Rei Bengu
August 8, 2021
IT FDN 110 B
Assignment 05

Dictionaries

Introduction

This module expands on the previous module which focused on the concept of sequences, by introducing Dictionaries. Dictionaries are similar to lists however each element has 2 parts, a key and a corresponding variable. By combining the concept of lists with dictionaries, a list of dictionaries can be created which is like a table, but each cell references its header. The beneficial part of dictionaries compared to a standard list is the way each element can be referenced. In a traditional sequence type you would have to index and reference the component of the sequence by finding its index. Dictionaries are mapping types, so each value is stored under a key, making it easier to reference and access values. The concept of version control and how groups can work on the same project was explored via github. The script and this knowledge document can be found at https://github.com/RBengu/Assignment_05 to see the progression of this assignment.

CD Inventory Script

To put to practice the skills learned in this module, the basic script created in last module which asked the user to enter basic CD data to create an expanding table of information was revisited and updated with the concepts learned in this module on Dictionaries, with more features as well.

After the script declares some initial variables used throughout the code it launches into the main input section of the program, seen below. The user is asked what they would like to do by entering a corresponding value from the menu option. The entire program is contained within a while loop that can only be broken if the user selects to exit when prompted at the menu, because of this the exit option is programmed first so that the program does not needlessly run through the remaining conditional statements should the user decide to exit.

```
# -- PRESENTATION (INPUT) -- #
# Get user Input
print('The Magic CD Inventory\n')
while True:
    # 1. Display menu allowing the user to choose:
    print('\n[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 at time of input
    print()

        # -- PROCESSING -- #
        if strChoice == 'x':
            # 5. Exit the program if the user chooses so
            break
```

The first choice the user has ‘load inventory from file’ is shown below. Once the file is open and read, each line is split by using the comma as a delimiter and saved as elements in a list. The list is then converted into a dictionary, assuming the format of the file is consistent with the standard format of the script “ID, Title, Artist”. Since the file data is read as a string, the ID is converted into an integer as it is assigned to a dictionary in order to keep it consistent with the rest of the script. The entries are then appended to the table in the event that the user has already manually added CD’s so as to not overwrite those entries. Once the entries are added to the dictionary, the file is closed to limit its unnecessary exposure to corruption.

```python
    if strChoice == 'l':
        objFile = open(strFileName,'r')
        for row in objFile:
            lstRow = row.strip().split(',')
            dicRow ={'id':int(lstRow[0]), 'artist':lstRow[1], 'title':lstRow[2]}
            dicTbl.append(dicRow)
        objFile.close()
```

The next choice the user has ‘add CD’ is to manually add an entry to the table. This is achieved by 3 simple input functions which ask the user for the ID, Title of the CD, and Artist of the CD. These 3 values are first stored as individual variables, and then assigned to corresponding keys in a dictionary which is then appended to the list of dictionaries to create a 2-D table of dictionaries.

```python
    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}
        dicTbl.append(dicRow)
```

The next choice the user has ‘display current inventory’ prints out the entries currently saved to the memory. The information is printed sequentially using 2 nested for loops which first break down the list of dictionaries into lists, and then each list goes through each key:value pair and saves the value to a string along with a delimiter. Once every key:value pair in the row is saved as a string, the last 2 characters are trimmed off as they are just a ‘, ‘ due to the iterative nature used to generate the string. Then the row is printed to the user and the script moves on to the next row.

```python
    elif strChoice == 'i':
        # 3. Display the current data to the user each time the user wants to display the data
        print('ID, CD Title, Artist')
        for row in dicTbl:
            strRow = ''
            for key, val in row.items():
                strRow += str(val) + ', '
            strRow = strRow[:-2]
            print(strRow)
```

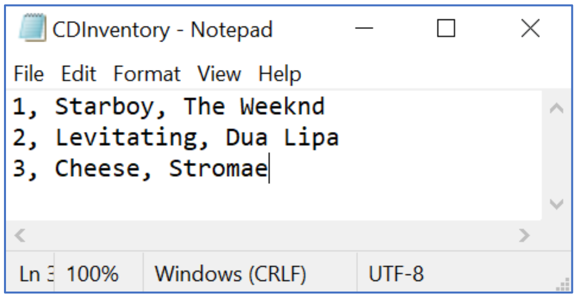
The next choice the user has 'delete CD from inventory' is a new addition to the program compared to the previous module and requires finding the row the user wants to be deleted, and then removing that row from the list of dictionaries. This function starts exactly the same as the display to show the user the current contents of the CD Inventory, and then asks them to pick the ID that corresponds to the entry they want removed. This is a perfect example of how functions can be utilized to make a code cleaner and easier to use as a programmer and limit the amount of duplication in a script. Once the user is presented with the current inventory, they select the ID they want to remove which is saved as an integer. The script then uses a sequence of for loops to go row-by-row, entry-by-entry within the list of dictionaries, and run a conditional statement that checks if the value of the dictionary entry matches the users entry, if so then the row is removed using the .remove() function. Since the script goes row-by-row and checks every key:value within the dictionary of each row, the code can be further improved by removing the conversion of the users input into a integer, so the user can have more flexibility with deleting CD's, such as entering an artists name to remove all CDs by that artist, etc.

```
    elif strChoice == 'd':
        print('\nID, CD Title, Artist')
        for row in dicTbl:
            strRow = ''
            for key, val in row.items():
                strRow += str(val) + ', '
            strRow = strRow[:-2]
            print(strRow)

        delID = int(input('\nPlease select the ID of the CD to be deleted: \n'))

        for row in dicTbl:
            for key, val in row.items():
                if val == delID:
                    dicTbl.remove(row)

```

The last choice the user has 'Save inventory to file' does exactly that, and saves the current memory into a text file 'CDInventory.txt'. The process is similar to that used in the previous module for lists, and almost identical to the method used to display the memory to the terminal that is already used twice within this program. Since the user has the choice that they want to go through the menu the code needs to be repeated in order to obtain the strings, but this time instead of printing to the terminal they are printed to a file. This is another example of how a function that returns the 'strRow' variable in the code below can be utilized throughout the script to either print to the terminal or a file and streamline the script as a whole.

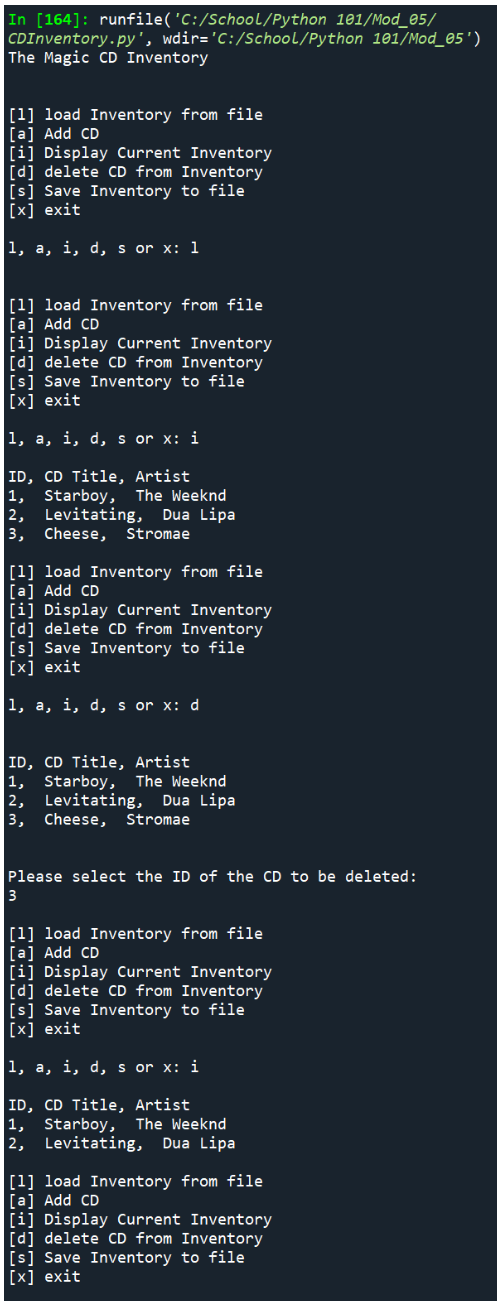
```
    elif strChoice == 's':
        # 4. Save the data to a text file CDInventory.txt if the user chooses so
        objFile = open(strFileName,'w')

        for row in dicTbl:
            strRow = ''
            for key, val in row.items():
                strRow += str(val) + ', '
                strRow = strRow[:-2] + '\n'
                    objFile.write(strRow)
                 objFile.close()
        else:
            print('Please choose either l, a, i, d, s or x!')
```

Figure 2 below shows the CDInventory.txt file with some pre-filled entries before the code is ran. An example of the output of this script can be seen below in in **Error! Reference source not found.** which is directly from the console of the Spyder IDE, going through every menu choice available as well as in which is directly from a terminal. **Error! Reference source not found.** shows the CDInventory.txt file after the program is finished running. Figure 4 shows an example of the script running in a terminal.

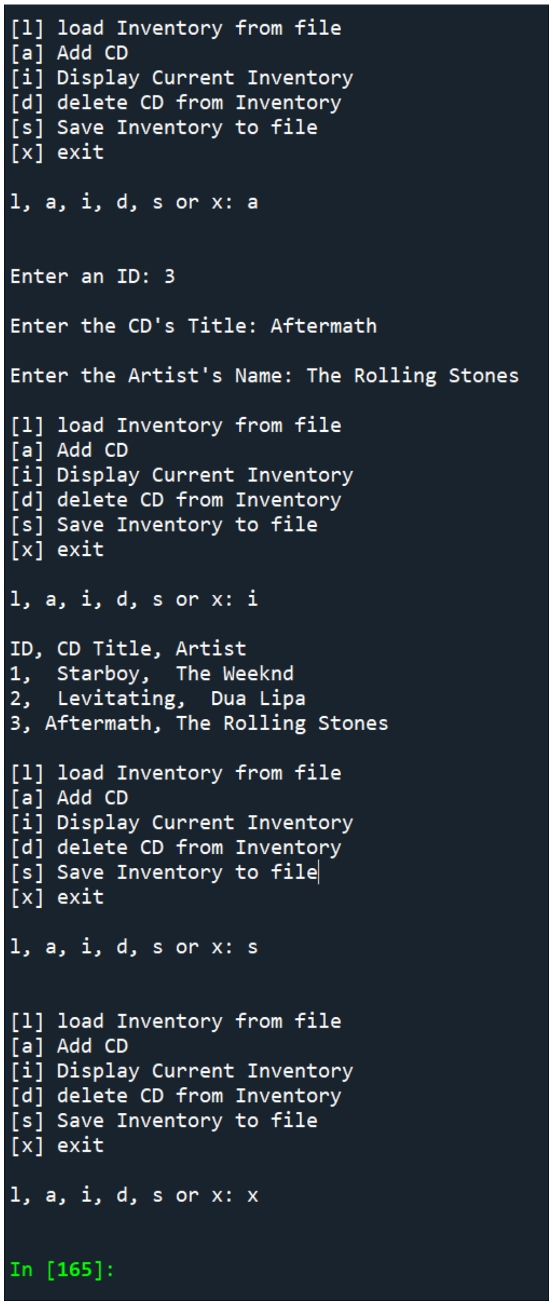


```
1, Starboy, The Weeknd
2, Levitating, Dua Lipa
3, Cheese, Stromae
```

Figure 1 - Initial 'CDInventory.txt' with pre-filled values.

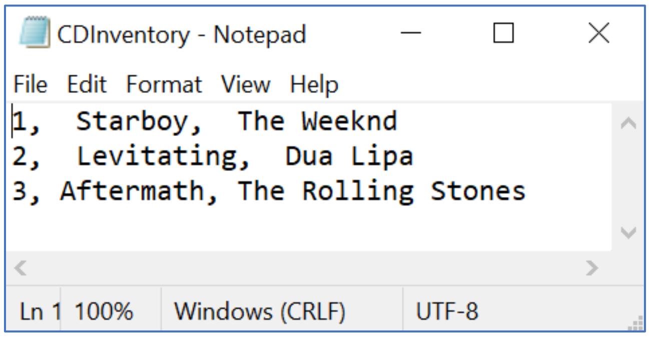


```
In [164]: runfile('C:/School/Python 101/Mod_05/
CDInventory.py', wdir='C:/School/Python 101/Mod_05')
The Magic CD Inventory


[l] load Inventory from file
[a] Add CD
[i] Display Current Inventory
[d] delete CD from Inventory
[s] Save Inventory to file
[x] exit

l, a, i, d, s or x: l


[l] load Inventory from file
[a] Add CD
[i] Display Current Inventory
[d] delete CD from Inventory
[s] Save Inventory to file
[x] exit

l, a, i, d, s or x: i

ID, CD Title, Artist
1,  Starboy,  The Weeknd
2,  Levitating,  Dua Lipa
3,  Cheese,  Stromae

[l] load Inventory from file
[a] Add CD
[i] Display Current Inventory
[d] delete CD from Inventory
[s] Save Inventory to file
[x] exit

l, a, i, d, s or x: d


ID, CD Title, Artist
1,  Starboy,  The Weeknd
2,  Levitating,  Dua Lipa
3,  Cheese,  Stromae


Please select the ID of the CD to be deleted:
3

[l] load Inventory from file
[a] Add CD
[i] Display Current Inventory
[d] delete CD from Inventory
[s] Save Inventory to file
[x] exit

l, a, i, d, s or x: i

ID, CD Title, Artist
1,  Starboy,  The Weeknd
2,  Levitating,  Dua Lipa

[l] load Inventory from file
[a] Add CD
[i] Display Current Inventory
[d] delete CD from Inventory
[s] Save Inventory to file
[x] exit
```



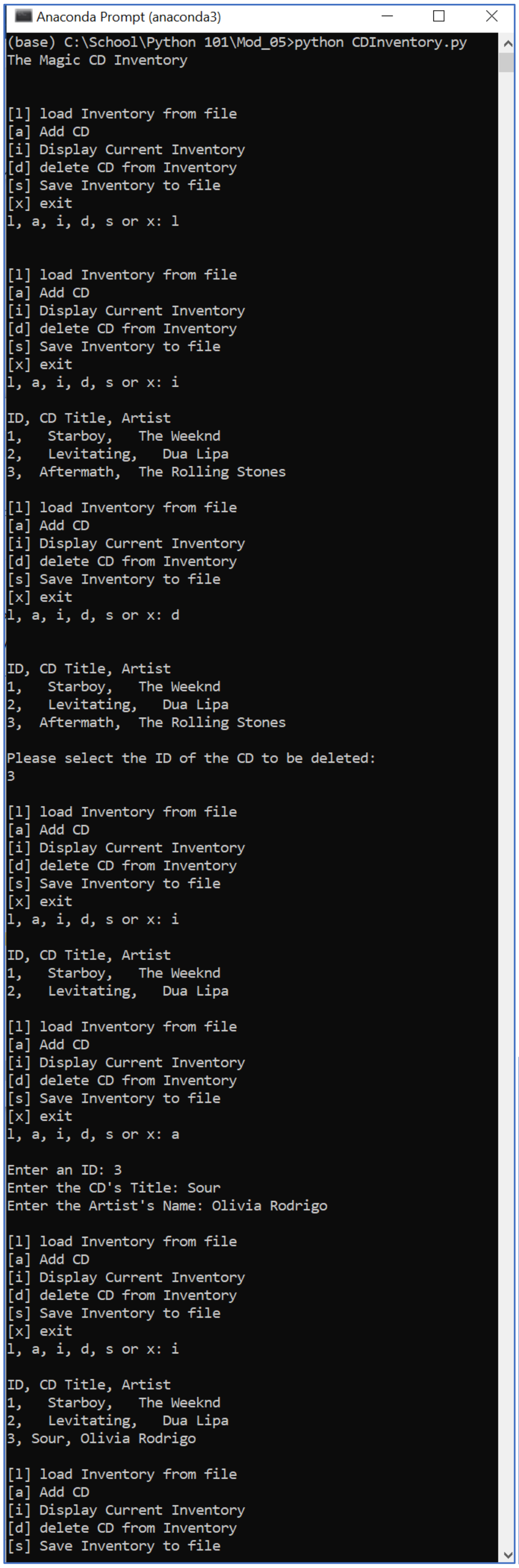
```
[l] load Inventory from file
[a] Add CD
[i] Display Current Inventory
[d] delete CD from Inventory
[s] Save Inventory to file
[x] exit

l, a, i, d, s or x: a


Enter an ID: 3

Enter the CD's Title: Aftermath

Enter the Artist's Name: The Rolling Stones

[l] load Inventory from file
[a] Add CD
[i] Display Current Inventory
[d] delete CD from Inventory
[s] Save Inventory to file
[x] exit

l, a, i, d, s or x: i

ID, CD Title, Artist
1,  Starboy,  The Weeknd
2,  Levitating,  Dua Lipa
3, Aftermath, The Rolling Stones

[l] load Inventory from file
[a] Add CD
[i] Display Current Inventory
[d] delete CD from Inventory
[s] Save Inventory to file
[x] exit

l, a, i, d, s or x: s


[l] load Inventory from file
[a] Add CD
[i] Display Current Inventory
[d] delete CD from Inventory
[s] Save Inventory to file
[x] exit

l, a, i, d, s or x: x


In [165]:
```

Figure 2 – Example output from working script in Spyder IDE

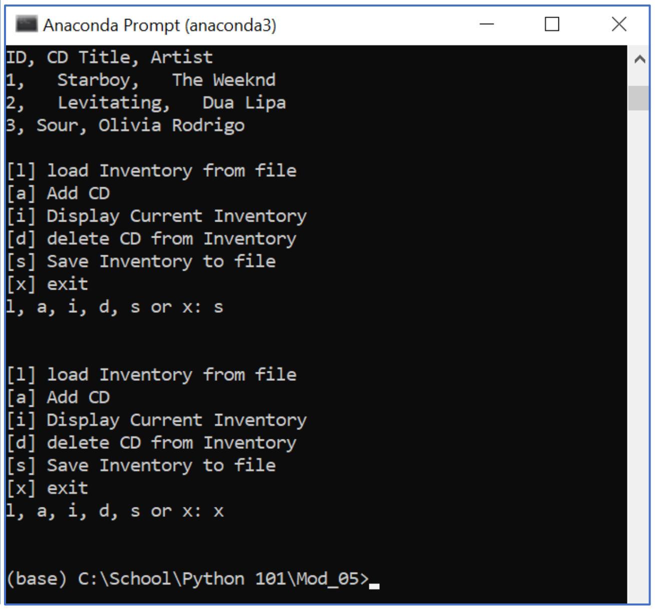


```
1,  Starboy,  The Weeknd
2,  Levitating,  Dua Lipa
3, Aftermath, The Rolling Stones
```

Figure 3 - CDInventory.txt generated by the script after it was run in Spyder IDE



```
(base) C:\School\Python 101\Mod_05>python CDInventory.py
The Magic CD Inventory


[l] load Inventory from file
[a] Add CD
[i] Display Current Inventory
[d] delete CD from Inventory
[s] Save Inventory to file
[x] exit
l, a, i, d, s or x: l


[l] load Inventory from file
[a] Add CD
[i] Display Current Inventory
[d] delete CD from Inventory
[s] Save Inventory to file
[x] exit
l, a, i, d, s or x: i

ID, CD Title, Artist
1,   Starboy,   The Weeknd
2,   Levitating,   Dua Lipa
3,  Aftermath,  The Rolling Stones

[l] load Inventory from file
[a] Add CD
[i] Display Current Inventory
[d] delete CD from Inventory
[s] Save Inventory to file
[x] exit
l, a, i, d, s or x: d


ID, CD Title, Artist
1,   Starboy,   The Weeknd
2,   Levitating,   Dua Lipa
3,  Aftermath,  The Rolling Stones

Please select the ID of the CD to be deleted:
3

[l] load Inventory from file
[a] Add CD
[i] Display Current Inventory
[d] delete CD from Inventory
[s] Save Inventory to file
[x] exit
l, a, i, d, s or x: i

ID, CD Title, Artist
1,   Starboy,   The Weeknd
2,   Levitating,   Dua Lipa

[l] load Inventory from file
[a] Add CD
[i] Display Current Inventory
[d] delete CD from Inventory
[s] Save Inventory to file
[x] exit
l, a, i, d, s or x: a

Enter an ID: 3
Enter the CD's Title: Sour
Enter the Artist's Name: Olivia Rodrigo

[l] load Inventory from file
[a] Add CD
[i] Display Current Inventory
[d] delete CD from Inventory
[s] Save Inventory to file
[x] exit
l, a, i, d, s or x: i

ID, CD Title, Artist
1,   Starboy,   The Weeknd
2,   Levitating,   Dua Lipa
3, Sour, Olivia Rodrigo

[l] load Inventory from file
[a] Add CD
[i] Display Current Inventory
[d] delete CD from Inventory
[s] Save Inventory to file
```



```
ID, CD Title, Artist
1,   Starboy,   The Weeknd
2,   Levitating,   Dua Lipa
3, Sour, Olivia Rodrigo

[l] load Inventory from file
[a] Add CD
[i] Display Current Inventory
[d] delete CD from Inventory
[s] Save Inventory to file
[x] exit
l, a, i, d, s or x: s


[l] load Inventory from file
[a] Add CD
[i] Display Current Inventory
[d] delete CD from Inventory
[s] Save Inventory to file
[x] exit
l, a, i, d, s or x: x


(base) C:\School\Python 101\Mod_05>
```

Figure 4 - Example of the program running in a terminal.

Summary

Throughout this module the foundation of how we store and handle data was further expanded with the introduction of dictionaries and how different types of data is handled between strings, lists and dictionaries. There are many areas in my CDInventory script that stick out to me that could use improvement. A lot of the functionality within the script is repeated, which means each update required will likely require three separate updates to similar functionality instead of just one update which gets consumed throughout the rest of the script and would likely condense the lines of code that require debugging with future updates considerably.