Vanessa Rubin

16 November 2022

Foundations of Programming: Python

Assignment 05

https://github.com/VRubin123/IntroToProg-Python

Using Dictionaries

Introduction

Dictionaries are another method of storing and calling data in Python. They are similar to lists, however, instead of using an index to recall objects stored in memory, they use keys. Keys replace index position, by assigning a string to a piece of data within a database. Since you no longer need to remember the index position of a desired piece of data, dictionaries may be used within a list. Allowing data to be easily retrieved.

To Do list using Dictionaries

Using lists in loops is essential to writing databases and manipulating data. This week, we would like to store data supplied by a user as a “To do list” list table, but as a dictionary within a list. Using the dictionary, categorizes the data, and allows for easier data retrieval and manipulation.

We are supplied with a starter code for this assignment. Constants and variables have already been added to the program. I changed objFile constant to equal “None”, since we will not be using the local path to run the program, Figure 1. Additionally, I then assigned the constant “strFile” to equal the text file “ToDolist.txt”, Figure 1.

```
objFile = None  # Changed objFile to None-VLR
strFile = "ToDoList.txt"   # An object that represents a file # changed objFile to strFile-VLR
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, modifying given constants

The first task in Assignment05 is to open the “ToDolist” file and load the data into a python list of dictionary rows. This was done by assigning the constant objFile to open the text file as a readable document. Using a for loop for each row in the to do list, the loop will parse out data based on a coma using the split function. Further, a variable to represent a dictionary row is set to assign a key to the data and strip the return space in the list, Figure 2.

```
# -- 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)
objFile = open(strFile, "r")

for row in objFile:
    strData = row.split(",")
    dicRow = {"Task":strData[0].strip("\n")}
    lstTable.append(dicRow)
objFile.close()
```

Figure 2, processing “ToDolist.txt” file

Next, we are tasked to write code to perform tasks set forth by the “Menu of Options”, Figure 3.

```
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
""")
```

Figure 3, To Do list program menu

This menu is written within a while(True) loop in order to prompt the user to other menu tasks until the user chooses to exit, option 5.

The first task asks the user if they would like to display the current data. Since the file has been loaded as a list of dictionary rows, I was able to use the key assigned earlier to call on the data of the to do list, Figure 4.

```
if (strChoice.strip() == '1'):
    for row in lstTable:
        print(row["Task"])

    continue
```

Figure 4, display current data saved to list

This was written a for loop to query each row of the list that is assigned to the key “Task”.

Choice 2 allows the user to add new tasks to the to do list. A while (True) loop was used to query the user to input the data. The input is stored into a dictionary with the same key “Task” used to load the data. Next, the dictionary row is appended to the list. Finally, the user is asked to return to the menu or continue adding to the list. If “y” is selected, the while loop is broken, and the user returns to the menu, Figure 5.

```
elif (strChoice.strip() == '2'):
    while(True):
        strData = input("To do item:")
        dicRow = {"Task":strData}
        lstTable.append(dicRow)
        strChoice = input("Return to menu? ('y/n'):")
        if strChoice.lower() == 'y':
            break
    continue
```

Figure 5, add data to list

Choice 3 allows the user to remove items from the list. I used a similar piece of code to perform this action. However, instead of using the append prompt, I used remove, Figure 6.

```
elif (strChoice.strip() == '3'):
    while(True):
        strItem = input("Item to Remove:")
        for row in lstTable:
            if row["Task"] == strItem:
                lstTable.remove(row)
                print ("row removed")
            else:
                print ("row not found")
        strChoice = input("Return to menu? ('y/n'):")
        if strChoice.lower() == 'y':
            break
    continue
```

Figure 5, remove item from list

Choice 4 allows the user to save the data to “ToDoList.txt” file. Using similar code to process the data at the beginning of the program, I wrote the list table to the file, had it saved as a string without the format of a dictionary, Figure 6.

```
# Step 6 - Save tasks to the ToDoToDoList.txt file
elif (strChoice.strip() == '4'):
    objFile = open(strFile, "w")
    for row in lstTable:
        objFile.write(str(row["Task"]) + "\n")
    objFile.close()
    continue
```

Figure 6, saving to text file

Finally, choice 5 prompts the user to exit the program by breaking the initial while loop, Figure 7.

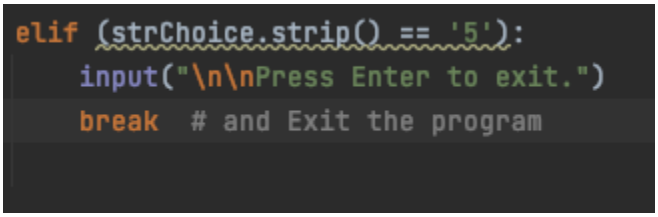

Figure 7, ending the program

Conclusion

Like earlier assignments, using a while loop allowed the user to interact with the program at several levels. However, unlike before, the use of dictionary rows within a list made it easier to call specific information without having to remember the index. This attribute allows a programmer to design high dimensional databases with callable categories related to the information within.