Nitasha Woodward

November 28, 2021

IT FDN 110 B Au 21: Foundations of Programming: Python

Assignment 07

Working with Error Handling and Binary Files

# Introduction

In this assignment I will explain the steps I used to modify Assignment 06 CD Inventory program to add structured error handling and store data in a binary format versus text. This program incorporates reading and writing data to and from binary files, stress testing the code to identify areas where error handling should be added and modifying preexisting docstrings.

# Creating the Code

### Adding Error Handling before changing data storage format

I first Looked for places in the code where there was IO and data conversion [Ex: string to integer] in order to identify places where errors could occur and try/except statements should be introduced to handle such errors. I started with the DataProcessor class and first tried to add error handling for the ID, strID, conversion to an integer in add\_cd function. However, I realized that if I waited to do the error handling in the add\_cd function, it would not prevent the user from entering an invalid ID input, since this happens in the IO class, get\_cd function. I removed the ID conversion to an integer logic from the add\_cd function and added try/except code to the get\_cd function.

I also added a try/except functionality in the ‘i’ Display Current Inventory elif statement to handle if the table is empty. This was a problem when I was first running the program and tried to display data before having added any. While I knew I had not added any data, another user may not.

I added error handling to the ‘d’ Delete CD elif statement as well to confirm that the user was entering an integer that is listed in the CD Inventory. If not, they receive and error message asking them to try again.

### Areas evaluated but skipped adding structured Error Handling

There were several places I considered adding error handling but did not because they already seemed able to handle erroneous inputs. These sections included:

* write\_file function, if the file did not exist it would be created, so no need to test for its existence.
* menu\_choice already includes a while loop that asks the user for a valid input and if not valid, repeats request
* read\_file conversion of string to int, not needed because non-int’s can no longer be written into the file

### Changing data storage from Text to Binary

I updated the read\_file and write\_file functions in order to effectuate this change. I stripped all the formatting logic out of the write\_file function and simply wrote the table (list of dictionaries) directly to the file. The rest of the script was already written to work with a list of dictionaries, so there was not much to change as a result of this update. Updating the read\_file section was a bit trickier; I added the FileNotFoundError exception after rerunning the program without a preexisting .dat file. After this, I realized that my .dat file could be empty and added the EOFError exception. With each exception I made sure that my table remained an empty list that I could return from the function. The way I have achieved this seems a bit inefficient, but not sure of a better way to handle when the program is initially being run.

The most difficult part of this assignment (for me) had nothing to do with making the requested updates, unfortunately. After updating the read\_file function I struggled for a long time with why my table was not getting returned, with data, into the ‘l’ Load Inventory elif statement. I FINALLY figured out that I needed to assign the output to a variable, lstTbl.

Before realizing that I could write the list of dictionaries to the binary file, I was still writing strings. I struggled for a long time with this approach in terms of getting all the data back OUT of the file. I was only able to get the first line, without repeating the .load operation and it did not seem appropriate to repeat this for an unknown amount of times. I spent a lot of time trying to look up how to get all data out of a binary file and unpickle it before concluding that the best this to do was to load only a single list of data.

### Testing

I tried to use the Console a bit more this assignment since I found testing while making edits to be really tough in terms of identifying where something went wrong or what type of error message could be returned/captured.

# Saving the Script

After writing and testing my code in Spyder using a copy of my Assignment06 program, I navigated to the folder in my Home Folder called ‘Python’, in which I created a folder for ‘Assignment07’ and saved my text file as CDInventory.py.

# Running the Script

Once I saved my final script, I ran the program successful in Spyder as well as my Terminal.

Text

Description automatically generated

Figure - Testing in Spyder [1 of 5]

Text

Description automatically generated

Figure - Testing in Spyder [2 of 5]

Text

Description automatically generated

Figure - Testing in Spyder [3 of 5]

Text

Description automatically generated

Figure - Testing in Spyder [4 of 5]

Text

Description automatically generated

Figure - Testing in Spyder [5 of 5]

Text

Description automatically generated

Figure - Testing in Terminal [1 of 4]

Text, letter

Description automatically generated

Figure - Testing in Terminal [2 of 4]

Text

Description automatically generated

Figure - Testing in Terminal [3 of 4]

Text

Description automatically generated

Figure - Testing in Terminal [4 of 4]

# Summary

After completing the assigned reading, labs, and videos in Module 07 I was able to edit and execute a script where I added structured error handling and data storage in a binary format to the CD Inventory program. I have posted the assignment .py file and this document to GitHub for peer review:

# Appendix

### Webpages visited to research Exception Handling

* <https://docs.python.org/3/tutorial/errors.html> - I was confused about 1/3 of the way down this page with the Class B, C, D example and stopped reading.
* <https://www.geeksforgeeks.org/python-exception-handling/> - I like the syntax examples on this site, first at the conceptual level and then with example data. This was really helpful, but the page took a long time to load/reload and had a number of pop up windows which I didn’t enjoy.
* <https://www.w3schools.com/python/python_try_except.asp> - This was my FAVORITE page because there were interactive examples I could work through that were at a very basic level.

### Webpages visited to research Pickling in Python

* <https://www.synopsys.com/blogs/software-security/python-pickling/> - I did not like this site, found the terminology confusing and without sufficient explanation for a beginner level Python user.
* <https://www.geeksforgeeks.org/understanding-python-pickling-example/> - I found this page to be helpful, like the Exception Handling content but with the same downsides.
* <https://realpython.com/python-pickle-module/> - I found this page to be confusing because it contained a lot of content that I am not familiar with, which was not explained at a beginner level

## <https://highlight.hohli.com/index.php> (external[[1]](#footnote-1)) web page

1. *#------------------------------------------#*
2. *# Title: CDInventory.py*
3. *# Desc: Working with classes,functions, error handling, and binary files.*
4. *# Change Log: (Who, When, What)*
5. ***# DBiesinger, 2030-Jan-01, Created File***
6. *# NWoodward, 2021-Nov-21, Added functions for Data Processing and IO*
7. *# NWoodward, 2021-Nov-28, Added Error Handling. Changed data storage from text to binary. Removed #DONE notations. Update Desc.*
8. *#------------------------------------------#*
9. **import** pickle
11. *# -- DATA -- #*
12. strChoice = '' *# User input*
13. lstTbl = [] *# list of lists to hold data*
14. dicRow = {} *# list of data row*
15. **strFileName = 'CDInventory.dat' *# binary data storage file***
16. objFile = None *# file object*

19. *# -- PROCESSING -- #*
20. **class DataProcessor:**
21. """ Functions for processing data """
23. @staticmethod
24. **def** add\_cd(intID, strTitle, strArtist, table):
25. **""" Function that allows user to add a CD to the inventory in memory. The CD must be**
26. saved, choice 's', in order for the CD to be written to a binary file.
28. Args:
29. intID (string): ID number entered by user
30. **strArtist (string): Artist name input by user**
31. strTitle (string): CD title input by user
32. table (list of dict): 2D data structure (list of dicts) that holds the data during runtime
34. Returns:
35. **table (list of dict): Updated list of dictionaries that contains CD data. Each dictionary represents one CD.**
36. """
38. dicRow = {'ID': intID, 'Title': strTitle, 'Artist': strArtist}
39. table.append(dicRow)
40. **return table**

43. @staticmethod
44. **def** del\_cd(table, ID):
45. **""" Function that allows user to delete a CD from the inventory in memory.**
47. Args:
48. table (list of dict): 2D data structure (list of dicts) that holds the data during runtime
49. ID: ID number of the CD the user would like to delete
51. Returns:
52. message: Message to the user to tell them if their attempt to delete a CD was successful
53. """
54. intRowNr = -1
55. **blnCDRemoved = False**
56. **for** row **in** table:
57. intRowNr += 1
58. **if** row['ID'] == ID:
59. **del** table[intRowNr]
60. **blnCDRemoved = True**
61. **break**
62. **if** blnCDRemoved:
63. message = **print**('The CD was removed')
64. **else**:
65. **message = print('Could not find this CD!')**
66. **return** message
68. **class** FileProcessor:
69. """Processing the data to and from binary file"""
71. @staticmethod
72. **def** read\_file(file\_name, table):
73. """Function to manage data ingestion from binary file to a list of dictionaries
75. **Reads the data from file identified by file\_name into a 2D table (list of dicts).**
76. Each dictionary represents one CD.
78. Args:
79. file\_name (string): name of file used to read the data from
80. **table (list of dict): 2D data structure (list of dicts) that holds the data during runtime**
82. Returns:
83. table (list of dict): Updated list of dictionaries that contains CD data. Each dictionary represents one CD.
84. """
85. **table.clear() *# this clears existing data and allows to load data from file***
86. **try**:
87. objFile = open(file\_name, 'rb')
88. table = pickle.load(objFile)
89. **except** FileNotFoundError:
90. **print('Creating file.')**
91. objFile = open(file\_name, 'ab')
92. table = []
93. **except** EOFError **as** e:
94. **print**('File is blank. Please add CD**\'**s')
95. **print(type(e))**
96. table = []
97. **finally**:
98. objFile.close()
99. **return** table
101. @staticmethod
102. **def** write\_file(file\_name, table):
103. """Function to write data from a 2D list to a binary file
105. **Writes data from 2D list (list of dictionaries) identified by table into a binary file indentified**
106. by file\_name.
108. Args:
109. file\_name (string): name fo file used to write data to
110. **table (list of dict): 2D data structure (list of dicts) that holds the data during runtime.**
112. Returns:
113. None.
114. """
115. **with open(file\_name, 'wb') as objFile:**
116. pickle.dump(table, objFile)
118. *# -- PRESENTATION (Input/Output) -- #*
120. **class IO:**
121. """Functions handling Input / Output"""
123. @staticmethod
124. **def** print\_menu():
125. **"""Displays a menu of choices to the user**
127. Args:
128. None.
130. **Returns:**
131. None.
132. """
134. **print**('Menu**\n\n**[l] load Inventory from file**\n**[a] Add CD**\n**[i] Display Current Inventory')
135. **print('[d] delete CD from Inventory\n[s] Save Inventory to file\n[x] exit\n')**
137. @staticmethod
138. **def** menu\_choice():
139. """Gets user input for menu selection
141. Args:
142. None.
144. Returns:
145. **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.
162. """
163. **print**('======= The Current Inventory: =======')
164. **print**('ID**\t**CD Title (by: Artist)**\n**')
165. **for row in table:**
166. **print**('{}**\t**{} (by:{})'.format(\*row.values()))
167. **print**('======================================')
169. @staticmethod
170. **def get\_cd():**
171. """ Function to enable user to add a new CD to in memory to a list of dictionaries
173. Args:
174. None.
176. Returns:
177. strID (sting): ID number entered by user
178. strArtist (string): Artist name input by user
179. strTitle (string): CD title input by user
181. """
182. **try**:
183. strID = input('Enter ID: ').strip()
184. strID = int(strID)
185. **strTitle = input('What is the CD\'s title? ').strip()**
186. strArtist = input('What is the Artist**\'**s name? ').strip()
187. **return** strID, strTitle, strArtist
188. **except** ValueError **as** e:
189. **print**('Please enter a whole number. You entered {}'.format(strID))
190. **print(type(e))**

193. *# 1. When program starts, read in the currently saved Inventory*
194. **try**:
195. **lstTbl = FileProcessor.read\_file(strFileName, lstTbl)**
196. **print**(lstTbl)
197. **except** Exception **as** e:
198. **print**(type(e))
200. ***# 2. start main loop***
201. **while** True:
202. *# 2.1 Display Menu to user and get choice*
203. IO.print\_menu()
204. strChoice = IO.menu\_choice()
206. *# 3. Process menu selection*
207. *# 3.1 process exit first*
208. **if** strChoice == 'x':
209. **break**
210. ***# 3.2 process load inventory***
211. **if** strChoice == 'l':
212. **print**('WARNING: If you continue, all unsaved data will be lost and the Inventory re-loaded from file.')
213. strYesNo = input('type **\'**yes**\'** to continue and reload from file. otherwise reload will be canceled: ')
214. **if** strYesNo.lower() == 'yes':
215. **print('reloading...')**
216. lstTbl = FileProcessor.read\_file(strFileName, lstTbl)
217. IO.show\_inventory(lstTbl)
218. **else**:
219. input('canceling... Inventory data NOT reloaded. Press [ENTER] to continue to the menu.')
220. **IO.show\_inventory(lstTbl)**
221. **continue** *# start loop back at top.*
222. *# 3.3 process add a CD*
223. **elif** strChoice == 'a':
224. *# 3.3.1 Ask user for new ID, CD Title and Artist*
225. **try:**
226. strID, strArtist, strTitle = IO.get\_cd()
227. *# 3.3.2 Add item to the table*
228. lstTbl = DataProcessor.add\_cd(strID, strArtist, strTitle, lstTbl)
229. IO.show\_inventory(lstTbl)
230. **except TypeError as e:**
231. **print**('Please enter valid CD inputs. Select **\'**a**\'** again to retry.')
232. **print**(type(e))
233. **finally**:
234. **continue** *# start loop back at top.*
235. ***# 3.4 process display current inventory***
236. **elif** strChoice == 'i':
237. **try**:
238. IO.show\_inventory(lstTbl)
239. **except** TypeError **as** e:
240. **print('The CD Inventory is blank. Please add content.')**
241. **print**(type(e))
242. **finally**:
243. **continue** *# start loop back at top.*
244. *# 3.5 process delete a CD*
245. **elif strChoice == 'd':**
246. *# 3.5.1 get Userinput for which CD to delete*
247. *# 3.5.1.1 display Inventory to user*
248. IO.show\_inventory(lstTbl)
249. *# 3.5.1.2 ask user which ID to remove*
250. **try:**
251. intIDDel = int(input('Which ID would you like to delete? ').strip())
252. *# 3.5.2 search thru table and delete CD*
253. DataProcessor.del\_cd(lstTbl,intIDDel)
254. **print**() *# Add extra space for layout*
255. **IO.show\_inventory(lstTbl)**
256. **except** ValueError **as** e:
257. **print**('Please enter a whole number listed as ID in inventory. Select **\'**d**\'** again to retry.')
258. **print**(type(e))
259. **finally**:
260. **continue *# start loop back at top.***
261. *# 3.6 process save inventory to file*
262. **elif** strChoice == 's':
263. *# 3.6.1 Display current inventory and ask user for confirmation to save*
264. IO.show\_inventory(lstTbl)
265. **strYesNo = input('Save this inventory to file? [y/n] ').strip().lower()**
266. *# 3.6.2 Process choice*
267. **if** strYesNo == 'y':
268. *# 3.6.2.1 save data*
269. FileProcessor.write\_file(strFileName, lstTbl)
270. **else:**
271. input('The inventory was NOT saved to file. Press [ENTER] to return to the menu.')
272. **continue** *# start loop back at top.*
273. *# 3.7 catch-all should not be possible, as user choice gets vetted in IO, but to be save:*
274. **else**:
275. **print('General Error')**

1. Accessed November 28, 2021 [↑](#footnote-ref-1)