\_dic in the main script, rather than unpack the return tuple into task\_str and priority\_str?