Christine Buffalow
2021-Mar-6
Foundations of Programming
Assignment 8

Creating a CD Inventory with Custom Classes

Introduction

This module introduced us to Object Oriented Programming! For this assignment, we were provided with a starter script that included the pseudocode outline for the latest version of our CD Inventory. At first, I was completely overwhelmed because it felt like I had to write the entire script from scratch using my very novice understanding of OOP. This quote from realpython.com summarizes how I finally got past the fear of the blank script:

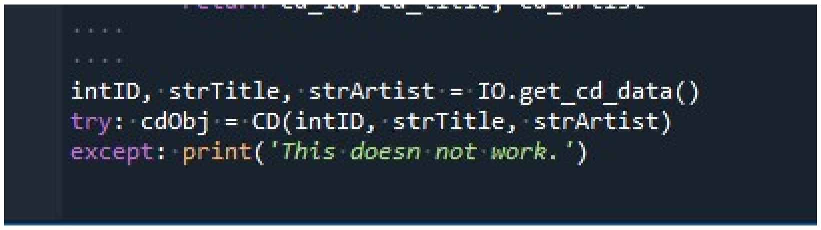
> *"Problems (in life and also in computer science) can often seem big and scary. But if we keep chipping away at them, more often than not we can break them down into smaller chunks trivial enough to solve."* [1]

As I started working through the script bit by bit, I realized that a lot of components could be used from the previous assignments as long as they were adjusted slightly. However, my limited grasp of OOP certainly led to some complications along the way…

class CD

Kernel Crashing & Issues with Recursion

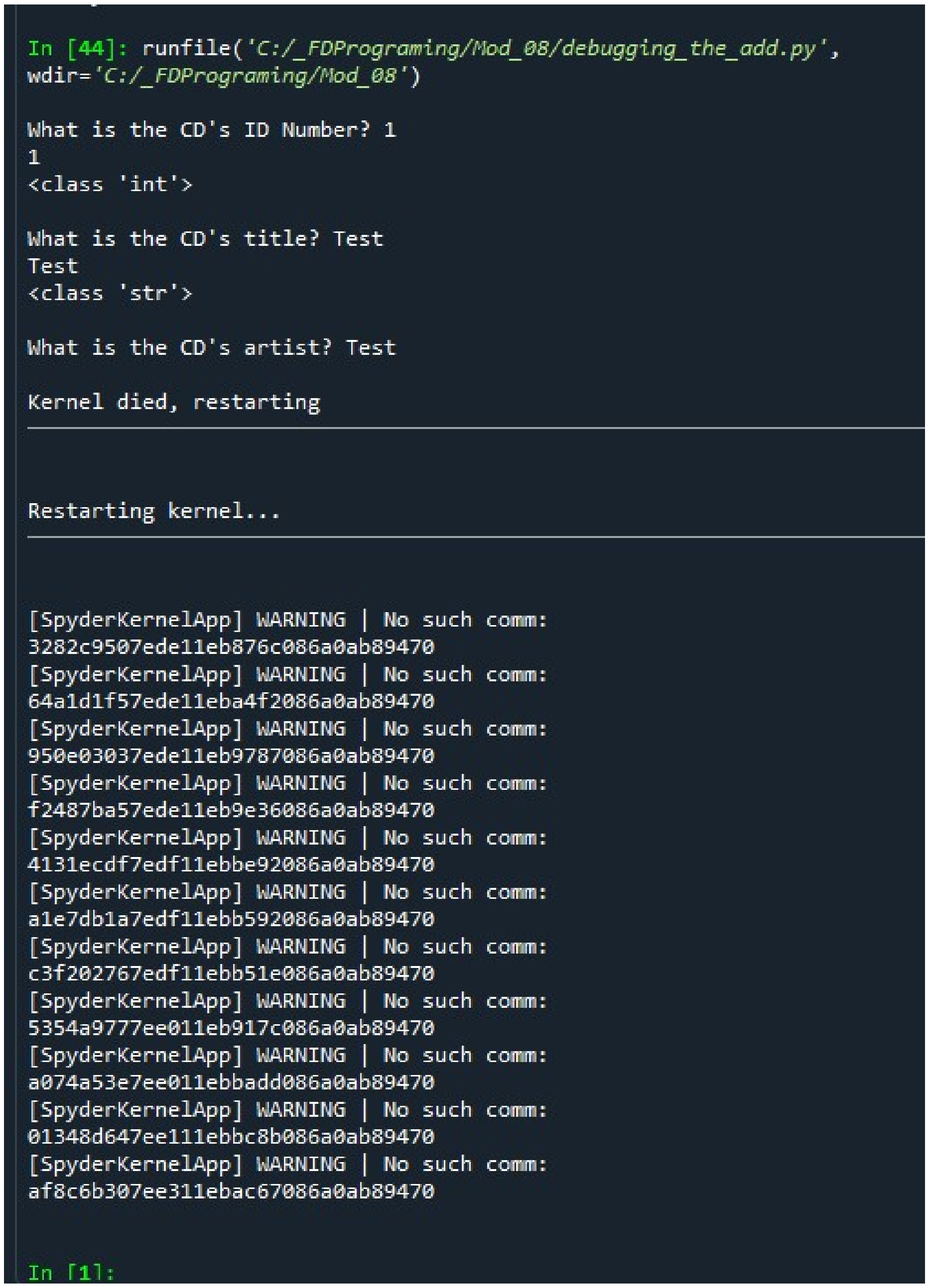
Creating the code inside the custom class `'CD'` was the most difficult and time-intensive aspect of this script because (1) I am still working on understanding the different components and nuances of Object Oriented Programming and (2) I was doing something in the code that kept causing my kernel to crash and restart. Although I hadn't done a complete test of the entire program, I knew that the error was at least happening any time I attempted to add a CD. As such, I knew the error could involve either my CD class, my `IO.get_cd_data` function, or something in the main script. To isolate the issue, I copied all the code from the 'CD' class, the `IO.get_cd_info` function, and a few relevant lines of code from the main script into a new script. I also added a handful of print statements throughout. A small selection of code from the new script can be seen in Figure 1 and the results can be seen in Figure 2.



```
    ....
    ....
    intID, strTitle, strArtist = IO.get_cd_data()
    try: cdObj = CD(intID, strTitle, strArtist)
    except: print('This doesn not work.')
```

Figure 1 - Test script to figure out error in CD class and/or IO.get_cd_info function

[1] https://realpython.com/python-thinking-recursively/, accessed 2021-Mar-7



```
In [44]: runfile('C:/_FDPrograming/Mod_08/debugging_the_add.py',
wdir='C:/_FDPrograming/Mod_08')

What is the CD's ID Number? 1
1
<class 'int'>

What is the CD's title? Test
Test
<class 'str'>

What is the CD's artist? Test

Kernel died, restarting


Restarting kernel...



[SpyderKernelApp] WARNING | No such comm:
3282c9507ede11eb876c086a0ab89470
[SpyderKernelApp] WARNING | No such comm:
64a1d1f57ede11eba4f2086a0ab89470
[SpyderKernelApp] WARNING | No such comm:
950e03037ede11eb9787086a0ab89470
[SpyderKernelApp] WARNING | No such comm:
f2487ba57ede11eb9e36086a0ab89470
[SpyderKernelApp] WARNING | No such comm:
4131ecdf7edf11ebbe92086a0ab89470
[SpyderKernelApp] WARNING | No such comm:
a1e7db1a7edf11ebb592086a0ab89470
[SpyderKernelApp] WARNING | No such comm:
c3f202767edf11ebb51e086a0ab89470
[SpyderKernelApp] WARNING | No such comm:
5354a9777ee011eb917c086a0ab89470
[SpyderKernelApp] WARNING | No such comm:
a074a53e7ee011ebbadd086a0ab89470
[SpyderKernelApp] WARNING | No such comm:
01348d647ee111ebbc8b086a0ab89470
[SpyderKernelApp] WARNING | No such comm:
af8c6b307ee311ebac67086a0ab89470


In [1]:
```

Figure 2 - Kernel dying/restarting issue

As a next step, I created a second new script with just the CD class and wrote a simple script that would create a new object and then print each component of this object. The crashing kernel continued. At this point, I started a third new script and just wrote a class with one attribute and one getter/setter property combo. Once I got that to work, I added the second and then the third. Somehow, in the process of slowing down and focusing on one small piece at a time, I was able to write the code correctly. After our discussion in Office Hours, I believe my lack of double underscores in a couple places may be to blame for causing a recursion issue. From my research:

"Recursion is a common mathematical and programming concept. It means that a function calls itself. This has the benefit of meaning that you can loop through data to reach a result. The developer should be very careful with recursion as it can be quite easy to slip into writing a function which never

terminates, or one that uses excess amounts of memory or processor power. However, when written correctly recursion can be a very efficient and mathematically-elegant approach to programming." [2]

From my understanding, this happened because I was (unintentionally) asking Spyder to perform a function call within the CD class repeatedly and indefinitely. Once this issue was resolved, I only encountered small errors from syntax errors that were easy to resolve.

Constructor & Attributes

The constructor of class CD has 4 arguments: the reference to itself, `cd_id`, `cd_title`, and `cd_artist`. Whenever a new object is instantiated in the script, the constructor does its thing. In this case, it sets up the three attributes inside the object and give them their initial values. Because I want the incoming information provided in the argument to be processed through the property setters, I created three empty attributes (`__cd_id__cd_title`, and `__cd_artist`, all assigned the value of `'None'`) and then created function calls to their respective "property setter" method (`self.cd_id`, `self.cd_title`, and `self.cd_artist`). As the information (`cd_id`, `cd_title`, and `cd_artist`) is processed through their appropriate "property setter", the resulting information is assigned to the attributes. The code for this can be seen in Listing 1.

```
    # -- Constructor -- #
    def __init__(self, cd_id, cd_title, cd_artist):
    #    -- Attributes -- #
        self.__cd_id = None
        self.__cd_title = None
        self.__cd_artist = None
        self.cd_id = cd_id
        self.cd_title = cd_title
        self.cd_artist = cd_artist
```

Listing 1 - Constructor and Attributes of CD class

Properties

For each of the three attributes (`__cd_id__cd_title`, and `__cd_artist`), I created a property "getter" and a property "setter." For each "setter", there are two arguments – the first is the `'self'` reference and the second is the proposed new value of the respective attribute (`new_id` for `__cd_id`, for example). If the value type is of the desired type (`'int'` for `__cd_id` and `'str'` for `__cd_title` and `__cd_artist`), the value is assigned to the appropriate attribute and returned by the property "getter". One example of this can be seen in Listing 2.

```
 # -- Properties -- #
    @property
    def cd_id(self):
        return self.__cd_id

    @cd_id.setter
    def cd_id(self, new_id):
        if type(new_id) == int:
            self.__cd_id = new_id
        else:
            raise Exception('This is not an integer.')
```

Listing 2 - Example of property getter and setter for CD's ID number

[2] https://www.w3schools.com/python/gloss_python_function_recursion.asp#:~:text=Python%20also%20accepts%20function%20recursion,that%20a%20function%20calls%20itself.&text=The%20recursion%20ends%20when%20the,i.e.%20when%20it%20is%200)., accessed 2021-Mar-07

Methods

In order to display and/or format the information from an object in a custom way, it is necessary to create a string method. The `__str__` method is connected to the `print` function in that if you use the `print` function on the object, the `__str__` method determines what is displayed from the object and how it is formatted. In this case, I overwrote the default `__str__` method with my own because I wanted to display the information about each CD in a particular way within each row of my inventory. As with my previous assignment, I specified an exact length for each component of the string and designated that the text would be justified to the left. I also cast the ID to a string since it is currently an integer and needs to be a string in order to be added to a string. The method then returns a formatted string. The code for this can be seen in Listing 3.

```
    def __str__(self):
        """Returns a formatted string with cd_id, cd_title, and cd_artist

        Args:
            None.

        Returns:
            cdRow (str): CD Info in string formatted for display

        """
        cdRow = '{:<10}{:<25}{:<25}'.format(str(self.cd_id), self.cd_title, self.cd_artist)
        return cdRow
```

Listing 3 - __str__ method for CD class

In order to facilitate the process of saving the inventory to a comma-separated text file, I also created a `file_string` method that would format the data in a comma separated string with a new line character at the end. As before, the method returns a formatted string. The code for this can be seen in Listing 4.

```
        def file_string(self):
              """Returns a formatted string with cd_id, cd_title, and cd_artist for a text file

              Args:
                  None.

              Returns:
                  fileRow (str): CD Info in string formatted for txt file

              """
              fileRow = '{},{},{}\n'.format(str(self.cd_id), self.cd_title, self.cd_artist)
              return fileRow
```

Listing 4 - file_string method for CD class

class FileIO

For the `FileIO` class, we were directed to add two methods: one to process data from a file and one to process data to a file. Although I had originally created these using binary data via pickling, I was instructed to use a txt file instead during Office Hours. As such, I took the code from my Module 6 assignment and adapted it for this assignment. Since the code has already been explained in detail in my previous knowledge documents, I will keep my discussion brief and primarily focusing on the changes made for this assignment. The updated code for these methods can be seen in Appendix CD_Inventory.py unless otherwise specified.

All FileIO Methods – Error Functioning

The code for all methods were put inside general 'try/except' blocks that will catch any error that occurs and will return the user to the main menu with an error statement. In addition, for the load_inventory function, the 'FileNotFoundError' was specifically addressed.

FileIO.save_inventory

1. The content inside the 'for' loop was changed to reflect that we are now working with a list of objects rather than a list of dictionaries. As such, it is no longer necessary to extract the values from each dictionary and save them to a list.
2. Formatting can be done via a method within the CD class (the file_string method is discussed in class CD section) rather than within the 'for' loop.

In summary, the text file is opened with 'write' access. Then each object (i.e. row) in the list 'table' (i.e. inventory) is formatted for CSV using the file_string method with the CD class and then saved out to the text file. Once the 'for' loop has completed all the objects in the list, the 'for' loop is completed and the file is closed. The code for this can be seen in Listing 5.

```
        try:
            objFile = open(file_name, 'w')
            for obj in table:
                objFile.write(obj.file_string())
            objFile.close()
        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.')
```

Listing 5 - Code within FileIO.save_inventory function

FileIO.load_inventory

A small amount of code was updated to reflect that we are now working with a list of objects rather than a list of dictionaries. As before, the data is broken out into separate chunks based on the location of commas and new line characters and saved into a list called 'data'. However, instead using the list elements inside a dictionary, I am creating a new object using the CD class. The three elements of the list, 'data', serve as the three required arguments to create an object using the CD class (i.e. data[0], data[1], and data[2]). This object is then appending to the list, 'table', which is returned at the end of the function call. The code for this can be seen in Listing 6.

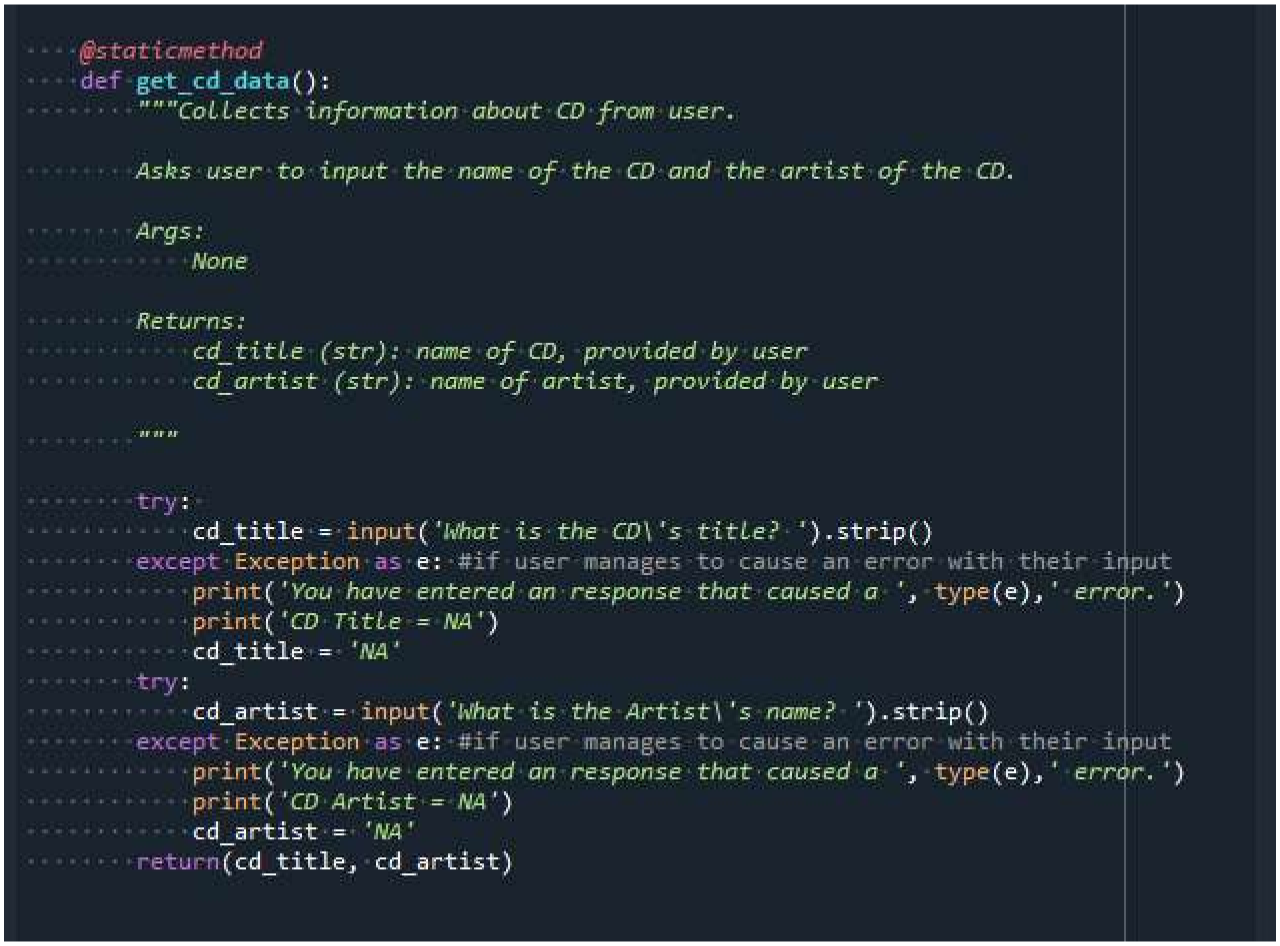
```
        table = []
        try:
            objFile = open(file_name, 'r')
            for row in objFile:
                data = row.strip().split(',')
                obj = CD(int(data[0]), data[1], data[2])
                table.append(obj)
            objFile.close()
        except FileNotFoundError:
            print('File not found. Inventory is still empty.')
        except Exception as e:
            print('There has been a ', type(e), ' error with the read process. Inventory is still empty.')
        return table
```

Listing 6 - Code within FileIO.load_inventory function

class IO

The majority of the code from this section was taken directly from my previous assignment. Since the code has already been explained in detail in my previous knowledge document, I will only be discussing the changes made for this assignment. The updated code for these methods can be seen in Appendix CD_Inventory.py unless otherwise specified.

1. I removed all references to "deleting a CD" from the function `IO.menu_choice()` since this assignment does not require that component.
2. Since formatting of the inventory rows (i.e. objects) is now done in the CD class via the `__str__` method, I removed all formatting of the rows in the function `IO.show_inventory()` and simply put `'print(row)'`.
3. Due to the fact that I already put a lot of time into this class this week learning about OOP and working through a recursion error in this assignment, I did not take the time to adapt my `cd_id` generating function from the previous assignment (although it would have been good practice working with variables in the 'Fields' section of the class!). Instead, I simply asked the user to provide an ID number. Since this program does not involve deleting CDs, I figured this would be fine since it is not essential that CD IDs are unique in this program.
4. I simplified the error functioning inside `IO.get_cd_data()`. First, I removed the creation of "dummy data" in response to errors (ex: cd_title = 'NA', etc.). Because I didn't want to add the complexity of a `'while' loop` for each `input` statement and I no longer wanted to replace error-causing data with dummy data, I set it up so the user would simply be returned to the main menu with an error statement if they triggered an error. I found the best and most efficient way to do that was to include all three `input` functions inside the one `try/except` block. For a reference to the previous assignment, see Figure 3. The new code is shown in Listing 7.



```
    @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)
```

Figure 3 - get_cd_data function from Assignment 07

```
            try:
                cd_id = int(input('What is the CD\'s ID Number? '))
                cd_title = input('What is the CD\'s title? ')
                cd_artist = input('What is the CD\'s artist? ')
                return cd_id, cd_title, cd_artist
            except ValueError:
                print('The number you entered for the CD\'s ID 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.')
```

Listing 7 - Code within IO.get_cd_data function for current assignment

Main Script

The majority of the code from this section was taken directly from my previous assignment. Since the code has already been explained in detail in my previous knowledge document, I will only be discussing the changes made for this assignment. The updated code for these methods can be seen in Appendix CD_Inventory.py.

1. Variable names were updated appropriately. (Some variable name changes reflect the fact that we're using objects instead of dictionaries, etc.)
2. The function call to `DataProcessor.calcualte_cd_id` and the entire section about deleting CDs was removed since this functionality was not carried over to this version of the CD inventory.
3. The function call to the `DataProcessor.process_cd_data` was removed since the data is now processed and formatted within the `CD` class via the properties and string methods.

Summary

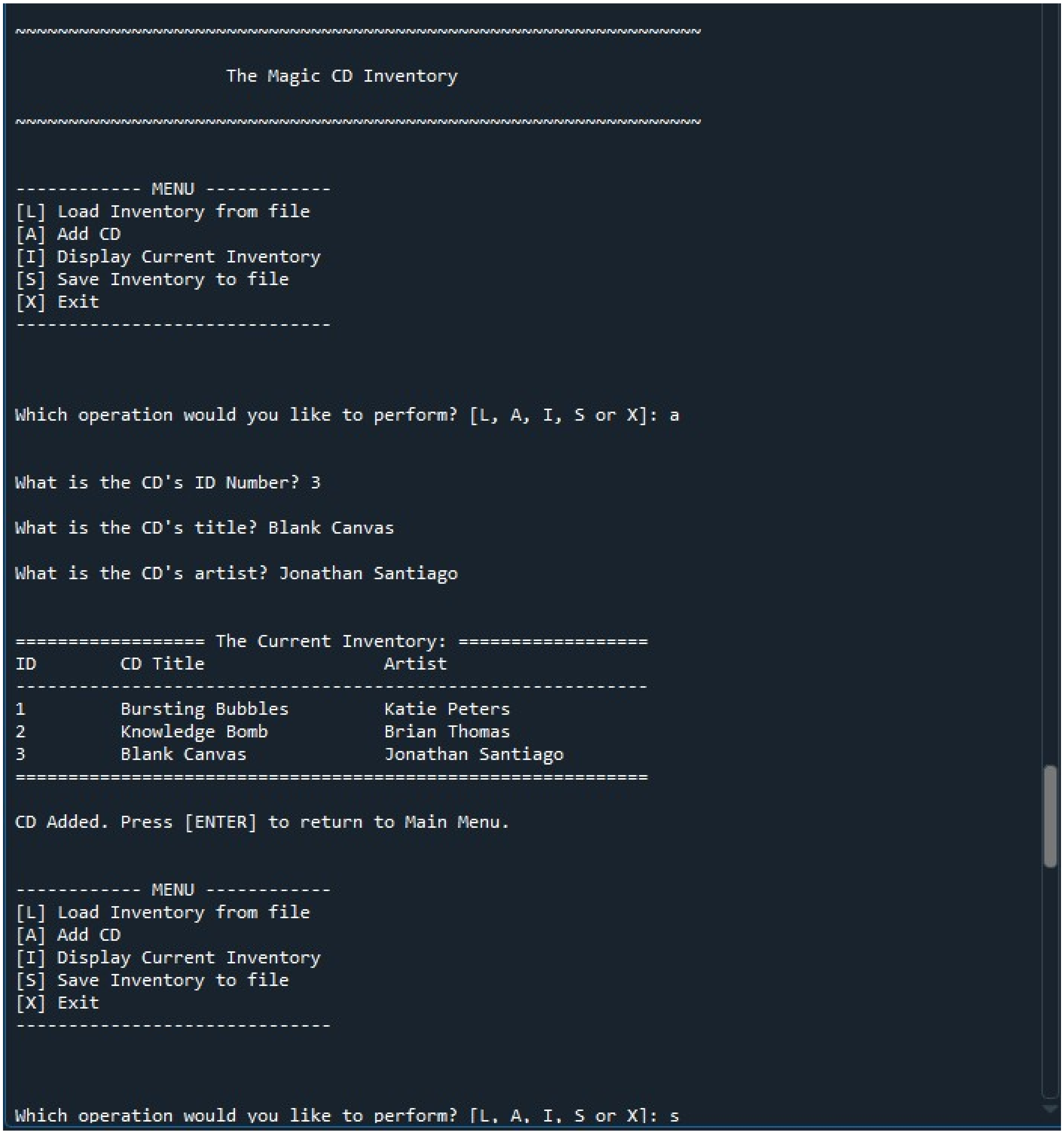
The most difficult part of this assignment was working through the recursion error and figuring out all the nuances of all the different components within a class. Although I managed to get a working script that performed all the expected tasks in the end, I know I still have a lot to learn in regards to OOP.

GitHub

The GitHub link for this assignment is: https://github.com/cmb225/Assignment_08

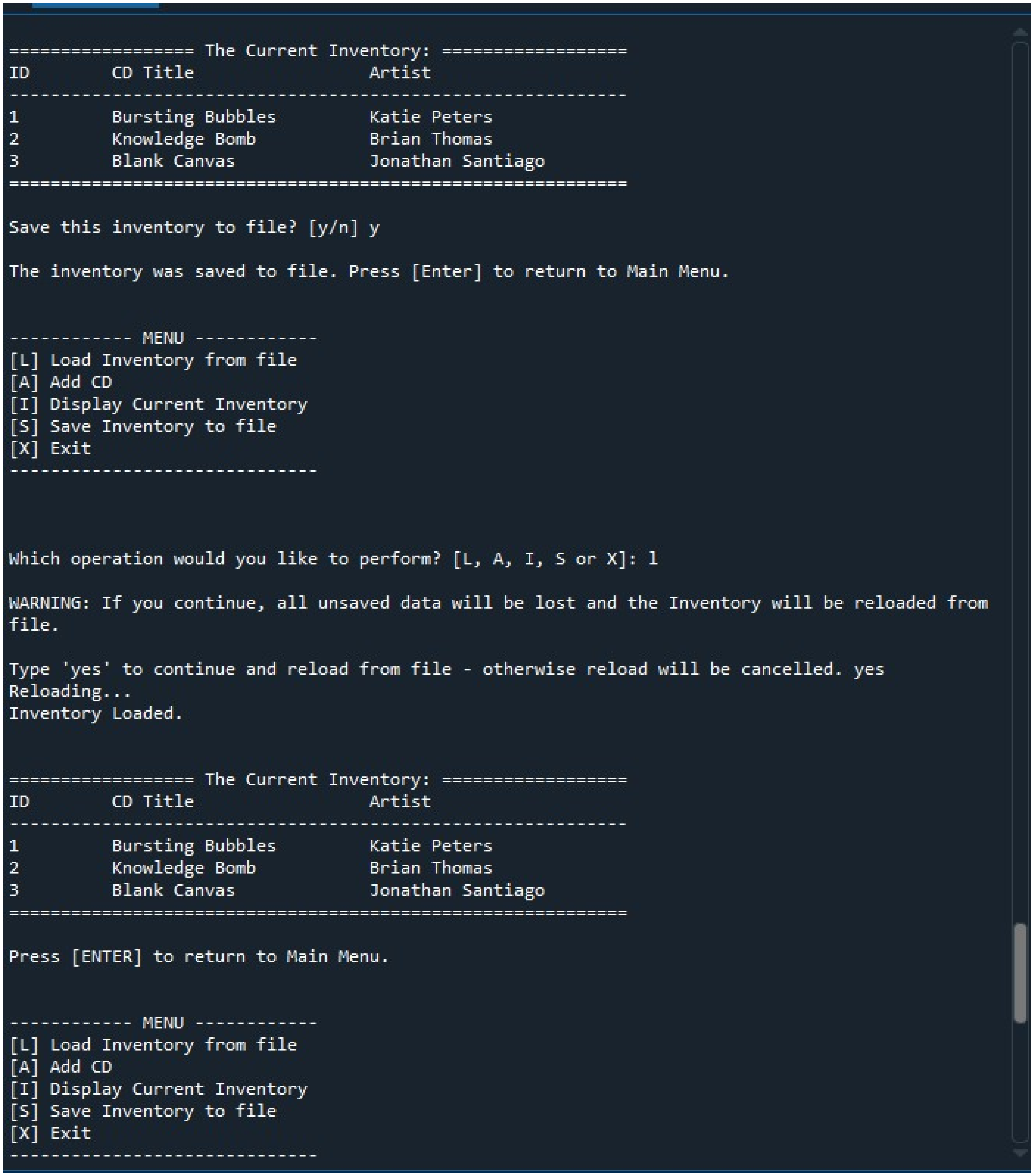
Functionality in Spyder

Program ran as expected in Spyder. See Figures 4 – 6. Resulting text file in Figure 9.



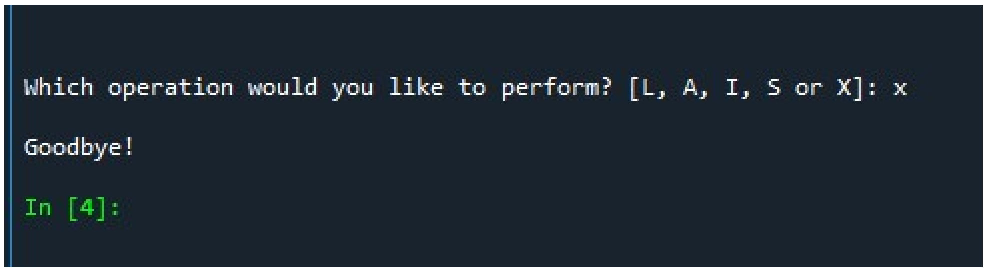
```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

                    The Magic CD Inventory

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


------------ MENU ------------
[L] Load Inventory from file
[A] Add CD
[I] Display Current Inventory
[S] Save Inventory to file
[X] Exit
------------------------------



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


What is the CD's ID Number? 3

What is the CD's title? Blank Canvas

What is the CD's artist? Jonathan Santiago


================== The Current Inventory: ==================
ID        CD Title                 Artist
------------------------------------------------------------
1         Bursting Bubbles         Katie Peters
2         Knowledge Bomb           Brian Thomas
3         Blank Canvas             Jonathan Santiago
============================================================

CD Added. Press [ENTER] to return to Main Menu.


------------ MENU ------------
[L] Load Inventory from file
[A] Add CD
[I] Display Current Inventory
[S] Save Inventory to file
[X] Exit
------------------------------



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

Figure 4 - Running script in Spyder



```
================== The Current Inventory: ==================
ID        CD Title                 Artist
------------------------------------------------------------
1         Bursting Bubbles         Katie Peters
2         Knowledge Bomb           Brian Thomas
3         Blank Canvas             Jonathan Santiago
============================================================

Save this inventory to file? [y/n] y

The inventory was saved to file. Press [Enter] to return to Main Menu.


------------ MENU ------------
[L] Load Inventory from file
[A] Add CD
[I] Display Current Inventory
[S] Save Inventory to file
[X] Exit
------------------------------



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

WARNING: If you continue, all unsaved data will be lost and the Inventory will be reloaded from
file.

Type 'yes' to continue and reload from file - otherwise reload will be cancelled. yes
Reloading...
Inventory Loaded.


================== The Current Inventory: ==================
ID        CD Title                 Artist
------------------------------------------------------------
1         Bursting Bubbles         Katie Peters
2         Knowledge Bomb           Brian Thomas
3         Blank Canvas             Jonathan Santiago
============================================================

Press [ENTER] to return to Main Menu.


------------ MENU ------------
[L] Load Inventory from file
[A] Add CD
[I] Display Current Inventory
[S] Save Inventory to file
[X] Exit
------------------------------
```

Figure 5 - Running script in Spyder

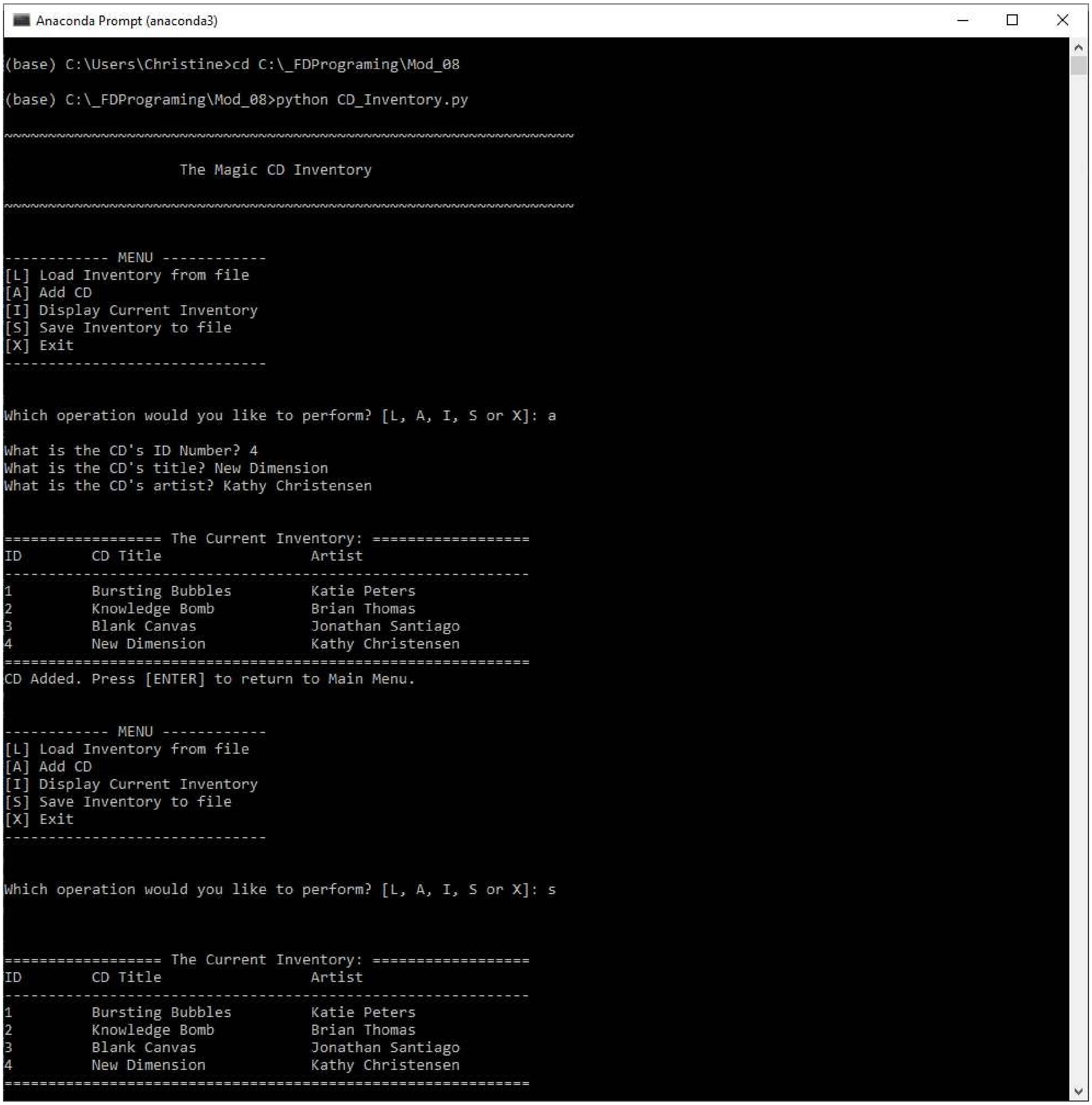


```
Which operation would you like to perform? [L, A, I, S or X]: x

Goodbye!

In [4]:
```

Figure 6 - Running script in Spyder

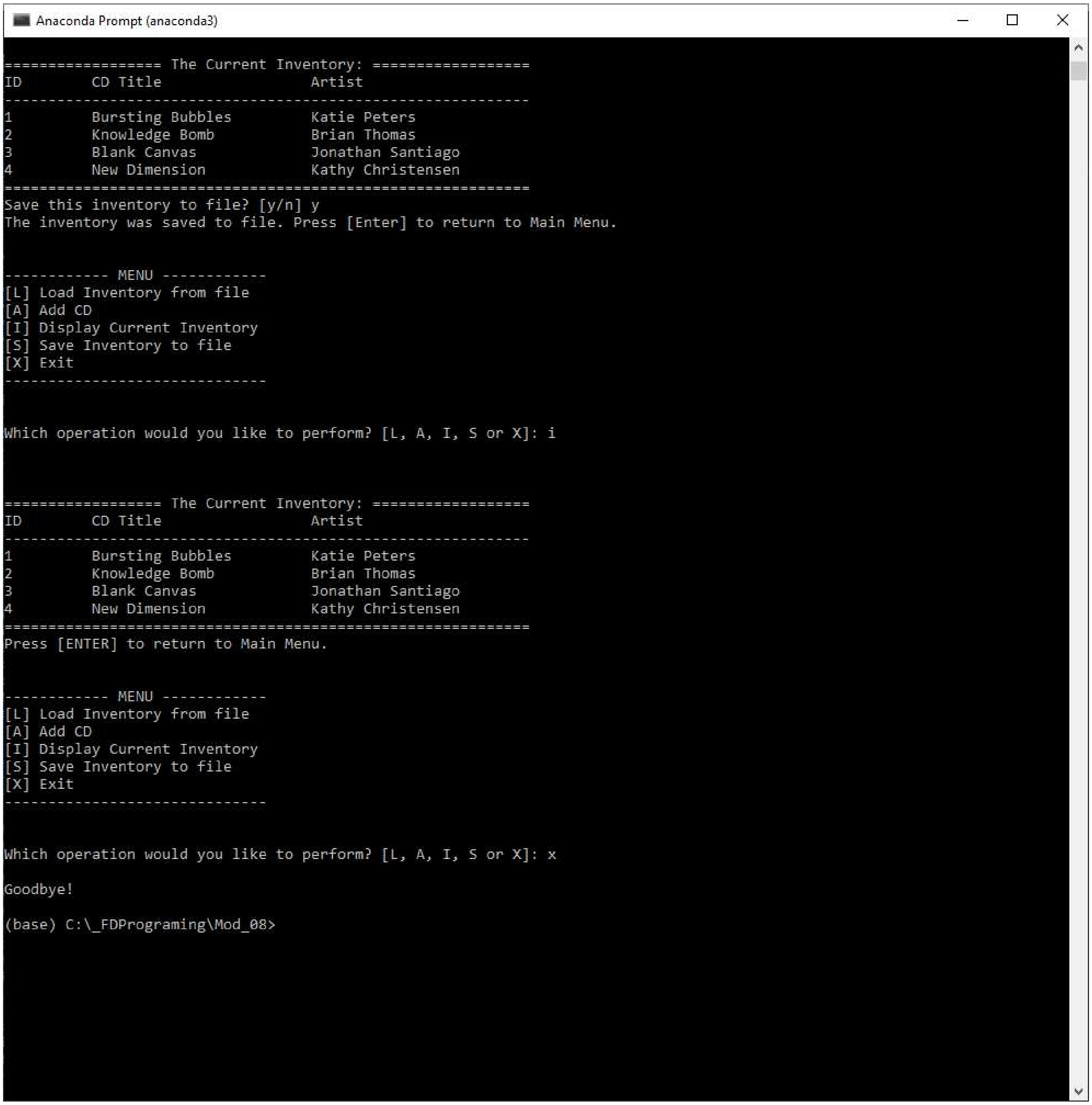
Functionality in Anaconda Prompt

Program ran as expected in Anaconda Prompt. See Figures 4 – 6. Resulting text file in Figure 9.



```
(base) C:\Users\Christine>cd C:\_FDPrograming\Mod_08

(base) C:\_FDPrograming\Mod_08>python CD_Inventory.py

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

                    The Magic CD Inventory

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



------------ MENU ------------
[L] Load Inventory from file
[A] Add CD
[I] Display Current Inventory
[S] Save Inventory to file
[X] Exit
------------------------------


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

What is the CD's ID Number? 4
What is the CD's title? New Dimension
What is the CD's artist? Kathy Christensen


================== The Current Inventory: ==================
ID        CD Title                 Artist
------------------------------------------------------------
1         Bursting Bubbles         Katie Peters
2         Knowledge Bomb           Brian Thomas
3         Blank Canvas             Jonathan Santiago
4         New Dimension            Kathy Christensen
============================================================
CD Added. Press [ENTER] to return to Main Menu.


------------ MENU ------------
[L] Load Inventory from file
[A] Add CD
[I] Display Current Inventory
[S] Save Inventory to file
[X] Exit
------------------------------


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



================== The Current Inventory: ==================
ID        CD Title                 Artist
------------------------------------------------------------
1         Bursting Bubbles         Katie Peters
2         Knowledge Bomb           Brian Thomas
3         Blank Canvas             Jonathan Santiago
4         New Dimension            Kathy Christensen
============================================================
```

Figure 7 - Running script in Anaconda Prompt



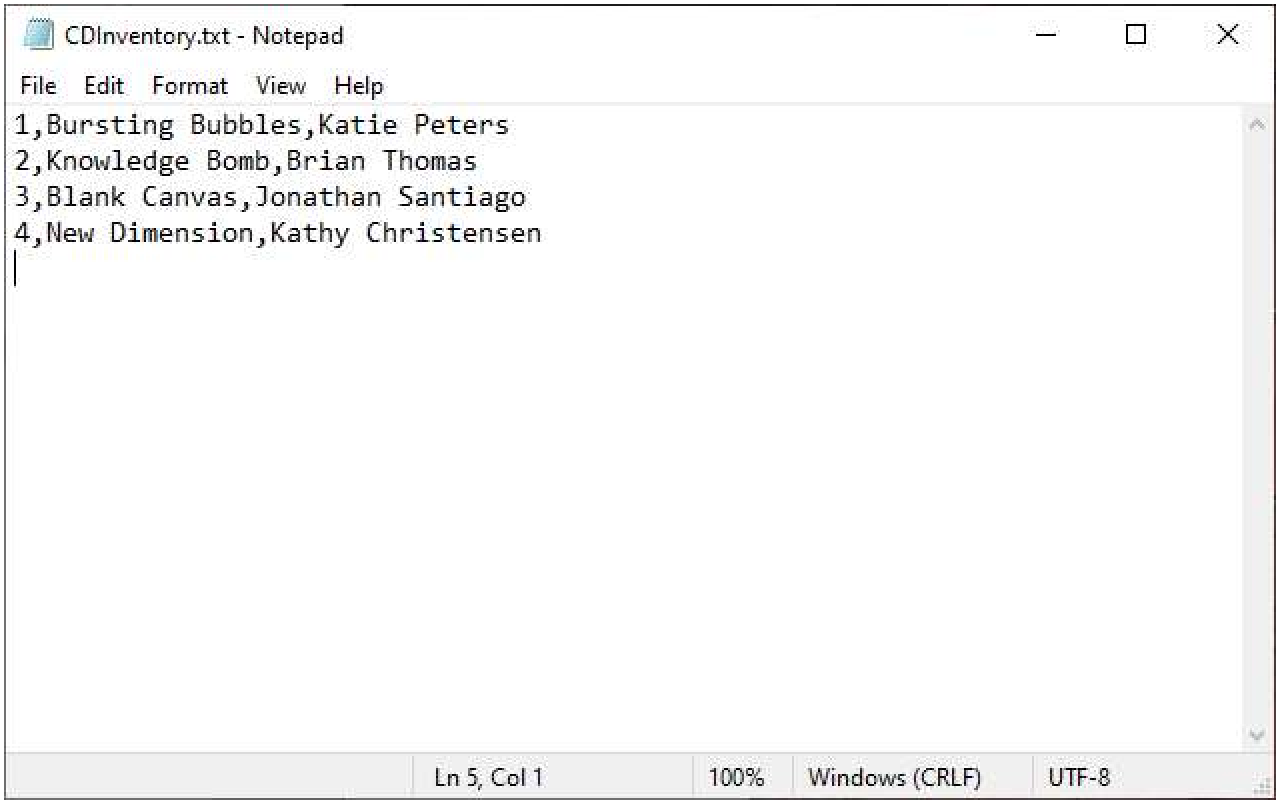
Anaconda Prompt (anaconda3)

```
================== The Current Inventory: ==================
ID        CD Title                 Artist
------------------------------------------------------------
1         Bursting Bubbles         Katie Peters
2         Knowledge Bomb           Brian Thomas
3         Blank Canvas             Jonathan Santiago
4         New Dimension            Kathy Christensen
============================================================
Save this inventory to file? [y/n] y
The inventory was saved to file. Press [Enter] to return to Main Menu.


------------ MENU ------------
[L] Load Inventory from file
[A] Add CD
[I] Display Current Inventory
[S] Save Inventory to file
[X] Exit
------------------------------


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



================== The Current Inventory: ==================
ID        CD Title                 Artist
------------------------------------------------------------
1         Bursting Bubbles         Katie Peters
2         Knowledge Bomb           Brian Thomas
3         Blank Canvas             Jonathan Santiago
4         New Dimension            Kathy Christensen
============================================================
Press [ENTER] to return to Main Menu.


------------ MENU ------------
[L] Load Inventory from file
[A] Add CD
[I] Display Current Inventory
[S] Save Inventory to file
[X] Exit
------------------------------


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

Goodbye!

(base) C:\_FDPrograming\Mod_08>
```

Figure 8 - Running script in Anaconda Prompt

Resulting Text Data File

Resulting text file from running script in Spyder and Anaconda Prompt in Figure 9.



```
1,Bursting Bubbles,Katie Peters
2,Knowledge Bomb,Brian Thomas
3,Blank Canvas,Jonathan Santiago
4,New Dimension,Kathy Christensen
```

Figure 9 - Data File

Appendix

Listing CD_Inventory.py

```
#------------------------------------------#
# Title: CD_Inventory.py
# Desc: Assignnment 08 - Working with classes
# Change Log: (Who, When, What)
# DBiesinger, 2030-Jan-01, created file
# DBiesinger, 2030-Jan-01, added pseudocode to complete assignment 08
# CBuffalow, 2021-Mar-06, added code to class CD, added save/load
#    inventory functions to FileIO class, and added code to IO class
# CBuffalow, 2021-Mar-06, added code to main body, added __str__ method to CD class
# CBuffalow, 2021-Mar-06, updated docstrings, changed from dat to txt data
#   storage,  added file_string function
# CBuffalow, 2021-Mar-07, updated save_inventory function, updated docstrings
#------------------------------------------#

# -- DATA -- #
#Global Variables
strFileName = 'CdInventory.txt'  # text storage file
lstOfCDObjects = [] # table to hold data (list of objects)
strChoice = '' # user input (string)
strYesNo = '' # user input for yes/no question (string)
intID = 0 # ID number of CD (integer)
strTitle = '' # user input (string)
strArtist = '' # user input (string)
cdObj = None # object with CD info
```

```python

class CD:
    """Stores data about a CD:

    properties:
        cd_id: (int) with CD ID
        cd_title: (string) with the title of the CD
        cd_artist: (string) with the artist of the CD
    methods:
        __str__(): returns a formatted string with cd_id, cd_title, and cd_artist
        file_string(): returns a formatted string with cd_id, cd_title, and cd_artist for txt file
    """

    # -- Fields -- #
    # -- Constructor -- #
    def __init__(self, cd_id, cd_title, cd_artist):
    #    -- Attributes -- #
        self.__cd_id = None
        self.__cd_title = None
        self.__cd_artist = None
        self.cd_id = cd_id
        self.cd_title = cd_title
        self.cd_artist = cd_artist

    # -- Properties -- #
    @property
    def cd_id(self):
        return self.__cd_id

    @cd_id.setter
    def cd_id(self, new_id):
        if type(new_id) == int:
            self.__cd_id = new_id
        else:
            raise Exception('This is not an integer.')

    @property
    def cd_title(self):
        return self.__cd_title

    @cd_title.setter
    def cd_title(self, new_title):
        if type(new_title) == str:
            self.__cd_title = new_title
        else:
            raise Exception('This is not a string.')

    @property
    def cd_artist(self):
        return self.__cd_artist

    @cd_artist.setter
    def cd_artist(self, new_artist):
        if type(new_artist) == str:
            self.__cd_artist = new_artist
        else:
            raise Exception('This is not a string.')

    # -- Methods -- #
    def __str__(self):
        """Returns a formatted string with cd_id, cd_title, and cd_artist

        Args:
            None.

        Returns:
            cdRow (str): CD Info in string formatted for display

        """
        cdRow = '{:<10}{:<25}{:<25}'.format(str(self.cd_id), self.cd_title, self.cd_artist)
        return cdRow

    def file_string(self):
        """Returns a formatted string with cd_id, cd_title, and cd_artist for a text file

            Args:
                None.

            Returns:
                fileRow (str): CD Info in string formatted for comma-separated txt file

            """
            fileRow = '{},{},{}\n'.format(str(self.cd_id), self.cd_title, self.cd_artist)
            return fileRow


# -- PROCESSING -- #
class FileIO:
    """Processes data to and from file:

    properties:

    methods:
        save_inventory(file_name, table): -> None
        load_inventory(file_name): -> (a list of CD objects)

    """

    # -- Fields -- #
    # -- Constructor -- #
    #    -- Attributes -- #
    # -- Properties -- #
    # -- Methods -- #

    @staticmethod
    def save_inventory(file_name, table):
        """Function to write data from current runtime into a text file.

        For each row, the data is converted into a string with a comma separating
        each piece of data and then a new line is started.  The data is then saved
        out to the designated txt file.

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

        Returns:
            None.
        """
        try:
            objFile = open(file_name, 'w')
            for obj in table:
                objFile.write(obj.file_string())
            objFile.close()
        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.')


    @staticmethod
    def load_inventory(file_name):
        """Function to manage data ingestion from file to a list of objects

        Function checks to make sure specified txt file exists.  If yes, continues by reading
        the data from file identified by file_name into a 2D table
        (list of objects) table. One line in the file represents one row in table.

        Args:
            file_name (string): name of file used to read the data from

        Returns:
            table (list of objects): 2D data structure (list of objects) that holds the data during runtime
        """
        table = []
        try:
            objFile = open(file_name, 'r')
            for row in objFile:
                data = row.strip().split(',')
                obj = CD(int(data[0]), data[1], data[2])
                table.append(obj)
            objFile.close()
        except FileNotFoundError:
            print('File not found. Inventory is still empty.')
        except Exception as e:
            print('There has been a ', type(e), ' error with the read process. Inventory is still empty.')
        return table

# -- PRESENTATION (Input/Output) -- #
class IO:
    """Handling Input / Output

    properties: None

    methods:
        print_menu(): -> None
        menu_choice(): -> (string of choice)
        show_inventory(a list of CD objects): -> None
        get_cd_data(): _> (integer of cd_id, str of cd_title, str of cd_artist)
    """
    # -- Fields -- #
    # -- Constructor -- #
    #    -- Attributes -- #
    # -- Properties -- #
    # -- Methods -- #

    @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('[S] Save Inventory to file\n[X] Exit')
        print('-'*30)
        print('\n')

```

```python
    @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, s or x

        """
        choice = ' '
        while choice not in ['L', 'A', 'I', 'S', 'X', 'l', 'a', 'i', 's', 'x']:
            choice = input('Which operation would you like to perform? [L, A, I, 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 obj): 2D data structure (list of obj) 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(row)
        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_id (int): ID number of CD, provided by user
            cd_title (str): name of CD, provided by user
            cd_artist (str): name of artist, provided by user

        """
        try:
            cd_id = int(input('What is the CD\'s ID Number? '))
            cd_title = input('What is the CD\'s title? ')
            cd_artist = input('What is the CD\'s artist? ')
            return cd_id, cd_title, cd_artist
        except ValueError: #if user enters a non-integer for cd_id
            print('The number you entered for the CD\'s ID 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.')


# -- Main Body of Script -- #

# 0 general blanket try/except block around main script to catch anything I missed
try:
# 1. When program starts, read in the currently saved inventory, print program header
    lstOfCDObjects = FileIO.load_inventory(strFileName)
    print('\n~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n')
    print('The Magic CD Inventory'.center(62))
    print('\n~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~')

# 2. start main loop
    while True:
        # 2.1 Display Menu to user and get choice
        IO.print_menu()
        strChoice = IO.menu_choice()
        # 3. Process menu selection
        # 3.1 process exit first
        if strChoice == 'x':
            print('Goodbye!')
            break
        # 3.2 process load inventory
        if strChoice == 'l':
            print('WARNING: If you continue, all unsaved data will be lost and the Inventory will be reloaded from file.')
            try:
                strYesNo = input('Type \'yes\' to continue and reload from file - otherwise reload will be cancelled. ')
                strYesNo.lower() #testing to see if lower() causes an 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. Returning to Main Menu.')
                continue #returns user to Main Menu
            else:
                if strYesNo.lower() == 'yes':
                    print('Reloading...')
                    lstOfCDObjects = FileIO.load_inventory(strFileName)
                    print('Inventory Loaded.')
                else:
                    print('Cancelling... Inventory data NOT reloaded.')
                IO.show_inventory(lstOfCDObjects)
            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. Returning to Main Menu.')
            finally: continue  # start loop back at top.
        # 3.3 process add a CD
        elif strChoice == 'a':
            # 3.3.1 Generate ID and Ask user for CD Title and Artist
            #intID = DataProcessor.calculate_cd_id(lstTbl)
            try:
                intID, strTitle, strArtist = IO.get_cd_data()
            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. Returning to Main Menu.')
                continue
            # 3.3.2 Add item to the table
            cdObj = CD(intID, strTitle, strArtist)
            lstOfCDObjects.append(cdObj)
            IO.show_inventory(lstOfCDObjects)
            try: input('CD Added. 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. Returning to Main Menu.')
            finally: continue  # start loop back at top.
```

```python
                # 3.4 process display current inventory
                elif strChoice == 'i':
                    IO.show_inventory(lstOfCDObjects)
                    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. Returning to Main Menu.')
                    finally: continue  # start loop back at top.
                # 3.5 process save inventory to file
                elif strChoice == 's':
                    # 3.5.1 Display current inventory and ask user for confirmation to save
                    IO.show_inventory(lstOfCDObjects)
                    try:
                        strYesNo = input('Save this inventory to file? [y/n] ').strip().lower()
                    # 3.5.2 Process choice
                        if strYesNo == 'y':
                            # 3.5.2.1 save data
                            FileIO.save_inventory(strFileName, lstOfCDObjects)
                            input('The inventory was saved to file. Press [Enter] to return to Main Menu.')
                        else:
                            input('The inventory was NOT saved to file. 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. Returning to Main Menu.')
                    finally:
                        continue  # start loop back at top.
                # 3.6 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')
```