Arielle Waller

August 12, 2020

Foundations of Programming (Python)

Assignment05

Assignment05

Introduction

The task for this assignment was to start with last week's program and change the inner data type of the two-dimensional table from a list to a dictionary. In other words, we wanted to change a list of lists to a list of dictionaries. In addition, we were instructed to modify the script so that it could load existing data and delete an entry.

Declare Variables

To complete this assignment, my first approach was to work in chunks. I began with the 'Declare variables' section. Since we were instructed to replace the list of lists with a list of dictionaries, I started by changing the variable *lstRow* to *dicRow*. I made *dicRow* an empty dictionary instead of an empty list.

```
dicRow = {}  # dictionary of data row
```

Figure 1 - Changing an empty list to an empty dictionary.

Loading Existing Data

In the section of code for user input, the next "to-do" was to add code allowing the user to load existing data. I used the *open()* function with *strFileName* and 'r' as arguments. This would create/open a file object called *CDInventory.txt* and open it for reading as discussed here (external site)[1].

In this file, each CD's entry would be a string. First, I needed to convert the string to a list using the *split()* method. Since the ID, CD title, and artist would be separated by a comma, I could use a comma as the argument for the *split()* method. This would split the string on every comma and make a list of elements. Furthermore, in order to remove the newline character, I could use the *strip()* method. Thus, I would be using the *strip()* and *split()* methods on each row in the file.

Since I would be iterating through each row in the file, I created a *for* loop.

[1] Retrieved 2020-Aug-12

```
for row in objFile:
```
Figure 2 - Creating a for loop to iterate through each string row in the file.

I created a new variable called *lstRow* and assigned it to the value returned from calling the *strip()* and *split()* methods on each row. In doing so, I also added *lstRow* back to the 'Declare Variables' section of code and assigned it to an empty list (I had previously replaced it with *dicRow*).

```
lstRow = row.strip().split(',')
```
Figure 3 - Creating a list from the string loaded in the file, with elements being separated by a comma.

To create the dictionary, I assigned the keys *ID, Title*, and *Artist* to the elements at indices 0, 1, and 2 of *lstRow*. For the value for the key *ID*, I cast the string represented by *lstRow[0*] to an integer.

```
dicRow = {'ID':int(lstRow[0]), 'Title':lstRow[1],'Artist':lstRow[2]}
```
Figure 4 - Creating a dictionary using the elements of the list as values.

Then, I used the *append()* method to add this dictionary to the table *lstTbl*.

```
lstTbl.append(dicRow)
```
Figure 5 - Adding the dictionary created from the string in the file to the table.

I used a *print()* statement with *lstTbl* as the argument to display the table back to the user.

```
print(lstTbl)
```
Figure 6 - Displaying the table back to the user.

Finally, I called the *close()* method on *ObjFile* to close the file and minimize the time necessary for file access.

```
objFile.close()
```
Figure 7 - Closing the file.

Adding Data to the Table

For the user to add data to the table as a dictionary rather than a list, only a few modifications had to be made. The user's input for the CD's ID, title, and artist would be the values for the appropriately named keys. I used brackets to create a dictionary and assigned the keys *ID*, *Title*, and *Artist* to *strID*, *strTitle*, and *strArtist* respectively. I then went back and changed the value for *ID* to *intID* as I saw that I was assigning a string to a key that should hold an integer. I assigned this dictionary to *dicRow* and deleted *lstRow.*

```
dicRow = {'ID':intID, 'Title':strTitle, 'Artist':strArtist}
```
Figure 8 - Creating a dictionary with user input as values.

Next, I changed the *append()* method's argument on *lstTbl* from *lstRow* to *dicRow.* Now the new dictionary would be appended to the table.

```
lstTbl.append(dicRow)
```
Figure 9 - Adding the new dictionary to the table.

Displaying the Current Data to the User

Since this section did not have any to-do's as comments or anything that stood out to me blatantly, I almost skipped modifying this section of code. As I ran the code and tested out this section, however, I saw that only 'ID, Title, Artist' was being displayed. These were the dictionary's keys rather than the dictionary's values. I looked at the *for* loop and saw that it was looping through each row in the table. Since each row in the table was a dictionary, I needed to use the *values()* method so that the values would be printed rather than the keys.

```
for row in lstTbl:
    print(*row.values(), sep = ', ')
```
Figure 10 - Iterating through each dictionary in the table and printing its values.

Deleting an Entry from the Table

For a user to delete an entry, I figured they would need to specify the value for the CD's ID, title, or artist. I would not be able to delete a dictionary in the table based on a key since all of the dictionaries had identical keys.

To start with, I decided I would allow the user to enter an ID only. Referencing this page (external site)[2], I found that I could iterate through each element of my list (each dictionary). If the key in the dictionary at the specified index in my list had a value equal to the user's input, I could delete the dictionary at that specified index.

As a result, I called the *input()* function with a string argument prompting the user for the ID of the entry they wanted to delete. I cast the user's input to an integer and assigned it to the variable *delValue.*

```
delValue = int(input('Enter the ID of the entry you wish to delete: '))
```
Figure 11 - Capturing the ID of the entry the user wants to delete as an integer.

Next, I used *len()* with *lstTbl* as an argument to calculate the number of elements (the number of dictionaries) in the table. I nested this inside the *range()* function to create a list of numbers

[2] Retrieved 2020-Aug-12

based on the number of elements in the table. Finally, I used a *for* loop to iterate through this list.

```
for i in range(len(lstTbl)):
```
Figure 12 - Calculating the number of elements in the table, using the range function to generate a list of numbers up to that length, and iterating through that list.

The element at index *i* of *lstTbl* would be the dictionary at that index. If the dictionary at that index had a value equal to *delValue* for the key *ID*, that dictionary needed to be deleted. I did note that the program would rely on the user to create sequential indices beginning at 0.

Thus, I created an *if* statement with that as my condition.

```
if lstTbl[i]['ID'] == delValue:
```
Figure 13 - If statement to see if the key 'ID' in the dictionary at index i has a value equal to delValue.

I used the *del()* functon to delete the dictionary at the table index where the condition was true.

```
del(lstTbl[i])
```
Figure 14 - Deleting the dictionary at the table index where the if statement is true.

I added a *print* statement with an *f-string* to display the new table post-entry-deletion to ensure that the result was as expected.

```
print(f'Your new table consists of: {lstTbl}')
```
Figure 15 – Print statement to check that row was deleted from table.

I also added a *for* loop, iterating through the table and printing the values of each dictionary.

Saving the Data to a Text File

To save the data to a text file, I added the *values()* method on my *row* object so that the *for* loop would iterate over the dictionary values.

```
for item in row.values():
```
Figure 16 - Modifying the for loop to iterate over the values in the table's dictionary.

Since every section now had statements, I removed the *pass* statements so that I would be able to ensure each section was working as expected.

Testing the Code

In the section where the user has the option to load existing data, I noticed that the data loaded from the file was being put into the table in memory with the new data entered by the

user. If the user chose to save the CD inventory data from that session, there would be a lot of duplicates. From the lab, I knew that I could use the *clear()* method on the table to clear the current table in memory for the loaded data.

```
lstTbl.clear() # clear the current table in memory for loaded data
```

Figure 17 - Clearing the current table in memory to keep the loaded data separate.

There could still be the potential for the loaded data to be saved back into the file, creating duplicates in the file. I considered loading data from the file into memory before the user is able to add data and restricting the user from adding IDs that were already in the dictionary.

Since I did not include any rows as test data in my table, the table in memory was empty when I ran the script. I thought there could be some confusion if the user chose 'i' to display the current data and only 'ID, CD Title, Artist' appeared. As a result, I decided to add an *if* statement with *lstTbl* as the condition. If there was data in *lstTbl*, the data would display.

```
if lstTbl:
    print('ID, CD Title, Artist')
    for row in lstTbl:
        print(*row.values(), sep = ', ')
```

Figure 18 - Adding an if statement for the dictionary to display only if it contains data.

I added an *else* statement to account for an empty table with a *print* statement to inform the user that there were no entries to display.

```
else:
    print('No entries exist to display. Choose \'a\' from menu.')
```

Figure 19 - Inform the user that they must first add entries if the table is empty.

I followed the same process to account for the user wanting to delete entries from an empty table.

```
elif strChoice == 'd':
    if lstTbl:
        delValue = int(input('Enter the ID of the entry you wish to delete: '))
        for i in range(len(lstTbl)):
            if lstTbl[i]['ID'] == delValue:
                del(lstTbl[i])
                print(f'Your new table consists of: {lstTbl}')
        for row in lstTbl:
            print(*row.values(), sep = ', ')
    else:
        print('No entries exist to delete.')
```

I considered adding another conditional statement to account for the possibility that all of the entries in the table were deleted. A message could be displayed back to the user that the new table was now empty if there were no entries left in *lstTbl* after deletion. However, I thought

that the code block was already starting to look crowded, and we were asked not to create our own functions for this assignment.

Continuing to test different conditions, I noticed I got an error if I tried to load existing data before the file was created via saving the inventory to the file. I wanted to create another *if* statement to check whether or not my file existed before attempting to load data.

First, I tried making an *if* statement with the condition that the file existed. If the file did not exist, a message would be displayed telling the user they needed to create a file first.

```
    if objFile:
        lstTbl.clear() # clear the current table in memory for loaded data
        objFile = open(strFileName, 'r')
        for row in objFile:
            lstRow = row.strip().split(',')
            dicRow = {'ID':int(lstRow[0]), 'Title':lstRow[1],'Artist':lstRow[2]}
            lstTbl.append(dicRow)
        print(lstTbl)
        objFile.close()
    else:
        print('You need to create a file first. Enter \'s\'\n')
```

Figure 20 - Creating an if statement to check if a file exists before attempting to load data from it.

Through [this website] (external site)[3], I found that this was not recommended because of race conditions. I amended the code to use *try* and *except.*

```
        try:
            objFile = open(strFileName, 'r')
            lstTbl.clear() # clear the current table in memory for loaded data
            for row in objFile:
                lstRow = row.strip().split(',')
                dicRow = {'ID':int(lstRow[0]), 'Title':lstRow[1],'Artist':lstRow[2]}
                lstTbl.append(dicRow)
            objFile.close()
            print(lstTbl)
        except IOError:
            print('No file exists. Create a file first. \n')
```

Figure 21 - Attempting to open the file. Loading the data if the file exists. Displaying an error if the file does not exist.

Running the Script

With all of the modifications, I found the script ran as expected in both Spyder and the terminal.

[3] Retrieved 2020-Aug-12

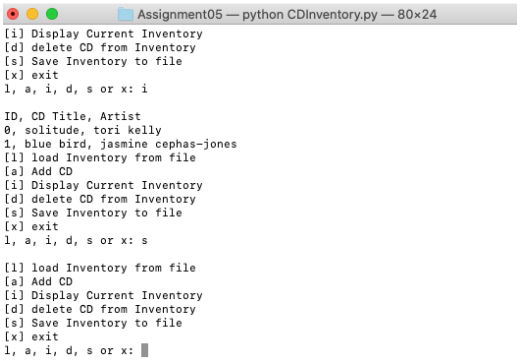

Figure 22 - CDInventory.py running in the terminal.

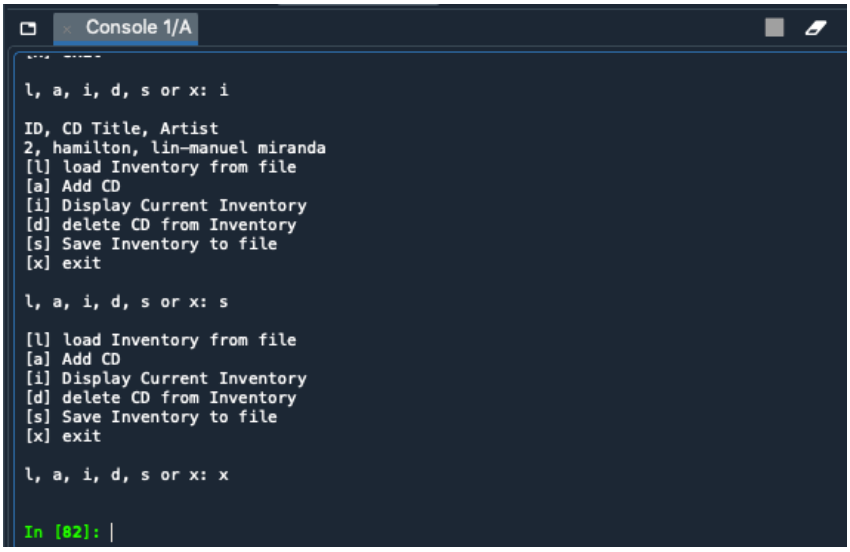

Figure 23 - CDInventory.py running in Spyder.

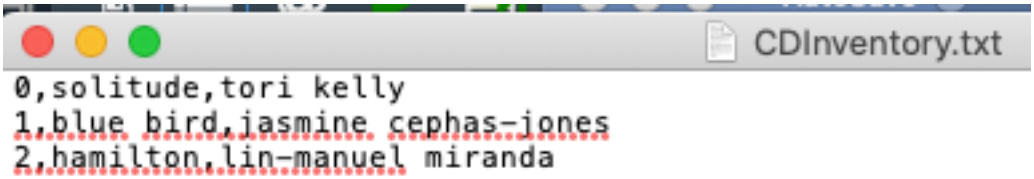

Figure 24 - CDInventory.txt in TextEdit

Summary

I found this week's assignment to be more challenging than those of previous weeks. I kept reading *lst* in *lstTbl* as *1st* and making other spelling errors when I called on variables that I had not named myself.

I create my knowledge document after writing my code, so this week also required more effort in going back and seeing what specifically I had modified in the starter code.

In converting the code to work with a list of dictionaries instead of a list of lists, I found, too, that a lot of my errors were semantic rather than syntactic (ex. Not using the *values()* method on my dictionary objects. This was difficult for me since I could not just rely on Spyder to point out what I needed to fix.

Accounting for the different possible use cases was also more time-consuming than usual as there seemed to be a lot of potential errors and confusion based on the user input.

Appendix

Using planetb's webpage[4] (external site):

```
#------------------------------------------#
# Title: CDInventory.py
# Desc: Create a CD inventory. Allow the user to exit the program, load existing
#       data, add data to the table, display the data in memory, delete entries,
#       and save the data in memory to a file.
# Change Log: (Who, When, What)
# AWaller, 2020-Aug-12, Created file
#------------------------------------------#

# Declare variables

strChoice = '' # User input
lstTbl = []  # list of dictionaries to hold data
dicRow = {}  # dictionary of data row
lstRow = []  # list of data row
strFileName = 'CDInventory.txt'  # data storage file
objFile = None  # file object

# Get user Input

print('The Magic CD Inventory\n')
while True:
    # 1. Display menu allowing the user to choose:
    print('[l] load Inventory from file\n[a] Add CD\n[i] Display Current Inventory')
    print('[d] delete CD from Inventory\n[s] Save Inventory to file\n[x] exit')
    strChoice = input('l, a, i, d, s or x: ').lower()  # convert choice to lower case a
t time of input
    print()

    if strChoice == 'x':
        # 5. Exit the program if the user chooses so
        break
    if strChoice == 'l':
        # Loading existing data
        try:
            # Checking if file object exists and executing code block if so.
            objFile = open(strFileName, 'r')
            lstTbl.clear() # clear the current table in memory for loaded data
            for row in objFile:
                lstRow = row.strip().split(',')
                dicRow = {'ID':int(lstRow[0]), 'Title':lstRow[1],'Artist':lstRow[2]}
                lstTbl.append(dicRow)
```

[4] Retrieved 2020-Aug-12

```
            objFile.close()
            print(lstTbl)
        except IOError:
            # Raising an error if the file does not exist.
            print('No file exists. Create a file first. \n')
    elif strChoice == 'a':  # no elif necessary, as this code is only reached if strChoice is not 'exit'
        # 2. Add data to the table (2d-list) each time the user wants to add data
        strID = input('Enter an ID: ')
        strTitle = input('Enter the CD\'s Title: ')
        strArtist = input('Enter the Artist\'s Name: ')
        intID = int(strID)
        dicRow = {'ID':intID, 'Title':strTitle, 'Artist':strArtist}
        lstTbl.append(dicRow)
    elif strChoice == 'i':
        # 3. Display the current data to the user each time the user wants to display the data
        if lstTbl:
            # Checking to make sure there is data to display in the table.
            print('ID, CD Title, Artist')
            for row in lstTbl:
                print(*row.values(), sep = ', ')
        else:
            print('No entries exist to display. Choose \'a\' from menu.\n')
    elif strChoice == 'd':
        # Delete an entry
        if lstTbl:
            # Check that there are entries in the table to delete.
            delValue = int(input('Enter the ID of the entry you wish to delete: '))
            for i in range(len(lstTbl)):
                if lstTbl[i]['ID'] == delValue:
                    del(lstTbl[i])
                    print(f'Your new table consists of: {lstTbl}')
            for row in lstTbl:
                print(*row.values(), sep = ', ')
        else:
            print('No entries exist to delete. Select a different option. \n')
    elif strChoice == 's':
        # 4. Save the data to a text file CDInventory.txt if the user chooses so
        objFile = open(strFileName, 'a')
        for row in lstTbl:
            strRow= ''
            for item in row.values():
                strRow += str(item) + ','
            strRow = strRow[:-1] + '\n'
            objFile.write(strRow)
        objFile.close()
    else:
        print('Please choose either l, a, i, d, s or x!')
```