Andrew Hanson

2020 Aug 27

IT FDN 110 B Su 20

Assignment07

Pickle, binary data, and error handling

# Introduction

This module covered pickle, binary data, and error handling. Pickle was used to store python objects in binary data. Error handling was implemented with the try except model. Additionally, there was some material about markdown.

# Data

The main data of the program is the CD inventory that is stored as a 2-d list of dictionaries. This data is worked with in memory and stored in a data file. By loading the file into memory, entries can be added and deleted and then the file can be overwritten with the new inventory. By using pickle, python objects can be stored in their native state which simplifies the process as opposed to storing the data in text format.

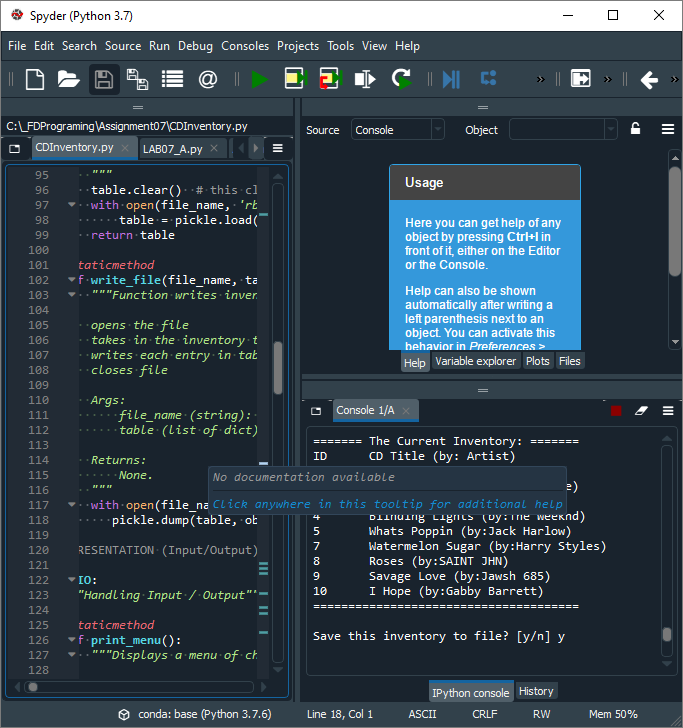


Figure - data loaded from inventory file (Spyder)

# Processing

## [l] load Inventory from file

The function read\_file in the class FileProcessor is responsible for loading data from the inventory file into memory. This allows easier manipulation of the file data. The function requires two arguments: filename and table. Using pickle has shortened the amount of code required for this function and simplified handling the data in other ways.

1. @staticmethod
2. **def** read\_file(file\_name, table):
3. """Function to manage data ingestion from file to a list of dictionaries
5. Reads the data from file identified by file\_name into a 2D table
6. (list of dicts) table one line in the file represents one dictionary row in table.
8. Args:
9. file\_name (string): name of file used to read the data from
10. table (list of dict): 2D data structure (list of dicts) that holds the data during runtime
12. Returns:
13. None.
14. """
15. table.clear()  # this clears existing data and allows to load data from file
16. with open(file\_name, 'rb') as objFile:
17. table = pickle.load(objFile) #note: load() loads one line of data
18. **return** table

Listing - code for read\_file function

## [a] Add CD

The function add\_2 in the class DataProcessor is responsible for making the user’s entries accessible by the program. It does this by organizing the user’s entries into a dictionary with keywords related to the purpose of the data. It then adds the dictionary into a list of similar dictionaries of user inputs to comprise the inventory data. The function takes in two argument: userinputs and table. This function has been modified to work with a list of tuples so that multiple entries can be passed to it at once.

1. @staticmethod
2. **def** add\_2(userinputs, table):
3. """proccesses user inputs into dictionary entries and puts dictionary in table
4. Appends dictionary of user inputs to lstTbl as global.
6. loops through list of tuples
7. unpacks tuple of user inputs
8. sorts indiviual user inputs into dictionary entries
9. appends dictionary into table
11. Args:
12. lstTplUserinput (list of tuples): 2D data structure (list of tuples) that holds multiple user input tuples. from add\_1
13. table (list of dict): 2D data structure (list of dicts) that holds the data during runtime.
15. Returns:
16. None.
17. """
18. **for** item **in** userinputs:
19. intID = int(item[0])
20. strTitle = item[1]
21. strArtist = item[2]
22. dicRow = {'ID': intID, 'Title': strTitle, 'Artist': strArtist}
23. table.append(dicRow)

Listing - code for add\_2 function

## [d] delete CD from Inventory

The function delete\_entry in the class DataProcessor is responsible for removing entries from the inventory table in memory. The function cycles through the dictionaries in the table looking for an entry with a matching ID. If it finds one, it will remove that entry. User feedback is given based on whether a matching ID is found. The inventory table is also displayed before and after the delete function is run to show the user what is available to delete and what the new inventory table looks like. The function takes two arguments: ID and table.

1. @staticmethod
2. **def** delete\_entry(ID, table):
3. """Deletes an entry from memory.
4. Modifies table to remove selected entry.
6. Args:
7. table (list of dict): 2D data structure (list of dicts) that holds the data during runtime
8. ID (integer): numerical ID for dict entry
10. Returns:
11. None.
12. """
13. intRowNr = -1
14. blnCDRemoved = False
15. **for** row **in** table:
16. intRowNr += 1
17. **if** row['ID'] == ID:
18. **del** lstTbl[intRowNr]
19. blnCDRemoved = True
20. **break**
21. **if** blnCDRemoved:
22. **print**('The CD was removed')
23. **else**:
24. **print**('Could not find this CD!')

Listing - code for delete\_entry function

## [s] Save Inventory to file

The function write\_file in the class FileProcessor is responsible for overwriting the inventory file with the table inventory. Overwriting the file allows deleting entries from the file by deleting them from the table inventory which is stored in memory. The function takes two arguments: file\_name and table. By using pickle this function has been simplified.

1. @staticmethod
2. **def** write\_file(file\_name, table):
3. """Function writes inventory to file
5. opens the file
6. takes in the inventory table
7. writes each entry in table on seperate line
8. closes file
10. Args:
11. file\_name (string): name of file used to read the data from
12. table (list of dict): 2D data structure (list of dicts) that holds the data during runtime
14. Returns:
15. None.
16. """
17. with open(file\_name, 'wb') as objFile:
18. pickle.dump(table, objFile)

Listing - code for write\_file function

# Presentation

## The menu

The programs user interactivity is handled mostly by the main menu. One function is responsible for displaying the menu choices to the user. Another function is responsible for taking asking for user input and returning their choice. Some parts of the program have localized menus or user choices not handled by the main menu.

### print\_menu

The function print\_menu in the class IO is responsible for making the user aware of their options and how to interact with the program. Placing the menu in a function allows additional functionality to be more easily added and helps with the separation of concerns. The options are: l, a, i, d, s or x.

1. @staticmethod
2. **def** print\_menu():
3. """Displays a menu of choices to the user
5. Args:
6. None.
8. Returns:
9. None.
10. """
12. **print**('Menu\n\n[l] load Inventory from file\n[a] Add CD\n[i] Display Current Inventory')
13. **print**('[d] delete CD from Inventory\n[s] Save Inventory to file\n[x] exit\n')

Listing - code for print\_menu function

### menu\_choice

the function menu\_choice in the class IO is responsible for asking for the user’s choice input and returning it to the while loop so that the program can fulfil their desires. This is the main interface by which the user interacts with the program.

1. @staticmethod
2. **def** menu\_choice():
3. """Gets user input for menu selection
5. Args:
6. None.
8. Returns:
9. choice (string): a lower case sting of the users input out of the choices l, a, i, d, s or x
11. """
12. choice = ' '
13. **while** choice **not** **in** ['l', 'a', 'i', 'd', 's', 'x']:
14. choice = input('Which operation would you like to perform? [l, a, i, d, s or x]: ').lower().strip()
15. **print**()  # Add extra space for layout
16. **return** choice

Listing - code for the menu\_choice function

## [a] Add CD

The function add\_1 in the class IO is responsible for taking user inputs and passing them to the add\_2 function. It asks the user for inputs for the CD’s ID, name, and artist. This function has been modified to loop so that multiple entries may be made by the user without returning to the main menu. The function returns a list of tupled values: strID, strTitle, and strArtist.

1. @staticmethod
2. **def** add\_1():
3. """
4. takes initial user input
5. checks that the ID entry is an Integer before return.
6. loops until input is 'back'
7. returns the user's inputs as a list of tuples.
9. Args:
10. None.
12. Returns:
13. lstTplUserinput (list of tuples): 2D data structure (list of tuples) that holds multiple user input tuples.
15. """
16. #TODO: check for duplicate entries
17. #TODO: check for empty fields
18. **print**('input "BACK" to exit')
19. lstTplUserinputs = []
20. **while** True:
21. **while** True:
22. strID = input('Enter ID: ').strip()
23. **if** strID.lower().strip() == 'back':
24. **break**
25. **try**:
26. int(strID)
27. **break**
28. **except**(ValueError):
29. **print**('Invalid entry, please enter a number.')
30. **continue**
31. **if** strID.lower().strip() == 'back':
32. **break**
33. strTitle = input('What is the CD\'s title? ').strip()
34. **if** strTitle.lower().strip() == 'back':
35. **break**
36. strArtist = input('What is the Artist\'s name? ').strip()
37. **if** strArtist.lower().strip() == 'back':
38. **break**
39. tplUserinputs = (strID,strTitle,strArtist)
40. lstTplUserinputs.append(tplUserinputs)
41. **return** lstTplUserinputs

Listing - code for add\_1 function

## [i] Display Current Inventory

The function show\_inventory in the class IO is responsible for printing the inventory table for the user’s viewing pleasure. This allows the user to know what entries are already on the list. This function takes one argument: table.

1. @staticmethod
2. **def** show\_inventory(table):
3. """Displays current inventory table
5. Args:
6. table (list of dict): 2D data structure (list of dicts) that holds the data during runtime.
8. Returns:
9. None.
11. """
12. #TODO:sort inventory by ID prior to display
13. **print**('======= The Current Inventory: =======')
14. **print**('ID\tCD Title (by: Artist)')
15. **for** row **in** table:
16. **print**('{}\t{} (by:{})'.format(\*row.values()))
17. **print**('======================================')

Listing - code for the show\_inventory function

# Challenges

It took some time and experimentation to understand how pickle worked and how to format my code to work with a data file, however once it was implemented it made handling file data simpler because the data is already stored as a list when it is read from the file. No text formatting or parsing is needed.

Working with the error handling was fairly simple for me because I have used it in earlier assignments. There are also parts of the program that prevent errors from happening without the use of the try except model.

# Summary

This module taught us about exceptions and custom exceptions. We also learned about pickle and binary data. We learned about data files. We also learned about the markdown language. We learned about using specific exceptions.

# Appendix

Sources:

<https://www.youtube.com/watch?v=2Tw39kZIbhs&t=264s> (pickle)

<https://www.youtube.com/watch?v=NIWwJbo-9_8> (error handling)

## Repository

<https://github.com/clichekhfan/Assignment07>

## questions

### • What are the benefits of using structured error handling?

Structure error handling can prevent your program from crashing and make the program more responsive to user input.

### • What are the differences between a text file and a binary file?

Text files store textual data. Binary files can store any kind of data. With pickle, python objects can be stored in binary code.

### • How is the Exception class used?

The exception class can be used to create custom child classes of the exception class or to find more detail about a specific error.

### • How do you "derive" a new class from the Exception class?

By creating a class that inherits from the exception class.

### • When might you create a class derived from the Exception class?

When you want to show the user an error that is not already part of the standard exceptions available.

### • What is the Markdown language?

Markdown is a way of formatting text.

## CDInventory.py

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. # AHanson, 2020-Aug-15, moved code into functions
7. # AHanson, 2020-Aug-15, added docstring to new functions.
8. # AHanson, 2020-Aug-15, added @static method to functions.
9. # AHanson, 2020-Aug-24, fixed bug with non integer inputs for IDs
10. # AHanson, 2020-Aug-25, amended program to use .dat file instead of .txt
11. # AHanson, 2020-Aug-25, amended program to loop when requesting user entries to allow multiple user entries

14. #------------------------------------------#
15. **import** pickle
17. # -- DATA -- #
18. strChoice = '' # User input
19. lstTbl = []  # list of lists to hold data
20. dicRow = {}  # list of data row
21. strFileName = 'CDInventory.dat'  # data storage file
22. objFile = None  # file object
23. # lstTbl = [{'ID':1,'title':'WAP','artist':'Cardi B'},{'ID':2,'title':'Laugh Now Cry Later','artist':'Drake'},{'ID':3,'title':'Rockstar','artist':'DaBaby'},{'ID':4,'title':'Blinding Lights','artist':'The Weeknd'},{'ID':5,'title':'Whats Poppin','artist':'Jack Harlow'},{'ID':6,'title':'7 Summers','artist':'Morgan Wallen'},{'ID':7,'title':'Watermelon Sugar','artist':'Harry Styles'},{'ID':8,'title':'Roses','artist':'SAINT JHN'},{'ID':9,'title':'Savage Love','artist':'Jawsh 685'},{'ID':10,'title':'I Hope','artist':'Gabby Barrett'}]
25. # -- PROCESSING -- #
26. **class** DataProcessor:
28. @staticmethod
29. **def** add\_2(userinputs, table):
30. """proccesses user inputs into dictionary entries and puts dictionary in table
31. Appends dictionary of user inputs to lstTbl as global.
33. loops through list of tuples
34. unpacks tuple of user inputs
35. sorts indiviual user inputs into dictionary entries
36. appends dictionary into table
38. Args:
39. lstTplUserinput (list of tuples): 2D data structure (list of tuples) that holds multiple user input tuples. from add\_1
40. table (list of dict): 2D data structure (list of dicts) that holds the data during runtime.
42. Returns:
43. None.
44. """
45. **for** item **in** userinputs:
46. intID = int(item[0])
47. strTitle = item[1]
48. strArtist = item[2]
49. dicRow = {'ID': intID, 'Title': strTitle, 'Artist': strArtist}
50. table.append(dicRow)

53. @staticmethod
54. **def** delete\_entry(ID, table):
55. """Deletes an entry from memory.
56. Modifies table to remove selected entry.
58. Args:
59. table (list of dict): 2D data structure (list of dicts) that holds the data during runtime
60. ID (integer): numerical ID for dict entry
62. Returns:
63. None.
64. """
65. intRowNr = -1
66. blnCDRemoved = False
67. **for** row **in** table:
68. intRowNr += 1
69. **if** row['ID'] == ID:
70. **del** lstTbl[intRowNr]
71. blnCDRemoved = True
72. **break**
73. **if** blnCDRemoved:
74. **print**('The CD was removed')
75. **else**:
76. **print**('Could not find this CD!')

79. **class** FileProcessor:
80. """Processing the data to and from text file"""
82. @staticmethod
83. **def** read\_file(file\_name, table):
84. """Function to manage data ingestion from file to a list of dictionaries
86. Reads the data from file identified by file\_name into a 2D table
87. (list of dicts) table one line in the file represents one dictionary row in table.
89. Args:
90. file\_name (string): name of file used to read the data from
91. table (list of dict): 2D data structure (list of dicts) that holds the data during runtime
93. Returns:
94. None.
95. """
96. table.clear()  # this clears existing data and allows to load data from file
97. with open(file\_name, 'rb') as objFile:
98. table = pickle.load(objFile) #note: load() loads one line of data
99. **return** table
101. @staticmethod
102. **def** write\_file(file\_name, table):
103. """Function writes inventory to file
105. opens the file
106. takes in the inventory table
107. writes each entry in table on seperate line
108. closes file
110. Args:
111. file\_name (string): name of file used to read the data from
112. table (list of dict): 2D data structure (list of dicts) that holds the data during runtime
114. Returns:
115. None.
116. """
117. with open(file\_name, 'wb') as objFile:
118. pickle.dump(table, objFile)
120. # -- PRESENTATION (Input/Output) -- #
122. **class** IO:
123. """Handling Input / Output"""
125. @staticmethod
126. **def** print\_menu():
127. """Displays a menu of choices to the user
129. Args:
130. None.
132. Returns:
133. None.
134. """
136. **print**('Menu\n\n[l] load Inventory from file\n[a] Add CD\n[i] Display Current Inventory')
137. **print**('[d] delete CD from Inventory\n[s] Save Inventory to file\n[x] exit\n')
139. @staticmethod
140. **def** menu\_choice():
141. """Gets user input for menu selection
143. Args:
144. None.
146. Returns:
147. choice (string): a lower case sting of the users input out of the choices l, a, i, d, s or x
149. """
150. choice = ' '
151. **while** choice **not** **in** ['l', 'a', 'i', 'd', 's', 'x']:
152. choice = input('Which operation would you like to perform? [l, a, i, d, s or x]: ').lower().strip()
153. **print**()  # Add extra space for layout
154. **return** choice
156. @staticmethod
157. **def** show\_inventory(table):
158. """Displays current inventory table

161. Args:
162. table (list of dict): 2D data structure (list of dicts) that holds the data during runtime.
164. Returns:
165. None.
167. """
168. **print**('======= The Current Inventory: =======')
169. **print**('ID\tCD Title (by: Artist)')
170. **for** row **in** table:
171. **print**('{}\t{} (by:{})'.format(\*row.values()))
172. **print**('======================================')
174. @staticmethod
175. **def** add\_1():
176. """
177. takes initial user input
178. checks that the ID entry is an Integer before return.
179. loops until input is 'back'
180. returns the user's inputs as a list of tuples.
182. Args:
183. None.
185. Returns:
186. lstTplUserinput (list of tuples): 2D data structure (list of tuples) that holds multiple user input tuples.
188. """
189. **print**('input "BACK" to exit')
190. lstTplUserinputs = []
191. **while** True:
192. **while** True:
193. strID = input('Enter ID: ').strip()
194. **if** strID.lower().strip() == 'back':
195. **break**
196. **try**:
197. int(strID)
198. **break**
199. **except**(ValueError):
200. **print**('Invalid entry, please enter a number.')
201. **continue**
202. **if** strID.lower().strip() == 'back':
203. **break**
204. strTitle = input('What is the CD\'s title? ').strip()
205. **if** strTitle.lower().strip() == 'back':
206. **break**
207. strArtist = input('What is the Artist\'s name? ').strip()
208. **if** strArtist.lower().strip() == 'back':
209. **break**
210. tplUserinputs = (strID,strTitle,strArtist)
211. lstTplUserinputs.append(tplUserinputs)
212. **return** lstTplUserinputs
214. # 0. Create inventory file if none exists
215. with open(strFileName, 'a') as objFile:
216. **pass**
218. # 1. When program starts, read in the currently saved Inventory
219. lstTbl = FileProcessor.read\_file(strFileName, lstTbl)
221. # 2. start main loop
222. **while** True:
223. # 2.1 Display Menu to user and get choice
224. IO.print\_menu()
225. strChoice = IO.menu\_choice()
226. # 3. Process menu selection
227. # 3.1 process exit first
228. **if** strChoice == 'x':
229. **break**
230. # 3.2 process load inventory
231. **if** strChoice == 'l':
232. **print**('WARNING: If you continue, all unsaved data will be lost and the Inventory re-loaded from file.')
233. strYesNo = input('type \'yes\' to continue and reload from file. otherwise reload will be canceled: ')
234. **if** strYesNo.lower() == 'yes':
235. **print**('reloading...')
236. lstTbl = FileProcessor.read\_file(strFileName, lstTbl)
237. IO.show\_inventory(lstTbl)
238. **else**:
239. input('canceling... Inventory data NOT reloaded. Press [ENTER] to continue to the menu.')
240. IO.show\_inventory(lstTbl)
241. **continue**  # start loop back at top.
242. # 3.3 process add a CD
243. **elif** strChoice == 'a':
244. # 3.3.1 Ask user for new ID, CD Title and Artist
245. lstTplUserInputs = IO.add\_1()
246. # 3.3.2 Add item to the table
247. DataProcessor.add\_2(lstTplUserInputs, lstTbl)
248. IO.show\_inventory(lstTbl)
249. **continue**  # start loop back at top.
250. # 3.4 process display current inventory
251. **elif** strChoice == 'i':
252. IO.show\_inventory(lstTbl)
253. **continue**  # start loop back at top.
254. # 3.5 process delete a CD
255. **elif** strChoice == 'd':
256. # 3.5.1 get Userinput for which CD to delete
257. # 3.5.1.1 display Inventory to user
258. IO.show\_inventory(lstTbl)
259. # 3.5.1.2 ask user which ID to remove
260. intIDDel = int(input('Which ID would you like to delete? ').strip())
261. # 3.5.2 search thru table and delete CD
262. DataProcessor.delete\_entry(intIDDel, lstTbl)
263. IO.show\_inventory(lstTbl)
264. **continue**  # start loop back at top.
265. # 3.6 process save inventory to file
266. **elif** strChoice == 's':
267. # 3.6.1 Display current inventory and ask user for confirmation to save
268. IO.show\_inventory(lstTbl)
269. strYesNo = input('Save this inventory to file? [y/n] ').strip().lower()
270. # 3.6.2 Process choice
271. **if** strYesNo == 'y':
272. # 3.6.2.1 save data
273. FileProcessor.write\_file(strFileName, lstTbl)
274. **else**:
275. input('The inventory was NOT saved to file. Press [ENTER] to return to the menu.')
276. **continue**  # start loop back at top.
277. # 3.7 catch-all should not be possible, as user choice gets vetted in IO, but to be save:
278. **else**:
279. **print**('General Error')