Josephine Gnanaraj
02\28\2021
Foundations Of Programming: Python
Assignment_07

Introduction
In this week's module we learnt working with text and binary files and cover structured error handling. Learnt about the Pickle class used for dealing with binary files, such as “dumping”, which is a memory dump into a binary file (.dat).

Working with text files

To Open a Text File in Python
To open a file, we need to use the built-in open function. The Python open file function returns a file object that contains methods and attributes to perform various operations for opening files in Python.

```
file_object = open("filename", "mode")
```

Here,
- **filename:** gives name of the file that the file object has opened.
- **mode:** attribute of a file object tells you which mode a file was opened in.

To Create a Text File in Python
With Python Write to File, we can create a .text files (guru99.txt) by using the code, we have demonstrated here:

```
f= open("Code99.txt","w+")
```

- We declared the variable f to open a file named guru99.txt. Open takes 2 arguments, the file that we want to open and a string that represents the kinds of permission or operation we want to do on the file
- Here, we used "w" letter in our argument, which indicates Python write to file and it will create a file if it does not exist in library
- Plus sign indicates both read and write for Python create file operation.

```
for i in range(10): f.write("This is line %d\r\n" % (i+1))
```

- We have a for loop that runs over a range of 10 numbers.
- Using the **write** function to enter data into the file.
- The output we want to iterate in the file is "this is line number", which we declare with Python write to text file function and then percent d (displays integer)
- So basically, we are putting in the line number that we are writing, then putting it in a carriage return and a new line character

```
f.close()
```

- This will close the instance of the file Code99.txt stored

To Append to a File in Python

We can also append/add a new text to the already existing file or a new file.

```
f=open("Code99.txt", "a+")
for i in range(2): f.write("Appended line %d\r\n" % (i+1))
```

if we could see a plus sign in the code, it indicates that it will create a new file if it does not exist.

To Read Files in Python

We can read a file in Python by calling .txt file in a "read mode"(r).

Working with Binary Files

A binary file is a computer file that is not a text file.

To write text a binary file, you must prefix the string with the character 'b' to tell Python that it is a binary string, so convert it into a sequence of bytes.

```
>>> fH = open('writtenInBinary.txt', 'wb')
>>> fH.write(b'hi')
2
>>> fH.close()

>>> fH = open('writtenInBinary.txt', 'rb')
>>> fH.read()
b'hi'
>>> fH.close()
```

What this 'b' in the beginning does is that it converts the string into a sequence of bytes. This produces the same result as using the built-in bytes method on the same string.

The second argument we supplied in the bytes function is the character encoding that the string will be written in. Encoding, is the process of transforming the string into a specialized format for efficient storage or transmission. In other words, encoding is the process of transforming content into sequence of bytes, which will ideally make sense again when it is decoded with the same encoding type with which it was encoded. Character encoding is used to represent the entire list of characters that belong in an encoding system.

Unicode covers almost every character there is. It contains over 128 thousand characters, covering 135 modern and historic scripts, as well as multiple symbol sets, as per Wikipedia. Unicode is the standard character set of Python and is denoted by utf-8.

Structured Error Handling

Error handling increases the robustness of your code, which guards against potential failures that would cause your program to exit in an uncontrolled fashion.

Errors cannot be handled, while Python exceptions can be handled at the run time. An error can be a `syntax` (parsing) error, while there can be many types of exceptions that could occur during the execution and are not unconditionally inoperable. 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. Errors are a form of an unchecked exception and are irrecoverable like an `OutOfMemoryError`, which a programmer should not try to handle. To recount a few :

- ZeroDivisionError: If we give our math program 0 as the second number.
- ValueError: if our input for the CD Inventory ID cannot be converted to an integer, or any other instance we are expecting an input we can convert to integer.
- FileNotFound: if we try to open a file with attribute r that does not (yet) exist.

Summary

In this assignment\module we worked with text and binary files and cover structured error handling. Learnt about the Pickle class used for dealing with binary files, such as “dumping”, which is a memory dump into a binary file (.dat). We continued creating functions to organize our code. We looked at using Structured Error Handling to make our programs behave in a desired way in case of Exceptions. We introduced creating custom Exception classes.

Appendix

The script below modified last week’s assignment, added code to catch errors around file operations and Add code to catch errors around the user input operations. Handle non-numeric inputs and value zero input for the number used as denominator separately. type casting (string to int) or file access operations.

CDInventory.py

```
#------------------------------------------#
# Title: CDInventory.py
# Desc: working with text and binary files and error handling.
# Change Log: (Who, When, What)
# DBiesinger, 2030-Jan-01, Created File
# Jgnanaraj, 2021-Feb-28, Modified File
#------------------------------------------#

import pickle

# -- DATA -- #
strChoice = '' # User input
lstTbl = []  # list of lists to hold data
dicRow = {}  # list of data row
```

```
15.  strFileName = 'CDInventory.dat'  # data storage file
16.  objFile = None  # file object
17.
18.
19.  # -- PROCESSING -- #
20.  class DataProcessor:
21.      # TODOne add functions for processing here
22.      @staticmethod
23.      def input_data_process(intID, cdTitle, cdArtist, lstTbl):
24.          """Function to add user input data to table
25.
26.          Reads the data from file identified by file_name into a 2
     D table
27.          (list of dicts) table one line in the file represents one dict
     ionary row in table.
28.
29.          Args:
30.              The ID, Title and Artist newly input by the user
31.
32.          Returns:
33.              None.
34.          """
35.          dicRow = {'ID': intID, 'Title': cdTitle, 'Artist': cdArtist}
36.          lstTbl.append(dicRow)
37.
38.      @staticmethod
39.      def delete_row(rowId, lstTbl):
40.          """Function to delete row from the inventory
41.
42.          Args:
43.              The ID of the row intended to be deleted
44.
45.          Returns:
46.              None.
```

```
        """
        intRowNr = -1
        blnCDRemoved = False
        for row in lstTbl:
            intRowNr += 1
            if row['ID'] == rowId:
                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):
        """Function to manage data ingestion from file to a list of dictionaries

        Reads the data from file identified by file_name into a 2D table
        (list of dicts) table one line in the file represents one dictionary row in table.

        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 runtime

```

```
76.         Returns:
77.             None.
78.         """
79.         table.clear()  # this clears existing data and allows to lo
    ad data from file
80.
81.         # catching errors which are empty file or file not found
82.         try:
83.             objFile = open(file_name, 'rb')
84.             row_line = []
85.
86.             data = pickle.load(objFile)
87.
88.             lstData = data.strip().split('\n')
89.
90.             for item in lstData:
91.                 row_line = item.strip().split(',')
92.                 dicRow = {'ID': int(row_line[0]), 'Title': row_line[1], '
    Artist': row_line[2]}
93.                 table.append(dicRow)
94.
95.             objFile.close()
96.
97.         except FileNotFoundError as e:
98.             print("The file does not exist.")
99.             print("Error: ")
100.            print(type(e),e,e.__doc__, sep="\n")
101.
102.        except ValueError as e:
103.            print("The file is empty.")
104.            print("Error: ")
105.            print(type(e),e,e.__doc__, sep="\n")
106.
107.
```

```
108.     def save_file(file_name, table):
109.         """Function to save the text file
110.
111.         Writes the data from a 2D table (list of dicts) into a long
     string and saved into a text file.
112.
113.         Args:
114.             file_name (string): name of file used to write the data
     to
115.             table (list of dict): 2D data structure (list of dicts) that
     holds the data during runtime
116.
117.         Returns:
118.             None.
119.         """
120.         objFile = open(file_name, 'wb')
121.         new_line = ""
122.
123.         for row in table:
124.             print(row)
125.             for item in row.values():
126.                 new_line = new_line + str(item) + ","
127.             new_line = new_line[:-1] + "\n"
128.
129.         pickle.dump(new_line, objFile)
130.         objFile.close()
131.         print("Data saved!")
132.
133.
134. # -- PRESENTATION (Input/Output) -- #
135.
136. class IO:
137.     """Handling Input / Output"""
138.
```

```
139.     @staticmethod
140.     def print_menu():
141.         """Displays a menu of choices to the user
142.
143.         Args:
144.             None.
145.
146.         Returns:
147.             None.
148.         """
149.
150.         print('Menu\n\n[l] load Inventory from file\n[a] Add CD\n
    [i] Display Current Inventory')
151.         print('[d] delete CD from Inventory\n[s] Save Inventory t
    o file\n[x] exit\n')
152.
153.     @staticmethod
154.     def menu_choice():
155.         """Gets user input for menu selection
156.
157.         Args:
158.             None.
159.
160.         Returns:
161.             choice (string): a lower case sting of the users input o
    ut of the choices l, a, i, d, s or x
162.
163.         """
164.         choice = ' '
165.         while choice not in ['l', 'a', 'i', 'd', 's', 'x']:
166.             choice = input('Which operation would you like to perf
    orm? [l, a, i, d, s or x]: ').lower().strip()
167.         print()  # Add extra space for layout
168.         return choice
```

```
169.
170.     @staticmethod
171.     def show_inventory(table):
172.         """Displays current inventory table
173.
174.
175.         Args:
176.             table (list of dict): 2D data structure (list of dicts) that
    holds the data during runtime.
177.
178.         Returns:
179.             None.
180.
181.         """
182.         print('======= The Current Inventory: =======')
183.         print('ID\tCD Title (by: Artist)\n')
184.         for row in table:
185.             print('{}\t{} (by:{})'.format(*row.values()))
186.         print('======================================
    =')
187.
188.     # TODOne add I/O functions as needed
189.     @staticmethod
190.     def ask_user_data():
191.         """Asks for user data
192.
193.         Args: None
194.         Returns: The ID, the CD Title and the Artist of the title
195.         """
196.
197.         # catching errors like entering non-numeric entries
198.         try:
199.             ID = int(input('Enter ID: ').strip())
200.             Title = input('What is the CD\'s title? ').strip()
```

```
            Artist = input('What is the Artist\'s name? ').strip()
            return ID, Title, Artist
        except ValueError as e:
            print("Only numbers allowed for ID")
            print("Error: ")
            print(type(e),e,e.__doc__, sep="\n")


# 1. When program starts, read in the currently saved Inventory
FileProcessor.read_file(strFileName, lstTbl)

# 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':
        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)
```

```
232.         else:
233.             input('canceling... Inventory data NOT reloaded. Pres
    s [ENTER] to continue to the menu.')
234.             IO.show_inventory(lstTbl)
235.         continue  # start loop back at top.
236.
237.     # 3.3 process add a CD
238.     elif strChoice == 'a':
239.
240.         # catching error when erroneous data was not passed fr
    om IO.ask_user_data()
241.         try:
242.             # 3.3.1 Ask user for new ID, CD Title and Artist
243.             # TODOne move IO code into function
244.             intID, strTitle, stArtist = IO.ask_user_data()
245.
246.         except TypeError as e:
247.             print("Error in data entry.")
248.             print("Error: ")
249.             print(type(e),e, sep="\n")
250.             continue
251.
252.         # 3.3.2 Add item to the table
253.         # TODOne move processing code into function
254.         DataProcessor.input_data_process(intID, strTitle, stArti
    st, lstTbl)
255.         IO.show_inventory(lstTbl)
256.         continue  # start loop back at top.
257.
258.     # 3.4 process display current inventory
259.     elif strChoice == 'i':
260.         IO.show_inventory(lstTbl)
261.         continue  # start loop back at top.
262.
```

```
263.     # 3.5 process delete a CD
264.     elif strChoice == 'd':
265.         # 3.5.1 get Userinput for which CD to delete
266.         # 3.5.1.1 display Inventory to user
267.         IO.show_inventory(lstTbl)
268.
269.         # catching error when non-
    numeric data is entered by user
270.         try:
271.             # 3.5.1.2 ask user which ID to remove
272.             intIDDel = int(input('Which ID would you like to delete
    ? ').strip())
273.             # 3.5.2 search thru table and delete CD
274.             # TODOne move processing code into function
275.             DataProcessor.delete_row(intIDDel, lstTbl)
276.         except ValueError as e:
277.             print("Only numbers are allowed!")
278.             print("Error: ")
279.             print(type(e),e,e.__doc__, sep="\n")
280.             continue
281.
282.         IO.show_inventory(lstTbl)
283.         continue  # start loop back at top.
284.
285.     # 3.6 process save inventory to file
286.     elif strChoice == 's':
287.         # 3.6.1 Display current inventory and ask user for confir
    mation to save
288.         IO.show_inventory(lstTbl)
289.         strYesNo = input('Save this inventory to file? [y/n] ').strip
    ().lower()
290.         # 3.6.2 Process choice
291.         if strYesNo == 'y':
292.             # 3.6.2.1 save data
```

```
293.             # TODOne move processing code into function
294.             FileProcessor.save_file(strFileName, lstTbl)
295.         else:
296.             input('The inventory was NOT saved to file. Press [E
    NTER] to return to the menu.')
297.         continue  # start loop back at top.
298.
299.     # 3.7 catch-
    all should not be possible, as user choice gets vetted in IO, but
    to be save:
300.     else:
301.         print('General Error')
```

Created a script and Compiled to verify output:

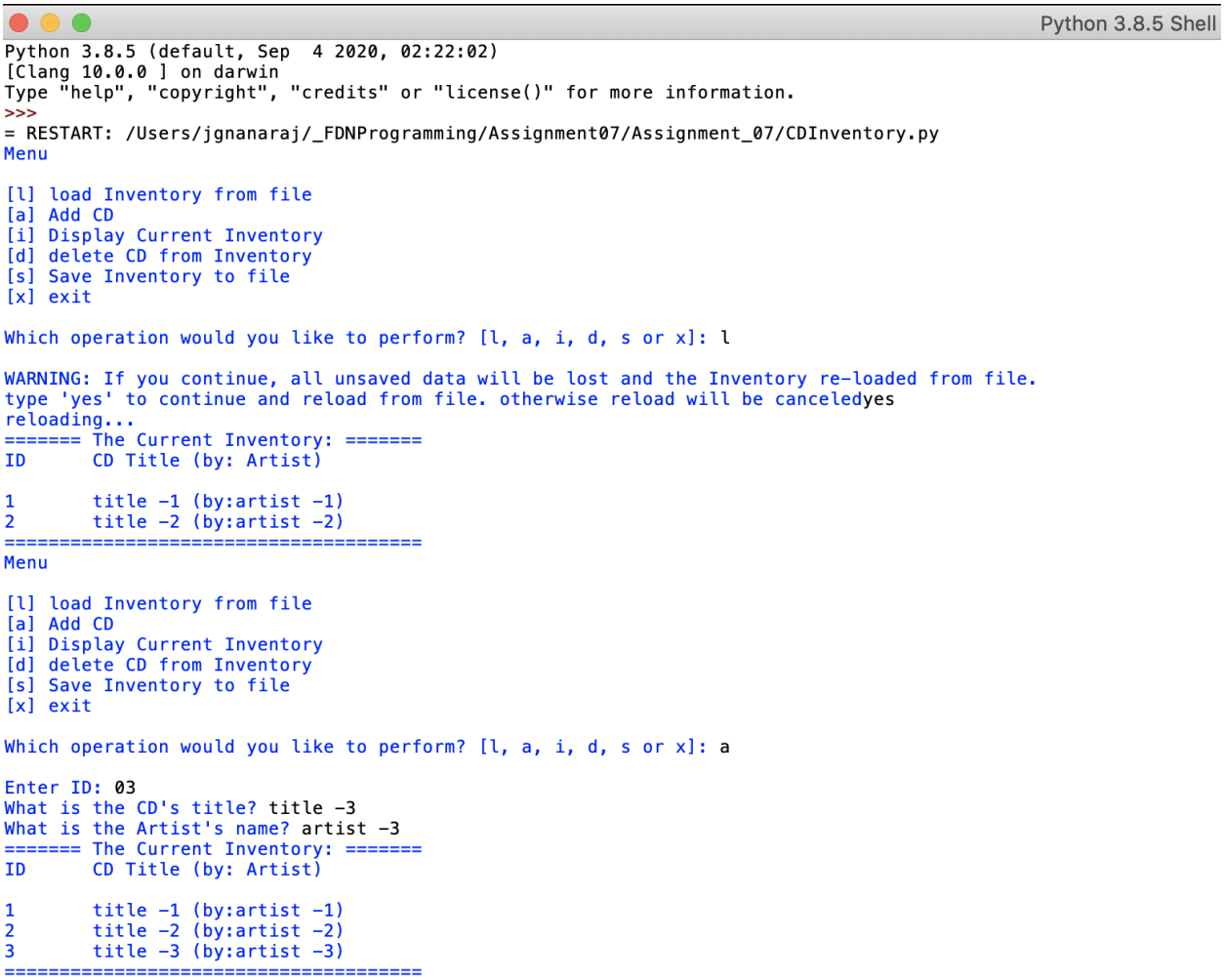

Figure 1 - Screen capture

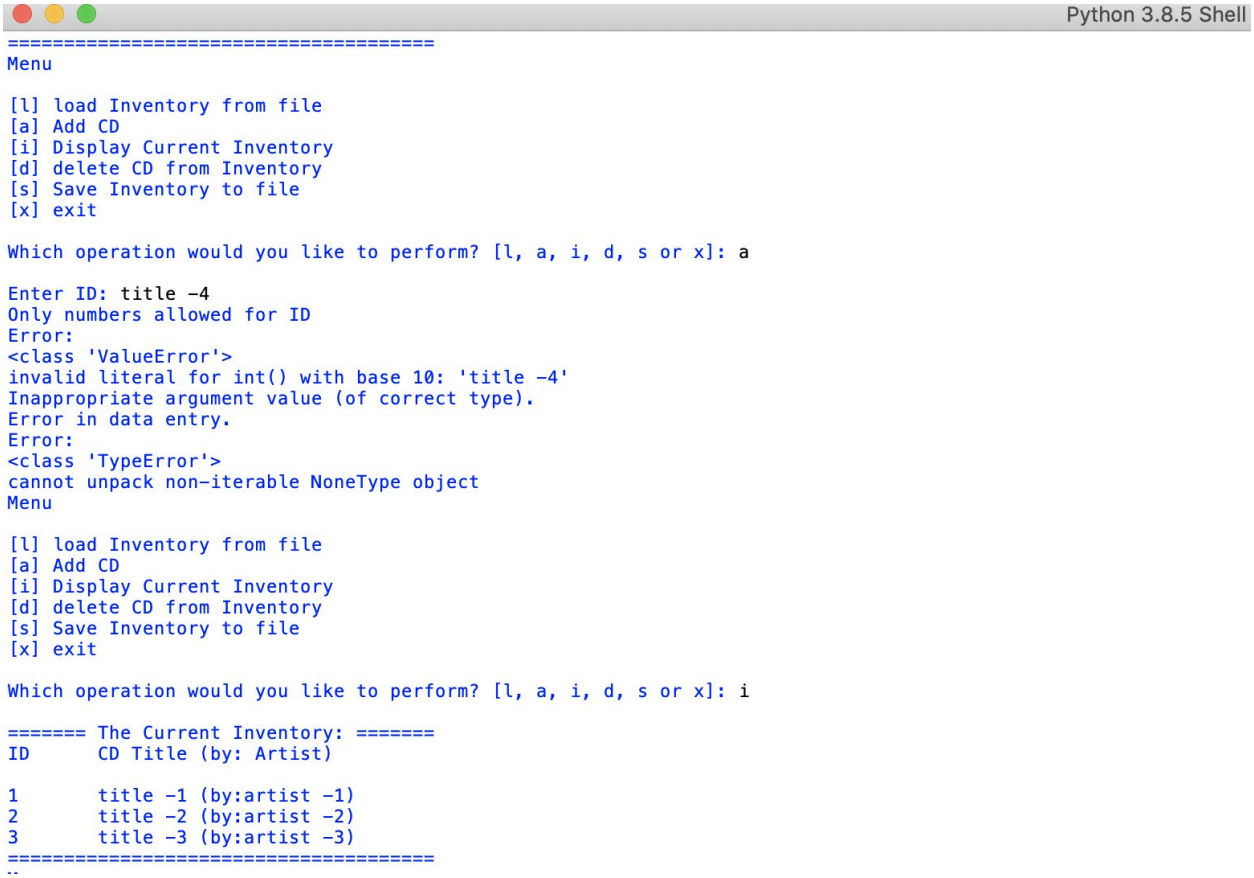

Figure 2 - Screen capture

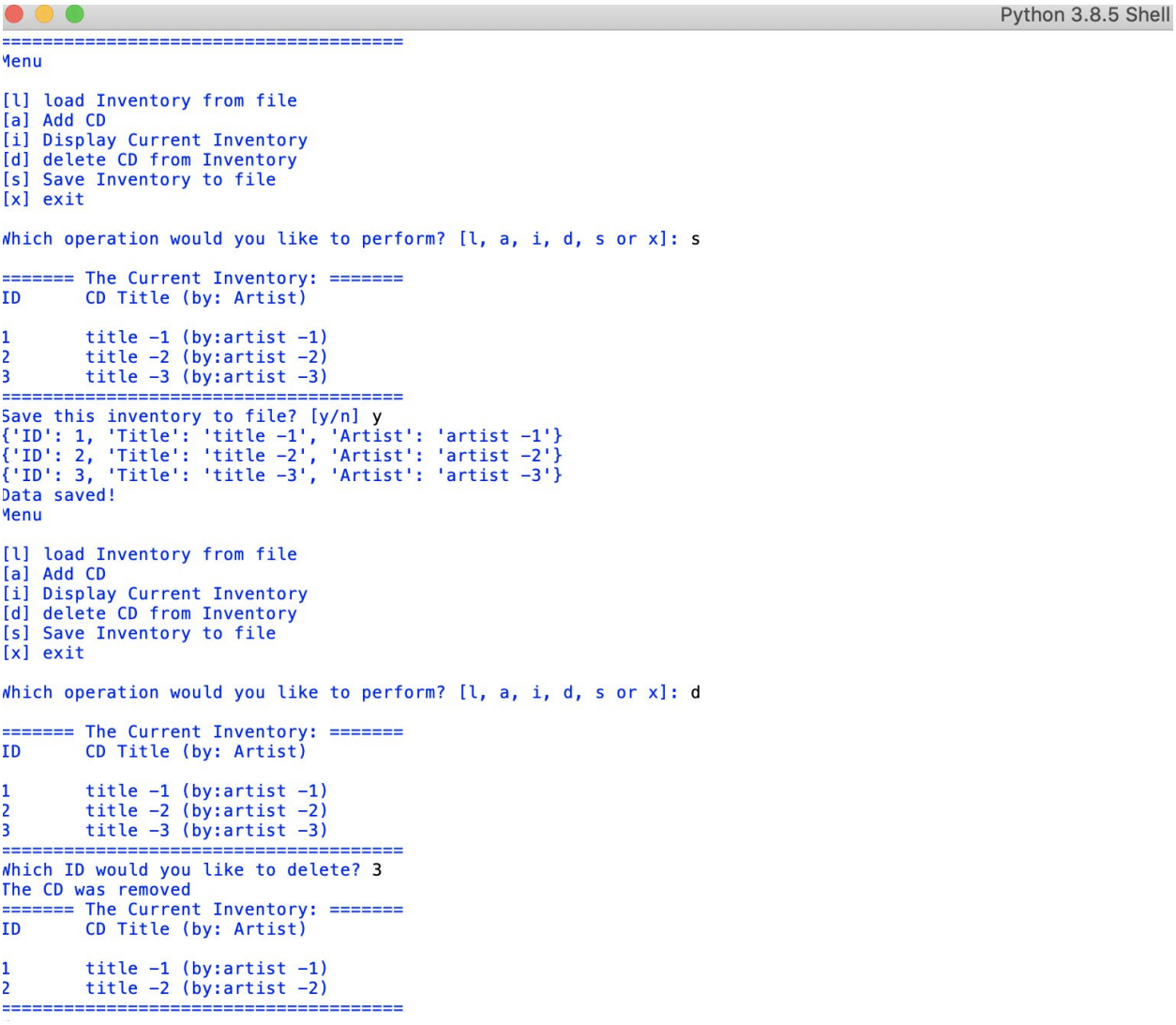

Figure 3 - Screen capture

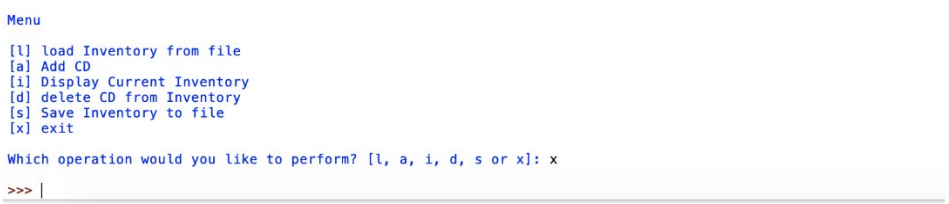

Figure 4 - Screen capture

References:

https://www.guru99.com/reading-and-writing-files-in-python.html
https://www.djangospin.com/working-binary-files-python/
https://www.datacamp.com/community/tutorials/exception-handling-python

FDN_Py_Module_07.pdf