Tiffany Hou  
2020 March 9  
IT FDN 100B  
Assignment 07

Module 7

# Overview

In this module, we dive deeper into working with text files, reading data, and how we can use binary files to store data instead (pickling). We also learn about structured error handling.

On a larger scale, we are also asked to explore and find resources outside of the materials provided by the class. So much of programming is being able to research and find answers by yourself, so we are encouraged to do that and start identifying what kind of sources are helpful to us. I reference several resources for each of our main topics (pickling, structured error handling) in the overview below and briefly note what I liked about each resource in the footnotes.

# Key Concepts

## Binary Files & Pickling

To further conserve resources, we can also store data in binary versus human-readable plaintext. Through a process known as *serialization* and *deserialization[[1]](#footnote-1)*, we can save more than just values; we can save Python objects. The pickle module takes an object, encodes it into a binary byte stream to store (*pickling)*, then transforms it back into plaintext when we need to access it again (*unpickling*)[[2]](#footnote-2).

Because an entire object is being stored, it can actually be used in another Python script[[3]](#footnote-3). We’re not just saving the value of an object, but we can save what the object actually is (e.g. 7 by itself doesn’t mean anything, but with pickling, we can now note that it is an integer representing a CD ID). This means we can preserve data between user sessions[[4]](#footnote-4), transfer it to a different application altogether, or pass it through a network to someone else.

The pickle module has several advantages over another Python serialization module, marshal[[5]](#footnote-5):

* Tracks data that has already been serialized, meaning it can handle *recursive objects* (objects that refer to themselves)
* Can manage *object sharing* (objects being referred in multiple places, so it references the original object that has already been serialized instead of reserializing it as a new object)
* Supports the serialization of *user-defined classes* (the custom classes we create) as long as the classes are importable and are in the same module the object is stored in

Pickling and unpickling data does have its disadvantages – most notably, the module offers no protection against malicious or poorly constructed data. This is a security concern and also means our script could crash if we import an object that is incompatible with our script (e.g. what if the imported object doesn’t match the structure of lstTbl in our CDInventory.py script?).

## Structured Error Handling

While it is impossible to foresee all possibilities where things could go wrong and crash our script, we can use s*tructured error handling* to try to address the most probable ones. These are most likely to happen when there is user input, file interaction, or function calls.

Try except[[6]](#footnote-6) blocks are a good way to address these scenarios.

|  |
| --- |
| try:  *code to be executed* except *ExceptionType*:  *code to be executed if error* |

While the exception type is optional, but it is best to be as specific as possible about the kind of error we are expecting. This way, we can better address unexpected errors if they do happen[[7]](#footnote-7).

A try except block can have multiple except clauses if you want different actions to be taken depending on error. You can also have multiple exception types for a within a single except clause if you want the same action to be taken for those errors[[8]](#footnote-8). Like other statement blocks, exceptions should be stated in the order we want them caught[[9]](#footnote-9).

Beyond the [built-in exceptions](https://docs.python.org/3.7/library/exceptions.html#bltin-exceptions)[[10]](#footnote-10), we can also define and use custom exception classes. These can be used to address limitations that we want imposed[[11]](#footnote-11). These are particularly helpful if we need to store or display information about an exception.

# CDInventory.PY

This week, we continue to expand and modify our CD Inventory code from Module 6 to add structured error handling for user inputs and use pickling to store the inventory as binary data.

Prior to working on these features, I cleaned up my submission from Assignment 6 based on the feedback I received from Doug Klos. This primarily comprised of passing in global variables as local variables for all functions and not referring to functions from within another function, just to avoid the classes from crossing with one another. I also modified my CD ID auto-generator so it starts with ID 1 to check if it’s available.

As usual, code snippets have been formatted using PlanetB’s syntax highlighter[[12]](#footnote-12).

## Pickling

To implement pickling, we begin by importing the pickle module at the start of our script:

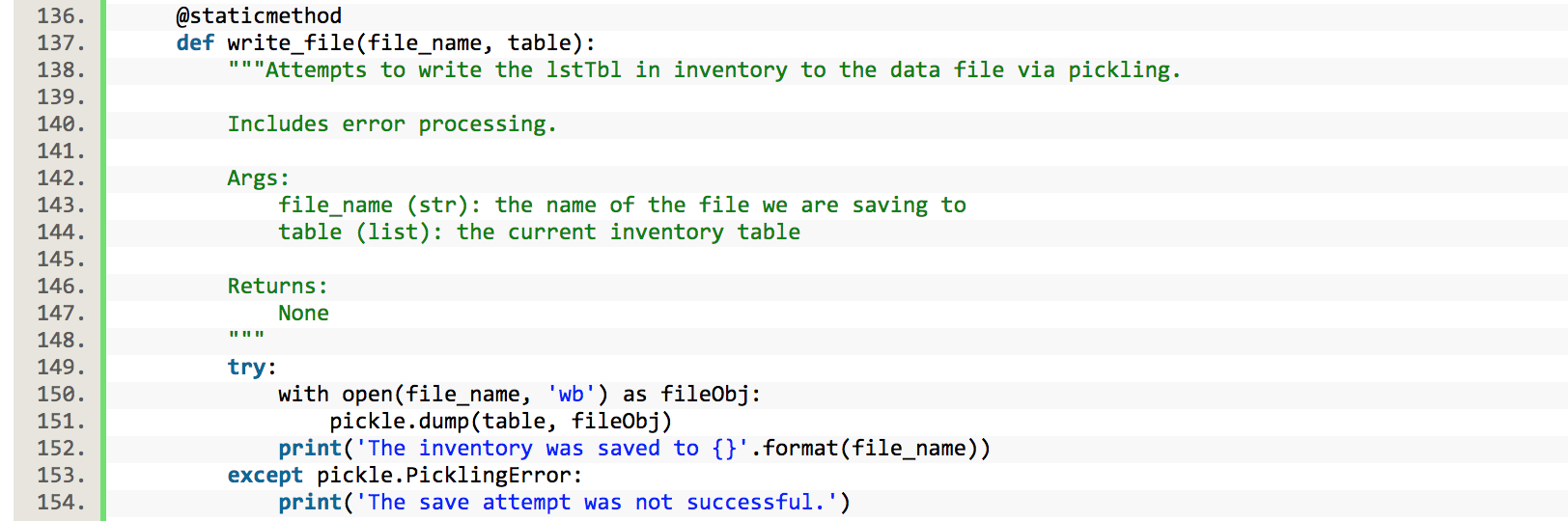


Listing 1 - Importing the pickle module

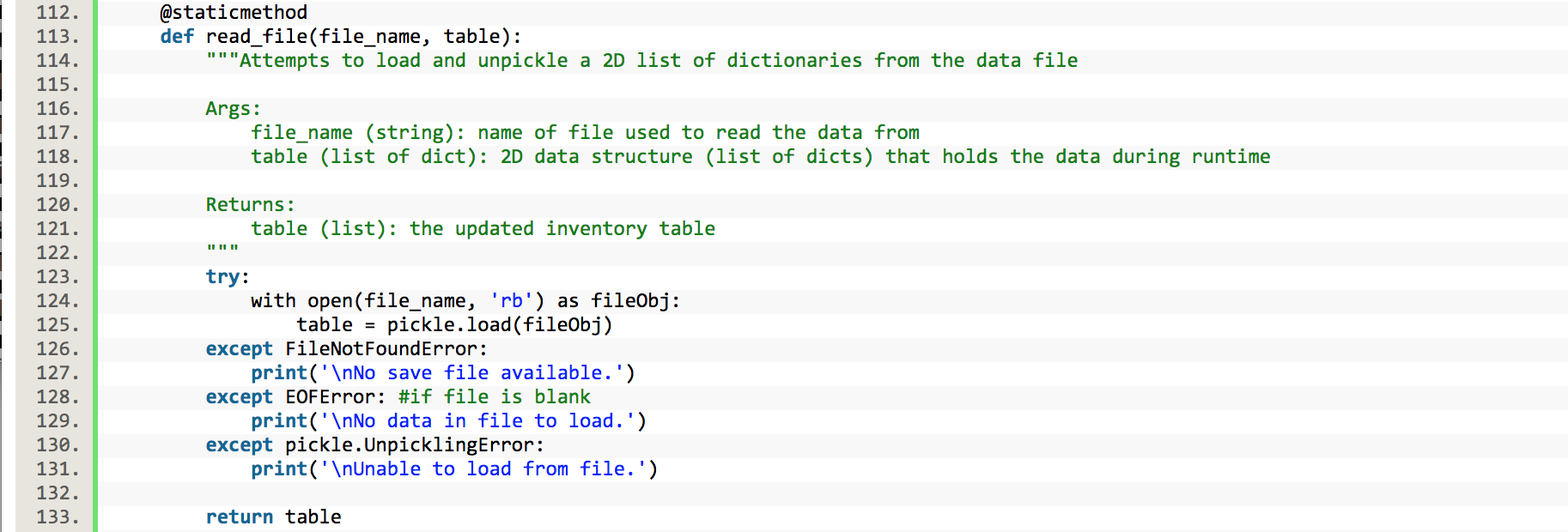
Next, we have to update our read and write functions. The previous functionality of the script was to:

* overwrite the entire save file when saving the current inventory
* overwrite the entire current inventory when loading from a save

Compounded with the pickle.load() command, which only reads one line at a time, and the fact that pickling saves the object information, each function ended up being quite simple: dump or load the entire table. The fact that the inventory is a 2D-list doesn’t matter since the entire list object is pickled.



Listing 2 – Pickling the Inventory table in write\_file()



Listing 3 - Unpickling the data in read\_file()

The error handling for the two functions above will be discussed in the section below.

## Structured Error Handling in the Code

We are asked to add in structured error handling for areas where:

1. There is user interaction, such as when:
   1. Entering a menu choice
   2. Entering a CD ID
   3. Entering the CD entry fields (ID, title, artist)
2. Type casting is needed (in our case, turning a string into an integer) for the:
   1. CD ID of the entry to be deleted
   2. CD ID of added entries (this is not an issue in my code with the auto-generator ID, but I’ll discuss it below with type casting)
3. File handling happens, such as:
   1. Within the write\_file() function when saving to an external file
   2. Within the read\_file() function when loading from an external file

Not every one of these operations will require error handling, so I will look at each of the above to decide if it is necessary or if it has already been addressed in other way.

### Error Handling for User Interaction

In the original Assignment 6 starter code, we have 3 areas for user interaction to look at to see if additional error handling is required:

* **Entering a menu choice**: IO.menu\_choice() and the while loop already vet the user inputs to see if they are valid entries, so no additional error handling is required here.
* **Entering the CD entry fields (ID, title, artist)**: The title and artist are entered as strings and remain as such. If we were manually entering IDs, we would normally need to be cast as an integer (thus needing error handling), but our auto-ID generator function takes care of that.  
  We could also create an custom error class to check the CD IDs (e.g. unique values, has to be greater than 0, etc.).
* **Entering a CD ID**: we will need some sort of error handling here when the user enters the CD ID of the entry they want deleted.

#### entering a CD ID

First, we ask for the user input in the main body of the script. Once we get it, we pass it into the get\_int\_input() function, which handles the type casting (see below). If we are returned a valid ID, it’s then passed into the delete\_entry() function.

A screenshot of a cell phone

Description automatically generated

Listing 4 - Deleting an Entry

### Error Handling for Type Casting

Type casting the CD ID is a good way for the script to crash. Normally, if a string cannot be turned into an integer, the script raises a ValueError.

There are two potential points where this could happen:

1. When entering the CD ID of the entry to be deleted
2. When adding the CD ID for new entries  
   (*Due to our auto-generator, this is not an issue. However, the solution used below can be used for to address this.*)

To address the ValueError, we can use a try except block to attempt to cast the input as an integer, then an exception if it cannot be done.

A screenshot of a social media post

Description automatically generated

Listing 5 - get\_int\_input()

Since this is separate from the actual deleting function, this could be used anytime an input needs to be cast as an integer. If the value is incompatible, it informs the user and returns a value of None.

The get\_int\_input() is really only used when we’re trying to get the ID of an entry to be deleted. However, if we didn’t have the ID auto-generator function when adding a new CD ID, we could also call this function there.

### Error Handling for File Handling

In regards to file handling, we need structured error handling:

* Within the write\_file() function when saving to an external file
* Within the read\_file() function when loading from an external file

#### Error Handling When Loading with read\_file()

As noted in last week’s feedback, while I had a try except statement for my initial load in the main body, which catches if there is not an existing CDInventory.txt file when the script first starts. However, it did not catch it if the user tries to load from the menu choice.

To address this, I moved the try except statement to within the read\_file() function. Now both the initial load and manual selection from menu directly call the script:

A picture containing screenshot

Description automatically generated

Listing 6 - Initial Load in CDInventory.py

A screenshot of a cell phone

Description automatically generated

Listing 7 - Load Inventory Selection from Menu in CDInventory.py

As can be seen in listing 3 (shown again below), we have an except clause to address the FileNotFoundError (if the file does not exist). I also added some other common errors that could happen with pickling:

* EOFError – if the file does exist, but is blank (the end of the file (EOF) is reached without any data being read)
* UnpicklingError – if the file does exist and contains data, but the data cannot be unpickled

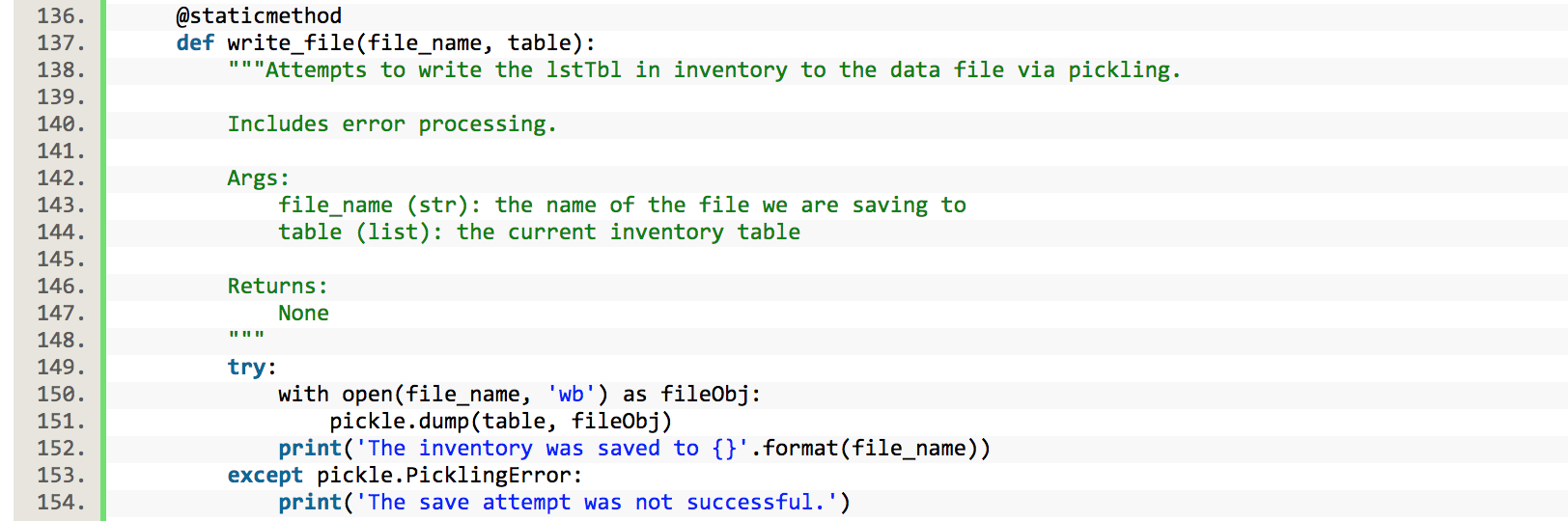
A screenshot of a cell phone

Description automatically generated

Listing 3 - Unpickling the data in read\_file()

#### Error Handling when saving with write\_file()

While we’re dumping the entire inventory table, which is top-level and picklable, there are certain objects that are unpicklable. While this is unlikely to happen with our table, I added a try except block that raises an exception if there is a PicklingError[[13]](#footnote-13) (just in case).



Listing 2 – Pickling the Inventory table in write\_file()

# Summary

In this week’s module, we update our submission from the previous week to incorporate what we learned about serialization of data with the pickle module and add structured error handling to cover the most probable errors that could occur.

Pickling is a very handy way to easily save Python objects. Last week, it took us 6 lines and a for loop to open a file, write each line within a 2-D list to a text file, and then close the file again. Pickling allows us to open, write the entire table object, and then close the file again with just 2 lines. Again, while it does have its disadvantages (security, data structure), it’s a great introduction to help us understand the portability of objects.

Structured error handling allows us to address errors that happen rather than the script simply crashing and ending. While we try to design our code and functions to prevent errors from user interactions or impossible actions, structured error handling can help when we miss something.

## Screenshots of Results

Screenshots of the script running in both Spyder and Terminal are available in the Appendix.

## GitHub Repository

Repository link: <https://github.com/tiffhou/Assignment_07>

# Appendix

## Listing – 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. # THou, 2020-Feb-27, added try statement to check for existing save file;
7. #   added functions for adding entry, deleting entry; updated display
8. # THou, 2020-Feb-29, added auto-generating ID for new CD entries; cleaned up formatting
9. # THou, 2020-Mar-02, added more formatting and comments; tweaked naming conventions;
10. #   added get\_int\_input()
11. # THou, 2020-Mar-07, incorporated feedback from mod 6 submission;
12. #   added error handling for initial load, loading from file;
13. #   imported the pickle module and implemented in load/save functions
14. # THou, 2020-Mar-09, updated docstrings; updated data file to .dat
15. #------------------------------------------#
17. **import** pickle
19. # -- DATA -- #
20. strChoice = '' # User input
21. lstTbl = []  # list of lists to hold data
22. dicRow = {}  # list of data row
23. strFileName = 'CDInventory.dat'  # data storage file
24. objFile = None  # file object
25. intID = '' #variable to hold a CD ID
26. strTitle = '' #variable to hold a CD title
27. strArtist = '' #variable to hold a CD's artist
29. # -- PROCESSING -- #
30. **class** DataProcessor:
31. """processess data in memory"""
33. @staticmethod
34. **def** add\_entry(cd\_id, cd\_title, cd\_artist, inv\_table):
35. """Adds the CD entry to a dictionary and appends it to the lstTbl.
37. Args:
38. cd\_id (int): CD ID to be added to the dictionary
39. cd\_title (string): CD title to be added to the dictionary
40. cd\_artist (string): CD artist to be added to the dictionary
41. inv\_table (list): current inventory list
43. Return:
44. inv\_table (list): updated inventory list
45. """
46. dicRow = {'ID': cd\_id, 'Title': cd\_title, 'Artist': cd\_artist}
47. inv\_table.append(dicRow)
48. **return** inv\_table
50. @staticmethod
51. **def** delete\_entry(delID, table):
52. """Deletes a CD entry based on inputted ID (delID).
54. Args:
55. delID (int): the ID inputted by the user for the entry to be deleted
56. table (list): the current inventory list
58. Return:
59. table (list): the updated inventory list
60. """
61. intRowNr = -1
62. blnCDRemoved = False
63. **for** row **in** table:
64. intRowNr += 1
65. **if** row['ID'] == delID:
66. **del** lstTbl[intRowNr]
67. blnCDRemoved = True
68. **break**
69. **if** blnCDRemoved:
70. **print**('The CD was removed.\n')
71. **else**:
72. **print**('Could not find this CD!\n')
73. **return** table
75. @staticmethod
76. **def** generate\_id(inv\_table):
77. """Generates a CD ID based on lstTbl length. Starts at 1 and increments until CD ID is a unique value.
79. Args:
80. inv\_table (list): the current inventory list
82. Returns:
83. cd\_id (int): the auto-generated CD ID
84. """
85. cd\_id = '' #local variable for generating CD ID
86. cd\_id = 1
87. **while** any(cd\_id == row['ID'] **for** row **in** inv\_table):
88. cd\_id = cd\_id + 1
89. **return** cd\_id
91. @staticmethod
92. **def** get\_int\_input(input\_val):
93. """Attempts to cast the inputted value as an integer.
95. Args:
96. input\_val (string): the inputted value
98. Returns:
99. int\_val (int): the input value cast as an integer
100. """
101. **try**:
102. int\_val = int(input\_val)
103. **return** int\_val
104. **except** ValueError:
105. **print** ('Input must be an integer. Returning to menu.')
106. **return** None

109. **class** FileProcessor:
110. """Processing the data to and from text file"""
112. @staticmethod
113. **def** read\_file(file\_name, table):
114. """Attempts to load and unpickle a 2D list of dictionaries from the data file
116. Args:
117. file\_name (string): name of file used to read the data from
118. table (list of dict): 2D data structure (list of dicts) that holds the data during runtime
120. Returns:
121. table (list): the updated inventory table
122. """
123. **try**:
124. with open(file\_name, 'rb') as fileObj:
125. table = pickle.load(fileObj)
126. **except** FileNotFoundError:
127. **print**('\nNo save file available.')
128. **except** EOFError: #if file is blank
129. **print**('\nNo data in file to load.')
130. **except** pickle.UnpicklingError:
131. **print**('\nUnable to load from file.')
132. **return** table

135. @staticmethod
136. **def** write\_file(file\_name, table):
137. """Attempts to write the lstTbl in inventory to the data file via pickling.
139. Includes error processing.
141. Args:
142. file\_name (str): the name of the file we are saving to
143. table (list): the current inventory table
145. Returns:
146. None
147. """
148. **try**:
149. with open(file\_name, 'wb') as fileObj:
150. pickle.dump(table, fileObj)
151. **print**('The inventory was saved to {}'.format(file\_name))
152. **except** pickle.PicklingError:
153. **print**('The save attempt was not successful.')

156. # -- PRESENTATION (Input/Output) -- #
157. **class** IO:
158. """Handling Input / Output"""
160. @staticmethod
161. **def** print\_menu():
162. """Displays a menu of choices to the user
164. Args:
165. None.
167. Returns:
168. None.
169. """
170. **print**('\n\n[[ Menu ]]\n\n[l] load Inventory from file\n[a] Add CD\n[i] Display Current Inventory')
171. **print**('[d] delete CD from Inventory\n[s] Save Inventory to file\n[x] exit\n')
173. @staticmethod
174. **def** menu\_choice():
175. """Gets user input for menu selection.
177. Args:
178. None.
180. Returns:
181. choice (string): a lower case sting of the users input out of the choices l, a, i, d, s or x
182. """
183. choice = ' '
184. **while** choice **not** **in** ['l', 'a', 'i', 'd', 's', 'x']:
185. choice = input('Which operation would you like to perform? [l, a, i, d, s or x]: ').lower().strip()
186. **print**()  # Add extra space for layout
187. **return** choice
189. @staticmethod
190. **def** show\_inventory(table):
191. """Displays current inventory table.
193. Args:
194. table (list of dict): 2D data structure (list of dicts) that holds the data during runtime.
196. Returns:
197. None.
198. """
199. **if** len(table) > 0:
200. **print**('======= The Current Inventory: =======')
201. **print**('ID\tCD Title (by: Artist)\n')
202. **for** row **in** table:
203. **print**('{}\t{} (by: {})'.format(\*row.values()))
204. **print**('======================================')
205. **else**:
206. **print**('No entries in inventory table to show.\n')
208. @staticmethod
209. **def** input\_cd(cd\_id, inv\_table):
210. """Gets the CD entry fields (ID, Title, Artist). Calls generate\_id() to get auto ID instead of user input.
212. Args:
213. cd\_id (int): the auto-generated ID from DataProcessor.generate\_id()
214. inv\_table (list): the current inventory table
216. Returns:
217. cd\_id (int): cleaned and validated inputted ID for a CD
218. cd\_title (str): cleaned inputted CD title
219. cd\_artist (string): cleaned inputted CD artist
220. """
221. **print**('CD ID: ', cd\_id)
222. cd\_title = input('What is the CD\'s title? ').strip()
223. cd\_artist = input('What is the Artist\'s name? ').strip()
224. **return** cd\_id, cd\_title, cd\_artist

227. # -- MAIN BODY -- #
228. # 1. When program starts, read in the currently saved Inventory
229. lstTbl = FileProcessor.read\_file(strFileName, lstTbl)
230. IO.show\_inventory(lstTbl)

233. # 2. start main loop
234. **while** True:
235. # 2.1 Display Menu to user and get choice
236. IO.print\_menu()
237. strChoice = IO.menu\_choice()

240. # 3. Process menu selection
241. # 3.1 process exit first
242. **if** strChoice == 'x':
243. **break**

246. # 3.2 process load inventory
247. **if** strChoice == 'l':
248. **print**('WARNING: If you continue, all unsaved data will be lost and the Inventory re-loaded from file.')
249. strYesNo = input('type \'yes\' to continue and reload from file, otherwise reload will be canceled.\n')
250. **if** strYesNo.lower() == 'yes':
251. **print**('reloading...')
252. lstTbl = FileProcessor.read\_file(strFileName, lstTbl)
253. IO.show\_inventory(lstTbl)
254. **else**:
255. input('canceling... Inventory data NOT reloaded. Press [ENTER] to continue to the menu.')
256. IO.show\_inventory(lstTbl)
257. **continue**  # start loop back at top.

260. # 3.3 process add a CD
261. **elif** strChoice == 'a':
262. # 3.3.1 Ask user for new ID, CD Title and Artist
263. intID = DataProcessor.generate\_id(lstTbl)
264. intID, strTitle, strArtist = IO.input\_cd(intID, lstTbl)
266. # 3.3.2 Add item to the table
267. lstTbl = DataProcessor.add\_entry(intID, strTitle, strArtist, lstTbl)
269. IO.show\_inventory(lstTbl)
270. **continue**  # start loop back at top.

273. # 3.4 process display current inventory
274. **elif** strChoice == 'i':
275. IO.show\_inventory(lstTbl)
276. **continue**  # start loop back at top.

279. # 3.5 process delete a CD
280. **elif** strChoice == 'd':
281. # 3.5.1 get Userinput for which CD to delete
282. # 3.5.1.1 display Inventory to user
283. IO.show\_inventory(lstTbl)
284. # 3.5.1.2 ask user which ID to remove
285. strID = input('Which ID would you like to delete? ').strip()
286. intIDDel = DataProcessor.get\_int\_input(strID) #tries to cast as int
288. # 3.5.2 search thru table and delete CD
289. **if** intIDDel:
290. lstTbl = DataProcessor.delete\_entry(intIDDel, lstTbl)
292. IO.show\_inventory(lstTbl)
293. **continue**  # start loop back at top.

296. # 3.6 process save inventory to file
297. **elif** strChoice == 's':
298. # 3.6.1 Display current inventory and ask user for confirmation to save
299. IO.show\_inventory(lstTbl)
300. strYesNo = input('Save this inventory to file? [y/n] ').strip().lower()
302. # 3.6.2 Process choice
303. **if** strYesNo == 'y':
304. # 3.6.2.1 save data
305. FileProcessor.write\_file(strFileName, lstTbl)
306. **else**:
307. input('The inventory was NOT saved to file. Press [ENTER] to return to the menu.')
308. **continue**  # start loop back at top.

311. # 3.7 catch-all should not be possible, as user choice gets vetted in IO, but to be save:
312. **else**:
313. **print**('General Error')

## Output from Spyder

A screenshot of a cell phone

Description automatically generated

Spyder Output 1 - Saving and exiting the script

## OUtput from Terminal

A screenshot of a cell phone

Description automatically generated

Terminal Output 1 - Deleting an entry in Terminal

1. <https://www.geeksforgeeks.org/serialization-in-java/>, accessed 2020-Mar-07 [↑](#footnote-ref-1)
2. <https://www.geeksforgeeks.org/understanding-python-pickling-example/>, accessed 2020-Mar-07  
   *great example; very understandable* [↑](#footnote-ref-2)
3. <https://stackoverflow.com/questions/8968884/python-serialization-why-pickle>, accessed 2020-Mar-07  
   *StackOverflow is a good Q&A forum and users often break down tricky concepts into a very accessible manner* [↑](#footnote-ref-3)
4. <https://www.thoughtco.com/using-pickle-to-save-objects-2813661>, accessed 2020-Mar-07 [↑](#footnote-ref-4)
5. <https://docs.python.org/3.7/library/pickle.html>, accessed 2020-Mar-07  
   *The Python source documentation library is THE original documentation, thus reliable. It also allows you to refer to documentation for different versions. Can be dry and very technical.* [↑](#footnote-ref-5)
6. <https://docs.python.org/3.7/reference/compound_stmts.html#try>, accessed 2020-Mar-08 [↑](#footnote-ref-6)
7. Doug Klos, Module 6 Feedback, 2020-Mar-03 [↑](#footnote-ref-7)
8. <https://www.geeksforgeeks.org/multiple-exception-handling-in-python/>, accessed 2020-Mar-07  
   *good examples of different ways to handle multiple exceptions* [↑](#footnote-ref-8)
9. <https://www.codementor.io/@sheena/how-to-write-python-custom-exceptions-du107ufv9>, accessed 2020-Mar-07, *great examples and summary* [↑](#footnote-ref-9)
10. A full list of built in exceptions in Python 3.7, accessed 2020-Mar-07 [↑](#footnote-ref-10)
11. <https://www.programiz.com/python-programming/user-defined-exception>, 2020-Mar-07  
    *demonstrates the standard way to define user-defined exceptions, using a base class and deriving individual classes from it* [↑](#footnote-ref-11)
12. <http://www.planetb.ca/projects/syntaxHighlighter/>, accessed 2020-Mar-08 [↑](#footnote-ref-12)
13. <https://docs.python.org/2/library/pickle.html#pickle.PicklingError>, accessed 2020-Mar-8 [↑](#footnote-ref-13)