Alex Angelico
August 11, 2020
Foundations of Programming (Python)
CD Inventory With Dictionaries

Dictionaries and Code Organization

Introduction

Expanding on the concept of data collections, this assignment will primarily discuss dictionaries, and how they can be employed in Python programming. This will be achieved by comparatively analyzing the differences between dictionaries and lists. To illustrate these differences, the assignment will examine a modified version of CDInventory.py. In doing so, the importance of 2-dimensional data collections will also be elaborated on. Subsequently, the assignment will answer some questions about best practices for organizing code.

Describing Dictionaries in Comparison to Lists

While lists are a sequence type of data collection, dictionaries are a mapping type of data collection. What this means is that unlike in lists, where individual items of data are accessed in linear order, data items in a dictionary are mapped to non-linear referents which may be accessed in any order.[1]

```
37.     #view inventory contents
38.     if strChoice == 'l':
39.         objFile = open(strFileName, 'r')
40.         print("ID | Artist | Title | # Songs")
41.         for row in objFile:
42.             tempRow = row.strip().split(',')
43.             dicRow = {'ID': int(tempRow[0]), 'Artist': tempRow[1], 'Title': tempRow[2], '# Songs': te
    mpRow[3]}
44.             dispTbl.append(dicRow)
45.             dicRow = f"{dicRow['ID']} | {dicRow['Artist']} | {dicRow['Title']} | {dicRow['# Songs']}"

46.             print(dicRow)
47.         objFile.close()
48.         print()
```

Listing 1 - Excerpt, Appendix A

The section of code taken from Appendix A in Listing 1 provides a succinct example of the differing mechanisms used to access data items in lists and dictionaries. On line 42, a list variable called tempRow is created by extractive and parsing items of data from each line, or row, of a text file. Subsequently, on line 43, an index of tempRow, denoted by the [] pairs containing integers, is used to transcribe the individual data items from the list into a new dictionary variable called dicRow. In the context of the dictionary, these whole data items are reduced to values, which are only accessible from the dictionary with the use of a corresponding key that is listed to the left of the colon next to each tempRow index.

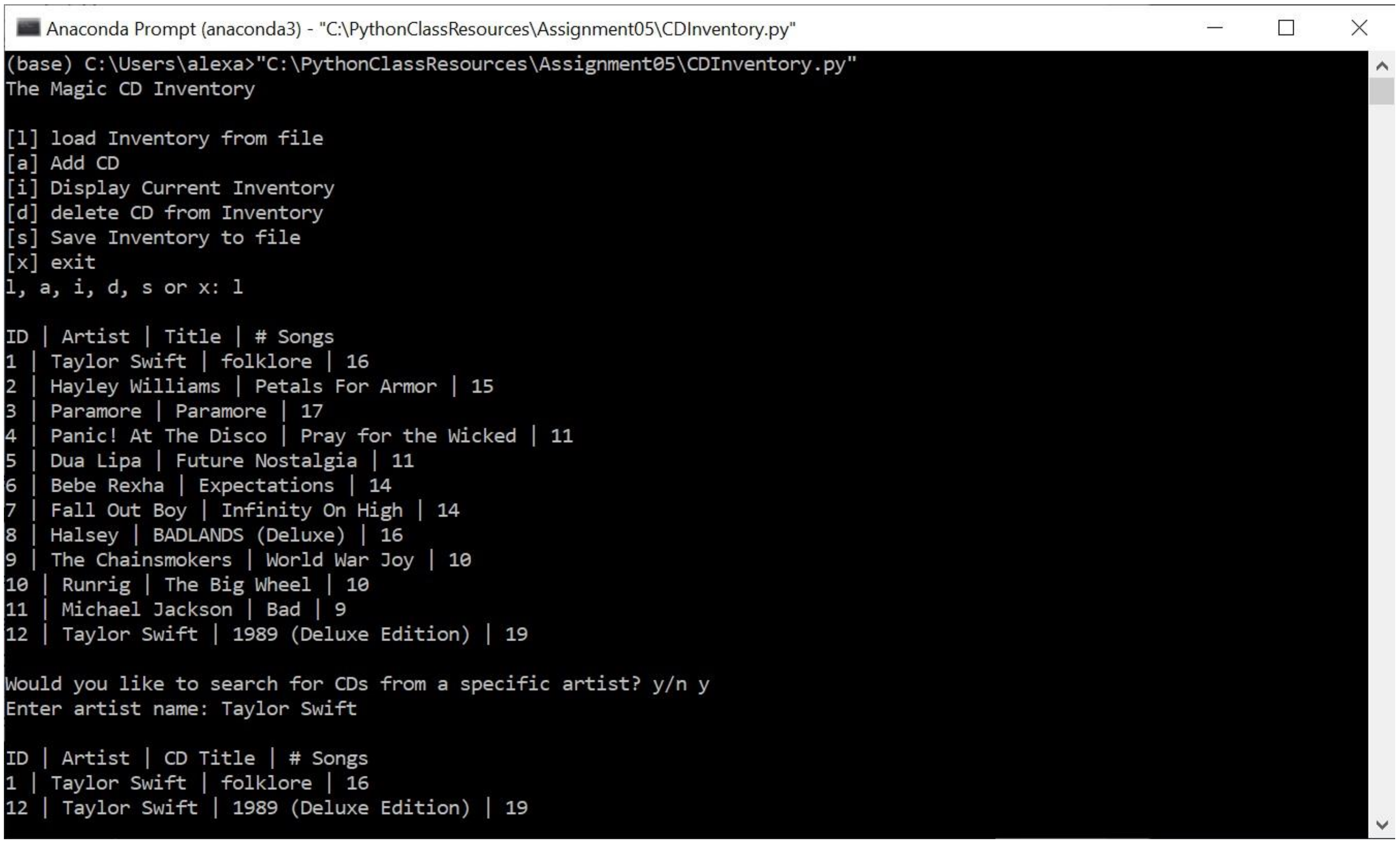
What is critical to note is that although the tempRow items were mapped to dicRow in the same order they were initially sequenced in the list, this is not necessary in order for Python to correctly interpret how data is being transformed from one variable to the other, nor to preserve the functionality of the dictionary. Linear numerical ordering has no bearing on how data is organized in a dictionary. Rather than needing to know a value's numerical position in a dictionary to access it, all that is needed is its associated key. This is because while an index only provides simple positional information, a key is a contextual identifier for a specific value.

[1] "Module 05 of 10", *Foundations of Programming (Python),* Dirk Biesinger, p7 (accessed August 09, 2020).

```python
        #optional search parameter for CDs by artist
        subsearch = input('Would you like to search for CDs from a specific artist? y/n ').lower()
        if subsearch == 'y':
            artist_search = input('Enter artist name: ')
            print('\nID | Artist | CD Title | # Songs')
            rowcount = int()
            cdcount = int()
            while rowcount < len(dispTbl):
                for row in dispTbl:
                    row.get(artist_search)
                    if artist_search == row['Artist']:
                        rowcount += 1
                        cdcount += 1
                        row = f"{row['ID']} | {row['Artist']} | {row['Title']} | {row['# Songs']}"
                        print(row)
                    else:
                        rowcount += 1
                        continue
            if cdcount == 0: print('There are no CDs by that artist.')
```

Listing 2 - Excerpt, Appendix A

The utility of keys to designing clearer programming is evidenced above. Due to how dictionaries are structured, the 2-dimensional list of dictionaries created in Listing 1, dispTbl, can now be easily and rapidly manipulated to return a subset of the overall CDInventory.txt file. All that is required is a single user input search parameter—an artist name—which can then be cross-referenced against all the values attributed to the key 'Artist' on each row of the list. This same set of instructions is also capable of determining if user input does not match any values for 'Artist' and notifying the user. Incidentally, queries of this nature portray why 2-dimensional data organization is such a powerful concept: it enables programmers to use small blocks of code to efficiently extract information in a highly readable way for users.


Anaconda Prompt (anaconda3) - "C:\PythonClassResources\Assignment05\CDInventory.py"

```
(base) C:\Users\alexa>"C:\PythonClassResources\Assignment05\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

ID | Artist | Title | # Songs
1 | Taylor Swift | folklore | 16
2 | Hayley Williams | Petals For Armor | 15
3 | Paramore | Paramore | 17
4 | Panic! At The Disco | Pray for the Wicked | 11
5 | Dua Lipa | Future Nostalgia | 11
6 | Bebe Rexha | Expectations | 14
7 | Fall Out Boy | Infinity On High | 14
8 | Halsey | BADLANDS (Deluxe) | 16
9 | The Chainsmokers | World War Joy | 10
10 | Runrig | The Big Wheel | 10
11 | Michael Jackson | Bad | 9
12 | Taylor Swift | 1989 (Deluxe Edition) | 19

Would you like to search for CDs from a specific artist? y/n y
Enter artist name: Taylor Swift

ID | Artist | CD Title | # Songs
1 | Taylor Swift | folklore | 16
12 | Taylor Swift | 1989 (Deluxe Edition) | 19
```

Figure 1 - Combined function of inventory view and search code blocks, featuring 2-dimensional list of dictionaries

Figure 1 shows how the use of dictionaries in a 2-dimensional structure facilitate a good user experience.

Various Best Practices for Organizing Code

As programs become more sophisticated, such as a CD Inventory program with many user options that implement complex data structures, it becomes important for programmers to organize their code in a way that will make it understandable to the original author, or others, long after the code was originally written. A few ways to do that are described below.

Separation of Concerns

The principle of separation of concerns merely requires programmers to write and visually demarcate their code in organized blocks that correspond to the main functional components of the program. Generally speaking, these components are:

1. Data
2. Processing
3. Presentation (Input/Output)

For additional clarity, these demarcations can be included in comments.[2] To an extent, this approach has been implemented in the CD Inventory program, as can be seen by the comments which highlight the function of individual code blocks in Appendix A.

Functions

Functions provide two significant and related benefits to programmers: more economical code, and greater ability to adhere to separation of concerns. This is because functions must be defined before they can be called in Python, after which they can be executed simply by calling the function's name.[3] This might, for example, allow the processing component to be cleanly separated from the presentation component, where otherwise some form of processing would have to happen concurrent with input or output. As currently written, the CD Inventory program does not take advantage of functions; however, one can see where a function call might replace existing code for retrieving and output text file data in the "view" and "delete" sections.

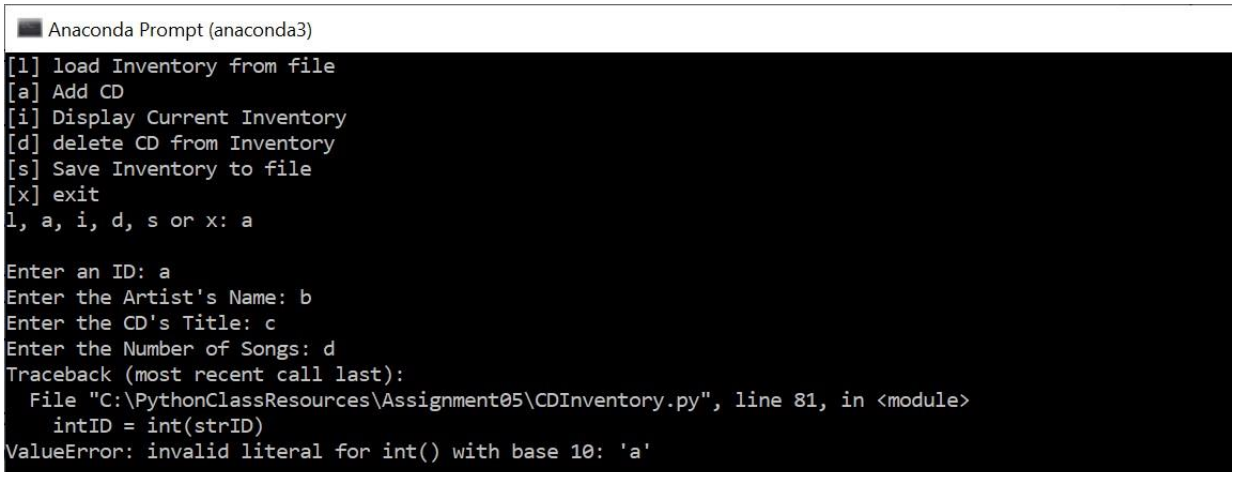
Script Templates

At a more basic level, script templates can be used to make the creation of new scripts a more convenient and organized process. A template can lay out the commented sections that a programmer fill with descriptive information about a script, as well as documentation of its changes. Not only that, a template could be configured to preload a new script with clear labels for separation of concerns.

Structured Error Handling

Python provides a useful construct that allows programmers to control how scripts respond to errors in a way that averts execution crashes and potential data loss.[4] It is called try-except, and implementing it allows a script to output a context-rich error message without interrupting execution in the event of a data manipulation or processing failure. In the case of the CD Inventory program, a potential application of try-except would be handle errors arising from non-numerical user input in the ID field of the "add" section, which currently results in a crash when subsequent instructions attempt to convert the data to an integer-type variable. An example of this crash can be seen below in Figure 2.

[2] Ibid, p13 (accessed August 09, 2020).
[3] Ibid, p14 (accessed August 09, 2020).
[4] Ibid, p16 (accessed August 09, 2020).



```
Anaconda Prompt (anaconda3)
[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: a
Enter the Artist's Name: b
Enter the CD's Title: c
Enter the Number of Songs: d
Traceback (most recent call last):
  File "C:\PythonClassResources\Assignment05\CDInventory.py", line 81, in <module>
    intID = int(strID)
ValueError: invalid literal for int() with base 10: 'a'
```

Figure 2 - Uncontrolled error resulting from unexpected input type

Github

Lastly, serious programmers, whether they are coding in Python or any other language, will almost certainly be using a version control system of some kind. Version control systems allow individual programmers or teams of programmers to collaborate on a shared codebase while tracking changes and preserving past iterations of projects that are available to revert to and back up from, in case of local data loss or changes to code that cause a serious malfunction. Perhaps the most popular solution for version control is Github, because it is entirely based online and easy to scale to any kind of collaboration. Not to mention, it has the creepily cute mascot Octocat.

Summary

Becoming familiarized with dictionaries in this assignment in the context of modifying existing code was a challenge that I feel heightened my learning of all the concepts and practices involved. I look forward to making use of separation of concerns and functions in subsequent weeks.

Appendix

A – Modified CD Inventory Code

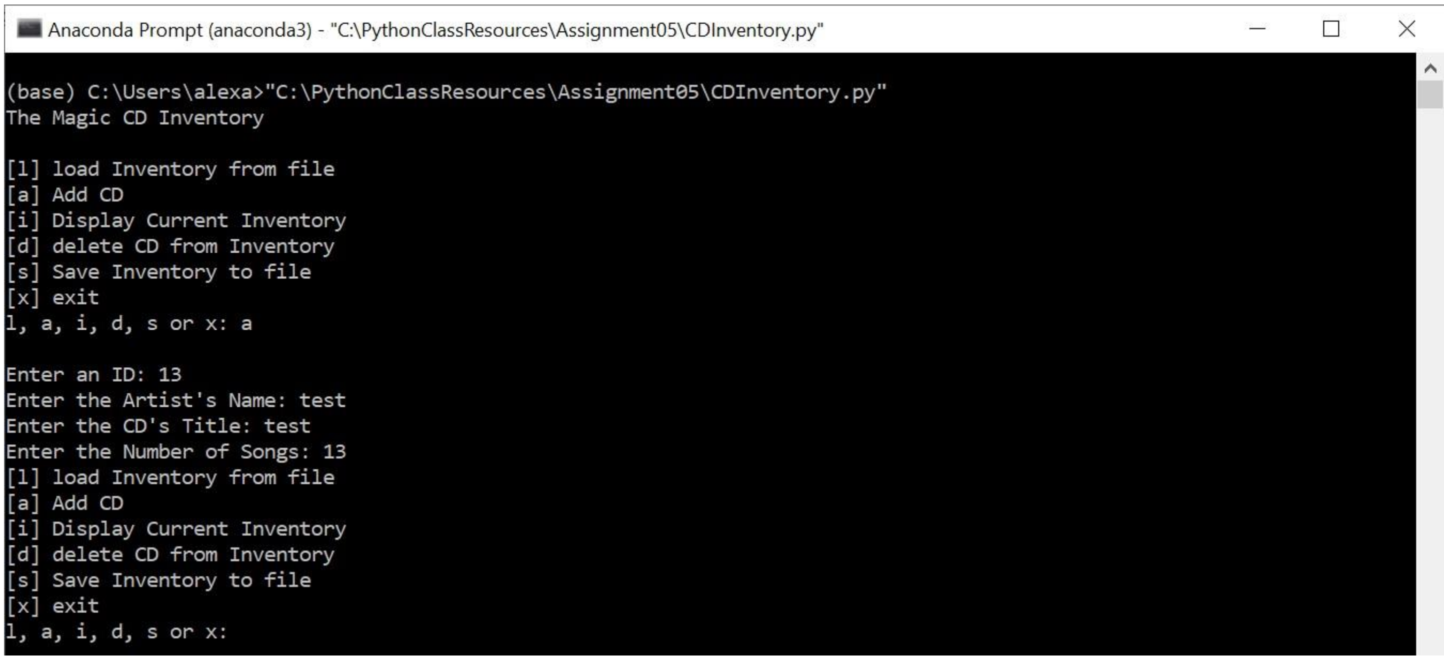
```
#declare variables
strChoice = ''
lstTbl = []
dispTbl = []
delTbl = []
dicRow = {}
strFileName = 'CDInventory.txt'
objFile = None

#generate user input with reiterating menu
print('The Magic CD Inventory\n')
while True:
    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()
    print()

    #exit program
    if strChoice == 'x':
        break

    #view inventory contents
    if strChoice == 'l':
        objFile = open(strFileName, 'r')
```

```python
        print("ID | Artist | Title | # Songs")
        for row in objFile:
            tempRow = row.strip().split(',')
            dicRow = {'ID': int(tempRow[0]), 'Artist': tempRow[1], 'Title': tempRow[2], '# Songs': tempRow[3]}
            dispTbl.append(dicRow)
            dicRow = f"{dicRow['ID']} | {dicRow['Artist']} | {dicRow['Title']} | {dicRow['# Songs']}"
            print(dicRow)
        objFile.close()
        print()

        #optional search parameter for CDs by artist
        subsearch = input('Would you like to search for CDs from a specific artist? y/n ').lower()
        if subsearch == 'y':
            artist_search = input('Enter artist name: ')
            print('\nID | Artist | CD Title | # Songs')
            rowcount = int()
            cdcount = int()
            while rowcount < len(dispTbl):
                for row in dispTbl:
                    row.get(artist_search)
                    if artist_search == row['Artist']:
                        rowcount += 1
                        cdcount += 1
                        row = f"{row['ID']} | {row['Artist']} | {row['Title']} | {row['# Songs']}"
                        print(row)
                    else:
                        rowcount += 1
                        continue
            if cdcount == 0: print('There are no CDs by that artist.')
            dispTbl.clear()
        elif subsearch == 'n':
            dispTbl.clear()
            print()
        print()

    #add new CD data collection
    elif strChoice == 'a':
        strID = input('Enter an ID: ')
        strArtist = input('Enter the Artist\'s Name: ')
        strTitle = input('Enter the CD\'s Title: ')
        strSongNum = input('Enter the Number of Songs: ')
        intID = int(strID)
        dicRow = {'ID': intID, 'Artist': strArtist, 'Title': strTitle, '# Songs': strSongNum}
        lstTbl.append(dicRow)

    #view new CD data collections ready for inventory file
    elif strChoice == 'i':
        print('ID | Artist | CD Title | # Songs')
        for row in lstTbl:
            row = f"{row['ID']} | {row['Artist']} | {row['Title']} | {row['# Songs']}"
            print(row)
        print()

    #delete CD from inventory
    elif strChoice == 'd':
        #I/O for CD selection
        objFile = open(strFileName, 'r')
        print("ID | Artist | Title | # Songs")
        for row in objFile:
            tempRow = row.strip().split(',')
            dicRow = {'ID': int(tempRow[0]), 'Artist': tempRow[1], 'Title': tempRow[2], '# Songs': tempRow[3]}
            delTbl.append(dicRow)
            dicRow = f"{dicRow['ID']} | {dicRow['Artist']} | {dicRow['Title']} | {dicRow['# Songs']}"
            print(dicRow)
```

```
                objFile.close()
                print()
                id_search = int(input('Enter the ID you want to delete: '))
                for row in delTbl:
                    row.get(id_search)
                    if id_search == row['ID']:
                        delTbl.remove(row)
                    else:
                        continue

                #simulated deletion via file rewrite
                lstTbl = delTbl
                print()
                objFile = open(strFileName, 'w')
                for row in lstTbl:
                    row = (row['ID'],row['Artist'],row['Title'],row['# Songs'])
                    strRow = ''
                    for item in row:
                        strRow += str(item) + ','
                    strRow = strRow[:-1] + '\n'
                    objFile.write(strRow)
                objFile.close()

            #save new CD(s) to inventory
            elif strChoice == 's':
                objFile = open(strFileName, 'a')
                for row in lstTbl:
                    row = (row['ID'],row['Artist'],row['Title'],row['# Songs'])
                    strRow = ''
                    for item in row:
                        strRow += str(item) + ','
                    strRow = strRow[:-1] + '\n'
                    objFile.write(strRow)
                objFile.close()
            else:
                print('Please choose either l, a, i, d, s or x!')
```

B – Command Line Execution of CD Inventory Options

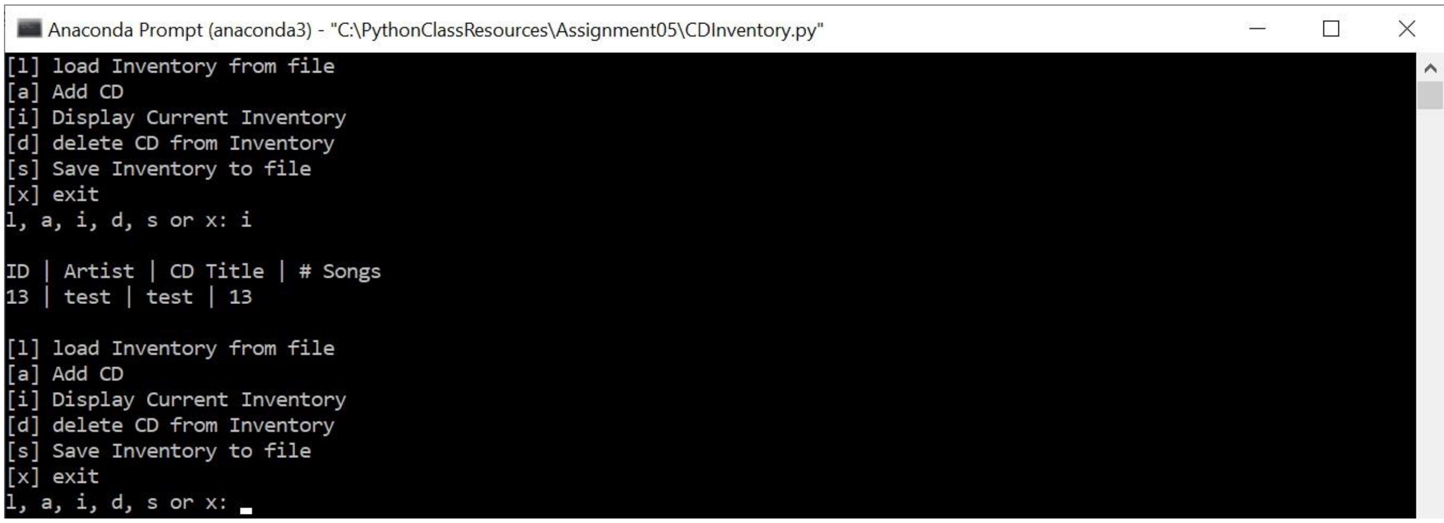
I. Add CD data



Anaconda Prompt (anaconda3) - "C:\PythonClassResources\Assignment05\CDInventory.py"

```
(base) C:\Users\alexa>"C:\PythonClassResources\Assignment05\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: a

Enter an ID: 13
Enter the Artist's Name: test
Enter the CD's Title: test
Enter the Number of Songs: 13
[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:
```

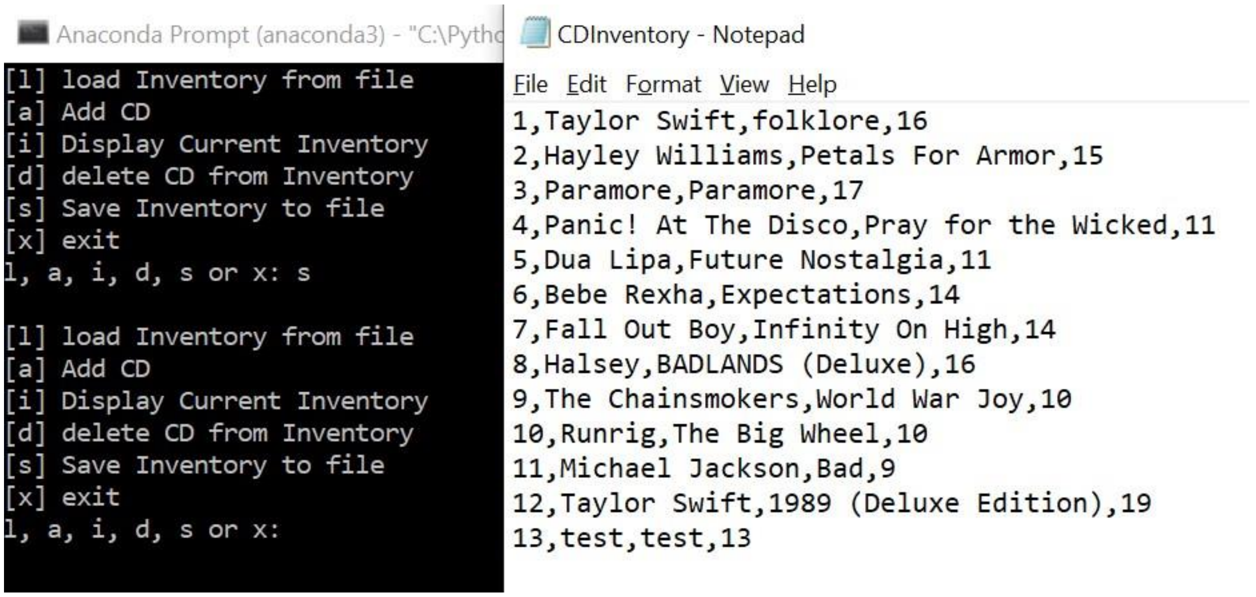
II. Display new additions



Anaconda Prompt (anaconda3) - "C:\PythonClassResources\Assignment05\CDInventory.py"

```
[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 | Artist | CD Title | # Songs
13 | test | test | 13

[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:
```

III. Save data to file



Anaconda Prompt (anaconda3) - "C:\Pytho

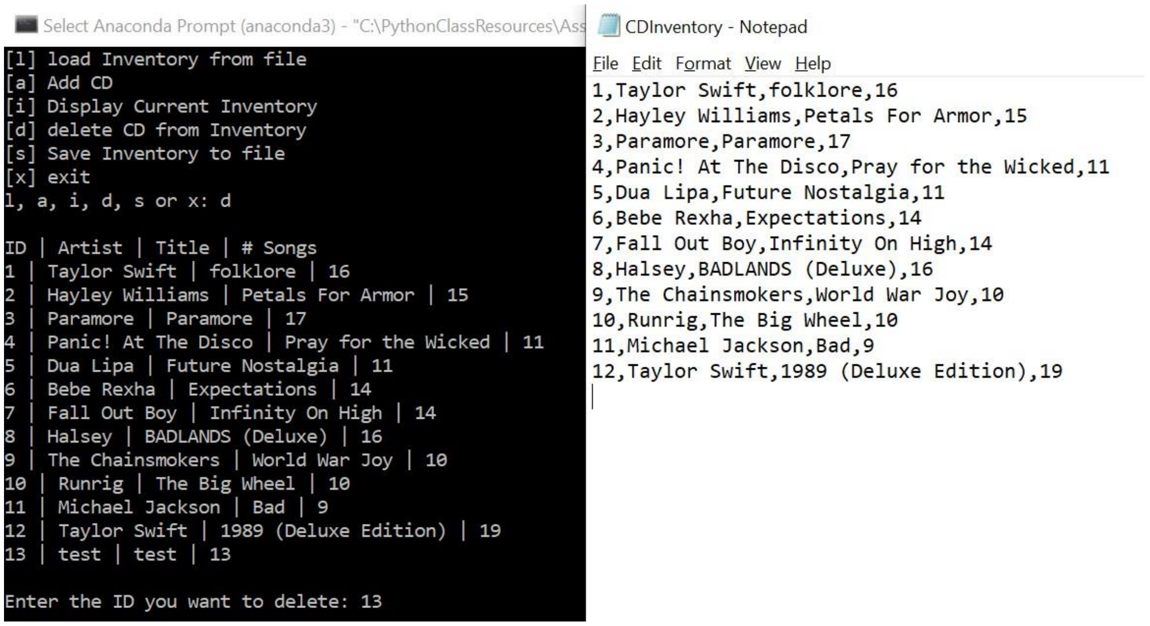
```
[l] load Inventory from file
[a] Add CD
[i] Display Current Inventory
[d] delete CD from Inventory
[s] Save Inventory to file
[x] exit
l, a, i, d, s or x: s

[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:
```

CDInventory - Notepad

File Edit Format View Help

```
1,Taylor Swift,folklore,16
2,Hayley Williams,Petals For Armor,15
3,Paramore,Paramore,17
4,Panic! At The Disco,Pray for the Wicked,11
5,Dua Lipa,Future Nostalgia,11
6,Bebe Rexha,Expectations,14
7,Fall Out Boy,Infinity On High,14
8,Halsey,BADLANDS (Deluxe),16
9,The Chainsmokers,World War Joy,10
10,Runrig,The Big Wheel,10
11,Michael Jackson,Bad,9
12,Taylor Swift,1989 (Deluxe Edition),19
13,test,test,13
```

IV. Delete data from file



Select Anaconda Prompt (anaconda3) - "C:\PythonClassResources\Ass

```
[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 | Artist | Title | # Songs
1 | Taylor Swift | folklore | 16
2 | Hayley Williams | Petals For Armor | 15
3 | Paramore | Paramore | 17
4 | Panic! At The Disco | Pray for the Wicked | 11
5 | Dua Lipa | Future Nostalgia | 11
6 | Bebe Rexha | Expectations | 14
7 | Fall Out Boy | Infinity On High | 14
8 | Halsey | BADLANDS (Deluxe) | 16
9 | The Chainsmokers | World War Joy | 10
10 | Runrig | The Big Wheel | 10
11 | Michael Jackson | Bad | 9
12 | Taylor Swift | 1989 (Deluxe Edition) | 19
13 | test | test | 13

Enter the ID you want to delete: 13
```

CDInventory - Notepad

File Edit Format View Help

```
1,Taylor Swift,folklore,16
2,Hayley Williams,Petals For Armor,15
3,Paramore,Paramore,17
4,Panic! At The Disco,Pray for the Wicked,11
5,Dua Lipa,Future Nostalgia,11
6,Bebe Rexha,Expectations,14
7,Fall Out Boy,Infinity On High,14
8,Halsey,BADLANDS (Deluxe),16
9,The Chainsmokers,World War Joy,10
10,Runrig,The Big Wheel,10
11,Michael Jackson,Bad,9
12,Taylor Swift,1989 (Deluxe Edition),19
```

C – Github Link

https://github.com/Alex-Angelico/Assignment_05