Natalie E. Tolliver

5/16/2020

ITNFDN 100A

Assignment 04

Script Using Add/Remove Items for a Task List and Printing to a File

# Introduction

The purpose of this assignment is to modify a script that manages a “ToDo list.” The “ToDo” file contains two columns of data, “Task” and “Priority.” The columns are loaded into a Dictionary object with each dictionary object representing one row of data. These rows are added to a Python List object to create a table of data. I used the PyCharm IDE to write the program and ran it in PyCharm as well as an OS command/shell window. I verified the functionality not only with the print statements but also by locating and opening the text file to verify that it contained the ToDo list in the desired format. Finally, I uploaded the code to GitHub.

# Establishing a GitHub Account

I already had a GitHub Account which had been used only for downloading files, not for uploading code. To learn how to use GitHub for uploading files, I clicked on “Read the Guide” and followed the steps to create a “Hello World” repository, start and manage a new branch, make changes to the file and commit, and then open and merge the pull request (https://guides.github.com/activities/hello-world/, 5/11/2020 (External)). I then created an “IntroToProg-Python” repository, uploaded the code for Assignment 5 along with the accompanying MS Word document, and committed the changes.

# Step 1 – Loading Data from Text File into Dictionary Rows

The first step was to copy the starter file provided with the assignment, modify the header, and enter the code under the “To Do” sections. I started by writing the code to prompt the user for the “Task name” and “Task Priority” and assign the data to a row in a dictionary and print it out using the code in Figure 1.

dicRow = {"Task": "strName", "Priority": "strPriority"}

print(dicRow["Task"], dicRow["Priority"])

Figure 1 - Assigning Task & Priority to a Dictionary Row

This assignment dealt with one table of data. The challenge was the many ways this data can be represented in the Python language. To me, a ToDo list simply looks like a matrix of data in an Excel spreadsheet with multiple rows and two columns, the first row consisting of header information.

In this code, I will use three different representations of the same data. The “list” type will be used in the code to store the data because with a list, it is easy to change, add, or remove data. The “list row” will, as the name implies, store a row of data from the list. It can be a bit confusing, but the “list table” will be used to store the rows. There is no type called a table. The table is simply used here to represent all the rows of data. The “type” for the rows is a list and the “type” for a table is a list. Both rows and tables are lists. The list.append() command is used to add a row and list.remove is used to delete a row.

A regular row in the list is just data that is accessed by using an index associated with its location in the table. So, as an example, if a comma delimited format were used, list row 1 would be “Send resume to Cindy,1.” Item 1 in the row is the task, “Send resume to Cindy” and item two is the priority, which is “1”. As the table gets larger, it becomes increasingly more cumbersome for the user to count the columns and access the data by its column number. To make it easier to refer to data, instead of using its column number, you can just use the column’s name. Note, the file does not actually contain the column’s name. You can read the file and assign “Send resume to Cindy” to the “Task” column of the row and assign the priority “1” to the priority column of the row. This type of assignment is called a “Dictionary row” because you look it up using words (called a “key”) like you would in a dictionary instead of looking it up by using column numbers (called an “index).

What makes this assignment tricky is the fact that although you can keep the dictionary formatting for appending, removing, changing, and printing the data, you cannot directly use the dictionary format for reading the data. The code in Figure 2 is used for reading the data from the text file and storing it in memory.

objFile = open("ToDoList.txt", "r")  
for row in objFile:  
 lstRow = row.split(",")  
 dicRow = {"Task": lstRow[0], "Priority": lstRow[1]}  
 lstTable.append(dicRow)  
 print(dicRow["Task"] + ',' + dicRow["Priority"].strip())  
objFile.close()

Figure 2 - 1st Iteration of Code to Read from File to Memory

Alternatively, each piece of data in the comma delimited text file, can be “unpacked” into the variables that will be used directly in the dictionary as opposed to being held in the list row to be put into the dictionary by using the commands in Figure 3.

objFile = open("ToDoList.txt", "r")  
for row in objFile:  
 strTask, strPriority = row.split(",")  
 dicRow = {"Task": strTask, "Priority": strPriority}  
 lstTable.append(dicRow)  
 print(dicRow["Task"] + ',' + dicRow["Priority"].strip())  
objFile.close()

Figure 3 - Another option for reading data from the file

I tried it both ways and they both gave me the same output as shown in Figure 4.

A screenshot of a cell phone

Description automatically generated

Figure 4 - Output of reading data from a file

I chose to use the method in Figure 3 because it eliminated the need to use the lstRow in the program.

# Step 2 – Display a Menu of Choices to the User.

The code for the menu was given to us in the starter file and used a simple print command with the triple quotes as the delimiter for the beginning and end of the print statement. I considered reading the menu into the variable strMenu because this variable was included in the starter code. However, the starter code also had the statement for printing the simple menu, so I left it unchanged since it was one statement instead of two (one to assign the menu to the variable and another to print the variable).

# Step 3 – Show the Current Items in the Table

I experimented with different methods of printing the data so that I could see the different formats. I used the “for row in lstTable” loop to go through all the rows in the table and print the data. In a first iteration, just to get the program working, I kept it simple using the code in Figure 5.

# Step 3 - Show the current items in the table  
if (strChoice.strip() == '1'):  
 # TODO: Add Code Here  
 for row in lstTable:  
 print(row)  
 continue

Figure 5 - 1st Iteration of printing the items in the table

The output was the unsightly and non-user-friendly data shown in Figure 6.

A screen shot of a computer

Description automatically generated

Figure 6 - Default format for printing the table

I changed to the syntax as shown in Figure 7so that it would print out in a more user friendly format.

print(row["strName"] + ',' + row["strPriority"] + '\n')

Figure 7 - Syntax to print out user friendly format

# Step 4 – Add a New Item to the List/Table

For this step, the user was prompted to enter a task which was assigned to variable strName and a priority which was assigned to variable strPriority. I then assigned these variables to dictionary rows and appended the dictionary row to the table using the statements in Figure 8. The code to accomplish this task is straight forward.

strName = input("Enter task name: ")  
strPriority = input("Enter task priority: ")  
dicRow = {"strName": strName, "strPriority": strPriority}  
lstTable.append(dicRow)

Figure 8 - Assign task & priority to a dictionary row

# Step 5 – Remove a New item from the List/Table

The remove item list option was the last segment of the program to run successfully. Originally, I had the if/then/else loop inside the option “3” code in a format like what was shown in class (<https://canvas.uw.edu/courses/1359964/discussion_topics/5484996>, 5/12/2020, (External)).

Machine generated alternative text:
input("ltem to Remove: 
strltem = 
for row in 1st Table: 
row["ltem"] . lower() 
strltem . lower() : 
1st Table . remove(row) 
print(" row removed") 
print(1stTab1e, 
'<< List with Dictionary objects') 
else: 
print ("row not found") 
print(1stTab1e, 
'<< List with Dictionary objects') 
— input ("Exit? ('y/ n'): 
strChoice - 
if strChoice. lower() ' 
break 
print(1stTab1e, 
'<< List with Dictionary objects') 

When I used that method to find the item to remove, for each row that it checked, if the object were not found, it would print out “row not found.” It would then go to the next row and print out “row not found.” If, for example, the list had 12 rows, and the item was in row 12, it would print out “row not found” 11 times until it found the item in row 12. In order to resolve this issue, I had to take the “row not found” statement outside of the “for loop.” However, that meant that every time the “for” loop was executed, at the end, the next statement printed would be “row not found.” This was not what was desired either. Finally, I created a flag called “strflag” and initialized it to “false” in the beginning of the program. I then set the flag to “true” when the row was located in the “for loop.” After the for loop was done, then I added a statement to test to see if the flag was true. If the flag was false, then the program would print out one time “row not found, please choose option 3 and try re-entering the task to be removed.” The code is shown in Figure 9.

elif (strChoice.strip() == '3'):  
 # TODO: Add Code Here  
 strItem = (input("Which item would you like to remove? "))  
 for row in lstTable:  
 if row["strName"].lower() == strItem.lower():  
 lstTable.remove(row)  
 print(strItem, "was removed.")  
 strFlag = "true"  
 if (strFlag != "true"):  
 print("Row not found, please choose option 3 and try re-entering the task to be removed")  
 continue

Figure 9 - Added flag to track whether row was found

A screen shot of the original text file is shown in Figure 10.

A screenshot of a cell phone

Description automatically generated

Figure 10 - Original text file

Figure 11 shows the program as run in PyCharm.

A screenshot of a cell phone

Description automatically generated

A screenshot of a cell phone

Description automatically generatedA screenshot of a cell phone

Description automatically generated

Figure 11 - Program run results in PyCharm

When the program was run to try and capture the screen shots, one last change had to be made. The flag to track whether the item to be removed was found was set to “true” once the item was found. On the second iteration, it would still be true, so even if the item were not found again, the program would not print “item not found.” I had to go back and make sure I set the flag to “false” before the program started looping through the rows. Notice that when “Call Mom” was entered, it correctly gave the row not found message.

The new contents of the ToDo List is shown in Figure 12.

A close up of a keyboard

Description automatically generated

Figure 12 - ToDo List

Notice that “Call Katie” has been added and “Buy dog food” has been removed.

Figure 13 shows the results from running the code in the command menu.

A screenshot of a cell phone

Description automatically generated

Figure 13 - Results from running code in command window

In order for the code to execute properly in the command menu, the input() needed to be used at the end of the program, otherwise, it would close before the user had time to read the results or take screen shots. Instead of just input(), I used input(“Press enter to exit the program”).

# Summary

This program introduced me to the concepts of using list data, reading data from a file into memory, writing from memory to a file, adding data to the file in memory, and deleting data from a file in memory. It introduced the complexity of nested loops and dictionary rows and continued the usage of structured code to read in data, process data, and write data to file. The concepts were combined into a program that performs the useful function of managing a todo list. I look forward to learning new concepts and expanding upon old concepts to perhaps build a better list. Note, I have actually been using the program and the text file to manage my real todo list. So, the text file no longer contains the same data shown in Figure 10. Great app!