Kristen Doehlert

August 8, 2022

ITFDN 110 B

Assignment 05

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

Lists and Dictionaries

# Introduction

In module 5 we learned how to use lists and dictionaries. We also learned how to read data from a file. In this paper I am going to discuss how I created a script that reads data from a ToDoList text file into a list of dictionaries, adds items to and removes items from the list, and saves those changes to the file.

# Loading the data

Because the Data section declaring the variables was already complete in the starter file, the first step I completed was processing data from a text file into memory. I added code to open a file named ToDoList.txt in read mode. Then, I created a for loop to split the items on one line by a comma and assign these elements to *lstRow.* The item in position 0 is then assigned as the value to the key “Task” and the item in position 1 is assigned to the value of the key “Priority” in the dictionary *dicRow*. This dictionary is appended to the list *lstTable*, and then the loop continues to the next line in the text file. Once this is successful, the file is closed and statement confirming that the data has been retrieved is printed.

This code is wrapped in a try-except block so that there will not be an error if the text file ToDoList.txt does not already exist. If the file does not exist, “File not found, will make new file when you save” is printed and the program moves on to the menu. Figure 1 shows this section of code.

Text

Description automatically generated

***Figure 1. Code to load data from the file into a list of dictionaries***

# Displaying the list

Step 2, displaying the menu and getting user input, was also already completed in the starter file. Step 3 includes code to display the current items in the table if the user inputted ‘1’ in the menu selection. This code also uses a for loop to loop through the dictionaries in *lstTable*. The string *strData* is created using the keys to get the values of Task and Priority for each row and concatenating these elements with text and a new line for readability. Then *strData* is printed and the loop continues to the next dictionary row in the list.

Graphical user interface, text, application

Description automatically generated

***Figure 2. Code to show current items in the list***

# Adding items to the list

Step 4 adds new items to the list if the user inputs ‘2’ in the menu. This code gets the user input for *strTask* and *strPriority*, then adds these values to the dictionary *dicRow* with the keys “Task” and “Priority”. After the dictionary is created, it is appended to the list *lstTable,* which contains the other dictionaries for any other items on the to-do list. Finally, a print statement is included to confirm that the item was added.

Text, letter

Description automatically generated

***Figure 3. Code to add new items to the list***

Figure 4 below shows what this code looks like when run in the Command Prompt. In this example, the Task “Do dishes” and the Priority “1” were added.

Text

Description automatically generated

***Figure 4. Result of the code from Step 4 run in the Command Prompt***

# Removing items from the list

Step 5 removes an existing item from the list if the user inputs ‘3’ in the menu. This code gets the user input and assigns it to the variable *strTask*. Then it loops through the dictionary rows in *lstTable* and checks to see if *strTask* matches the value paired with the “Task” key in that row. The .lower() method is used so that the user input is not case sensitive. If there is a match, the del statement is used to delete the dictionary element from *lstTable*. The index() method is needed to return the index of the row in the list, since the del method cannot use a value and must use an index to specify which element should be deleted. After the item is deleted, “Item removed” is printed. If no match is found, no action is taken and the program loops back to the menu.

Graphical user interface, text

Description automatically generated

***Figure 5. Code to remove an item from the list***

When testing my code, I displayed the items in the list, removed the task “Do dishes”, then displayed the items again. The result of this testing is shown in Figure 6.

Text

Description automatically generated

***Figure 6. Result of code to display the list, remove an item, then display the list again***

# Saving data to the file

If the user enters ‘4’ in the menu, Step 6 of the code is run to save the data to a file. The file ToDoList.txt is opened in write mode, and a for loop is used to loop through each dictionary row in *lstTable*. The value assigned to the key “Task” and key “Priority” for that row are written to the file, separated by a comma and with a new line at the end. Then a print statement is used to print that the data has been saved.

Graphical user interface, text, chat or text message

Description automatically generated

***Figure 7. Code to save data to a file***

When testing my code in the Command Prompt, I selected choice ‘4’ and the program indicated that the file was saved. This is demonstrated in Figure 8.

Text

Description automatically generated

***Figure 8. Result of running code from Step 6***

After saving the data, I opened the ToDoList.txt. file to confirm the data was saved as expected. The task and the priority are listed on one line separated by a comma. Figure 9 shows the content of the text file.

Graphical user interface, text, application, Word

Description automatically generated

***Figure 9. Text file with data in comma separated list***

# Exiting the program

The final step in the code exits the program if ‘5’ is chosen. Before it exits, user input is assigned to the variable *strSave* to check if they want to save their data before they exit. This is so they do not accidentally lose their data. If ‘y’ is selected, the same code used in Step 6 is run, “File Saved. Goodbye” is printed, and the program is closed. If anything else is entered, “Goodbye” is printed, and the program is closed. Figure 10 shows this code.

Graphical user interface, text, application, chat or text message

Description automatically generated

***Figure 10. Code to exit the program***

Figure 11 below shows the result of running the code to exit the program.

Text

Description automatically generated

***Figure 11. Result of running code to exit the program***

# Summary

In this paper I discussed how I created a script that creates a to-do list. The script loads data from an existing text file, adds new items, removes existing items, displays the items, and saves the data to the text file.