Kenny Lu

2020-Mar-09

IT FDN 100 B Winter 2020

Assignment07

GitHub <https://github.com/lukenny/Assignment_07>

# Introduction

In this seventh module, we cover structured error handling, exception class, text vs. binary data file, pickling and markdown language. It’s by far the toughest module and its assignment is also the hardest because there’re no TODO hints like assignment06 so it’s up to us to determine where to apply structured error handling. Also, with pickling it requires a complete rewrite of both “write file” and “read file” functions. All in all, it’s the most challenging.

Topic 1 – Structured error handling

It’s very common to encounter program where error handling is not done properly. For instance, an invalid input or something that’s not idempotent that causes the entire program to crash. From a programming perspective, it’s bad user experience if the program crashes or they’re presented with unnecessary stack trace. Two very common error exceptions we get is “ValueError” (Listing 1) and “FileNotFound” (Listing 2). With Listing 1, my try-except block verifies the input is integer and catches if it’s not and raise the exception before the program goes any further. I wanted to catch it before it gets to processing.

1. **while** True:
2. str\_cd\_id = input('Enter ID (Integer only): ').strip()
3. # Allow user to input 'exit' to get out of the loop
4. **if** str\_cd\_id.lower() == 'exit':
5. **break**
6. # Try-except to ensure input is integer
7. **try**:
8. cd\_id = int(str\_cd\_id)
9. **except** ValueError as e:
10. **print**("Oops! Please enter integer only. Try again or type \'exit\' to return to the menu.")
11. **continue**
12. cd\_title = input('What is the CD\'s title? ').strip()
13. cd\_artist = input('What is the Artist\'s name? ').strip()
14. **print**(cd\_id, cd\_title, cd\_artist)
15. **return** cd\_id, cd\_title, cd\_artist

Listing 1 – sample code to demonstrate ValueError exception

As for Listing 2, the concept is similar to Listing 1 we are simply trying to catch the potential error if in case the target file doesn’t exist, the program shouldn’t just crash. It’s very likely when dealing with underlying file system, file gets deleted permission gets altered or general file system error.

1. **try**:
2. with open(binFileName, 'rb') as fileObj:
3. table = pickle.load(fileObj)
4. **except** FileNotFoundError:
5. **print**("The file {} could not be loaded".format(file\_name))
7. **return** table

Listing 2 – sample output to demonstrate FileNotFoundError exception

Using [w3schools](https://www.w3schools.com/python/python_try_except.asp) (external reference) webpage

Using [python.org](https://docs.python.org/3/library/exceptions.html) (external reference) webpage

Using [stackoveflow](https://stackoverflow.com/questions/6041981/python-check-if-value-is-in-a-list-of-dicts) (external reference) webpage ## check against duplicate id

# Topic 2 – Pickling

In Python, pickle is used to serialize or deserialize object structure. It serializes the object first before writing to a file in binary format; it converts python object; for instance, list or dict, into a character stream. One of the advantages of using pickling is that the character stream can then be leveraged and de-serialized back to a Python object. It’s also a very common practice to use pickling if the data being saved doesn’t need to be human readable in the file system. In general, whenever we save anything that’s human readable, it requires additional steps to perform conversion or processing as Python handles data stored in memory in binary format natively. It’s also very important to keep in mind that pickling is not compression or encryption; character stream can be re-constructed easily even though it’s not human readable. Let’s take a look at Listing 3 with pickling and unpickling. As you can see, the syntax to write binary data is very different than writing data to text file. Also, line 8 shows there’re ASCII characters instead of just human readable plain text; however, the character stream can easily be de-serialized or unpickled.

1. In [15]: **import** pickle
2. ...: fc = {1:"Sounders",2:"Manchester United",3:"Liverpool FC",4:"Barcalona FC",5:"Real Madrid"}
3. ...: pickling = open("fc.pickle","wb")
4. ...: pickle.dump(fc, pickling)
5. ...: pickling.close()
7. In [16]: cat fc.pickle
8. �}q(KSoundersqKXManchester UnitedqKX
9. Liverpool FCqKX
10. Barcalona FCqKX
11. Real Madridqu.
12. In [17]: unpickling = open("fc.pickle","rb")
13. ...: fc = pickle.load(unpickling)
14. ...: **print**(fc)
15. {1: 'Sounders', 2: 'Manchester United', 3: 'Liverpool FC', 4: 'Barcalona FC', 5: 'Real Madrid'}

Listing 3 – sample code pickle and unpickle

Using [pythonforthelab](https://www.pythonforthelab.com/blog/storing-binary-data-and-serializing/) (external reference) webpage

Using [geeksforgeeks](https://www.geeksforgeeks.org/understanding-python-pickling-example/) (external reference) webpage

Using [datacamp](https://www.datacamp.com/community/tutorials/pickle-python-tutorial) (external reference) webpage

Using [pythoncentral](https://www.pythoncentral.io/how-to-pickle-unpickle-tutorial/) (external reference) webpage

# Summary

All in all, this assignment was more challenging than the rest for me because firstly I went deep when TODO deliverables were not spelled out explicitly. For instance, with structured error handling, I wanted to make sure everything was covered which required me to go over every possible area where it needed error handling. Secondly, in order to complete the assignment, it’s a requirement to be very familiar with assignment06 as we’re building more features on top of it. Nonetheless, part of assignment06 had been given to us and not written by us from scratch; it took me a little bit to make sure I completely understood each and every component of assignment06 before starting assignment07. For instance, last thing I wanted was to introduce new bugs where we didn’t encounter with assignment06. Last but not least, I spent a long time with pickling because I had no idea how it would impact other components of the program. Also, when the data saved was not human readable, it was rather confusing for me to debug as I wasn’t sure if the data was saved correctly or formatted properly. With text file, at least I could simply cat the file to ascertain if the data was saved correctly or not.

# Appendix

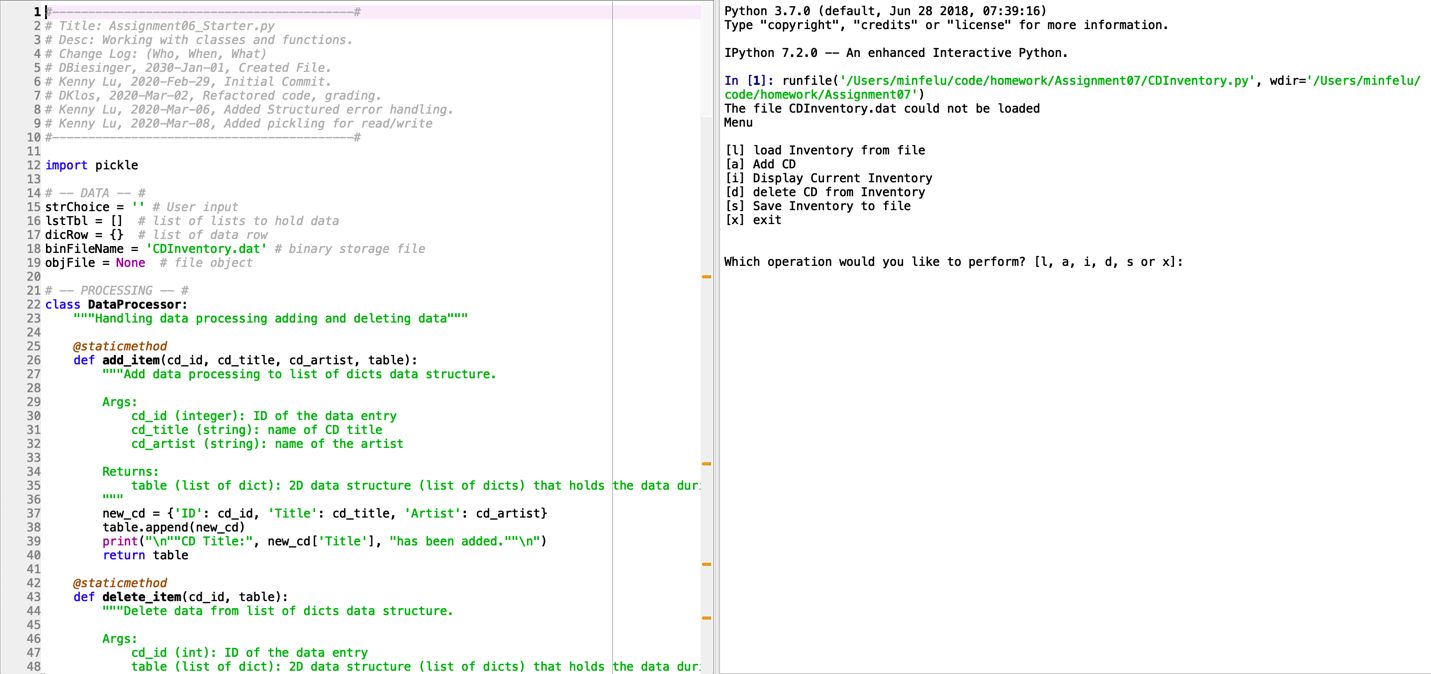


Figure 1 – Example output from script initial loaded in Spyder

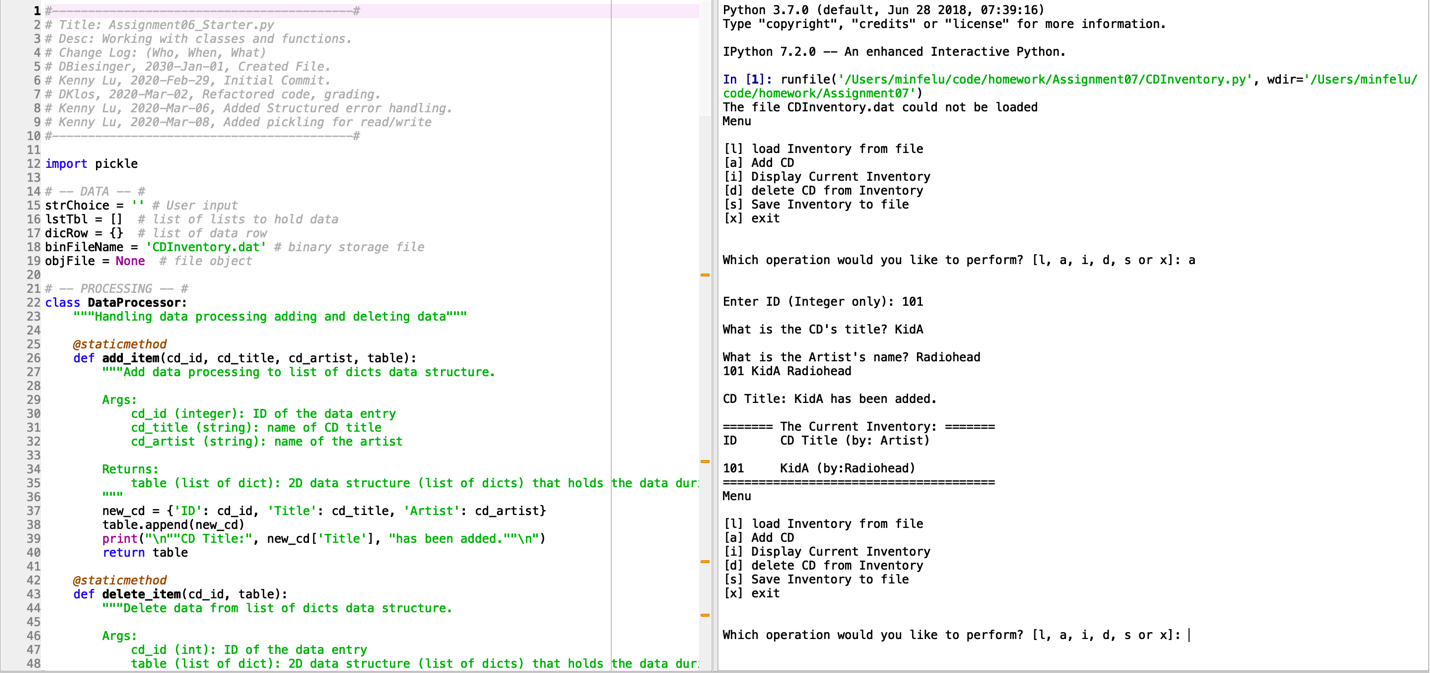


Figure 2 – Example output from script add data and display inventory

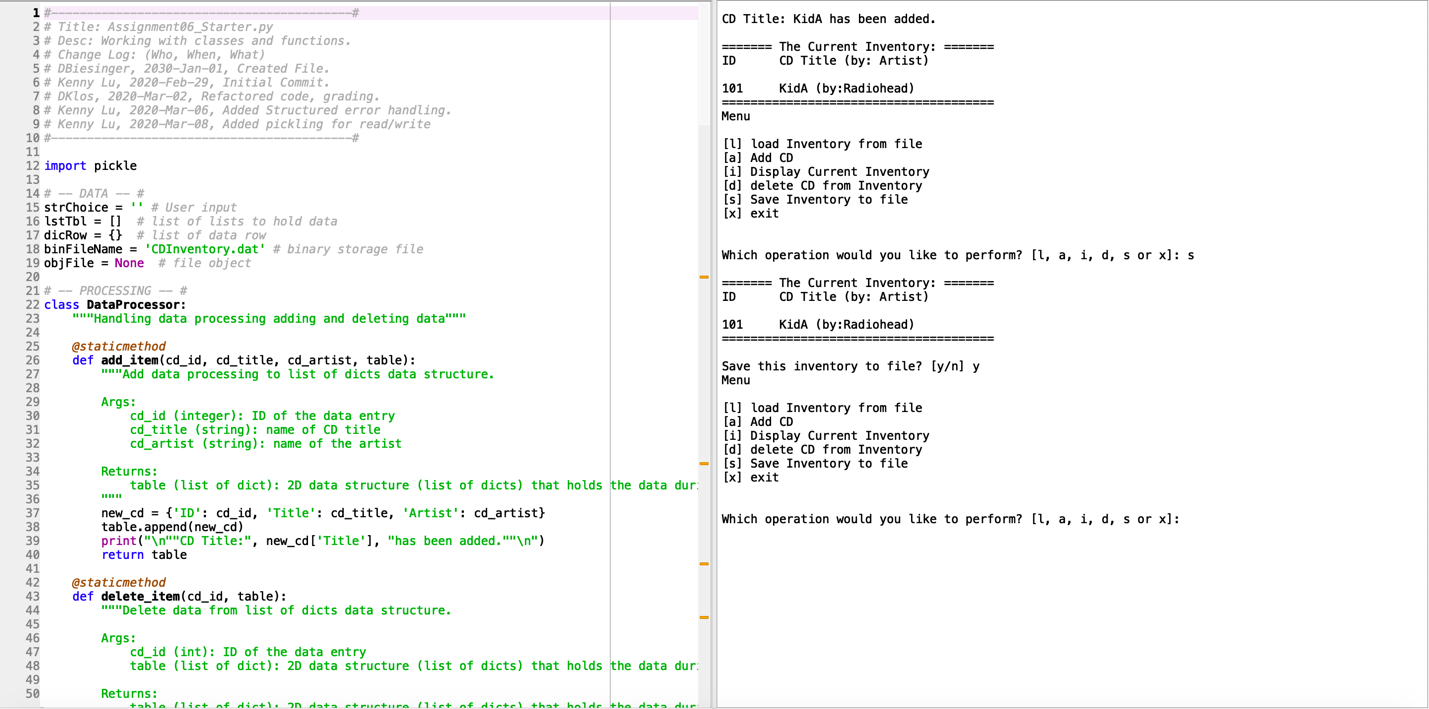


Figure 3 – Example output from script save inventory

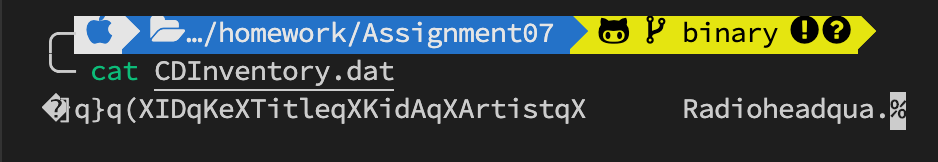
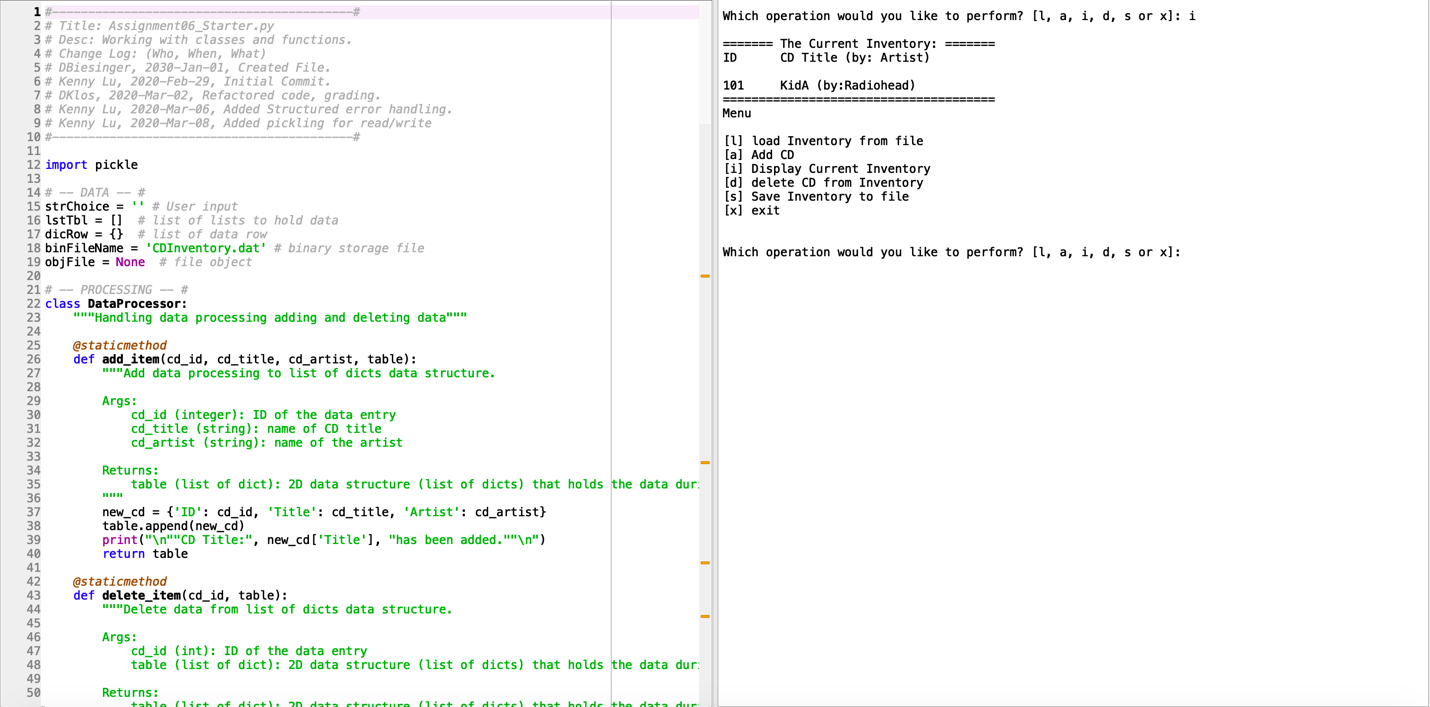


Figure 4 – Verify CDInventory.dat has saved inventory

  
Figure 5 – Example output from script display inventory in memory

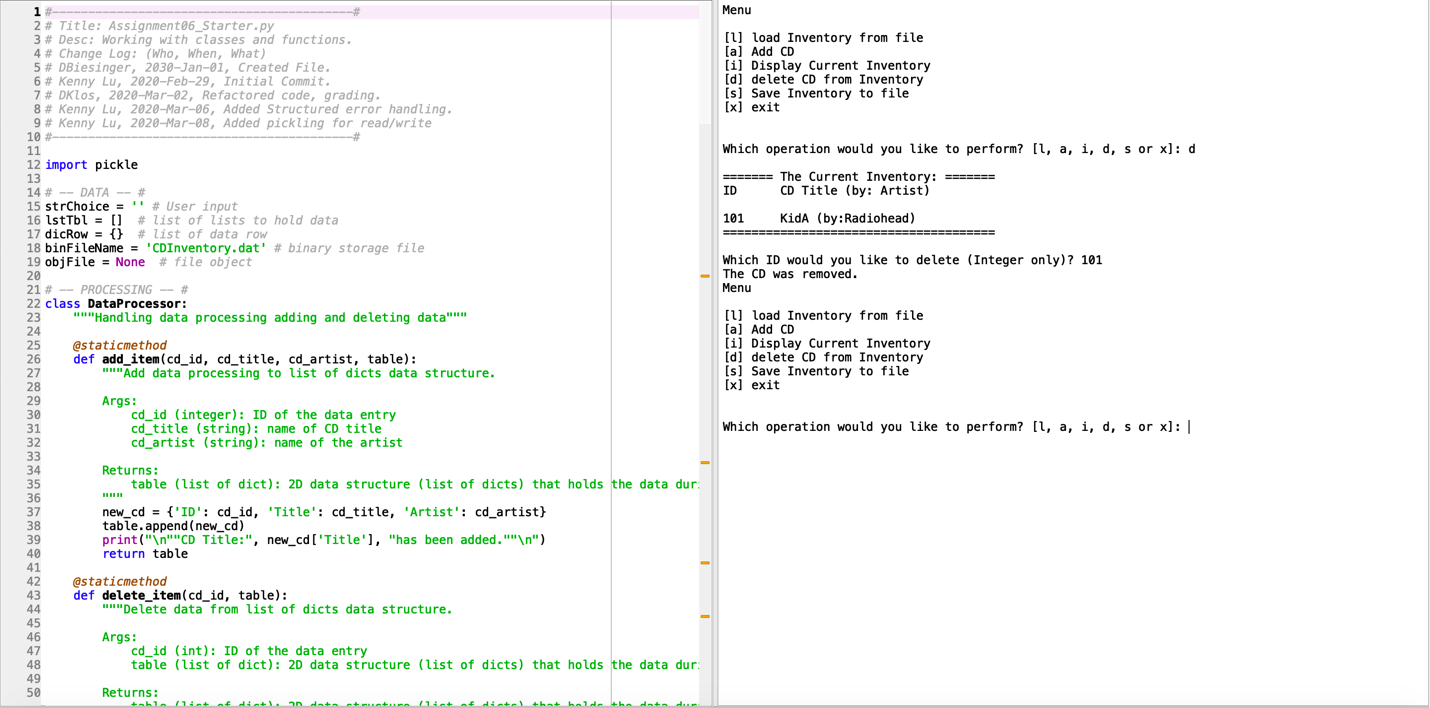


Figure 6 – Example output from script delete data from memory

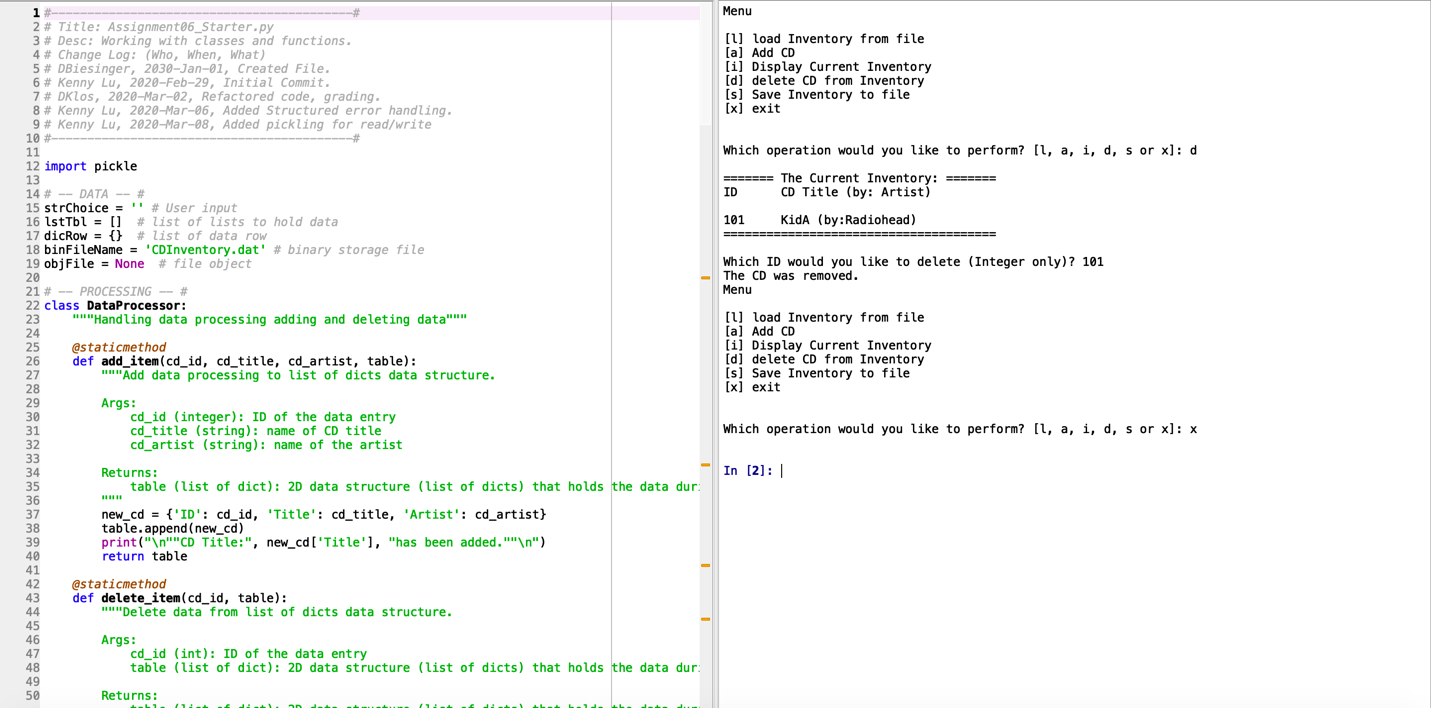


Figure 7 – Example output from script exit program

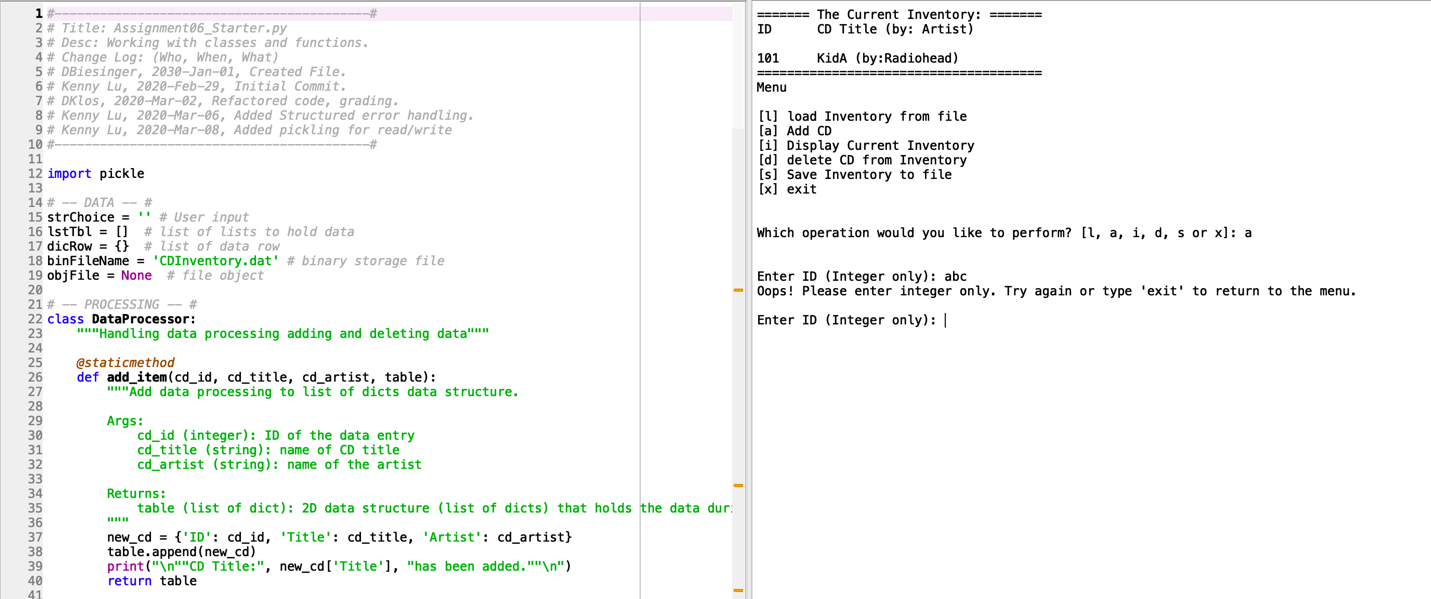


Figure 8 – Example output from script input ID non-integer with create

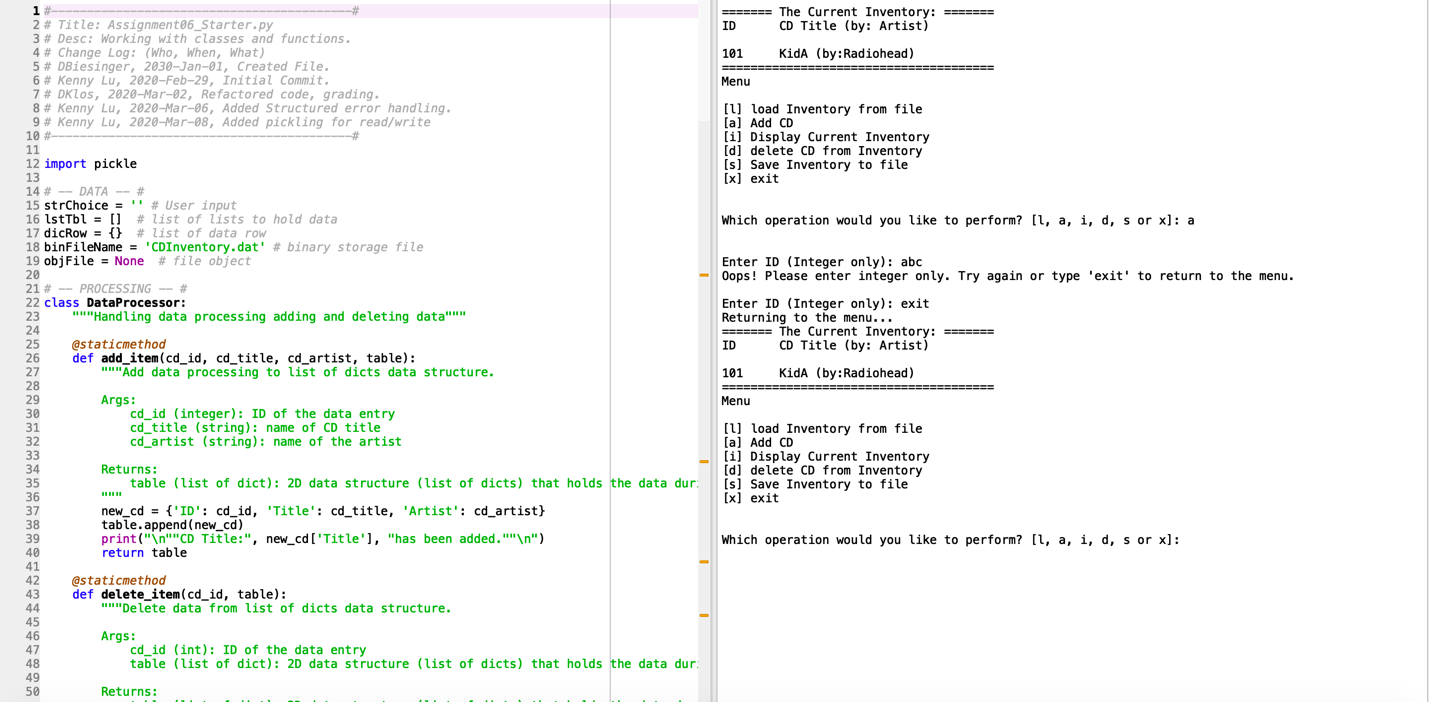


Figure 9 – Example output from script exit out of the loop when non-integer entered

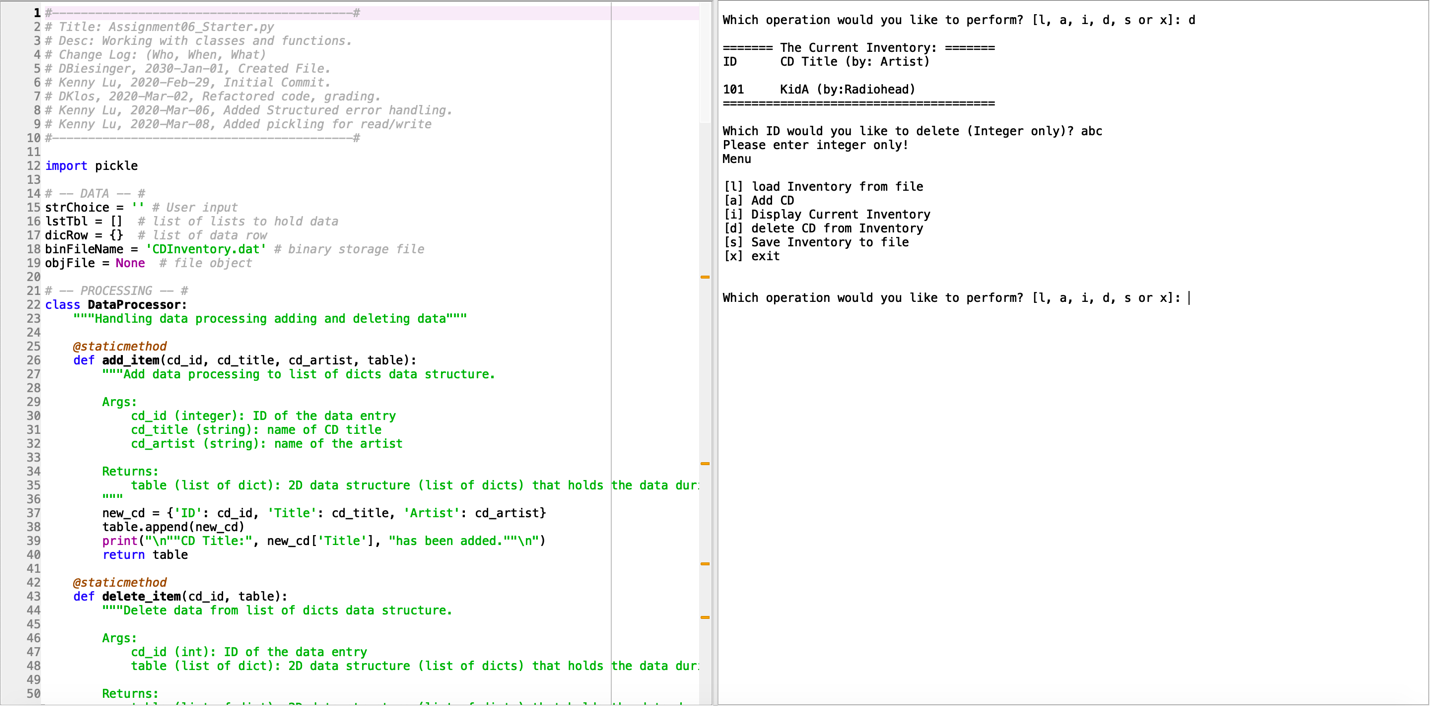


Figure 10 – Example output from script input ID non-integer with delete

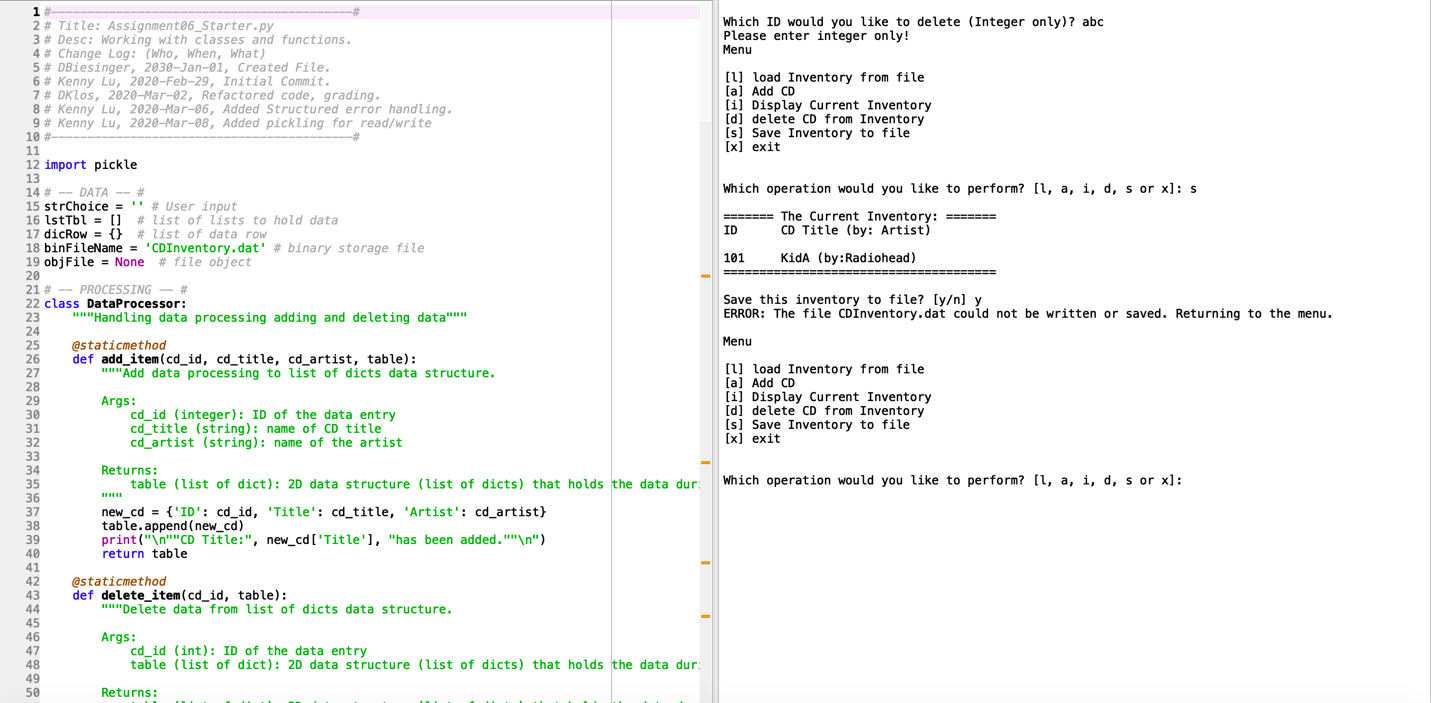


Figure 11 – Example output from script save operation cannot be performed permission error

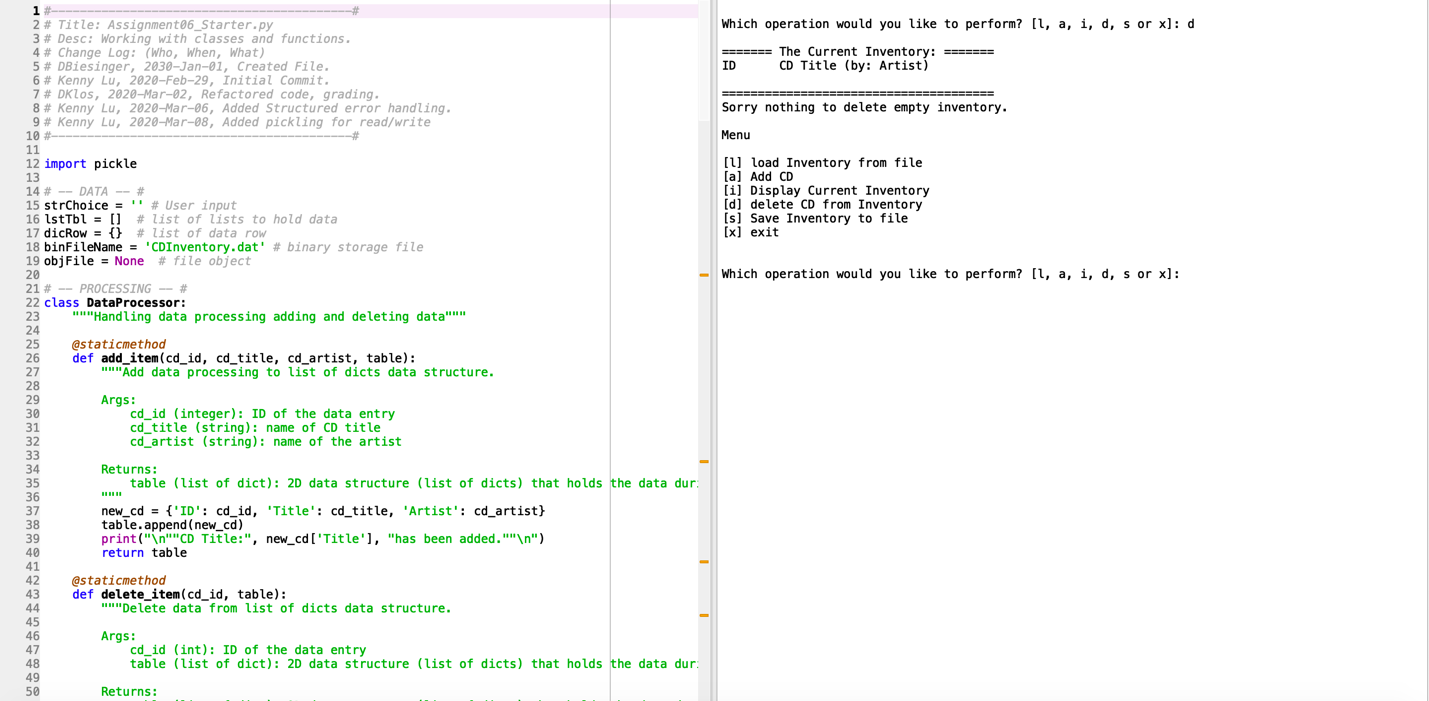


Figure 12 – Example output from script when there’s nothing in the inventory to delete

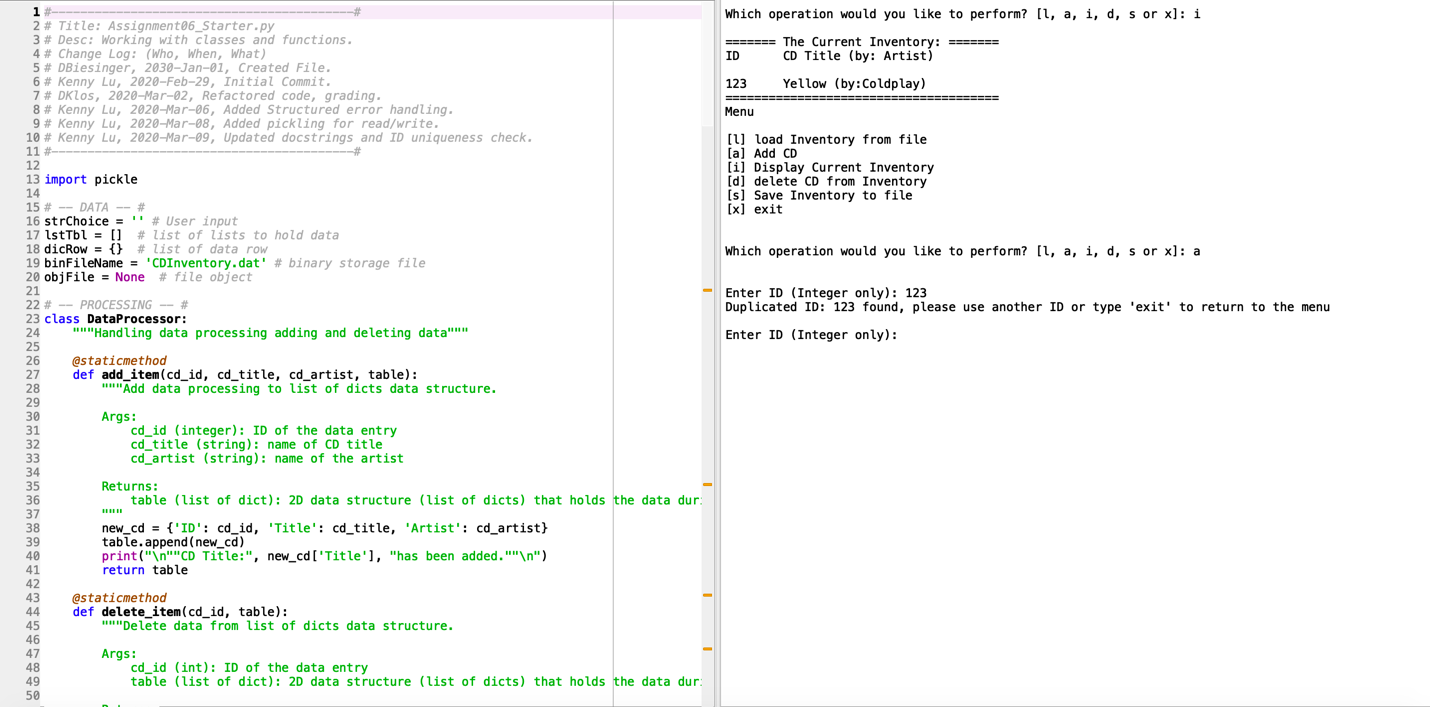


Figure 13 – Example output from script check against duplicate ID

1. #------------------------------------------#
2. # Title: Assignment06\_Starter.py
3. # Desc: Working with classes and functions.
4. # Change Log: (Who, When, What)
5. # DBiesinger, 2030-Jan-01, Created File.
6. # Kenny Lu, 2020-Feb-29, Initial Commit.
7. # DKlos, 2020-Mar-02, Refactored code, grading.
8. # Kenny Lu, 2020-Mar-06, Added Structured error handling.
9. # Kenny Lu, 2020-Mar-08, Added pickling for read/write.
10. # Kenny Lu, 2020-Mar-09, Updated docstrings.
11. #------------------------------------------#
13. **import** pickle
15. # -- DATA -- #
16. strChoice = '' # User input
17. lstTbl = []  # list of lists to hold data
18. dicRow = {}  # list of data row
19. binFileName = 'CDInventory.dat' # binary storage file
20. objFile = None  # file object
22. # -- PROCESSING -- #
23. **class** DataProcessor:
24. """Handling data processing adding and deleting data"""
26. @staticmethod
27. **def** add\_item(cd\_id, cd\_title, cd\_artist, table):
28. """Add data processing to list of dicts data structure.
30. Args:
31. cd\_id (integer): ID of the data entry
32. cd\_title (string): name of CD title
33. cd\_artist (string): name of the artist
35. Returns:
36. table (list of dict): 2D data structure (list of dicts) that holds the data during runtime
37. """
38. new\_cd = {'ID': cd\_id, 'Title': cd\_title, 'Artist': cd\_artist}
39. table.append(new\_cd)
40. **print**("\n""CD Title:", new\_cd['Title'], "has been added.""\n")
41. **return** table
43. @staticmethod
44. **def** delete\_item(cd\_id, table):
45. """Delete data from list of dicts data structure.
47. Args:
48. cd\_id (int): ID of the data entry
49. table (list of dict): 2D data structure (list of dicts) that holds the data during runtime
51. Returns:
52. table (list of dict): 2D data structure (list of dicts) that holds the data during runtime
54. """
55. intRowNr = -1
56. blnCDRemoved = False
57. **for** row **in** table:
58. intRowNr += 1
59. **if** row['ID'] == cd\_id:
60. **del** table[intRowNr]
61. blnCDRemoved = True
62. **break**
63. **if** blnCDRemoved:
64. **print**('The CD was removed.')
65. **else**:
66. **print**('Could not find this CD!')
68. **return** table
70. **class** FileProcessor:
71. """Processing the data to and from text file"""
73. @staticmethod
74. **def** read\_file(file\_name, table):
75. """Function to manage data ingestion from file to a list of dictionaries
77. Unpickle the data from binary file identified by file\_name
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. table (list of dict): 2D data structure (list of dicts) that holds the data during runtime
85. """
86. # Try-except to not crash the program if target file doesn't exist
87. **try**:
88. with open(binFileName, 'rb') as fileObj:
89. table = pickle.load(fileObj)
90. **except** FileNotFoundError:
91. **print**("The file {} could not be loaded".format(file\_name))
93. **return** table

96. @staticmethod
97. **def** write\_file(file\_name, table):
98. # def write\_file(file\_name, table):
99. """Function to write data in memory to file
101. Pickle the data from memory and write to a binary data storage file
103. Args:
104. file\_name (string): name of file used to read the data from
105. table (list of dict): 2D data structure (list of dicts) that holds the data during runtime
107. Returns:
108. None.
109. """
110. # Try-except to not crash the program if target file cannot be written or saved
111. **try**:
112. with open(binFileName, "wb") as fileObj:
113. pickle.dump(lstTbl, fileObj)
114. **except** IOError as e:
115. **print**("ERROR: The file {} could not be written or saved. Returning to the menu.".format(file\_name) + '\n')
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 string 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
157. Args:
158. table (list of dict): 2D data structure (list of dicts) that holds the data during runtime.
160. Returns:
161. None.
163. """
164. **print**('======= The Current Inventory: =======')
165. **print**('ID\tCD Title (by: Artist)\n')
166. **for** row **in** table:
167. **print**('{}\t{} (by:{})'.format(\*row.values()))
168. **print**('======================================')
170. @staticmethod
171. **def** add\_data():
172. """Get data from user to be added into list of dicts data structure.
174. Args:
175. None.
177. Returns:
178. cd\_id (integer): ID of the new CD
179. cd\_title (string): Title of the new CD
180. cd\_artist (string): Artist of the new CD
182. """
183. **while** True:
184. str\_cd\_id = input('Enter ID (Integer only): ').strip()
185. # Allow user to input 'exit' to get out of the loop
186. **if** str\_cd\_id.lower() == 'exit':
187. **break**
188. # Try-except to ensure input is integer
189. **try**:
190. cd\_id = int(str\_cd\_id)
191. **except** ValueError as e:
192. **print**("Oops! Please enter integer only. Try again or type \'exit\' to return to the menu.")
193. **continue**
194. cd\_title = input('What is the CD\'s title? ').strip()
195. cd\_artist = input('What is the Artist\'s name? ').strip()
196. **return** cd\_id, cd\_title, cd\_artist
198. # 1. When program starts, read in the currently saved Inventory
199. lstTbl = FileProcessor.read\_file(binFileName, lstTbl)
201. # 2. start main loop
202. **while** True:
203. # 2.1 Display Menu to user and get choice
204. IO.print\_menu()
205. strChoice = IO.menu\_choice()
207. # 3. Process menu selection
208. # 3.1 process exit first
209. **if** strChoice == 'x':
210. **break**
211. # 3.2 process load inventory
212. **if** strChoice == 'l':
213. **print**('WARNING: If you continue, all unsaved data will be lost and the Inventory re-loaded from file.')
214. strYesNo = input('type \'yes\' to continue and reload from file. otherwise reload will be canceled')
215. **if** strYesNo.lower() == 'yes':
216. **print**('reloading...')
217. lstTbl = FileProcessor.read\_file(binFileName, lstTbl)
218. IO.show\_inventory(lstTbl)
219. **else**:
220. input('canceling... Inventory data NOT reloaded. Press [ENTER] to continue to the menu.')
221. IO.show\_inventory(lstTbl)
222. **continue**  # start loop back at top.
223. # 3.3 process add a CD
224. **elif** strChoice == 'a':
225. # 3.3.1 Ask user for new ID, CD Title and Artist
226. # 3.3.2 Add item to the table
227. # Try-except in case user wants to exit out of the loop
228. **try**:
229. lstTbl = DataProcessor.add\_item(\*IO.add\_data(), lstTbl)
230. **except** Exception as e:
231. **print**("Returning to the menu...")
232. IO.show\_inventory(lstTbl)
233. **continue**  # start loop back at top.
234. # 3.4 process display current inventory
235. **elif** strChoice == 'i':
236. IO.show\_inventory(lstTbl)
237. **continue**  # start loop back at top.
238. # 3.5 process delete a CD
239. **elif** strChoice == 'd':
240. # 3.5.1 get Userinput for which CD to delete
241. # 3.5.1.1 display Inventory to user
242. IO.show\_inventory(lstTbl)
243. # 3.5.1.2 ask user which ID to remove
244. # Check if inventory is empty, no need to delete if there's nothing in inventory
245. **if** len(lstTbl) != 0:
246. strIDDel = input('Which ID would you like to delete (Integer only)? ').strip()
247. # 3.5.2 search thru table and delete CD
248. # Try-except to ensure input is integer
249. **try**:
250. intIDDel = int(strIDDel)
251. **except** ValueError as e:
252. **print**("Please enter integer only!")
253. **continue**
254. lstTbl = DataProcessor.delete\_item(intIDDel, lstTbl)
255. **else**:
256. **print**("Sorry nothing to delete empty inventory." '\n')
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. FileProcessor.write\_file(binFileName, lstTbl)
267. **else**:
268. input('The inventory was NOT saved to file. Press [ENTER] to return to the menu.')
269. **continue**  # start loop back at top.
270. # 3.7 catch-all should not be possible, as user choice gets vetted in IO, but to be save:
271. **else**:
272. **print**('General Error')

Figure 14 – CDInventory.py