Laurel Warner

12/6/2021

Foundations of Programming

Assignment08

Intro to Software Objects

# Introduction

This module introduced the subject of software objects. Some of this I had seen before, but I just didn’t know that it WAS part of software objects. Examples are: classes and decorators like “@staticmethod”. This module also gave more information about git and GitHub as well. This module introduced classes, objects as well as constructors, fields, attributes and methods.

# Classes

A class is like a “blueprint” for an object. The class packages the data and functionality of an object. The object itself is what’s known as an “instantiation” of the class. It’s 1 “instance” of the class with unique attributes. The object as the same functionality as the class that it is a part of. The components that make up a class are fields, constructors, attributes, properties, and methods. Fields are the data stores of a class. They get created the same way as variables, but you don’t need to call them out specifically. I know this is true because even though they are mentioned in this module, they are NOT mentioned at all in the book or in any other web sources I could find. And indeed, once a constructor and attributes are created you can just go ahead and delete your fields if you had previously created them (which is what happened in Lab C). Constructors are invoked when creating an object. They are a way to ensure proper datatypes in the fields. They also allow for population default values for the fields as well so that they all start out the same way before they get changed either by the program or by user input. They are created by calling the \_\_init\_\_() method. (Called “dunder init method”.) Attributes are internal fields or variables that hold data. (So having fields and attributes both is redundant I believe). Attributes are variables and unless you write code to validate values before they are assigned you won’t have any control over what goes into them or how they change during runtime. Properties are methods that have control mechanisms built in that control the validity of values assigned to the attributes and if you make your attributes private then you would use private properties and public properties both in tandem to enforce the interaction with the attributes in such a way that they can’t be changed in ways you don’t want. There are 2 properties for each attribute if you choose to go that route. They are the “getter” and “setter” properties. The “setter” writes or sets the attribute and the “getter” reads or “gets” it. Getter methods let you add code to format the fields or attributes data and the setter methods let you add validation or error handling to the code. Methods are like functions in a script. They let you organize your statements into blocks that can be invoked by calling the method’s name. Calling a method also submits a reference to the object it’s invoked on so the first attribute for a method is the “self” reference.

# Decorators, Docstrings, and the Self Keyword

The keyword “self” is the first parameter in every method. Every method automatically receives a reference to that object, so you have to put “self” so the class knows on which object to use the method. Self is used in all your constructors, as well as in all the attributes, properties, and methods. It is ALWAYS used except for static methods. Decorators are what gets typed before your properties and say things like “@property” or “@property.setter” as well as our old friend “@staticmethod”. They basically change the behavior or scope of functions. They wrap functionality around a method. Using @staticmethod is done when you want a method to be called on the class level and not on an instance level. This allows you to keep track of information across a whole class. This is the only time when you don’t need to use the “self” keyword. Docstrings are included in classes, properties, and methods so that you can type in the pertinent information about it. When you do that, then it allows hints and helps to populate in your IDE environment.

# Labwork

The labwork this week was not too bad. Here is a screenshot of my Lab E which has most everything else in it from Labs A-D (except for the Fields from Lab A).

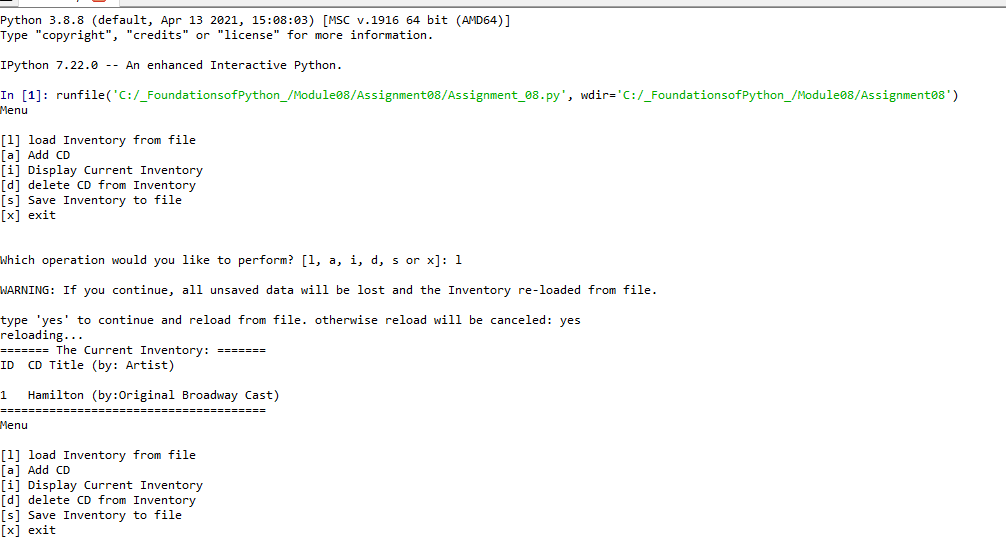


Figure 1

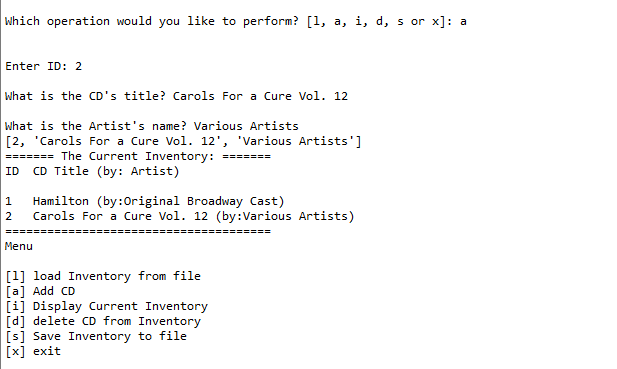
I got to see how to properly construct a class with its’ attributes, properties and methods. This led directly into the assignment for the week.

# Assignment Code

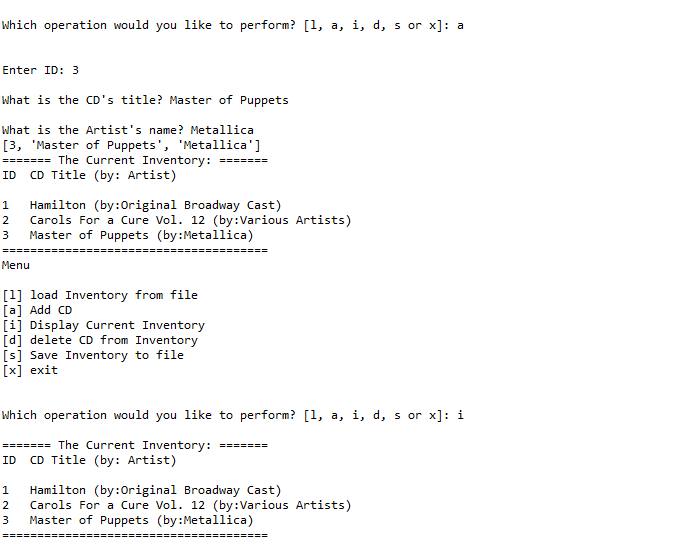
For this assignment I wound up copying and pasting most of my data from lab E (but not the method) and just changing the attributes and properties to match the CD Title, Artist and CD ID that were given. I don’t know how this bears on the rest of the code though, because it doesn’t look like it ever gets called anywhere else in the code. But maybe it’s supposed to and that is 1 area that I messed up on. I added a new class of DataProcessor because it wasn’t there in the starter code. It was in the assignment for prior assignments so I just went ahead and added it here because I knew it would most likely be needed. I updated and changed all the instances of “lstTbl” and “dicRow” to be “lstOfCDObjects” and “objRow” to match this weeks tasks of working with CD objects and not tables of lists or dictionaries. In the FileIO class I copied and pasted my code from prior weeks for the load\_inventory and save\_inventory methods that were given in the starter code. Also with some error handling. Here, also, I updated the instances of “lstTbl” and “tblRow” and “dicRow” (whichever one as the case may be) to match the fact that we are working with objects this week. For the IO class, I copied and pasted my code from Assignment 6 and 7 and it didn’t appear to need much done to it, except in the “enter\_row()” method. I just had to update the Args to be cd\_id, cd\_title, and cd\_artist instead of “strID”, “strTitle”, or “strArtist”. Then in the main body of the code I copied and pasted my code from Assignment 6 and updated all the instances that call different functions to make sure the names of the classes and methods match up. I did get it to the point where Spyder didn’t have any error messages in the margins saying that there was a syntax error. My first time running the code through on Sunday, I got errors upon Loading the Data and when I attempted to Add Data. My helpful tutor was not available to address the errors until after the deadline had passed, but now the errors had been addressed. One issue was in line 103 which says: “ if row[0]==intIDDel”, and I previously had it as “if row[“ID”]==intIDDel”, which my tutor informed me was still a remnant of dictionaries. So I updated it to have the index of the spot that contains the ID rather than the dictionary key. I found a total of 2 instances in my code that needed to have that particular fix done. That solved that problem! Another error was in various spots where I had code that referenced “row.values” instead of just “row”. That was another remnant of dictionaries as well. Finally, my code for loading the inventory threw an error because I was trying to pass 2 arguments instead of 1 into it, which it didn’t like. So I got rid of that. Here are the screenshots of my code working properly in both Spyder and the Terminal Window:



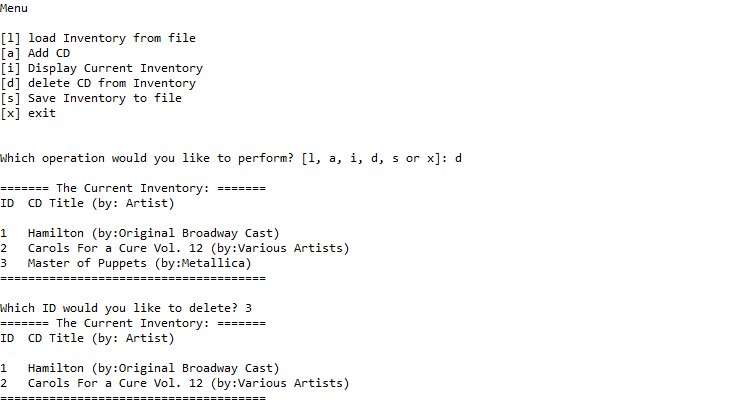
Figure



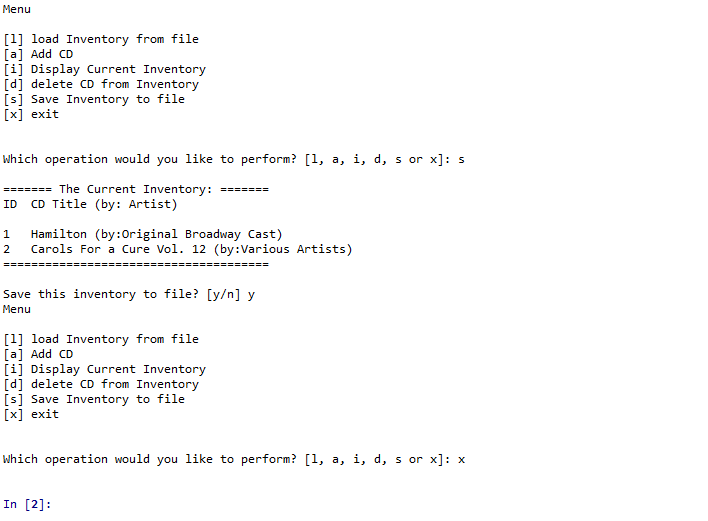
Figure



Figure

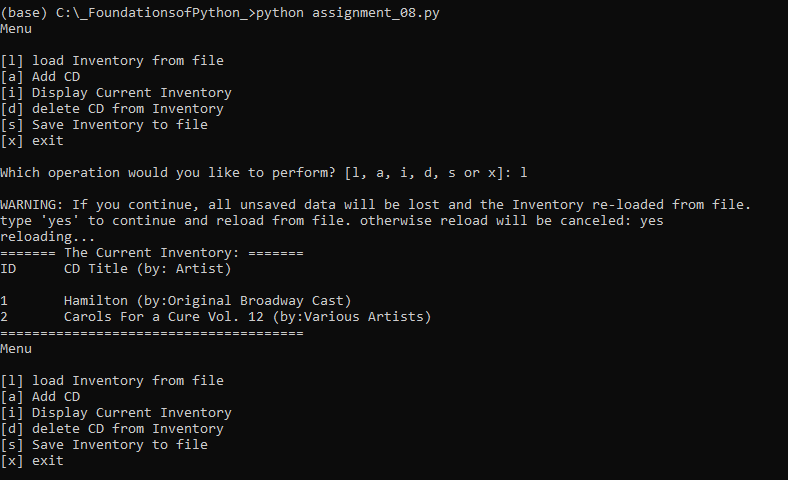


Figure

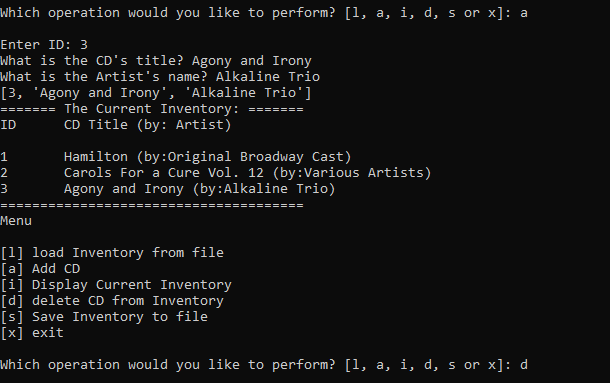


Figure

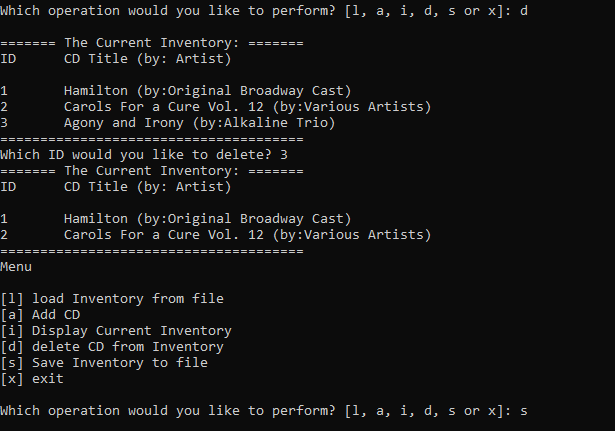
And here are the screenshots from the Terminal Window:



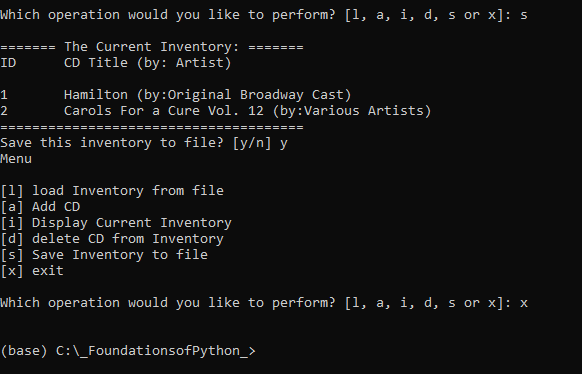
Figure



Figure



Figure



Figure

I uploaded my files to GitHub at the following address: https://github.com/lellenn/Assignment\_08.

# Summary

In this module I did learn about classes, attributes, properties, and methods. I learned how to set each of these items up with very simplistic information. I learned about what all of these things are. However, I did have errors with my code as I laid out in the prior section and I could not solve them on my own until after the deadline. They have been fixed and I am resubmitting the assignment even with the docking of points.

# Appendix

1. *#------------------------------------------#*
2. *# Title: Assignment08.py*
3. *# Desc: Assignment 08 - Working with classes*
4. *# Change Log: (Who, When, What)*
5. ***# DBiesinger, 2030-Jan-01, created file***
6. *# DBiesinger, 2030-Jan-01, added pseudocode to complete assignment 08*
7. *# LWarner, 2021-Dec-05, added code and docstrings for functionality*
8. *#------------------------------------------#*
10. **import pprint**
12. *# -- DATA -- #*
13. strFileName = 'cdInventory.txt'
14. lstOfCDObjects = []
16. **class** CD:
17. """Stores data about a CD:
19. properties:
20. **cd\_id: (int) with CD ID**
21. cd\_title: (string) with the title of the CD
22. cd\_artist: (string) with the artist of the CD
23. methods:
25. **"""**
26. *#--Constructor--#*
27. **def** \_\_init\_\_(self, cd\_id, cd\_title, cd\_artist):
28. *#--Attributes--#*
29. self.\_\_cd\_id = cd\_id
30. **self.\_\_cd\_title = cd\_title**
31. self.\_\_cd\_artist = cd\_artist
33. *#--Properties--#*
35. **@property**
36. **def** cd\_id(self):
37. **return** self.\_\_cd\_id
39. @cd\_id.setter
40. **def cd\_id(self, cd\_id):**
41. **if** str(cd\_id).isnumeric():
42. **return** cd\_id
43. **else**:
44. **raise** Exception ('ID must be a number!')
46. @property
47. **def** cd\_title(self):
48. **return** self.\_\_cd\_title
50. **@cd\_title.setter**
51. **def** cd\_title(self, cd\_title):
52. **if** str(cd\_title).isnumeric():
53. **raise** Exception('Title must be a string.')
54. **else**:
55. **return cd\_title**
57. @property
58. **def** cd\_artist(self):
59. **return** self.\_\_cd\_artist
61. @cd\_artist.setter
62. **def** cd\_artist(self, cd\_artist):
63. **if** str(cd\_artist).isnumeric():
64. **raise** Exception('Artist must be a string.')
65. **else:**
66. **return** cd\_artist
68. *# -- PROCESSING -- #*
69. **class** DataProcessor:
70. **""" Functions to add, delete, and save data in the table"""**
72. @staticmethod
73. **def** add\_row(objRow):
74. """Function to append a row to the table
75. **Gathers the user input for ID, Title and Artist and appends it to the table in memory.**
77. Args:
78. row: the row of ID, Title, and Artist
79. table: the 2D data structure that holds the data during runtime
81. Returns:
82. Shows inventory
83. """
84. pprint.pprint(objRow)
85. **objRow[0] = int(objRow[0])**
86. lstOfCDObjects.append(objRow)
88. @staticmethod
89. **def** delete\_row(intRowNr, lstTbl):
90. **""" Function to delete a row from the table if the user wants.**
92. Args:
93. intRowNr: the ID number for the row to be deleted
94. lstTbl: the table with the data
96. Returns:
97. The information showing whether or not the row was successfully deleted.
98. """
99. intRowNr = -1
100. **blnCDRemoved = False**
101. **for** row **in** lstTbl:
102. intRowNr += 1
103. **if** row[0] == intIDDel:
104. **del** lstTbl[intRowNr]
105. **blnCDRemoved = True**
106. **break**
107. **if** blnCDRemoved:
108. **print**('The CD was removed')
109. **else**:
110. **print('Could not find this CD!')**
112. **class** FileIO:
113. """Processes data to and from file:
115. **properties:**
117. methods:
118. save\_inventory(file\_name, lst\_Inventory): -> None
119. load\_inventory(file\_name): -> (a list of CD objects)
121. """
122. @staticmethod
123. **def** load\_inventory(file\_name):
124. """Function to manage data ingestion from file to a list of objects
126. Reads the data from file identified by file\_name into a 2D table
127. (list of objects) table one line in the file represents one object row in table.
129. Args:
130. **file\_name (string): name of file used to read the data from**
132. Returns:
133. lstOfCDObjects (list of objects): 2D data structure (list of objects) that holds the data during runtime
134. """
135. **lstOfCDObjects.clear() *# this clears existing data and allows to load data from file***
136. **try**:
137. objFile = open(file\_name, 'r')
138. **for** line **in** objFile:
139. data = line.strip().split(',')
140. **objRow = [int(data[0]), data[1], data[2]]**
141. lstOfCDObjects.append(objRow)
142. objFile.close()
143. **except** OSError:
144. **print**('File could not be opened!')
146. @staticmethod
147. **def** save\_inventory(file\_name, lst\_Inventory):
148. """ Function to allow data to be written to the .txt file that has previously been created
150. **Args:**
151. strFileName (string): name of file used to write the data to
152. lstOfCDObjects (list of objects): 2D data structure (list of objects) that holds the data during runtime
154. Returns:
155. **None.**
156. """
157. **try**:
158. objFile = open(strFileName, 'w')
159. **for** row **in** lstOfCDObjects:
160. **lstValues = list(row)**
161. lstValues[0] = str(lstValues[0])
162. objFile.write(','.join(lstValues) + '**\n**')
163. objFile.close()
164. **except** OSError:
165. **print('File could not be opened!')**
167. *# -- PRESENTATION (Input/Output) -- #*
168. **class** IO:
169. """Handling Input / Output"""
171. @staticmethod
172. **def** print\_menu():
173. """Displays a menu of choices to the user
175. **Args:**
176. None.
178. Returns:
179. None.
180. **"""**
182. **print**('Menu**\n\n**[l] load Inventory from file**\n**[a] Add CD**\n**[i] Display Current Inventory')
183. **print**('[d] delete CD from Inventory**\n**[s] Save Inventory to file**\n**[x] exit**\n**')
185. **@staticmethod**
186. **def** menu\_choice():
187. """Gets user input for menu selection
189. Args:
190. **None.**
192. Returns:
193. choice (string): a lower case sting of the users input out of the choices l, a, i, d, s or x
195. **"""**
196. choice = ' '
197. **while** choice **not** **in** ['l', 'a', 'i', 'd', 's', 'x']:
198. choice = input('Which operation would you like to perform? [l, a, i, d, s or x]: ').lower().strip()
199. **print**() *# Add extra space for layout*
200. **return choice**
202. @staticmethod
203. **def** show\_inventory(table):
204. """Displays current inventory table

207. Args:
208. lstOfCDObjects (list of objects): 2D data structure (list of objects) that holds the data during runtime.
210. **Returns:**
211. None.
213. """
214. **print**('======= The Current Inventory: =======')
215. **print('ID\tCD Title (by: Artist)\n')**
216. **for** row **in** lstOfCDObjects:
217. **print**('{}**\t**{} (by:{})'.format(\*row))
218. **print**('======================================')
220. **@staticmethod**
221. **def** enter\_row():
222. """ Gets user input for adding a new CD to the inventory
224. Args:
225. **cd\_id: The ID of the CD to be added**
226. cd\_title: the Title of the CD to be added
227. cd\_artist: the Artist of the CD to be added
229. Returns:
230. **The ID, Title, and Artist info for each CD rendered as a string.**
231. """
232. **while** True:
233. **try**:
234. cd\_id = int(input('Enter ID: ').strip())
235. **break**
236. **except** ValueError:
237. **print**('You have not entered a number. Please enter a number')
239. cd\_title = input('What is the CD**\'**s title? ').strip()
240. **cd\_artist = input('What is the Artist\'s name? ').strip()**
241. **return** [ cd\_id, cd\_title, cd\_artist]

244. *# -- Main Body of Script -- #*
245. ***# 1. When program starts, read in the currently saved Inventory***
246. FileIO.load\_inventory(strFileName)
248. *# 2. start main loop*
249. **while** True:
250. ***# 2.1 Display Menu to user and get choice***
251. IO.print\_menu()
252. strChoice = IO.menu\_choice()
254. *# 3. Process menu selection*
255. ***# 3.1 process exit first***
256. **if** strChoice == 'x':
257. **break**
258. *# 3.2 process load inventory*
259. **if** strChoice == 'l':
260. **print('WARNING: If you continue, all unsaved data will be lost and the Inventory re-loaded from file.')**
261. strYesNo = input('type **\'**yes**\'** to continue and reload from file. otherwise reload will be canceled: ')
262. **if** strYesNo.lower() == 'yes':
263. **print**('reloading...')
264. FileIO.load\_inventory(strFileName)
265. **IO.show\_inventory(lstOfCDObjects)**
266. **else**:
267. input('canceling... Inventory data NOT reloaded. Press [ENTER] to continue to the menu.')
268. IO.show\_inventory(lstOfCDObjects)
269. **continue** *# start loop back at top.*
270. ***# 3.3 process add a CD***
271. **elif** strChoice == 'a':
272. *# 3.3.1 Ask user for new ID, CD Title and Artist*
273. DataProcessor.add\_row(IO.enter\_row())
274. IO.show\_inventory(lstOfCDObjects)
275. **continue *# start loop back at top.***
276. *# 3.4 process display current inventory*
277. **elif** strChoice == 'i':
278. IO.show\_inventory(lstOfCDObjects)
279. **continue** *# start loop back at top.*
280. ***# 3.5 process delete a CD***
281. **elif** strChoice == 'd':
282. *# 3.5.1 get Userinput for which CD to delete*
283. *# 3.5.1.1 display Inventory to user*
284. IO.show\_inventory(lstOfCDObjects)
285. ***# 3.5.1.2 ask user which ID to remove***
286. intIDDel = int(input('Which ID would you like to delete? ').strip())
287. *# 3.5.2 search thru table and delete CD*
288. DataProcessor.delete\_row(intIDDel, lstOfCDObjects)
289. IO.show\_inventory(lstOfCDObjects)
290. **continue *# start loop back at top.***
291. *# 3.6 process save inventory to file*
292. **elif** strChoice == 's':
293. *# 3.6.1 Display current inventory and ask user for confirmation to save*
294. IO.show\_inventory(lstOfCDObjects)
295. **strYesNo = input('Save this inventory to file? [y/n] ').strip().lower()**
296. *# 3.6.2 Process choice*
297. **if** strYesNo == 'y':
298. *# 3.6.2.1 save data*
299. FileIO.save\_inventory(strFileName, lstOfCDObjects)
300. **else:**
301. input('The inventory was NOT saved to file. Press [ENTER] to return to the menu.')
302. **continue** *# start loop back at top.*
303. *# 3.7 catch-all should not be possible, as user choice gets vetted in IO, but to be save:*
304. **else**:
305. **print('General Error')**