Evan Malina

11/28/2021

Foundations of Programming Python

Assignment 7

# Introduction

For this assignment, I modified an existing script to add error handling for typecasting strings to integers, and for reading files that do not exist. Additionally, the script was modified to read/write binary rather than text. This assignment was mainly focused on binary (and pickles), and error handling.

# Code

## Github

Github link to my code and knowledge document here;

<https://github.com/ewmalina/Assignment_07>

## Binary

I modified the file name in the data section to be the .dat binary version

### Write

Pickle dump was not working when I tried to for loop. It wrote one line, but not the 2nd… Still not sure why, but I modified the code. Once I realized that this wasn’t working, pickle.dump made this work perfectly!

### Read

I was able to simply use pickle.load to add the binary data to a new table of dictionaries. This worked much much nicer!

Note that both the read and write sections were modified to swap the ‘r’ and ‘w’ to be ‘rb’ and ‘wb’

## Error Handling

### FileNotFound

I added try, except FileNotFoundError as e to the read section. This works nicely, and since it effectively displays an empty inventory after. I considered modifying the ‘rb’ to be some ‘rb+’ but think the script works fine as written. Figure 1 I did have to further modify this to return an empty table because the show inventory failed after the load\_inventory.

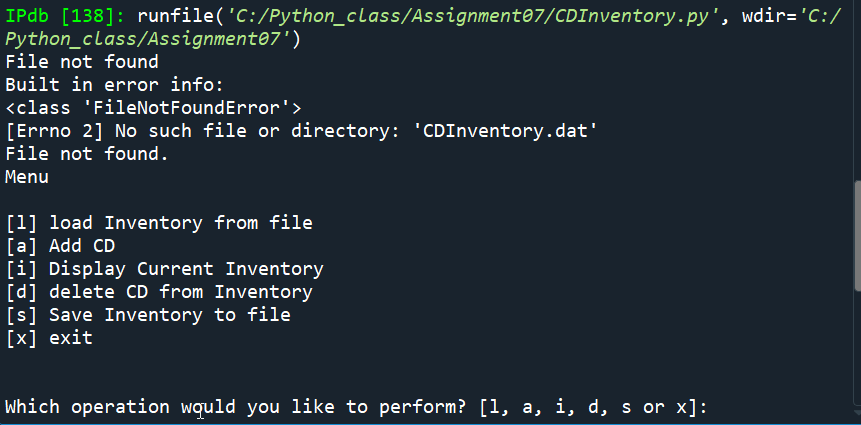


Figure - File not found error

### User Input

I am modifying this section to say I didn’t find any true user input locations that required error handling. However, see the typecasting section because it is similar 😊. However, I could’ve added a general error handling to prevent adding ID that exists

I did not modify the menu\_choice function. I tested with ‘true’, 1, 1.0 and nothing affected the While, if/elif/else loops so I don’t think this requires error handling.

I used this same rationale for the load inventory section, which simply has a confirmation string. See Figure 1.

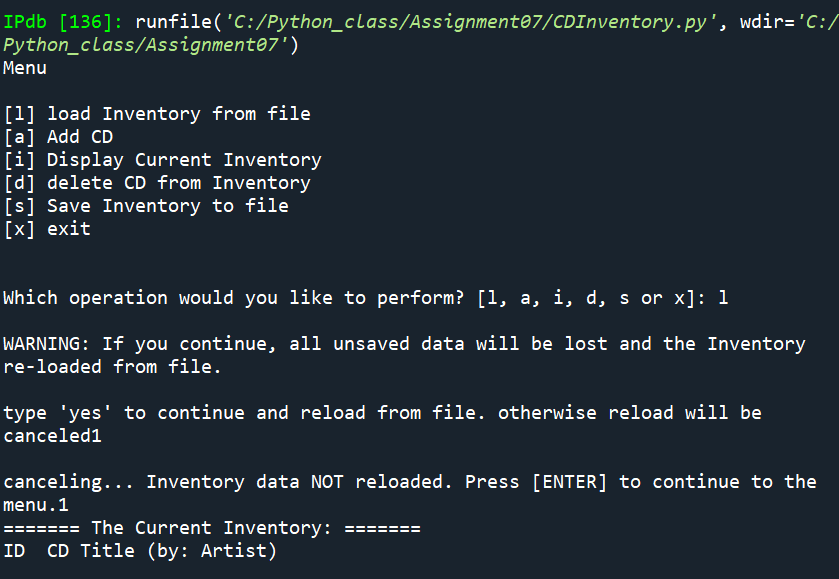


Figure – Load inventory confirmation, not requiring error handling

### Type Casting - Integers

The delete CD section worked great. I added try, except ValueError. In the try, if the input was indeed an integer, the break exited. Otherwise the while loop would return the error information for any non integer type. The same initial input for a ID integer continues to pop up indefinitely as inputs are non-integers.

I used the same exact code in the add\_data function for the ID number. This worked perfectly! See Figure 2

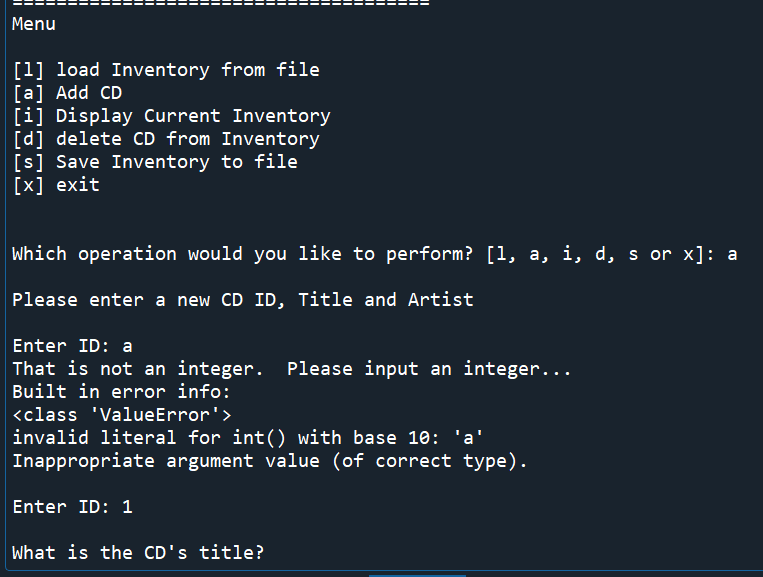


Figure - Integer ID add ValueError

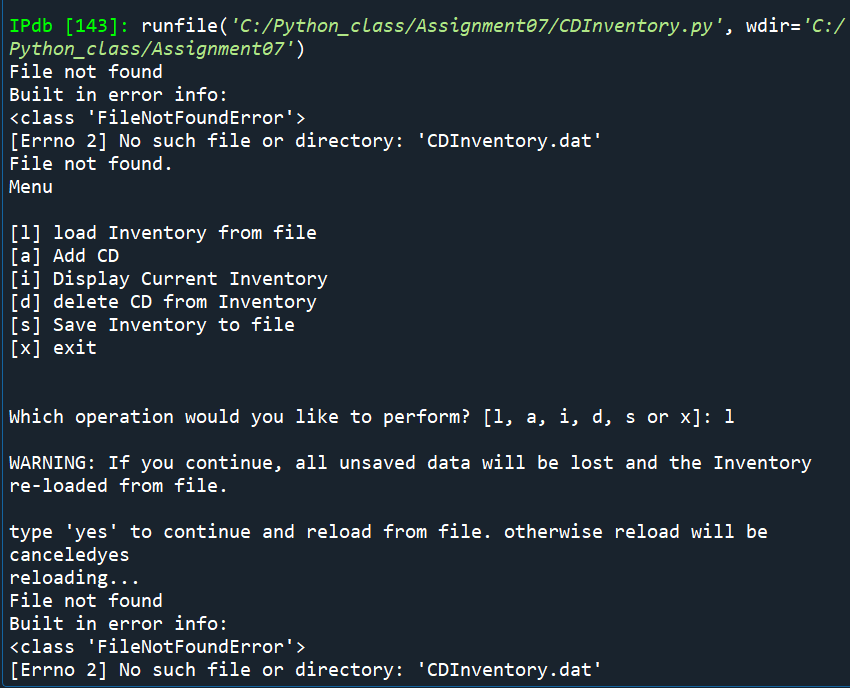


Figure 4 - Spyder Script 1

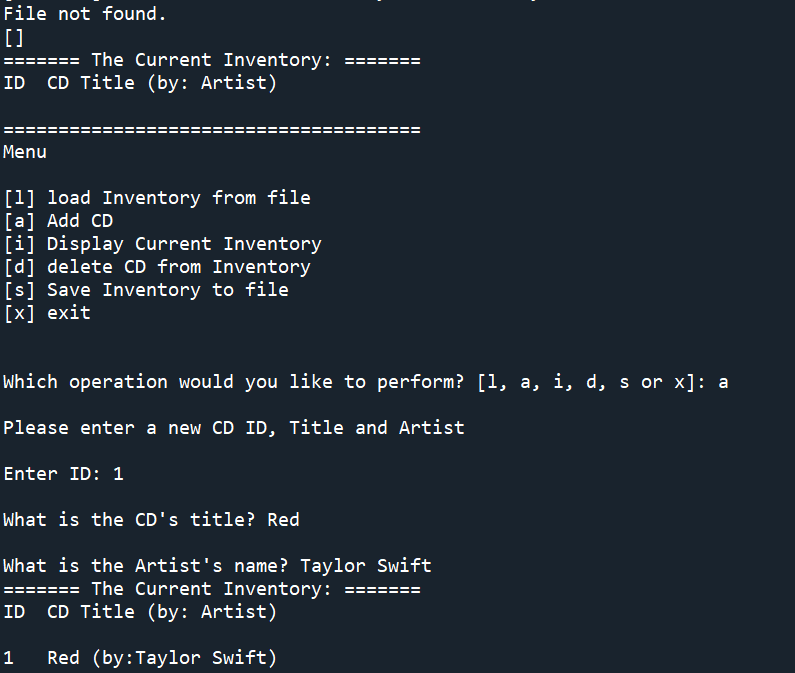


Figure 5 - Spyder Script 2

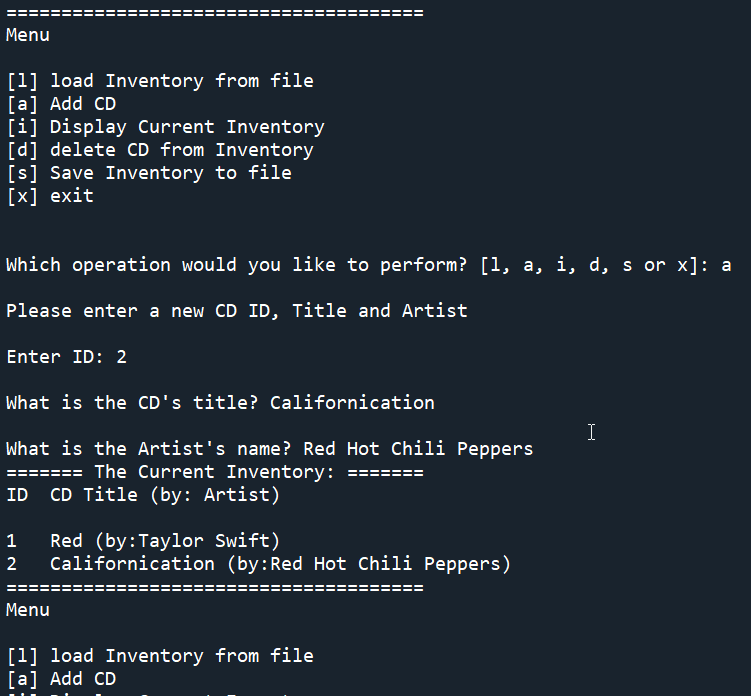


Figure - Spyder script run part 3

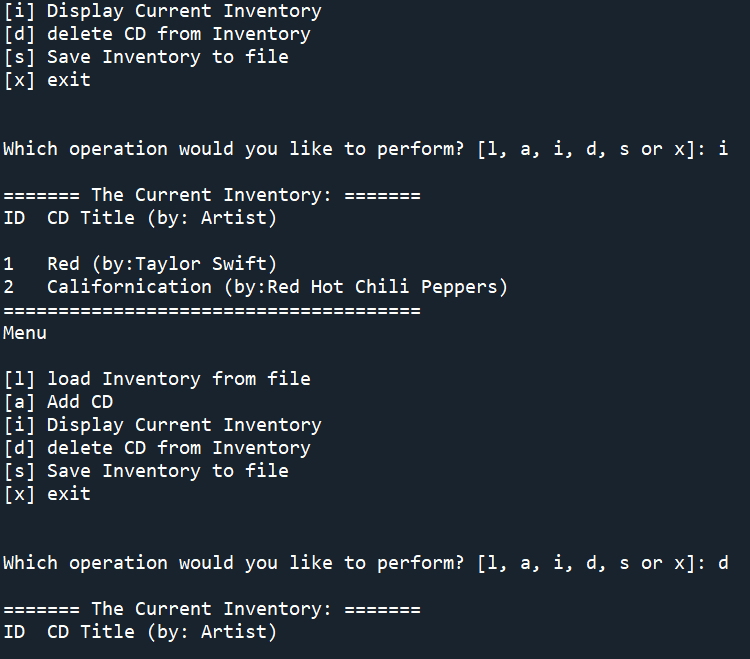


Figure - Spyder script run part 4

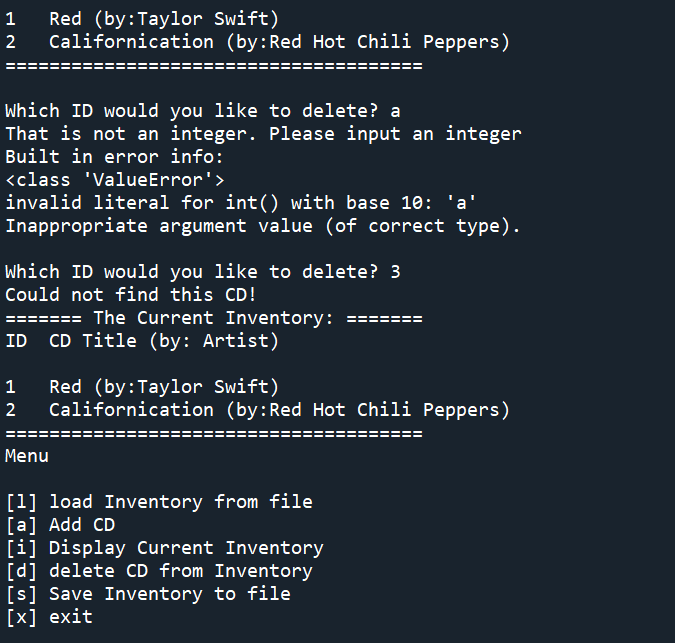


Figure - Spyder script run part 5

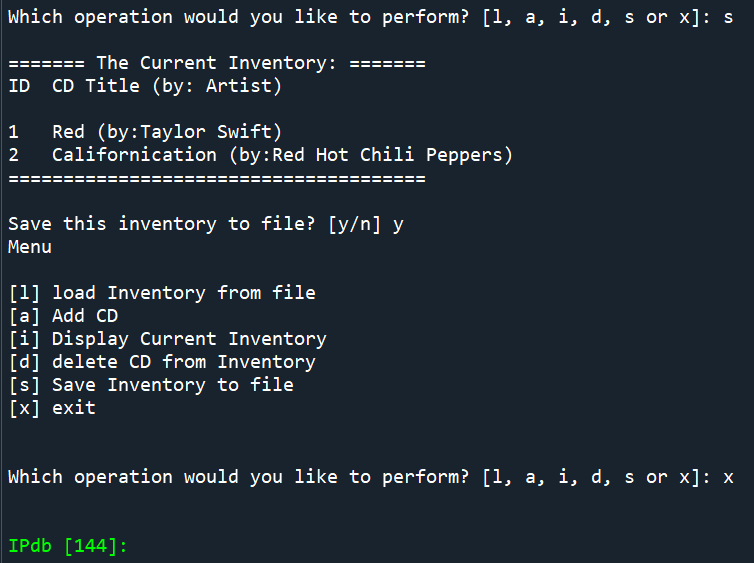


Figure - Spyder

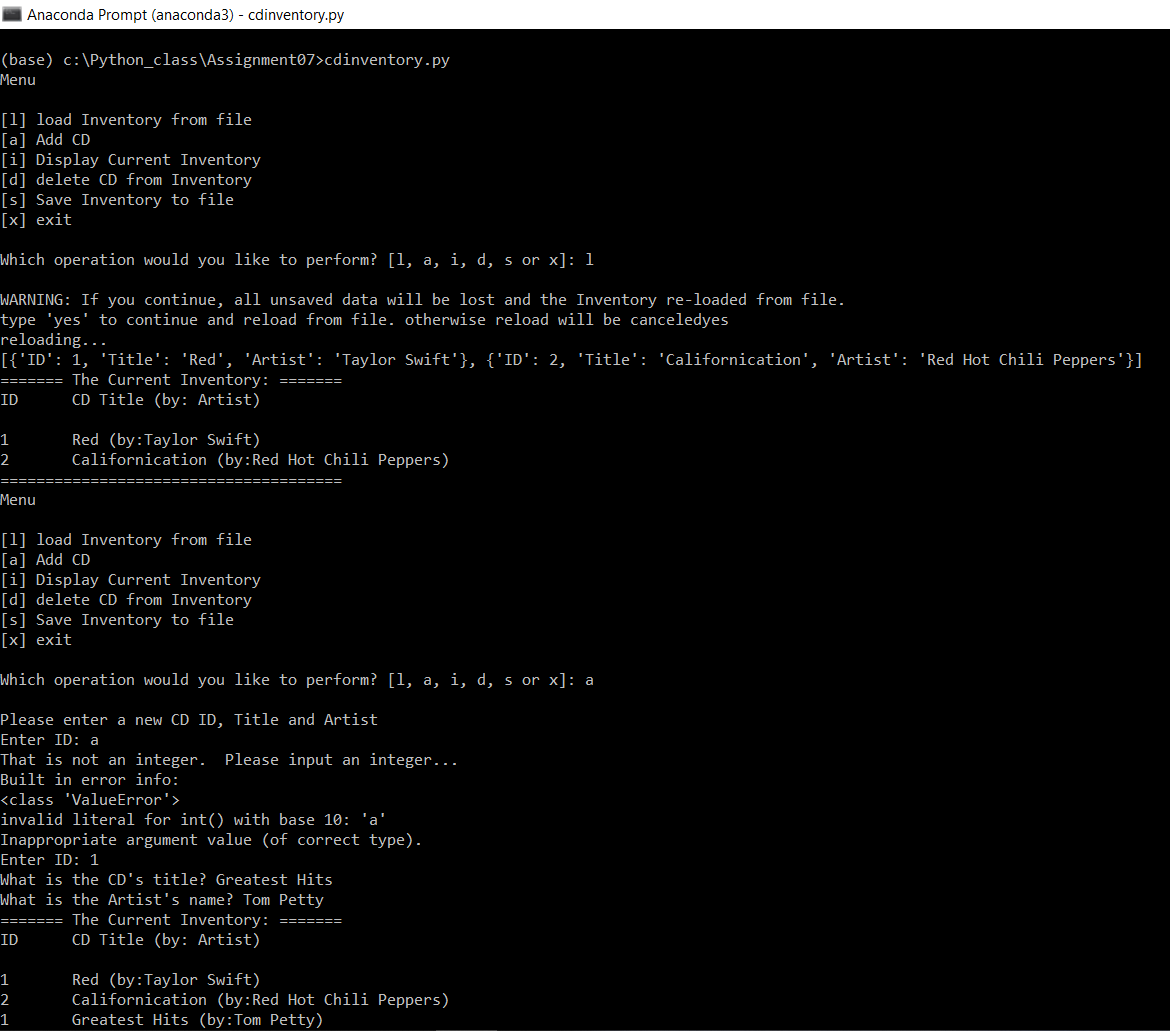


Figure - shell run of script

1. *#------------------------------------------#*
2. *# Title: CDInventory.py*
3. *# Desc: Working with classes and functions.*
4. *# Change Log: (Who, When, What)*
5. ***# DBiesinger, 2030-Jan-01, Created File***
6. *# Evan Malina, 2021-Nov-21, Updated file to add, delete, write CD data*
7. *# Evan Malina, 2021-Nov-28, Updated for error handling and binary*
8. *#------------------------------------------#*
10. **import pickle *# pickle used for writing/reading binary***
12. *# -- DATA -- #*
13. strChoice = '' *# User input*
14. lstTbl = [] *# list of lists to hold data*
15. **dicRow = {} *# list of data row***
16. strFileName = 'CDInventory.dat' *# data storage file*
17. *# updated strFileName to binary on 11/28/2021*
18. objFile = None *# file object*

21. *# -- PROCESSING -- #*
22. **class** DataProcessor:
23. *# TODOne add functions for processing here*
24. """Functions I added for processing CD data into a table"""
25. **@staticmethod**
26. **def** input\_CD(strNum, strCDTitle, str\_Artist, table):
27. """Function to add inputs into table
29. Args:
30. **strNum (string): First input is the ID number**
31. strCDTitle (string): Second input is album Title
32. str\_Artist (string): 3rd input is album artist (TYPO)
33. table (list): current inventory list of dictionaries
35. **Returns:**
36. table (list): a list of dictionaries of CD entries
37. """
38. *# intID = int(strNum) #convert string to integer*
39. dicRow = {'ID': strNum, 'Title': strCDTitle, 'Artist': str\_Artist}
40. **table.append(dicRow)**
41. **return** table
43. @staticmethod
44. **def** delete\_CD(delNum, table):
45. **"""Function to find and delete a specific ID from inventory**
46. Args:
47. delNum (int): ID to delete from inventory
48. table (list): cd inventory table
50. **Returns:**
51. table (list): cd inventory table
53. """
54. intRowNr = -1
55. **blnCDRemoved = False**
56. **for** row **in** table:
57. intRowNr += 1
58. **if** row['ID'] == delNum:
59. **del** table[intRowNr]
60. **blnCDRemoved = True**
61. **break**
62. **if** blnCDRemoved:
63. **print**('The CD was removed')
64. **else**:
65. **print('Could not find this CD!')**
66. **return** table

69. **class** FileProcessor:
70. **"""Processing the data to and from text file"""**
72. @staticmethod
73. **def** read\_file(file\_name, table):
74. """Function to manage data ingestion from file to a list of dictionaries
76. Reads the data from file identified by file\_name into a 2D table
77. (list of dicts) table one line in the file represents one dictionary row in table.
79. Args:
80. **file\_name (string): name of file used to read the data from**
81. table (list of dict): 2D data structure (list of dicts) that holds the data during runtime
83. Returns:
84. None.
85. **"""**
86. table.clear() *# this clears existing data and allows to load data from file*
87. **try**:
88. objFile = open(file\_name, 'rb')
89. table = pickle.load(objFile)
90. ***# print(table)***
91. objFile.close()
92. **return** table
93. **except** FileNotFoundError **as** e:
94. **print**('File not found')
95. **print('Built in error info: ')**
96. **print**(type(e), e, e.\_\_doc\_\_, sep='**\n**')
97. table = []
98. **return** table

101. @staticmethod
102. **def** write\_file(file\_name, table):
103. *# TODone Add code here*
104. """Function to save the CD to file
105. **Args:**
106. file\_name (str): file name for saving
107. table (list): current inventory of CDs. A list of dictionaries
109. Returns: none
110. **"""**
112. objFile = open(file\_name, 'wb')
113. *# print(table)*
114. pickle.dump(table, objFile)
115. **objFile.close()**
117. *# -- PRESENTATION (Input/Output) -- #*
119. **class** IO:
120. **"""Handling Input / Output"""**
122. @staticmethod
123. **def** print\_menu():
124. """Displays a menu of choices to the user
126. Args:
127. None.
129. Returns:
130. **None.**
131. """
133. **print**('Menu**\n\n**[l] load Inventory from file**\n**[a] Add CD**\n**[i] Display Current Inventory')
134. **print**('[d] delete CD from Inventory**\n**[s] Save Inventory to file**\n**[x] exit**\n**')
136. @staticmethod
137. **def** menu\_choice():
138. """Gets user input for menu selection
140. **Args:**
141. None.
143. Returns:
144. choice (string): a lower case sting of the users input out of the choices l, a, i, d, s or x
146. """
147. choice = ' '
148. **while** choice **not** **in** ['l', 'a', 'i', 'd', 's', 'x']:
149. choice = input('Which operation would you like to perform? [l, a, i, d, s or x]: ').lower().strip()
150. **print() *# Add extra space for layout***
151. **return** choice
153. @staticmethod
154. **def** show\_inventory(table):
155. **"""Displays current inventory table**

158. Args:
159. table (list of dict): 2D data structure (list of dicts) that holds the data during runtime.
161. Returns:
162. None.
164. """
165. **print('======= The Current Inventory: =======')**
166. **print**('ID**\t**CD Title (by: Artist)**\n**')
167. **for** row **in** table:
168. **print**('{}**\t**{} (by:{})'.format(\*row.values()))
169. **print**('======================================')
171. *# TODone add I/O functions as needed*
172. @staticmethod
173. **def** add\_data():
174. """requests user to input new CD information
176. Args: none
178. Returns: tuple of 3;
179. var1 (int): ID number
180. **var2 (str): CD title**
181. var3 (str): CD artist
182. """
183. **print**('Please enter a new CD ID, Title and Artist')
184. **while** True:
185. **try:**
186. var1 = int(input('Enter ID: ').strip())
187. **break**
188. **except** ValueError **as** e:
189. **print**('That is not an integer. Please input an integer...')
190. **print('Built in error info: ')**
191. **print**(type(e), e, e.\_\_doc\_\_, sep='**\n**')
193. var2 = input('What is the CD**\'**s title? ').strip()
194. var3 = input('What is the Artist**\'**s name? ').strip()
195. **return(var1, var2, var3)**


199. *# 1. When program starts, read in the currently saved Inventory*
200. **FileProcessor.read\_file(strFileName, lstTbl)**
202. *# 2. start main loop*
203. **while** True:
204. *# 2.1 Display Menu to user and get choice*
205. **IO.print\_menu()**
206. strChoice = IO.menu\_choice()
208. *# 3. Process menu selection*
209. *# 3.1 process exit first*
210. **if strChoice == 'x':**
211. **break**
212. *# 3.2 process load inventory*
213. **if** strChoice == 'l':
214. **print**('WARNING: If you continue, all unsaved data will be lost and the Inventory re-loaded from file.')
215. **strYesNo = input('type \'yes\' to continue and reload from file. otherwise reload will be canceled')**
216. **if** strYesNo.lower() == 'yes':
217. **print**('reloading...')
218. lstTbl = FileProcessor.read\_file(strFileName, lstTbl)
219. **print**(lstTbl)
220. **IO.show\_inventory(lstTbl)**
221. **else**:
222. input('canceling... Inventory data NOT reloaded. Press [ENTER] to continue to the menu.')
223. IO.show\_inventory(lstTbl)
224. **continue** *# start loop back at top.*
225. ***# 3.3 process add a CD***
226. **elif** strChoice == 'a':
227. *# 3.3.1 Ask user for new ID, CD Title and Artist*
228. *# TODOne move IO code into function*
229. strID, strTitle, stArtist = IO.add\_data()
230. ***# 3.3.2 Add item to the table***
231. *# TODOne move processing code into function*
232. DataProcessor.input\_CD(strID, strTitle, stArtist, lstTbl)
233. IO.show\_inventory(lstTbl)
234. **continue** *# start loop back at top.*
235. ***# 3.4 process display current inventory***
236. **elif** strChoice == 'i':
237. IO.show\_inventory(lstTbl)
238. **continue** *# start loop back at top.*
239. *# 3.5 process delete a CD*
240. **elif strChoice == 'd':**
241. *# 3.5.1 get Userinput for which CD to delete*
242. *# 3.5.1.1 display Inventory to user*
243. IO.show\_inventory(lstTbl)
244. *# 3.5.1.2 ask user which ID to remove*
245. **while True:**
246. **try**:
247. intIDDel = int(input('Which ID would you like to delete? ').strip())
248. **break**
249. **except** ValueError **as** e:
250. **print('That is not an integer. Please input an integer')**
251. **print**('Built in error info: ')
252. **print**(type(e), e, e.\_\_doc\_\_, sep='**\n**')
253. *# 3.5.2 search thru table and delete CD*
254. *# TODone move processing code into function*
255. **DataProcessor.delete\_CD(intIDDel, lstTbl)**
256. IO.show\_inventory(lstTbl)
257. **continue** *# start loop back at top.*
258. *# 3.6 process save inventory to file*
259. **elif** strChoice == 's':
260. ***# 3.6.1 Display current inventory and ask user for confirmation to save***
261. IO.show\_inventory(lstTbl)
262. strYesNo = input('Save this inventory to file? [y/n] ').strip().lower()
263. *# 3.6.2 Process choice*
264. **if** strYesNo == 'y':
265. ***# 3.6.2.1 save data***
266. *# TODO move processing code into function*
267. FileProcessor.write\_file(strFileName, lstTbl)
268. **else**:
269. ('The inventory was NOT saved to file. Press [ENTER] to return to the menu.')
270. **continue *# start loop back at top.***
271. *# 3.7 catch-all should not be possible, as user choice gets vetted in IO, but to be save:*
272. **else**:
273. **print**('General Error')

Listing - CDInventory.py script

# Summary

This ended up going really well. I was a little bit rushed, and ended up even panicking after having an error problem due to the FileNotFound error. What happened was that initially I acknowledged that it was acceptable for the table to return as empty. I ended up having to return an empty table to solve something that I no longer recall why. I ended up making the empty table a dictionary instead and received even more errors! Anyhow, this was a good lesson on not waiting until the last minute, but overall it went well and I just had to work through things. During the labs, I was pretty confused by B, and thought I could simply write text into a .dat file. I also learned how to run python in spyder with arguments 😊. This was the lab that threw me for the biggest loop, and was much more difficult than the assignment. I am very happy to write to binary with tables/dictionaries because the code is soooo much simpler.