Kelsey Sorge-Toomey  
2020 Aug 27  
IT FDN 110 B Su 20: Foundations Of Programming: Python  
Assignment 07

Module 07 Assignment 07

# Introduction

The goal of the seventh’s week’s homework was to explore working with binary and text files, including pickling. We also covered Structured Error Handling and were introduced to creating custom exception classes.

This week was another difficult one for me. I’ve never worked with binary files, nor structured error handling and exceptions.

# Assignment 07:

I started by making changes to CDInventory.py from last week as suggested by Doug. This included removing global variables from my functions and replacing them with the appropriate arguments. Arguments still throw me for a bit of a loop, but I understand them much better now. In the past, I’ve had a hard time understanding how to make arguments non-specific and then use the specific arguments when calling the function in the main body.

Next, I flailed around with trying to get pickling to work for FileProcessor.read\_file and FileProcessor.save\_entry (Figure and ). I was struggling to get both to work so I decided to switch gears to working on structured error handling.

A screenshot of a cell phone

Description automatically generated

Figure - read\_file trying to unpickle but not working

A screenshot of a cell phone screen with text

Description automatically generated

Figure - save\_entry trying in vain to pickle

# Structured Error Handling

I identified three areas to add error handling; handling if a file doesn’t exist at FileProcessor.read\_file, handling if the user enters a non-integer at IO.user\_entry, and handling if the user enters a non-integer when deleting an entry in the main body.

First, I modified IO.user\_entry to first check if the user entered a number for strID (Figure 3). I originally tried this code at DataProcessor.add\_user\_data (Figure 4), but it would run through adding the CD title and CD artist before the error message would pop. I referenced [RealPython](https://realpython.com/python-exceptions/) (external site) for more details on exceptions.

A close up of text on a screen

Description automatically generated

Figure - Checking if user entered a non-integer

A screen shot of a computer

Description automatically generated

Figure - First try at DataProcessor.add\_user\_entry

Then I modified FileProcessor.read\_file to check if CDInventory.txt existed and could be opened. If it didn’t exist, the program would return to the main menu (Figure 5).

A screenshot of a cell phone

Description automatically generated

Figure - Error handling for read\_file

My last structured error handling was at elif strChoice == ‘d’ where a user would enter an ID to be deleted from the inventory (Figure 6). Honestly, with this I wasn’t sure where was best practiced to put exceptions; whether it was ok to put it in the main body of the script or if it should go in the function. I tried google some but I didn’t find anything useful. I opted for in the main body because it seemed the most straight forward to me.

A close up of text on a screen

Description automatically generated

Figure - Error handling at input for delete ID

# Pickling

I came back to working on the pickling. Before, I was trying to add unpickling at FileProcessor.read\_file and FileProcessor.save\_entry. After looking over the assignment document again, it said to “Modify the permanent data store to use binary data.” So that sounded like to me that the only place that needed pickling was at FileProcessor.save\_entry and that FileProcessor.read\_file should still read CDInventory.txt.

At first, I left a bunch of the code in save\_entry from when it was saving to a .txt file that formatting the data into rows in a table with commas separating CD ID, CD Name, and CD Artist (Figure 7).

A screenshot of a cell phone

Description automatically generated

Figure - Trying to keep too much from previous assignment 6

Of course, this wasn’t working. I tried several combinations and orders for the code before I finally realized I just needed the code for pickle.dump(). I removed everything else but save\_entry was still not saving a .dat file (Figure 8). After reviewing [tutorialspoint](https://www.tutorialspoint.com/python-pickling) (external site) with more details on pickling, I realized the problem was I had the file first and the object second in my arguments when it needed to be the other way around. Once I made that change, save\_entry would save the appropriate .dat file (Figure 9).

A screen shot of a computer

Description automatically generated

Figure - Removed excess code but still not working

A screenshot of a cell phone screen with text

Description automatically generated

Figure - Correct order for arguments

# Final Code

1. #------------------------------------------#
2. # Title: Assignment07.py
3. # Desc: Working with classes and functions.
4. # Change Log: (Who, When, What)
5. # DBiesinger, 2030-Jan-01, Created File
6. # KSorge-Toomey, 2020-Aug-19, Turned some tasks from while loops into functions within the appropriate class, Added docstrings
7. # Ksorge-Toomey, 2020-Aug-26, Added exception handling
8. # KSorge-Toomey, 2020-Aug-27, Changed to save as binary file
9. #------------------------------------------#
11. **import** pickle
13. # -- DATA -- #
14. strChoice = '' # User input
15. lstTbl = []  # list of lists to hold data
16. dicRow = {}  # list of data row
17. strFileName = 'CDInventory.txt'  # data storage file
18. strBinaryName = 'CDInventory.dat' # data storage binary file
19. objFile = None  # file object

22. # -- PROCESSING -- #
24. **class** DataProcessor:
25. """Handling data in memory"""
27. @staticmethod
28. **def** add\_user\_data(cd\_id, cd\_title, cd\_artist, table):
29. """Function to receive new entry data held in memory
31. Args:
32. cd\_id (int): entry id added by user
33. cd\_title (string): name of cd title added by user
34. cd\_artist (sting): name of cd artist added by user
35. table (list of dict): 2D data structure (list of dicts) that holds the data during runtime
37. Returns:
38. None.
39. """
40. intID = int(cd\_id)
41. dicRow = {'ID': intID, 'Title': cd\_title, 'Artist': cd\_artist}
42. table.append(dicRow)

45. @staticmethod
46. **def** delete\_entry(id\_to\_remove, table):
47. """Function to delete entry chosen by ID number
49. Args:
50. id\_to\_remove (int): entry id to delete
51. table (list of dict): 2d data structure (lists of dicts) that hold the data during runtime
53. Returns:
54. None.
55. """
56. intRowNr = -1
57. blnCDRemoved = False
58. **for** row **in** table:
59. intRowNr += 1
60. **if** row['ID'] == id\_to\_remove:
61. **del** table[intRowNr]
62. blnCDRemoved = True
63. **break**
64. **if** blnCDRemoved:
65. **print**('The CD was removed')
66. **else**:
67. **print**('Could not find this CD!')

70. **class** FileProcessor:
71. """Processing the data to and from binary file"""
73. @staticmethod
74. **def** read\_file(file\_name, table):
75. """Function to manage data ingestion from file to a list of dictionaries
77. Reads the data from file identified by file\_name into a 2D table
78. (list of dicts) table one line in the file represents one dictionary row in table.
80. Args:
81. file\_name (string): name of file used to read the data from
82. table (list of dict): 2D data structure (list of dicts) that holds the data during runtime
84. Returns:
85. None.
86. """
87. table.clear()  # this clears existing data and allows to load data from file
88. **try**:
89. objFile = open(file\_name, 'r')
90. **except** FileNotFoundError as e:
91. **print**('\nFile not found.')
92. **print**(e)
93. **else**:
94. **for** line **in** objFile:
95. data = line.strip().split(',')
96. dicRow = {'ID': int(data[0]), 'Title': data[1], 'Artist': data[2]}
97. table.append(dicRow)
98. objFile.close()
100. @staticmethod
101. **def** write\_file(file\_name):
102. """Function to create CDInventory.txt if file does not already exist
104. Args:
105. file\_name (string):name of file to create .txt
107. Returns:
108. None.
109. """
110. objFile = open(strFileName, 'a')
111. objFile.close()
113. @staticmethod
114. **def** save\_entry(file\_name, table):
115. """Function to save current entries in memory to file
117. Args:
118. file\_name (string): name of file to save data to
119. table (list of dict): 2D data structure (list of dicts) that hold the data during runtime
121. Returns:
122. None.
123. """
124. **if** strYesNo == 'y':
125. with open(file\_name,'wb') as fileObj:
126. pickle.dump(table, fileObj)
127. **print**('Inventory saved to .dat file.')
128. **else**:
129. input('The inventory was NOT saved to file. Press [ENTER] to return to the menu.')

132. # -- PRESENTATION (Input/Output) -- #
134. **class** IO:
135. """Handling Input / Output"""
137. @staticmethod
138. **def** print\_menu():
139. """Displays a menu of choices to the user
141. Args:
142. None.
144. Returns:
145. None.
146. """
147. **print**('Menu\n\n[l] Load Inventory from .txt file\n[a] Add CD\n[i] Display Current Inventory')
148. **print**('[d] Delete CD from Inventory\n[s] Save Inventory to binary file\n[x] Exit\n')
150. @staticmethod
151. **def** menu\_choice():
152. """Gets user input for menu selection
154. Args:
155. None.
157. Returns:
158. choice (string): a lower case sting of the users input out of the choices l, a, i, d, s or x
160. """
161. choice = ' '
162. **while** choice **not** **in** ['l', 'a', 'i', 'd', 's', 'x']:
163. choice = input('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(table):
169. """Displays current inventory table

172. Args:
173. table (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** table:
182. **print**('{}\t{} (by:{})'.format(\*row.values()))
183. **print**('======================================')
185. @staticmethod
186. **def** user\_entry():
187. """Gets user input for CD entry
189. Args:
190. None.
192. Returns:
193. None.
194. """
195. **try**:
196. strID = int(input('Enter ID: ').strip())
197. **except** ValueError as e:
198. **print**('That\'s not a number!')
199. **print**(e)
200. **pass**
201. **else**:
202. strTitle = input('What is the CD\'s title? ').strip()
203. stArtist = input('What is the Artist\'s name? ').strip()
204. **return** strID, strTitle, stArtist

207. # 1. When program starts,
208. FileProcessor.write\_file(strFileName)
209. FileProcessor.read\_file(strFileName, lstTbl)
211. # 2. start main loop
212. **while** True:
213. # 2.1 Display Menu to user and get choice
214. IO.print\_menu()
215. strChoice = IO.menu\_choice()
217. # 3. Process menu selection
218. # 3.1 process exit first
219. **if** strChoice == 'x':
220. **break**
221. # 3.2 process load inventory
222. **if** strChoice == 'l':
223. **print**('WARNING: If you continue, all unsaved data will be lost and the Inventory re-loaded from file.')
224. strYesNo = input('Type \'yes\' to continue and reload from file. otherwise reload will be canceled: ')
225. **if** strYesNo.lower() == 'yes':
226. **print**('Reloading...')
227. FileProcessor.read\_file(strFileName, lstTbl)
228. IO.show\_inventory(lstTbl)
229. **else**:
230. input('Canceling... Inventory data NOT reloaded. Press [ENTER] to continue to the menu.')
231. IO.show\_inventory(lstTbl)
232. **continue**  # start loop back at top.
233. # 3.3 process add a CD
234. **elif** strChoice == 'a':
235. # 3.3.1 Ask user for new ID, CD Title and Artist
236. cd\_id, cd\_title, cd\_artist = IO.user\_entry()
237. # 3.3.2 Add item to the table
238. DataProcessor.add\_user\_data(cd\_id, cd\_title, cd\_artist, lstTbl)
239. # 3.3.3 Display modified inventory to user
240. IO.show\_inventory(lstTbl)
241. **continue**  # start loop back at top.
242. # 3.4 process display current inventory
243. **elif** strChoice == 'i':
244. IO.show\_inventory(lstTbl)
245. **continue**  # start loop back at top.
246. # 3.5 process delete a CD
247. **elif** strChoice == 'd':
248. # 3.5.1 get Userinput for which CD to delete
249. # 3.5.1.1 display Inventory to user
250. IO.show\_inventory(lstTbl)
251. # 3.5.1.2 ask user which ID to remove
252. **try**:
253. intIDDel = int(input('Which ID would you like to delete? ').strip())
254. **except** ValueError as e:
255. **print**('\nThat\'s not a number.')
256. **print**(e)
257. # 3.5.2 search thru table and delete CD
258. **else**:
259. DataProcessor.delete\_entry(intIDDel, lstTbl)
260. # 3.5.3 Display modified inventory to user
261. IO.show\_inventory(lstTbl)
262. **continue**  # start loop back at top.
263. # 3.6 process save inventory to file
264. **elif** strChoice == 's':
265. # 3.6.1 Display current inventory and ask user for confirmation to save
266. IO.show\_inventory(lstTbl)
267. strYesNo = input('Save this inventory to binary file? [y/n] ').strip().lower()
268. # 3.6.2 Process choice
269. FileProcessor.save\_entry(strBinaryName, lstTbl)
270. **continue**  # start loop back at top.
271. # 3.7 catch-all should not be possible, as user choice gets vetted in IO, but to be safe:
272. **else**:
273. **print**('General Error')

# Running

A screenshot of a cell phone

Description automatically generated

Figure - Spyder output, loading inventory from .txt file

A screenshot of a cell phone screen with text

Description automatically generated

Figure - Entering non-integer at entering ID for delete

A screen shot of a computer

Description automatically generated

Figure - Saving inventory to .dat file

A screenshot of a cell phone

Description automatically generated

Figure - Saved .dat file

# Summary

In this week, I learned about pickling and working with text and binary files, as well as structured error handling. Applying this new knowledge, I successfully completed this week’s homework assignment.