Jennifer Magat

August 22, 2021

Foundations of Programming (Python)

Assignment_07

Applying Error Handling and Storing data as binary data in a script

Introduction

For the 7th Module, we learned about the benefits of exception handling functions, pickling, and storing data as binary file. We were also tasked with viewing other websites (and videos) further explaining the concepts. The purpose of this document is to explore some of those learnings and apply those in a script.

Exception Handling

An Error might indicate critical problems that a reasonable application should not try to catch, while an Exception might indicate conditions that an application should try to catch.[1] A very informative website that discusses exception handling in detail is the Programiz website <https://www.programiz.com/python-programming/exception-handling>. It provided a tutorial covering how to handle exceptions in Python using try, except and finally statements. It included screenshots of the code and the output results that were readable and easy to follow. In addition, they also had a video that explains the concept very well. Another great website is the Tutorials Teacher website <https://www.tutorialsteacher.com/python/exception-handling-in-python> which also explained Exception Handling in clear and simple language that is easy to follow. The example code and output results were also very helpful.

Pickling

"Pickling" is the process whereby a Python object hierarchy is converted into a byte stream, and *"unpickling"* is the inverse operation, whereby a byte stream (from a binary file or bytes-like object) is converted back into an object hierarchy. [2] The Python Library <https://docs.python.org/3/library/pickle.html> is a good website to use as a reference because it contains the official definitions for Python. Another great website is the Real Python website <https://realpython.com/python-pickle-module/> which covered the 3 built in modules in Python for serialization: marshall, json, and pickle. What's great about this website is it covered the pros and cons of each modules and provided a recommendation of what is most straight forward to use which is the pickle module. This website also had a video covering the pickle module which was very informative and easy to follow.

Modifying the CDInventory.py script.

For this assignment, the CDInventory.py script from last week is used a starting to implement changes that included the inclusion of exceptions and storing of data as binary data.

---

[1] https://www.datacamp.com/community/tutorials/exception-handling-python, retrieved 2021-Aug-22

[2] https://docs.python.org/3/library/pickle.html , retrieved 2021-Aug-22

Under the class DataProcessor, I updated the existing code for the cd_addition defined function to include exception handling by applying the try and except statements. Figure 1 below the modified cd_addition function.

```
23  class DataProcessor:
24      # TODOne add functions for processing here
25
26      @staticmethod
27      def cd_addition(strID, strTitle, strArtist, table):
28
29          """Function to manage user input and add to the dictionary list
45          try:
46              intID = int(strID)
47              dicRow = {'ID': intID, 'Title': strTitle, 'Artist': strArtist}
48              table.append(dicRow)
49          except ValueError as e:
50              print('That is not valid CD ID!')
51              print('Build in error info:')
52              print(type(e), e, e.__doc__, sep='\n')
53          except Exception as e:
54              print('There was a general error')
55              print('Build in error info:')
56              print(type(e), e, e.__doc__, sep='\n')
57
```

Figure 1 –modified cd_addition function with exception handling

Next the I updated the read_file defined function that was also under the DataProcessor class. Previously, this code was reading a text data and now has been modified to read a binary file using the pickling module and applying the pickle.load function. In addition, I also added exception handling in this code using the try and except functions. The exception is useful to provide the user a message if the .dat is not found or a general error has occurred. Figure 2 below displays updated code reading a .dat file into the program.

```
93      def read_file(file_name, table): ## Previously this was reading in a txt file, converted to .dat file.
94          """Function to read binary data file
108         ## Loads binary data
109         try:
110             table.clear()    # this clears existing data and allows to load data from file
111             with open(file_name, 'rb') as objFile:
112                 data = pickle.load(objFile) # Note: load () loads one line of data
113             return data
114             for line in data:
115                 table.append(line)
116             objFile.close()
117         ## Added Error handling
118         except FileNotFoundError as e:
119             print('Text file does not exist')
120             print('Build in error info:')
121             print(type(e), e, e.__doc__, sep='\n')
122         except Exception as e:
123             print('There was a general error')
124             print('Build in error info:')
125             print(type(e), e, e.__doc__, sep='\n')
126
```

Figure 2 – modified read_file function to read .dat file with exception handling

I also updated the write_file defined function under the DataProcessor class. Previously, this code was saving the data as a text file and now has been modified to save the binary data using the pickling module and applying the pickle.dump function. In addition, I also added exception handling in this code using the try and except statements. The exception is useful to provide the user a message if an error is encountered while saving the file. Figure 3 below displays updated code writing a .dat file as an output.

```
129     def write_file(file_name, table): ## Previously being saved as a txt file, now saving as binary file.
130         # TODOne Add code here
131         """Function to save current CD Inventory list as binary data
145         ## Save binary data
146         try:
147             with open(file_name, 'wb') as objFile:
148                 pickle.dump(table, objFile)
149         ## Added Error Handling
150         except FileNotFoundError as e:
151             print('Text file does not exist')
152             print('Build in error info:')
153             print(type(e), e, e._doc_, sep='\n')
154         except Exception as e:
155             print('There was a general error')
156             print('Build in error info:')
157             print(type(e), e, e._doc_, sep='\n')
158
```

Figure 3 – modified write_file function to write the data as .dat file with exception handling

Another section where I included error handling is the cd_data defined function where the user provides information as a CD entry to add into the inventory. Exception handling is best applied in where there is data being collected from the user. Figure 4 below displays the modified code using the try and except statements to add exception handling.

```
217     @staticmethod
218     def cd_data():
219
220         """Function to collect CD Data from the user: CD ID, Album, Artist
235         ## Add Error handling
236         try:
237             strID = input('Enter ID: ').strip()
238             strTitle = input('What is the CD\'s title? ').strip()
239             strArtist = input('What is the Artist\'s name? ').strip()
240             return strID, strTitle, strArtist
241         except ValueError as e:
242             print('That is not valid CD ID!')
243             print('Build in error info:')
244             print(type(e), e, e._doc_, sep='\n')
245         except Exception as e:
246             print('There was a general error')
247             print('Build in error info:')
248             print(type(e), e, e._doc_, sep='\n')
```

Figure 4 – modified cd_data function to collect CD data with exception handling

Lastly, I modified the code for Task 3.5.1.2 where the user has the option of entering a CD ID from the inventory. Again, this is a good section to apply the try and except statement for exception handling since this section requires user input or interaction. Figure 5 below displays the modified code for receiving user input for task 3.5.1.2.

```
295         # 3.5.1.2 ask user which ID to remove
296         ## Added Error Handling
297         try:
298             intIDDel = int(input('Which ID would you like to delete? ').strip())
299         except ValueError as e:
300             print('That is not valid CD ID!')
301             print('Build in error info:')
302             print(type(e), e, e._doc_, sep='\n')
303         except Exception as e:
304             print('There was a general error')
305             print('Build in error info:')
306             print(type(e), e, e._doc_, sep='\n')
307
```

Figure 5 – modified code for receiving user input for CD ID to delete with exception handling

Challenges

One of the areas I struggled with was in reading in the original CDInvetory.txt file. The issue was that I modified that section of the code already to use the pickle module to read a binary file. Figure 6 below displays the errors I was getting.

```
In [90]: runfile('C:/_FDProgramming/Mod_07/CDInventory_JM3.py', wdir='C:/_FDProgramming/Mod_07')
There was a general error
Build in error info:
Traceback (most recent call last):

  File "C:\_FDProgramming\Mod_07\CDInventory_JM3.py", line 110, in read_file
    table.clear()    # this clears existing data and allows to load data from file

NameError: name 'table' is not defined


During handling of the above exception, another exception occurred:

Traceback (most recent call last):

  File "C:\_FDProgramming\Mod_07\CDInventory_JM3.py", line 249, in <module>
    FileProcessor.read_file(strFileName)

  File "C:\_FDProgramming\Mod_07\CDInventory_JM3.py", line 123, in read_file
    print(type(e), e, e._doc_, sep='\n')

AttributeError: 'NameError' object has no attribute '_doc_'
```

Figure 6 – Error read_file

Eventually I figured that I just needed a .dat file program needed to access. So I just made sure to first save the inventory as a .dat file and once I re-ran the program, the code was able access the CDInventory.dat file accordingly. I also added a message saying that the file was read successfully once read_file function is completed without errors. Figure 7 below displays the updated code reflecting the printed message.

```
250    # 1. When program starts, read in the currently saved Inventory
251    FileProcessor.read_file(strFileName, lstTbl)
252    print('Successfully read CDInventory.dat file')
```

Figure 7 – Print message when the file is read into the program successfully.

Running the Python Script

After saving the file, I ran the CDInventory script in both Spyder and the Terminal. Figures 8 and 9 below display the script working on the computer ensuring that all options are running correctly.

```
In [2]: runfile('C:/_FDProgramming/Mod_07/Assignment07/CDInventory.py', wdir='C:/_FDProgramming/
Mod_07/Assignment07')
Successfully read CDInventory.dat file
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? The Big Wheel

What is the Artist's name? Runrig
======= The Current Inventory: =======
ID  CD Title (by: Artist)

1   The Big Wheel (by:Runrig)
======================================
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
```

Figure 8 – Running CDInventory.py in Spyder (Succesfully read .dat file when it first opened and Add CD)

```
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   The Big Wheel (by:Runrig)
=======================================

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]: x


In [3]:
```

Figure 8a – Running CDInventory.py in Spyder – continued (Save file and exit)

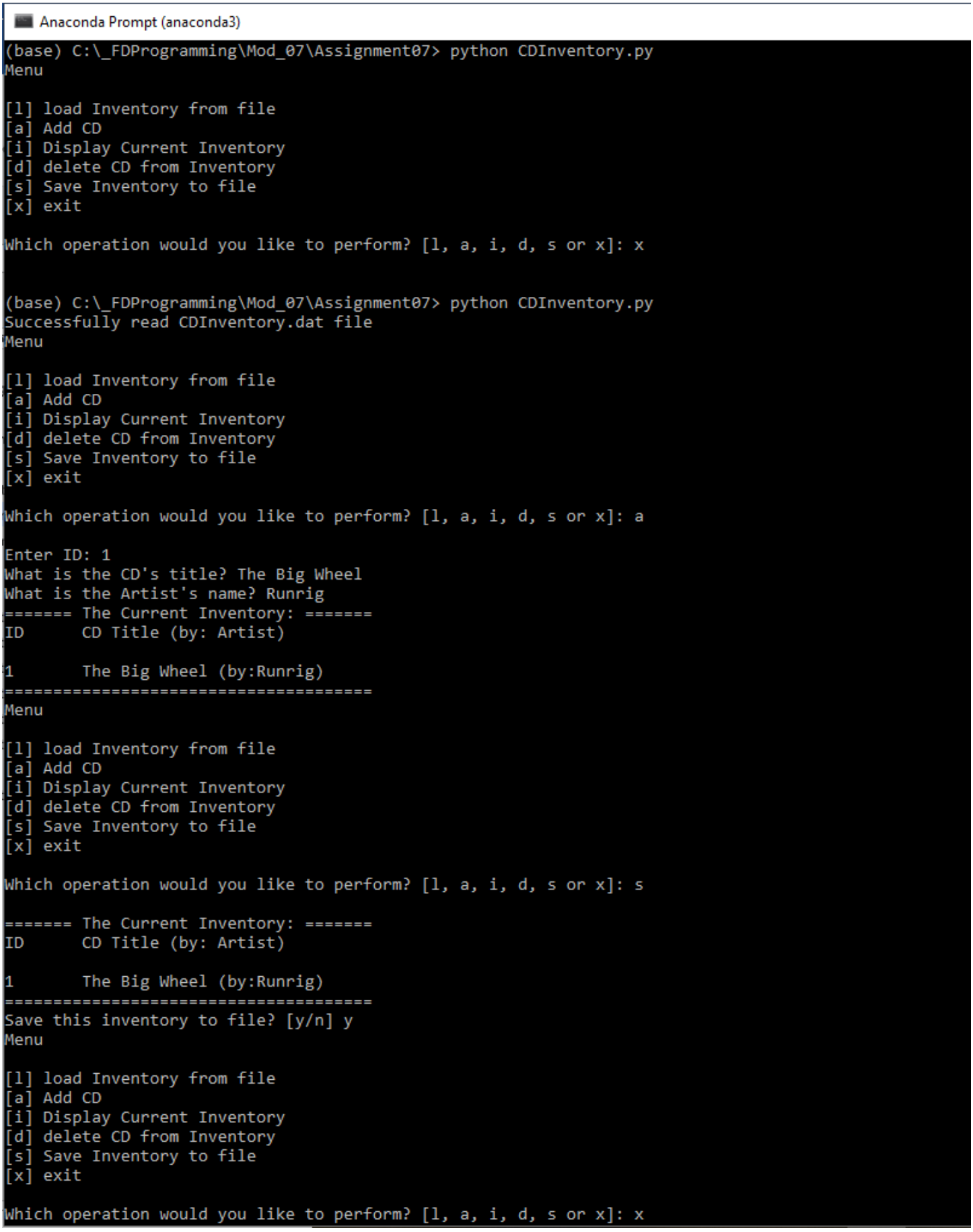

Figure 9 – Running CDInventory.py in Anaconda

GitHub

The assignment files including CDInventory.py, CDInventory.dat, and the Know_Doc_07.pdf have been upload to GitHub - https://github.com/jlmagat/Assignment_07

Summary

For this assignment, I explored various websites that had information related to exception handling and the pickle module. It was great to see different approaches telling a similar story. I prefer python tutorial websites that also include videos as it is helpful to see in action how the instructor is implementing the code and the resulting output. I also modified the existing CDInventory.py code to include code for handling exceptions and using the pickle module for accessing and storing binary data. The assignments continue to be more challenging each module and I remain excited with learning to code in python.

Appendix

Listing CDInventory.py – part 1

```
#------------------------------------------#
# Title: CDInventory.py
# Desc: Working with classes and functions.
# Change Log: (Who, When, What)
# DBiesinger, 2030-Jan-01, Created File
# JMagat, 2021-Aug-15, Modified file to use functions
#  moving code from the main loop to the class as functions
#  cd_addition, cd_deletion, write_file and cd_data
# JMagat, 2021-Aug-21, Modified file to add error handling and use of binary data for storage
#------------------------------------------#

import pickle

# -- DATA -- #
strChoice = '' # User input
lstTbl = []  # list of lists to hold data
dicRow = {}  # list of data row
strFileName = 'CDInventory.dat'  # data storage file, previously a txt file
objFile = None  # file object


# -- PROCESSING -- #
class DataProcessor:
    # TODOne add functions for processing here

    @staticmethod
    def cd_addition(strID, strTitle, strArtist, table):

        """Function to manage user input and add to the dictionary list

        Processes user data and formats it into a 2D table
        (list of dicts) table one line in the file represents one dictionary row in table.
        User data of CD information is collected and added as a row in the dictionary list.
        The table is appended to include the additional row in the dictionary list.

        Args:
            ID (string): this is the CD ID entered by the user
            Title (string): this is the CD's title
            Artist (string): this is the Artist of the CD
            table (list of dict): 2D data structure (list of dicts) that holds the data during runtime

        Returns:
            None.
        """
        try:
            intID = int(strID)
            dicRow = {'ID': intID, 'Title': strTitle, 'Artist': strArtist}
            table.append(dicRow)
        except ValueError as e:
            print('That is not valid CD ID!')
            print('Build in error info:')
            print(type(e), e, e.__doc__, sep='\n')
```

Listing CDInventory.py – part 2

```
            print(type(e), e, e.__doc__, sep='\n')
        except Exception as e:
            print('There was a general error')
            print('Build in error info:')
            print(type(e), e, e.__doc__, sep='\n')


    @staticmethod
    def cd_deletion(table):

        """Function to manage user input and delete items from the dictionary list

        Processes user data that identifies the CD to be deleted and deletes from the  2D table
        (list of dicts). The user identifies the CD ID to be deleted, and all data associated with the ID
        is deleted.

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

        Returns:
            None.
        """
        intRowNr = -1
        blnCDRemoved = False
        for row in lstTbl:
            intRowNr += 1
            if row['ID'] == intIDDel:
                del lstTbl[intRowNr]
                blnCDRemoved = True
                break
        if blnCDRemoved:
            print('The CD was removed')
        else:
            print('Could not find this CD!')


class FileProcessor:
    """Processing the data to and from text file"""

    @staticmethod
    def read_file(file_name, table): ## Previously this was reading in a txt file, converted to .dat file.
        """Function to read binary data file

        Args:
            file_name (string): name of file used to read the data from
            table (list of dict): 2D data structure (list of dicts) that holds the data during runtimeable

        Returns:
            data (Binary data file): Creates binary data file

        Raises:
```

Listing CDInventory.py – part 3

```
        Raises:
            FileNotFoundError: Text file does not exist
            Exception: Any exception

        """
        ## Loads binary data
        try:
            table.clear()    # this clears existing data and allows to load data from file
            with open(file_name, 'rb') as objFile:
                data = pickle.load(objFile) # Note: load () loads one line of data
            return data
            for line in data:
                table.append(line)
            objFile.close()
        ## Added Error handling
        except FileNotFoundError as e:
            print('Text file does not exist')
            print('Build in error info:')
            print(type(e), e, e.__doc__, sep='\n')
        except Exception as e:
            print('There was a general error')
            print('Build in error info:')
            print(type(e), e, e.__doc__, sep='\n')


    @staticmethod
    def write_file(file_name, table): ## Previously being saved as a txt file, now saving as binary file.
        # TODOne Add code here
        """Function to save current CD Inventory list as binary data

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

        Returns:
            None.

        Raises:
            FileNotFoundError: Text file does not exist
            Exception: Any exception

        """
        ## Save binary data
        try:
            with open(file_name, 'wb') as objFile:
                pickle.dump(table, objFile)
        ## Added Error Handling
        except FileNotFoundError as e:
            print('Text file does not exist')
            print('Build in error info:')
            print(type(e), e, e.__doc__, sep='\n')
        except Exception as e:
```

Listing CDInventory.py – part 4

```
        except Exception as e:
            print('There was a general error')
            print('Build in error info:')
            print(type(e), e, e.__doc__, sep='\n')


# -- 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('Menu\n\n[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\n')

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

        Args:
            None.

        Returns:
            choice (string): a lower 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']:
            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:
```

Listing CDInventory.py – part 5

```
205             Returns:
206                 None.
207 
208             """
209             print('======= The Current Inventory: =======')
210             print('ID\tCD Title (by: Artist)\n')
211             for row in table:
212                 print('{}\t{} (by:{})'.format(*row.values()))
213             print('======================================')
214 
215         # TODOne add I/O functions as needed
216 
217         @staticmethod
218         def cd_data():
219 
220             """Function to collect CD Data from the user: CD ID, Album, Artist
221 
222             Args:
223                 None.
224 
225             Returns:
226                 strID (string): this is the CD ID entered by the user
227                 strTitle (string): this is the CD's title
228                 strArtist (string): this is the Artist of the CD
229 
230             Raises:
231                 ValueError: When value entered is not a number
232                 Exception: Any exceptions
233 
234             """
235             ## Add Error handling
236             try:
237                 strID = input('Enter ID: ').strip()
238                 strTitle = input('What is the CD\'s title? ').strip()
239                 strArtist = input('What is the Artist\'s name? ').strip()
240                 return strID, strTitle, strArtist
241             except ValueError as e:
242                 print('That is not valid CD ID!')
243                 print('Build in error info:')
244                 print(type(e), e, e.__doc__, sep='\n')
245             except Exception as e:
246                 print('There was a general error')
247                 print('Build in error info:')
248                 print(type(e), e, e.__doc__, sep='\n')
249 
250 # 1. When program starts, read in the currently saved Inventory
251 FileProcessor.read_file(strFileName, lstTbl)
252 
253 # 2. start main loop
254 while True:
255     # 2.1 Display Menu to user and get choice
256     IO.print_menu()
```

Listing CDInventory.py – part 6

```
    IO.print_menu()
    strChoice = IO.menu_choice()

    # 3. Process menu selection
    # 3.1 process exit first
    if strChoice == 'x':
        break
    # 3.2 process load inventory
    if strChoice == 'l':
        print('WARNING: If you continue, all unsaved data will be lost and the Inventory re-loaded from file.')
        strYesNo = input('type \'yes\' to continue and reload from file. otherwise reload will be canceled')
        if strYesNo.lower() == 'yes':
            print('reloading...')
            FileProcessor.read_file(strFileName, lstTbl)
            IO.show_inventory(lstTbl)
        else:
            input('canceling... Inventory data NOT reloaded. Press [ENTER] to continue to the menu.')
            IO.show_inventory(lstTbl)
        continue  # start loop back at top.
    # 3.3 process add a CD
    elif strChoice == 'a':
        # 3.3.1 Ask user for new ID, CD Title and Artist
        # TODOne move IO code into function
        strID, strTitle, strArtist = IO.cd_data()

        # 3.3.2 Add item to the table
        # TODOne move processing code into function
        DataProcessor.cd_addition(strID, strTitle, strArtist, lstTbl)
        IO.show_inventory(lstTbl)
        continue  # start loop back at top.
    # 3.4 process display current inventory
    elif strChoice == 'i':
        IO.show_inventory(lstTbl)
        continue  # start loop back at top.
    # 3.5 process delete a CD
    elif strChoice == 'd':
        # 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
        ## Added Error Handling
        try:
            intIDDel = int(input('Which ID would you like to delete? ').strip())
        except ValueError as e:
            print('That is not valid CD ID!')
            print('Build in error info:')
            print(type(e), e, e.__doc__, sep='\n')
        except Exception as e:
            print('There was a general error')
            print('Build in error info:')
            print(type(e), e, e.__doc__, sep='\n')

```

Listing CDInventory.py – part 7

```
307
308
309             # 3.5.2 search thru table and delete CD
310             # TODOne move processing code into function
311             DataProcessor.cd_deletion(lstTbl)
312             IO.show_inventory(lstTbl)
313             continue  # start loop back at top.
314         # 3.6 process save inventory to file
315         elif strChoice == 's':
316             # 3.6.1 Display current inventory and ask user for confirmation to save
317             IO.show_inventory(lstTbl)
318             strYesNo = input('Save this inventory to file? [y/n] ').strip().lower()
319             # 3.6.2 Process choice
320             if strYesNo == 'y':
321                 # 3.6.2.1 save data
322                 # TODOne move processing code into function
323                 FileProcessor.write_file(strFileName, lstTbl)
324             else:
325                 input('The inventory was NOT saved to file. Press [ENTER] to return to the menu.')
326             continue  # start loop back at top.
327         # 3.7 catch-all should not be possible, as user choice gets vetted in IO, but to be save:
328         else:
329             print('General Error')
```