Kelly Ros

2020-Mar-09

Foundations of Programming, Python

Assignment 07

Binary Files and Structured Error Handling

# Introduction

In this module, we went over binary files, how to use the pickle library, and structured error handling.

# Binary Files and Pickling

Binary files are any file, where the format isn’t made up of readable characters. In python, we open a binary file as we would a normal file, but would add a ‘b’ when specifying the mode of the open function. The differences between a text file and a binary file include, readability, editing ability, and extension specific programs.[[1]](#footnote-1) Texts files are a type of binary file, but not all binary files are text files. If a binary file were treated as a text file, the bytes of data can lead to corrupted data.

Pickling data, as I’ve read multiple times, is the process of conserving and storing data for use later. In software terms, pickling is used for serializing and de-serializing Python object structures. Serialization is when an object in memory is converted to a byte stream that can be stored on a disk or sent over a network.[[2]](#footnote-2) An example of pickled data is shown in figure 1.

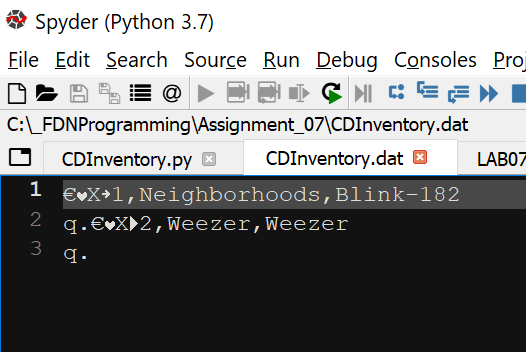


Figure 1 – CDInventory.dat was created using CDInventory.py

Using CDInventory.py’s write\_file function, we serialize the 2D table that we created by taking each element of the list and converting its contents into a string and we use pickle’s dump function to write the serialized data into the file. We can see the code in listing 1.

1. **def** write\_file(file\_name, table):
2. """Function that Saves inventory to text file
3. Reads 2D table and formats line by line to write to a text file
5. Args:
6. file\_name (string): name of file used to save data to
7. table (list of dict): 2D data structure that holds data during runtime
9. Returns:
10. None
11. """
12. with open(file\_name, 'wb') as objFile:
13. **for** row **in** table:
14. lstValues = list(row.values())
15. lstValues[0] = str(lstValues[0])
16. tblString = ','.join(lstValues) + '\n'
17. pickle.dump(tblString, objFile)

Listing 1. Write\_file function from CDInventory.py

In the program’s read\_file function, we are able to de-serialize the data and load it into the 2D table as shown in figure 2. In the function, an empty list is created, and the binary file is loaded line by line and appended into the list, then the list is converted into a dictionary and appended to the 2D table. In order to perform this, I made use of structure error handling.

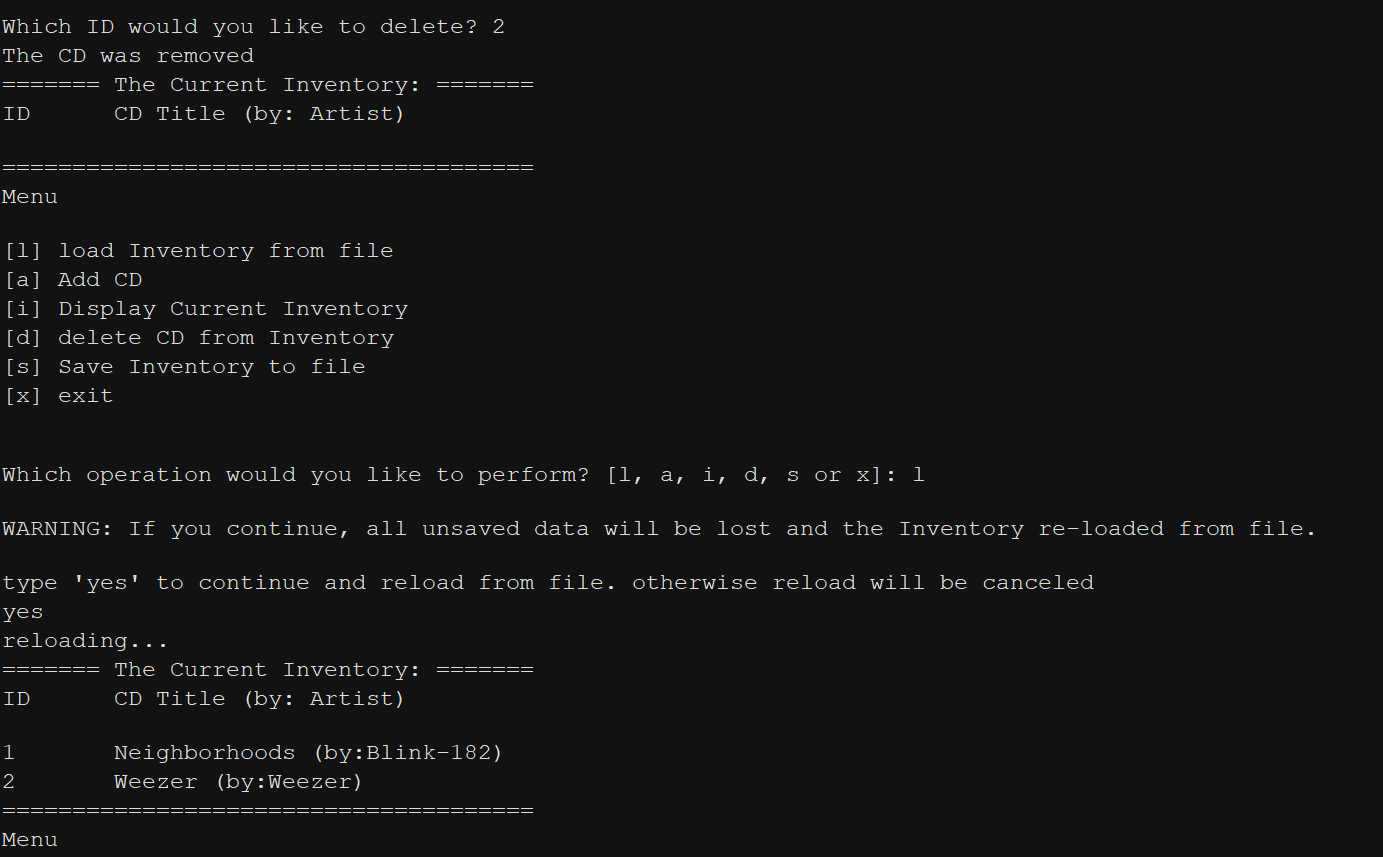


Figure 2 – read\_file function

# Structured Error Handling

There are plenty of times in this course where I’ve encountered an error that breaks my program and causes it to halt. In last week’s module, I included a try-except loop in my code in order to not have to program crash without having the correct text file in the same directory. The loop that I previously used, wasn’t the best practice. In order to keep good habits, we must structure our error handling using the Exception class. The Exception class includes many errors and their names, such as ZeroDivisionError, FileNotFoundError, etc.[[3]](#footnote-3) Figure 3, shows the most common error that would happen when you run this program for the first time, however instead of halting the program, it continues and shows the menu.

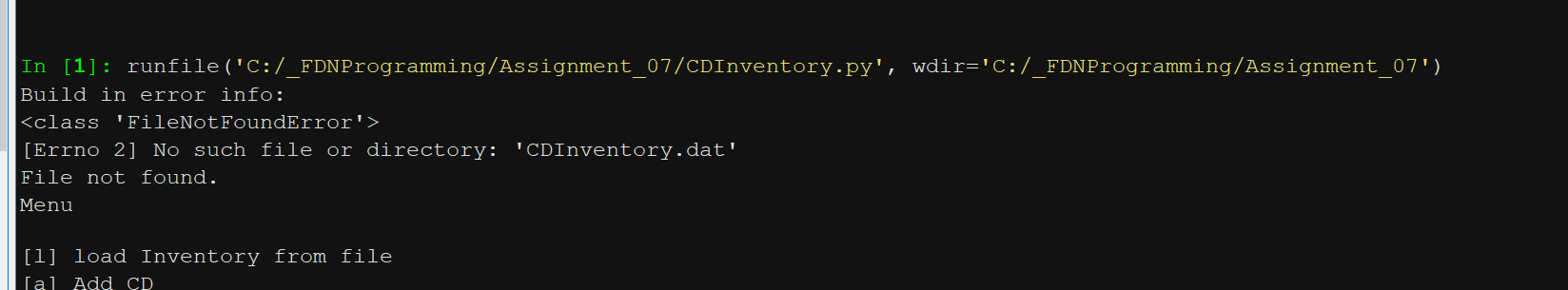


Figure 3 – FileNotFoundError upon running the program without CDInventory.dat

1. **def** read\_file(file\_name, table):
2. """Function to manage data ingestion from file to a list of dictionaries
3. FileNotFound error is raised if CDInventory.dat does not exist
4. EOFError is raised when the end of CDInventory file is reached
6. Reads the data from file identified by file\_name into a 2D table
7. (list of dicts) table one line in the file represents one dictionary row in table.
8. If file does not exist, a file will be created
10. Args:
11. file\_name (string): name of file used to read the data from
12. table (list of dicts): 2D data structure (list of dicts) that holds the data during runtime
14. Returns:
15. table (list of dicts): table returned from opening file
16. """
17. **try**:
18. cdInfo = []
19. with open(file\_name, 'rb') as objFile:
20. table.clear()  # this clears existing data and allows to load data from file
21. **while** True:
22. **try**:
23. cdInfo.append(pickle.load(objFile))
24. **except** EOFError as e:
25. **print**(type(e), e, e.\_\_doc\_\_, sep='\n')
26. **print**("Reached end of File!")
27. **break**
28. **for** line **in** cdInfo:
29. data = line.strip().split(',')
30. dicRow = {'ID': int(data[0]), 'Title': data[1], 'Artist': data[2]}
31. table.append(dicRow)
32. **except** FileNotFoundError as e:
33. **print**('Build in error info:')
34. **print**(type(e), e, e.\_\_doc\_\_, sep='\n')
36. **return** table

Listing 2 – FileNotFoundError in except block

In listing 2, we can see the program “try” to open the binary file, and if it fails to find the file, the exception is raised and prints out the error information and then continues with the program. Also in listing 2, a second try-except loop was used when loading information from the binary file. In this case, the EOFError exception in raised when there are no more lines in the binary file to be loaded as shown in figure 4. Figure 5 shows the ValueError.

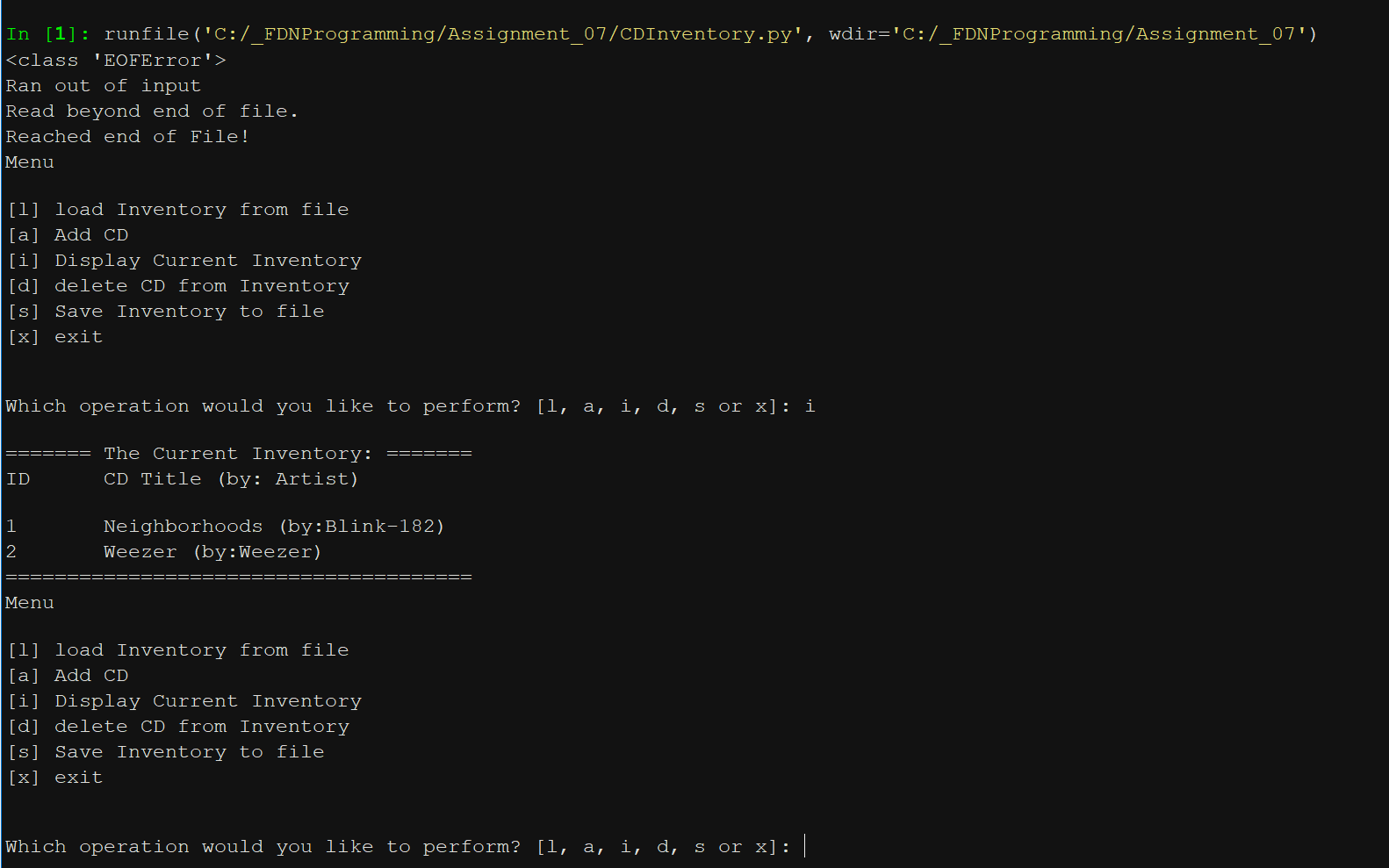


Figure 4 – EOFError is raised when the end of CDInventory.dat is reached and no more data can be loaded

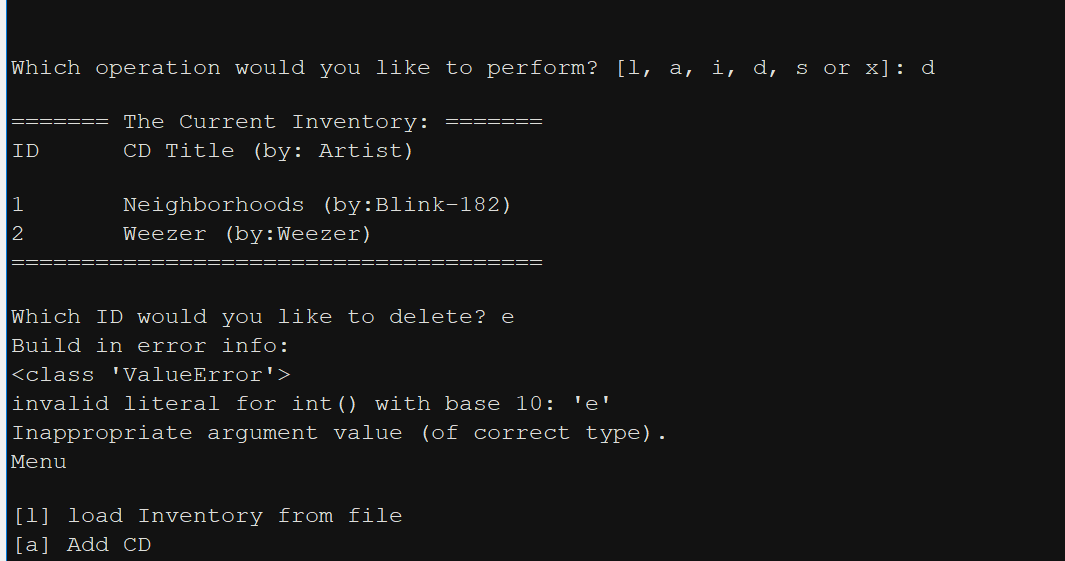


Figure 5 – ValueError occurs when a string is entered when the program was expecting an int or float

# Summary

In this week’s module, we went more in depth on the topic of reading and writing files, which includes text and binary files, the pickle library, and structured error handling. The concepts from the pickle library were a little difficult to understand and use at first, but after seeing multiple examples on the web, I was able to understand and use it effectively.

# Appendix

## Listing CDInventory.py

1. #------------------------------------------#
2. # Title: CDInventory.py
3. # Desc: Updated Assignment 06 with Error Handling and Binary files
4. # Change Log: (Who, When, What)
5. # DBiesinger, 2030-Jan-01, Created File
6. # KRos, 2020-Feb-28, added write\_file function
7. # KRos, 2020-Feb-29, created add\_CD and delete\_CD functions, added Docstrings
8. # KRos, 2020-Mar-01, added in try-except for case where file does not exist
9. # KRos, 2020-Mar-08, added in structured error handling and changed program to read and write binary files
10. # KRos, 2020-Mar-09, updated some doc strings
11. #------------------------------------------#
13. # -- IMPORT MODULES -- #
14. **import** pickle
16. # -- DATA -- #
17. strChoice = '' # User input
18. lstTbl = []  # list of lists to hold data
19. dicRow = {}  # list of data row
20. strFileName = 'CDInventory.dat'  # data storage file
21. objFile = None  # file object

24. # -- PROCESSING -- #
25. **class** DataProcessor:
26. """Data Processing Functions"""
28. @staticmethod
29. **def** add\_CD(idno, title, artist, table):
30. """Function that adds CD to list of dictionaries, rasies ValueError ifidno is not an int
32. Args:
33. idno (string): ID number of CD, which will be converted to int
34. title (string): CD title
35. artist (string): CD artist
36. table (list of dicts): 2D table that will be appended with new CD entry
38. Returns:
39. table (list of dicts): 2D table that has a CD added to it
40. """
41. **try**:
42. dicRow  = {'ID': int(idno), 'Title': title, 'Artist': artist}
43. table.append(dicRow)
44. **except** ValueError as e:
45. **print**('Build in error info:')
46. **print**(type(e), e, e.\_\_doc\_\_, sep='\n')
47. **return** table
49. @staticmethod
50. **def** delete\_CD(idno, table):
51. """Function that deletes a CD entry from a 2D table
53. Args:
54. idno (int): CD ID number to be deleted
55. table (list of dicts): 2D list that CD should be removed from
57. Returns:
58. table (list of dicts): 2D list that has been edited
59. """
60. intRowNr = -1
61. blnCDRemoved = False
62. **for** row **in** table:
63. intRowNr += 1
64. **if** row['ID'] == idno:
65. **del** table[intRowNr]
66. blnCDRemoved = True
67. **break**
68. **if** blnCDRemoved:
69. **print**('The CD was removed')
70. **else**:
71. **print**('Could not find this CD!')
72. **return** table
74. **class** FileProcessor:
75. """Processing the data to and from text file"""
77. @staticmethod
78. **def** read\_file(file\_name, table):
79. """Function to manage data ingestion from file to a list of dictionaries
80. FileNotFound error is raised if CDInventory.dat does not exist
81. EOFError is raised when the end of CDInventory file is reached
83. Reads the data from file identified by file\_name into a 2D table
84. (list of dicts) table one line in the file represents one dictionary row in table.
85. If file does not exist, a file will be created
87. Args:
88. file\_name (string): name of file used to read the data from
89. table (list of dicts): 2D data structure (list of dicts) that holds the data during runtime
91. Returns:
92. table (list of dicts): table returned from opening file
93. """
94. **try**:
95. cdInfo = []
96. with open(file\_name, 'rb') as objFile:
97. table.clear()  # this clears existing data and allows to load data from file
98. **while** True:
99. **try**:
100. cdInfo.append(pickle.load(objFile))
101. **except** EOFError as e:
102. **print**(type(e), e, e.\_\_doc\_\_, sep='\n')
103. **print**("Reached end of File!")
104. **break**
105. **for** line **in** cdInfo:
106. data = line.strip().split(',')
107. dicRow = {'ID': int(data[0]), 'Title': data[1], 'Artist': data[2]}
108. table.append(dicRow)
109. **except** FileNotFoundError as e:
110. **print**('Build in error info:')
111. **print**(type(e), e, e.\_\_doc\_\_, sep='\n')
113. **return** table
114. @staticmethod
115. **def** write\_file(file\_name, table):
116. """Function that Saves inventory to text file
117. Reads 2D table and formats line by line to write to a text file
119. Args:
120. file\_name (string): name of file used to save data to
121. table (list of dict): 2D data structure that holds data during runtime
123. Returns:
124. None
125. """
126. with open(file\_name, 'wb') as objFile:
127. **for** row **in** table:
128. lstValues = list(row.values())
129. lstValues[0] = str(lstValues[0])
130. tblString = ','.join(lstValues) + '\n'
131. pickle.dump(tblString, objFile)
133. # -- PRESENTATION (Input/Output) -- #
135. **class** IO:
136. """Handling Input / Output"""
138. @staticmethod
139. **def** print\_menu():
140. """Displays a menu of choices to the user
142. Args:
143. None.
145. Returns:
146. None.
147. """
149. **print**('Menu\n\n[l] load Inventory from file\n[a] Add CD\n[i] Display Current Inventory')
150. **print**('[d] delete CD from Inventory\n[s] Save Inventory to file\n[x] exit\n')
152. @staticmethod
153. **def** menu\_choice():
154. """Gets user input for menu selection
156. Args:
157. None.
159. Returns:
160. choice (string): a lower case sting of the users input out of the choices l, a, i, d, s or x
162. """
163. choice = ' '
164. **while** choice **not** **in** ['l', 'a', 'i', 'd', 's', 'x']:
165. choice = input('Which operation would you like to perform? [l, a, i, d, s or x]: ').lower().strip()
166. **print**()  # Add extra space for layout
167. **return** choice
169. @staticmethod
170. **def** show\_inventory(table):
171. """Displays current inventory table

174. Args:
175. table (list of dict): 2D data structure (list of dicts) that holds the data during runtime.
177. Returns:
178. None.
180. """
181. **print**('======= The Current Inventory: =======')
182. **print**('ID\tCD Title (by: Artist)\n')
183. **for** row **in** table:
184. **print**('{}\t{} (by:{})'.format(\*row.values()))
185. **print**('======================================')
187. @staticmethod
188. **def** input\_CD():
189. """Asks user for CD inputs
191. Args:
192. None
194. Returns:
195. ID, CD title, Artist
197. """
198. strID = input('Enter ID: ').strip()
199. strTitle = input('What is the CD\'s title? ').strip()
200. stArtist = input('What is the Artist\'s name? ').strip()
201. **return** strID, strTitle, stArtist
203. # 1. When program starts, read in the currently saved Inventory
204. FileProcessor.read\_file(strFileName, lstTbl)
206. # 2. start main loop
207. **while** True:
208. # 2.1 Display Menu to user and get choice
209. IO.print\_menu()
210. strChoice = IO.menu\_choice()
212. # 3. Process menu selection
213. # 3.1 process exit first
214. **if** strChoice == 'x':
215. **break**
216. # 3.2 process load inventory
217. **if** strChoice == 'l':
218. **print**('WARNING: If you continue, all unsaved data will be lost and the Inventory re-loaded from file.')
219. strYesNo = input('type \'yes\' to continue and reload from file. otherwise reload will be canceled \n')
220. **if** strYesNo.lower() == 'yes':
221. **print**('reloading...')
222. FileProcessor.read\_file(strFileName, lstTbl)
223. IO.show\_inventory(lstTbl)
224. **else**:
225. input('canceling... Inventory data NOT reloaded. Press [ENTER] to continue to the menu.')
226. IO.show\_inventory(lstTbl)
227. **continue**  # start loop back at top.
228. # 3.3 process add a CD
229. **elif** strChoice == 'a':
230. # 3.3.1 Ask user for new ID, CD Title and Artist
231. strID, strTitle, stArtist = IO.input\_CD()
232. # 3.3.2 Add item to the table
233. DataProcessor.add\_CD(strID, strTitle, stArtist, lstTbl)
234. IO.show\_inventory(lstTbl)
235. **continue**  # start loop back at top.
236. # 3.4 process display current inventory
237. **elif** strChoice == 'i':
238. IO.show\_inventory(lstTbl)
239. **continue**  # start loop back at top.
240. # 3.5 process delete a CD
241. **elif** strChoice == 'd':
242. # 3.5.1 get Userinput for which CD to delete
243. # 3.5.1.1 display Inventory to user
244. IO.show\_inventory(lstTbl)
245. # 3.5.1.2 ask user which ID to remove
246. **try**:
247. intIDDel = int(input('Which ID would you like to delete? ').strip())
248. # 3.5.2 search thru table and delete CD
249. DataProcessor.delete\_CD(intIDDel, lstTbl)
250. IO.show\_inventory(lstTbl)
251. **except** ValueError as e:
252. **print**('Build in error info:')
253. **print**(type(e), e, e.\_\_doc\_\_, sep='\n')
254. **continue**  # start loop back at top.
255. # 3.6 process save inventory to file
256. **elif** strChoice == 's':
257. # 3.6.1 Display current inventory and ask user for confirmation to save
258. IO.show\_inventory(lstTbl)
259. strYesNo = input('Save this inventory to file? [y/n] ').strip().lower()
260. # 3.6.2 Process choice
261. **if** strYesNo == 'y':
262. # 3.6.2.1 save data
263. FileProcessor.write\_file(strFileName, lstTbl)
264. **else**:
265. input('The inventory was NOT saved to file. Press [ENTER] to return to the menu.')
266. **continue**  # start loop back at top.
267. # 3.7 catch-all should not be possible, as user choice gets vetted in IO, but to be save:
268. **else**:
269. **print**('General Error')

1. Retrieved from <https://www.nayuki.io/page/what-are-binary-and-text-files>. On 2020-Mar-08 [↑](#footnote-ref-1)
2. Retrieved from <https://www.datacamp.com/community/tutorials/pickle-python-tutorial#what>. On 2020-Mar-08 [↑](#footnote-ref-2)
3. Retrieved from <https://www.tutorialspoint.com/python3/python_exceptions.htm> on 08-Mar-2020 [↑](#footnote-ref-3)