Maricha Friedman

August 14, 2021

IT FDN 110: Introduction to Programming (Python)

Assignment 6

CD Inventory Script with Functions

# Introduction

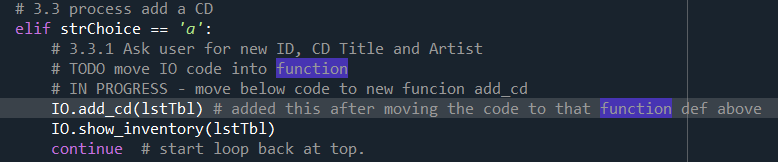
For this assignment, we had to adapt an existing script with a few to-dos. The end result is the same as the previous assignment, to create a CD inventory, but uses functions grouped in classes.

# Understanding the existing script

At first, understanding what I was supposed to do with the script was a challenge. I read through the starter script, and I know it was supposed to have the same functionality of the previous assignment, just with functions. Of course, the annotations built into the starter script helped break down what each section was meant to do. The blue exclamation TODO flags along the line numbers helped me keep track of where there were to dos. The “#TODO put IO code into Function” made it clear that I was meant to move the code below into the associated function above that had not been completed (in #3.3 process to add a CD)

I got lost in my place scrolling up and down so I added a placeholder comment: “# IN PROGRESS - move below code to new funcion add\_cd”. I did this throughout the script as I worked, as well as adding notes of things to look out for and check for when I run and test the script. I left all TODOs and notes of my progress, knowing I’d have to read through and remove them once my script functioned correctly.

Copy/pasting existing code into new functions meant the indentations weren’t always right, but the line-by-line error notifications in Spyder really helped me fix this without running the script. I had never used this feature before this assignment.



Listing 1 - Example of comment notes to TODOs.

# Functions: Matching arguments and docstrings

I created the “add\_cd” and “remove\_cd” functions for this assignment. When moving code to the functions, I needed to replace “global” terms like “lstTbl” with “table” as the argument (or parameter) in the function definition. I’m not sure “global” is the right term to use, since these aren’t explicitly cast as global variables, but they exist in the main code outside of any functions. The variable “intIDDel” is an example. This is the input value the user needs to enter to identify which CD ID in the inventory list should be deleted. I almost left it in the function, which I think would have been bad practice, since the initial input command and definition of that variable name is outside of the function. It still worked that way, but for consistency’s sake, I decided to use a local variable name that only exists in the function (I called it “IDno”) and included it as an argument.

I almost forgot to add docstrings for the “add\_cd” and “remove\_cd” functions that I created, but I did add them, even though they were pretty straightforward.

# Loading and saving data to a file

I had completed all the TODOs except the first one in the “DataProcessors” class. I wasn’t sure which function needed to go there. When I ran the script, I got an error regarding the “CDInventory.txt”:

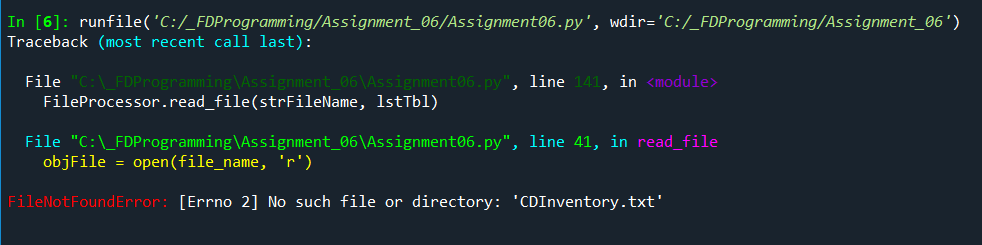


Figure 1 - Error, no such file

The file did not exist, because I hadn’t run the program yet nor saved any data, so there was no “CDInventory.txt” file to read from for the very first function that was executed: FileProcessor**.**read\_file**(**strFileName**,** lstTbl**)**. I thought maybe changing the open mode in the “read\_file” function from ‘r’ to ‘a’ would be an easy fix, because ‘a’ mode creates the file if it doesn’t exist, but I wasn’t sure if that would be an appropriate thing to do. So I created a new, very simple function called “create\_file” and put it in the DataProcessors class:

**def** create\_file**(**file\_name**):**

'''Opens the .txt file in append mode, to create if it doesn't exist

Args:

file\_name (string): name of file used to read the data from and write to it

Returns:

None.

'''

objFile **=** **open(**file\_name**,** 'a'**)**

objFile**.**close**()**

The only purpose of this was to create the file if it didn’t already exist, or just open it in append mode if it did exist, and then close it. Once I added this, I did get the error anymore, but I did need to call this function as the first line of code.

# Summary

This assignment was straightforward except that I wasn’t sure what needed to be included in the “Data Processor” class grouping. I think I included all the functions necessary, as the script works as expected. It was the longest script I had worked with before and it makes sense why functions are so useful with long, complicated scripts.

# Appendix

## Listing CDInventory.py

1. *#------------------------------------------#*
2. *# Title: CDInventory.py*
3. *# Desc: Working with classes and functions - creating CD inventory*
4. *# Change Log: (Who, When, What)*
5. ***# DBiesinger, 2030-Jan-01, Created File***
6. *# M Friedman, 8/14, updated to complete assignment - add functions and organize code*
7. *#------------------------------------------#*
9. *# -- DATA -- #*
10. **strChoice = '' *# User input***
11. lstTbl = [] *# list of lists to hold data*
12. dicRow = {} *# 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** create\_file(file\_name):
22. '''Opens the .txt file in append mode, to create if it doesn't exist
24. Args:
25. **file\_name (string): name of file used to read the data from and write to it**
27. Returns:
28. None.
29. '''
30. **objFile = open(file\_name, 'a')**
31. objFile.close()
33. **pass**

36. **class** FileProcessor:
37. """Processing the data to and from text file"""
39. @staticmethod
40. **def read\_file(file\_name, table):**
41. """Function to manage data ingestion from file to a list of dictionaries
43. Reads the data from file identified by file\_name into a 2D table
44. (list of dicts) table one line in the file represents one dictionary row in table.
46. Args:
47. file\_name (string): name of file used to read the data from
48. table (list of dict): 2D data structure (list of dicts) that holds the data during runtime
50. **Returns:**
51. None.
52. """
53. table.clear() *# this clears existing data and allows to load data from file*
54. objFile = open(file\_name, 'r')
55. **for line in objFile:**
56. data = line.strip().split(',')
57. dicRow = {'ID': int(data[0]), 'Title': data[1], 'Artist': data[2]}
58. table.append(dicRow)
59. objFile.close()
61. @staticmethod
62. **def** write\_file(file\_name, table):
63. '''Writes the in-memory inventory to a .txt file
65. **Args:**
66. file\_name (string): name of file used to write the data to
67. table (list of dict): 2D data structure (list of dicts) that holds the data during runtime
69. Returns:
70. **None.**
71. '''
72. objFile = open(file\_name, 'w')
73. **for** row **in** table:
74. lstValues = list(row.values())
75. **lstValues[0] = str(lstValues[0])**
76. objFile.write(','.join(lstValues) + '**\n**')
77. objFile.close()
78. **pass**

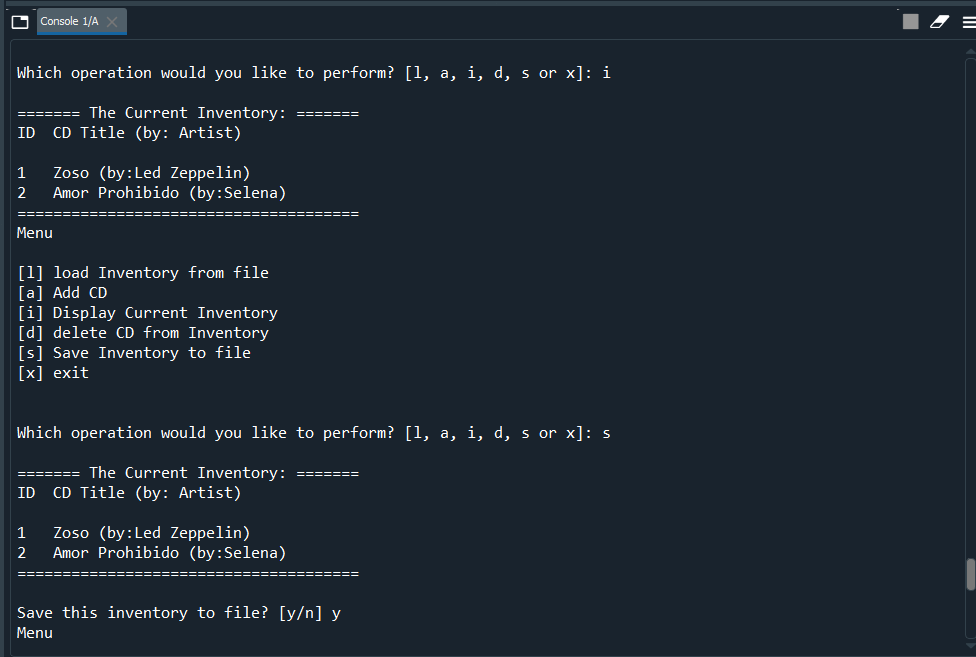
81. *# -- PRESENTATION (Input/Output) -- #*
83. **class** IO:
84. """Handling Input / Output"""
86. @staticmethod
87. **def** print\_menu():
88. """Displays a menu of choices to the user
90. **Args:**
91. None.
93. Returns:
94. None.
95. **"""**
97. **print**('Menu**\n\n**[l] load Inventory from file**\n**[a] Add CD**\n**[i] Display Current Inventory')
98. **print**('[d] delete CD from Inventory**\n**[s] Save Inventory to file**\n**[x] exit**\n**')
100. **@staticmethod**
101. **def** menu\_choice():
102. """Gets user input for menu selection
104. Args:
105. **None.**
107. Returns:
108. choice (string): a lower case sting of the users input out of the choices l, a, i, d, s or x
110. **"""**
111. choice = ' '
112. **while** choice **not** **in** ['l', 'a', 'i', 'd', 's', 'x']:
113. choice = input('Which operation would you like to perform? [l, a, i, d, s or x]: ').lower().strip()
114. **print**() *# Add extra space for layout*
115. **return choice**
117. @staticmethod
118. **def** show\_inventory(table):
119. """Displays current inventory table

122. Args:
123. table (list of dict): 2D data structure (list of dicts) that holds the data during runtime.
125. **Returns:**
126. None.
128. """
129. **print**('======= The Current Inventory: =======')
130. **print('ID\tCD Title (by: Artist)\n')**
131. **for** row **in** table:
132. **print**('{}**\t**{} (by:{})'.format(\*row.values()))
133. **print**('======================================')

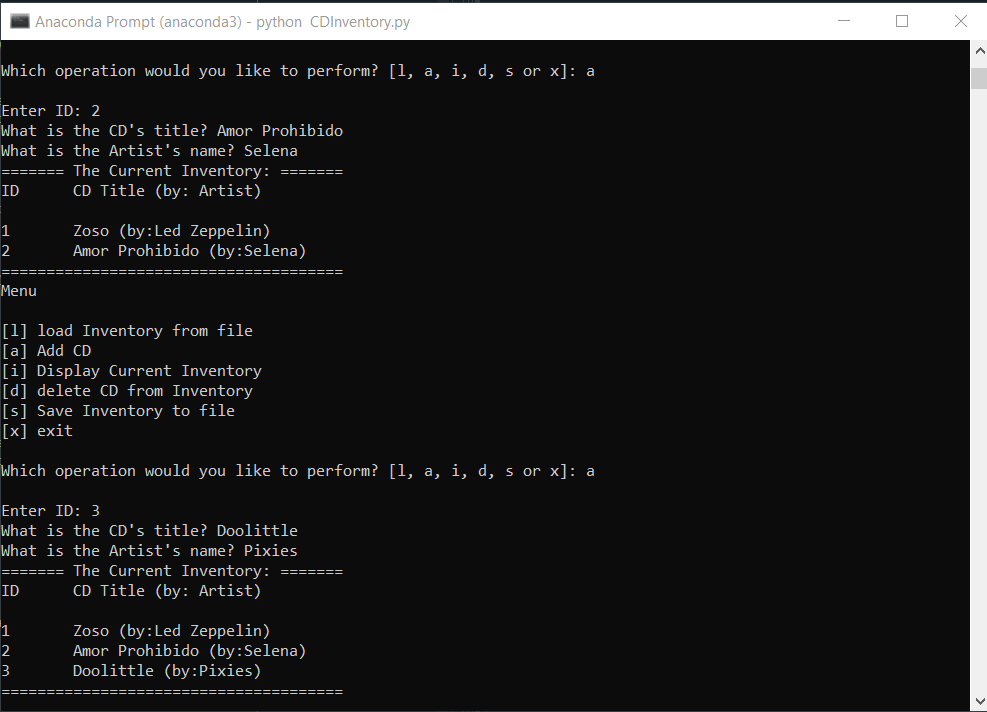
136. @staticmethod
137. **def** add\_cd(table):
138. '''Adds individual CD data to 2D list in memory
140. **Args:**
141. table: table (list of dict): 2D data structure (list of dicts) that holds the data during runtime.
143. Retuns:
144. None.
145. **'''**
146. strID = input('Enter ID: ').strip()
147. strTitle = input('What is the CD**\'**s title? ').strip()
148. stArtist = input('What is the Artist**\'**s name? ').strip()
149. *# 3.3.2 Add item to the table*
150. **intID = int(strID)**
151. dicRow = {'ID': intID, 'Title': strTitle, 'Artist': stArtist}
152. table.append(dicRow)
154. @staticmethod
155. **def remove\_cd(table, IDno):**
156. '''Removes individual CD data from 2D list in memory
158. Args:
159. table: table (list of dict): 2D data structure (list of dicts) that holds the data during runtime.
161. Retuns:
162. None.
163. '''
164. intRowNr = -1
165. **blnCDRemoved = False**
166. **for** row **in** table:
167. intRowNr += 1
168. **if** row['ID'] == IDno:
169. **del** lstTbl[intRowNr]
170. **blnCDRemoved = True**
171. **break**
172. **if** blnCDRemoved:
173. **print**('The CD was removed**\n**')
174. **else**:
175. **print('Could not find this CD!\n')**


179. *# 1. When program starts, read in the currently saved Inventory*
180. **DataProcessor.create\_file(strFileName)**
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()
188. *# 3. Process menu selection*
189. *# 3.1 process exit first*
190. **if strChoice == 'x':**
191. **break**
192. *# 3.2 process load inventory*
193. **if** strChoice == 'l':
194. **print**('WARNING: If you continue, all unsaved data will be lost and the Inventory re-loaded from file.')
195. **strYesNo = input('type \'yes\' to continue and reload from file. Otherwise, reload will be canceled: ')**
196. **if** strYesNo.lower() == 'yes':
197. **print**('reloading...')
198. FileProcessor.read\_file(strFileName, lstTbl)
199. IO.show\_inventory(lstTbl)
200. **else:**
201. input('canceling... Inventory data NOT reloaded. Press [ENTER] to continue to the menu.')
202. IO.show\_inventory(lstTbl)
203. **continue** *# start loop back at top.*
204. *# 3.3 process add a CD*
205. **elif strChoice == 'a':**
206. *# 3.3.1 Ask user for new ID, CD Title and Artist*
207. IO.add\_cd(lstTbl) *# added this after moving the code to that function def above*
208. IO.show\_inventory(lstTbl)
209. **continue** *# start loop back at top.*
210. ***# 3.4 process display current inventory***
211. **elif** strChoice == 'i':
212. IO.show\_inventory(lstTbl)
213. **continue** *# start loop back at top.*
214. *# 3.5 process delete a CD*
215. **elif strChoice == 'd':**
216. *# 3.5.1 get Userinput for which CD to delete*
217. *# 3.5.1.1 display Inventory to user*
218. IO.show\_inventory(lstTbl)
219. *# 3.5.1.2 ask user which ID to remove*
220. **intIDDel = int(input('Which ID would you like to delete? ').strip())**
221. *# 3.5.2 search thru table and delete CD*
222. IO.remove\_cd(lstTbl, intIDDel)
223. IO.show\_inventory(lstTbl)
224. **continue** *# start loop back at top.*
225. ***# 3.6 process save inventory to file***
226. **elif** strChoice == 's':
227. *# 3.6.1 Display current inventory and ask user for confirmation to save*
228. IO.show\_inventory(lstTbl)
229. strYesNo = input('Save this inventory to file? [y/n] ').strip().lower()
230. ***# 3.6.2 Process choice***
231. **if** strYesNo == 'y':
232. *# 3.6.2.1 save data*
233. FileProcessor.write\_file(strFileName, lstTbl)
234. **else**:
235. **input('The inventory was NOT saved to file. Press [ENTER] to return to the menu.')**
236. **continue** *# start loop back at top.*
237. *# 3.7 catch-all should not be possible, as user choice gets vetted in IO, but to be save:*
238. **else**:
239. **print**('General Error')

# Script running in Spyder



# Script running in Terminal



# GitHub Repository

Here is the link for my repository for Assignment 6:

<https://github.com/marichaf/Assignment_06>