Joseph Prado

10 March 2020

## IT FDN 100 B

Assignment 06

**Modifying a program to increase the use of functions.**

# Introduction

In this assignment, I will do several things.

1. Modify a script to use functions as directed by the TODO lines provided by the instructor in the script.
2. Add docstrings to additional functions that I create in the program.
3. Verify the script functions in Spyder and in the Terminal.
4. Upload my script and knowledge document into GitHub.

# Reviewing and modifying the script

**To accomplish this assignment, I will start by going to the main (while) loop in the second half of the sample code, and determine locations where the TODO says I should move the code into a function.**

**We first run into a TODO at Section 3.3.1 in the sample code. This code requests user input for a new CD to be added to the inventory.**

**Since per the annotation, this code “asks user for new ID, CD Title and Aritst”, I’ll move these lines of code to a function I’ll create under class “IO”. Classes give us a way to group functions with common charateristics together.**[[1]](#footnote-1) **Under IO, we’re grouping functions that are responsible for the collection and return of data together. Before the function, I place the word @staticmethod above it for reasons we are not required to understand right now.**[[2]](#footnote-2)

1. @staticmethod
2. **def** input\_new\_cd():

**I make no changes to how the code request input from the user, except that I change the name of the variables to make it clear that they are local and limited to the scope of the function. By doing this, I create local variables that cannot be accessed by any other function or code outside of this scope.**[[3]](#footnote-3)

1. cdID = input('Enter ID: ').strip()
2. title = input('What is the CD\'s title? ').strip()
3. artist = input('What is the Artist\'s name? ').strip()

**I then need to return this code to the main while loop. I can only return multiple return values as a collection**[[4]](#footnote-4)**, so I group them together separated by commas, creating a tuple.**

1. **return** cdID, title, artist

**Then once I am back in the while loop at 3.3.1, I need to call this function and unpack this tuple into the three variables I need to work with, as demonstrated by the instructor.**[[5]](#footnote-5) **I do this with the following line of code**

        strID, strTitle, strArtist = input\_new\_cd()

**OK, I’m getting an error in the code: “Undefined name”. Aha, I need to mention the class that the function is grouped under. Since this is part of the class IO, I need to invoke it using dot notation.** [[6]](#footnote-6)

        strID, strTitle, strArtist = IO.input\_new\_cd()

**Now, let’s work on the moving the next bit of code in 3.3.2 into a function.**

**This is processing code, so I will move it under the Class DataProcessor. I’ll create the function add\_cd**

1. @staticmethod
2. **def** add\_cd():

**I initially move the code with no modifications under the function I created. A second, thought…one of the lines of code calls the function IO.show\_inventory(lstTbl). Probably not good to have a function call another function. I’m going to pull that out and put it back in the main while Loop.**

**Now, I need to modify the function so that it will accept the return values from the previous function that I worked that I unpacked into variables. To do that, let’s create positional parameters**[[7]](#footnote-7) **for this function.**

1. **﻿**     **def** add\_cd(cdID, title, artist, row, table):

**I’ll also need to make sure the I change the name of the variables in the function to ones that match the parameters:**

1. intID = int(cdID)
2. row = {'ID': intID, 'Title': title, 'Artist': artist}
3. table.append(row)

**Now, we need to go back to the while loop at 3.3.2 and reference this function. ﻿**

        DataProcessor.add\_cd(strID, strTitle, strArtist, dicRow, lstTbl)

**In this line of code, the arguments come from the return values of the previous function that I unpacked, and the list and dictionary (lstTbl and dicRow) defined at the beginning at the beginning of the program.**

**The next TODO is at 3.5.2, where we will move the processing code to delete a CD from the table to the top of the program as a function under the class DataProcessor.**

**I will name the function the delete\_cd**

1. @staticmethod
2. **def** delete\_cd()

**I will need to provide the function the ID for the CD that the user would like to delete, so we’ll give the function parameters.**

1. **def** delete\_cd(selectID, table):

**The rest of the code I will modify to match the parameters I selected to use.**

1. intRowNr = -1
2. blnCDRemoved = False
3. **for** row **in** table:
4. intRowNr += 1
5. **if** row['ID'] == selectID:
6. **del** table[intRowNr]
7. blnCDRemoved = True
8. **break**
9. **if** blnCDRemoved:
10. **print**('The CD was removed')
11. **else**:
12. **print**('Could not find this CD!')

**Now, I need to go back to while loop at 3.5.2 and call the function, using the CD I that the user would like to delete as one of the arguments.**

        DataProcessor.delete\_cd(intIDDel, lstTbl)

**The last bit of code with a TODO is at 3.6.2.1. Here we will move the processing code to save inventory to file as a function under the class FileProcessor. Here the function is predefined as write\_file with parameters file\_name and table.**

1. @staticmethod
2. **def** write\_file(file\_name, table):

**Matching the variables to the parameter names gives us the following:**

1. objFile = open(file\_name, 'w')
2. **for** row **in** table:
3. lstValues = list(row.values())
4. lstValues[0] = str(lstValues[0])
5. objFile.write(','.join(lstValues) + '\n')
6. objFile.close()

# Adding docstrings

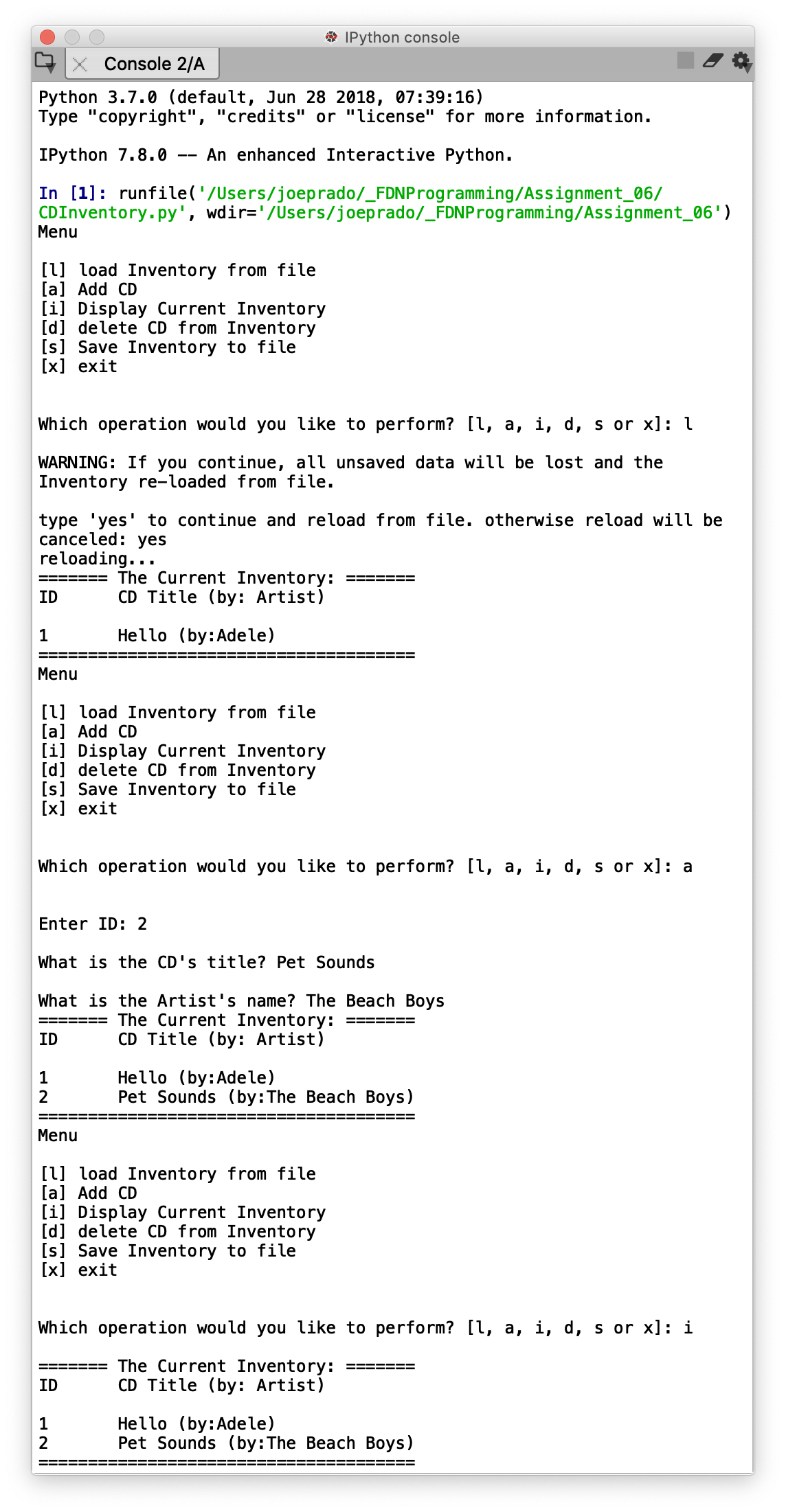
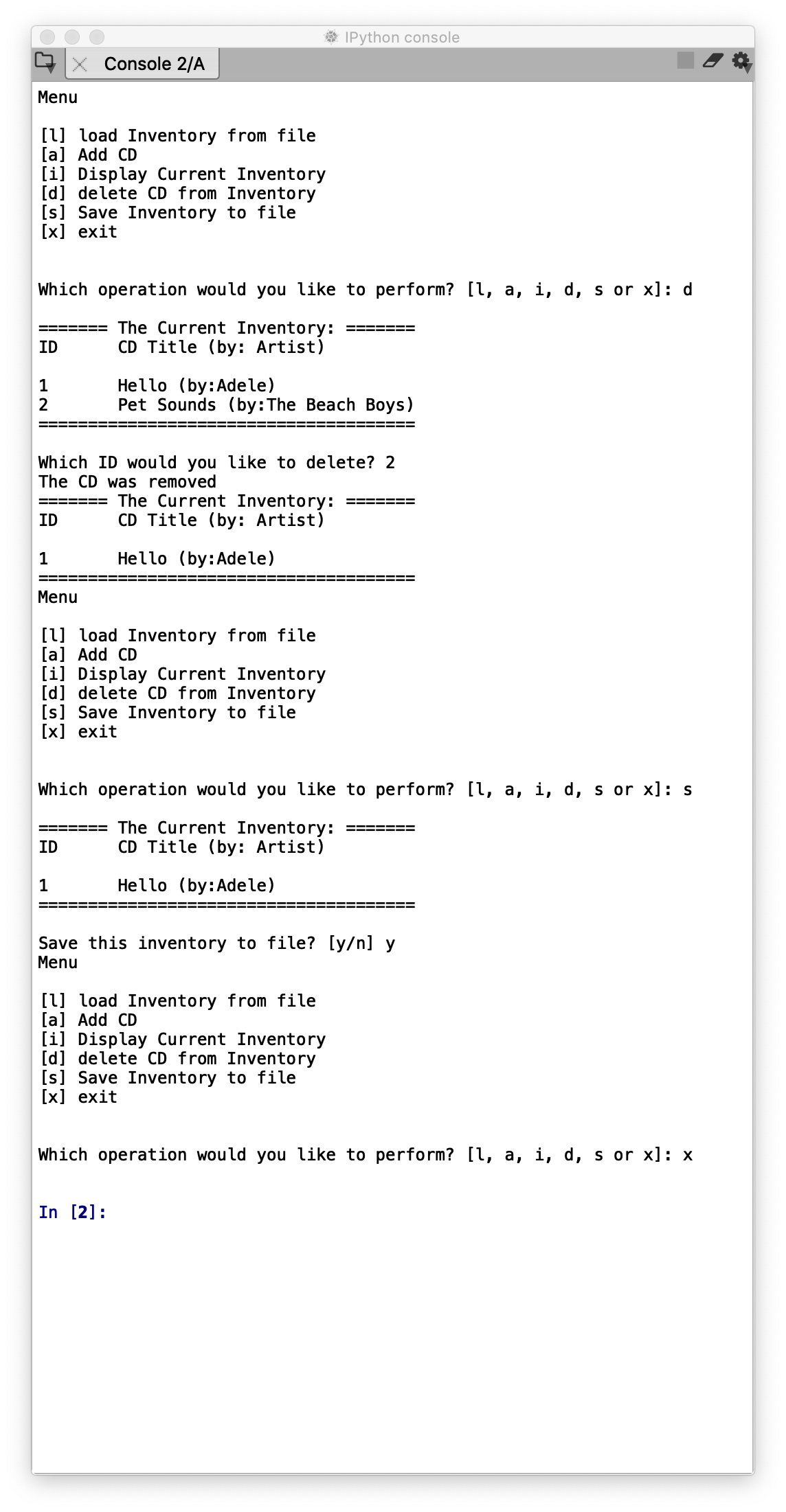
For each of the functions, I created, I will describe it using a docstring. The docstring is a triple quoted string that [describes what the code does](https://www.datacamp.com/community/tutorials/docstrings-python).[[8]](#footnote-8) The docstring can consist of single line as demonstrated in the textbook[[9]](#footnote-9), or consist of multiple parts (such as Google format) with sections devoted to a more detailed description, the arguments that feed into the function, what the function returns, and raises.[[10]](#footnote-10)

Following the example for the Google format provided by the instructor, I created docstrings for remaining functions.

# Running the script

Running the script in Spyder results in the following:

(see next page)

*Figure 1 – Output from Spyder*

**

*Figure 2 – CDInventory.txt as written to by Spyder*

Running the script in the Terminal results in the following:

# 

# *Figure 2 – Output from Terminal*

**

*Figure 3 – CDInventory.txt as written to by the Terminal.*

Link to GitHub

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

Summary

In this assignment, I took an existing script and:

1. Modified a script to add additional functions as directed by the TODO lines provided by the instructor in the script.
2. Added docstrings to the additional functions that I created in the script.
3. Verified the script functions in Spyder and in the Terminal.
4. Uploaded my script and knowledge document into GitHub.

# Appendix

## Listing CDInventory.py

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. # joeprado, 2020-Mar-09, Began modifying Script
7. # joeprado, 2020-Mar-10, Added docstrings.
8. # joeprado, 2020-Mar-10, Finished script modifications
9. #------------------------------------------#
11. # -- DATA -- #
12. strChoice = '' # User input
13. lstTbl = []  # list of lists to hold data
14. dicRow = {}  # list of data row
15. strFileName = 'CDInventory.txt'  # data storage file
16. objFile = None  # file object

19. # -- PROCESSING -- #
20. **class** DataProcessor:
21. @staticmethod
22. **def** add\_cd(cdID, title, artist, row, table):
23. """Adding user data for new CD to a table.
25. Takes user input fed into the function via parameters,formats it as a set key:value pairs in a
26. dictionary, and then appends that dictionary as row nested inside a list.
28. Args:
29. cdID (string): ID number of CD as entered by the user
30. title (string): Title of CD as entered by the user
31. artist (string): Artist name for the CD as entered by the user
32. row (dictionary): empty data row in the form of dictionary
33. table (list of dict): list of dictionaries that holds our data in volatile memory
35. Returns:
36. None.
37. """
38. intID = int(cdID)
39. row = {'ID': intID, 'Title': title, 'Artist': artist}
40. table.append(row)
42. @staticmethod
43. **def** delete\_cd(selectID, table):
44. """Delete a CD selected by user based on ID.
46. Takes user input for the ID number of a CD the user would like to delete, searches for the row (dictionary) the ID is in,
47. then deletes that row (dictionary) in the table (list).
49. Args:
50. selectID (integer): user selection for CD ID number the user would like to delete.
51. table (list of dict): list of dictionaries that holds our data in volatile memory
53. Returns:
54. None.
55. """
56. intRowNr = -1
57. blnCDRemoved = False
58. **for** row **in** table:
59. intRowNr += 1
60. **if** row['ID'] == selectID:
61. **del** table[intRowNr]
62. blnCDRemoved = True
63. **break**
64. **if** blnCDRemoved:
65. **print**('The CD was removed')
66. **else**:
67. **print**('Could not find this CD!')

70. **class** FileProcessor:
71. """Processing the data to and from text file"""
73. @staticmethod
74. **def** read\_file(file\_name, table):
75. """Function to manage data ingestion from file to a list of dictionaries.
77. Reads the data from file identified by file\_name into a 2D table
78. (list of dicts) table one line in the file represents one dictionary row in table.
80. Args:
81. file\_name (string): name of file used to read the data from
82. table (list of dict): 2D data structure (list of dicts) that holds the data during runtime
84. Returns:
85. None.
86. """
87. table.clear()  # this clears existing data and allows to load data from file
88. objFile = open(file\_name, 'r')
89. **for** line **in** objFile:
90. data = line.strip().split(',')
91. dicRow = {'ID': int(data[0]), 'Title': data[1], 'Artist': data[2]}
92. table.append(dicRow)
93. objFile.close()
95. @staticmethod
96. **def** write\_file(file\_name, table):
97. """Function to save the contents of CD Inventory in volatile memory into a text file.
99. Takes the list of dictionaries identified by table and saves it into text file
100. identified by file\_name.

103. Args:
104. file\_name (string): name of file used to save data to.
105. table (list of dict): 2D data structure (list of dicts) that holds the data during runtime
107. Returns:
108. None
109. """
110. objFile = open(file\_name, 'w')
111. **for** row **in** table:
112. lstValues = list(row.values())
113. lstValues[0] = str(lstValues[0])
114. objFile.write(','.join(lstValues) + '\n')
115. objFile.close()

118. # -- PRESENTATION (Input/Output) -- #
120. **class** IO:
121. """Handling Input / Output"""
123. @staticmethod
124. **def** print\_menu():
125. """Displays a menu of choices to the user
127. Args:
128. None.
130. Returns:
131. None.
132. """
134. **print**('Menu\n\n[l] load Inventory from file\n[a] Add CD\n[i] Display Current Inventory')
135. **print**('[d] delete CD from Inventory\n[s] Save Inventory to file\n[x] exit\n')
137. @staticmethod
138. **def** menu\_choice():
139. """Gets user input for menu selection
141. Args:
142. None.
144. Returns:
145. choice (string): a lower case sting of the users input out of the choices l, a, i, d, s or x
147. """
148. choice = ' '
149. **while** choice **not** **in** ['l', 'a', 'i', 'd', 's', 'x']:
150. choice = input('Which operation would you like to perform? [l, a, i, d, s or x]: ').lower().strip()
151. **print**()  # Add extra space for layout
152. **return** choice
154. @staticmethod
155. **def** show\_inventory(table):
156. """Displays current inventory table

159. Args:
160. table (list of dict): 2D data structure (list of dicts) that holds the data during runtime.
162. Returns:
163. None.
164. """
165. **print**('======= The Current Inventory: =======')
166. **print**('ID\tCD Title (by: Artist)\n')
167. **for** row **in** table:
168. **print**('{}\t{} (by:{})'.format(\*row.values()))
169. **print**('======================================')
171. @staticmethod
172. **def** input\_new\_cd():
173. """Function that collects user input for a new CD to be added to inventory.
175. Args:
176. None.
178. Returns:
179. cdID (string): string representing ID number user entered for CD
180. title (string): string representing CD title entered by user
181. artist (string): string representing artist name entered by user
182. """
183. cdID = input('Enter ID: ').strip()
184. title = input('What is the CD\'s title? ').strip()
185. artist = input('What is the Artist\'s name? ').strip()
186. **return** cdID, title, artist

189. # 1. When program starts, calls function that reads in the currently saved Inventory
190. FileProcessor.read\_file(strFileName, lstTbl)
192. # 2. start main loop
193. **while** True:
194. # 2.1 Display Menu to user and get choice
195. IO.print\_menu()
196. strChoice = IO.menu\_choice()
198. # 3. Process menu selection
199. # 3.1 process exit first
200. **if** strChoice == 'x':
201. **break**
202. # 3.2 process load inventory
203. **if** strChoice == 'l':
204. **print**('WARNING: If you continue, all unsaved data will be lost and the Inventory re-loaded from file.')
205. strYesNo = input('type \'yes\' to continue and reload from file. otherwise reload will be canceled: ')
206. **if** strYesNo.lower() == 'yes':
207. **print**('reloading...')
208. FileProcessor.read\_file(strFileName, lstTbl) #Calls function that loads text file containing CD inventory into runtime.
209. IO.show\_inventory(lstTbl) #Calls function that displays inventory to user
210. **else**:
211. input('canceling... Inventory data NOT reloaded. Press [ENTER] to continue to the menu.')
212. IO.show\_inventory(lstTbl) #Calls function that displays inventory to user
213. **continue**  # start loop back at top.
214. # 3.3 process add a CD
215. **elif** strChoice == 'a':
216. # 3.3.1 Calls function that asks user for new ID, CD Title and Artist
217. strID, strTitle, strArtist = IO.input\_new\_cd()
218. # 3.3.2 Calls the function that adds item to the table
219. DataProcessor.add\_cd(strID, strTitle, strArtist, dicRow, lstTbl)
220. #Calls function that displays inventory with added CD
221. IO.show\_inventory(lstTbl)
222. **continue**  # start loop back at top.
223. # 3.4 process display current inventory
224. **elif** strChoice == 'i':
225. IO.show\_inventory(lstTbl) # Calls function that displays current inventory
226. **continue**  # start loop back at top.
227. # 3.5 process delete a CD
228. **elif** strChoice == 'd':
229. # 3.5.1 get Userinput for which CD to delete
230. # 3.5.1.1 Calls function that displays inventory to user
231. IO.show\_inventory(lstTbl)
232. # 3.5.1.2 ask user which ID to remove
233. intIDDel = int(input('Which ID would you like to delete? ').strip())
234. # 3.5.2 Calls function that searches thru table and deletes CD
235. DataProcessor.delete\_cd(intIDDel, lstTbl)
236. IO.show\_inventory(lstTbl) #Calls function that displays inventory to user
237. **continue**  # start loop back at top.
238. # 3.6 process save inventory to file
239. **elif** strChoice == 's':
240. # 3.6.1 Calls function that displays current inventory.
241. IO.show\_inventory(lstTbl)
242. strYesNo = input('Save this inventory to file? [y/n] ').strip().lower() #asks user for confirmation to save
243. # 3.6.2 Process choice
244. **if** strYesNo == 'y':
245. # 3.6.2.1 Calls function that saves data
246. FileProcessor.write\_file(strFileName, lstTbl)
247. **else**:
248. input('The inventory was NOT saved to file. Press [ENTER] to return to the menu.')
249. **continue**  # start loop back at top.
250. # 3.7 catch-all should not be possible, as user choice gets vetted in IO, but to be save:
251. **else**:
252. **print**('General Error')

1. Module 06, Page 21 [↑](#footnote-ref-1)
2. Module 06, Page 21 [↑](#footnote-ref-2)
3. Page 172, Python Programming for the absolute beginner, Third Edition. [↑](#footnote-ref-3)
4. Module 06, Page 06 [↑](#footnote-ref-4)
5. Module 06, Page 07 [↑](#footnote-ref-5)
6. Page 222, Python Programming for the absolute beginner, Third Edition. [↑](#footnote-ref-6)
7. Page 168, Python Programming for the absolute beginner, Third Edition. [↑](#footnote-ref-7)
8. Accessed 10 March 2020 [↑](#footnote-ref-8)
9. Page 161, Python Programming for the absolute beginner, Third Edition. [↑](#footnote-ref-9)
10. Module 06, Page 19 [↑](#footnote-ref-10)