Bill McGinty

March 14, 2022

Foundations of Programming: Python

Assignment 07

GitHub Link: https://github.com/wwm787/Assignment\_07

# Working With Binary Files and Structure Error Handling

# Introduction

In this assignment, I learned the benefits of using structured error handling and the difference between reading and writing to a text file vs a binary file. Details are contained in the summary below.

# Assignment 07 – CD Inventory Program Using Binary Files With Structured Error Handling

In assignment 07, the script allows the user to inventory a CD collection. The program asks the user to either load data from a binary file, add item to memory, view inventory, delete an item from memory, save to binary file or exit the program. It is like assignment 05, except the IO and data processing are transferred to functions. And, it is like assignment 06, except reading and writing use binary file format

* Lines 1 -- 2: Import sys library to gain access to exit() function and pickle library
* Lines 4 – 9: Define variables, list of lists, dictionary and binary file name
* Lines 12 – 65: Group data processing functions into a class
  + Functions
    - Custom error class
    - Add a record containing ID, Title and Artist parameter and loads it into table
    - Delete a record based on ID parameter, called when user selects ‘d’ in menu
* Lines 68 – 126: Group binary file read and write functions into a class
  + Functions
    - Read binary file that is called at program startup and when inventory is selected in menu
      * Added structured error handling to check for missing files
    - Write to binary file that is called when save is selected from the menu
      * Added structured error handling to check for missing files
* Lines 131 – 230: Group IO functions into a class
  + Functions
    - Display menu at startup
    - Get user menu choice
    - Show current inventory in memory (not saved to binary file)
    - Add function when user selects ‘a’ in program menu that allows the user to enter a new CD ID, Title and Artist via input command
      * Added structured error handling to check for
        + The ID is an integer between 1 and 500
        + Title and Artist are not blank
* Line 234: Load currently saved inventory from binary file at startup
* Lines 237 – 297: Series of if, elif and else statements to process menu items and call the specified data processing, read/write or IO functions
  + Choices are the exit the program, load from binary file, view inventory, add a record, delete a record or save to binary file.
  + For choice ‘d’ delete, added structured error handling to ensure only integers are entered

1. **import** sys
2. **import** pickle
4. # -- DATA -- #
5. strChoice **=** ""  # User input
6. lstTbl **=** []  # list of lists to hold data
7. dicRow **=** {}  # list of data row
8. strFileName **=** "CDInventory.dat"  # binary data storage file
9. objFile **=** None  # file object

12. **class** CDIDoutOfRange(Exception):
13. """ custom error class for value not within upper or lower bounds """
15. **def** \_\_str\_\_(self):
16. **return** "CD ID must be greater than 0 and less than 500!"
18. # -- PROCESSING -- #
20. **class** DataProcessor:
22. @staticmethod
23. **def** myAddProcCode(myID, myTitle, myArtist):
24. """ Function to process ID, Title and Artist
26. Args:
27. myID (string): ID of CD.
28. myTitle (string): Title of CD.
29. myArtist (string): Artist name.
31. Returns:
32. None.
34. """
35. # Add item to the table
36. intID **=** int(myID)
37. dicRow **=** {"ID": intID, "Title": myTitle, "Artist": myArtist}
38. lstTbl.append(dicRow)
39. IO.show\_inventory(lstTbl)
41. **def** myDeleteDataProcFunc(intIDDelReceived):
42. """ Function to delete CD based on ID passed to function
44. Args:
45. intIDDelReceived (int): ID of CD to delete.
47. Returns:
48. None.
50. """
51. # search thru table and delete CD
52. intRowNr **=** **-**1
53. blnCDRemoved **=** False
54. **for** row **in** lstTbl:
55. intRowNr **+=** 1
56. **if** row["ID"] **==** intIDDelReceived:
57. **del** lstTbl[intRowNr]
58. blnCDRemoved **=** True
59. **break**
60. **if** blnCDRemoved:
61. print("The CD was removed")
62. **else**:
63. print("\nCould not find this CD!!!\n")
64. IO.show\_inventory(lstTbl)  # display inventory
65. **return**

68. **class** FileProcessor:
69. """Processing the data to and from binary file"""
71. @staticmethod
72. **def** read\_file(file\_name, startTbl):
73. """Function to manage data ingestion from file to a list of dictionaries
75. Reads the data from file identified by file\_name into a 2D table
76. (list of dicts) table one line in the file represents one dictionary row in table.
78. Structure error handlind added for missing read file
80. Args:
81. file\_name (string): name of file used to read the data from
82. table (list of dictionary): 2D data structure (list of dicts) that holds the data during runtime
84. Returns:
85. None.
86. """
87. **try**: #structured error handling, display error if file not found
88. startTbl.clear()  # this clears existing data and allows to load data from file
89. billStrIncr **=** **-**1  # disctionary row increment set to zero
90. fileObj **=** open(file\_name, "rb")  # open .dat in read binary mode
91. billString **=** pickle.load(fileObj)  # set list billString equal to all pickled objects
92. **for** row **in** billString:  # parse all rows in list
93. billStrIncr **+=** 1
94. startTbl.append(billString[billStrIncr]) # appends dictionary row to list of lists
95. fileObj.close()
96. **except** FileNotFoundError as e: #detailed error information
97. print("\nYou need to create a CDInventory.dat file first!")
98. print("Build in error info: ")
99. print(type(e), e, e.\_\_doc\_\_, sep**=**"\n")
100. print("Exiting Program\n")
101. sys.exit()
103. @staticmethod
104. **def** write\_file(file\_name, recTbl):  # save data
105. """ Function to save table data to binary file
107. Structure error handlind added for missing write file
108. Args:
109. file\_name (string): name of the file used to write data to.
110. recTbl (list of dictionary): 2D data structure (list of dicts) that holds the data during runtime.
112. Returns:
113. None.
115. """
116. **try**: #structured error handling, display error if file not found
117. fileObj **=** open(file\_name, "wb")  # open .dat in write binary mode
118. # write the pickled list of lists to binary file
119. pickle.dump(recTbl, fileObj) # write picked data to binary file
120. fileObj.close()
121. **except** FileNotFoundError as e: #detailed error information
122. print("\nCDInventory.dat is missing!")
123. print("Build in error info: ")
124. print(type(e), e, e.\_\_doc\_\_, sep**=**"\n")
125. print("Exiting Program\n")
126. sys.exit()
128. # -- PRESENTATION (Input/Output) -- #

131. **class** IO:
132. """Handling Input / Output"""
134. @staticmethod
135. **def** print\_menu():
136. """Displays a menu of choices to the user
138. Args:
139. None.
141. Returns:
142. None.
143. """
145. print(
146. "Menu\n\n[l] load Inventory from file\n[a] Add CD\n[i] Display Current Inventory")
147. print("[d] delete CD from Inventory\n[s] Save Inventory to file\n[x] exit\n")
149. @staticmethod
150. **def** menu\_choice():
151. """Gets user input for menu selection
153. Args:
154. None.
156. Returns:
157. choice (string): a lower case sting of the users input out of the choices l, a, i, d, s or x
159. """
160. choice **=** " "
161. **while** choice **not** **in** ["l", "a", "i", "d", "s", "x"]:
162. choice **=** input(
163. "Which operation would you like to perform? [l, a, i, d, s or x]: ").lower().strip()
164. print()  # Add extra space for layout
165. **return** choice
167. @staticmethod
168. **def** show\_inventory(invTbl):
169. """Displays current inventory of table invTbl

172. Args:
173. invTbl (list of dict): 2D data structure (list of dicts) that holds the data during runtime.
175. Returns:
176. None.
178. """
179. print("======= The Current Inventory: =======")
180. print("ID\tCD Title (by: Artist)\n")
181. **for** row **in** invTbl:
182. print("{}\t{} (by:{})".format(**\***row.values()))
183. print("======================================")
185. @staticmethod
186. **def** myAddIOFunc():
187. """ Function for input / ouput
188. Ask the user CD ID, Title and Artist
190. Returns:
191. strID1 (string): User inputted CD ID.
192. strTitle1 (string): User inputted CD Title.
193. strArtist1 (string): User inputted CD Artist.
195. """
196. **while** True:
197. strID1 **=** input("Enter ID: ").strip()
198. **try**: #check if integer and within range
199. myTmpID1 **=** int(strID1)
200. **if** **not** 0 < myTmpID1 < 500:
201. **raise** print(CDIDoutOfRange())
202. **except** ValueError as e:
203. print("\n")
204. print("That is not an Integer!")
205. print("Build in error info: ")
206. print(type(e), e, e.\_\_doc\_\_, sep**=**"\n")
207. **continue**
208. **except** Exception:
209. **continue**
210. **while** True:
211. strTitle1 **=** input("What is the CDs title? ").strip()
212. **try**: # if blank raise error and start over
213. **if** len(strTitle1) **==** 0:
214. **raise** ValueError("You must enter a Title!")
215. **except** ValueError as e:
216. print("\n")
217. print("Build in error info: ")
218. print(type(e), e, e.\_\_doc\_\_, sep**=**"\n")
219. **continue**
220. **while** True:
221. strArtist1 **=** input("What is the Artist\"s name? ").strip()
222. **try**: # if blank raise error and start over
223. **if** len(strArtist1) **==** 0:
224. **raise** ValueError("You must enter an Artist!")
225. **except** ValueError as e:
226. print("\n")
227. print("Build in error info: ")
228. print(type(e), e, e.\_\_doc\_\_, sep**=**"\n")
229. **continue**
230. **return** strID1, strTitle1, strArtist1

233. # When program starts, read in the currently saved Inventory
234. FileProcessor.read\_file(strFileName, lstTbl)
236. # Start main loop
237. **while** True:
238. # Display Menu to user and get choice
239. IO.print\_menu()
240. strChoice **=** IO.menu\_choice()
242. # Process menu selection
243. # Process exit first
244. **if** strChoice **==** "x":
245. **break**
246. # Process load inventory
247. **if** strChoice **==** "l":
248. print("WARNING: If you continue, all unsaved data will be lost and the Inventory re-loaded from file.")
249. strYesNo **=** input(
250. "type \"yes\" to continue and reload from file. otherwise reload will be canceled: ")
251. **if** strYesNo.lower() **==** "yes":
252. print("reloading...")
253. FileProcessor.read\_file(strFileName, lstTbl)
254. IO.show\_inventory(lstTbl)
255. **else**:
256. input(
257. "canceling... Inventory data NOT reloaded. Press [ENTER] to continue to the menu.")
258. IO.show\_inventory(lstTbl)
259. **continue**  # start loop back at top.
260. # Process add a CD
261. **elif** strChoice **==** "a":
262. # Ask user for new ID, CD Title and Artist
263. strID, strTitle, strArtist **=** IO.myAddIOFunc()
264. DataProcessor.myAddProcCode(strID, strTitle, strArtist)
265. **continue**  # start loop back at top.
266. # Process display current inventory
267. **elif** strChoice **==** "i":
268. IO.show\_inventory(lstTbl)
269. **continue**  # start loop back at top.
270. **elif** strChoice **==** "d":  # process delete a CD
271. IO.show\_inventory(lstTbl)  # display inventory
272. # Get Userinput for which CD to delete
273. **try**:
274. intIDDelInput **=** int(
275. input("Which ID would you like to delete? ").strip())
276. **except** ValueError as e:
277. print("\n")
278. print("That is not an Integer!")
279. print("Build in error info: ")
280. print(type(e), e, e.\_\_doc\_\_, sep**=**"\n")
281. **continue**
282. DataProcessor.myDeleteDataProcFunc(intIDDelInput)
283. **continue**  # start loop back at top.
284. **elif** strChoice **==** "s":  # process save inventory to file
285. # Display current inventory and ask user for confirmation to save
286. IO.show\_inventory(lstTbl)
287. strYesNo **=** input("Save this inventory to file? [y/n] ").strip().lower()
288. # Process choice
289. **if** strYesNo **==** "y":
290. FileProcessor.write\_file(strFileName, lstTbl)
291. **else**:
292. input(
293. "The inventory was NOT saved to file. Press [ENTER] to return to the menu.")
294. **continue**  # start loop back at top.
295. # Catch-all should not be possible, as user choice gets vetted in IO, but to be save:
296. **else**:
297. print("General Error")

Listing - CDInventory.py code

|  |  |
| --- | --- |
| Figure - CDInventory.py Spyder Demo | Figure - CDInventory.py Spyder Demo |

|  |  |
| --- | --- |
| Figure - CDInventory.py Terminal Window Demo | Figure - CDInventory.py Terminal Window Demo |

# LAB 07-A: Working with Files and Lists:

To handle two numbers on one input line use the split() function (see example below).

strNumA, strNumB = input("enter two numbers: ").split()

# LAB 07-B: Working with Files and Dictionaries:

Binary is useful when the user wants to store data in a more compact way that is closer to the way a computer processes data and is therefore more efficient.

# LAB 07-C: Working with Files and Dictionaries:

The addition of structured error handling in this lab allows the user to view details of an error without stopping or breaking the code processing. For example, without structured error handling, if the user entered a string where text is required the code would crash and break without details. With structured error handling, the user is given error details and told to try again without stopping the code from running.

# Summary

Regarding text vs binary files, binary file structure is useful when it is required to store complex data in files. Whereas text files are easier for humans to read and understand, they require additional memory / processor intensive pre and post formatting. Binary files are like the way computers store memory and are therefore more efficient in processing and memory. Structured error handling in the form of Try: and Except: statements have numerous benefits over allowing the code to crash. Without structured error handling, even the slightest mistake will stop the program leaving the user guessing as to the cause. With structured error handling, the user can trap the error, retrieve the details, notify the user of the issue and prompt the user for a correction while the program is still running. To borrow from Dirk’s module, structured error handling makes sense when there are interactions with humans, files or situation where programmers may interact with your code.

# Appendix

## Listing CDInventory.py

1. #------------------------------------------#
2. # Title: CDInventory.py
3. # Desc: Working with binary files and structured error handling
4. # Change Log: (Who, When, What)
5. # DBiesinger, 2030-Jan-01, Created File
6. # Bill McGinty, 2022-Mar-14, Modified File to add functionality
7. #------------------------------------------#
9. **import** sys
10. **import** pickle
12. # -- DATA -- #
13. strChoice **=** ""  # User input
14. lstTbl **=** []  # list of lists to hold data
15. dicRow **=** {}  # list of data row
16. strFileName **=** "CDInventory.dat"  # binary data storage file
17. objFile **=** None  # file object

20. **class** CDIDoutOfRange(Exception):
21. """ custom error class for value not within upper or lower bounds """
23. **def** \_\_str\_\_(self):
24. **return** "CD ID must be greater than 0 and less than 500!"
26. # -- PROCESSING -- #
28. **class** DataProcessor:
30. @staticmethod
31. **def** myAddProcCode(myID, myTitle, myArtist):
32. """ Function to process ID, Title and Artist
34. Args:
35. myID (string): ID of CD.
36. myTitle (string): Title of CD.
37. myArtist (string): Artist name.
39. Returns:
40. None.
42. """
43. # Add item to the table
44. intID **=** int(myID)
45. dicRow **=** {"ID": intID, "Title": myTitle, "Artist": myArtist}
46. lstTbl.append(dicRow)
47. IO.show\_inventory(lstTbl)
49. **def** myDeleteDataProcFunc(intIDDelReceived):
50. """ Function to delete CD based on ID passed to function
52. Args:
53. intIDDelReceived (int): ID of CD to delete.
55. Returns:
56. None.
58. """
59. # search thru table and delete CD
60. intRowNr **=** **-**1
61. blnCDRemoved **=** False
62. **for** row **in** lstTbl:
63. intRowNr **+=** 1
64. **if** row["ID"] **==** intIDDelReceived:
65. **del** lstTbl[intRowNr]
66. blnCDRemoved **=** True
67. **break**
68. **if** blnCDRemoved:
69. print("The CD was removed")
70. **else**:
71. print("\nCould not find this CD!!!\n")
72. IO.show\_inventory(lstTbl)  # display inventory
73. **return**

76. **class** FileProcessor:
77. """Processing the data to and from binary file"""
79. @staticmethod
80. **def** read\_file(file\_name, startTbl):
81. """Function to manage data ingestion from file to a list of dictionaries
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.
86. Structure error handlind added for missing read file
88. Args:
89. file\_name (string): name of file used to read the data from
90. table (list of dictionary): 2D data structure (list of dicts) that holds the data during runtime
92. Returns:
93. None.
94. """
95. **try**: #structured error handling, display error if file not found
96. startTbl.clear()  # this clears existing data and allows to load data from file
97. billStrIncr **=** **-**1  # disctionary row increment set to zero
98. fileObj **=** open(file\_name, "rb")  # open .dat in read binary mode
99. billString **=** pickle.load(fileObj)  # set list billString equal to all pickled objects
100. **for** row **in** billString:  # parse all rows in list
101. billStrIncr **+=** 1
102. startTbl.append(billString[billStrIncr]) # appends dictionary row to list of lists
103. fileObj.close()
104. **except** FileNotFoundError as e: #detailed error information
105. print("\nYou need to create a CDInventory.dat file first!")
106. print("Build in error info: ")
107. print(type(e), e, e.\_\_doc\_\_, sep**=**"\n")
108. print("Exiting Program\n")
109. sys.exit()
111. @staticmethod
112. **def** write\_file(file\_name, recTbl):  # save data
113. """ Function to save table data to binary file
115. Structure error handlind added for missing write file
116. Args:
117. file\_name (string): name of the file used to write data to.
118. recTbl (list of dictionary): 2D data structure (list of dicts) that holds the data during runtime.
120. Returns:
121. None.
123. """
124. **try**: #structured error handling, display error if file not found
125. fileObj **=** open(file\_name, "wb")  # open .dat in write binary mode
126. # write the pickled list of lists to binary file
127. pickle.dump(recTbl, fileObj) # write picked data to binary file
128. fileObj.close()
129. **except** FileNotFoundError as e: #detailed error information
130. print("\nCDInventory.dat is missing!")
131. print("Build in error info: ")
132. print(type(e), e, e.\_\_doc\_\_, sep**=**"\n")
133. print("Exiting Program\n")
134. sys.exit()
136. # -- PRESENTATION (Input/Output) -- #

139. **class** IO:
140. """Handling Input / Output"""
142. @staticmethod
143. **def** print\_menu():
144. """Displays a menu of choices to the user
146. Args:
147. None.
149. Returns:
150. None.
151. """
153. print(
154. "Menu\n\n[l] load Inventory from file\n[a] Add CD\n[i] Display Current Inventory")
155. print("[d] delete CD from Inventory\n[s] Save Inventory to file\n[x] exit\n")
157. @staticmethod
158. **def** menu\_choice():
159. """Gets user input for menu selection
161. Args:
162. None.
164. Returns:
165. choice (string): a lower case sting of the users input out of the choices l, a, i, d, s or x
167. """
168. choice **=** " "
169. **while** choice **not** **in** ["l", "a", "i", "d", "s", "x"]:
170. choice **=** input(
171. "Which operation would you like to perform? [l, a, i, d, s or x]: ").lower().strip()
172. print()  # Add extra space for layout
173. **return** choice
175. @staticmethod
176. **def** show\_inventory(invTbl):
177. """Displays current inventory of table invTbl

180. Args:
181. invTbl (list of dict): 2D data structure (list of dicts) that holds the data during runtime.
183. Returns:
184. None.
186. """
187. print("======= The Current Inventory: =======")
188. print("ID\tCD Title (by: Artist)\n")
189. **for** row **in** invTbl:
190. print("{}\t{} (by:{})".format(**\***row.values()))
191. print("======================================")
193. @staticmethod
194. **def** myAddIOFunc():
195. """ Function for input / ouput
196. Ask the user CD ID, Title and Artist
198. Returns:
199. strID1 (string): User inputted CD ID.
200. strTitle1 (string): User inputted CD Title.
201. strArtist1 (string): User inputted CD Artist.
203. """
204. **while** True:
205. strID1 **=** input("Enter ID: ").strip()
206. **try**: #check if integer and within range
207. myTmpID1 **=** int(strID1)
208. **if** **not** 0 < myTmpID1 < 500:
209. **raise** print(CDIDoutOfRange())
210. **except** ValueError as e:
211. print("\n")
212. print("That is not an Integer!")
213. print("Build in error info: ")
214. print(type(e), e, e.\_\_doc\_\_, sep**=**"\n")
215. **continue**
216. **except** Exception:
217. **continue**
218. **while** True:
219. strTitle1 **=** input("What is the CDs title? ").strip()
220. **try**: # if blank raise error and start over
221. **if** len(strTitle1) **==** 0:
222. **raise** ValueError("You must enter a Title!")
223. **except** ValueError as e:
224. print("\n")
225. print("Build in error info: ")
226. print(type(e), e, e.\_\_doc\_\_, sep**=**"\n")
227. **continue**
228. **while** True:
229. strArtist1 **=** input("What is the Artist\"s name? ").strip()
230. **try**: # if blank raise error and start over
231. **if** len(strArtist1) **==** 0:
232. **raise** ValueError("You must enter an Artist!")
233. **except** ValueError as e:
234. print("\n")
235. print("Build in error info: ")
236. print(type(e), e, e.\_\_doc\_\_, sep**=**"\n")
237. **continue**
238. **return** strID1, strTitle1, strArtist1

241. # When program starts, read in the currently saved Inventory
242. FileProcessor.read\_file(strFileName, lstTbl)
244. # Start main loop
245. **while** True:
246. # Display Menu to user and get choice
247. IO.print\_menu()
248. strChoice **=** IO.menu\_choice()
250. # Process menu selection
251. # Process exit first
252. **if** strChoice **==** "x":
253. **break**
254. # Process load inventory
255. **if** strChoice **==** "l":
256. print("WARNING: If you continue, all unsaved data will be lost and the Inventory re-loaded from file.")
257. strYesNo **=** input(
258. "type \"yes\" to continue and reload from file. otherwise reload will be canceled: ")
259. **if** strYesNo.lower() **==** "yes":
260. print("reloading...")
261. FileProcessor.read\_file(strFileName, lstTbl)
262. IO.show\_inventory(lstTbl)
263. **else**:
264. input(
265. "canceling... Inventory data NOT reloaded. Press [ENTER] to continue to the menu.")
266. IO.show\_inventory(lstTbl)
267. **continue**  # start loop back at top.
268. # Process add a CD
269. **elif** strChoice **==** "a":
270. # Ask user for new ID, CD Title and Artist
271. strID, strTitle, strArtist **=** IO.myAddIOFunc()
272. DataProcessor.myAddProcCode(strID, strTitle, strArtist)
273. **continue**  # start loop back at top.
274. # Process display current inventory
275. **elif** strChoice **==** "i":
276. IO.show\_inventory(lstTbl)
277. **continue**  # start loop back at top.
278. **elif** strChoice **==** "d":  # process delete a CD
279. IO.show\_inventory(lstTbl)  # display inventory
280. # Get Userinput for which CD to delete
281. **try**:
282. intIDDelInput **=** int(
283. input("Which ID would you like to delete? ").strip())
284. **except** ValueError as e:
285. print("\n")
286. print("That is not an Integer!")
287. print("Build in error info: ")
288. print(type(e), e, e.\_\_doc\_\_, sep**=**"\n")
289. **continue**
290. DataProcessor.myDeleteDataProcFunc(intIDDelInput)
291. **continue**  # start loop back at top.
292. **elif** strChoice **==** "s":  # process save inventory to file
293. # Display current inventory and ask user for confirmation to save
294. IO.show\_inventory(lstTbl)
295. strYesNo **=** input("Save this inventory to file? [y/n] ").strip().lower()
296. # Process choice
297. **if** strYesNo **==** "y":
298. FileProcessor.write\_file(strFileName, lstTbl)
299. **else**:
300. input(
301. "The inventory was NOT saved to file. Press [ENTER] to return to the menu.")
302. **continue**  # start loop back at top.
303. # Catch-all should not be possible, as user choice gets vetted in IO, but to be save:
304. **else**:
305. print("General Error")