Tom Williams

Date: 08/25/2020

IT FDN 110: Introduction to Programming (Python)

[Assignment\_06](https://github.com/tomwill98103/assignment06)

CD Inventory version 3.0

# Introduction

The goal of the assignment is to rewrite and update a version of the CD Inventory program using functions. The functional requirements are that users can to enter CD data, view the current inventory, save data to a CDInventory.txt data file, load the inventory file into memory, and delete items from inventory.

# Functions and Classes

I am familiar with custom functions from other programming languages, and from math. The python implementation is elegant and minimal, as expected. Classes have always been harder for me to understand. I have heard the “blueprint to house” analogy many times but have never felt comfortable with the concept. The way we used them in this assignment was straightforward, and not very valuable except for organization. I assume there is more to it and that both the complexity and the value will increase.

# Separation of Concerns

Back in the dawn of time I learned the concept of two-tier, three-tier, and n-tier applications. Separation of Concerns seems to mimic the three-tier pattern and so makes a lot of sense. It also lends itself to using functions, as SoC can be thought of a supporting encapsulation[[1]](#footnote-1). However in practice I struggled somewhat to cleanly define the edge between IO and other processes.

# CD Inventory Version 3.0

The instructions were to start with the starter script and at a minimum to make the changes called out by the TODO comments. I chose to try to eliminate nearly all code except function calls from my main loop and create a user friendly control scheme. Overall the program seemed to run as expected.

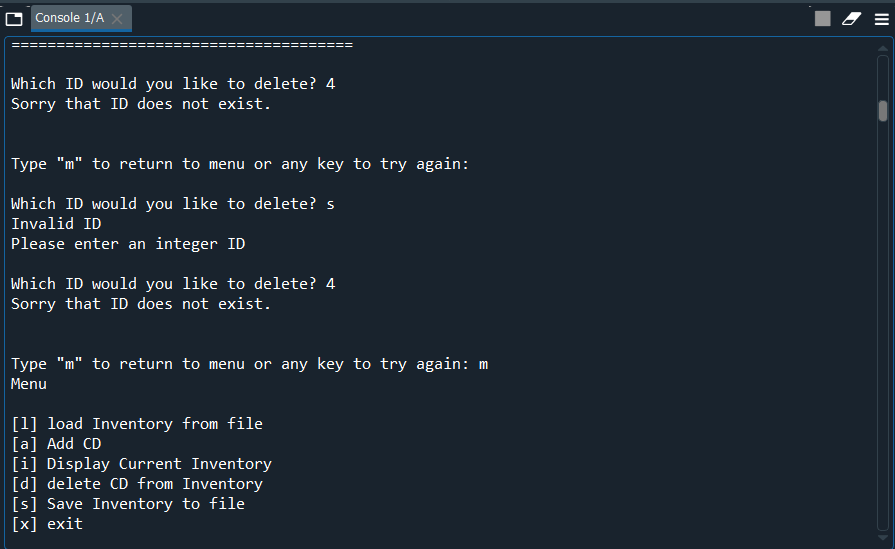


Figure 1: Spyder Console

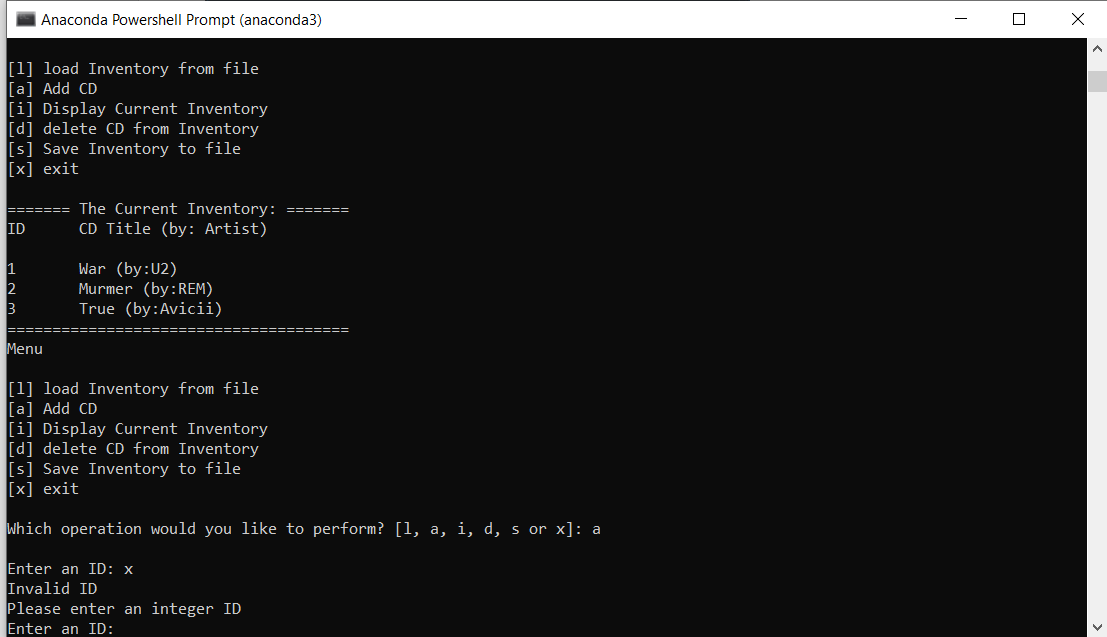


Figure 2: Anaconda Console

## Exception Handling

I really wanted to create a program that could not crash and so included error handling routines in several of my functions. I had not previously used the ‘try..except’ syntax but was able to figure out it from my code review of BrentK’s Assignment05[[2]](#footnote-2) and from DKlos’s model solution[[3]](#footnote-3) to confirm that the user entered an integer for an ID, as seen in Listing 1. This listing also shows a for loop and if statement test for duplicates.

1. def add\_cd(table):

…

1. while True:
2. try:
3. cd\_id = int(input("Enter an ID: "))
4. for cd in table:
5. if cd\_id == cd['ID']:
6. print("Duplicate ID\nPlease try again")
7. break
8. else:
9. break
10. except ValueError:
11. print("Invalid ID\nPlease enter an integer ID")

Listing 1

I also adapted from Assignement05 importing the OS module and testing to make sure the CDInventory.txt file existed as seen in Listing 2

1. def check\_file(file\_name):

…

1. import os
2. while not os.path.exists(file\_name):
3. objFile = open(file\_name, 'a')
4. objFile.close()

Listing 2

## Confirmation of User Intent

User experience is an important aspect of any program and I thought a lot about how to confirm key user choices. The most important cases were those when data could be erased, so loading the file into memory, deleting items, and saving data to file. A basic confirmation routine existed for the load case but needed to be redone as a function, and no routines existed for saving and deleting.

When I started planning my code I could not figure out how to have the confirmation messages work without calling the File or Data processing function from the IO function. Although this is permitted[[4]](#footnote-4), I looked at several ways of avoiding it. I was able to pass the confirmation back to the main loop and use it in an if statement but this did not seem any better. I experimented with returning a confirmation variable that was an argument in the processing loop and instructed it to simply skip execution, but I was nervous about the implications, so I ended up staying with my initial approach.

1. def msg\_sav(filename,table):

…

1. str\_yn = input('Save this inventory to file? [y/n] ').strip().lower()
2. if str\_yn == 'y':
3. FileProcessor().write\_file(filename, table) # Call save function
4. print('Inventory saved\n')
5. else:
6. input('The inventory was NOT saved to file. Press [ENTER] to return to the menu.\n')

Listing 3

## Test Plan

The directions emphasized avoiding crashes so I decided to create a test plan and wrote several scenarios that I thought tested the full range of potential user behavior. I did discover one crashing bug: if after starting the application the user deletes the ‘CDInventory.txt’ file through the shell and then chooses ‘l’ from the menu to load it will crash, as the check\_file function only runs at startup. I considered adding it to the main load function but decided this was akin to trying to prevent the user from accidentally shutting off their machine, and beyond the scope of the assignment.

# Summary

The goal of the assignment was to rewrite and update a version of the CD Inventory program using functions. The functional requirements are that users can to enter CD data, view the current inventory, save data to a CDInventory.txt data file, load the inventory file into memory, and delete items from inventory. I was able to achieve all functionality with nothing but function calls in my main loop. I also tired to observe Separation of Concern, though struggled somewhat in the delineation between IO and processing functions.

# Appendix

Full Code

1. # -- DATA -- #
2. strChoice = '' # User input
3. lstTbl = [] # list of lists to hold data
4. dicRow = {} # list of data row
5. strFileName = 'CDInventory.txt' # data storage file
6. objFile = None # file object
7. tupNewCD = () # holds new CD info
8. strLoadMsg = 'loading data...\n' # load confirmation messag
9. # -- PROCESSING -- #
10. class DataProcessor:
11. """Add and deleting data to list in memory"""
13. @staticmethod
14. def add\_cd(new\_CD, table):
15. """Adds new CD to list in memory
17. Takes user input as tuple and produces entry as dictionary
18. Adds dictionary to list and returns revised inventory
20. Args:
21. new\_CD (tuple):string entries for new CD
22. table (list of dict): 2D data structure that holds data
24. Return:
25. table (list of dict): 2D data structure that holds data
26. """
27. dicRow = {'ID': int(new\_CD[0]), 'Title': new\_CD[1], 'Artist': new\_CD[2]}
28. table.append(dicRow)
29. return table
30. @staticmethod
31. def del\_cd(del\_id,table):
32. """Deletes CD from list in memory
34. Takes user input as int and searches list for corresponding row
35. Returns revised table
37. Args:
38. del\_id (int): entries for new CD
39. table (list of dict): 2D data structure that holds data
41. Return:
42. table (list of dict): 2D data structure that holds data
43. """
44. intRowNr = -1
45. for row in table:
46. intRowNr += 1
47. if row['ID'] == del\_id:
48. del table[intRowNr]
49. break
50. return table
52. class FileProcessor:
53. """Processing the data to and from text file"""
55. @staticmethod
56. def check\_file(file\_name):
57. """Function to check for existence of file and create if needed
58. Imports OS module to check for file
59. Creates empty if doesn't exist
60. Args:
61. file\_name (string): name of file used to read the data from
62. Returns:
63. None.
64. """
65. # check to see if file exists, if not create
66. import os
67. while not os.path.exists(file\_name):
68. objFile = open(file\_name, 'a')
69. objFile.close()
70. @staticmethod
71. def read\_file(file\_name, message, table):
72. """Function to manage data ingestion from file to a list of dictionaries
73. Reads the data from file identified by file\_name into a 2D table.
74. Args:
75. file\_name (string): name of file used to read the data from
76. message (string): confirmation message
77. table (list of dict): 2D data structure (list of dicts) that holds the data during runtime
78. Returns:
79. None.
80. """
82. table.clear() # this clears existing data and allows to load data from file
83. objFile = open(file\_name, 'r')
84. for line in objFile:
85. data = line.strip().split(',')
86. dicRow = {'ID': int(data[0]), 'Title': data[1], 'Artist': data[2]}
87. table.append(dicRow)
88. objFile.close()
89. print(message)
90. @staticmethod
91. def write\_file(file\_name, table):
92. """Function to write inventory to file
93. Reads data row by row from 2D table
94. writes each row as strings to file.
95. Args:
96. file\_name (string): name of file used to read the data from
97. message (string): confirmation message
98. table (list of dict): 2D data structure (list of dicts) that holds the data during runtime
99. Returns:
100. None.
101. """
102. objFile = open(file\_name, 'w')
103. for row in table:
104. lstValues = list(row.values())
105. lstValues[0] = str(lstValues[0])
106. objFile.write(','.join(lstValues) + '\n')
107. objFile.close()
108. # -- PRESENTATION (Input/Output) -- #
109. class IO:
110. """Handling Input / Output"""
111. @staticmethod
112. def print\_menu():
113. """Displays a menu of choices to the user
114. Args:
115. None.
116. Returns:
117. None.
118. """
119. print('Menu\n\n[l] load Inventory from file\n[a] Add CD\n[i] Display Current Inventory')
120. print('[d] delete CD from Inventory\n[s] Save Inventory to file\n[x] exit\n')
121. @staticmethod
122. def menu\_choice():
123. """Gets user input for menu selection
124. Args:
125. None.
126. Returns:
127. choice (string): a lower case sting of the users input out of the choices l, a, i, d, s or x
128. """
129. choice = ' '
130. while choice not in ['l', 'a', 'i', 'd', 's', 'x']:
131. choice = input('Which operation would you like to perform? [l, a, i, d, s or x]: ').lower().strip()
132. print() # Add extra space for layout
133. return choice
135. @staticmethod
136. def msg\_load(filename,table):
137. """Loads data from file into memory
139. Presents warning and gets user confirmtion
140. If confirmed calls load function
141. Args:
142. None.
143. Returns:
144. None.
145. """
146. print('WARNING: If you continue, all unsaved data will be lost and the Inventory re-loaded from file.')
147. str\_yn = input('Type \'yes\' to continue and reload from file. otherwise reload will be canceled: ')
148. if str\_yn.lower() == 'yes':
149. load\_msg='reloading...\n'
150. FileProcessor.read\_file(filename, load\_msg, table) # Call load function
151. else:
152. input('canceling... Inventory data NOT reloaded. Press [ENTER] to continue to the menu.')
153. @staticmethod
154. def show\_inventory(table):
155. """Displays current inventory table
156. Args:
157. table (list of dict): 2D data structure (list of dicts) that holds the data during runtime.
158. Returns:
159. None.
160. """
161. print('======= The Current Inventory: =======')
162. print('ID\tCD Title (by: Artist)\n')
163. for row in table:
164. print('{}\t{} (by:{})'.format(\*row.values()))
165. print('======================================')
166. @staticmethod
167. def add\_cd(table):
168. """Gets user input to add new CD to list in memory
170. Takes user input as integer and strings and returns tuple.
171. Checks for duplicate and invalid ids
173. Args:
174. table (list of dict): 2D data structure that holds data
176. Return:
177. cd\_id: integer id of cd
178. cd\_title: string title of cd
179. cd\_atrist: string artist of cd
180. """
181. while True:
182. try:
183. cd\_id = int(input("Enter an ID: "))
184. for cd in table:
185. if cd\_id == cd['ID']:
186. print("Duplicate ID\nPlease try again")
187. break
188. else:
189. break
190. except ValueError:
191. print("Invalid ID\nPlease enter an integer ID")
193. cd\_title = input('What is the CD\'s title? ').strip()
194. cd\_artist = input('What is the Artist\'s name? ').strip()
195. return cd\_id,cd\_title,cd\_artist
197. @staticmethod
198. def del\_cd(table):
199. """Function to get user input to delete CD from list in memory
201. Takes user input as integer.
202. If found calls delete function
203. If not found gives choice to restart or return to menu with original table
204. Error handling for non-integer entries
206. Args:
207. table (list of dict): current inventory as 2D table
209. Return:
210. table (list of dict): current or revised inventory as 2D table
211. """
212. check\_id = 1
213. while check\_id == 1:
214. try:
215. del\_id=int(input('Which ID would you like to delete? '))
216. for cd in table:
217. if del\_id == cd['ID']:
218. del\_id=cd['ID']
219. check\_id = 0
220. table=DataProcessor().del\_cd(del\_id,table) # Call delete function
221. print('The CD was removed\n')
222. break
223. if check\_id:
224. print('Sorry that ID does not exist.')
225. del\_esc = input('\nType "m" to return to menu or any key to try again: ')
226. if del\_esc == 'm':
227. IO.print\_menu() # Call menu dispaly function
228. break
229. except ValueError:
230. print("Invalid ID\nPlease enter an integer ID")
231. return table
232. @staticmethod
233. def msg\_sav(filename,table):
234. """Gets user input to save list in memory to file
236. Takes user input as string.
237. Calls save function if user confirms
239. Args:
240. filename (string): name of save file
241. table (list of dict): currently inventory in memory as 2D data structure that holds data
243. Return:
244. None
245. """
246. str\_yn = input('Save this inventory to file? [y/n] ').strip().lower()
247. if str\_yn == 'y':
248. FileProcessor().write\_file(filename, table) # Call save function
249. print('Inventory saved\n')
250. else:
251. input('The inventory was NOT saved to file. Press [ENTER] to return to the menu.\n')
252. # 1. When program starts, read in the currently saved Inventory
253. # 2.1 Check to see if file exists, if not create
254. FileProcessor.check\_file(strFileName)
255. # 2.2 Load file and confirm to user
256. FileProcessor.read\_file(strFileName,strLoadMsg,lstTbl)
257. # 2. start main loop
258. while True:
259. # 2.1 Display Menu to user and get choice
260. IO.print\_menu()
261. strChoice = IO.menu\_choice()
262. # 3. Process menu selection
263. # 3.1 process exit first
264. if strChoice == 'x':
265. break
266. # 3.2 process load inventory
267. if strChoice == 'l':
268. # 3.2.1 display warning and get confirmation to call load function
269. IO.msg\_load(strFileName, lstTbl)
270. # 3.2.2 shows inventory
271. IO.show\_inventory(lstTbl)
272. continue # start loop back at top.
273. # 3.3 process add a CD
274. elif strChoice == 'a':
275. # 3.3.1 Ask user for new ID, CD Title and Artist
276. tupNewCD=IO.add\_cd(lstTbl)
277. # 3.3.2 Add item to the table
278. lstTbl=DataProcessor.add\_cd(tupNewCD,lstTbl)
279. # 3.3.3 display updated inventory
280. IO.show\_inventory(lstTbl)
281. continue # start loop back at top.
282. # 3.4 process display current inventory
283. elif strChoice == 'i':
284. IO.show\_inventory(lstTbl)
285. continue # start loop back at top.
286. # 3.5 process delete a CD
287. elif strChoice == 'd':
288. # 3.5.1 display Inventory to user
289. IO.show\_inventory(lstTbl)
290. # 3.5.2 ask user which ID to remove and call del function if confirmed
291. lstTbl = IO.del\_cd(lstTbl)
292. # 3.5.3 display updated inventory
293. IO.show\_inventory(lstTbl)
294. continue # start loop back at top.
295. # 3.6 process save inventory to file
296. elif strChoice == 's':
297. # 3.6.1 Display current inventory
298. IO.show\_inventory(lstTbl)
299. # 3.6.2 Ask user for confirmation and call save function
300. IO.msg\_sav(strFileName, lstTbl)
301. continue # start loop back at top.
302. # 3.7 catch-all should not be possible, as user choice gets vetted in IO, but to be save:
303. else:
304. print('General Error')

1. Dawson, Python Programming for the Absolute Beginner, p. 165 [↑](#footnote-ref-1)
2. <https://github.com/Brent-K/Assignment_05/blob/master/CDInventory.py> retrieved 8/25 [↑](#footnote-ref-2)
3. No link available [↑](#footnote-ref-3)
4. Dawson, Python Programming for the Absolute Beginner, p. 180 [↑](#footnote-ref-4)