Name: Stephanie Tarczynski

Date: 11/24/2020

Course: IT FDN 110 A

Assignment: Module 06, Assignment 06

GitHub Link: <https://github.com/steptar/IntroToProg-Python-Mod06.git>

How to Use Classes and Functions to Create and Edit a File

# Understanding the Task

Before creating the script, I made sure I understood the task at hand. Assignment 06 requires the program to collect and manipulate user input data to create a To-Do List. The program gives the user the option to add data, delete data, save data, reload original data, and exit the program. The data is saved to a text file. A starter python file is provided. This assignment is similar to Assignment 05, however this time the program tasks are accomplished through a variety of functions. The assignment requires this code to be run in PyCharm and the OS console window.

# Creating the Script

For this program, I created the code in PyCharm using the Assignment06 starter file provided. I edited the header change log to capture my changes to the code. I used the pseudo-code to guide me through creating the code for the various tasks the program is supposed to perform. The overall objective of this assignment is to create a “To-Do” list in a separate file from Python. The user is given a “Menu of Options” while running this program:

1) Add a new task

2) Removing an existing task

3) Save data to file

4) Reload data from file

5) Exit Program

The “Menu of Options” is displayed within a while loop and is displayed after each choice the user makes. Th while loop is the main body of the script. The while loop continues to display these options until the user chooses user exits the program. Some of the tasks are already accomplished through the pre-written code in the starter file. I walked through each option in the menu as if I was the using the program in order to complete the code for each section. The functions to complete these tasks are contained within the “Processor” class (performing tasks with the data) and the “IO” (taking inputs and providing outputs) classes.

## Initial Processing

The code to initially import the To-Do List into a “list of rows” or “lstTable” to work with the data was already pre-written into the starter module file. This is accomplished by calling the “read\_data\_from\_file()” function within the “Processor” class. The current data contained within lstTable is displayed each time the menu of options is displayed.

## Option 1: Adding New Data

New data is added to the list table by collecting new user inputs for the task and priority from the input\_new\_task\_and\_priority() function in the IO class. The values are assigned to a dictionary row task and key through the add\_data\_to\_list() function in the Processor task. The new dictionary row is appended to the list table. The script for Option 1 is shown in Figure 1 and the IO and Processing functions are shown in Figures 2 and 3, respectively.

*Figure 3. Function to add new data to the list of rows*

def add\_data\_to\_list(task, priority, list\_of\_rows):  
 row = {**"Task"**: str(task).strip(), **"Priority"**: str(priority).strip()}  
 list\_of\_rows.append(row)  
 return list\_of\_rows, **'Success'**

*Figure 2. New task and priority collection function*

*Figure 1. Option 1 while loop script*

def input\_new\_task\_and\_priority():  
 task = input(**"Enter a task: "**)  
 priority = input(**"Enter the priority: "**)  
 return task, priority  
 pass

if strChoice.strip() == **'1'**: *# Add a new Task  
 # IO sequence collects a new task and priority, Processor adds the new data to the lstTable* strTask, strPriority = IO.input\_new\_task\_and\_priority()  
 lstTable, strStatus = Processor.add\_data\_to\_list(strTask, strPriority, lstTable)  
 IO.input\_press\_to\_continue(strStatus)  
 continue *# to show the menu*

## Option 2: Removing Data

A task can be removed by selecting Option 2. The input\_task\_to\_remove() in the IO class collects the task to be removed. This task and the current list of rows is passed to the remove\_data\_from\_list() function in the Processor class. The program loops through each dictionary row in the list table and checks if strRemove matches the value in any of the “Task” keys in each of the dictionary rows. If there is a match the row is deleted and a message to the user is printed that indicates the task was deleted. If the task is not found, the program will print a message that the task is not in the list table and no removals will take place. The script for Option 2 is shown in Figure 4 and the IO and Processing functions are shown in Figures 5 and 6, respectively.

*Figure 4. Option 2 while loop script*

elif strChoice == **'2'**: *# Remove an existing Task  
 # IO sequence collects task, Processor sequence removes the task and updates the lstTable* strRemove = IO.input\_task\_to\_remove()  
 lstTable, strStatus = Processor.remove\_data\_from\_list(strRemove, lstTable)  
 IO.input\_press\_to\_continue(strStatus)  
 continue *# to show the menu*

*Figure 6. Function to delete a task from the To Do list*

def remove\_data\_from\_list(task, list\_of\_rows):  
 for dicRow in list\_of\_rows:  
 if task in dicRow[**"Task"**]:  
 list\_of\_rows.remove(dicRow)  
 print(**"Task deleted."**)  
 elif task not in dicRow[**"Task"**]:  
 print(**"Task not found."**)  
 return list\_of\_rows, **'Success'**

*Figure 5. The function that collects the task to be deleted*

def input\_task\_to\_remove():  
 task = str(input(**"Enter the task you wish to remove: "**)).lower().strip()  
 return task  
 pass

## Option 3: Saving Data

The list table data can be saved to the “ToDoList.txt” file by selecting Option 3 at any point in the program run. This option pass through the yes or no question as to whether or not to save the file to the input\_yes\_no\_choice() function in the IO class. If the user selects “y”, the file and list table is passed to the write\_data\_to\_file() function in the Processor class and the data is written to the text file. The input\_press\_to\_continue() function is used to continue the program. This function can also be used to cancel the save operation if the user selects “n” by not running the file writing function. The script for Option 3 is shown in Figure 7 and the input\_yes\_no\_choice() and the write\_data\_to\_file() functions are shown in Figures 8 and 9, respectively.

*Figure 8. The function that collects and returns a yes or no question*

def input\_yes\_no\_choice(message):  
 *""" Gets a yes or no choice from the user  
  
 :return: string  
 """* return str(input(message)).strip().lower()

*Figure 7. Option 3 Script*

elif strChoice == **'3'**: *# Save Data to File* strChoice = IO.input\_yes\_no\_choice(**"Save this data to file? (y/n) - "**)  
 if strChoice.lower() == **"y"**:  
 lstTable, strStatus = Processor.write\_data\_to\_file(strFileName, lstTable)  
 IO.input\_press\_to\_continue(strStatus)  
 else:  
 IO.input\_press\_to\_continue(**"Save Cancelled!"**)  
 continue *# to show the menu*

*Figure 9. The function that write the list table of to do items and priorities to the text file*

def write\_data\_to\_file(file\_name, list\_of\_rows):  
 file = open(file\_name, **"w"**)  
 for dicRow in list\_of\_rows:  
 file.write(dicRow[**"Task"**] + **","** + dicRow[**"Priority"**] + **"**\n**"**)  
 file.close()  
 return list\_of\_rows, **'Success'**

## Option 4: Reloading Data

Option 4 allows the user to reload the original data in the To-Do list file that was present when the program was first initiated. This can overwrite any new changes the user made during their program run. This option asks the user through the yes or no question function (Figure 8) if they would like to reload the original data. If the user answers “y” the list table will be reset to the original data by reading the data directly from the file. So, if the file hasn’t been saved and new options written to the file by selecting Option 3 then the original data can be re-obtained. The read\_data\_from\_file() function in the Processor class resets the lstTable value to the value from the text file. (This is the same function that called at the beginning of the program to get the initial text file values) The script for Option 4 is shown in Figure 10 and the read\_data\_from\_file() function is shown in Figure 11.

*Figure 11. The function to read data from the text file*

def read\_data\_from\_file(strFileName, list\_of\_rows):  
 *""" Reads data from a file into a list of dictionary rows  
 :rtype: object  
 :param file\_name: (string) with name of file:  
 :param list\_of\_rows: (list) you want filled with file data:  
 :return: (list) of dictionary rows  
 """* list\_of\_rows.clear() *# clear current data* file = open(strFileName, **"r"**)  
 for line in file:  
 task, priority = line.split(**","**)  
 row = {**"Task"**: task.strip(), **"Priority"**: priority.strip()}  
 list\_of\_rows.append(row)  
 file.close()  
 return list\_of\_rows, **'Success'**

*Figure 10. Option 4 Script*

elif strChoice == **'4'**: *# Reload Data from File* print(**"Warning: Unsaved Data Will Be Lost!"**)  
 strChoice = IO.input\_yes\_no\_choice(**"Are you sure you want to reload data from file? (y/n) - "**)  
 if strChoice.lower() == **'y'**:  
 lstTable, strStatus = Processor.read\_data\_from\_file(strFileName, lstTable)  
 IO.input\_press\_to\_continue(strStatus)  
 else:  
 IO.input\_press\_to\_continue(**"File Reload Cancelled!"**)  
 continue *# to show the menu*

## Option 5: Exiting the program

Finally, the user can exit the program by choosing Option 5. A final message is printed and the while loop is broken. This option does not write any new data or changes to the “ToDoList.txt” file. Option 3 must be selected to do this. The script for Option 5 is shown in Figure 12.

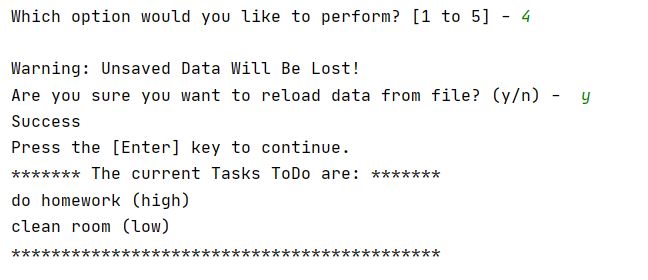
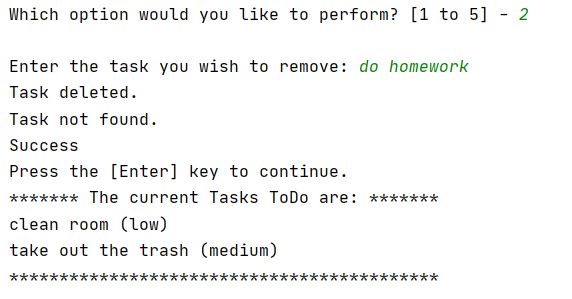
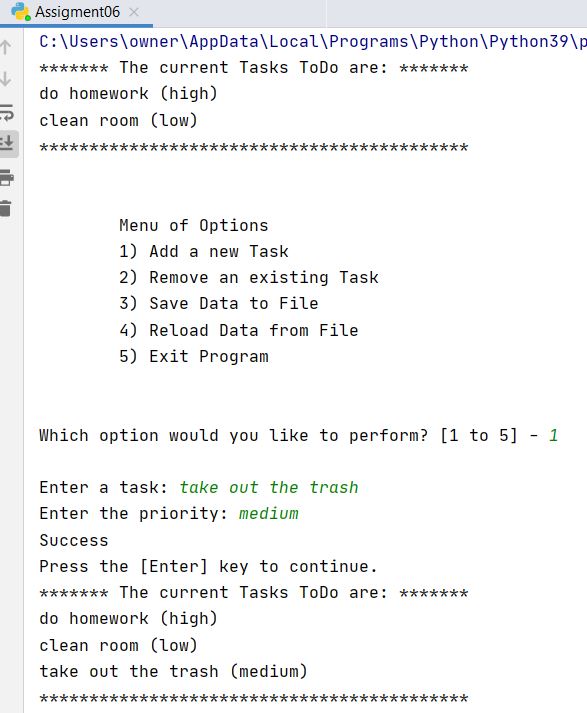
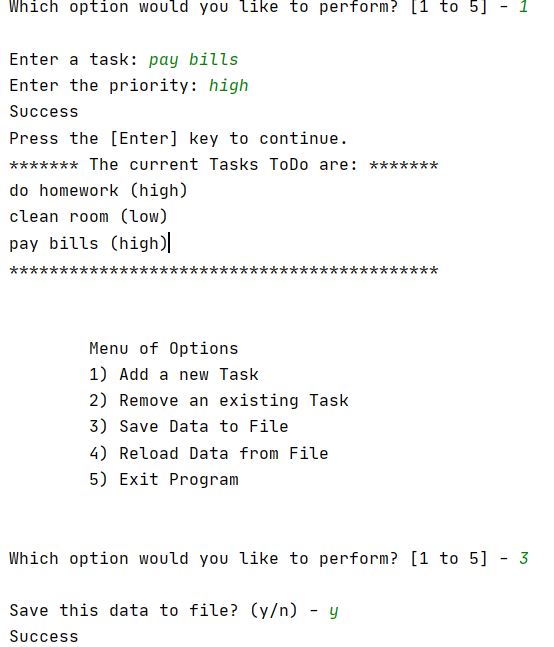
*Figure 12. Option 5 Script*

elif strChoice == **'5'**: *# Exit Program* print(**"Goodbye!"**)  
 break *# and Exit*

Please reference the Assignment06.py file to see the script in its entirety.

# Running the Code

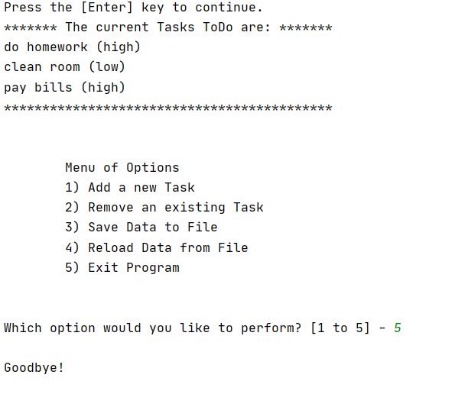
To run the code, I first tested the code in PyCharm. I ran the program, followed the prompts and verified that my inputs were written to the ToDoFile.txt file. The program worked in PyCharm as shown in Figure 13 (images are labeled 1-5 to show the order of steps taken) and wrote the final list to the ToDoList.txt file as shown in Figure 14.



2

3

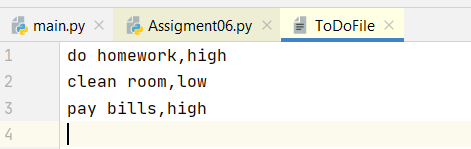
1



5

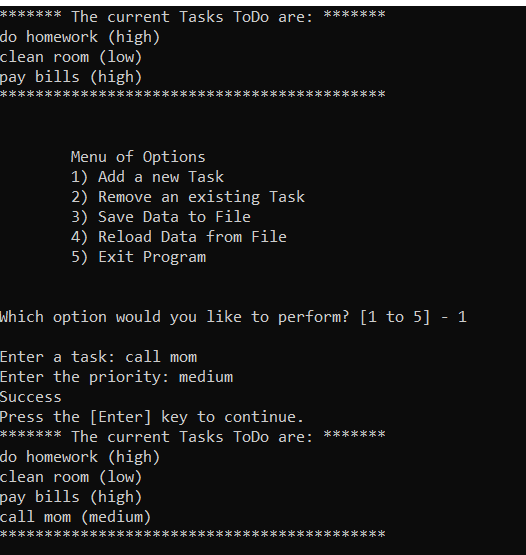
4

*Figure 13. Assignment 06 running PyCharm*

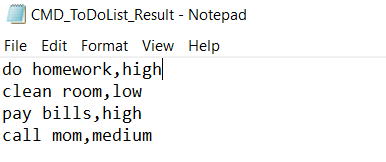


*Figure 14. ToDoFile.txt output from running Assignment0.Py in PyCharm*

After testing the code in PyCharm, I then ran the same program in the command console. The program can be seen running in Figure 15. I performed only one option just to demonstrate that the program can be run in the command console.



*Figure 15. Assignment06 running in the console window*



*Figure 16. ToDoFile.txt output from running Assignment06.Py in OS console window*

# Summary

To complete this assignment, I used the starter file for Assignment 06 provided and edited the script in PyCharm. I followed the prompts laid out in the pseudo-code and comments to accomplish the assignment tasks. A while loop in the main body of the script was used to give the user the options to add data to, remove data from, reload data from, and save data to a file until the user exits the program. The tasks were accomplished through various functions contained within the IO and Processor classes. This assignment demonstrated how to create and edit a to-do list text file using functions to break the tasks into logical and manageable sections.