Rex Sarabia

4 August 2022

IT FDN 110 B

Assignment05

GitHub: <https://github.com/rsar-uw/IntroToProg-Python> (External)

Python Script: To Do List v1.0

Table of Contents

[2 Introduction 2](#_Toc110958764)

[3 My system information 2](#_Toc110958765)

[3.1 Operating system (OS) 2](#_Toc110958766)

[3.2 Console application 3](#_Toc110958767)

[3.3 Shell 3](#_Toc110958768)

[3.4 Python 3](#_Toc110958769)

[3.5 Integrated Development Environment (IDE) 3](#_Toc110958770)

[3.6 Directory / File path 3](#_Toc110958771)

[4 Module assignment 4](#_Toc110958772)

[4.1 Requirements 4](#_Toc110958773)

[4.1.1 Out of scope 5](#_Toc110958774)

[4.1.2 Lessons learned 5](#_Toc110958775)

[4.2 Design 5](#_Toc110958776)

[4.2.1 Standard elements 6](#_Toc110958777)

[4.2.1.1 Code style 6](#_Toc110958778)

[4.2.1.2 Exceptions 6](#_Toc110958779)

[4.2.1.3 Script header 6](#_Toc110958780)

[4.2.1.4 Program start 6](#_Toc110958781)

[4.2.1.5 Saving files 7](#_Toc110958782)

[4.2.2 Program architecture 7](#_Toc110958783)

[4.2.2.1 Processing 8](#_Toc110958784)

[4.2.2.2 Menu 10](#_Toc110958785)

[4.2.2.3 Menu option 1: Show current data 11](#_Toc110958786)

[4.2.2.4 Menu option 2: Add a new item 14](#_Toc110958787)

[4.2.2.5 Menu option 3: Remove an existing item 16](#_Toc110958788)

[4.2.2.6 Menu option 4: Save data to file 18](#_Toc110958789)

[4.2.2.7 Menu option 5: Exit program 22](#_Toc110958790)

[4.2.3 Proposed solution 24](#_Toc110958791)

[4.3 Test 28](#_Toc110958792)

[4.3.1 Procedure 28](#_Toc110958793)

[4.3.2 Results 33](#_Toc110958794)

[4.4 Execution 34](#_Toc110958795)

[4.4.1 Terminal 34](#_Toc110958796)

[4.4.2 Results 38](#_Toc110958797)

[5 Summary 39](#_Toc110958798)

[6 References 39](#_Toc110958799)

[6.1 Schema 39](#_Toc110958800)

[6.1.1 Books 39](#_Toc110958801)

[6.1.2 Websites 39](#_Toc110958802)

[6.2 Sources 39](#_Toc110958803)

# Introduction

As each weekly cycle of this course passes, it is beginning to feel like a condensed sprint cycle. Documentation has become more predictable if not less tedious. Although I can usually get things to work, I have found that the exercise of documenting my rationale, approach to the assignment, and program behaviors has me feeling oddly reminiscent of my earlier life as an IT auditor. Considering so much of this assignment’s code is: (1) provided as part of the assignment and (2) copied / adapted from Assignment04, I would expect to spend less time troubleshooting like I normally would.

[Table of Contents]

# My system information

I do not expect to include this information in every assignment. The purpose of including this information is to serve as a reference documentation for future needs (e.g., troubleshooting).

Where applicable, the information below will be updated after significant system component version updates, and the addition of new or replacement of existing system components (i.e., deltas from this information will be recorded).

[Table of Contents]

## Operating system (OS)

**macOS Monterey version 12.4**

*For more information, refer to A01-RSar.docx – Section 3.1.*

[Table of Contents]

## Console application

**Terminal version 2.12.7 (445)**

*For more information, refer to A01-RSar.docx – Section 3.2.*

[Table of Contents]

## Shell

**zsh version 5.9**

*For more information, refer to A01-RSar.docx – Section 3.3.*

[Table of Contents]

## Python

**Python version 3.10.5**

*For more information, refer to A01-RSar.docx – Section 3.4.*

[Table of Contents]

## Integrated Development Environment (IDE)

**PyCharm version 2022.1.3 (Community Edition)**

*For more information, refer to A03-RSar.docx – Section 3.5.*

[Table of Contents]

## Directory / File path

Open Terminal

Navigate to Assignment05 working folder.

Enter the following command (see Figure 1):

|  |
| --- |
| cd documents/\_pythonclass/module05/a05rsarabia |

Figure . Command in Terminal for navigating to assignment directory

This folder and its contents will be compressed into .zip file and submitted as Assignment05.

Enter the following command in Terminal to return the directory path (see Figure 2):

|  |
| --- |
| pwd |

Figure . Command in Terminal to return directory path

Enter the following command in Terminal to return the directory content (see Figure 3):

|  |
| --- |
| ls -la |

Figure . Command in Terminal to return directory contents (visible and invisible files)

**/Users/rex/Documents/\_PythonClass/Module05/A05RSarabia** (see Figure 4)

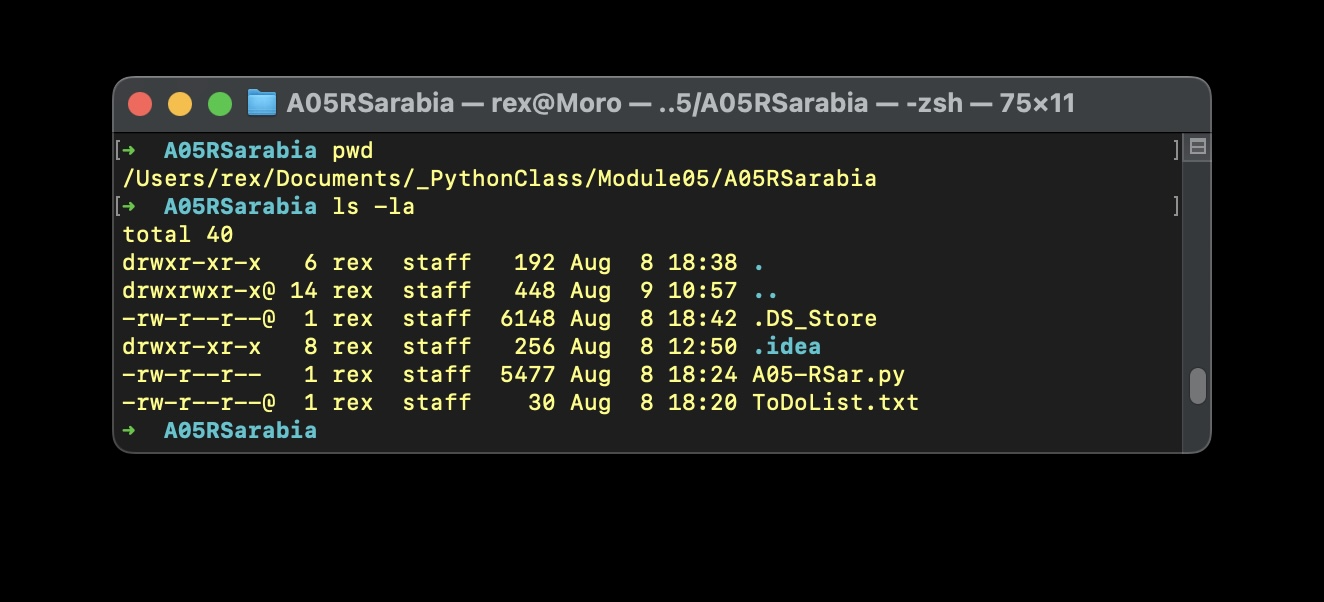


Figure . Screen capture of assignment directory and contents

Not seen in program files (see Figure 4) are the other Python (.py) files to serve as intermediary development pipelines (one for each major component of the final program file: menu, add, display, and save) allowing independent development and testing actions without interfering with other modules. For more information, see Section 4.2.2 (Program architecture).

[Table of Contents]

# Module assignment

## Requirements

Modify a new script that manages a “To Do list”.

The “To Do” file will contain two columns of data: (1) Task and (2) Priority.

Load the columns into a Python Dictionary object. Each dictionary object represents one row of data, and these rows must be added to a Python List object. Create a table of data (see Lab 5-2).

Modify the starting template (Assigment05\_Starter.py). Incomplete code needed for menu functions:

1. Show current items in the table (Step 3)
2. Add a new item to the list/Table (Step 4)
3. Remove a new item from the list/Table (Step 5)
4. Save tasks to the ToDoList.txt file (Step 6)
5. Exit program (Step 7)

Additional details of program requirements included in corresponding write-up, see Section 4.2.2.x.

(Randall R., Assignment05\_instructions.docx, Self-published, 2022).

[Table of Contents]

### Out of scope

For the purposes of this document, additional assignment tasks related to “Post your Files to GitHub” (Step 7) through “Perform Peer Review” (Step 10) are not included in this write-up.

[Table of Contents]

### Lessons learned

As weeks progress and assignments are graded, I thought it would be helpful to keep a log of comments received from prior assignments (Figure 5). The simplest benefit is to serve as a log of lessons learned along the way and more practically, as immediate reminders to not repeat the same mistakes. *Steel sharpens steel.*

| **Assignment filename** | **Comment** | **Penalties** |
| --- | --- | --- |
| A01-RSar.docx | end the file with  print(‘(Press Enter to End Program)’) #Conclusion line  input() | -2 |
| A02-RSar.docx | any figure# in a caption below a images/code snippet is referenced by figure # in the text written up above | -1 |
| A03-RSar.docx | Filename should be: HomeInventory.py | -2 |
| A04-RSar.docx | the save/exit option didn't ask me if i wanted to save to the file  Note: see Figure 5 - Assignment04\_instructions.docx | -2 |

Figure . Table of review comments from prior assignments

[Table of Contents]

## Design

This section includes the relevant components in the Python code that compose my proposed solution to the assignment.

[Table of Contents]

### Standard elements

#### Code style

By default, PyCharm will automatically perform checks of code style for PEP 8 (JetBrains, <https://www.jetbrains.com/help/pycharm/tutorial-code-quality-assistance-tips-and-tricks.html#df2e3bcf>, 2022) (External site). I made a conscious decision to follow the Problems/Solutions PyCharm identified, which in some cases conflict with the code style taught in class / prior assignments – honestly, it makes reading for errors in PyCharm a lot easier without having to mess with config settings of the feature.

For more information about PEP 8 code style guide: <https://peps.python.org/pep-0008/>

[Table of Contents]

#### Exceptions

Since I will be modifying “base code” provided, I have not changed the variable naming convention to keep code more comparable to the code submitted by classmates. In other circumstances, I will confirm to the code style specified in Section 4.2.1.1.

[Table of Contents]

#### Script header

In the script header (top of the script file), the following information must be recorded in the following format (Figure 6):

|  |
| --- |
| # ------------------------------- #  # Title: Assignment##  # Dev: RSar  # Desc: short description of script  # ChangeLog: (date,name,change)  # yyyy/mm/dd, Dev, Created File  # ------------------------------- # |

Figure . Source code for header

[Table of Contents]

#### Program start

Although not explicitly required, I’ve decided to begin all programs with the program title as the first line of display to the user. To ‘future-proof’ expansion on this feature, it will be declared as a variable and called in print() statement (Figure 7).

|  |
| --- |
| strProgramTitle = "To Do List v1.0" # Program name  print("\nWelcome to " + strProgramTitle + "!" # Display program name  "\n\n\tOpened file: " + objFile) |

Figure . Source code for program start

Since the base code for this program starts by defining variable for the data file (ToDoList.txt), I’ve modified the program start to include a message to the user that the data file has been opened. I manually created the text file to avoid errors if the file is opened in “read mode”.

[Table of Contents]

#### Saving files

“PyCharm automatically saves changes that you make in your files. Saving is triggered by various events, such as compiling, running, debugging, performing version control operations, closing a file or a project, or quitting the IDE.” (JetBrains, <https://www>.jetbrains.com/help/pycharm/saving-and-reverting-changes.html, 2022) (External site)

Per assignment requirements (Section 4.1), script filename and data output filename as follows:

* Script filename: **A05-RSar.py**
* Data output filename: **ToDoList.txt**
* Directory / file path: see Figure 4

[Table of Contents]

### Program architecture

I did not need to plan out the program holistically since the initial base code for this program had already been provided and we were explicitly required to work off that structure (e.g., not to use custom-defined functions). The following pseudocode is a modified version of the base code provided (Figure 8).

|  |
| --- |
| # \_\_Header  #  # \_\_Program start  #  # Data  #  # Processing  #  # Input / Output  #  # Menu  #  # Action 1 – Show current items  #  # Action 2 – Add a new item  #  # Action 3 – Remove item  #  # Action 4 – Save tasks to the ToDoList.txt  #  # Action 5 – Exit program |

Figure . Pseudocode to get the brain juices running

Based on the design as laid out in Figure 8, the basic structure will be similar to Assignment04:

* Put Menu in loop
* Inside the Menu loop, each input by user will trigger different action/ activity
* Actions 1 (display) and 4 (save) will be linear – after the action has been performed then user is automatically returned to the Menu loop
* Actions 2 (add) and 3 (remove) will need to run in their own respective loop – after user is done performing the action, then the user can choose to return to the Menu loop
* Action 5 (exit) will include additional prompt to confirm exit

Additionally, since many of the functions in this program are leveraged from Assignment04, statements will still need to be modified to accommodate the use of dictionary-type collection.

[Table of Contents]

#### Processing

*Requirement 1: When the program starts, load the data you have in a text file called ToDoList.txt into a Python list of dictionaries rows (like Lab 5-2).*

The program begins with declaring variables and then opens data file: ToDoList.txt (Figure 9). Additionally, although not required, I wanted to notify the user:

1. Data file was found and opened.
2. Data file already has information stored.

|  |
| --- |
| # -- Data -- #  # declare variables and constants  strProgramTitle = “To Do List v1.0” # Program name  objFile = “ToDoList.txt” # An object that represents a file  objFileData = “” # A count of no. of lines of records loaded  # from file lstData  dicRow = {} # A row of data separated into elements of a dictionary  # {Task,Priority}  lstTable = [] # A list that acts as a ‘table’ of rows  # -- Processing -- #  f = open(objFile, “r”) # open text file  print(“\nWelcome to “ + strProgramTitle + “!” # Display program name  “\n\n\tOpened file: “ + objFile)  objFileData = len(open(objFile).readlines()) # Counts rows in file  if objFileData >= 1: # if file has data  print(“\t” + str(objFileData) + “ row(s) of data found.”)  for row in f: # Add row from file to dictionary row  lstData = row.split(“,”)  dicRow = {“Task”: lstData[0], “Priority”: lstData[1].strip()}  lstTable.append(dicRow)  else: # if file does not have data  print(“\tNo data found.”) |

Figure . Source code for menu function

If there is already data in the file, then the program displays the number of rows of data found (Figure 10). Knowing that there is already data in the file, the user may be triggered to inspect what data is stored in the file (Option 1). Based on the data displayed via Option 1, the user may then choose to Add a new item (Option 2) or Remove an existing item (Option 3).

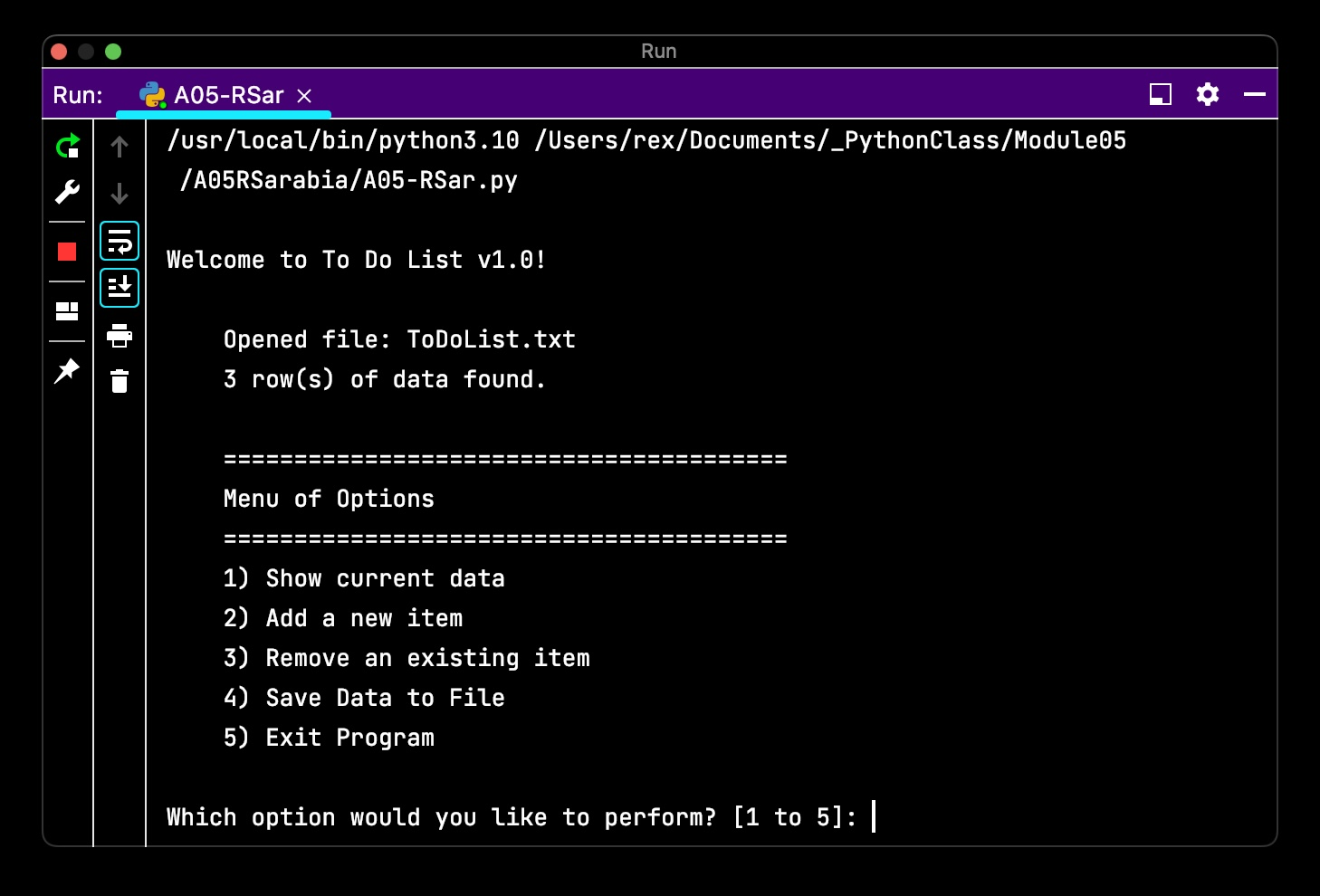


Figure . Screen capture of menu function

[Table of Contents]

#### Menu

*Requirement 2: Display a menu of choices to the user.*

Similar to Assignment04, Menu (and majority of the program is inside a while True: loop (Figure 11). Simplified the Menu statement by moving the string values of the menu to the variable strMenu. Additional formatting carried over from Assignment04 to better delineate between displaying data to the user versus Menu options (Figure 10).

|  |
| --- |
| # -- Data -- #  # declare variables and constants  strMenu = "\n\t" + "=" \* 40 + \  "\n\tMenu of Options" + \  "\n\t" + "=" \* 40 + \  """  \t1) Show current data  \t2) Add a new item  \t3) Remove an existing item  \t4) Save Data to File  \t5) Exit Program  """ # A menu of user options  strChoice = "" # A Capture the user option selection  # -- Input/Output -- #  # Step 2 - Display a menu of choices to the user  while True:  print(strMenu). # Display menu  strChoice = str(input("Which option would you like to perform? "  "[1 to 5]: ")) |

Figure . Source code for displaying menu options to user and prompt for user instruction

[Table of Contents]

#### Menu option 1: Show current data

*Requirement 3: Show the current items in the table*

Code for this requirement is very similar to Assignment04 with the exception that the for row in lstTable loop adds data from the data file into a dictionary collection before adding the dictionary to list of dictionaries: lstTable (Figure 12).

|  |
| --- |
| # Step 3 - Show the current items in the table  if strChoice.strip() == '1':  if len(lstTable) > 0:  print("\n\tDisplaying current data.") # Display tasks  print("\n\t" + "-" \* 40)  print("\tTask | Priority")  print("\t" + "-" \* 40)  for row in lstTable:  print("\t" + row["Task"], row["Priority"], sep=" | ")  print("\n\t/end of data")  else: # Alert user if there is no data collected  print("\n\tNo data found.")  continue |

Figure . Source code to display list table contents to user

After the about statement runs, the user is returned to the Menu via continue statement (Figure 13). I added additional text to notify the user when the list of data ends: print("\n\t/end of data"). Unlike Assignment04, I am prohibited from defining a custom reference for this function, however, this function would be a good example as it is reused several times throughout the program (i.e., option 3 and option 5).

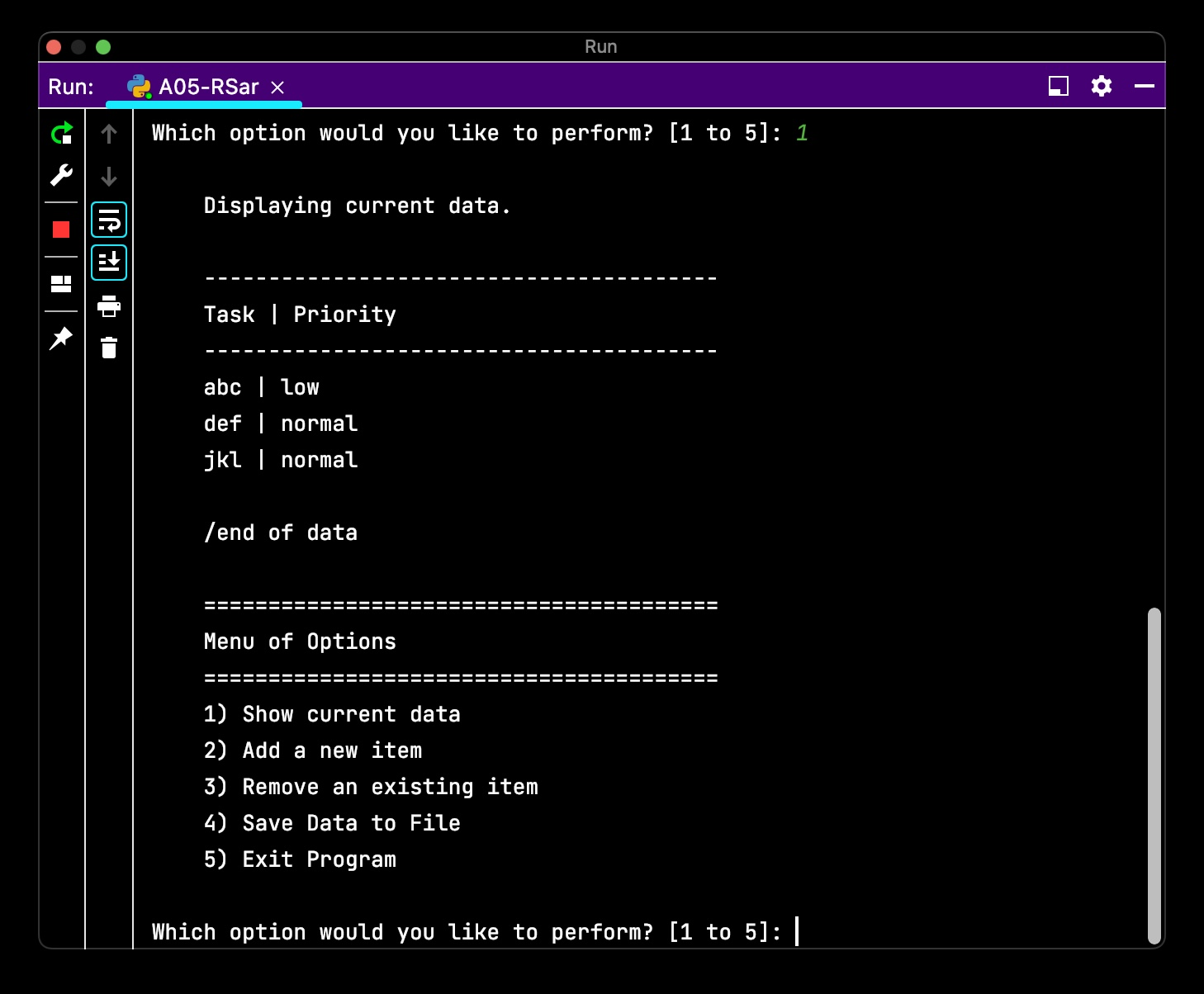


Figure . Screen capture of list table displayed to user

Similar to other functions in this program, what is displayed to the user is conditional on whether there is data in lstTable. If there is no data in lstTable then the user will be notified that there is no data found (Figure 14).

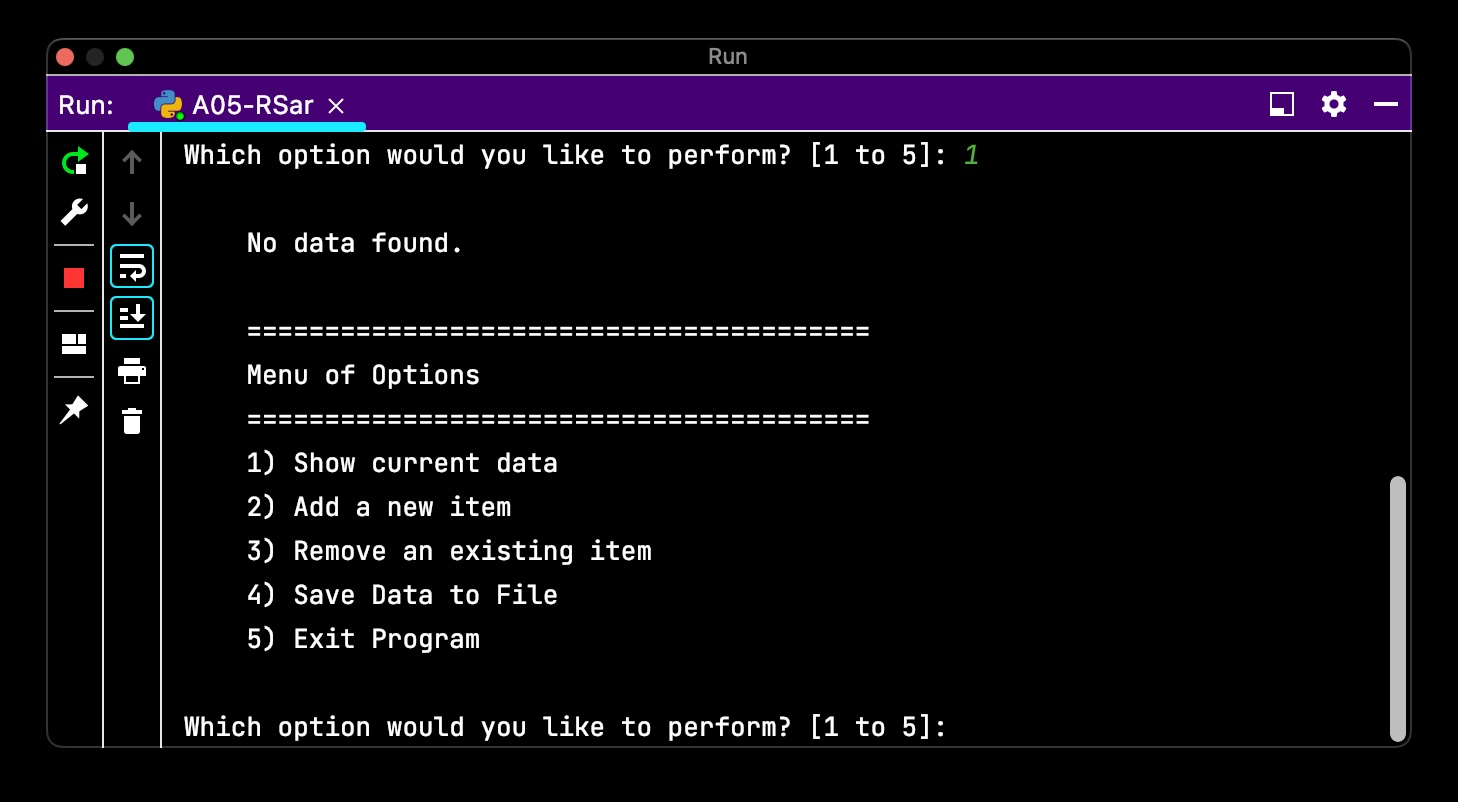


Figure . Screen capture of program display if list table is empty

[Table of Contents]

#### Menu option 2: Add a new item

*Requirement 4: Continue to ask user for more data until they return to the menu.*

Code for this requirement is very similar to Assignment04 with the exception that the values entered by the user are added to a dictionary collection dicRow = {"Task": strTask, "Priority": strPriority} before adding the dictionary to list of dictionaries: lstTable.append(dicRow) (Figure 15).

|  |
| --- |
| elif strChoice.strip() == '2':  while True: # Continue to ask user for more data until they  # return to the menu  # Instruct user and collect user input  print("\n\tEnter a Task and its Priority."  "\n\tEnter \"M\" at anytime to return to the Menu.")  strTask = input("\nWhat is the task? ")  if strTask.lower() == "m":  break  strPriority = input("What is the priority of task \'"  + strTask + "\' ? ")  if strPriority.lower() == "m":  break  # Store data in a two-dimensional list, which each item  # and price is a row of data  # Add user input as new list in table  dicRow = {"Task": strTask, "Priority": strPriority}  lstTable.append(dicRow)  continue |

Figure . Source code to write list table data to text file and quit the program

From the user’s perspective, the mechanics of this function are nearly identical to Assignment04 (Figure 14). The prompt for the user to input Household Item is replaced with Task and the prompt to input Estimated Value is replaced with Priority.

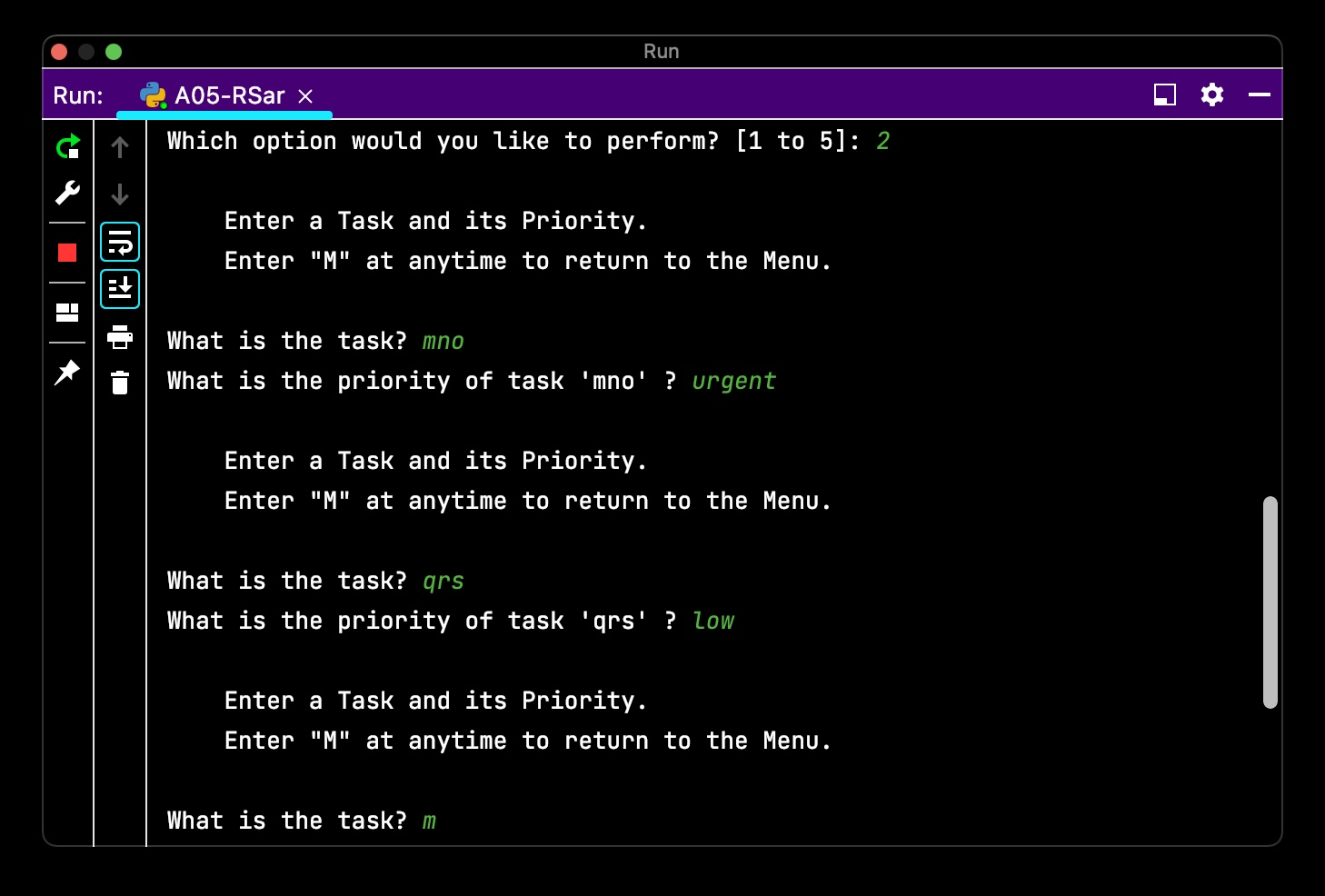


Figure 16. Screen capture of user adding data

The user can enter “M” or “m” into either prompt to return to the menu (Figure 17).

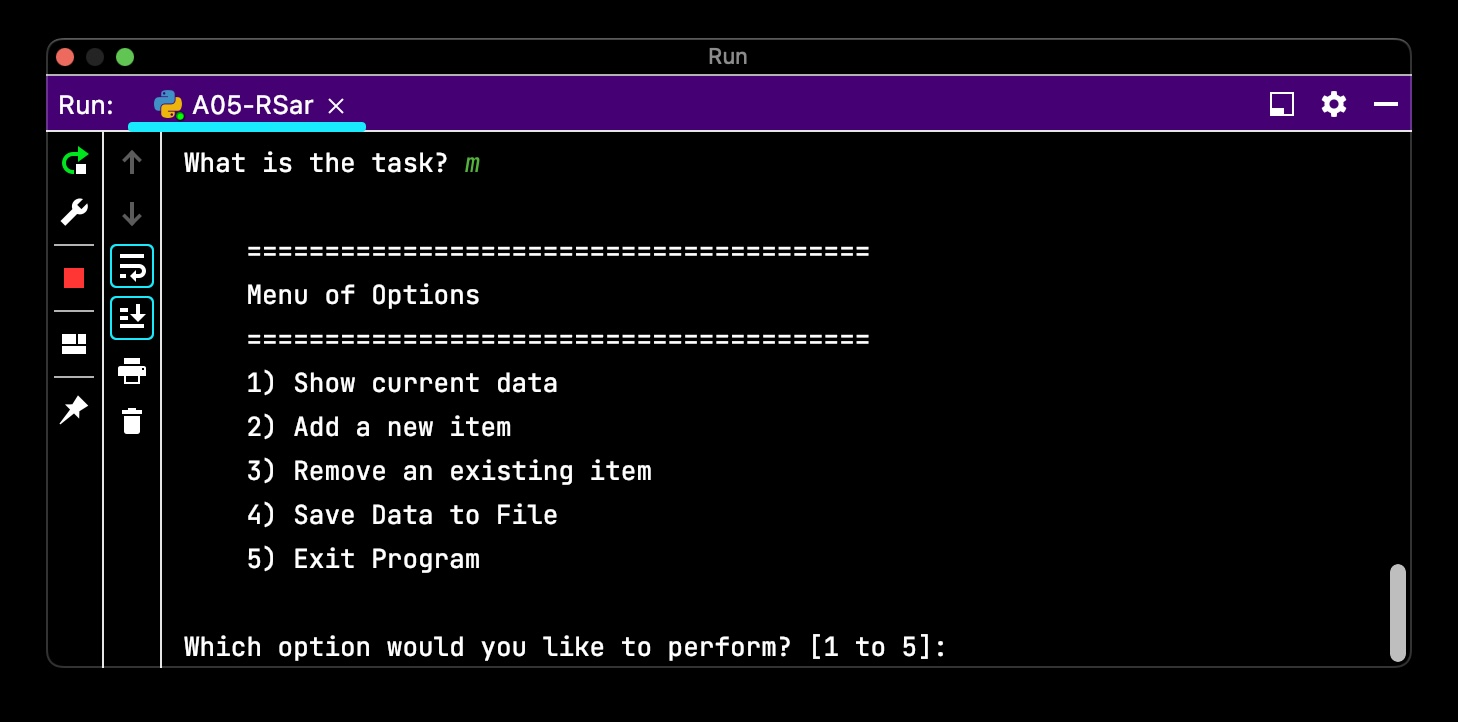


Figure 17. Screen capture of user returning to menu from option 1

[Table of Contents]

#### Menu option 3: Remove an existing item

*Requirement 5: Remove an item from the list/Table.*

Rather than limiting the user to only removing “new item”, I decided to enable the user to remove data that was already stored in the data file (Section 4.2.2.3) as well as any new data they had entered (Section 4.2.2.4). If I wanted to limit the user to removing only new items, then I would have created another list table for new items (e.g., lstTableNew), then when saving data, I would open the data file in “append mode”, open(“ToDoList.txt”, “a”). Additionally, I included the same code for displaying the data to improve “ease of use” of this function (Figure 18).

|  |
| --- |
| elif strChoice.strip() == '3':  while True:  if len(lstTable) > 0: # Display tasks  print("\n\tDisplaying current data.")  print("\n\t" + "-" \* 40)  print("\tTask | Priority")  print("\t" + "-" \* 40)  for row in lstTable:  print("\t" + row["Task"], row["Priority"],  sep=" | ")  print("\n\t/end of data")  print("\n\tEnter the task you want removed."  "\n\tEnter \"M\" to return to the [M]enu.")  r = input("\nWhat task do you want to remove? ")  if r.lower() == "m":  break  for row in range(len(lstTable)): # Remove task  if lstTable[row]["Task"].lower() == r.lower():  del lstTable[row]  print("\n\tRemoved task \'" + r + "\'.")  break  else:  print("\n\tNo data found.")  break  continue |

Figure . Source code to remove task from list

The user is notified of item removed if successful (Figure 19). This function loops until user navigates back to Menu or there are no more items to remove.

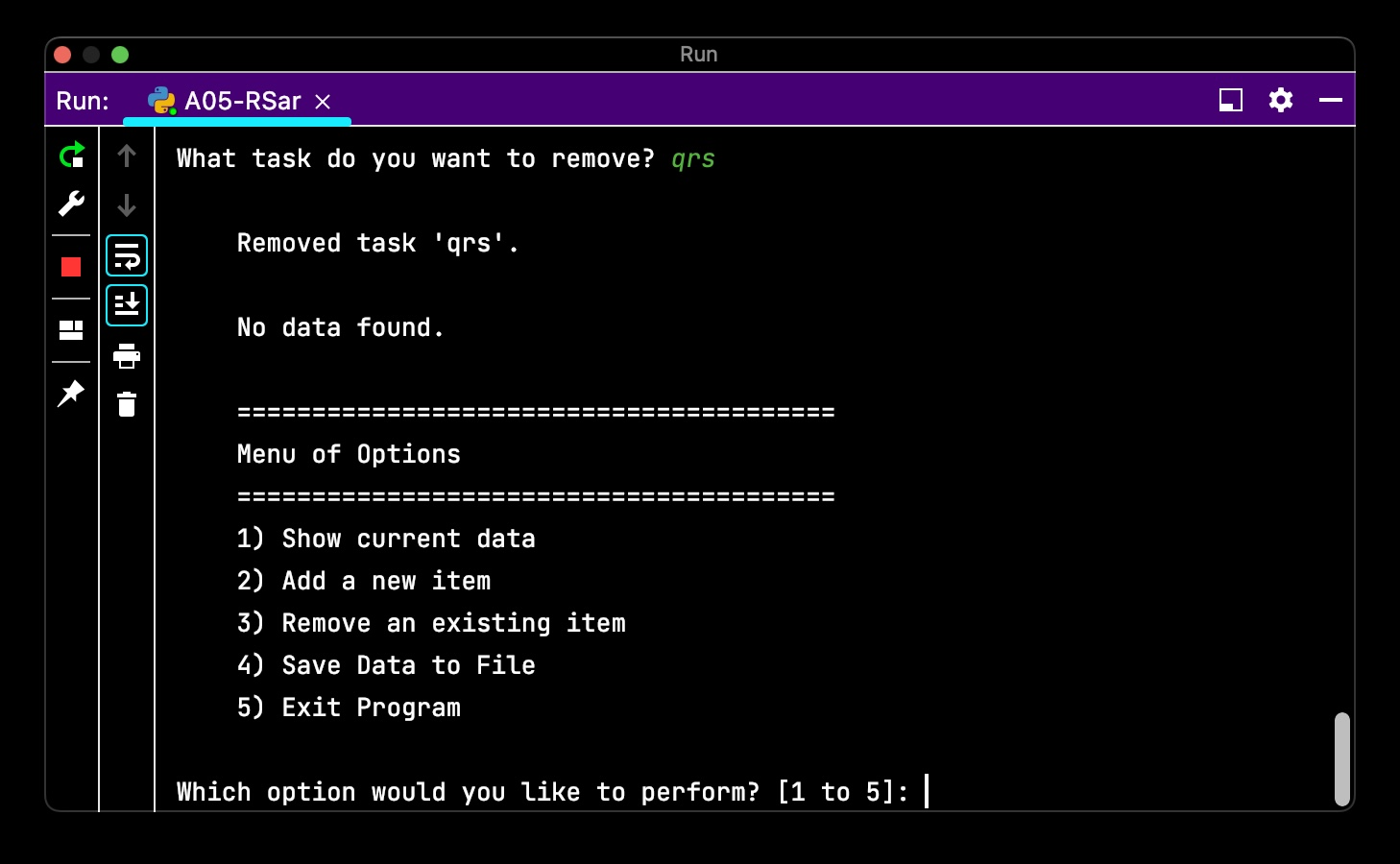


Figure 19. Screen capture of program writing list table data to text file and closing

[Table of Contents]

#### Menu option 4: Save data to file

*Requirement 6: Save tasks to the ToDoList.txt file.*

Unlike Assignment04, the Save and Exit functions of this program can be called by the user independent from one another. Similar to other functions (e.g., option 1 and option 3), I repeated the display function to the user, so they have a final opportunity to review their data before saving (i.e., destructive event) (Figure 20).

|  |
| --- |
| elif strChoice.strip() == '4':  if len(lstTable) > 0: # Condition: lstTable not empty  print("\n\tDisplaying current data.")  print("\n\t" + "-" \* 40)  print("\tTask | Priority")  print("\t" + "-" \* 40)  for row in lstTable:  print("\t" + row["Task"], row["Priority"], sep=" | ")  print("\n\t/end of data")  print("\n\tEnter \"S\" to [S]ave"  "\n\tPress ENTER key to return to the Menu.")  s = input("\nDo you want to save data to file? ")  if s.lower() == "s":  f = open(objFile, "w")  for row in lstTable:  f.writelines(str(row["Task"]) + "," +  str(row["Priority"]) + "\n")  f.close()  print("\n\tData saved to: " + objFile)  continue # Return to menu loop file data is saved  else:  print("\n\tNo data found.")  continue |

Figure . Source code to write list table data to text file

The program automatically displays the current data to the user before confirming if the user wants to save the data to file or return to the Menu (Figure 21). If the user chooses to save the data to file, then the program notifies the user of the filename where the data was written before returning the user to the Menu.

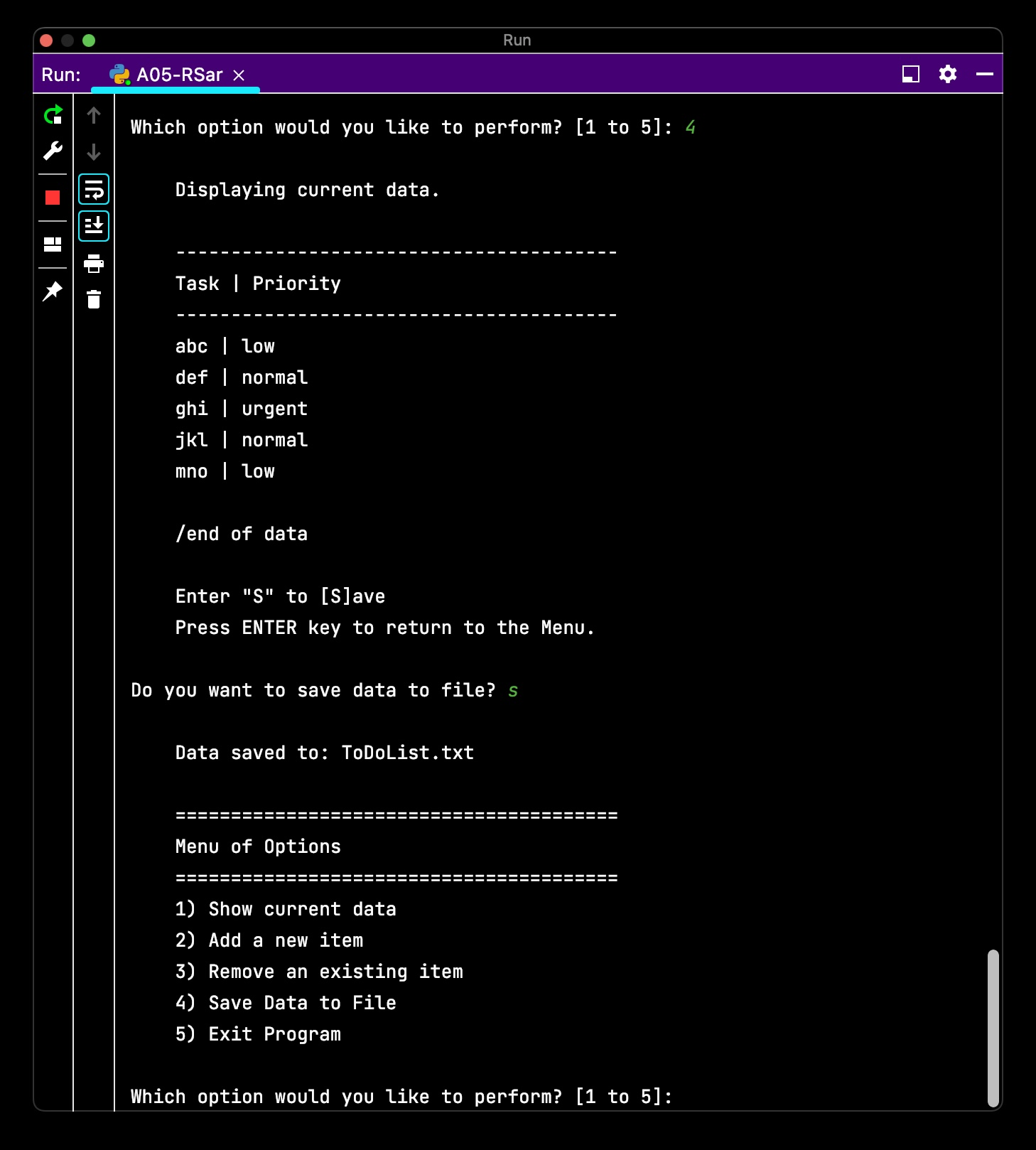


Figure . Screen capture of program writing list table data to text file and closing

Also similar to other functions where the display function is called, the “Save” function is effectively disabled (not performed) if there is no data stored in the program – the user is returned to Menu (Figure 22).

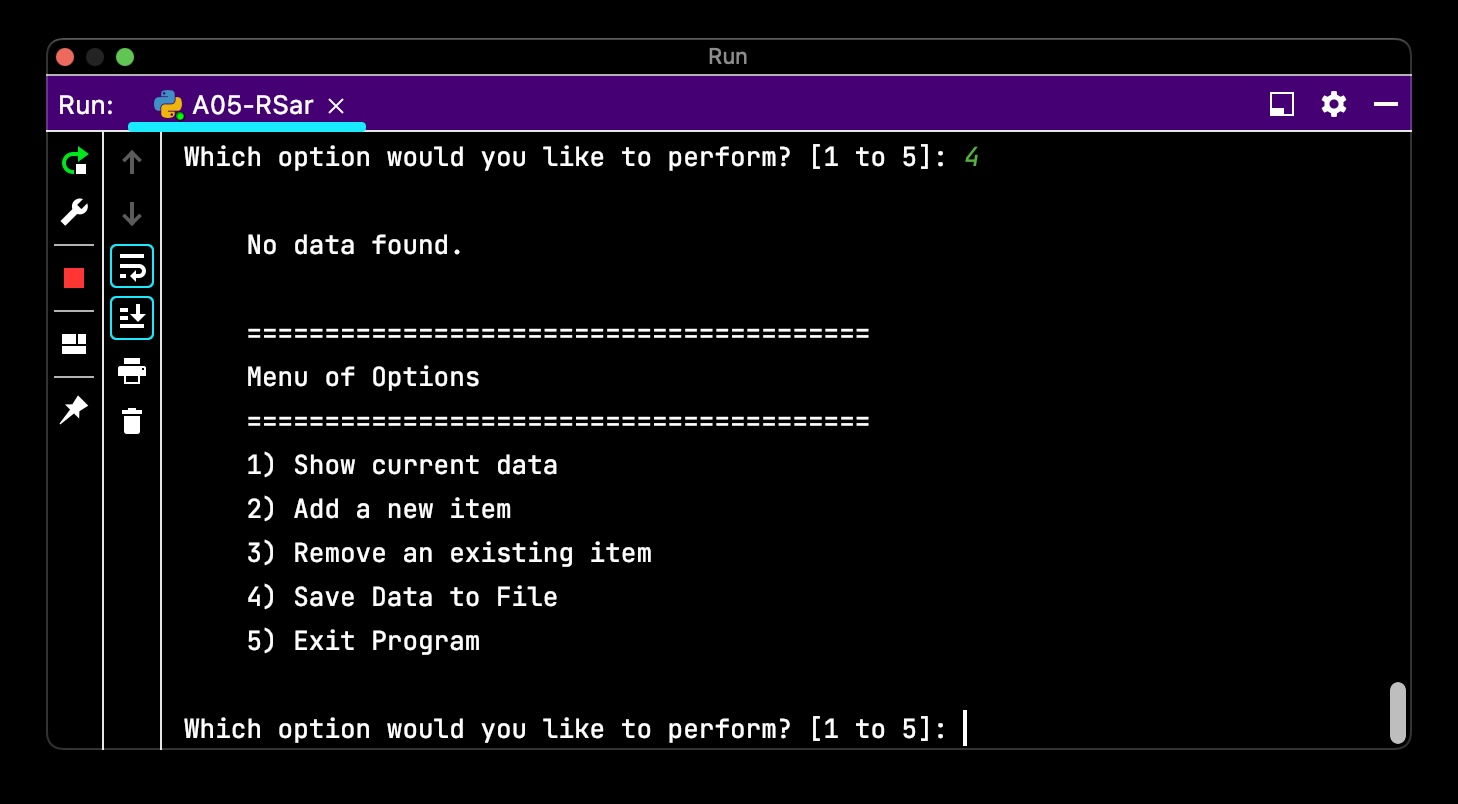


Figure . Screen capture if user selects Option 3 when there is no data entered

Similar to other functions, the user also the option to return to the menu without saving (Figure 23). For example, the user chooses to save, but upon seeing the current data displayed, realizes that they had forgotten to remove or add a task, then the user may want to go back to make the correction or update (or quit without saving).

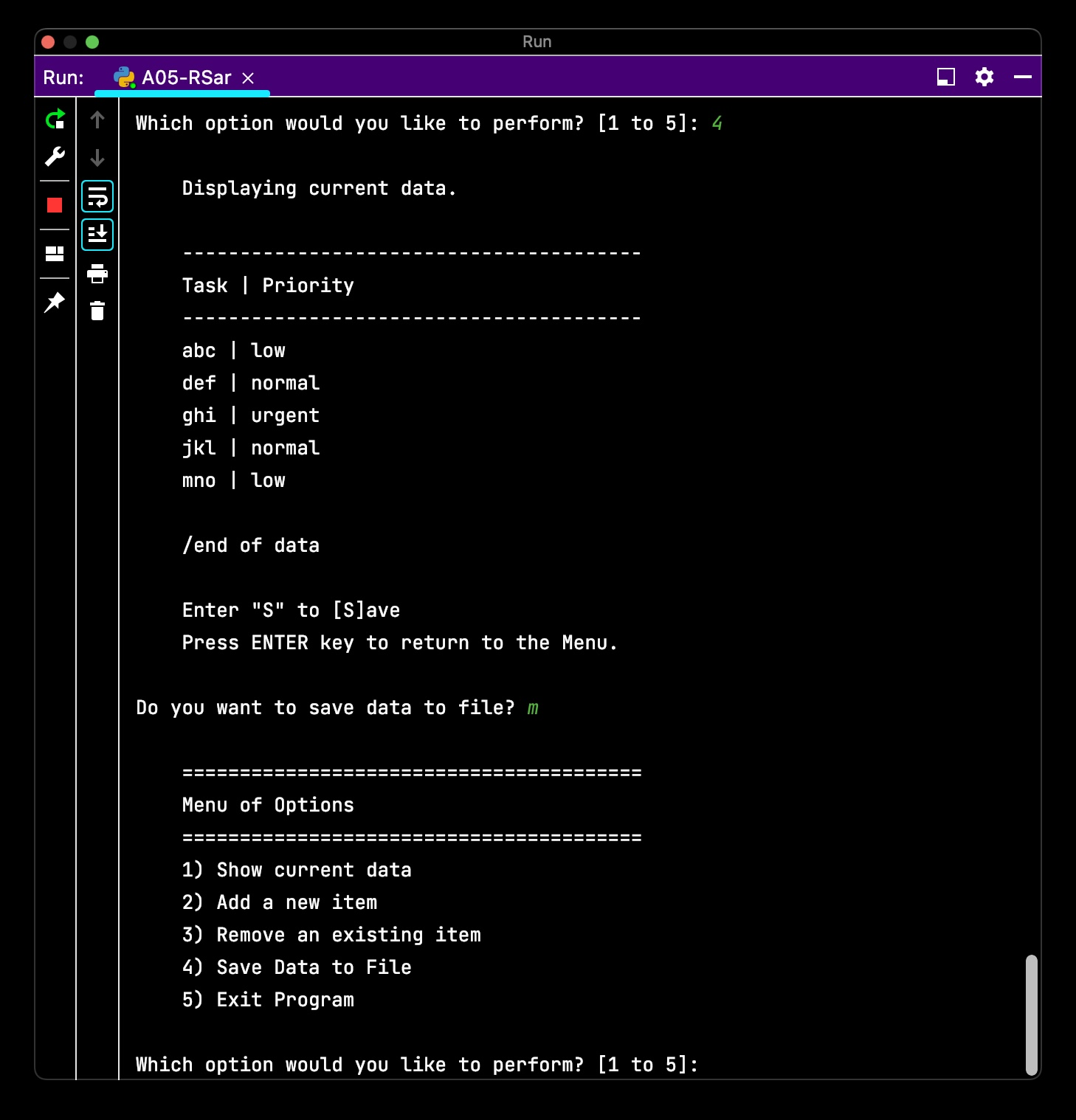


Figure . Screen capture if user selects Menu in option 4

[Table of Contents]

#### Menu option 5: Exit program

*Requirement 7: Exit the program*

The exit function essentially triggers the break statement for the while True: loop for the menu / program (Figure 24).

|  |
| --- |
| elif strChoice.strip() == '5':  print("\n\tEnter \"Q\" to [Q]uit"  "\n\tPress ENTER key to return to the Menu.")  c = input("\nAre you sure you want to quit? ")  if c.lower() == "q":  print("\n\tGoodbye!")  input("\n[Press the ENTER key to quit.]")  break # and Exit the program |

Figure . Source code to quit the program

When the user chooses to quit the program, final farewell message before interrupting the quit with a final input from the user (Figure 25).

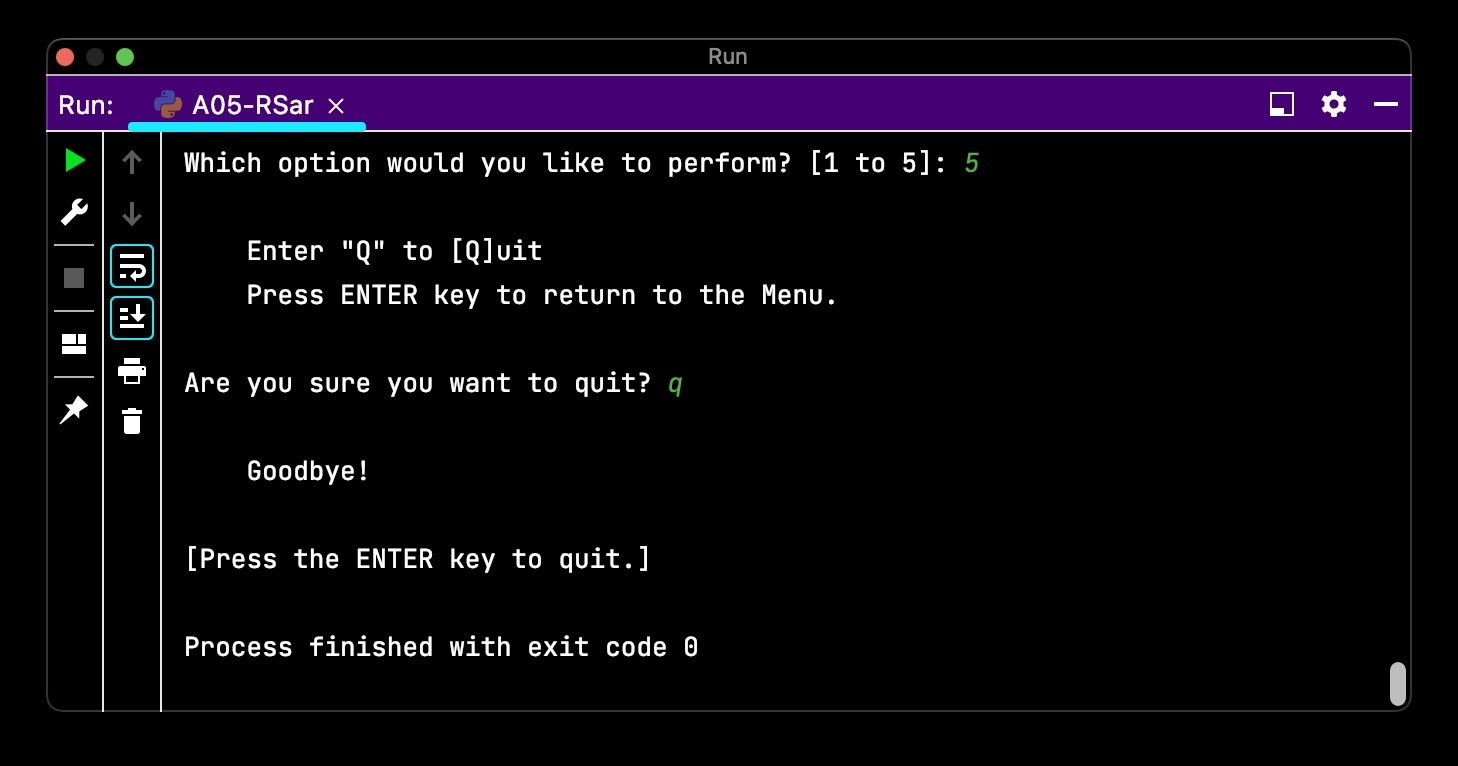


Figure . Screen capture if user chooses to Quit the program

Keeping the user experience as consistent as possible so performance of functions and navigation through the program becomes more predictable and efficient for the user. The user has the option to return to the menu before confirming they want to quit the program (Figure 26).

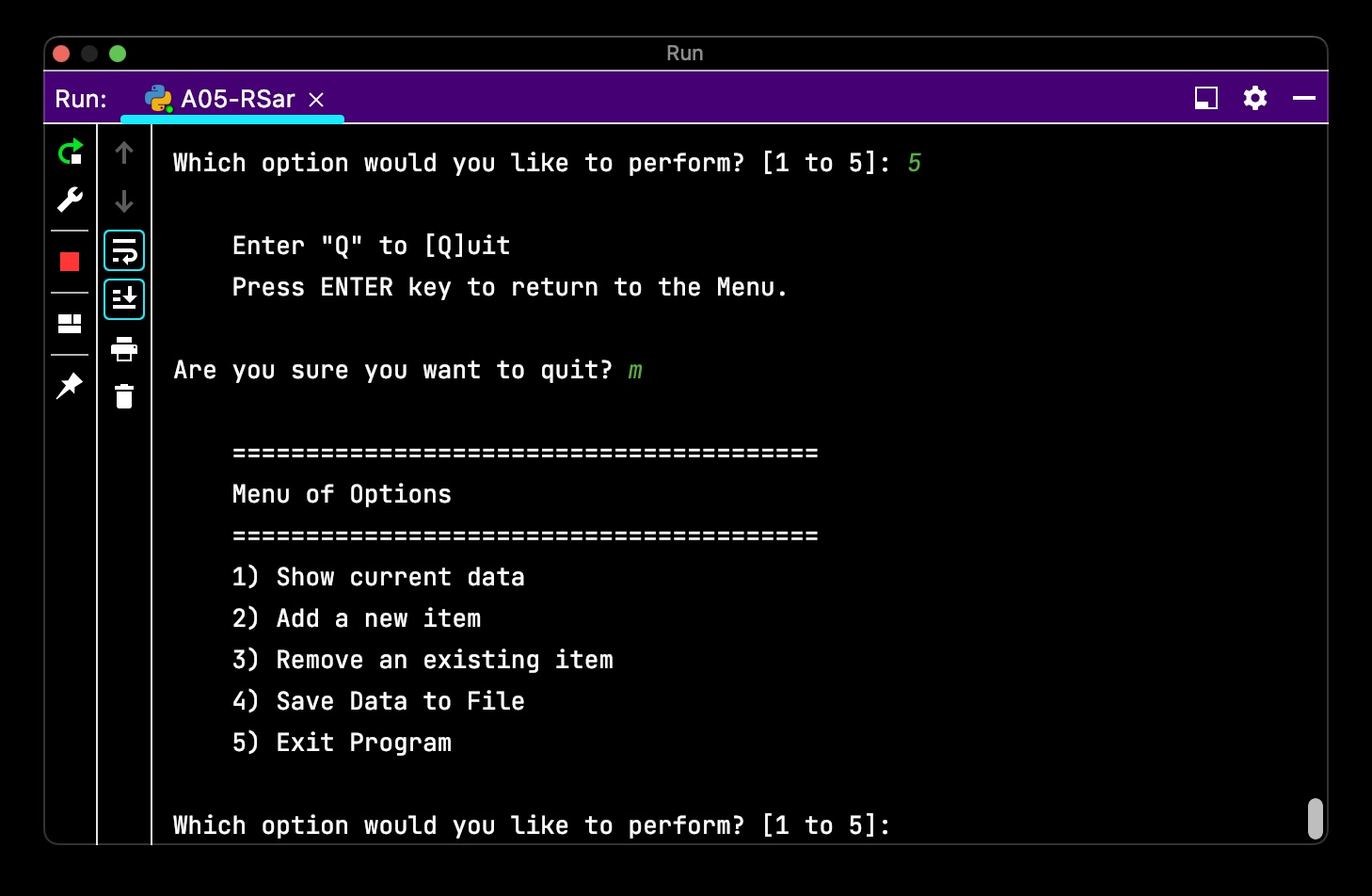


Figure . Screen capture if user selects Menu from option 5

**Future feature**: Prompt user to save file if they are exiting without saving. Implementation: Create counter array-variable[0] where save returns value to zero and other actions: (1) add, (2) remove will increment the counter +1. If variable is > 0, then prompt user to save.

[Table of Contents]

### Proposed solution

The following source code is my program for Assignment05 (Figure 27).

|  |
| --- |
| # ------------------------------- #  # Title: Assignment05  # Dev: RSar  # Desc: Working with Dictionaries and Files  # When the program starts, load each "row" of data in  # "ToDoToDoList.txt" into a python Dictionary.  # Add each dictionary "row" to a python list "table"  # ChangeLog: (when, who, what)  # 2022/01/01, RRoot, Created starter script  # 2022/08/08, RSar, Added code for each menu option  # 2022/08/09, RSar, Prepare program for submission  # ------------------------------- #  # -- Data -- #  # declare variables and constants  strProgramTitle = "To Do List v1.0" # Program name  objFile = "ToDoList.txt" # An object that represents a file  objFileData = "" # A count of no. of lines of records loaded  # from 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 = "\n\t" + "=" \* 40 + \  "\n\tMenu of Options" + \  "\n\t" + "=" \* 40 + \  """  \t1) Show current data  \t2) Add a new item  \t3) Remove an existing item  \t4) Save Data to File  \t5) Exit Program  """ # A menu of user options  strChoice = "" # A Capture the user option selection  # -- Processing -- #  # Step 1 - When the program starts, load the data you have  # in a text file called ToDoList.txt into a python list of dictionaries  # rows (like Lab 5-2)  f = open(objFile, "r") # open text file  print("\nWelcome to " + strProgramTitle + "!" # Display program name  "\n\n\tOpened file: " + objFile)  # noinspection PyRedeclaration  objFileData = len(open(objFile).readlines()) # Counts rows in file  if objFileData >= 1: # if file has data  print("\t" + str(objFileData) + " row(s) of data found.")  for row in f: # Add row from file to dictionary row  lstData = row.split(",")  dicRow = {"Task": lstData[0], "Priority": lstData[1].strip()}  lstTable.append(dicRow)  else: # if file does not have data  print("\tNo data found.")  # -- Input/Output -- #  # Step 2 - Display a menu of choices to the user  while True:  print(strMenu) # Display menu  strChoice = str(input("Which option would you like to perform? "  "[1 to 5]: "))  # Step 3 - Show the current items in the table  if strChoice.strip() == '1':  if len(lstTable) > 0:  print("\n\tDisplaying current data.") # Display tasks  print("\n\t" + "-" \* 40)  print("\tTask | Priority")  print("\t" + "-" \* 40)  for row in lstTable:  print("\t" + row["Task"], row["Priority"], sep=" | ")  print("\n\t/end of data")  else: # Alert user if there is no data collected  print("\n\tNo data found.")  continue  # Step 4 - Add a new item to the list/Table  elif strChoice.strip() == '2':  while True: # Continue to ask user for more data until they  # return to the menu  # Instruct user and collect user input  print("\n\tEnter a Task and its Priority."  "\n\tEnter \"M\" at anytime to return to the Menu.")  strTask = input("\nWhat is the task? ")  if strTask.lower() == "m":  break  strPriority = input("What is the priority of task \'"  + strTask + "\' ? ")  if strPriority.lower() == "m":  break  # Store data in a two-dimensional list, which each item  # and price is a row of data  # Add user input as new list in table  dicRow = {"Task": strTask, "Priority": strPriority}  lstTable.append(dicRow)  continue  # Step 5 - Remove an item from the list/Table  elif strChoice.strip() == '3':  while True:  if len(lstTable) > 0: # Display tasks  print("\n\tDisplaying current data.")  print("\n\t" + "-" \* 40)  print("\tTask | Priority")  print("\t" + "-" \* 40)  for row in lstTable:  print("\t" + row["Task"], row["Priority"],  sep=" | ")  print("\n\t/end of data")  print("\n\tEnter the task you want removed."  "\n\tEnter \"M\" to return to the [M]enu.")  r = input("\nWhat task do you want to remove? ")  if r.lower() == "m":  break  for row in range(len(lstTable)): # Remove task  if lstTable[row]["Task"].lower() == r.lower():  del lstTable[row]  print("\n\tRemoved task \'" + r + "\'.")  break  else:  print("\n\tNo data found.")  break  continue  # Step 6 - Save tasks to the ToDoToDoList.txt file  elif strChoice.strip() == '4':  if len(lstTable) > 0: # Condition: lstTable not empty  print("\n\tDisplaying current data.")  print("\n\t" + "-" \* 40)  print("\tTask | Priority")  print("\t" + "-" \* 40)  for row in lstTable:  print("\t" + row["Task"], row["Priority"], sep=" | ")  print("\n\t/end of data")  print("\n\tEnter \"S\" to [S]ave"  "\n\tPress ENTER key to return to the Menu.")  s = input("\nDo you want to save data to file? ")  if s.lower() == "s":  f = open(objFile, "w")  for row in lstTable:  f.writelines(str(row["Task"]) + "," +  str(row["Priority"]) + "\n")  f.close()  print("\n\tData saved to: " + objFile)  else:  print("\n\tNo data found.")  continue  # Step 7 - Exit program  elif strChoice.strip() == '5':  print("\n\tEnter \"Q\" to [Q]uit"  "\n\tPress ENTER key to return to the Menu.")  c = input("\nAre you sure you want to quit? ")  if c.lower() == "q":  print("\n\tGoodbye!")  input("\n[Press the ENTER key to quit.]")  break # and Exit the program |

Figure . Source code for my proposed solution to Assignment05

[Table of Contents]

## Test

### Procedure

For the purpose of this assignment, testing is performed in PyCharm IDE.

Open PyCharm

PyCharm > File > Open > directory path (Section 3.6) > A05-RSar.py

PyCharm > Run > “A05-RSar”

For this assignment and based on the limitations placed on the inputs of the user, I intend to limit my test cases to expected errors and a few valid input types (Figure 28).

| **Test flow ID** | **Test description** | **Actual output** | **Result** |
| --- | --- | --- | --- |
|  | With no data in data file, start the program and perform menu actions. | Open data file    Program start    . . .  // To limit redundant content, other menu options excluded.  . . .  Menu option 4: Save Data to File | Pass |
|  | Flow: Add tasks to Save | Enter data    Save file    View output data file | Pass |
|  | Flow: Remove task to Save | Remove a task    Save file    View output data file | Pass |

Figure 28. Summary of tests performed and results in PyCharm IDE

[Table of Contents]

### Results

Results from test cases all passed as the actual result matched what I was expecting for each of the value combinations input.

[Table of Contents]

## Execution

For the purpose of this assignment, execution is done via the Terminal which simulates executing in Production environment (PROD) as running the program in PyCharm IDE simulates testing environment (TEST).

[Table of Contents]

### Terminal

Open Terminal

Enter the following command (Figure 29):

|  |
| --- |
| python3 [file path][file name] |

Figure . Command for executing script in Terminal

Where:

file path: Figure 4

file name: Section 4.2.1.5

Re-perform test procedures (see Figure 27). For this assignment, the data inputted into HomeInventory.txt file has been erased and so the file is null.

| **Test flow ID** | **Test description** | **Actual output** | **Result** |
| --- | --- | --- | --- |
|  | With no data in data file, start the program and perform menu actions. | Open data file    Program start    . . .  // To limit redundant content, other menu options excluded.  . . .  Menu option 3: Remove an existing item | Pass |
|  | Flow: Add tasks to Save | Enter data    Save file    View output data file | Pass |
|  | Flow: Remove task to Save | Remove a task    Save file    View output data file | Pass |

Figure . Summary of tests performed and results in Terminal

[Table of Contents]

### Results

Using the same input values per **Section 4.3.1**, produced the same results as testing for both error messages of invalid input as well as correctly formatted and stored outputs of valid user inputs.

[Table of Contents]

# Summary

After five weeks of assignments submitted for this course, I think I have learned to appreciate how Python is both accessible yet infinitely complex. Learning Python reminds me of what a buddy had once told me about learning Texas Hold ‘Em Poker: five minutes to learn, a lifetime to master. I think learning Python has taken me a bit longer than five minutes and I have no illusions of mastery in one lifetime, but abstractly, I appreciate the way in which Python leads me to think. This assignment was challenging because of the new elements (e.g., working with dictionaries, GitHub, working off somebody else’s code), but if I have noticed a pattern, then it is the simple truth that the body of knowledge in learning a computer language is infinite and highly contextual. Independently, I doubt I would have reached this level of learning in the same amount of time. I do feel like the style and format of this course is a good shortcut to learning Python.

[Table of Contents]

# References

## Schema

### Books

“Quoted text“ (Author Last name Author First name initial., Title of book, Publisher, Year published)

### Websites

“Quoted text“ (Website name, URL, yearAccessed) (External site)

[Table of Contents]

## Sources

*Note: It is assumed that knowledge builds, therefore, duplicate sources already included in prior assignments have been removed – unless it has been directly referenced within this assignment.*

Sarabia R., A01-RSar.docx, Self-published, 2022

Sarabia R., A02-RSar.docx, Self-published, 2022

Sarabia R., A03-RSar.docx, Self-published, 2022

Sarabia R., A04-RSar.docx, Self-published, 2022

JetBrains, <https://www.jetbrains.com/help/pycharm/saving-and-reverting-changes.html>, 2022 (External site)

JetBrains, <https://www.jetbrains.com/help/pycharm/tutorial-code-quality-assistance-tips-and-tricks.html#df2e3bcf>, 2022 (External site)

Randall R., \_Mod5PythonProgrammingNotes.docx, Self-published, 2019

Randall R., Assigment05\_Starter.py, Self-published, 2022

GeeksforGeeks, <https://www.geeksforgeeks.org/python-removing-dictionary-from-list-of-dictionaries/>, 2022 (External site): Python | Removing dictionary from list of dictionaries

DelfStack, <https://www.delftstack.com/howto/python/python-search-list-of-dictionaries/>, 2022 (External site): Search a list of Dictionaries in Python

[Table of Contents]