Nick Culbert

2021 Feb 21

IT FDN 110 A Wi 21

Assignment 06

CD Inventory Version 3

# Introduction

For this assignment, I modified an existing Python script to organize the code, adding functions which replaced code in the main body and more accurately adhered to the principle of the separation of concerns. I also added documentation to each new function.

# Processing

## Class DataProcessor

For any additional function that changed the value of the global variable lstTbl, rather than using lstTbl within the function, I instead added another argument, which I labeled “table” to follow Professor Biesinger’s convention. This way, I would not have to use the global keyword to ensure that lstTbl retained its changes, and it also allows future developers to more easily add multiple global 2D data structures.

I added the following functions to the DataProcessor class: add\_tbl\_item(strId, strTitle, strArtist, table) and CD\_delete(intIDDel, table).

The content of either function was kept the same apart from replacing instances of lstTbl with the function argument “table” (Figure 1).

1. **def** add\_tbl\_item(strID, strTitle, strArtist, table):
2. """Adds items to the table lstTbl
4. Args:
5. strID (string): user input for the ID of the CD being added
6. strTitle (string): user input for the title of the CD being added
7. strArtist (string): user input for the Artist's name on the CD being added
8. table (list of dicts): 2D data structure (list of dicts) that holds the data during runtime
10. Returns:
11. None.
12. """
13. intID = int(strID)
14. dictRow = {'ID': intID, 'Title': strTitle, 'Artist': strArtist}
15. table.append(dictRow)
16. IO.show\_inventory(table)
18. @staticmethod
19. **def** CD\_delete(intIDDel, table):
20. """Deletes a CD based on previous user input
22. Args:
23. intIDDel (integer): user input, converted into an integer, which designates which CD to delete
24. table (list of dicts): 2D data structure (list of dicts) that holds the data during runtime
26. Returns:
27. None.
28. """
29. intRowNr = -1
30. blnCDRemoved = False
31. **for** row **in** table:
32. intRowNr += 1
33. **if** row['ID'] == intIDDel:
34. **del** table[intRowNr]
35. blnCDRemoved = True
36. **break**
37. **if** blnCDRemoved:
38. **print**('The CD was removed')
39. **else**:
40. **print**('Could not find this CD!')
41. IO.show\_inventory(table)

Figure - Adding functions to the DataProcessor class

## Class FileProcessor

In the FileProcessor class, I changed the read\_file(file\_name, table) function to replace the second parameter in the open() function with the argument “a+”. This way, the script can create the file CDInventory.txt if it does not yet exist and can also read the file. I also added the line objFile.seek(0) as a safety forcing mechanism so that the subsequent for loop always begins at the file’s beginning (Figure 2).

1. objFile = open(file\_name, 'a+')
2. objFile.seek(0)
3. **for** line **in** objFile:
4. data = line.strip().split(',')
5. dictRow = {'ID': int(data[0]), 'Title': data[1], 'Artist': data[2]}
6. table.append(dictRow)
7. objFile.close()

Figure - Adding functionality to Professor Biesinger's code

I also added code to the write\_file(file\_name, table) function, copy-pasting from what Professor Biesinger had left in the script’s main body (Figure 3).

1. **def** write\_file(file\_name, table):
2. """Writes current inventory data to text file
4. Args:
5. file\_name (string): name of file used to read the data from
6. table (list of dicts): 2D data structure (list of dicts) that holds the data during runtime
8. Returns:
9. None.
10. """
11. objFile = open(strFileName, 'w')
12. **for** row **in** table:
13. lstValues = list(row.values())
14. lstValues[0] = str(lstValues[0])
15. objFile.write(','.join(lstValues) + '\n')
16. objFile.close()

Figure - Creating the write\_file(file\_name, table) function

# Presentation

In the IO class, I added the function get\_new\_CD(). It assigns multiple variables to various user inputs and then returns those variables (Figure 4).

1. **def** get\_new\_CD():
2. """Gets user input for adding a new CD
4. Args:
5. None.
7. Returns:
8. strID (string): user input for the ID of the CD being added
9. strTitle (string): user input for the title of the CD being added
10. strArtist (string): user input for the Artist's name on the CD being added
11. """
12. strID = input('Enter ID: ').strip()
13. strTitle = input('What is the CD\'s title? ').strip()
14. strArtist = input('What is the Artist\'s name? ').strip()
15. **return** strID, strTitle, strArtist

Figure - Creating the get\_new\_CD() function

# Main Body

Adding more functions to the script allowed me to reduce the amount of code in the main body. To me, this new organization is much more skimmable, which I expect would be important when working on teams with many developers who have to constantly switch between each other’s code.

In section 3.3, the process to add a CD, I included a line that simultaneously assigned values to the variables strID, strTitle, and strArtist based on the returns from the get\_new\_CD() function before using those variables as arguments in the subsequent function (Figure 5).

1. # 3.3 process add a CD
2. **elif** strChoice == 'a':
3. strID, strTitle, strArtist = IO.get\_new\_CD()
4. DataProcessor.add\_tbl\_item(strID, strTitle, strArtist, lstTbl)
5. **continue** #start loop back at top

Figure - Simultaneous assignment

# Running the script

Figures 6 and 7 are screenshots of my script running in Spyder and Terminal, respectively. The Terminal window screenshot shows the process of deleting an entry.

Text

Description automatically generated

Figure - Running the script in Spyder

Text

Description automatically generated

Figure - Running the script in Terminal

# Summary

I successfully modified Professor Biesinger’s script to add more functions and clean up the main body, making sure to modify the code to account for global vs. local variables.

# Appendix

1. #------------------------------------------#
2. # Title: CDInventory.py
3. # Desc: Working with classes and functions.
4. # Change Log: (Who, When, What)
5. # DBiesinger, 2030-Jan-01, Created File
6. # nickculbert, 2021-Feb-21, added functions
7. #------------------------------------------#
9. # -- DATA -- #
10. strChoice = '' # User input
11. lstTbl = []  # list of lists to hold data
12. dictRow = {}  # list of data row
13. strFileName = 'CDInventory.txt'  # data storage file
14. objFile = None  # file object

17. # -- PROCESSING -- #
18. **class** DataProcessor:
20. @staticmethod
21. **def** add\_tbl\_item(strID, strTitle, strArtist, table):
22. """Adds items to the table lstTbl
24. Args:
25. strID (string): user input for the ID of the CD being added
26. strTitle (string): user input for the title of the CD being added
27. strArtist (string): user input for the Artist's name on the CD being added
28. table (list of dicts): 2D data structure (list of dicts) that holds the data during runtime
30. Returns:
31. None.
32. """
33. intID = int(strID)
34. dictRow = {'ID': intID, 'Title': strTitle, 'Artist': strArtist}
35. table.append(dictRow)
36. IO.show\_inventory(table)
38. @staticmethod
39. **def** CD\_delete(intIDDel, table):
40. """Deletes a CD based on previous user input
42. Args:
43. intIDDel (integer): user input, converted into an integer, which designates which CD to delete
44. table (list of dicts): 2D data structure (list of dicts) that holds the data during runtime
46. Returns:
47. None.
48. """
49. intRowNr = -1
50. blnCDRemoved = False
51. **for** row **in** table:
52. intRowNr += 1
53. **if** row['ID'] == intIDDel:
54. **del** table[intRowNr]
55. blnCDRemoved = True
56. **break**
57. **if** blnCDRemoved:
58. **print**('The CD was removed')
59. **else**:
60. **print**('Could not find this CD!')
61. IO.show\_inventory(table)

64. **class** FileProcessor:
65. """Processing the data to and from text file"""
67. @staticmethod
68. **def** read\_file(file\_name, table):
69. """Function to manage data ingestion from file to a list of dictionaries
71. Reads the data from file identified by file\_name into a 2D table
72. (list of dicts) one line in the file represents one dictionary row in table.
74. Args:
75. file\_name (string): name of file used to read the data from
76. table (list of dicts): 2D data structure (list of dicts) that holds the data during runtime
78. Returns:
79. None.
80. """
81. table.clear()  # this clears existing data and allows to load data from file
82. objFile = open(file\_name, 'a+')
83. objFile.seek(0)
84. **for** line **in** objFile:
85. data = line.strip().split(',')
86. dictRow = {'ID': int(data[0]), 'Title': data[1], 'Artist': data[2]}
87. table.append(dictRow)
88. objFile.close()
90. @staticmethod
91. **def** write\_file(file\_name, table):
92. """Writes current inventory data to text file
94. Args:
95. file\_name (string): name of file used to read the data from
96. table (list of dicts): 2D data structure (list of dicts) that holds the data during runtime
98. Returns:
99. None.
100. """
101. objFile = open(strFileName, 'w')
102. **for** row **in** table:
103. lstValues = list(row.values())
104. lstValues[0] = str(lstValues[0])
105. objFile.write(','.join(lstValues) + '\n')
106. objFile.close()

109. # -- PRESENTATION (Input/Output) -- #
111. **class** IO:
112. """Handling Input / Output"""
114. @staticmethod
115. **def** print\_menu():
116. """Displays a menu of choices to the user
118. Args:
119. None.
121. Returns:
122. None.
123. """
124. **print**('Menu\n\n[l] load Inventory from file\n[a] Add CD\n[i] Display Current Inventory')
125. **print**('[d] delete CD from Inventory\n[s] Save Inventory to file\n[x] exit\n')
127. @staticmethod
128. **def** menu\_choice():
129. """Gets user input for menu selection
131. Args:
132. None.
134. Returns:
135. choice (string): a lower case sting of the users input out of the choices l, a, i, d, s or x
137. """
138. choice = ' '
139. **while** choice **not** **in** ['l', 'a', 'i', 'd', 's', 'x']:
140. choice = input('Which operation would you like to perform? [l, a, i, d, s or x]: ').lower().strip()
141. **print**()  # Add extra space for layout
142. **return** choice
144. @staticmethod
145. **def** show\_inventory(table):
146. """Displays current inventory table
148. Args:
149. table (list of dict): 2D data structure (list of dicts) that holds the data during runtime.
151. Returns:
152. None.
154. """
155. **print**()
156. **print**('======= The Current Inventory: =======')
157. **print**('ID\tCD Title (by: Artist)\n')
158. **for** row **in** table:
159. **print**('{}\t{} (by:{})'.format(\*row.values()))
160. **print**('======================================')
161. **print**()
163. @staticmethod
164. **def** get\_new\_CD():
165. """Gets user input for adding a new CD
167. Args:
168. None.
170. Returns:
171. strID (string): user input for the ID of the CD being added
172. strTitle (string): user input for the title of the CD being added
173. strArtist (string): user input for the Artist's name on the CD being added
174. """
175. strID = input('Enter ID: ').strip()
176. strTitle = input('What is the CD\'s title? ').strip()
177. strArtist = input('What is the Artist\'s name? ').strip()
178. **return** strID, strTitle, strArtist
180. # 1. When program starts, read in the currently saved Inventory
181. FileProcessor.read\_file(strFileName, lstTbl)
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()
189. # 3. Process menu selection
191. # 3.1 process exit first
192. **if** strChoice == 'x':
193. **break**
195. # 3.2 process load inventory
196. **if** strChoice == 'l':
197. **print**('WARNING: If you continue, all unsaved data will be lost and the Inventory re-loaded from file.')
198. strYesNo = input('Type \'yes\' to continue and reload from file. Type anything else to cancel the reload. ')
199. **if** strYesNo.lower() == 'yes':
200. **print**('\nreloading...\n')
201. FileProcessor.read\_file(strFileName, lstTbl)
202. IO.show\_inventory(lstTbl)
203. **else**:
204. input('canceling... Inventory data NOT reloaded. Press [ENTER] to continue to the menu.')
205. IO.show\_inventory(lstTbl)
206. **continue**  # start loop back at top.
208. # 3.3 process add a CD
209. **elif** strChoice == 'a':
210. strID, strTitle, strArtist = IO.get\_new\_CD()
211. DataProcessor.add\_tbl\_item(strID, strTitle, strArtist, lstTbl)
212. **continue** #start loop back at top
214. # 3.4 process display current inventory
215. **elif** strChoice == 'i':
216. IO.show\_inventory(lstTbl)
217. **continue**  # start loop back at top.
219. # 3.5 process delete a CD
220. **elif** strChoice == 'd':
221. # 3.5.1 get user input for which CD to delete
222. # 3.5.1.1 display Inventory to user
223. IO.show\_inventory(lstTbl)
224. # 3.5.1.2 ask user which ID to remove
225. intIDDel = int(input('Which ID would you like to delete? ').strip())
226. DataProcessor.CD\_delete(intIDDel, lstTbl)
227. **continue**  # start loop back at top.
229. # 3.6 process save inventory to file
230. **elif** strChoice == 's':
231. # 3.6.1 Display current inventory and ask user for confirmation to save
232. IO.show\_inventory(lstTbl)
233. strYesNo = input('Save this inventory to file? [y/n] ').strip().lower()
234. # 3.6.2 Process choice
235. **if** strYesNo == 'y':
236. FileProcessor.write\_file(strFileName, lstTbl)
237. **else**:
238. input('The inventory was NOT saved to file. Press [ENTER] to return to the menu.')
239. **continue**  # start loop back at top.
241. # 3.7 catch-all should not be possible, as user choice gets vetted in IO, but to be safe:
242. **else**:
243. **print**('General Error')