Troy McFarland

2021-02-28

Foundations of Programming, Python

Assignment\_07

Git Hub: https://github.com/DJ-TMC/Assignment\_07

# Modfying Script by Adding Error Handling and Saving Binary Files via ‘Pickling’

# Introduction

In this paper, I’ll be going over the seventh set of lessons for the Foundations of Programming, Python class taught by Dirk Biesinge and assisted by Douglas Klos. I’ll go over things learned, insights I had coding longer programs in Spyder while working on assignment 6 and one unresolved issue.

# Overview of New Things Learned in Module 7

* What are the benefits of using structured error handling?
  + The main reason to use it is to prevent the program from ending, and the user having lost their work.
  + It also is a way to give the user feedback on their interaction with the program, and gives the programmer another opportunity to help the user enter correct data
* What are the differences between a text file and a binary file?
  + A text file can technically be human read (and modified) in any text editor. It contains strings of text. A binary file is a memory dump to a file. It isn’t completely human readable.
  + Similarities: Both can be used to store data, by having a program write to and read from it.
* How is the Exception class used?
  + You can capture the specific exception object and give details about the error to the user.
* How do you "derive" a new class from the Exception class?
  + You would create a new class, such as
    - Class ExampleErrorThingy(Exception):
    - “”” docstring goes here”””
    - def \_\_str\_\_(self):
      * return ‘Custom Error Message goes here’
* When might you create a class derived from the Exception class?
  + When you want to give custom feedback about specific errors to a user, you might create a class derived from the Exception class. This enables you to raise errors that normally wouldn’t be raised.
* What is the Markdown language?
  + It is a type of formatting that makes github readme files more readable. When I glanced at it, it reminded me a little bit of HTML mark**up** language. The end result is similar: It’s a way to format text.

# Coding in Python

Some interesting issues this time around: I learned how to open files with more efficient code ‘with open’, and wanted to try it out here. However, I was unable to use that code easily when loading a list of dictionaries. I ended up using an older method to get it to work.

In **Lab07-A**, I was initially getting a 2D list inadvertently with using the ‘with open’ method:

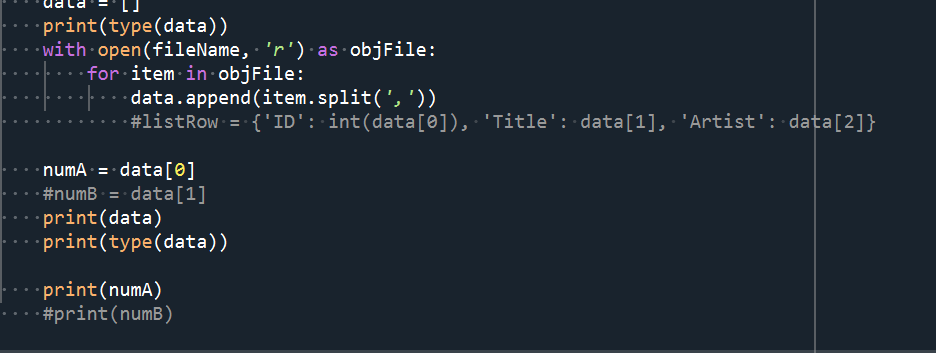


Figure 1: with open not working as expected

Result:

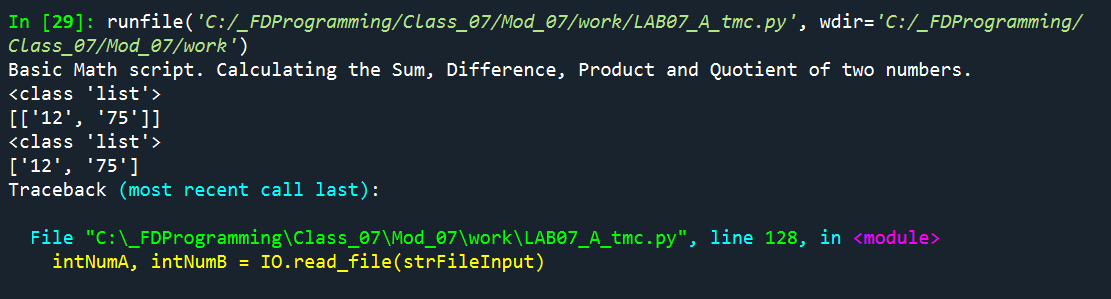


Figure 2: with open results

I finally solved the issue by using this syntax:

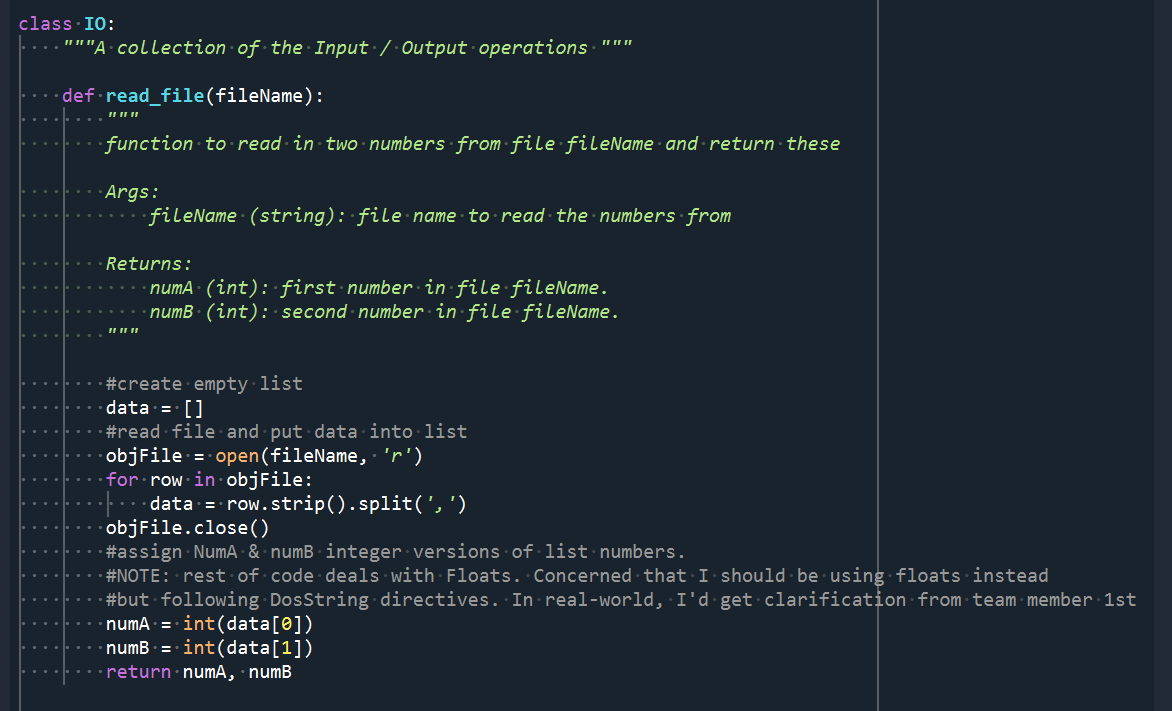


Figure 3: working code

How Lab 07-A Code works:

**read\_file** takes argument: filename (name of the file in which to read numbers from) and

Returns: numA & NumB, two integers

I use a for loop to strip out the numbers, splitting by comma. I convert them to integers, so other parts of the script can do calculations with them.

If I was taking multiple sets of numbers, I would create a list of lists, with each list being a line from a text file. I would do this by wrapping my current code in another for loop/

**write\_file** takes argument: filename (file to write to) and results (a list of results from all the calculations, as floats)

I then join the results together into a string by using the .join command, mapping it to a string, and adding a comma in between each part.  
  
If I was to write this to a file, I would look into ingesting a list of strings, and then adding another layer to my for loop to create multiple strings, joined together, and adding the \n slash to designate returns.

I struggled with Lab 7-B & C like many others, but will put more time into it after this submission.

Assignment 7  
For Assignment 7 itself, I was able to get the two new parts of code working fairly quickly (only a ¾ day’s worth of work). I updated the load and save code to dump binary data into a file through pickling. While the book & Dirk’s instructions were fairly clear, this webpage also helped: [Python Pickle Tutorial - DataCamp](https://www.datacamp.com/community/tutorials/pickle-python-tutorial) (external link). I thought it was funny that for lambda functions, the pickle package isn’t enough. One Also needs to load the ‘dill’ package!   
  
I’m looking forward to learning how to store files in the JSON format, as this will pertain more to the type of work I do professionally.   
  
I was happy to encounter an End of File Error (EOFError) when I tried to load up a blank .dat file. I was able to trap it with an exception before submitting the assignment!

# One Confusing Thing that Remains

After running my code in Spyder, I exited out of my program, but left Anaconda open. I wanted to delete the .dat file so I could then capture the program creating a .dat file from scratch. However, I was unable to delete the .dat file, and I got this error:

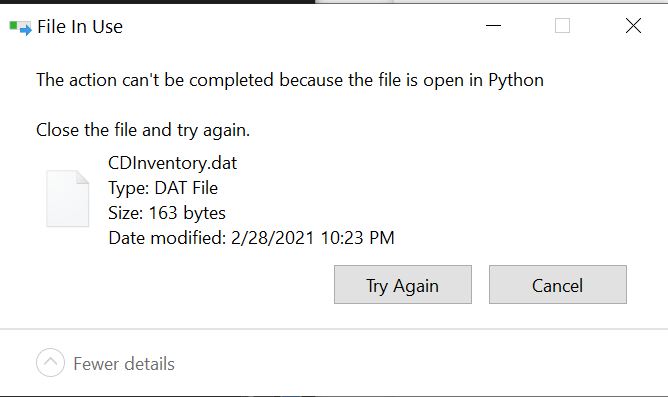


Figure 4: unresolved error message

I’m unsure exactly why this happened, but I think it has something to do with the fact that I initially tried to read from a blank .dat file. I had an exception that trapped that error, but it may not have been enough to release this file.

# Assignment 07Running in Spyder:

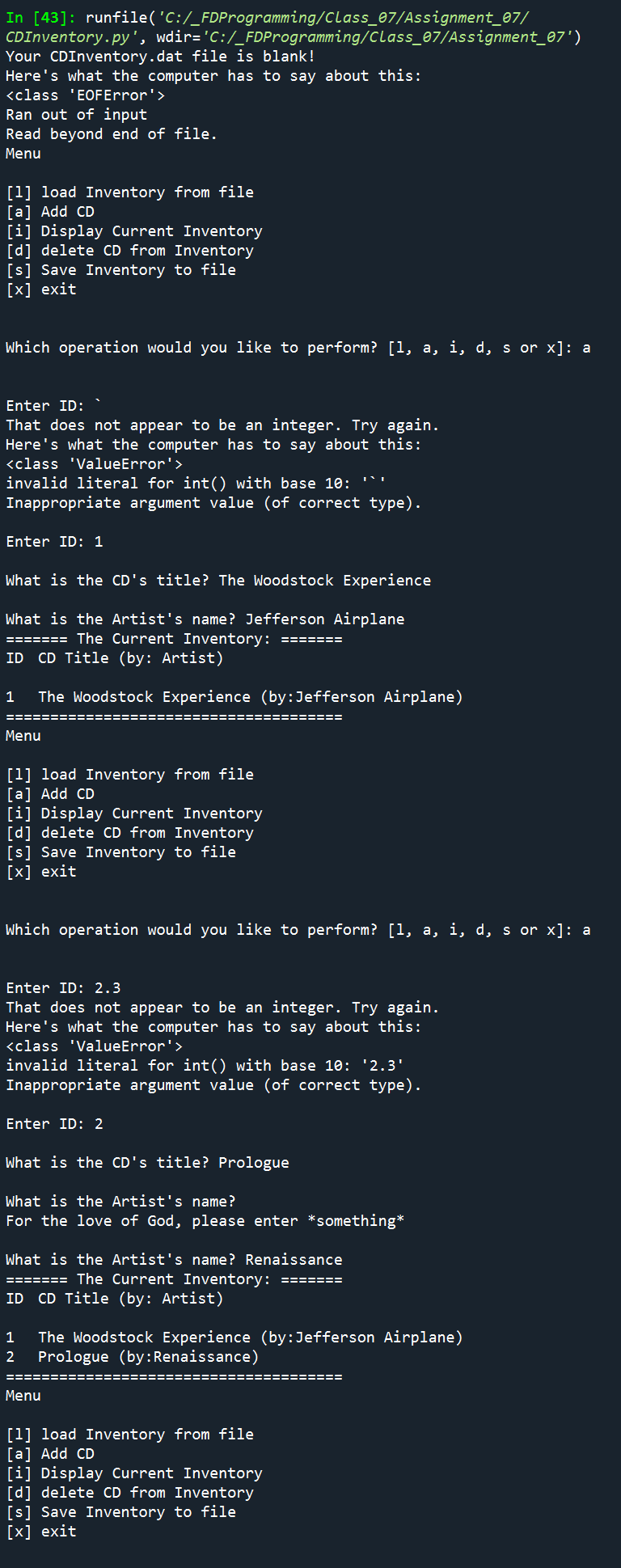
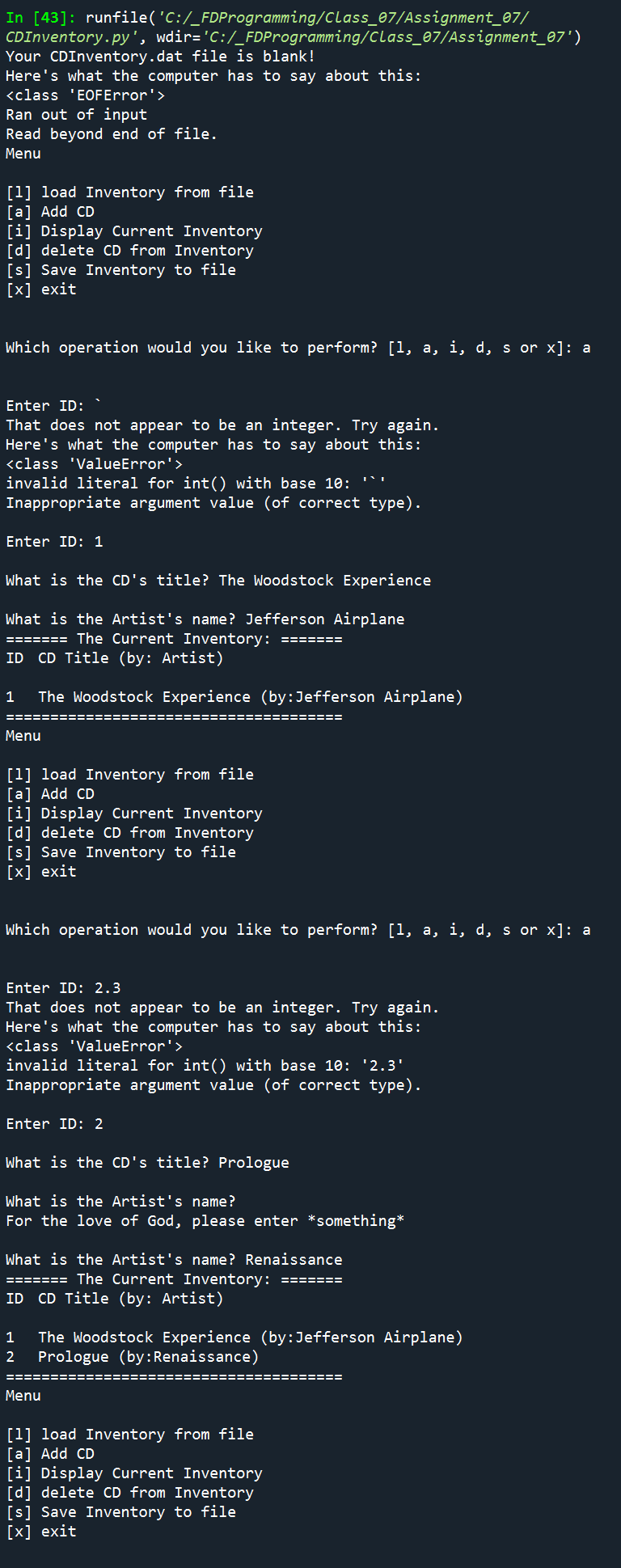
 

Figure 5: Spyder Results

# Assignment 07 Running in the Anaconda Terminal:

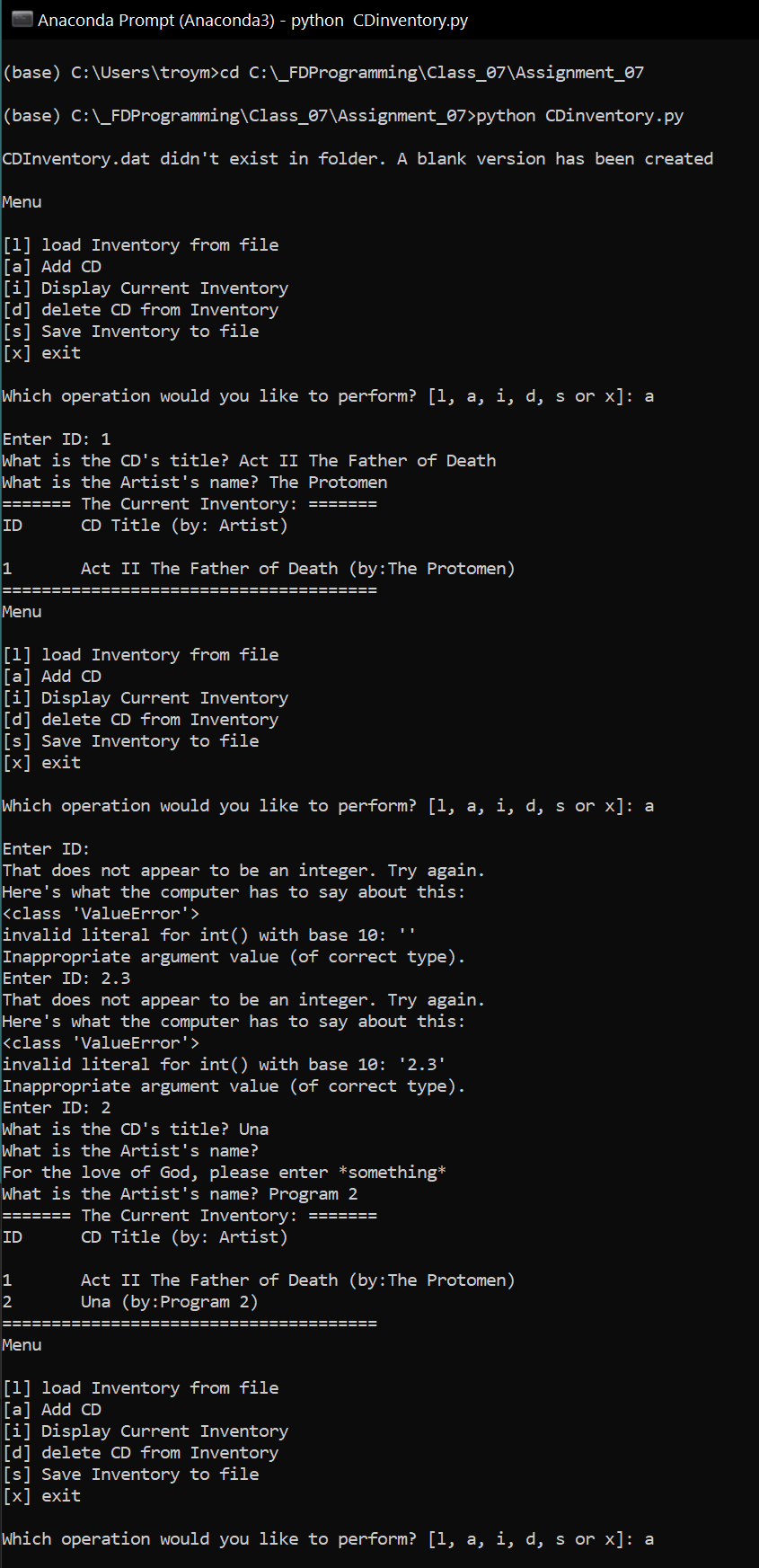


Figure 6: Anaconda Terminal Results

# Summary

In this paper, I described Error handling, and pickling techniques learned in the seventh set of lessons for the Foundations of Programming, Python class taught by Dirk Biesinge and assisted by Douglas Klos. I went over some struggles using new ‘with open’ syntax, and a perplexing inability to delete the .dat file after I closed my code, but left Spyder open.

# Appendix

Full Code of Assignment 7:

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. *# TMcFarland, 2021-Feb-19, started mods: adding functions, addressed existing TODOs*
7. *# TMcFarland, 2021-Feb-28, made code updates as suggested by DKlos for Assignment 6*
8. *# TMcFarland, 2021-Feb-28, changed data save & retreval to pickeld .dat file*
9. *# TMcFarland, 2021-Feb-28, introduced error handling with user entry and checking for exisiting database.*
10. *#------------------------------------------#*
11. *#Importing os.path for additional FileProcessor.read\_file functionality, to check to see if text file exists*
12. import os**.**path**,** pickle *#, shelve*
13. *# -- DATA -- #*
14. strChoice **=** '' *# User input*
15. lstTbl **=** **[]** *# list of dicts to hold data*
16. dicRow **=** **{}** *# dict of data row*
17. strFileName **=** 'CDInventory.dat' *# data storage file*
18. objFile **=** None *# file object*
19. *# -- PROCESSING -- #*
20. class DataProcessor**:**
21. @staticmethod
22. def delete\_dict**(**delDict**,** table**):**
23. """Deletes lstTbl dicitonary entry based on user input of ID number
24. Args:
25. delDict (integer): ID number representing table entry
26. Returns:
27. NONE
28. """
29. intRowNr **=** **-**1
30. blnCDRemoved **=** False
31. for row in table**:**
32. intRowNr **+=** 1
33. if row**[**'ID'**]** **==** delDict**:**
34. *# Again here we're using the local value of table*
35. *# del lstTbl[intRowNr]*
36. del table**[**intRowNr**]**
37. blnCDRemoved **=** True
38. break
39. *# Return the delete status*
40. return blnCDRemoved
41. @staticmethod
42. def append\_table**(**intID**,** strTitle**,** strArtist**,** table**):**
43. *# 3.3.2 Add item to the table*
44. """Takes arguments from Main loop append section and appends to 'table'.
45. Because this is a list, and lists are references, changes made to local variable 'table'
46. will automatically update the original list, as they are the same thing in memory
47. Args:
48. intID (string): ID number
49. strTitle (string): CD title
50. StrArtist (string); Artist Name
51. table (list); list of dictionary entries of above data
52. Returns:
53. None
54. """
55. dicRow **=** **{**'ID'**:** intID**,** 'Title'**:** strTitle**,** 'Artist'**:** strArtist**}**
56. *# Append to the table passed in. Since lstTbl is passed as a reference type,*
57. *# we do not have to return it.*
58. table**.**append**(**dicRow**)**
59. class FileProcessor**:**
60. """Processing the data to and from text file"""
61. @staticmethod
62. def read\_file**(**file\_name**,** table**):**
63. """Function to manage data ingestion from file to a list of dictionaries
64. Reads the data from file identified by file\_name into a 2D table
65. (list of dicts). One line in the file represents one dictionary row in table.
66. Args:
67. file\_name (string): name of file used to read the data from
68. table (list of dict): 2D data structure (list of dicts) that holds the data during runtime
69. Returns:
70. fileCreate (Bool) If true, Binary file didn't exist and was created'
71. """
72. table**.**clear**()** *# this clears existing data and allows to load data from file*
73. *#check to see if database file already exists. If not, create blank one.*
74. fileCreate **=** False
75. if os**.**path**.**exists**(**'CDInventory.dat'**):**
76. try**:**
77. f **=** open**(**file\_name**,** 'rb'**)**
78. table **=** pickle**.**load**(**f**)**
79. f**.**close**()**
80. except EOFError as e**:**
81. print**(**'Your CDInventory.dat file is blank!'**)**
82. print**(**'Here\'s what the computer has to say about this:'**)**
83. print**(**type**(**e**),** e**,** e**.**\_\_doc\_\_**,** sep **=** '\n'**)** *#syntax from FDN\_Py\_module\_07, pg 20, listing 12*
84. else**:**
85. *#create blank database in same folder as script*
86. objFile **=** open**(**strFileName**,** 'w'**)**
87. objFile**.**close**()**
88. fileCreate **=** True
89. return fileCreate**,** table
90. @staticmethod
91. def write\_file**(**table**):**
92. """Writes List of Dicitonaries lstTbl from memory into a text file.
93. Ensures proper comma seperated formatting for best storage and retrieval
94. Args:
95. None
96. Returns:
97. None
98. """
99. f **=** open**(**'CDInventory.dat'**,** 'wb'**)**
100. pickle**.**dump**(**table**,** f**)**
101. f**.**close**()**
102. *# -- PRESENTATION (Input/Output) -- #*
103. class IO**:**
104. """Handling Input / Output"""
105. @staticmethod
106. def print\_menu**():**
107. """Displays a menu of choices to the user
108. Args:
109. None.
110. Returns:
111. None.
112. """
113. print**(**'Menu\n\n[l] load Inventory from file\n[a] Add CD\n[i] Display Current Inventory'**)**
114. print**(**'[d] delete CD from Inventory\n[s] Save Inventory to file\n[x] exit\n'**)**
115. @staticmethod
116. def menu\_choice**():**
117. """Gets user input for menu selection
118. Args:
119. None.
120. Returns:
121. choice (string): a lower case sting of the users input out of the choices l, a, i, d, s or x
122. """
123. choice **=** ' '
124. while choice not in **[**'l'**,** 'a'**,** 'i'**,** 'd'**,** 's'**,** 'x'**]:**
125. choice **=** input**(**'Which operation would you like to perform? [l, a, i, d, s or x]: '**).**lower**().**strip**()**
126. print**()** *# Add extra space for layout*
127. return choice
128. @staticmethod
129. def show\_inventory**(**table**):**
130. """Displays current inventory table
131. Args:
132. table (list of dict): 2D data structure (list of dicts) that holds the data during runtime.
133. Returns:
134. None.
135. """
136. print**(**'======= The Current Inventory: ======='**)**
137. print**(**'ID\tCD Title (by: Artist)\n'**)**
138. for row in table**:**
139. print**(**'{}\t{} (by:{})'**.**format**(\***row**.**values**()))**
140. print**(**'======================================'**)**
141. @staticmethod
142. def user\_cd\_input**():**
143. """ Receives user input for CD ID number, Album Title, and Artist Name
144. Args:
145. None
146. Returns:
147. All strings: CD ID number, CD Title, CD Artist Name
148. """
149. *# 3.3.1 Ask user for new ID, CD Title and Artist*
150. while True**:**
151. try**:**
152. intID **=** int**(**input**(**'Enter ID: '**).**strip**())**
153. break
154. except ValueError as e**:**
155. print**(**'That does not appear to be an integer. Try again.'**)**
156. print**(**'Here\'s what the computer has to say about this:'**)**
157. print**(**type**(**e**),** e**,** e**.**\_\_doc\_\_**,** sep **=** '\n'**)** *#syntax from FDN\_Py\_module\_07, pg 20, listing 12*
158. except Exception as e**:**
159. print**(**'There was some sort of general error, please try again'**)**
160. print**(**'Here\'s what the computer has to say about this:'**)**
161. print**(**type**(**e**),** e**,** e**.**\_\_doc\_\_**,** sep **=** '\n'**)**
162. while True**:**
163. strTitle **=** input**(**'What is the CD\'s title? '**).**strip**()**
164. if strTitle **==** ''**:**
165. print**(**'For the love of God, please enter \*something\*'**)**
166. continue
167. else**:**
168. break
169. while True**:**
170. strArtist **=** input**(**'What is the Artist\'s name? '**).**strip**()**
171. if strArtist **==** ''**:**
172. print**(**'For the love of God, please enter \*something\*'**)**
173. continue
174. else**:**
175. break
176. return intID**,** strTitle**,** strArtist
177. *# -- MAIN PROGRAM -- #*
178. *# 1. When program starts, read in the currently saved Inventory*
179. noDatFilePresent**,** lstTbl **=** FileProcessor**.**read\_file**(**strFileName**,** lstTbl**)**
180. *#check to see if FileProcessor.read\_file returned a Bool of True. if so, DB file didn't exist*
181. if noDatFilePresent**:**
182. print**(**'\nCDInventory.dat didn\'t exist in folder. A blank version has been created\n'**)**
183. *# 2. start main loop*
184. while True**:**
185. *# 2.1 Display Menu to user and get choice*
186. IO**.**print\_menu**()**
187. strChoice **=** IO**.**menu\_choice**()**
188. *# 3. Process menu selection*
189. *# 3.1 EXIT process exit first*
190. if strChoice **==** 'x'**:**
191. break
192. *# 3.2 LOAD process load inventory*
193. if strChoice **==** 'l'**:**
194. print**(**'WARNING: If you continue, all unsaved data will be lost and the Inventory re-loaded from file.'**)**
195. strYesNo **=** input**(**'type \'yes\' to continue and reload from file. otherwise reload will be canceled: '**)**
196. if strYesNo**.**lower**()** **==** 'yes'**:**
197. print**(**'reloading...\n'**)**
198. fileCreate**,** lstTbl **=** FileProcessor**.**read\_file**(**strFileName**,** lstTbl**)**
199. IO**.**show\_inventory**(**lstTbl**)**
200. else**:**
201. input**(**'canceling... Inventory data NOT reloaded. Press [ENTER] to continue to the menu.\n'**)**
202. IO**.**show\_inventory**(**lstTbl**)**
203. continue
204. *# 3.3 ADD CD process add a CD*
205. elif strChoice **==** 'a'**:**
206. *#assign received data from user variables from return of IO.user\_cd\_input*
207. recIntId**,** recStrTitle**,** recStrArtist **=** IO**.**user\_cd\_input**()**
208. *#Feed received data from user into DataProcessor.append\_table*
209. DataProcessor**.**append\_table**(**recIntId**,** recStrTitle**,** recStrArtist**,** lstTbl**)**
210. IO**.**show\_inventory**(**lstTbl**)**
211. continue
212. *# 3.4 DISPLAY INVENTORY process display current inventory*
213. elif strChoice **==** 'i'**:**
214. IO**.**show\_inventory**(**lstTbl**)**
215. continue
216. *# 3.5 DELETE process delete a CD*
217. elif strChoice **==** 'd'**:**
218. *# 3.5.1 get Userinput for which CD to delete*
219. *# 3.5.1.1 display Inventory to user*
220. IO**.**show\_inventory**(**lstTbl**)**
221. *# 3.5.1.2 ask user which ID to remove*
222. intIDDel **=** int**(**input**(**'Which ID would you like to delete? '**).**strip**())**
223. *# 3.5.2 search thru table and delete CD*
224. cd\_removed **=** DataProcessor**.**delete\_dict**(**intIDDel**,** lstTbl**)**
225. *# Using that returned bool we can now display the delete status.*
226. if cd\_removed**:** *#could add '== True', but this is more efficient*
227. print**(**'The CD was removed\n'**)**
228. else**:**
229. print**(**'Could not find this CD!\n'**)**
230. IO**.**show\_inventory**(**lstTbl**)**
231. continue
232. *# 3.6 SAVE process save inventory to file*
233. elif strChoice **==** 's'**:**
234. *# 3.6.1 Display current inventory and ask user for confirmation to save*
235. IO**.**show\_inventory**(**lstTbl**)**
236. strYesNo **=** input**(**'Save this inventory to file? [y/n] '**).**strip**().**lower**()**
237. *# 3.6.2 Process choice*
238. if strYesNo **==** 'y' or 'yes'**:**
239. FileProcessor**.**write\_file**(**lstTbl**)**
240. print**(**'Saved to file\n'**)**
241. else**:**
242. input**(**'The inventory was NOT saved to file. Press [ENTER] to return to the menu.'**)**
243. continue
244. *# 3.7 catch-all should not be possible, as user choice gets vetted in IO, but to be save:*
245. else**:**
246. print**(**'General Error'**)**