Christine Buffalow
2021-Feb-13
Foundations of Programming
Assignment 5

Creating a CD Inventory, version 2.0

Introduction

For this assignment, we were given a Python file with an example solution from our first CD Inventory project. For version 2.0, our task was to update the file so that it used dictionaries instead of lists for saving the CDs (updating all sections, as necessary). We were also tasked with adding the functionality of (1) loading existing data into memory from a text file and (2) deleting entries from our current inventory. Because of this additional functionality, it was necessary to update some of the code I had used in version 1.0 (specifically my code for automatically generating an id number for each CD).

Data: Loading Modules & Declaring Variables

In this module, we learned the value of organizing our code in a more orderly format by dividing our scripts up into "separation of concerns (SoC)". We were also informed to follow the general flow of (1) data, (2) processing, and (3) presentation. Although I wasn't able to strictly follow this format since I was not allowed the use of functions, I attempted to follow this format as much as I could. Following the lead of the starter script we were provided with, I loaded any necessary modules and declared all my variables at the beginning of the script. This can be seen in Listing 1.

```
16. # --- DATA --- #
17. #Loading modules
18.
19.
20. import os.path
21.
22. # Declaring variables
23. strChoice = '' # user input for menu
24. lstRow = [] # list to hold imported data
25. lstTbl = []  # list of dictionaries to hold data
26. newDataOnlyTbl = [] #list of dictionaries to hold data not saved to file
27. dicRow = {}  # dictionary row of data
28. strFileName = 'CDInventory.txt'  # data storage file
29. objFile = None  # file object
30. objFile_content = '' #empty string for holding file contents
31. objFile_rows = [] #empty list to hold file contents split into separate strings
32. indexFinalRow = 0 #index of final row in data file
33. cd_id = 0 # creating counter for cd_id generator
34. load = 0 # creating flag to prevent importing file more than once
35. deleteRow = -1 #creating delete counter for deleting CDs
36. skipRow = -1 #creating delete counter for deleting CDs
```

Listing 1 - Data Section: Loading Modules and Declaring Variables

The `cd_id` Counter

Because I still wanted to automatically generate a unique integer ID number for each entry, I grabbed the code from the `cd_id` generator I created for my `CDInventory.py`, v. 1.0, script (kindly revised by Doug Klos to be more efficient and to prevent an error from occurring if `CDInventory.txt` did not exist). This code figures out which ID number is next based on the number of lines of data in `CDInventory.txt`. This works by importing all the data into a giant string called `objFile_content`. This string is then split into smaller strings using the `split()` method based on

the location of the new line character ('\n') and saved into a list called `objFile_rows`. Because each CD is saved on a separate line in the data file, this effectively splits each CD into its own string. This means `cd_id` can be calculated by measuring the number of strings inside the `objFile_rows` list using the `len()` function. The resulting code can be seen Figure 1 (note that the `os.path` module was loaded and `cd_id` variable was declared in the "Data" section, see Listing 1 above).

```
# --- PROCESSING --- #

#counting lines of data in CDInventory.txt so I know where to start next ID number
if os.path.exists(strFileName):
    objFile = open(strFileName, 'r')
    objFile_content = objFile.read() #returns all content of file
    objFile_rows = objFile_content.split(sep= '\n') #separates data into separate items based on location of \n
    objFile.close()
    cd_id = len(objFile_rows)
else:
    cd_id = 0
```

Figure 1 - Original cd_id generator, with some revisions

However, after working through the "Deleting" and "Saving" sections, I realized I had an issue. If I continued to allow the user to delete items from the inventory and then overwrite the text file with their current inventory, the length of the text file would no longer be a reliable measure for the next value of cd_id as there could be gaps between ID values. This would essentially result in duplicate IDs and out-of-order IDs near the end of the file. At this time, I realized I needed to remove the "overwrite" functionality in the "Save" section, accept my faulty cd_id generator, or revise my cd_id generator to work under the new conditions.

I decided to go with the third option and find a way to look up what ID number was used in the last line of data in the text file. The way my program is designed, I don't believe there is a way to get the ID numbers out of order (assuming I fix the faulty `cd_id` generator!), so it should be in the last line of data in the file. (If I were to continue making iterations of this program, I would perhaps extract all ID numbers, sort them, and then select the highest value that way. Since I'm trying to not add too much more complexity, I'm going to stick with my assumption that it's not possible to get the ID numbers out of order as long as I use my new solution for generating ID numbers.) It will be possible to have gaps between ID numbers since it is possible to delete items, but I am not concerned with that as long as the ID numbers are unique and increase as you go down through the list.

To begin, I manually determined the necessary index to save the final row of data of `objFile_rows` into `objFile_finalRow` and printed it to confirm accuracy. I also printed the length of `objFile_rows` to see how much this number needed to be adjusted before it could be used as the index of the final row of data. This can all be seen in Figures 2 and 3.

```
#---PROCESSING---#

#counting lines of data in CDInventory.txt so I know where to start next ID number
if os.path.exists(strFileName):
    objFile = open(strFileName, 'r')
    objFile_content = objFile.read() #returns all content of file
    objFile_rows = objFile_content.split(sep= '\n') #separates data into separate items based on location of \n
    objFile_finalRow = objFile_rows[12]
    print(objFile_finalRow)
    print(len(objFile_rows))
    objFile.close()
    #cd_id = 0
else:
    cd_id = 0
```

Figure 2 - Working on a new cd_id generator: isolating final row of data & determining how to calculate index using number of rows in data file

```
13,Crossing a Bridge,Flora Fox
14
```

Figure 3 – Results of code in Figure 2

From this, I determined that it was necessary to subtract 2 from the length in order to get the index of the final row of actual data (this makes since as the last line of the file doesn't contain any useful data and the index starts at zero not one). I saved this information to the variable `indexFinalRow` and used that variable to print out the final row of data of `objFile_rows` and saved it to `objFile_FinalRow`. I included various print statements to test what was happening and to confirm whether or not it was working as expected. This can be seen in Figures 4 and 5.

```
#---PROCESSING---#

#counting lines of data in CDInventory.txt so I know where to start next ID number
if os.path.exists(strFileName):
    objFile = open(strFileName, 'r')
    objFile_content = objFile.read() #returns all content of file as a giant string
    objFile.close()
    objFile_rows = objFile_content.split(sep= '\n') #separates data into separate items based on location of \n
    indexFinalRow = len(objFile_rows) -2
    print(indexFinalRow)
    print(type(indexFinalRow))
    objFile_FinalRow = objFile_rows[indexFinalRow]
    print(objFile_FinalRow)
    print(type(objFile_FinalRow))
else:
    cd_id = 0
```

Figure 4 - Working on a new cd_id generator: using number of rows in data file to determine index of final row of data

```
12
<class 'int'>
13,Crossing a Bridge,Flora Fox
<class 'str'>
```

Figure 5 - Results of code in Figure 4

After this, I performed a split on `objFile_FinalRow` based on the location of the comma to create an iterable list rather than a string (if I didn't do this, I would have difficulty dealing with ID numbers with different numbers of digits). This can be seen in Figures 6 and 7.

```
#---- PROCESSING ----#

#counting lines of data in CDInventory.txt so I know where to start next ID number
if os.path.exists(strFileName):
    objFile = open(strFileName, 'r')
    objFile_content = objFile.read() #returns all content of file as a giant string
    objFile.close()
    objFile_rows = objFile_content.split(sep='\n') #separates data into separate items based on location of \n
    indexFinalRow = len(objFile_rows) -2
    objFile_FinalRow = objFile_rows[indexFinalRow].split(sep=',')
    print(objFile_FinalRow)
    print(type(objFile_FinalRow))
else:
    cd_id = 0
```

Figure 6 - Working on a new cd_id generator: splitting final row of data (currently saved as a string) into list of strings

```
['13', 'Crossing a Bridge', 'Flora Fox']
<class 'list'>
```

Figure 7 - Results of code in Figure 6

Once I confirmed that I had generated a list with the final row of data in the text file, I selected the item in index position zero (the ID number!), cast it to an integer (note that it is a string in Figure 7 above, as indicated by the apostrophes), and saved it to `cd_id`. Moving forward, a value of 1 will need to be added to `cd_id` before the any new cd is added. This is done in the "Adding a CD section". The final results of the `cd_id` generator can be seen in Listing 2.

```
40. # --- PROCESSING --- #
41.
42. #counting lines of data in CDInventory.txt so I know where to start next ID number
43. if os.path.exists(strFileName):
44.     objFile = open(strFileName, 'r')
45.     objFile_content = objFile.read() #returns all content of file as a giant string
46.     objFile.close()
47.     objFile_rows = objFile_content.split(sep= '\n') #separates data into separate items based on loca
    tion of \n
48.     indexFinalRow = len(objFile_rows) -2
49.     cd_id = int(objFile_rows[indexFinalRow].split(sep= ',')[0])
50. else:
51.     cd_id = 0
```

Listing 2 - new cd_id generator

Loading Data

I unintentionally jumped ahead last week and attempted to read from a text file in my script last week, so I already learned the hard way that it's important to add a check to see if the `CDInventory.txt` file exists before attempting to read from it. As I did with the `CD_ID` counter, I used the '`os.path.exist(strFileName)`' code to perform this check. If it is false, it jumps down to the `'else'` branch and prints a basic statement informing the user that the existing inventory does not exist and then returns to the main menu with a `'continue'` statement (much better than an error that would crash the program!). If it is true, the program proceeds to read from the file.

As another check, I also added a `load` flag. The default value of load is "0", which is declared in the beginning of the script. Once the file is loaded into memory, the value of `load` is set to "1". If `load` is equal to 1, the user is sent back to the main menu with an `continue` statement and the data is not loaded again.

Because I wanted to attempt to keep my inventory in order by ID number, I decided to use the following logic as shown in my pseudocode shown in Figure 8:

- When adding a CD, save to both lstTbl and newDataTbl
- When displaying an inventory, always display lstTbl
- When loading data:
  - Clear lstTbl (newly added but unsaved CD data still saved in newDataTbl)
  - Import data from text file into lstTbl
  - Append data from newDataTbl to the end of lstTbl
- When appending to a file during the save process:
  - Save contents of newDataTbl
  - Clear contents of newDataTbl afterwards because the CDs have now been saved to the text file (these CDs are still viewable in lstTbl if user goes to "Display Current Inventory")
- When overwriting to a file during the save process:
  - Save contents from lstTbl
  - Clear contents of newDataTbl afterwards because the CDs have now been saved to the text file via lstTbl (these CDs are still viewable in lstTbl if user goes to "Display Current Inventory")

Figure 8 - pseudocode for adding/loading/saving

After clearing `lstTbl` (reasoning explained in pseudocode above), I open my text file with read access and then read through each row of data using a `'for'` loop. First, each row is split into separate strings based on the location of the delimiter (specified as a comma in this case). Then each individual string is "stripped". In other words, any spaces or new line characters at the end or beginning of the string are removed. Finally, the three strings are saved into a list called `lstRow` (the new data replaces any existing data in the list). Following this, the list is saved into a dictionary called `dicRow`, as explained in the pseudocode in Figure 9.

```
dicRow = {'Key1' : 'Value1', 'Key2' : 'Value2', 'Key3' : 'Value3'}
     Key1 = 'id'
     Value1 = 1st index (position 0) of lstRow cast as an integer
     Key2 = 'title'
     Value2 = 2nd index (position 1) of lstRow
     Key3 = 'artist'
     Value3 = 3rd index (position 2) of lstRow
```

Figure 9 - Pseudocode of dicRow formatting

Once the data has been saved to the dictionary, `dicRow`, it is then appended to `lstTbl`. The list, `lstTbl`, is now a 2D list of dictionaries. (Because the keys are the same for each row, the existing key:value pairs are overwritten in `dicRow` each time as the interpreter runs through the `'for'` loop so it does not need to be cleared.) The process is then repeated for each row of data in the text file until it reaches the end. Once the process is complete, the text file is closed and `newDataOnlyTbl` is appended to the end of `lstTbl`. The table `newDataOnlyTbl` contains any CDs that have been added during the current run of the program but have not yet saved to the text file (explained in pseudocode shown in Figure 8). The code for this section is shown is Listing 3.

```
78.     #----------LOAD----------#
79.
80.     if strChoice == 'l': # Loading data
```

```
        if load == 1:
            print('\nYou have already loaded the inventory from file. Returning to Main Menu.\n')
            continue
        if os.path.exists(strFileName):

            lstTbl = [] #clearing current inventory as all unsaved cds are saved to newDataOnlyTbl
            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)
            objFile.close()
            lstTbl = lstTbl + newDataOnlyTbl #appending new cds to the end of list
            load = 1 #switching flag to 1 so that load process can not be repeated
            print('~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~')
            print('Data has been added. Go to "I - Display Current Inventory" to view!')
            print('~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n')
        else:
            print('~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~')
                  print('There is no existing inventory to load.')
                  print('~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n')
```

Listing 3 - Code for loading data from CDInventory.txt into memory

Adding Data

The "Adding Data" section was pretty straightforward as it had already been written. It just needed to be updated so that it would save the data into a dictionary rather than a list. The pseudocode of my method is shown in Figure 10. (Note that I used a slightly different method for creating the dictionaries in this section than I used in the loading data section. I did this since I am still new to the process and wanted to practice using different techniques.)

`dicRow['key'] = value`

- `key = 'id', 'title', or 'artist'`
- `value =`
  - `automatically generated for id`
  - `input from user for 'title' and 'artist'`

Figure 10 - Pseudocode for adding data

Once the three values have been generated or input by the user, the dictionary, `dicRow`, is appended to the list, `lstTbl`. Note that `cd_id` is increased by one before creating the dictionary to reflect that the ID of the newly added CD will be one higher than the previously calculated value. At the end of this process, a summary statement is printed and then the user is returned to the main menu. The code can be seen in Listing 4.

```
            #----------ADD----------#

            elif strChoice == 'a':  # Allowing user to add data to the table as dictionary
                print('~~~~~~~~')
                print('Add a CD')
                print('~~~~~~~~\n')
                dicRow = {} #preventing overwriting as I am using same keys for every dictionary row
                cd_id += 1
                dicRow['id'] = cd_id
                dicRow['title'] = input('Enter the CD\'s Title: ')
                dicRow['artist'] = input('Enter the Artist\'s Name: ')
                lstTbl.append(dicRow) #adding data to current inventory view
                newDataOnlyTbl.append(dicRow) #adding all unsaved data to a separate table
                print('\n~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~')
                print('Data has been added. Go to "I - Display Current Inventory" to view!')
```

```
119.            print('~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n')
```

Listing 4 - Code for adding additional CDs

Displaying Data

The "Displaying Data" section was also pretty simple as it had essentially already been written and just needed to be modified. In the beginning, I just replaced `'row'` with `'*row.values()'` as shown in Figure 11. This printed out a basic comma separated list as shown in Figure 12.

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

Figure 11 - Initial code for displaying data

```
~~~~~~~~~~~~~~~~~~~~
Current Inventory
~~~~~~~~~~~~~~~~~~~~

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

ID, CD Title, Artist

1, Bursting Bubbles, Katie Peters
2, Courage of Fools, Myrtle Warner
3, Knowledge Bomb, Brian Thomas
4, Blank Canvas, Jonthan Santiago
5, Emotional Wreckage, Gertrude Curry
6, Battleground, Brandi Watkins
7, The Bigger Fish, Jonathan Santiago
8, New Dimension, Kathy Christensen
9, No Ambition, Flora Fox
10, Rain Check, Adam Walker
11, Blissful Ignorance, Dana Burke
12, Zero Gravity, Adam Walker
13, Crossing a Bridge, Flora Fox
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

Figure 12 - Results of code from Figure 11

In a later iteration, I experimented more with formatting to make it more presentable. After reviewing some ideas on StackOverflow[1], I added a `'format()'` method with specific "column widths" as arguments (I didn't like the option of using tabs as separators between list items because it never quite lined up when you had CD Titles with different lengths!). The pseudocode of my method is shown in Figure 13. This same formatting was used for the headers ("ID", "CD Title", and "Artist") so that everything would line up. The code for this can be seen in Listing 5 and the results of my list with this type of formatting can be seen in Figure 14.

[1] https://stackoverflow.com/questions/17330139/python-printing-a-dictionary-as-a-horizontal-table-with-headers, accessed 2020-Feb-13.

```
print('{value1, aligned to left, 5 characters wide}{value2, aligned to left, 25 characters wide}{value3, aligned to left, 25 characters wide}'.format(values 1-3)))
```

Figure 13 - Pseudocode for formatting displayed list of CDs

```python
        #----------DISPLAY----------#

        elif strChoice == 'i':  # Displaying the current data to the user
            print('~~~~~~~~~~~~~~~~~')
            print('Current Inventory')
            print('~~~~~~~~~~~~~~~~~\n')
            print('~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n')
            print('{:<5}{:<25}{:<25}'.format('ID', 'CD Title', 'Artist'))
            print()
            for row in lstTbl:
                print('{:<5}{:<25}{:<25}'.format(*row.values()))
            print('~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n')
```

Listing 5 - Code for displaying data

```
~~~~~~~~~~~~~~~~~
Current Inventory
~~~~~~~~~~~~~~~~~

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

ID   CD Title                 Artist

1    Bursting Bubbles         Katie Peters
2    Courage of Fools         Myrtle Warner
3    Knowledge Bomb           Brian Thomas
4    Blank Canvas             Jonthan Santiago
5    Emotional Wreckage       Gertrude Curry
6    Battleground             Brandi Watkins
7    The Bigger Fish          Jonathan Santiago
8    New Dimension            Kathy Christensen
9    No Ambition              Flora Fox
10   Rain Check               Adam Walker
11   Blissful Ignorance       Dana Burke
12   Zero Gravity             Adam Walker
13   Crossing a Bridge        Flora Fox
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

Figure 14 - Result of formatting displayed list of CDs with designated "column widths"

In reality, the value within the brackets (either 5 or 25 in this case) is not a "column width" but rather the length of a new string with spaces used as padding to create a string length of exactly the specified number of characters. In this case, since I had 3 strings with no separation between them, I essentially concatenated a string of length of exactly 5 with two strings of length of exactly 25. I explored this more by experimenting in an empty script. As used in my script, the `format()` method essentially created is a string of length 55, which is equal to the length of the 3 separate strings added together. This can be seen in Figures 15 and 16.

```
myObj = '{:<5}{:<25}{:<25}'.format('ID', 'CD Title', 'Artist')
print(myObj)
print(type(myObj))
print(len(myObj))
x = 5+25+25
print(x)
```

Figure 15 - Exploring format() method

```
ID   CD Title                  Artist
<class 'str'>
55
55
```

Figure 16 - Results of code in Figure 15

Deleting an Entry

Figuring out how to delete an entry was by far the biggest challenge of this assignment. However, instead of going straight to the internet for a solution, I decided to see how far I could get using logic and the concepts I learned from the various learning materials of Module 5 (in addition to the printout written by Dirk Biesinger, the link to the "Dictionaries in Python" website from Real Python on our assignment sheet provided me with a lot of great building blocks that helped me know what was possible with dictionaries![2]).

I started by printing out the existing inventory so that I could see which ID numbers were available to delete. I then added a simple test inside of a `'for'` loop that would go through each row of `lstTbl` and print "True" or "False" if the input ID number was in one of the row's values (the integer ID numbers are saved as values in the dictionary). This can be seen in Figures 17 and 18.

```
    elif strChoice == 'd':
        # TODO Add functionality of deleting an entry
        print('~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n')
        print('ID, CD Title, Artist\n')
        for row in lstTbl:
            print(*row.values(), sep = ', ')
        print('~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n')
        deleteMe = input('Please enter the ID number of the entry you would like to delete: ')
        for row in lstTbl:
            if deleteMe in row.values():
                print('True')
            else: print('False')
```

Figure 17 - Code testing to see if specific ID numbers could be located within row.values() inside a 'for' loop

[2] https://realpython.com/python-dicts/, accessed 2021-Feb-13

```
L, A, I, D, S or X: d

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

ID, CD Title, Artist

1, Bursting Bubbles, Katie Peters
2, Courage of Fools, Myrtle Warner
3, Knowledge Bomb, Brian Thomas
4, Blank Canvas, Jonthan Santiago
5, Emotional Wreckage, Gertrude Curry
6, Battleground, Brandi Watkins
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~


Please enter the ID number of the entry you would like to delete: 2
False
True
False
False
False
False
[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
```

Figure 18 - Results of code in Figure 17

This method proved to be successful, so I continued on. Since each entry/row was an individual dictionary and I wanted to delete the entire entry/row, I next tried using the `clear()` method for dictionaries to clear the dictionary that came up as "True." This effectively removed all the contents of the dictionary but left the empty dictionary behind. This can be seen in Figures 19 and 20.

```
    elif strChoice == 'd':
        # TODO Add functionality of deleting an entry
        print('~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n')
        print('ID, CD Title, Artist\n')
        for row in lstTbl:
            print(*row.values(), sep = ', ')
        print('~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n')
        deleteMe = str(input('Please enter the ID number of the entry you would like to delete: '))
        for row in lstTbl:
            if deleteMe in row.values():
                row.clear()
            else: pass
        for row in lstTbl:
            print(*row.values(), sep = ', ')
```

Figure 19 - Clearing contents of specified dictionary using clear()

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

Please enter the ID number of the entry you would like to delete: 2
1, Bursting Bubbles, Katie Peters

3, Knowledge Bomb, Brian Thomas
4, Blank Canvas, Jonthan Santiago
5, Emotional Wreckage, Gertrude Curry
6, Battleground, Brandi Watkins
[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

~~~~~~~~~~~~~~~~~~~
Current Inventory
~~~~~~~~~~~~~~~~~~~

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

Figure 20 - Results of code in Figure 19 – note the empty dictionary is still visible between the 1st and 3rd CDs

I then used the pop() method to delete the empty dictionary from the list. However, the technique I used (taking the desired ID number and subtracting one and using the resulting number as the index), only worked for the first CD that was deleted. The code and the results after deleting one CD are shown in Figures 21 and 22.

```
    elif strChoice == 'd':
        # TODO Add functionality of deleting an entry
        print('~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n')
        print('ID, CD Title, Artist\n')
        for row in lstTbl:
            print(*row.values(), sep = ', ')
        print('~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n')
        deleteMe = str(input('Please enter the ID number of the entry you would like to delete: '))
        for row in lstTbl:
            if deleteMe in row.values():
                row.clear()
            else: pass
        lstTbl.pop(int(deleteMe)-1)
        for row in lstTbl:
            print(*row.values(), sep = ', ')
    elif strChoice == 's':
```

Figure 21 - Using pop() to delete a specified row from lstTbl

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

ID, CD Title, Artist

1, Bursting Bubbles, Katie Peters
2, Courage of Fools, Myrtle Warner
3, Knowledge Bomb, Brian Thomas
4, Blank Canvas, Jonthan Santiago
5, Emotional Wreckage, Gertrude Curry
6, Battleground, Brandi Watkins
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~


Please enter the ID number of the entry you would like to delete: 2
1, Bursting Bubbles, Katie Peters
3, Knowledge Bomb, Brian Thomas
4, Blank Canvas, Jonthan Santiago
5, Emotional Wreckage, Gertrude Curry
6, Battleground, Brandi Watkins
[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
```

Figure 22 - Results from code in Figure 21

As noted, this method only worked for the first CD that was deleted. In my next iteration, I created two new variables: `skipRow` and `deleteRow`. As I iterated through the rows in the for loop, the value of 1 would be added to the value of `skipRow` and the resulting integer would be printed (I later realized it would have been more efficient to put the `skipRow` counter outside of the `'if/else'` statement rather than put it in each one). If the desired ID number was found in `row.values()`, the interpreter would go down the `'if'` branch and save the current `skipRow` value to `deleteRow`. (In order for it to have the correct index number for the row, `skipRow` was set to '-1' in the "Declaring Variables" section, see Listing 1 above). The value of `deleteRow` could then be used as the index for deleting the appropriate row from the list. I also added a string that informed the user that CD# <insert id of CD> was deleted. The results can be seen in Figures 23 and 24.

```
            for row in lstTbl:
                if deleteMe in row.values():
                    row.clear()
                    skipRow += 1
                    deleteRow = skipRow
                    print('deleteRow = ', deleteRow)
                else:
                    skipRow += 1
                    print(skipRow)
            lstTbl.pop(deleteRow)
            print('\n~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~')
            print('CD #', deleteMe, ' deleted. Go to "I - Display Current Inventory" to view updated inventory!')
            print('~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n')
```

Figure 23 - Deleting a specific row using the variable 'deleteRow' as the index

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~


Please enter the ID number of the entry you would like to delete or "C" to Cancel: 5
0
1
2
3
deleteRow =  4
5
6
7
8
9
10

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
CD # 5  deleted. Go to "I - Display Current Inventory" to view updated inventory!
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

    *****MAIN MENU*****
[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

~~~~~~~~~~~~~~~~~
Current Inventory
~~~~~~~~~~~~~~~~~

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

ID, CD Title, Artist

1, Bursting Bubbles, Katie Peters
2, Courage of Fools, Myrtle Warner
3, Knowledge Bomb, Brian Thomas
4, Blank Canvas, Jonthan Santiago
6, Battleground, Brandi Watkins
7, The Bigger Fish, Jonathan Santiago
8, New Dimension, Kathy Christensen
9, No Ambition, Flora Fox
10, Rain Check, Adam Walker
11, Blissful Ignorance, Dana Burke
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

Figure 24 - Results of code in Figure 23

In my next iteration, I realized that if the variable `deleteRow` never changed, that meant that a CD was not found. As such, I just needed to find a default value of `deleteRow` that would be impossible to iterate to using my current structure of starting at "-1" and counting up by 1. Although I technically could have used any negative number, I went with "-1". I added this as an `if` statement after the `for` loop. If `deleteRow` was still equal to "-1", then I printed out a message saying the CD was not found. I also added a `try/except` block to prevent errors if the user entered a non-integer. If the value entered by the user could not be cast to an integer, the user is told that they provided an invalid input and returned to the main menu. At the end, I reset both counters to "-1". The code can be seen in Figure 25.

```
        try:
            int(deleteMe) #testing to see if deleteMe is valid entry
            print(lstTbl)
            print(row.values())
            for row in lstTbl:
                if int(deleteMe) in row.values():
                    row.clear()
                    skipRow += 1
                    deleteRow = skipRow
                    lstTbl.pop(deleteRow)
                    print('\n~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~')
                    print('CD #', deleteMe, ' deleted. Go to "I - Display Current Inventory" to view updated inventory!')
                    print('~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n')
                else:
                    skipRow += 1
            if deleteRow == -1: print('\nCD #', deleteMe, ' not found. Returning to Main Menu.\n')
            deleteRow = -1
            skipRow = -1
        except: print('\nThat is not a valid option. Returning to main menu.\n')
```

Figure 25 - Adding option of "CD not found" to the code

At this point, the code appeared to be fully functional from all my tests. However, later that day during office hours, I realized that I could make a few changes to make the code a little more efficient:

- Clearing the dictionary before deleting it was completely unnecessary, so I removed `row.clear()`
- Casting `deleteMe` to an integer immediately seemed more efficient rather than testing it and then actually casting it to an integer later. I also moved the `except` block higher and added a `continue` to skip the rest of code and return to the main menu if there was an error.
- I debated using `'if row['id'] == deleteMe'` instead of `'if deleteMe in row.values()'` because the first seemed better. However, my test found that my solution worked just fine. The `if/in` statement did not have trouble if the value of the CD Title or Artist were numbers (I assume this is because CD Title and Artist are both strings). I also confirmed that searching if 1 is in `row.values()` does not come up true for integers that have the number 1 in them (ex: 10, 11, etc).
- It would be more efficient to put the `skipRow` counter outside the `if/else` blocks since I was counting with every row
- I could `break` once I found the CD instead of continuing to iterate through the end of the list
- I liked `del` better than `pop()`

Although I also learned that I could use a True/False flag instead of using my `deleteRow` as a flag, I decided to stick with the little solution I came up with. I also added the option for the user to cancel and return to the menu by entering "0" since (1) it's an integer so it won't cause an error and (2) it's not a possible value of `cd_id` (the smallest possible is 1). The final code can be seen in Listing 6.

```
        #----------DELETE----------#

        elif strChoice == 'd': # Providing user the option to delete an entry or cancel
            if len(lstTbl) == 0:
                print('~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~')
                print('Your current inventory is empty. There are no entries to delete.')
                print('~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n')
                continue
            print('~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n')
            print('{:<4}\t{:<25}\t{:<25}'.format('ID', 'CD Title', 'Artist'))
            print()
            for row in lstTbl:
                print('{:<4}\t{:<25}\t{:<25}'.format(*row.values()))
            print('\n~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n')
            deleteMe = input('Please enter the ID number of the CD you would like to delete or "0" to Cancel: ')
                try: deleteMe = int(deleteMe.strip()) #testing to see if deleteMe is valid entry that can be cast to int
```

```
                except:
                    print('~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~')
                    print('That is not a valid option. Returning to main menu.')
                    print('~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n')
                    continue
                if deleteMe == 0:
                    print('\n~~~~~~~~~~~~~~~~~')
                    print('Process Cancelled')
                    print('~~~~~~~~~~~~~~~~~\n')
                    continue
                for row in lstTbl:
                    skipRow += 1
                    if deleteMe in row.values():
                        deleteRow = skipRow #when match found, row index number transferred to deleteRow variable
                        del lstTbl[deleteRow]
                        print('\n~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~')
                        print('CD #', deleteMe, ' deleted. Go to "I - Display Current Inventory" to view updated inventory!')
                        print('~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n')
                        break
                if deleteRow == -1: print('\nCD #', deleteMe, ' not found. Returning to Main Menu.\n') #deleteRow will only be "-1" if no match found
                deleteRow = -1 #reseting counters
                skipRow = -1 #reseting counters
```

Listing 6 - Code for deleting a CD from the current inventory

Saving

As with some of the earlier sections, the "Saving" section should have been very easy to complete as the code was already written. I did not need to make any changes to make it work with dictionaries. However, I did end up making some revisions. After working through the complexities of deleting entries, I decided to offer the user the option of overwriting the existing data in the text file with the current inventory as a simple way of essentially "updating" the file to reflect any deletions they may have made. However, I still wanted to keep the basic option of simply appending any new data to the existing file. As such, I added a basic menu to the beginning of the section to provide both of these options to the user. Since I was already going through the work of creating a menu, I also gave the user the option to cancel and return to the main menu.

In the menu, each option is presented to the user and then assigned with 1, 2, or 3. I left the inputs as strings rather than casting them to integers so that I would not get an error if the user input any characters that could not be converted to an integer. If the user entered anything other than 1, 2, or 3, the user is sent to the `'else'` branch that informs them that they entered an invalid option and returns them to the main menu. (As another option, I could have used the `'try/except'` blocks to test if the input could be converted to integers as I did in the "Deleting" section). The structure of my `'if/elif/else'` menu structure is shown in the pseudocode in Figure 26.

- `If input = '3' (cancel), 'continue' to return user to main menu`
- `Elif input = '1' (overwrite), overwrite file with 'w' access using lstTbl, clear newDataOnlyTbl`
- `Elif input = '2' (append), append file with 'a' access using newDataOnlyTbl, clear newDataOnlyTbl`
- `Else input = anything else, print "invalid" and return user to main menu`

Figure 26 - Pseudocode for structure within "Saving" section

Once I had the structure created, the remaining work was pretty straightforward as the code to "save" came with the starter file. My code only required few small revisions to allow for these changes in functionality between sections. For the overwriting section, I had to change the `'a'` to `'w'` to allow 'write' access instead of 'append' access to the text file. Write access will overwrite the existing contents of the text file with the current contents of `lstTbl`, which includes all CDs imported and added during the current run of the program. For the appending section, I changed the table to `newDataOnlyTbl`. This table is designed to only have CDs that were added during the current run but not yet saved out to the text file, as explained in the pseudocode of Figure 8. I didn't want the user loading all the data from the file and then appending all of it to the text file during the saving process or saving newly added CDs to the file multiple times. In both the overwriting and append sections, I cleared the contents of the `newDataOnlyTbl` table after the saving process since the CDs added during the current run have now been saved to the text file in both sections. Finally, the "Cancel" option takes the user back to the main menu with a `'Continue'` statement. The code for this can be seen in listing 7.

```
175.           #----------SAVE----------#
176.
177.           elif strChoice == 's':  # Providing user option to save new info or overwite into in CDInv
    entory.txt or cancel
178.               print("""
179.       Would you like to:
180.       (1) overwrite your saved inventory with your current inventory,
181.       (2) add your new additions to your saved inventory without
182.            overwriting the existing contents of the file, or
183.       (3) return to menu?
184.
185.       (Note: if you have deleted CDs from the current inventory, these changes will not
186.              be carried over to your saved file if you choose option 2.)
187.                       """)
188.               saveType = input('Please enter your selection (1, 2, 3): ')
189.               if saveType == '3':  #return to menu
190.                   print()
191.                   continue
192.               elif saveType == '1':
193.                   objFile = open(strFileName, 'w')
194.                   for row in lstTbl:
195.                       strRow = ''
196.                       for item in row.values():
197.                           strRow += str(item) + ','
198.                       strRow = strRow[:-1] + '\n'
199.                       objFile.write(strRow)
200.                   objFile.close()
201.                   newDataOnlyTbl = [] #prevents adding albums more than once
202.                   print('\n~~~~~~~~~~~~~~~~~~~~~~~~~~')
203.                   print('Your data has been saved.')
204.                   print('~~~~~~~~~~~~~~~~~~~~~~~~~~\n')
205.               elif saveType == '2':  #option to append
206.                   objFile = open(strFileName, 'a')
207.                   for row in newDataOnlyTbl:
208.                       strRow = ''
209.                       for item in row.values():
210.                           strRow += str(item) + ','
211.                       strRow = strRow[:-1] + '\n'
212.                       objFile.write(strRow)
213.                   objFile.close()
214.                   newDataOnlyTbl = [] #prevents adding albums more than once
215.                   print('\n~~~~~~~~~~~~~~~~~~~~~~~~~~')
216.                   print('Your data has been saved.')
```

```
217.                 print('~~~~~~~~~~~~~~~~~~~~~~~~~~~\n')
218.             else:
219.                 print('\nThat is not a valid option. Returning to Main Menu.\n')
```

Listing 7 - Code for "Save" section

As a final note, I realized that my "Append" section would not reflect any deletions the user made since the "Append" section pulls from newDataOnlyTbl, which is not touched in the "Delete" section. Since I've already put plenty of hours into this project, I decided to resolve this by simply advising the user in the menu that any deletions they made would not be reflected if they chose to "Append" instead of "Overwrite." This is shown in Figure 27.

```
Would you like to:
(1) overwrite your saved inventory with your current inventory,
(2) add your new additions to your saved inventory without
     overwriting the existing contents of the file, or
(3) return to menu?

(Note: if you have deleted CDs from the current inventory, these changes will not
       be carried over to your saved file if you choose option 2.)
```

Figure 27 - Menu within "Saving" section of program with note about append section

Formatting the Program Title

As my final step, I consulted the internet to figure out how to center text without using tabs[3]. Turns out there's a `center()` method available that will print out a string of the desired length with your original string centered in the middle. This is done by adding the necessary amount of padding on either side (the default character for this padding is a space). This can be seen in Listing 8.

```
16. # --- PRESENTATION (INPUT/OUTPUT) WITH SOME PROCESSING MIXED IN--- #
17.
18. print('~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n')
19. print('The Magic CD Inventory'.center(62))
20. print('\n~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~')
```

Listing 8 - Printing program header with center() method

Demonstrating Functionality in Spyder

The code was run successfully using Spyder as shown in Figures 28-33.

[3] https://www.w3schools.com/python/ref_string_center.asp, accessed 2021-Feb-13.

```
In [242]: runfile('C:/_FDPrograming/Mod_05/CDInventory.py', wdir='C:/_FDPrograming/Mod_05')
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

                    The Magic CD Inventory

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

    *****MAIN MENU*****
[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

~~~~~~~~~~
Add a CD
~~~~~~~~~~


Enter the CD's Title: Barrage of Noise

Enter the Artist's Name: Dana Burke

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Data has been added. Go to "I - Display Current Inventory" to view!
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

    *****MAIN MENU*****
[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

~~~~~~~~~~~~~~~~~~~~
Current Inventory
~~~~~~~~~~~~~~~~~~~~

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

ID   CD Title                Artist

14   Barrage of Noise        Dana Burke
```

Figure 28 - Demonstrating functionality in Spyder

```
14   Barrage of Noise         Dana Burke
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

    *****MAIN MENU*****
[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

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Data has been added. Go to "I - Display Current Inventory" to view!
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

    *****MAIN MENU*****
[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

~~~~~~~~~~~~~~~~~~~~
Current Inventory
~~~~~~~~~~~~~~~~~~~~

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

ID   CD Title                 Artist

1    Bursting Bubbles         Katie Peters
2    Courage of Fools         Myrtle Warner
3    Knowledge Bomb           Brian Thomas
4    Blank Canvas             Jonthan Santiago
5    Emotional Wreckage       Gertrude Curry
6    Battleground             Brandi Watkins
7    The Bigger Fish          Jonathan Santiago
8    New Dimension            Kathy Christensen
9    No Ambition              Flora Fox
10   Rain Check               Adam Walker
11   Blissful Ignorance       Dana Burke
12   Zero Gravity             Adam Walker
13   Crossing a Bridge        Flora Fox
14   Barrage of Noise         Dana Burke
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

IPython console History

Figure 29 - Demonstrating functionality in Spyder

Figure 30 - Demonstrating functionality in Spyder

```
~~~~~~~~~~~~~~~~~~~~
Current Inventory
~~~~~~~~~~~~~~~~~~~~

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

ID   CD Title                 Artist

1    Bursting Bubbles         Katie Peters
2    Courage of Fools         Myrtle Warner
3    Knowledge Bomb           Brian Thomas
4    Blank Canvas             Jonthan Santiago
5    Emotional Wreckage       Gertrude Curry
7    The Bigger Fish          Jonathan Santiago
8    New Dimension            Kathy Christensen
9    No Ambition              Flora Fox
10   Rain Check               Adam Walker
11   Blissful Ignorance       Dana Burke
12   Zero Gravity             Adam Walker
13   Crossing a Bridge        Flora Fox
14   Barrage of Noise         Dana Burke
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

    *****MAIN MENU*****
[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


Would you like to:
(1) overwrite your saved inventory with your current inventory,
(2) add your new additions to your saved inventory without
     overwriting the existing contents of the file, or
(3) return to menu?

(Note: if you have deleted CDs from the current inventory, these changes will not
       be carried over to your saved file if you choose option 2.)


Please enter your selection (1, 2, 3): 2

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Your data has been saved.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

Figure 31 - Demonstrating functionality in Spyder

```
    *****MAIN MENU*****
[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


~~~~~~~~~~
Goodbye.
~~~~~~~~~~

In [243]:
```

Figure 32 - Demonstrating functionality in Spyder

CDInventory.txt - Notepad

File Edit Format View Help

```
1,Bursting Bubbles,Katie Peters
2,Courage of Fools,Myrtle Warner
3,Knowledge Bomb,Brian Thomas
4,Blank Canvas,Jonthan Santiago
5,Emotional Wreckage,Gertrude Curry
6,Battleground,Brandi Watkins
7,The Bigger Fish,Jonathan Santiago
8,New Dimension,Kathy Christensen
9,No Ambition,Flora Fox
10,Rain Check,Adam Walker
11,Blissful Ignorance,Dana Burke
12,Zero Gravity,Adam Walker
13,Crossing a Bridge,Flora Fox
14,Barrage of Noise,Dana Burke
```

Figure 33 - Resulting text file (added CD# 14 during this run)

Demonstrating Functionality in Command Window

The code was run successfully using the Anaconda Prompt as shown in Figures 35 - 39. In this case, the text file CDInventory.txt was removed from the folder so that the error functioning could be demonstrated (see Figure 34)

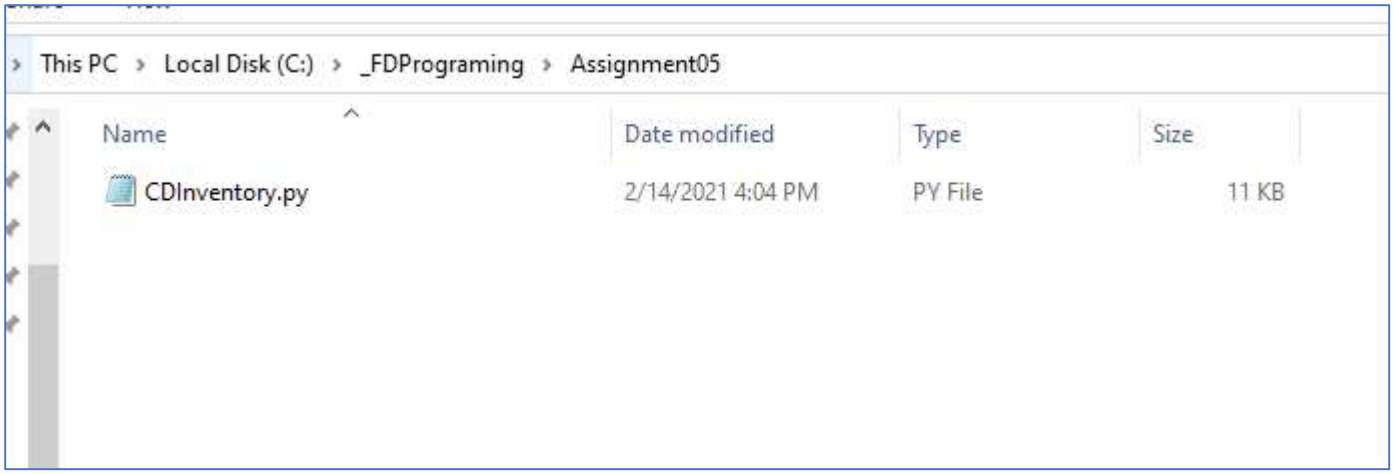

Figure 34 - Testing functionality of script if CDInventory.txt file not present.

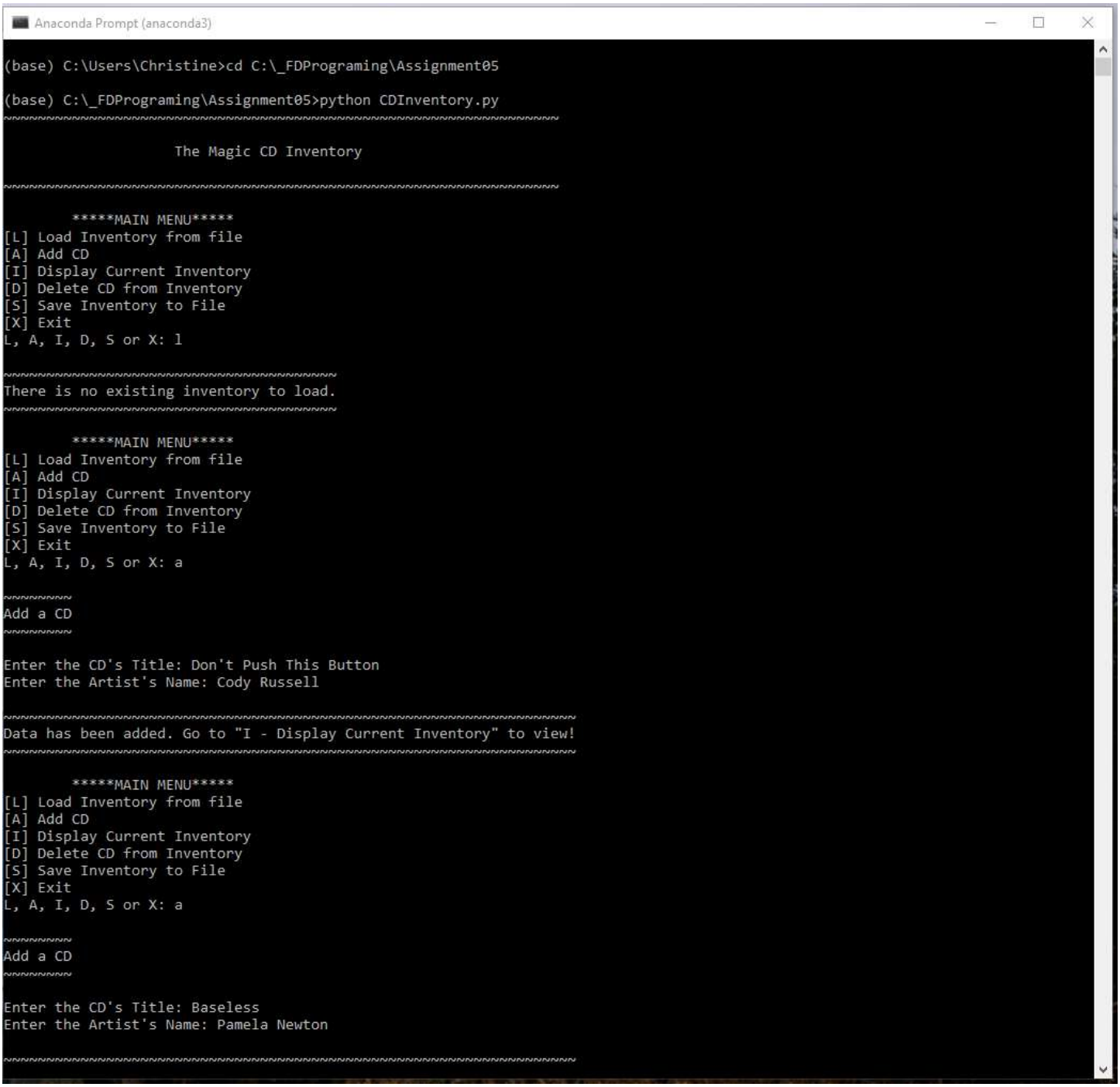

Figure 35 - Demonstrating functionality in Anaconda Prompt

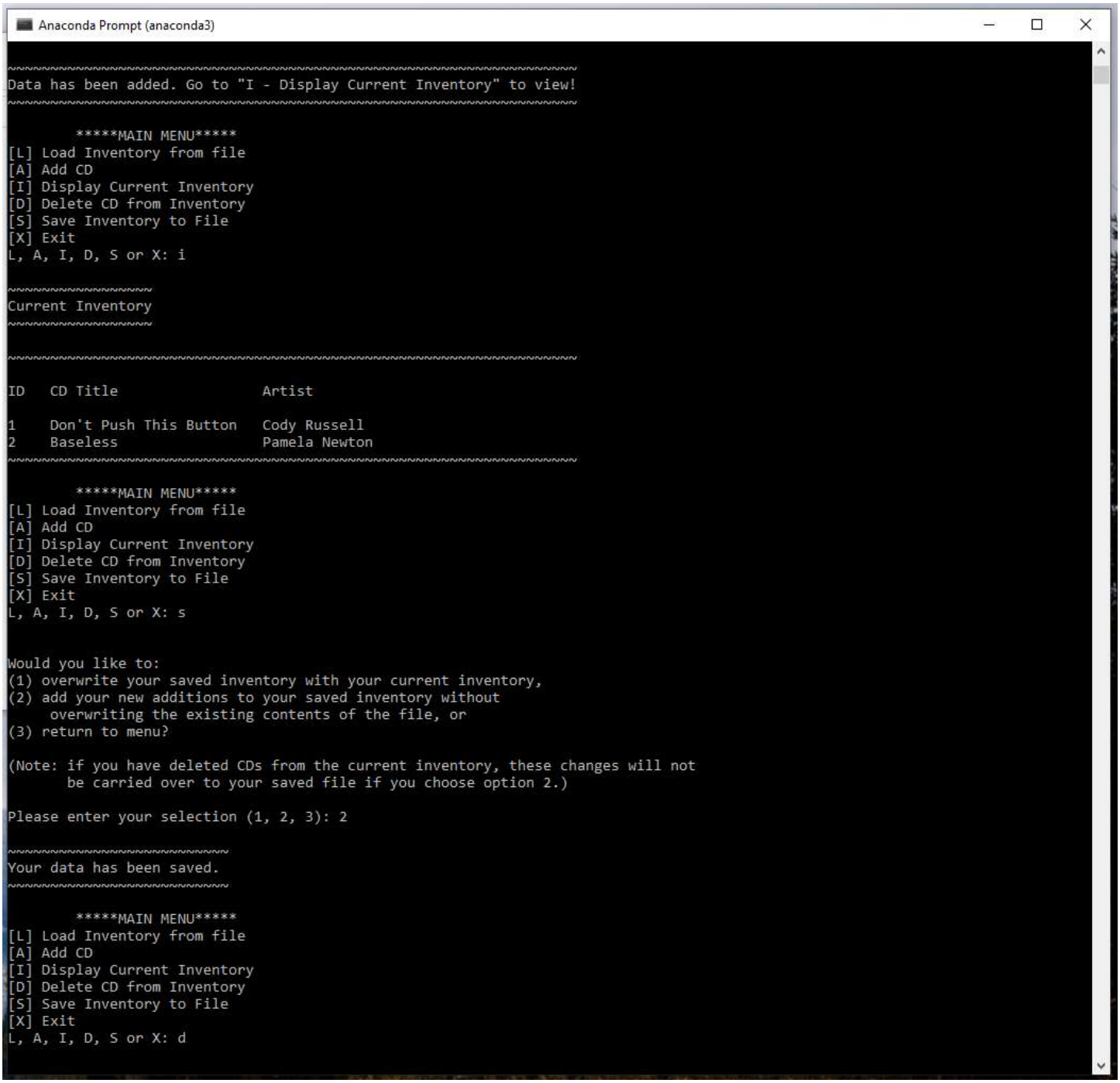

Figure 36 - Demonstrating functionality in Anaconda Prompt

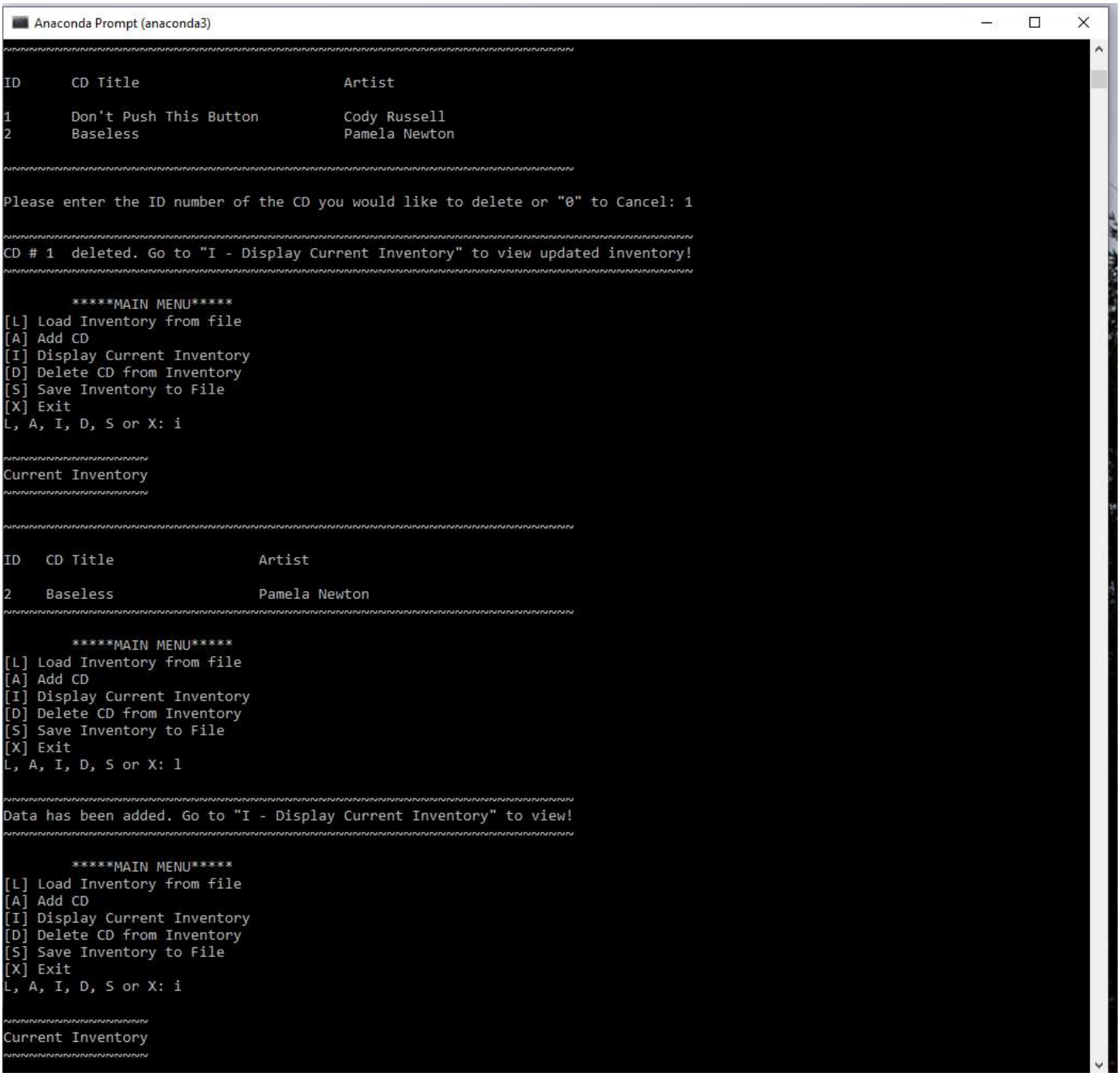

Figure 37 - Demonstrating functionality in Anaconda Prompt

```
~~~~~~~~~~~~~~~~~~~~
Current Inventory
~~~~~~~~~~~~~~~~~~~~

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

ID   CD Title                 Artist

1    Don't Push This Button   Cody Russell
2    Baseless                 Pamela Newton
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

        *****MAIN MENU*****
[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


~~~~~~~~~~
Goodbye.
~~~~~~~~~~

(base) C:\_FDPrograming\Assignment05>
```

Figure 38 - Demonstrating functionlity in Anaconda Prompt

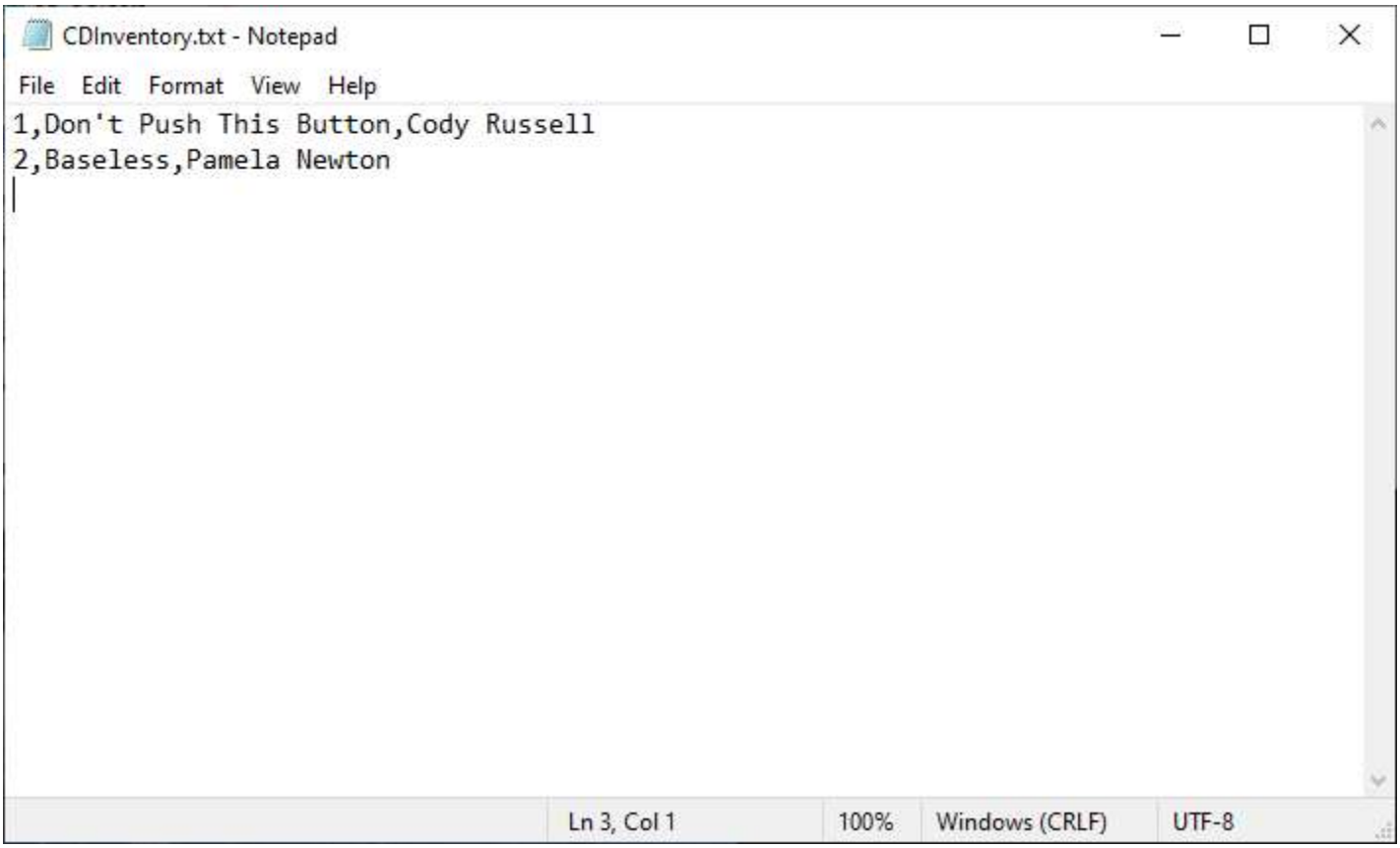

Figure 39 - Resulting text file (added CD# 1 & 2 during this run)

GitHub Link

Upon completing my project, I created an account on GitHub and created the Assignment_05 repository as requested. The link is: https://github.com/cmb225/Assignment_05

Summary

Overall, this assignment was a big breakthrough for me. I found that I was able to solve problems independently by slowly thinking through the logical steps necessary to solve a problem using the building blocks I have in my toolbox. In many cases, it involved starting with a broad idea of what needed to happen, breaking it up into little steps, and then translating each of those steps into code with lots of tests and refinements along the way. Although my solutions are not always the most efficient in the beginning, I'm still pleased that I am able to solve them. I still appreciated the guidance in Office Hours though, as it helped me make my code more efficient. (I have a ways to go before I am both effective AND efficient in my code!). I am also discovering how to test my code more thoroughly to catch as many potential errors as I can before submitting my final code.

Appendix

Listing CDInventory.py

```
1.  # ------------------------------------------------------------------------
2.  # Title: CDInventory.py
3.  # Desc: Script for Assignment 05
4.  # Change Log: (Who, When, What)
5.  # DBiesinger, 2030-Jan-01, Created File
```

```
# CBuffalow, 2021-Feb-12, Added dicRow variable, replaced menu letters with
#    capital letters to avoid confusion of 1 vs l, created consistent capitalization
#    of menu items, added code to section #2, added cd_id generator, added os.path check
# CBuffalow, 2021-Feb-13, added code to all sections, revising comments for clarity
# CBuffalow, 2021-Feb-13, revised Delete section
# CBuffalow, 2021-Feb-14, updated cd_id geneator to find highest ID in data file
#    rather than strictly number of rows
# ======================================================================


# --- DATA --- #
#Loading modules


import os.path

# Declaring variables
strChoice = '' # user input for menu
lstRow = [] # list to hold imported data
lstTbl = []  # list of dictionaries to hold data
newDataOnlyTbl = [] #list of dictionaries to hold data not saved to file
dicRow = {}  # dictionary row of data
strFileName = 'CDInventory.txt'  # data storage file
objFile = None  # file object
objFile_content = '' #empty string for holding file contents
objFile_rows = [] #empty list to hold file contents split into separate strings
indexFinalRow = 0 #index of final row in data file
cd_id = 0 # creating counter for cd_id generator
load = 0 # creating flag to prevent importing file more than once
deleteRow = -1 #creating delete counter for deleting CDs
skipRow = -1 #creating delete counter for deleting CDs



# --- PROCESSING --- #

#counting lines of data in CDInventory.txt so I know where to start next ID number
if os.path.exists(strFileName):
    objFile = open(strFileName, 'r')
    objFile_content = objFile.read() #returns all content of file as a giant string
    objFile.close()
    objFile_rows = objFile_content.split(sep= '\n') #separates data into separate items based on loca
tion of \n
    indexFinalRow = len(objFile_rows) -2
    cd_id = int(objFile_rows[indexFinalRow].split(sep= ',')[0])
else:
    cd_id = 0




# --- PRESENTATION (INPUT/OUTPUT) WITH SOME PROCESSING MIXED IN--- #

print('~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n')
print('The Magic CD Inventory'.center(62))
print('\n~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n')


while True:  #Display menu of options for user
    print('\t*****MAIN MENU*****')
    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().strip()  # convert choice to lower case at time
 of input & strips whitespace
    print()
```

```

    #----------EXIT----------#

    if strChoice == 'x': # Exiting the program
        print('\n~~~~~~~~')
        print('Goodbye.')
        print('~~~~~~~~')
        break

    #----------LOAD----------#

    if strChoice == 'l': # Loading data
        if load == 1:
            print('\nYou have already loaded the inventory from file. Returning to Main Menu.\n')
            continue
        if os.path.exists(strFileName):

            lstTbl = [] #clearing current inventory as all unsaved cds are saved to newDataOnlyTbl
            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)
            objFile.close()
            lstTbl = lstTbl + newDataOnlyTbl #appending new cds to the end of list
            load = 1 #switching flag to 1 so that load process can not be repeated
            print('~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~')
            print('Data has been added. Go to "I - Display Current Inventory" to view!')
            print('~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n')
        else:
            print('~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~')
                   print('There is no existing inventory to load.')
                   print('~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n')


           #----------ADD----------#

           elif strChoice == 'a':  # Allowing user to add data to the table as dictionary
               print('~~~~~~~~')
               print('Add a CD')
               print('~~~~~~~~\n')
               dicRow = {} #preventing overwriting as I am using same keys for every dictionary row
               cd_id += 1
               dicRow['id'] = cd_id
               dicRow['title'] = input('Enter the CD\'s Title: ')
               dicRow['artist'] = input('Enter the Artist\'s Name: ')
               lstTbl.append(dicRow) #adding data to current inventory view
               newDataOnlyTbl.append(dicRow) #adding all unsaved data to a separate table
               print('\n~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~')
               print('Data has been added. Go to "I - Display Current Inventory" to view!')
               print('~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n')

           #----------DISPLAY----------#

           elif strChoice == 'i':  # Displaying the current data to the user
               print('~~~~~~~~~~~~~~~~~')
               print('Current Inventory')
               print('~~~~~~~~~~~~~~~~~\n')
               print('~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n')
               print('{:<5}{:<25}{:<25}'.format('ID', 'CD Title', 'Artist'))
               print()
               for row in lstTbl:
                   print('{:<5}{:<25}{:<25}'.format(*row.values()))
               print('~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n')

```

```
        #----------DELETE----------#

        elif strChoice == 'd': # Providing user the option to delete an entry or cancel
            if len(lstTbl) == 0:
                print('~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~')
                print('Your current inventory is empty. There are no entries to delete.')
                print('~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n')
                continue
            print('~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n')
            print('{:<4}\t{:<25}\t{:<25}'.format('ID', 'CD Title', 'Artist'))
            print()
            for row in lstTbl:
                print('{:<4}\t{:<25}\t{:<25}'.format(*row.values()))
            print('\n~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n')
            deleteMe = input('Please enter the ID number of the CD you would like to delete or "0" to Cancel: ')
            try: deleteMe = int(deleteMe.strip()) #testing to see if deleteMe is valid entry that can be cast to int
            except:
                print('~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~')
                print('That is not a valid option. Returning to main menu.')
                print('~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n')
                continue
            if deleteMe == 0:
                print('\n~~~~~~~~~~~~~~~~~~')
                print('Process Cancelled')
                print('~~~~~~~~~~~~~~~~~\n')
                continue
            for row in lstTbl:
                skipRow += 1
                if deleteMe in row.values():
                    deleteRow = skipRow #when match found, row index number transferred to deleteRow variable
                    del lstTbl[deleteRow]
                    print('\n~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~')
                    print('CD #', deleteMe, ' deleted. Go to "I - Display Current Inventory" to view updated inventory!')
                    print('~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n')
                    break
            if deleteRow == -1: print('\nCD #', deleteMe, ' not found. Returning to Main Menu.\n') #deleteRow will only be "-1" if no match found
            deleteRow = -1 #reseting counters
            skipRow = -1 #reseting counters



        #----------SAVE----------#

        elif strChoice == 's':  # Providing user option to save new info or overwite into in CDInventory.txt or cancel
            print("""
    Would you like to:
    (1) overwrite your saved inventory with your current inventory,
    (2) add your new additions to your saved inventory without
         overwriting the existing contents of the file, or
    (3) return to menu?

    (Note: if you have deleted CDs from the current inventory, these changes will not
           be carried over to your saved file if you choose option 2.)
                    """)
            saveType = input('Please enter your selection (1, 2, 3): ')
            if saveType == '3':  #return to menu
```

```
                    print()
                    continue
                elif saveType == '1':
                    objFile = open(strFileName, 'w')
                    for row in lstTbl:
                        strRow = ''
                        for item in row.values():
                            strRow += str(item) + ','
                        strRow = strRow[:-1] + '\n'
                        objFile.write(strRow)
                    objFile.close()
                    newDataOnlyTbl = [] #prevents adding albums more than once
                    print('\n~~~~~~~~~~~~~~~~~~~~~~~~~~~')
                    print('Your data has been saved.')
                    print('~~~~~~~~~~~~~~~~~~~~~~~~~~~\n')
                elif saveType == '2':  #option to append
                    objFile = open(strFileName, 'a')
                    for row in newDataOnlyTbl:
                        strRow = ''
                        for item in row.values():
                            strRow += str(item) + ','
                        strRow = strRow[:-1] + '\n'
                        objFile.write(strRow)
                    objFile.close()
                    newDataOnlyTbl = [] #prevents adding albums more than once
                    print('\n~~~~~~~~~~~~~~~~~~~~~~~~~~~')
                    print('Your data has been saved.')
                    print('~~~~~~~~~~~~~~~~~~~~~~~~~~~\n')
                else:
                    print('\nThat is not a valid option. Returning to Main Menu.\n')

            #----------DEALING W/INCOMPATIBLE INPUT----------#

            else: #if user enters invalid menu option
                print('Please choose either L, A, I, D, S or X!\n')
```