Ryan Goding

8/26/2020

IT FDN 110: Introduction to Programming (Python)

Assignment 07

# Introduction

Assignment 07 asks us to modify CDInventory.py to add structured error handling and to modify the permanent data store to use binary data.

# Structured Error Handling

This assignment introduced me to structured error handling. The benefit of structured error handling is that the program can keep running if an input or file read or write has failed. The first function that I added error handling to was the add inventory function. I wanted to ensure that if a non-integer was added that it wouldn’t crash the program. I did this within the add\_inventorydata function itself, with a try, except structure, and can be seen in Figure 1.

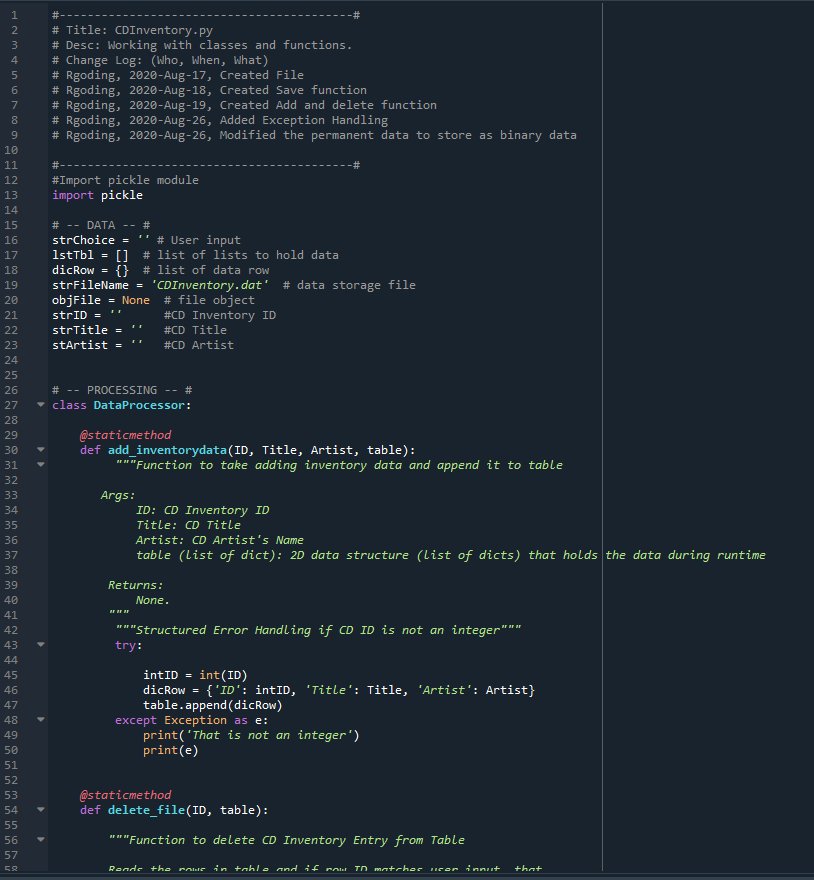


Figure 1: add\_inventorydata structured error handling

The other areas that required structured error handling was the delete function (if an integer wasn’t input), the initial read file (if no CDInventory data file existed), and load function (if no data file existed). An example of the error handling occurring if no CDInventory data file exists is shown in Figure 2. An example of error handling if a non-integer is entered for the delete function is shown in Figure 3.

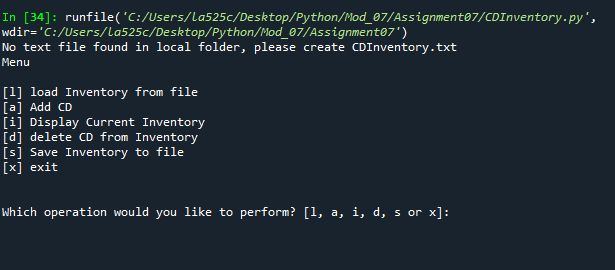


Figure 2: CDInventory.py Script Run in Spyder, with no data file

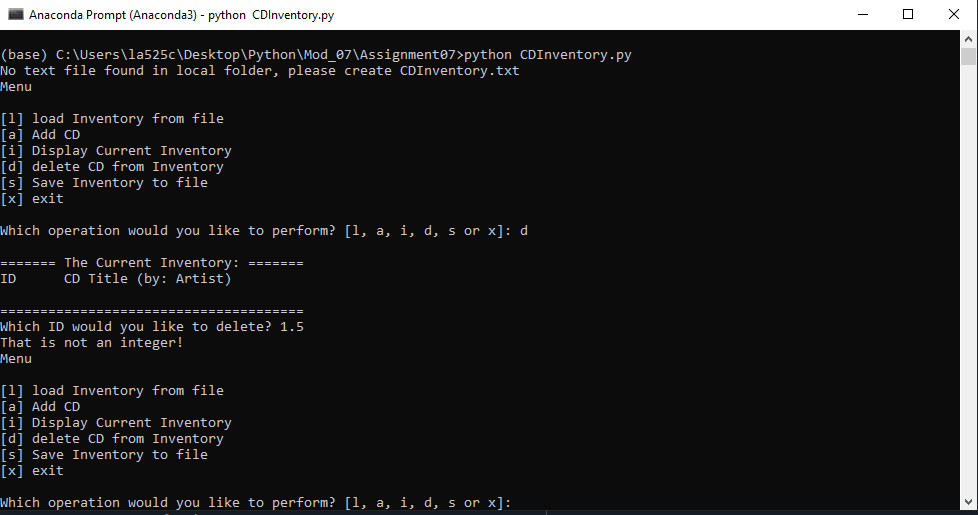


Figure 3: CDInventory.py Error handling in terminal

# Storing as Binary data

To modify the permanent data store to use binary data I first imported the pickle module. I did modified the write\_file function to open file\_name using ‘wb’ to write in binary. Then pickle.dump was used to take lstTbl and save it as binary as file\_name. The script for this can be seen in Figure 4.

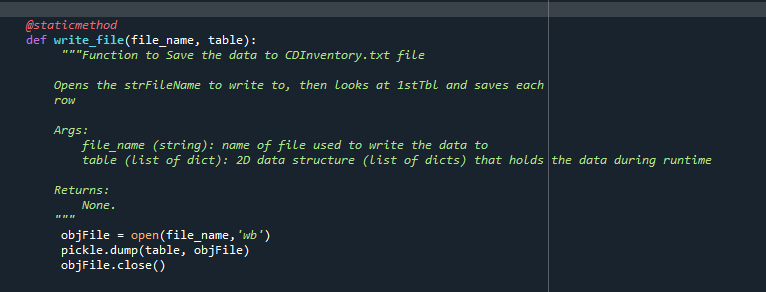


Figure 4: write\_file script to save as binary

This assignment also asked us to create an Assignment\_06 repository on Github to share our script and knowledge document. The link for my repository is <https://github.com/rgoding/Assignment_07>.

# Exception Handling in Python Research

The most useful page on Exception handling was reference 1. This page allowed you to run the example code, as well as providing different examples for the try except structure. Reference 2 also helped by just starting at the initial concepts and then working into examples later on in the article.

# Pickling Research

The best page for Pickling research I found was reference 3, as it provide basic examples as well as talking through the main concepts.

# Summary

I found this assignment difficult as I have never used error handling before. The biggest hurdle for my understanding of error handling was to understand the structure of try except. I tried using the try and except in both the main while loop and within the functions, but not sure which is the preferred method. Pickling was also abstract for me, because converting it to a text file is something that makes sense, whereas storing it as a binary file is hard to keep track of what is being stored.

# References

1. Programiz.com, “Python Exception Handling Using try, except and finally statement”, <https://www.programiz.com/python-programming/exception-handling>
2. Realpython.com, “Python Exceptions: An Introduction”, <https://realpython.com/python-exceptions/>
3. Geeksforgeeks.org, “Understanding Python Pickling example”, <https://www.geeksforgeeks.org/understanding-python-pickling-example/>

# Appendix

## Listing CDInventory.py

1. #------------------------------------------#
2. # Title: CDInventory.py
3. # Desc: Working with classes and functions.
4. # Change Log: (Who, When, What)
5. # Rgoding, 2020-Aug-17, Created File
6. # Rgoding, 2020-Aug-18, Created Save function
7. # Rgoding, 2020-Aug-19, Created Add and delete function
8. # Rgoding, 2020-Aug-26, Added Exception Handling
9. # Rgoding, 2020-Aug-26, Modified the permanent data to store as binary data
11. #------------------------------------------#
12. #Import pickle module
13. **import** pickle
15. # -- DATA -- #
16. strChoice = '' # User input
17. lstTbl = []  # list of lists to hold data
18. dicRow = {}  # list of data row
19. strFileName = 'CDInventory.dat'  # data storage file
20. objFile = None  # file object
21. strID = ''      #CD Inventory ID
22. strTitle = ''   #CD Title
23. stArtist = ''   #CD Artist

26. # -- PROCESSING -- #
27. **class** DataProcessor:
29. @staticmethod
30. **def** add\_inventorydata(ID, Title, Artist, table):
31. """Function to take adding inventory data and append it to table
33. Args:
34. ID: CD Inventory ID
35. Title: CD Title
36. Artist: CD Artist's Name
37. table (list of dict): 2D data structure (list of dicts) that holds the data during runtime
39. Returns:
40. None.
41. """
42. """Structured Error Handling if CD ID is not an integer"""
43. **try**:
45. intID = int(ID)
46. dicRow = {'ID': intID, 'Title': Title, 'Artist': Artist}
47. table.append(dicRow)
48. **except** Exception as e:
49. **print**('That is not an integer')
50. **print**(e)

53. @staticmethod
54. **def** delete\_file(ID, table):
56. """Function to delete CD Inventory Entry from Table
58. Reads the rows in table and if row ID matches user input, that
59. row and entry is removed from table
61. Args:
62. ID: CD Inventory ID to be deleted, input by user
63. table (list of dict): 2D data structure (list of dicts) that holds the data during runtime
65. Returns:
66. None.
67. """

70. intRowNr = -1
71. blnCDRemoved = False
72. **for** row **in** table:
73. intRowNr += 1
74. **if** row['ID'] == ID:
75. **del** table[intRowNr]
76. blnCDRemoved = True
77. **break**
78. **if** blnCDRemoved:
79. **print**('The CD was removed')
80. **else**:
81. **print**('Could not find this CD!')

84. **class** FileProcessor:
85. """Processing the data to and from text file"""
87. @staticmethod
88. **def** read\_file(file\_name, table):
89. """Function to manage data ingestion from file to a list of dictionaries
91. Reads the data from file identified by file\_name into a 2D table
92. (list of dicts) table one line in the file represents one dictionary row in table.
94. Args:
95. file\_name (string): name of file used to read the data from
96. table (list of dict): 2D data structure (list of dicts) that holds the data during runtime
98. Returns:
99. None.
100. """
101. objFile = open(file\_name, 'ab')
102. data = pickle.load(objFile)
103. pickle.dump(data, objFile)

106. @staticmethod
107. **def** write\_file(file\_name, table):
108. """Function to Save the data to CDInventory.txt file
110. Opens the strFileName to write to, then looks at 1stTbl and saves each
111. row
113. Args:
114. file\_name (string): name of file used to write the data to
115. table (list of dict): 2D data structure (list of dicts) that holds the data during runtime
117. Returns:
118. None.
119. """
120. objFile = open(file\_name,'wb')
121. pickle.dump(table, objFile)
122. objFile.close()


126. # -- PRESENTATION (Input/Output) -- #
128. **class** IO:
129. """Handling Input / Output"""
131. @staticmethod
132. **def** print\_menu():
133. """Displays a menu of choices to the user
135. Args:
136. None.
138. Returns:
139. None.
140. """
142. **print**('Menu\n\n[l] load Inventory from file\n[a] Add CD\n[i] Display Current Inventory')
143. **print**('[d] delete CD from Inventory\n[s] Save Inventory to file\n[x] exit\n')
145. @staticmethod
146. **def** menu\_choice():
147. """Gets user input for menu selection
149. Args:
150. None.
152. Returns:
153. choice (string): a lower case sting of the users input out of the choices l, a, i, d, s or x
155. """
156. choice = ' '
157. **while** choice **not** **in** ['l', 'a', 'i', 'd', 's', 'x']:
158. choice = input('Which operation would you like to perform? [l, a, i, d, s or x]: ').lower().strip()
159. **print**()  # Add extra space for layout
160. **return** choice
162. @staticmethod
163. **def** show\_inventory(table):
164. """Displays current inventory table

167. Args:
168. table (list of dict): 2D data structure (list of dicts) that holds the data during runtime.
170. Returns:
171. None.
173. """
174. **print**('======= The Current Inventory: =======')
175. **print**('ID\tCD Title (by: Artist)\n')
176. **for** row **in** table:
177. **print**('{}\t{} (by:{})'.format(\*row.values()))
178. **print**('======================================')
180. @staticmethod
181. **def** add\_inventory():
182. """Add Inventory to CDInventory.txt
184. Args:
185. None
187. Returns:
188. strID, strTitle, stArtist for data processing function to handle
190. """
191. strID = input('Enter ID: ').strip()
192. strTitle = input('What is the CD\'s title? ').strip()
193. stArtist = input('What is the Artist\'s name? ').strip()
194. DataProcessor.add\_inventorydata(strID,strTitle,stArtist,lstTbl)

197. # 1. When program starts, read in the currently saved Inventory
198. # Add Error exceptions to display error instead of closing program
199. **try**:
200. FileProcessor.read\_file(strFileName, lstTbl)
201. **except**:
202. **print**("No text file found in local folder, please create CDInventory.txt")

205. # 2. start main loop
206. **while** True:
207. # 2.1 Display Menu to user and get choice
208. IO.print\_menu()
209. strChoice = IO.menu\_choice()
211. # 3. Process menu selection
212. # 3.1 process exit first
213. **if** strChoice == 'x':
214. **break**
215. # 3.2 process load inventory
216. **if** strChoice == 'l':
217. **print**('WARNING: If you continue, all unsaved data will be lost and the Inventory re-loaded from file.')
218. strYesNo = input('type \'yes\' to continue and reload from file. otherwise reload will be canceled')
219. #Add Error handling if file is not created
220. **try**:
221. **if** strYesNo.lower() == 'yes':
222. **print**('reloading...')
223. FileProcessor.read\_file(strFileName, lstTbl)
224. IO.show\_inventory(lstTbl)
225. **except**:
226. **print**("CDInventory.txt does not exist in local directory")
227. **else**:
228. input('canceling... Inventory data NOT reloaded. Press [ENTER] to continue to the menu.')
229. IO.show\_inventory(lstTbl)
230. **continue**  # start loop back at top.
231. # 3.3 process add a CD
232. **elif** strChoice == 'a':
233. # 3.3.1 Ask user for new ID, CD Title and Artist
234. IO.add\_inventory()
235. # 3.3.2 Add item to the table
237. IO.show\_inventory(lstTbl)
238. **continue**  # start loop back at top.
239. # 3.4 process display current inventory
240. **elif** strChoice == 'i':
241. IO.show\_inventory(lstTbl)
242. **continue**  # start loop back at top.
243. # 3.5 process delete a CD
244. **elif** strChoice == 'd':
245. # 3.5.1 get Userinput for which CD to delete
246. # 3.5.1.1 display Inventory to user
247. IO.show\_inventory(lstTbl)
248. # 3.5.1.2 ask user which ID to remove, with error exception added
249. **try**:
250. intIDDel = int(input('Which ID would you like to delete? ').strip())
251. # 3.5.2 search thru table and delete CD
252. DataProcessor.delete\_file(intIDDel,lstTbl)
253. IO.show\_inventory(lstTbl)
254. **except**:
255. **print**("That is not an integer!")
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(strFileName,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')