Daniel Esporma

November 16th, 2021

IT FDN 110 A Foundations Of Programming: Python

Assignment05

<https://github.com/desporma/IntroToProg-Python>

Storing Collections of Data in Memory and Text Files Using Lists and Dictionaries

# Introduction

A dictionary is an option for storing a collection of data in a manner similar to a list, but with added functionality. Where the elements of a list are indexed by numeric characters, dictionaries can index elements by string characters defined by the author. Lists and dictionaries can store data in memory, but must write to or read from a text file to store data when not actively running a written program. In this assignment, we were tasked with writing a program that would read from a text file called ToDoList.txt, display a menu of options to a user, perform tasks on a table list dependent on the user’s input, write to the text file, and exit.

# Methodology

This assignment was broken down into three main components: Data, Processing, and Input/Output. The Data section detailed variables and their data types. The Processing section read the text file, extracted its elements, and populated a table. The Input/Output section presented the user with a menu of options and detailed the behavior of the program in response to user input.

The Data and Processing sections of Assignment05.py are decently simple. The Data section lists out all the variables to be put to use later in the script. As seen in Figure 1, data types are established though no values are assigned—dicRow is defined as a dictionary with its curly brackets and lstTable is defined as a list with its square brackets. objFile is the only variable where a value is defined; this enables easy changes to the script should the text file need to be changed or renamed.

# -- 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  
strChoice = "" # A Capture the user option selection

**Figure 1.** Variables were declared and left for definition later in the script.

The Processing section of Assignment05.py interacts with the text file. In order to extract data from the file, the file must first be opened and read (which is designated by the “r” in the third line of Figure 2) and its rows be separated with a for-loop. Within the for-loop, the elements of each row are split and stored as a string strData. The elements of strData then define the values that fall under the “Task” and “Priority” keys of the dicRow dictionary. The strip() function serves to remove a carriage return. Finally, the dictionary dicRow is appended to a table list lstTable. Once the for-loop ends, the text file is closed.

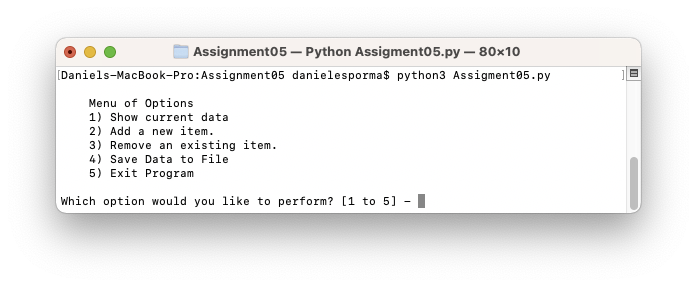
# -- Processing -- #  
# Read ToDoList.txt  
file = open(objFile, "r")  
for row in file:  
 # Break ToDoList.txt into separate rows  
 strData = row.split(",")  
 # Make elements of each row into elements of dictionary  
 dicRow = {"Task": strData[0], "Priority": strData[1].strip()}  
 # Append table with dictionary  
 lstTable.append(dicRow)  
# Close ToDoList.txt  
file.close()

**Figure 2.** The text file “ToDoList.txt” is read and its elements are used to populate the dicRow dictionary and append the lstTable table.

The Input/Output section of Assignment05.py is where the code for user interaction resides. As seen in line 2 of Figure 3a, the Input/Output section (Figures 3-8) resides within a while-loop. At the beginning of the loop, the user is presented with a menu of options. The user is then queried for a response using the input() command. This response defines strChoice.

# Step 2 - Display a menu of choices to the user  
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

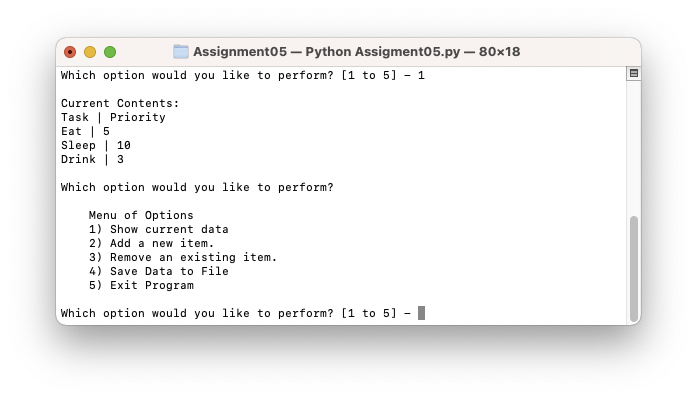
**Figure 3a.** A menu of options is presented to the user after each interaction.

**Figure 3b.** When the program starts, the user is presented with a menu of options.

If the user responds with a “1” when prompted by the menu, the program will display all items currently in lstTable. As seen in line 5 of Figure 4a, to make the output legible as a table, the column headers “Task” and “Priority” are separated by a “|” character. Each row of the lstTable is displayed as a new line by printing from within a for-loop. Here, because lstTable is composed of dicRow and the keys of dicRow are named, the print() command can rely on “Task” and “Priority” keys when printing from each row of lstTable within the for-loop. When the for-loop is finished, the user is once again prompted to select an option from the menu and the menu is displayed from within the while-loop.

# Step 3 - Show the current items in the table  
if (strChoice.strip() == '1'):  
 # Print headers  
 print("Current Contents:")  
 print("Task", "Priority", sep=" | ")  
 # Print data from table  
 for row in lstTable:  
 print(row["Task"],row["Priority"],sep=" | ")  
 # Repeat menu ask  
 print("\nWhich option would you like to perform?")  
 continue

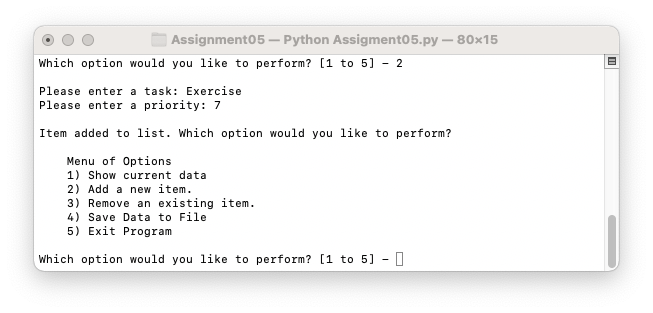
**Figure 4a.** Current items within lstTable are displayed to the user in a table format.

**Figure 4b.** The current contents of lstTable are displayed in a table format to the user.

If the user responds with a “2” when prompted by the menu, the program will add a new item to lstTable. As seen in Figure 5a, using the input() command, the program will ask the user to enter a task and a priority. The user input defines usertask and userpriority. These variables then define the values of the “Task” and “Priority” keys of dicRow, respectively. As seen in the Processing section of the script, dicRow is once again appended to lstTable. However, because dicRow has been changed with the user input, the dictionary is now a new line potentially containing new data unless otherwise entered by the user. Finally, the program prints a confirmation message to the user and asks the user to select an option before the while-loop restarts.

# Step 4 - Add a new item to the list/Table  
elif (strChoice.strip() == '2'):  
 # Query user  
 usertask = input("Please enter a task: ")  
 userpriority = input("Please enter a priority: ")  
 # Add user responses to dictionary  
 dicRow = {"Task" : usertask, "Priority" : userpriority}  
 # Append dictionary to table  
 lstTable.append(dicRow)  
 # Confirmation and menu ask  
 print("\nItem added to list. Which option would you like to perform?")  
 continue

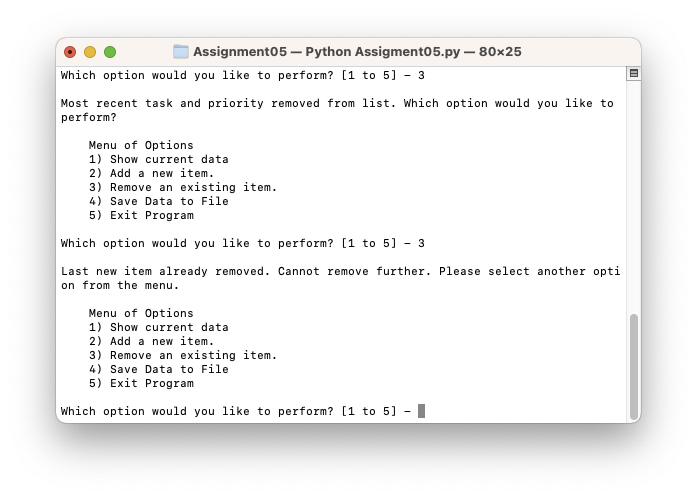
**Figure 5a.** The user is queried to enter a task and a priority. These two responses are then added to lstTable.

**Figure 5b.** The user is instructed to enter a task and a priority. These items are then added to lstTable.

If the user responds with a “3” when prompted by the menu, the most recently added item will be removed from lstTable. The remove function is employed in line 6 of Figure 6a to remove dicRow from lstTable. dicRow should be defined so long as the text file is not empty before starting the program and/or the user has entered a new item by selecting option 2 detailed above. If dicRow is defined, then the remove function is able to use dicRow as a string to find and remove an identical string from lstTable. However, should both of the aforementioned conditions fail and/or the user select “3” from the menu repeatedly without performing other steps in-between, lstTable.remove would result in an error. It is for this reason that the command is enclosed in a try-except construct. In the situations where the remove function succeeds, the user’s action is confirmed and the user is asked to select an option from the menu. When the remove function fails, the user is informed that the function cannot remove further and is instructed to select a different option from the menu.

# Step 5 - Remove a new item from the list/Table  
elif (strChoice.strip() == '3'):  
 # Enclose in try-except construct if cannot remove more items  
 try:  
 # Delete last row from table  
 lstTable.remove(dicRow)  
 # Confirmation and menu ask.  
 print("Most recent task and priority removed from list. Which option would you like to perform?")  
 except:  
 print("Last new item already removed. Cannot remove further. Please select another option from the menu.")  
 continue

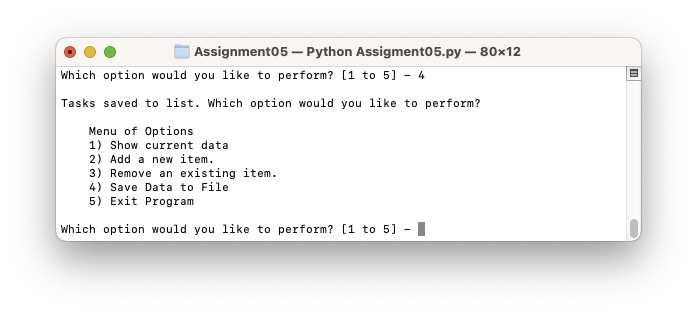
**Figure 6a.** The user may choose to remove the most recent item added to lstTable. This option cannot be repeated without adding items to the table in between removals.

**Figure 6b.** When the user selects “3”, the most recent task and priority are removed from the list. However, the user cannot repeatedly select “3” to continuously remove data from lstTable.

If the user responds with a “4” when prompted by the menu, the data stored in memory in lstTable is written to the ToDoList.txt text file. Within this elif statement, the open() command opens objFile (which is defined as ToDoList.txt in the Data section at the beginning of the script) and is set to write, indicated by the “w” in line 4 of Figure 7a. A for-loop is then employed to enter each row of lstTable into objFile as a separate line. lstTable is broken down into the “Task” and “Priority” keys, the values for which in each row are then written to objFile separated with a comma and followed by a new line. Following the end of the for-loop, the file is closed and the data has been written and is no longer solely stored in memory. The user is given a confirmation message and is once again asked to select an option from the menu of options presented by the while-loop.

# Step 6 - Save tasks to the ToDoList.txt file  
elif (strChoice.strip() == '4'):  
 # Open file  
 file = open(objFile, "w")  
 # Add rows of table to ToDoList.txt.  
 for row in lstTable:  
 file.write(row["Task"] + "," + row["Priority"] + "\n")  
 file.close()  
 # Confirmation and menu ask.  
 print("Tasks saved to list. Which option would you like to perform?")  
 continue

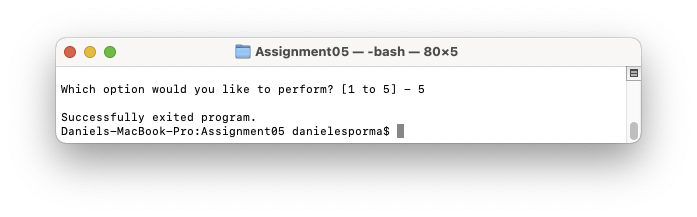
**Figure 7a.** Data stored in lstTable may be written to ToDoList.txt for longer-term storage. Without writing to the text file, the data will be lost when the program is closed.

**Figure 7b.** The user interacts with the program to write lstTable to ToDoList.txt.

If the user responds with a “5” when prompted by the menu, program displays that the user has successfully exited the program and the program closes. As seen in Figure 8, the program close is performed via a break, which closes the while-loop. Without the break, the menu of options would once again be displayed.

# Step 7 - Exit program  
elif (strChoice.strip() == '5'):  
 # Confirm successful exit  
 print("Successfully exited program.")  
 break # and Exit the program

**Figure 8a.** When the user enters “5” when prompted with the menu, the program will confirm successful exit and close.0

**Figure 8b.** The program confirms a successful exit before closing.

# Summary

Dictionaries can serve as powerful tools for collecting and organizing data. Though they can perform similarly to lists, dictionaries enable finer and more intuitive control of data stored within collections due to their ability to define keys. These keys can make it easier to communicate the flow of data to other coding collaborators and track which variables are directly related to which columns within a dataset. Reading and writing to a text file and moving data through the dictionary keys enable even greater flexibility when working with large datasets and making dynamic changes.