Christine Buffalow
2021-Feb-26
Foundations of Programming
Assignment 7

Creating a CD Inventory, version 4.0

Introduction

For this assignment, I took the CD Inventory from our previous assignment (version 3.0) and added (1) structured error handling around any "risky" or "error-prone" areas of my program and (2) modified the permanent data storage to use binary data instead of a text file. After completing the labs in this module, converting the data storage to binary and adding structured data handling, was very straightforward, but it was a little tricky making sure I had everything covered sufficiently.

Adding Structured Data Handling

In the assignment, we were specifically advised to "add structured error handling around the areas where there is user interaction, type casting (string to int) or file access operations."[1] To begin, I did some research about exception handling in Python. As always, w3schools is one of my favorite resources since they do such a great job at demonstrating all the basic components of a concept with plenty of simple examples[2]. I also liked Real Python's article on exception handling[3]. In addition to explanation and coding examples, they also have some very simple and clear graphical diagrams that explained how the various blocks are used, which helped me understand the 'try/except/else/finally' block structure. While reading this article, I ran across a link to an article warning about the general use of nonspecific try/except blocks[4]. Although it's perhaps a bit too high level for me at this point, I am glad that I am now aware of the downside of using general try/except blocks. They can be useful for preventing the program from crashing, but they can be dangerous if you rely on them to "make your program work" without doing the work to figure out what could cause errors (or IS causing errors) in your program. I like their idea of adding an error log to this structure so that errors won't crash the program, but you still have information about what is causing issues in the code (although it's perhaps too complicated to implement at this time). My takeaway from this as a beginner (who doesn't know how to add error logging to my programs yet) is to be extra mindful of general try/except blocks by making sure they at least have print statements that display the error type. This way errors won't be hidden and the user will be aware of what's happening as well.

FileProcessor.read_file

Within the `FileProcessor.read_file` function, I used the `'try/except'` block around the code to make sure the program won't crash if the specified file doesn't exist in the current directory. If the file does exist, the function loads the data from the file into the variable `table`. If it doesn't exist, the `'except FileNotFoundError'` block will simply print a message informing the user that the file was not found and it will assign an empty list to the variable `table`. If there is another type of error, it is caught by the general `'except Exception'` block. In this case, the `print` message informs the user what kind of error occurs by printing the type of exception and it will assign an empty list to the variable `table`. This way, the function can return a list called `table` whether or not there is an error. When I didn't do this, the function returned `table` with the value of `'None'` when the file didn't exist. This caused issues in other parts of my code that were expecting an iterable list from this function. The code can be seen in Listing 1.

[1] Biesinger, Dirk. "Assignment 07." Foundations of Programming (Python).
[2] https://www.w3schools.com/python/python_try_except.asp, accessed 2021-Feb-27.
[3] https://realpython.com/python-exceptions/, accessed 2021-Feb-27.
[4] https://realpython.com/the-most-diabolical-python-antipattern/, accessed 2021-Feb-27.

```
128.             @staticmethod
129.             def read_file(file_name):
130.                 """Function to read data from a .dat file using pickling.
131.
132.                 Function checks to make sure specified txt file exists.  If yes, it
133.                 continues by reading the data from .dat file and saving it to 'table'
134.                 using pickling. If no, an empty list is assigned to 'Table'.
135.
136.                 Args:
137.                     file_name (string): name of file used to read the data from
138.
139.                 Returns:
140.                     table (list of dict): 2D data structure (list of dicts)
141.                 """
142.                 try:
143.                     with open(file_name, 'rb') as objFile:
144.                         table = pickle.load(objFile)
145.                 except FileNotFoundError:
146.                     print('File not found. Inventory is still empty.')
147.                     table = []
148.                 except Exception as e:
149.                     print('There has been a ', type(e), ' error with the read process. Inventory is st
    ill empty.')
150.                     table = []
151.                 return table
```

Listing 1 - FileProcessor.read_file

FileProcessor.write_file

Within the `FileProcessor.write_file` function, I used the `'try/except'` blocks to handle any errors. Unlike the `read` function, the program will not generate an error if the specified file does not exist in the current directory. Instead, the program will simply create it. As such, I was not aware of anything that could cause an error, but I added the general `'try/except'` blocks around the code just in case. If an error is generated, the user is simply informed what kind of error occurred and that the file was not saved. The code can be seen in Listing 2.

```
155.             @staticmethod
156.             def write_file(file_name, table):
157.                 """Function to write data from current runtime to a .dat file using pickling.
158.
159.                 The data from 'table' is saved to the designated .dat file using
160.                 pickling.
161.
162.                 Args:
163.                     file_name (string): name of file used to write data to
164.                     table (list of dicts): 2D data structure (list of dicts) that holds data during ru
    ntime
165.
166.                 Returns:
167.                     None.
168.                 """
169.                 try:
170.                     with open(file_name, 'wb') as objFile:
171.                         pickle.dump(table, objFile)
172.                 except Exception as e: #if there is a general error with the save process
173.                     print('There has been a ', type(e), ' error with the save process. File not saved.
    ')
```

Listing 2 - FileProcessor.write_file

FileProcessor.calculate_cd_id

For this version, I revised my `calculate_cd_id` function to calculate the value based on the current inventory table rather than from the permanent data storage. I realized this would be possible since the data from permanent storage is immediately loaded into the current inventory when the program starts. This simplified my code in this function quite a bit since I no longer had to read from the file. Now I simply had to call out the last dictionary in the table by finding the length of the table and subtracting one (since the index starts at zero) and then find the value of the 'ID' key in that dictionary. The value should already be an integer, but I cast it to an integer just to be safe. I then add the value of one to generate the ID number for the next CD.

However, if the list `table` is empty, this will generate an `IndexError`. As such, I added a `'except IndexError'` block. If an error is generated, the value of 1 will be assigned to `cd_id` since this will be the first entry in the list. I also added a general `'except'` block to catch any weird errors that may occur. Again, I assigned the value of 1 to cd_id. This may lead to issue depending on the type of error, but it will not break my program if a few entries have the same ID number. The code for this can be seen in Listing 3.

```
39.     @staticmethod
40.     def calculate_cd_id(table):
41.         """Function to determine the appropriate ID number for the CD.
42.
43.         The function finds the last row of data in the current inventory and reports
44.         the ID number used in that row.  The function then adds 1 to the ID number
45.         to be used on the next CD.
46.
47.         Args:
48.             table (list of dict): 2D data structure (list of dicts)
49.
50.         Returns:
51.             cd_id (int): ID number for the CD
52.         """
53.
54.         try:
55.             tableIndex = len(table) -1
56.             cd_id = int(table[tableIndex]['ID']) + 1 #should already be an integer, but just in case.
    ..
57.         except IndexError:  #the IndexError will occur any time the table is empty & is not alarming,
     so simply assigning cd_id = 1 (1st entry)
58.             cd_id = 1
59.         except Exception as e: #in case something weird happens - still handling by assigning cd_id =
     1
60.             cd_id = 1
61.             print('There has been a ', type(e), ' error. The next ID number will be 1.')
62.         finally:
63.             return cd_id
```

Listing 3 - FileProcessor.calculate_cd_id

IO.menu_choice

The `IO.menu_choice` function was provided with the starter code in the previous assignment. It essentially has a type of error handling built in as it will only accept values from certain list. If the user enters anything other than a value from this list, they will be stuck in the `while` loop until they cooperate. Any other errors should be caught by the general `'try/except'` blocks I put around the main body of the program. The code can be seen in Appendix Listing CDInventory.py.

IO.get_cd_data

The `get_cd_data` function features multiple "open ended" `input` functions where the user can enter anything they want. As far as I'm aware, anything a user types into the computer can be saved as a string, so I'm not anticipating any

user-generated errors in this section. I also wasn't planning on converting these strings to any other data type, so I wasn't concerned about generating an error in that way either. (The ID number could be problematic area if you ask for that from user input since I would be casting it to an integer, but I wrote a function to generate this number automatically. I suppose is a form of error handling in a way!).

However, to be on the safe side, I added a `'try/except'` block around each `input` statement just in case. If the user manages to create an error, the user is informed that they entered an invalid response that generated an error, and that the value of 'NA' was assigned to the respective variable instead of the user's input. The code can be seen in Listing 4.

```
243.            @staticmethod
244.            def get_cd_data():
245.                """Collects information about CD from user.
246.
247.                Asks user to input the name of the CD and the artist of the CD.
248.
249.                Args:
250.                    None
251.
252.                Returns:
253.                    cd_title (str): name of CD, provided by user
254.                    cd_artist (str): name of artist, provided by user
255.
256.                """
257.
258.                try:
259.                    cd_title = input('What is the CD\'s title? ').strip()
260.                except Exception as e: #if user manages to cause an error with their input
261.                    print('You have entered an response that caused a ', type(e),' error.')
262.                    print('CD Title = NA')
263.                    cd_title = 'NA'
264.                try:
265.                    cd_artist = input('What is the Artist\'s name? ').strip()
266.                except Exception as e: #if user manages to cause an error with their input
267.                    print('You have entered an response that caused a ', type(e),' error.')
268.                    print('CD Artist = NA')
269.                    cd_artist = 'NA'
270.                return(cd_title, cd_artist)
```

Listing 4 - IO.get_cd_data

IO.process_cd_data & IO.process_delete

Although the IO.process_cd_data & IO.process_delete functions do not involve user interaction, type casting, or file access, I threw some generic try/except blocks around the code in case an error sneaks in from another area of the program. If an error is generated, the user is told that an error occurred, and default/dummy values are assigned to all return variables. The code for this can be seen in Appendix CDInventory.py.

Main Script

Within the main script, there are a handful of standard `input` statements to help the user navigate around the program and provide information. As with before, I'm not aware of any ways the user could cause an error, but I added some variety of the general `'try/except/else/finally'` blocks around each one just to be on the safe side. If the user manages to create an error, they will be informed that their response generated an error, and they will be returned to the main menu. Two such example are shown back-to-back in Listing 5. The rest can be seen in Appendix Listing CDInventory.py.

```
293.                    try:
294.                        strYesNo = input('Type \'yes\' to continue and reload from file - otherwise re
    load will be cancelled. ')
```

```
295.                     strYesNo.lower() #testing to see if lower() causes an error
296.                 except Exception as e: #if user manages to cause an error with their input
297.                     print('You have entered an response that caused a ', type(e),' error. Returnin
    g to Main Menu.')
298.                     continue #returns user to Main Menu
299.                 else:
300.                     if strYesNo.lower() == 'yes':
301.                         print('Reloading...')
302.                         lstTbl = FileProcessor.read_file(strFileName)
303.                         print('Inventory Loaded.')
304.                     else:
305.                         print('Cancelling... Inventory data NOT reloaded.')
306.                     IO.show_inventory(lstTbl)
307.                 try: input('Press [ENTER] to return to Main Menu. ')
308.                 except Exception as e: #if user manages to cause an error with their input
309.                     print('You have entered an response that caused a ', type(e),' error. Returnin
    g to Main Menu.')
310.                 finally: continue  # start loop back at top.
```

Listing 5 - Example of 'try/except/else/finally' blocks to protect inputs statement and other code

As mentioned before, I also added a general `'try/except'` block around the entire body of the Main Script to catch anything else that might occur. This can also be seen in Appendix Listing CDInventory.py.

It should be noted that there was one extra "risky area" that definitely needed to be addressed. In Section # 3.5.1.2, the user is asked for the ID number of the CD they want to delete. Since all inputs are strings, this value is then cast to an integer. By simply entering in a non-integer, the user can very easily generate a `ValueError`. To resolve this, I added `'try/except ValueError'` blocks around the `input` statement in addition to a general `'except'` block to catch any other errors the user may create. If the user enters a non-valid response, they are informed that they did not enter an integer and they are returned to the main menu. The code can be seen in Listing 6 (note the code continues with an `'else'` block, see Appendix Listing CDInventory.py for the remainder of the code).

```
336.                 # 3.5.1.2 ask user which ID to remove
337.                   try:
338.                     intIDDel = int(input('Which ID would you like to delete? ').strip())
339.                   except ValueError: #if user enters a non-integer
340.                     print('This is not an integer.')
341.                   except Exception as e: #if user manages to cause an error with their input
342.                     print('You have entered an response that caused a ', type(e),' error.')
```

Listing 6 - Using 'try/except' blocks to prevent ValueError or other error

Using Binary Data:

The second main objective of this assignment was to convert the permanent data storage to binary. Although I felt the explanation from the lectures and textbook were abundantly clear, I did some additional research as requested. I returned to RealPython again, although their explanation involved some complex topics we have not covered including serialization, marshalling, and object hierarchies[5]. The DataCamp tutorial, however, was much more appropriate to my current level of knowledge[6]. It was clearly written to a beginner audience, which I appreciated. In addition to explaining many of the conceptual ideas necessary to understanding pickling and its advantages and its limitations, they also included a lot of simple examples to demonstrate the syntax. One final resource I found was on tutorialspoint[7]. This short but informative tutorial was helpful, although I was confused by their use of txt files with pickling a list.

[5] https://realpython.com/python-pickle-module/, accessed 2021-Feb-27.
[6] https://www.datacamp.com/community/tutorials/pickle-python-tutorial, 2021-Feb-27.
[7] https://www.tutorialspoint.com/python-pickling , accessed 2021-Feb-27.

Armed with all this information, I set out to update my permanent data storage to binary. Thanks to functions, changing to binary data just involved altering a couple functions and a couple function calls.

FileProcessor.read_file

Within the `FileProcessor.read_file` function, I replaced all the code within the `'if'` branch with code that opens the specified data file with 'read binary' access and loads the data from the file into the local variable `table` using the `pickle` module. I also replaced the code within the `'else'` branch with `'table = []'` so that if the data file is empty or not found, a list will still be assigned to `table`. The function argument is the file name (`file_name`) and the function returns the variable `table`. The code can be seen in Listing 1.

FileProcessor.write_file

Within the `FileProcessor.write_file` function, I replaced all the code with code that opens the specified data file with write binary' access and save (or 'dumps') the data into the file from the local variable `table` to the specified data file using the `pickle` module. The function arguments are the file name (`file_name`) and the variable (`table`) that contains the data to be written to the file. The function does not return anything. The code can be seen in Listing 2.

Summary

Overall, the assignment went well. After working with this CD Inventory project for several weeks, I am quite familiar with my program, so it is easy to make adaptations pretty quickly. The most challenging aspect was knowing what could possibly cause errors in my program so that I would know how to handle them. I've certainly stumbled across a few over the past few weeks (which makes it easier since I know what I did and what error it caused), but it's hard to know what other errors might be out there that I haven't stumbled across yet. I tried to research to find additional ones, but I did not find any relevant ones that would cause issues in my program at this time. As such, I left a lot of general error messages in my program but had most of them print out the type of error that was encountered.

Functionality in Sypder

Function performed as expected in Spyder. Some screenshots shown in Figures 1-3.

Figure 1 - Program running in Spyder

```
Which operation would you like to perform? [L, A, I, D, S or X]: d


================== The Current Inventory: ==================
ID        CD Title                Artist
------------------------------------------------------------
1         Bursting Bubbles        Katie Peters
3         Blank Canvas            Jonathan Santiago
4         Battleground            Brandi Watkins
5         New Dimension           Kathy Christensen
6         Curiousity              Janie Stephens
7         Rain Check              Adam Walker
8         Crossing a Bridge       Flora Fox
============================================================

Which ID would you like to delete? 4
CD #4 deleted.


================== The Current Inventory: ==================
ID        CD Title                Artist
------------------------------------------------------------
1         Bursting Bubbles        Katie Peters
3         Blank Canvas            Jonathan Santiago
5         New Dimension           Kathy Christensen
6         Curiousity              Janie Stephens
7         Rain Check              Adam Walker
8         Crossing a Bridge       Flora Fox
============================================================

Press [ENTER] to return to Main Menu.


------------ 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
------------------------------


Which operation would you like to perform? [L, A, I, D, S or X]: i
```

Figure 2 - Program running in Spyder

```
Which operation would you like to perform? [L, A, I, D, S or X]: i


================== The Current Inventory: ==================
ID        CD Title                Artist
------------------------------------------------------------
1         Bursting Bubbles        Katie Peters
3         Blank Canvas            Jonathan Santiago
5         New Dimension           Kathy Christensen
6         Curiousity              Janie Stephens
7         Rain Check              Adam Walker
8         Crossing a Bridge       Flora Fox
============================================================

Press [ENTER] to return to Main Menu.


------------ 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
------------------------------


Which operation would you like to perform? [L, A, I, D, S or X]: x


In [92]:
```

Figure 3 - Program running in Spyder

Functionality in Anaconda Prompt

Function performed as expected in Anaconda Prompt. Some screenshots shown in Figures 4-7.

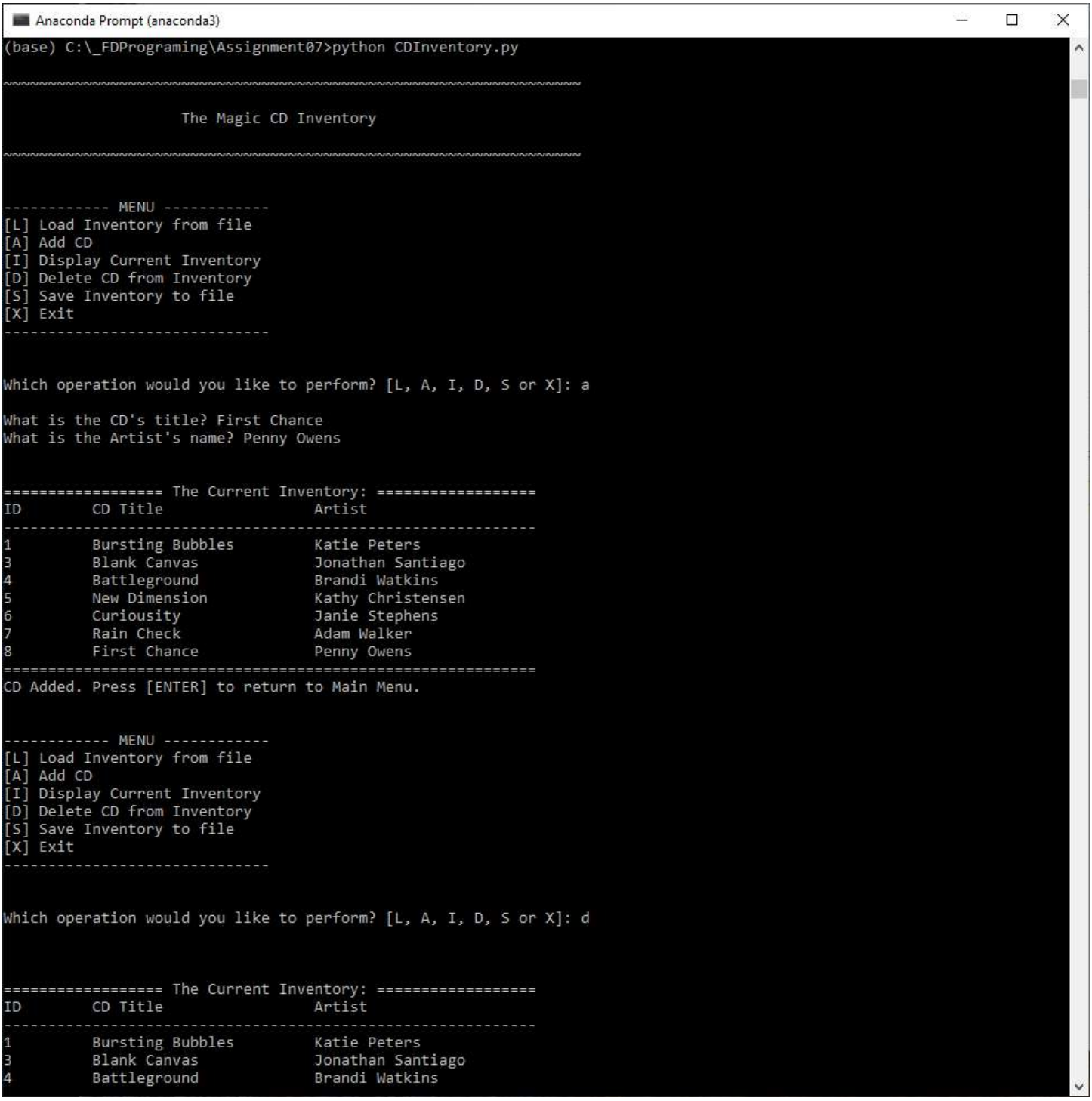

Figure 4 - Program running Anaconda Prompt

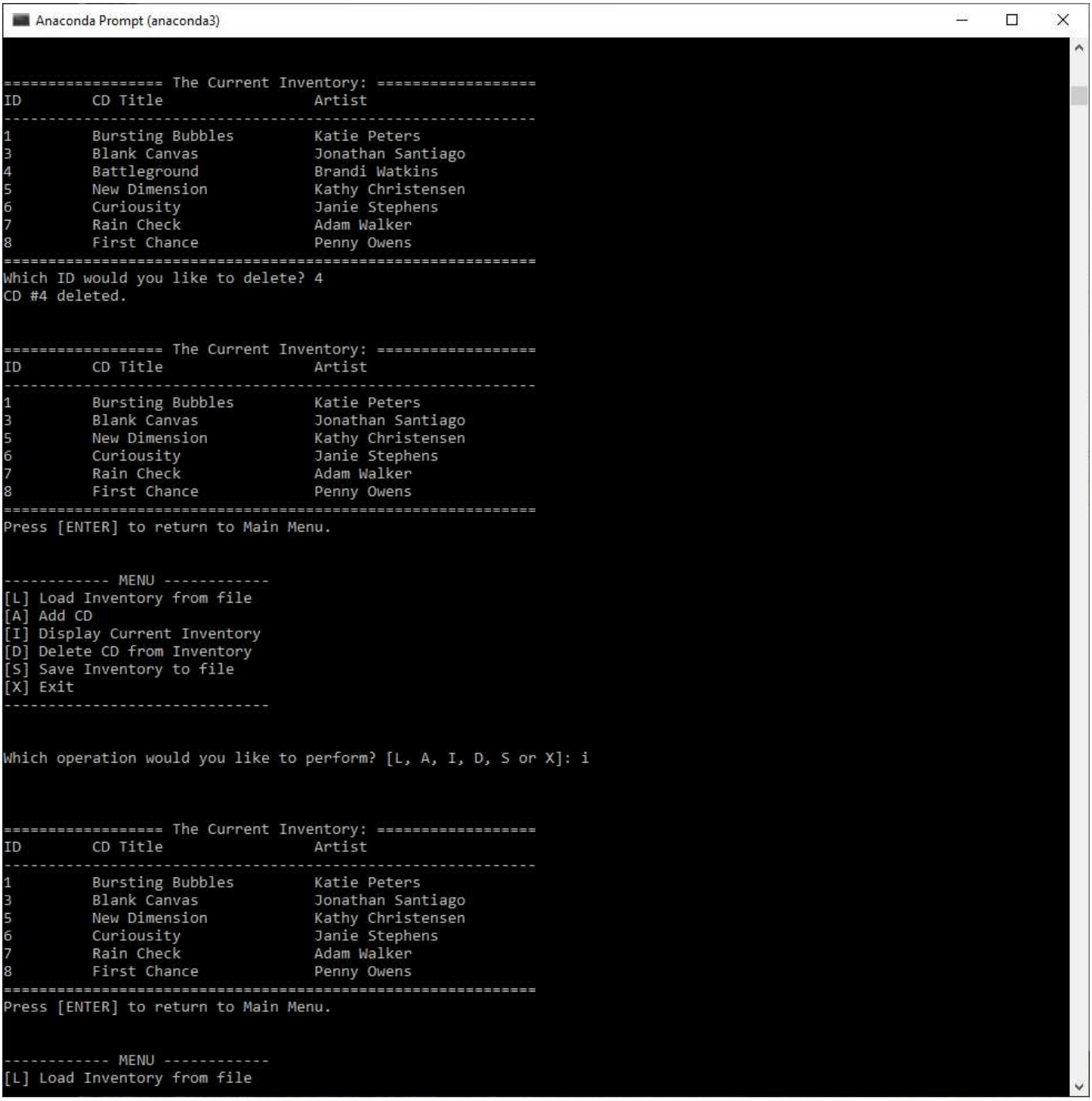

Figure 5 - Program running Anaconda Prompt

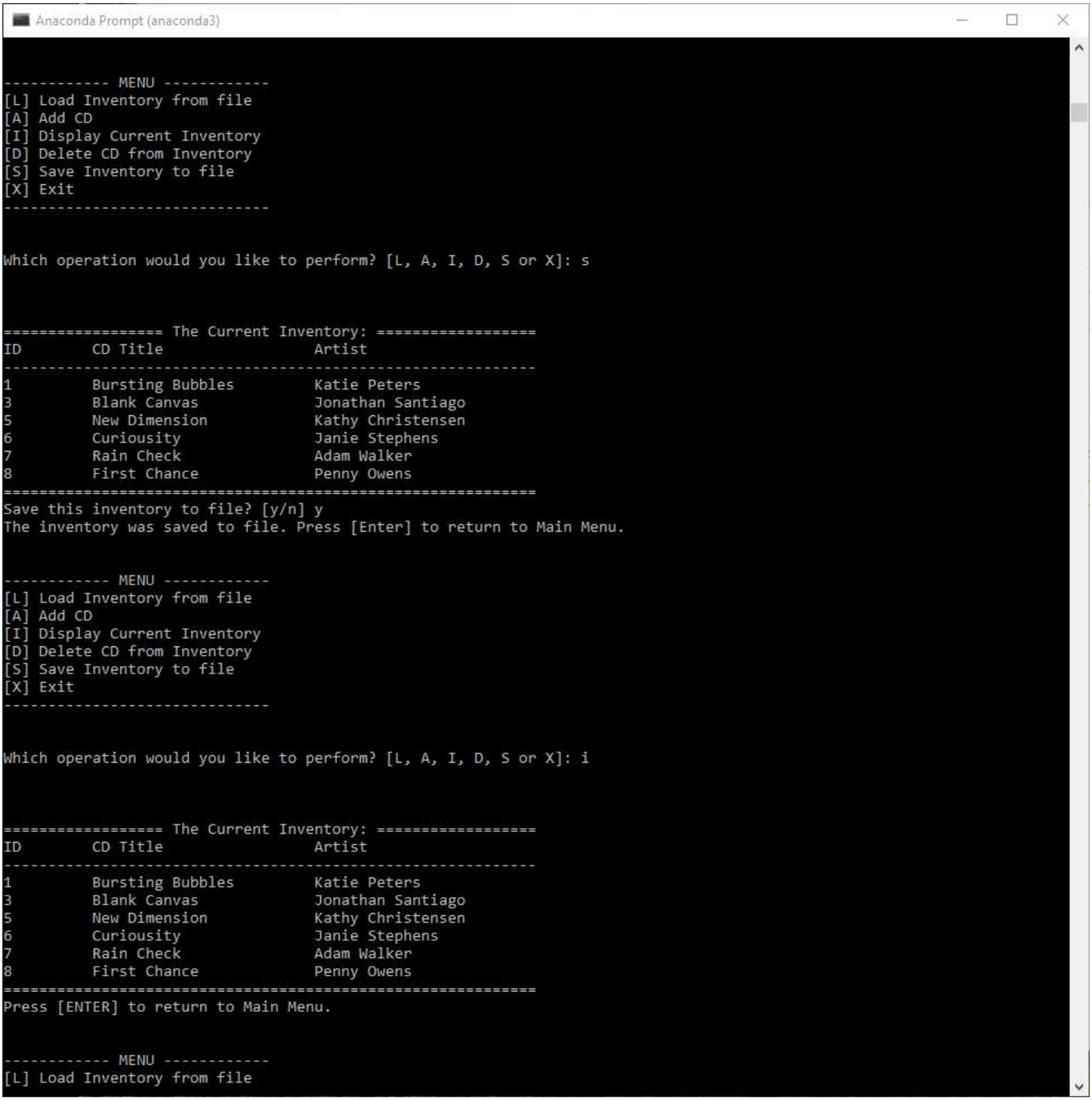

Figure 6 - Program running Anaconda Prompt

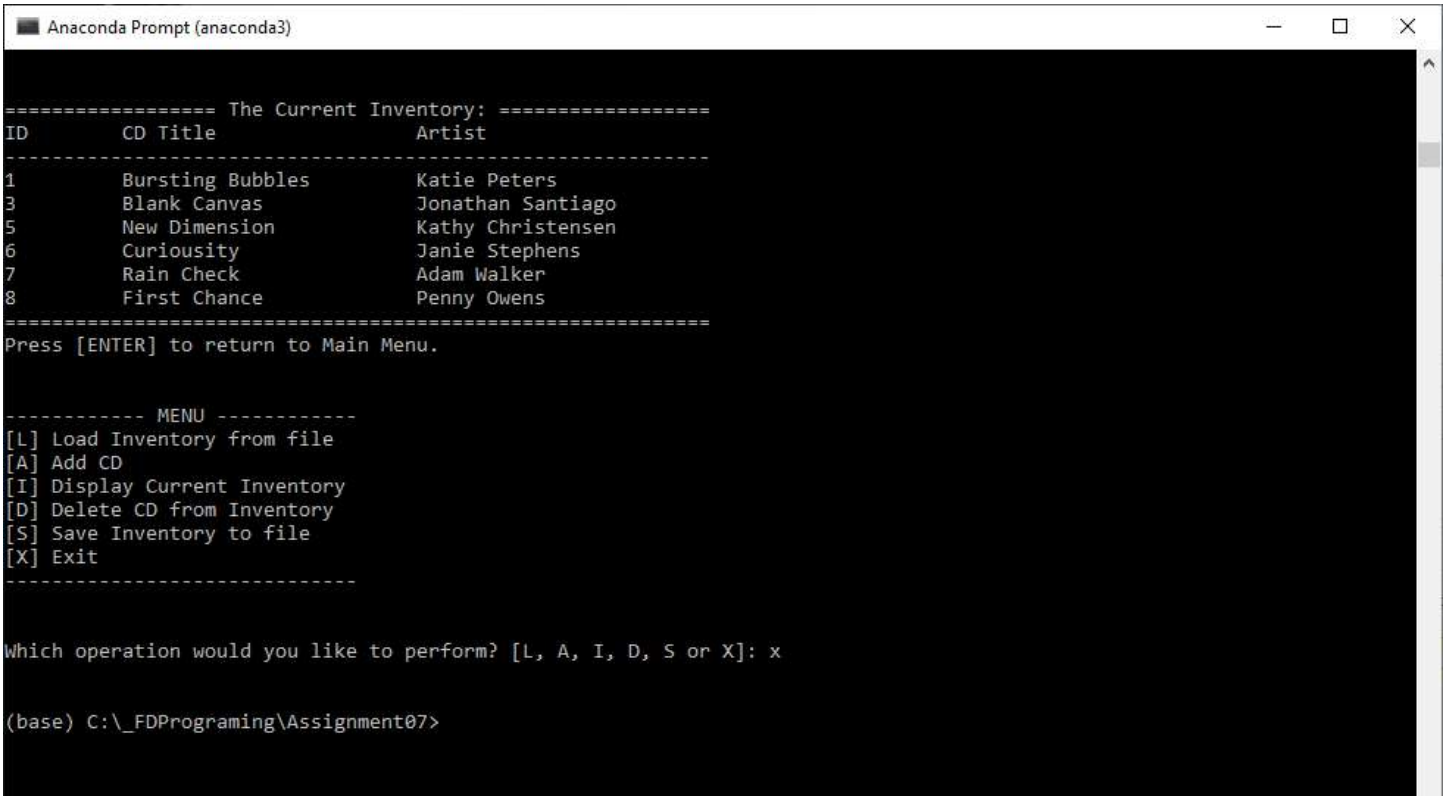

Figure 7 - Program running Anaconda Prompt

Appendix

Listing CDInventory.py

```
#------------------------------------------#
# Title: CDInventory.py
# Desc: Working with classes and functions.
# Change Log: (Who, When, What)
# DBiesinger, 2030-Jan-01, Created File
# CBuffalow, 2021-Feb-19, Added functions (process_cd_data, process_delete, write_file,
#   get_cd_data), added uppercase functionality to menu for better readability,
#   adjusted formatting, added program header
# CBuffalow, 2021-Feb-20, Added calculcate_cd_id function, adjusted
#    process_cd_data & get_cd_data functions, revised comments
# CBuffalow, 2021-Feb-25, Changed permanent data storage from txt to dat.
# CBuffalow, 2021-Feb-26, Revised cd_id program to generate from current inventory
#   instead of permament storage, added error handling around write, read, and input functions
# CBuffalow, 2021-Feb-27, Added additional error handling around program
#------------------------------------------#

# -- MODULES -- #
import pickle

# -- DATA -- #
#Global Variables
strChoice = '' # user input (string)
strYesNo = '' # user input for yes/no question (string)
indID = -1 # ID number of CD (integer)
strTitle = '' # user input (string)
strArtist = '' # user input (string)
lstTbl = []  # table to hold data (list of dictionaries)
dicRow = {}  # row of data (dictionary)
strFileName = 'CDInventory.dat'  # data storage file
```

```
intIDDel = '' # user selected ID to delete (integer)
intRowNr = '' # index of row to be deleted (integer)
blnCDRemoved = "False" # flag if desired CD found (Boolean)


# -- PROCESSING -- #
class DataProcessor:
    """Processing data within program's current runtime."""

    @staticmethod
    def calculate_cd_id(table):
        """Function to determine the appropriate ID number for the CD.

        The function finds the last row of data in the current inventory and reports
        the ID number used in that row.  The function then adds 1 to the ID number
        to be used on the next CD.

        Args:
            table (list of dict): 2D data structure (list of dicts)

        Returns:
            cd_id (int): ID number for the CD
        """

        try:
            tableIndex = len(table) -1
            cd_id = int(table[tableIndex]['ID']) + 1 #should already be an integer, but just in case.
..
        except IndexError:  #the IndexError will occur any time the table is empty & is not alarming,
 so simply assigning cd_id = 1 (1st entry)
            cd_id = 1
        except Exception as e: #in case something weird happens - still handling by assigning cd_id =
 1
            cd_id = 1
            print('There has been a ', type(e), ' error. The next ID number will be 1.')
        finally:
            return cd_id


    @staticmethod
    def process_cd_data(cd_id, cd_title, cd_artist):
        """Function that takes user input about CD and formats data into a dictionary.

        Keys are "ID", 'Title', and 'Artist' for each dictionary and the values are assigned
        from cd_id, cd_title, and cd_artist, respectively.

        Args:
            cd_id (int): ID of CD, generated by program
            cd_title (str): name of CD, provided by user
            cd_artist (str): name of artist, provided by user

        Returns:
            cd_info (dictionary): cd_id, cd_title, and cd_artist contained within a single dictionary

        """
        try:
            cd_info = {'ID': cd_id, 'Title': cd_title, 'Artist': cd_artist}
        except Exception as e: #in case something weird happens
            cd_info = {'ID': 0, 'Title': 'NA', 'Artist': 'NA'}
            print('There has been a ', type(e), ' error. A generic dictionary has been created instea
d.')
        finally:
            return cd_info

```

```
90.
91.
92.     @staticmethod
93.     def process_delete(cd_to_delete, table):
94.         """Function that processes which row index needs to be deleted based upon input from user.
95.
96.         For each dictionary, the function determines if the key:value pair in that row/dictionary
97.         with the key of "ID" has the desired ID number for its value.  If yes, row_number stops count
    ing
98.         at that row and CD_found is set to 'True'.
99.
100.                Args:
101.                    cd_to_delete (int): ID number of CD that user wants to delete
102.                    table (list of dicts): 2D data structure (list of dicts) that holds data during ru
    ntime
103.
104.                Returns:
105.                    row_number (int): counter of rows (stops on row if desired CD found, otherwise ret
    urns index of last row)
106.                    CD_found (Boolean): flag that states if desired CD found (True = yes, False = no)
107.                """
108.                row_number = -1
109.                CD_found = False
110.                try:
111.                    for row in table:
112.                        row_number += 1
113.                        if row['ID'] == cd_to_delete:
114.                            CD_found = True
115.                            break
116.                except Exception as e: #in case something weird happens
117.                    print('There has been a ', type(e), ' error. No CD will be deleted.')
118.                    row_number = -1 #doesn't really matter what this value is when CD_found is False
119.                    CD_found = False #CD_found as false will trigger the if/else in main script to NOT
     delete CD
120.                finally:
121.                    return row_number, CD_found
122.
123.
124.
125.        class FileProcessor:
126.            """Processing the data to and from text file"""
127.
128.            @staticmethod
129.            def read_file(file_name):
130.                """Function to read data from a .dat file using pickling.
131.
132.                Function checks to make sure specified txt file exists.  If yes, it
133.                continues by reading the data from .dat file and saving it to 'table'
134.                using pickling. If no, an empty list is assigned to 'Table'.
135.
136.                Args:
137.                    file_name (string): name of file used to read the data from
138.
139.                Returns:
140.                    table (list of dict): 2D data structure (list of dicts)
141.                """
142.                try:
143.                    with open(file_name, 'rb') as objFile:
144.                        table = pickle.load(objFile)
145.                except FileNotFoundError:
146.                    print('File not found. Inventory is still empty.')
147.                    table = []
148.                except Exception as e:
149.                    print('There has been a ', type(e), ' error with the read process. Inventory is st
    ill empty.')
```

```
                    table = []
                return table


            @staticmethod
            def write_file(file_name, table):
                """Function to write data from current runtime to a .dat file using pickling.

                The data from 'table' is saved to the designated .dat file using
                pickling.

                Args:
                    file_name (string): name of file used to write data to
                    table (list of dicts): 2D data structure (list of dicts) that holds data during ru
    ntime

                Returns:
                    None.
                """
                try:
                    with open(file_name, 'wb') as objFile:
                        pickle.dump(table, objFile)
                except Exception as e: #if there is a general error with the save process
                    print('There has been a ', type(e), ' error with the save process. File not saved.
    ')


        # -- PRESENTATION (Input/Output) -- #

        class IO:
            """Handling Input / Output"""

            @staticmethod
            def print_menu():
                """Displays a menu of choices to the user.


                Args:
                    None.

                Returns:
                    None.
                """
                print('\n')
                print(' MENU '.center(30,'-'))
                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')
                print('-'*30)
                print('\n')


            @staticmethod
            def menu_choice():
                """Gets user input for menu selection.


                Args:
                    None.

                Returns:
                    choice (string): an upper case sting of the users input out of the choices l, a, i
    , d, s or x

```

```
                """
                choice = ' '
                while choice not in ['L', 'A', 'I', 'D', 'S', 'X', 'l', 'a', 'i', 'd', 's', 'x']:
                    choice = input('Which operation would you like to perform? [L, A, I, D, S or X]: '
).lower().strip()
                print()  # Add extra space for layout
                return choice


            @staticmethod
            def show_inventory(table):
                """Displays current inventory table.


                Args:
                    table (list of dict): 2D data structure (list of dicts) that holds the data during
 runtime.

                Returns:
                    None.

                """
                print('\n')
                print(' The Current Inventory: '.center(60,'='))
                print('{:<10}{:<25}{:<25}'.format('ID', 'CD Title', 'Artist'))
                print('-'*60)
                for row in table:
                    print('{:<10}{:<25}{:<25}'.format(*row.values()))
                print('='*60)


            @staticmethod
            def get_cd_data():
                """Collects information about CD from user.

                Asks user to input the name of the CD and the artist of the CD.

                Args:
                    None

                Returns:
                    cd_title (str): name of CD, provided by user
                    cd_artist (str): name of artist, provided by user

                """

                try:
                    cd_title = input('What is the CD\'s title? ').strip()
                except Exception as e: #if user manages to cause an error with their input
                    print('You have entered an response that caused a ', type(e),' error.')
                    print('CD Title = NA')
                    cd_title = 'NA'
                try:
                    cd_artist = input('What is the Artist\'s name? ').strip()
                except Exception as e: #if user manages to cause an error with their input
                    print('You have entered an response that caused a ', type(e),' error.')
                    print('CD Artist = NA')
                    cd_artist = 'NA'
                return(cd_title, cd_artist)


        # 0 general blanket try/except block around main script to catch anything I missed
        try:
```

```
275.       # 1. When program starts, read in the currently saved inventory, print program header
276.           lstTbl = FileProcessor.read_file(strFileName)
277.           print('\n~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n')
278.           print('The Magic CD Inventory'.center(62))
279.           print('\n~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~')
280.
281.       # 2. start main loop
282.           while True:
283.               # 2.1 Display Menu to user and get choice
284.               IO.print_menu()
285.               strChoice = IO.menu_choice()
286.               # 3. Process menu selection
287.               # 3.1 process exit first
288.               if strChoice == 'x':
289.                   break
290.               # 3.2 process load inventory
291.               if strChoice == 'l':
292.                   print('WARNING: If you continue, all unsaved data will be lost and the Inventory w
    ill be reloaded from file.')
293.                   try:
294.                       strYesNo = input('Type \'yes\' to continue and reload from file - otherwise re
    load will be cancelled. ')
295.                       strYesNo.lower() #testing to see if lower() causes an error
296.                   except Exception as e: #if user manages to cause an error with their input
297.                       print('You have entered an response that caused a ', type(e),' error. Returnin
    g to Main Menu.')
298.                       continue #returns user to Main Menu
299.                   else:
300.                       if strYesNo.lower() == 'yes':
301.                           print('Reloading...')
302.                           lstTbl = FileProcessor.read_file(strFileName)
303.                           print('Inventory Loaded.')
304.                       else:
305.                           print('Cancelling... Inventory data NOT reloaded.')
306.                       IO.show_inventory(lstTbl)
307.                   try: input('Press [ENTER] to return to Main Menu. ')
308.                   except Exception as e: #if user manages to cause an error with their input
309.                       print('You have entered an response that caused a ', type(e),' error. Returnin
    g to Main Menu.')
310.                   finally: continue  # start loop back at top.
311.               # 3.3 process add a CD
312.               elif strChoice == 'a':
313.                   # 3.3.1 Generate ID and Ask user for CD Title and Artist
314.                   intID = DataProcessor.calculate_cd_id(lstTbl)
315.                   strTitle, strArtist = IO.get_cd_data()
316.                   # 3.3.2 Add item to the table
317.                   dicRow = DataProcessor.process_cd_data(intID, strTitle, strArtist)
318.                   lstTbl.append(dicRow)
319.                   IO.show_inventory(lstTbl)
320.                   try: input('CD Added. Press [ENTER] to return to Main Menu. ')
321.                   except Exception as e: #if user manages to cause an error with their input
322.                       print('You have entered an response that caused a ', type(e),' error. Returnin
    g to Main Menu.')
323.                   finally: continue  # start loop back at top.
324.               # 3.4 process display current inventory
325.               elif strChoice == 'i':
326.                   IO.show_inventory(lstTbl)
327.                   try: input('Press [ENTER] to return to Main Menu. ')
328.                   except Exception as e: #if user manages to cause an error with their input
329.                       print('You have entered an response that caused a ', type(e),' error. Returnin
    g to Main Menu.')
330.                   finally: continue  # start loop back at top.
331.               # 3.5 process delete a CD
332.               elif strChoice == 'd':
333.                   # 3.5.1 get Userinput for which CD to delete
```

```
                    # 3.5.1.1 display Inventory to user
                    IO.show_inventory(lstTbl)
                    # 3.5.1.2 ask user which ID to remove
                    try:
                        intIDDel = int(input('Which ID would you like to delete? ').strip())
                    except ValueError: #if user enters a non-integer
                        print('This is not an integer.')
                    except Exception as e: #if user manages to cause an error with their input
                        print('You have entered an response that caused a ', type(e),' error.')
                    # 3.5.2 search thru table and delete CD
                    else:
                        intRowNr, blnCDRemoved = DataProcessor.process_delete(intIDDel, lstTbl)
                        if blnCDRemoved:
                            del lstTbl[intRowNr]
                            print('CD #' + str(intIDDel) + ' deleted.')
                        else:
                            print('CD #' + str(intIDDel) + ' not found.')
                        IO.show_inventory(lstTbl)
                    try: input('Press [ENTER] to return to Main Menu. ')
                    except Exception as e: #if user manages to cause an error with their input
                        print('You have entered an response that caused a ', type(e),' error. Returnin
    g to Main Menu.')
                    finally: continue  # start loop back at top.
                # 3.6 process save inventory to file
                elif strChoice == 's':
                    # 3.6.1 Display current inventory and ask user for confirmation to save
                    IO.show_inventory(lstTbl)
                    try:
                        strYesNo = input('Save this inventory to file? [y/n] ').strip().lower()
                    # 3.6.2 Process choice
                        if strYesNo == 'y':
                            # 3.6.2.1 save data
                            FileProcessor.write_file(strFileName, lstTbl)
                            input('The inventory was saved to file. Press [Enter] to return to Main Me
    nu.')
                        else:
                            input('The inventory was NOT saved to file. Press [Enter] to return to Mai
    n Menu.')
                    except Exception as e: #if user manages to cause an error with their input
                        print('You have entered an response that caused a ', type(e),' error. Returnin
    g to Main Menu.')
                    finally:
                        continue  # start loop back at top.
                # 3.7 catch-
    all should not be possible, as user choice gets vetted in IO, but to be safe:
                else:
                    print('General Error')
        except Exception as e: #if user manages to cause an error with their input
            print('You have entered an response that caused a ', type(e),' error')
```