Rei Bengu
August 20, 2021
IT FDN 110 B
Assignment 07

Binary Files and Error Handling

Introduction

In this module we moved from the traditional file management methods we have been using of saving data as strings in a text file, then reading the text file and converting strings back into their intended data format and moved towards binary data files instead. With binary data files, we can save data as is into a file by pickling it, and upon unpickling the file, the data is read directly in its intended structure, trivializing the process of working with files. Additionally, we worked with structured error handling, to provide our programs with the means to overcome errors if a user input is an unintended format and cascades through the program, or other modes of failure.

The method used to work with binary files is known as pickling. Pickling saves the information by serializing object structures, similar to if you wanted to save vegetables for later. The Python docs on the Pickle function provide a lot of very useful information on how the pickle function behaves and can be invoked.

Structured error handling provides alternatives for a program in the event of an error event taking place. For example, if your script attempted to convert a string such as ‘four’ into an integer, normally the program would crash with the ‘Value Error’ message. If we establish structured error handling around the conversion to integer to account for this possibility, the script will route through that error message and not crash. Structured error handling is similar to conditional statements, if the main code throws an error, check the next error message, etc. The Python docs on errors and exceptions provide a lot of very useful information on how errors are treated and can be treated to avoid a fatal error of the script.

The script used to practice these methods as well as this knowledge document can also be found at github to see the progression of this assignment.

CD Inventory Script

To put to practice the skills learned in this module, the basic script created and refined over past modules which asked the user to enter basic CD data to create an expanding table of information was revisited and updated with the concepts learned in this module on functions and classes. Because this program has gone through several iterations, I will be focusing primarily on the new features of this iteration and its propagated effects on the program and will not re-iterate the contents of the program that can be found in previous modules.

The next iteration of this program will shift from utilizing text files to using binary data files instead via the pickling method. Right off the bat the benefits of pickling data instead of converting and saving to a text file are apparent to the programmer, as intermediary steps to convert data between saving and reading are no longer necessary, which means less chances to make a mistake, less code to debug, and an overall cleaner code. A drawback of using binary files instead of text files however is that the user cannot view or interpret the information without additional steps needing to be taken to convert the information out of binary.

The code below shows the FileProcessor class which consists of 2 objects ‘read_file()’ and ‘write_file()’ that utilize the pickle function to work with binary data. The pickle function needs to first be imported which is done at the start of the script and shown below in line 1. For the read_file() function, the pickle.load(objFile) function imports the binary data stored within the objFile. Because of how binary files work, the data is stored in its original format the way it was saved, so no further work is needed to ensure the format of the data is in the state it is needed in. The same is true for writing data to a file, covered in the write_fil() function, which uses the pickle.dump(table, objFile) to serialize and save the data in the 'table' to the 'objFile'. Similarly, to being able to read data as it was written, data can be written to a file the same way it is stored in local memory. This turns the several lines of code required to utilize text files into just a couple of lines instead.

Another new addition to the program which can be seen in the code below is the structured error handling from lines 23 through 32. Structured error handling is similar in format to conditional statements; the body of the code intended to be ran is contained within the 'try:' block, but if an error occurs, it gets routed through the appropriate 'except' block. In this case of reading files, the most common error that I ran into while trying to break the code was the FileNotFound error, thus I added an exception within the read_file object to handle this type of error, so that if it occurs the program will instead print out error information to the user and carry on, instead of crashing.

```
1. import pickle
2.
3. class FileProcessor:
4.     """Processing the data to and from text file"""
5.
6.     @staticmethod
7.     def read_file(file_name, table):
8.         """
9.         Function to manage data ingestion from file to a list of dictionaries
10.
11.             Reads the data from file identified by file_name into a 2D table
12.             (list of dicts) table one line in the file represents one dictionary row in table.
13.
14.             Args:
15.                 file_name (string): name of file used to read the data from
16.                 table (list of dict): 2D data structure (list of dicts) that holds the data during runtime
17.
18.             Returns:
19.                 table (list of dict): 2d data structure (list of dicts) that holds the data during runtime
20.
21.             """
22.
23.             try:
24.                 table.clear()
25.                 with open(file_name, 'rb') as objFile:
26.                     table = pickle.load(objFile)
27.                 return table
28.
29.             except FileNotFoundError as e:
30.                 print('Data file does not exist!')
31.                 print('Built in error info:')
32.                 print(type(e), e, e.__doc__, sep='\n')
```

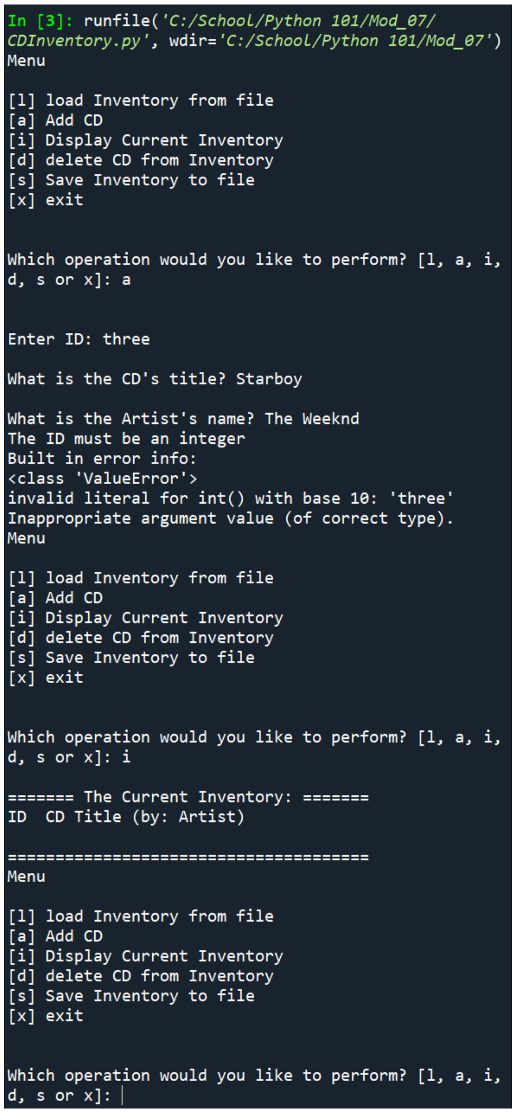
```


         @staticmethod
         def write_file(file_name, table):
             """
             Writes data from list of dictionaries to file

             Args:
                 file_name (string): name of file to save data to

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

             Returns:
                 None
             """

             with open(file_name, 'wb') as objFile:
                 pickle.dump(table, objFile)
             objFile.close()
```

Another instance of structured error handling used within this program can be seen below using the DataProcessor.add_cd() function. This function takes the users inputs, creates a dictionary, and appends it to the list of dictionaries used as local memory storage for the database of entries. However, one of the users' inputs needs to be converted to a integer, this provides room for errors if the user inputs something like 'four' instead of '4' which would return a 'ValueError' error and crash the program. Using structured error handling at the conversion of the string to integer for this error type allows the script to skip over this step and continue processing, and since this is where the users entries get added to the list of dictionaries used for storage, the users faulty entry gets ignored and they can then re attempt to add the entry. These types of error handlings are used throughout the script wherever an error is prone to occur in order to prevent a failure and crash.

```
class DataProcessor:
    """ Processes data during runtime """
    @staticmethod
    def add_cd(strID, strTitle, stArtist):
        """
        Adds CD to data structure (list of dicts)

        Args:
            strID (string): ID for new entry

                 strTitle (string): Title of CD to be added

                 stArtist (string): Artist of CD to be added

             Returns:
                 None.
             """
             try:
                 intID = int(strID)
                 dicRow = {'ID': intID, 'Title': strTitle, 'Artist': stArtist}
                 lstTbl.append(dicRow)

             except ValueError as e:
                 print('The ID must be an integer')
                 print('Built in error info:')
                 print(type(e), e, e.__doc__, sep='\n')
```

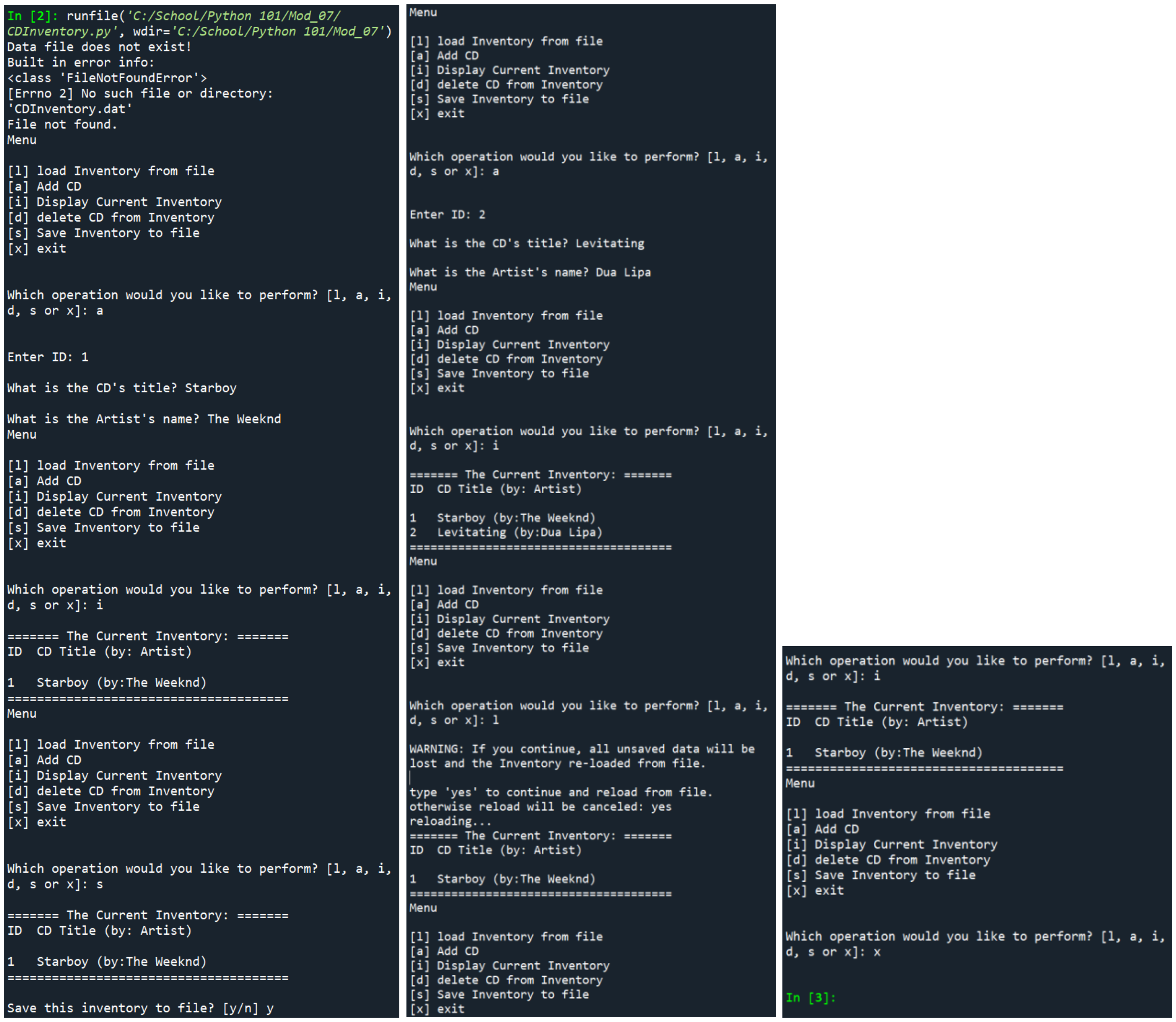
Figure 2 below shows the CDInventory.txt file with some pre-filled entries before the code is ran. An example of the output of this script can be seen below in in **Error! Reference source not found.** which is directly from the console of the Spyder IDE, going through every menu choice available as well as in which is directly from a terminal. **Error! Reference source not found.** shows the CDInventory.txt file after the program is finished running, note the capitalized 3rd row which can be seen being added in Figure 2. Figure 4 shows an example of the script running in a terminal.

Figure 1 shows how structured error handling works. Three is entered as user input for ID which cannot be converted to an integer, so structured error handling provides the user with some information on their error and continues to run the program. It can also be seen that the entry is not added to the memory as a result of this error, so it does not corrupt the database. Figure 2 shows an example output within the Spyder IDE. Figure 3 shows the contents of CDInventory.dat file after the run, because the file is written in binary it is not easily readable without manipulation. Figure 4



```
In [3]: runfile('C:/School/Python 101/Mod_07/
CDInventory.py', wdir='C:/School/Python 101/Mod_07')
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]: a


Enter ID: three

What is the CD's title? Starboy

What is the Artist's name? The Weeknd
The ID must be an integer
Built in error info:
<class 'ValueError'>
invalid literal for int() with base 10: 'three'
Inappropriate argument value (of correct type).
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

======= The Current Inventory: =======
ID  CD Title (by: Artist)

======================================
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]:
```

Figure 1 -Example of structured error handling.

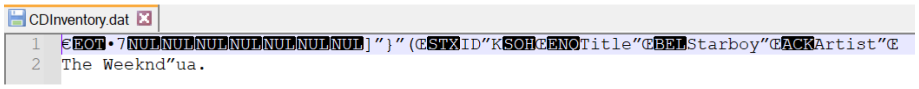


```
In [2]: runfile('C:/School/Python 101/Mod_07/
CDInventory.py', wdir='C:/School/Python 101/Mod_07')
Data file does not exist!
Built in error info:
<class 'FileNotFoundError'>
[Errno 2] No such file or directory:
'CDInventory.dat'
File not found.
Menu

[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]: a


Enter ID: 1

What is the CD's title? Starboy

What is the Artist's name? The Weeknd
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

======= The Current Inventory: =======
ID   CD Title (by: Artist)

1   Starboy (by:The Weeknd)
======================================
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]: s

======= The Current Inventory: =======
ID   CD Title (by: Artist)

1   Starboy (by:The Weeknd)
======================================

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

```
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]: a


Enter ID: 2

What is the CD's title? Levitating

What is the Artist's name? Dua Lipa
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

======= The Current Inventory: =======
ID   CD Title (by: Artist)

1   Starboy (by:The Weeknd)
2   Levitating (by:Dua Lipa)
======================================
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]: l

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

type 'yes' to continue and reload from file.
otherwise reload will be canceled: yes
reloading...
======= The Current Inventory: =======
ID   CD Title (by: Artist)

1   Starboy (by:The Weeknd)
======================================
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

======= The Current Inventory: =======
ID   CD Title (by: Artist)

1   Starboy (by:The Weeknd)
======================================
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 [3]:
```

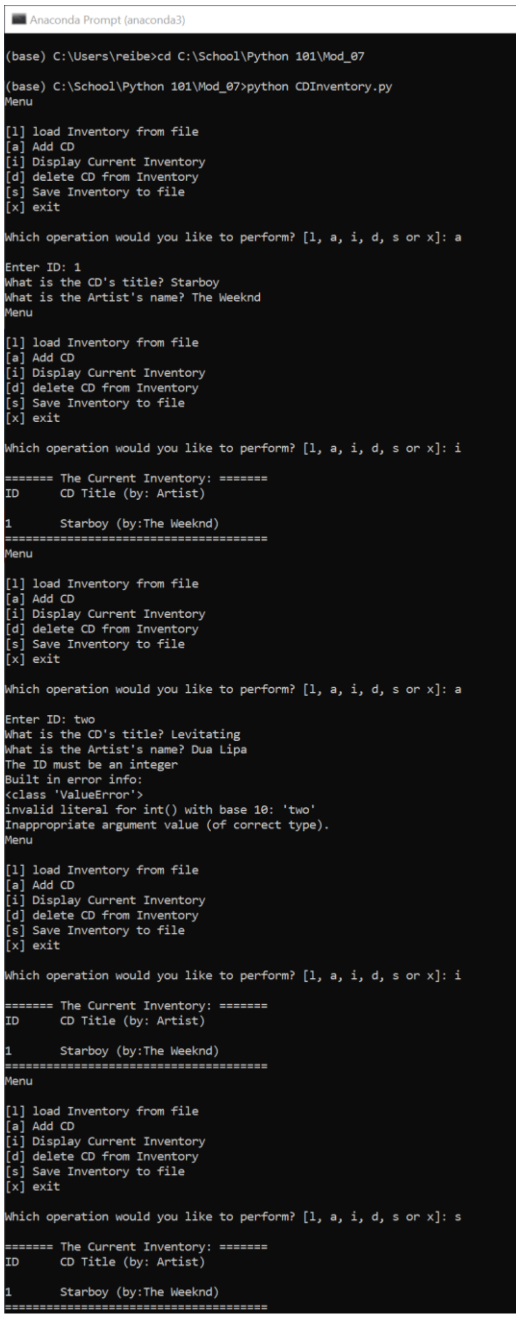
Figure 2 – Example output from working script in Spyder IDE



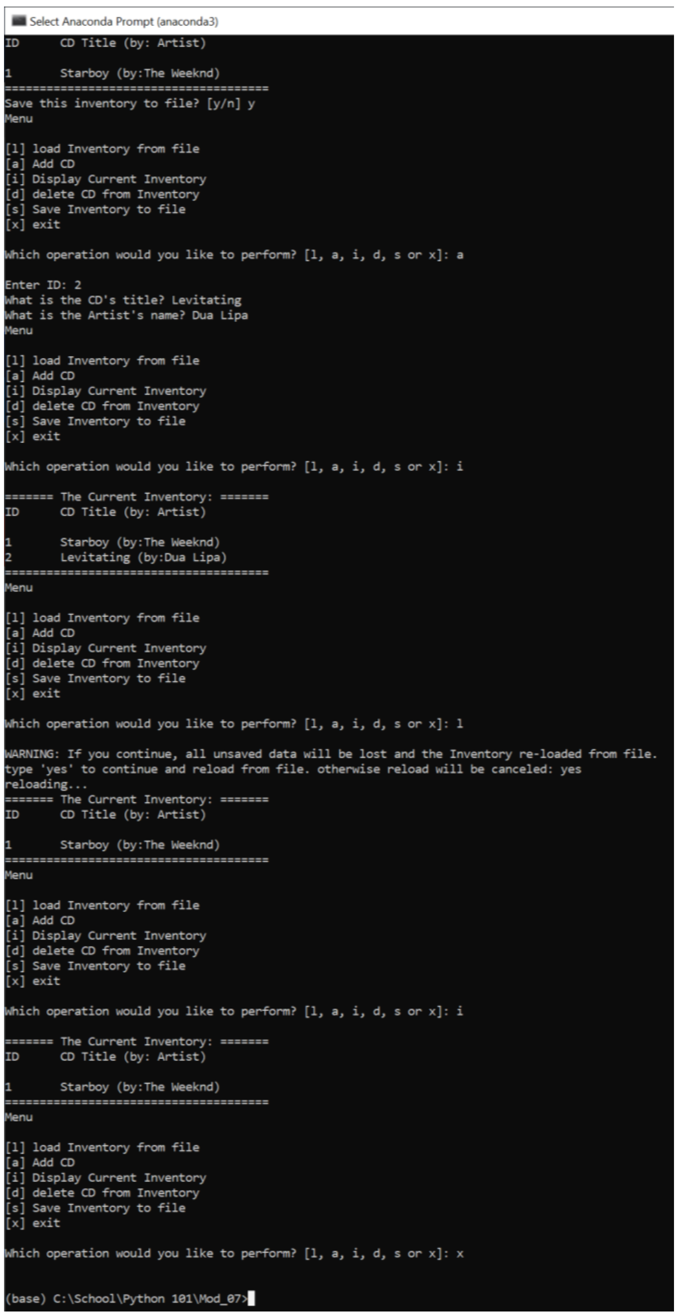
CDInventory.dat

```
€EOT•7NULNULNULNULNULNULNUL]”}”(ŒSTXID”KSOHŒENQTitle”ŒBELStarboy”ŒACKArtist”Œ
The Weeknd”ua.
```

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



```
(base) C:\Users\reibe>cd C:\School\Python 101\Mod_07

(base) C:\School\Python 101\Mod_07>python CDInventory.py
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]: a

Enter ID: 1
What is the CD's title? Starboy
What is the Artist's name? The Weeknd
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

======= The Current Inventory: =======
ID      CD Title (by: Artist)

1       Starboy (by:The Weeknd)
======================================
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]: a

Enter ID: two
What is the CD's title? Levitating
What is the Artist's name? Dua Lipa
The ID must be an integer
Built in error info:
<class 'ValueError'>
invalid literal for int() with base 10: 'two'
Inappropriate argument value (of correct type).
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

======= The Current Inventory: =======
ID      CD Title (by: Artist)

1       Starboy (by:The Weeknd)
======================================
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]: s

======= The Current Inventory: =======
ID      CD Title (by: Artist)

1       Starboy (by:The Weeknd)
======================================
```



```
ID      CD Title (by: Artist)

1       Starboy (by:The Weeknd)
======================================
Save this inventory to file? [y/n] y
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]: a

Enter ID: 2
What is the CD's title? Levitating
What is the Artist's name? Dua Lipa
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

======= The Current Inventory: =======
ID      CD Title (by: Artist)

1       Starboy (by:The Weeknd)
2       Levitating (by:Dua Lipa)
======================================
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]: l

WARNING: If you continue, all unsaved data will be lost and the Inventory re-loaded from file.
type 'yes' to continue and reload from file. otherwise reload will be canceled: yes
reloading...
======= The Current Inventory: =======
ID      CD Title (by: Artist)

1       Starboy (by:The Weeknd)
======================================
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

======= The Current Inventory: =======
ID      CD Title (by: Artist)

1       Starboy (by:The Weeknd)
======================================
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


(base) C:\School\Python 101\Mod_07>
```

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

Summary

Throughout this module, we focused on adding functionality to improve the experience of both the programmer and the user. The error handling prevents a program from crashing, and pickling allows the programmer to handle file management easier. One thing I had to grapple with when implementing pickling is how to check the program since the binary data files are not readable, making it harder to debug. In order to ensure everything was working appropriately I had to make intermediate variables to serve as test points, which I appreciate as it gave me an opportunity to learn more methods to debug my code.