**INTRODUCTION**

For the fifth assignment for IT FDN 110, students were tasked with creating a program that would take user input for a task and its priority and save it to a text file for later use and display, creating a “to-do list.” The objective was to store the data in a list of dictionaries while the code was running and then save the data from this structure to the file if the user chose to do so. This document will describe the process I used to successfully create this code. Note that some of the code was written by R. Root. His contributions are given in the header portion of the code itself.

**CODE WALKTHROUGH & REASONING**

Figure 1 shows the code written to perform the specified function. The first several lines encompass the header of the code, which is self-explanatory and will not be covered in detail in this document. The rest of the code will be explained by section: Data, Processing, and I/O.

**DATA**

The first section of code encompasses initialization of any variables that will be used later in the code. Most items are initialized as empty variables, such as *dicRow* for an empty dictionary and *lstTable* for an empty table. The only initialized variable that has a value is *objFile*, which is a string that defines the name of the input and output text file for the stored to-do list. R. Root created a *strMenu* variable, however it was not used as the menu was hard-coded into the I/O section of the code.

**PROCESSING**

The second section of code loads existing to-do list task and priority items and appends them to a list of dictionary items. The to-do list file is opened in read-mode and then looped through, line-by-line. Each item is assumed to be comma-separated, which is true if the list was created by this program. Thus, each row is split into two variables, the first being the Task and the second being the Priority. Each of these items is fed into a dictionary item, which is then appended to the variable *lstTable*. This is the list that will be operated on by the user for adding, removing, or saving items to the text file. The text file is closed at the end of the section.

**INPUT/OUTPUT**

This section begins by printing a menu of 5 options: showing current data, adding a new task to the to-do list, removing an existing task from the to-do list, saving data to the text file, and exiting the program. The user is prompted for a choice and the code will act based on this input. If no valid input is received, then a print statement will alert the user that their selection was not valid. The entirety of this section is wrapped in an infinite while loop to ensure the user will be able to make another selection until they explicitly choose to exit the program by selecting option 5, which contains a break statement to end the infinite loop and program.

Upon choosing option 1, a summary of existing tasks is printed out to the user. This is done by looping through the *lstTable* variable, which should contain all to-do list data (pre-existing or entered by the user in the current session). As each row in *lstTable* is a dictionary item, for each iteration of the loop, the Task and Priority are printed to the user by using the Task and Priority keys, which store their respective values per line item. After all items are displayed, the loop returns to the top via the *continue* statement.

If option 2 is selected, the user is immediately prompted for inputs for task name and priority. These inputs are fed directly into a dictionary item, which is then appended to the *lstTable* variable. This is very similar to the “PROCESSING” section, but instead of the data source being the “ToDoList.txt” file, it is the user’s inputs. The code then returns to the top of the loop.

If the user selects option 3, they must further confirm they wish to delete an item from the list with a separated “Y” or “N” input. This section is wrapped in a separate infinite loop from the main loop. If “N” is selected, the user returns to the top of the main loop via break statement. If neither option is chosen, then the user is returned to the top of the sub-loop to input again. If “Y” is selected, the user is asked to supply the name of a task to remove from the list. The code initializes two variables, *i* and *flag*, which are used to determine if the desired task is found in the list. The code loops through all of the rows in *lstTable* and unpacks the Task from each row. If the name of the task matches the user’s input for task-to-remove (non-case sensitive), then *flag* becomes 1 and the *i* counter is incremented. The code calls the “remove” method to delete the selected row of data. At this point, the code continues to loop through all of the remaining rows of data. If the counter *i* reaches the end of *lstTable* (evidenced by *i ==len(lstTable)*), and *flag* has not been set to 1 (meaning the user input for the task to delete never matched an option in *lstTable*), then the code prints out a statement that the desired task was not found. Regardless of the outcome, control is returned to the outer, main loop via break statement.

Upon selecting option 4, the to-do list text file is opened in write mode. This will cause the file to overwrite, and thus only the data in *lstTable* is written. The code loops through all data in *lstTable* and for each row, a line is written to the text file in the format “[Task],[Priority]” with a trailing newline character to move the cursor to the next line. At the end of the loop, a statement is printed to the user that data has been saved, and then the to-do list text file is closed.

Upon selecting option 5, the program prints a statement stating that the program will end upon the user hitting the “Enter” key. An input statement after this print statement ensures that it is the case. Upon striking “Enter,” a final break statement breaks the main infinite loop and ends the program.

Figures 2 through 9 show code operation in both PyCharm and the Windows command console. Figure 9 shows alternative options and what happens if the user enters a task to delete in option 3 that does not exist.

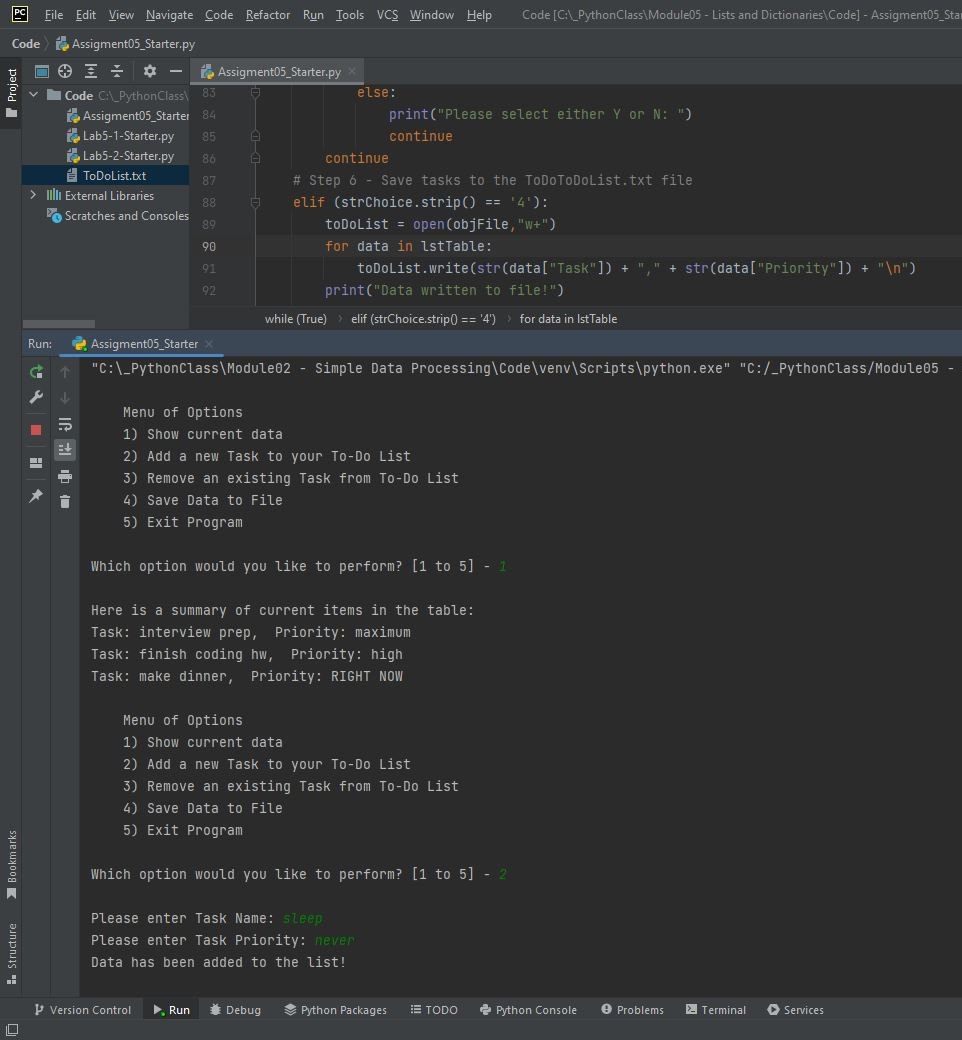
**SUMMARY**

In summary, this document describes the code behind a successful “to-do list” manager program that takes user inputs for a to-do list and each item’s priority and saves it to a “ToDoList.txt” file. Prior to asking if the user has any inputs, the code takes in existing to-do list data from the “ToDoList.txt” file, if it exists. All data is appended to a list of dictionary items (*lstTable).*

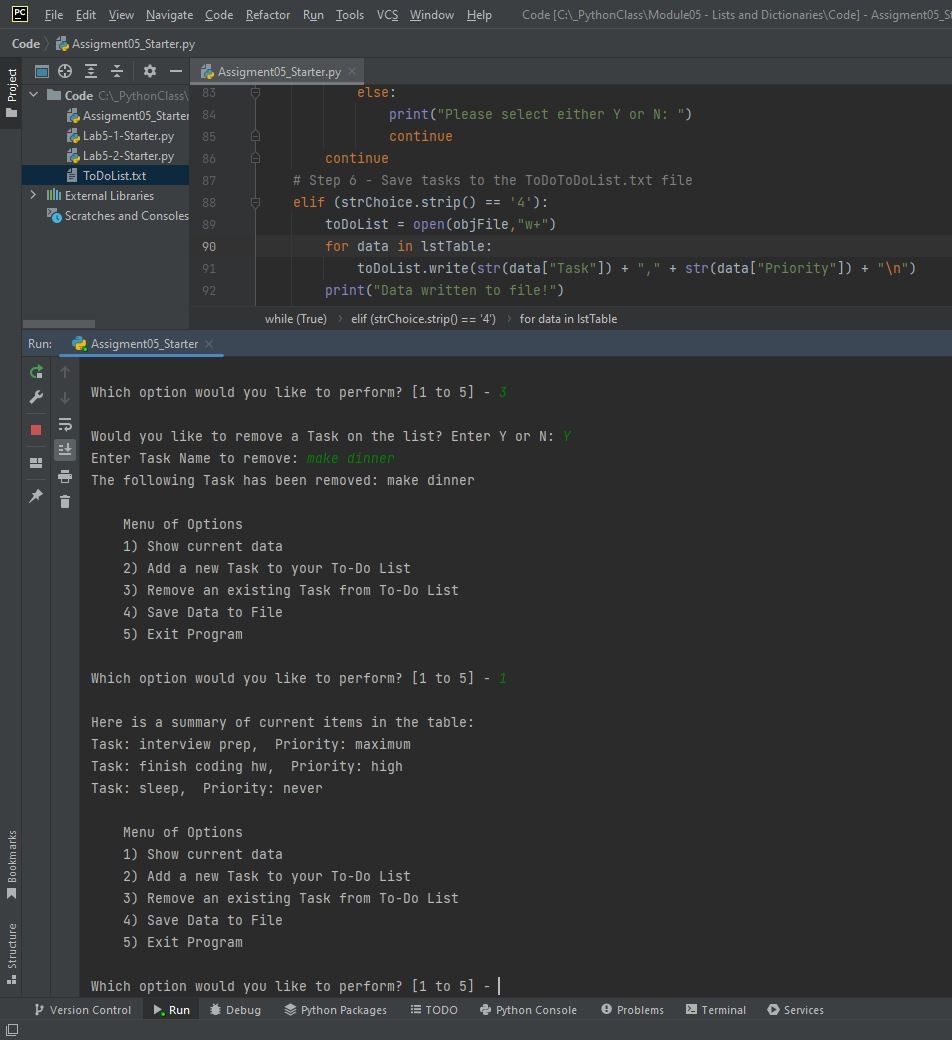
The code utilizes a menu structure to give the user the option to display existing tasks and priorities, add a new task and priority, remove an existing task and priority, save all data to a file, and exiting the program. This is all achieved via utilization of dictionary items that save each task and priority in a different row in the *lstTable* variable under the keys “Task” and “Priority.” When any specific data needs to be accessed, loops are used to unpack the data into individual task and priority items and displayed, written, or removed, or saved per user preference.

# ------------------------------------------------------------------------ #  
# Title: Assignment 05  
# Description: Working with Dictionaries and Files  
# When the program starts, load each "row" of data  
# in "ToDoList.txt" into a python Dictionary.  
# Add each dictionary "row" to a python list "table"  
# ChangeLog (Who,When,What):  
# RRoot,1.1.2030,Created started script  
# NSandhu,5/10/2022,Added code to begin assignment 5  
# NSandhu,5/15/2022,Finished main code, began testing  
# NSandhu,5/16/2022,Fixed following bugs: data not saving to text file, "data saved" printing multiple times,  
# ------------------------------------------------------------------------ #  
  
# -- Data -- #  
# declare variables and constants  
objFile = "ToDoList.txt" # An object that represents a file  
strData = "" # A row of text data from the file  
dicRow = {} # A row of data separated into elements of a dictionary {Task,Priority}  
lstTable = [] # A list that acts as a 'table' of rows  
# strMenu = "" # A menu of user options // THIS VARIABLE IS NOT USED, MENU IS HARD-CODED (NS, 5/10/22)  
strChoice = "" # A Capture of the user option selection  
  
# -- Processing -- #  
# Step 1 - When the program starts, load any data you have  
# in a text file called ToDoList.txt into a python list of dictionaries rows (like Lab 5-2)  
  
# Open list file in read mode; cursor position starts at the top of file and moves as it is read  
# Loop through all lines (assuming no header), assign data to dicRow, then append dicRow to lstTable  
toDoList = open(objFile, "r+")  
for strData in toDoList:  
 row = strData.split(",")  
 dicRow = {"Task": row[0], "Priority": row[1].strip()}  
 lstTable.append(dicRow)  
toDoList.close()  
  
# -- Input/Output -- #  
# Step 2 - Display a menu of choices to the user  
while True:  
 print("""  
 Menu of Options  
 1) Show current data  
 2) Add a new Task to your To-Do List  
 3) Remove an existing Task from To-Do List  
 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  
 # Step 3 - Show the current items in the table  
 if strChoice.strip() == '1':  
 print("Here is a summary of current items in the table:")  
 for row in lstTable:  
 print(f"Task: {row['Task']}, Priority: {row['Priority']}")  
 continue  
 # Step 4 - Add a new item to the list/Table  
 elif strChoice.strip() == '2':  
 dicRow = {"Task": input("Please enter Task Name: "), "Priority": input("Please enter Task Priority: ")}  
 lstTable.append(dicRow)  
 print("Data has been added to the list!")  
 continue  
 # Step 5 - Remove a new item from the list/Table  
 elif strChoice.strip() == '3':  
 userListDel = input("Would you like to remove a Task on the list? Enter Y or N: ")  
 while True:  
 if userListDel.lower() == "y":  
 userTaskDel = input("Enter Task Name to remove: ")  
 i = 0  
 flag = 0  
 for data in lstTable:  
 if data["Task"].lower() == userTaskDel.lower():  
 print(f"The following Task has been removed: {data['Task']}")  
 flag = 1  
 i += 1  
 lstTable.remove(data)  
 else:  
 i += 1  
 if i == len(lstTable) and flag == 0:  
 print("Task not found.")  
 break  
 elif userListDel.lower() == "n":  
 print("List unchanged. Returning to main menu...")  
 break  
 else:  
 print("Please select either Y or N: ")  
 continue  
 continue  
 # Step 6 - Save tasks to the ToDoToDoList.txt file  
 elif strChoice.strip() == '4':  
 toDoList = open(objFile, "w+")  
 for data in lstTable:  
 toDoList.write(str(data["Task"]) + "," + str(data["Priority"]) + "\n")  
 print("Data written to file!")  
 toDoList.close()  
 continue  
 # Step 7 - Exit program  
 elif strChoice.strip() == '5':  
 print("Program will end when 'Enter' is pushed...")  
 input()  
 break # and Exit the program  
 else:  
 print("This is not a valid menu option, please select again.")

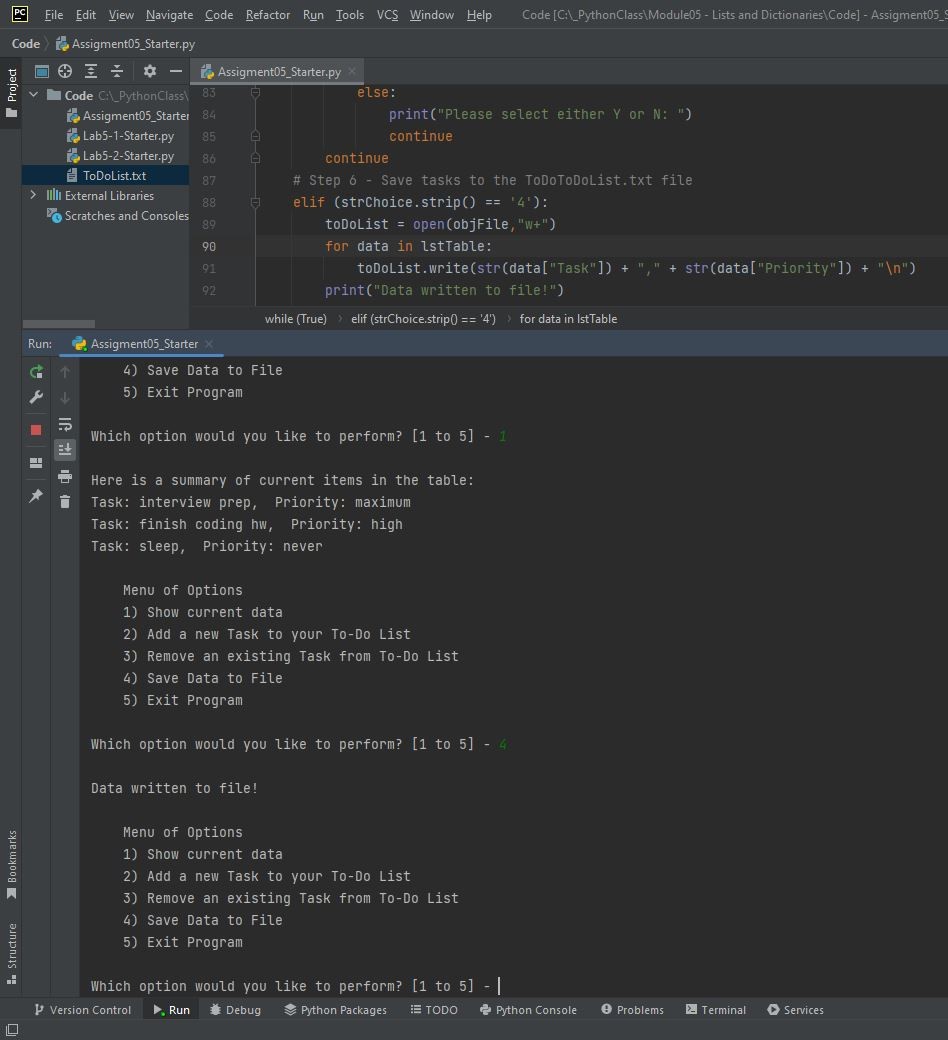
**Fig. 1** – Code written by both R. Root and N. Sandhu for creation of the to-do list manager program.



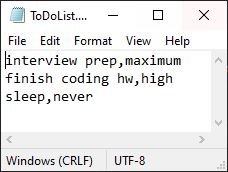
**Fig. 2.** – A task being added using PyCharm.



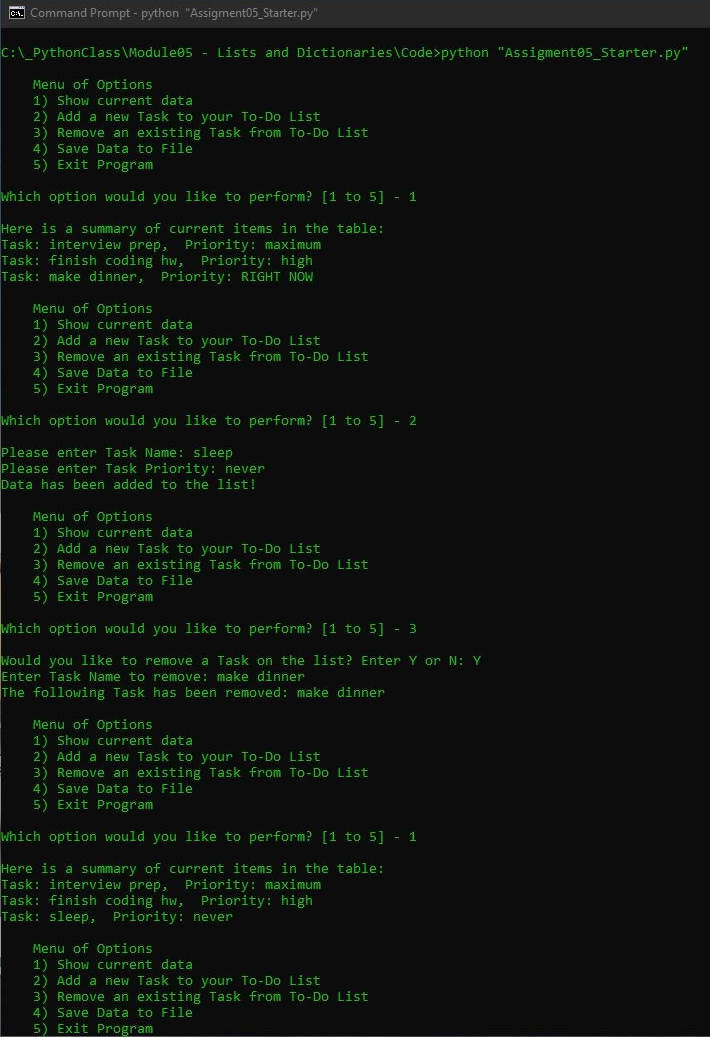
**Fig. 3.** – A continuation of Fig. 2. A task is removed using PyCharm.



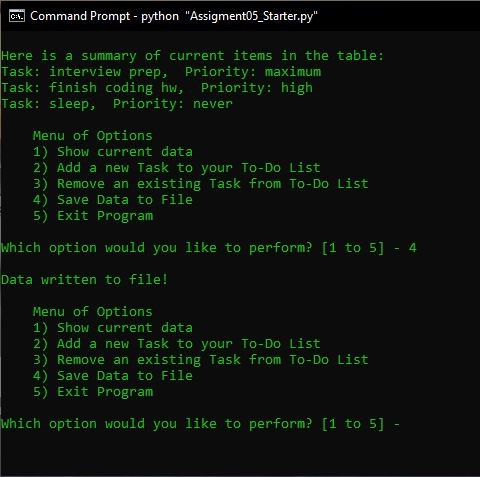
**Fig. 4.** – A continuation of Fig. 3. The to-do list is saved using PyCharm.



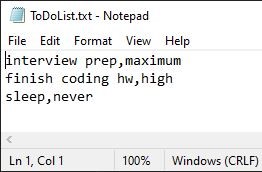
**Fig. 5.** – A continuation of Fig. 4. The resulting output file from all operations performed in Figs. 2 through 4. Note that the first two tasks were in a pre-existing “ToDoList.txt” file.



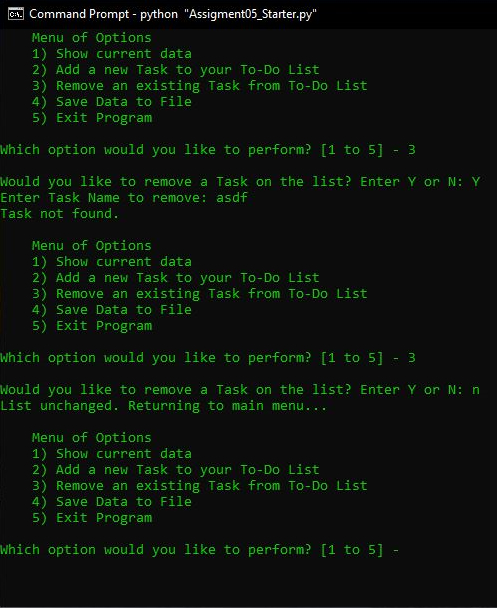
**Fig. 6.** – A task being added and a separated task being removed using the Windows Command Console.



**Fig. 7.** – A continuation of Fig. 6. The to-do list is saved using the Windows Command Console.



**Fig. 8.** – A continuation of Fig. 7. The resulting output file from all operations performed in Figs. 6 and 7. Note that the first two tasks were in a pre-existing “ToDoList.txt” file. This file is equivalent to Fig. 5, which was generated in PyCharm using the same commands.



**Fig. 9.** – An example of error handling and alternate options based on user selection. The first portion shows the result for a task to-be-deleted is not found. The second portion shows what happens if the user decides not to delete a task after selecting option 3.