Kenny Lu

2020-Feb-29

IT FDN 100 B Winter 2020

Assignment06

GitHub <https://github.com/lukenny/Assignment_06>

# Introduction

In this sixth module, we cover functions, classes and docstring. It’s a very practical module as we are getting into the general organization and best practices of coding python.

Topic 1 – Functions - Multiple Return Values and Print vs. Return

Function is a block or logical grouping of reusable code which runs when it’s called. We can pass parameters into a function and it can also return data as a result. Functions are powerful because it provides the benefits of modularity, consistency and reusability. We can return multiple values with functions; I will demonstrate this concept with the code block below.

1. **def** getCar():
2. make = "Mercedes-Benz"
3. model = "C300"
4. country\_origin = "DE"
5. color = "Black"
6. **return** make,model,country\_origin,color
8. make,model,country\_origin,color = getCar()
9. **print**(getCar()[0])
10. **print**(getCar()[1])
11. **print**(getCar()[2])
12. **print**(getCar()[3])

Listing 1 – sample code to demonstrate return multiple values with function

Another concept that often gets mixed up is ‘Print’ vs. ‘Return’. They behave very similarly at a glance but they are two different things. Let’s take a look at Listing 2 and Listing 3. We have two functions with Listing 2, ‘returnFunction’ and ‘printFunctiion’. And the only difference between the two functions is that one return and the other print. At a glance, when you run it with IDE, both behave very similarly, for instance when you assign a variable to either function, it calculates accordingly. However, print only prints it to the console and return returns it with an object type that can be used or referenced later on. Hence when checking for object type, return returns with int and print returns with NoneType. Long story short, print cannot be referenced as an object type outside of the function.

1. **def** returnFunction(num):
2. num = num\*2
3. **return** num
5. **def** printFunction(num):
6. num = num\*2
7. **print**(num)

Listing 2 – sample functions to demonstrate return and print

1. x = returnFunction(5)
3. x
4. Out[25]: 10
6. y = printFunction(5)
7. 10
9. y
11. type(x)
12. Out[28]: int
14. type(y)
15. Out[29]: NoneType

Listing 3 – sample output to demonstrate return and print

Using [w3schools](https://www.w3schools.com/python/python_functions.asp) (external reference) webpage

Using [youtube](https://www.youtube.com/watch?v=al3ChgHDmVE) (external reference) webpage

# Topic 2 – Static method

When we work with Python code base, we often come across @staticmethod, so I thought it’d be a good idea to go over static methods with Python. Static method can be called or referenced without creating an object or instance. Normally, calling a class method requires you to create an object and call the method. However, it’s not required with static methods. Let’s use some examples below to demonstrate static method and how we can call it directly. With Listing 4, when you run it, it simply prints start pretty straight-forward. However, if you try to run it without creating an object in Listing 5, it throws an error “Spaceship.start()

TypeError: start() missing 1 required positional argument: 'self'”

1. **class** Spaceship(object):
2. **def** start(self):
3. **print**('start')
5. obj = Spaceship()
6. obj.start()

Listing 4 – sample code non static method

1. **class** Spaceship(object):
2. **def** start(self):
3. **print**('start')
5. Spaceship.start()

Listing 5 sample code non static method

With @staticmethod, it does not require to create an object and can be referenced directly as long as the class is defined see Listing 6.

1. **class** Spaceship:
3. @staticmethod
4. **def** start():
5. **print**('start')
7. Spaceship.start()

Listing 6 sample code static method

Using [pythonbasics](https://pythonbasics.org/static-method/) (external reference) webpage

Using [pythonprogramminglanguage](https://pythonprogramminglanguage.com/static-methods/) (external reference) webpage

# Summary

Again, another very useful and practical module where we can apply the knowledge immediately to the assignment and even code base at work. For me, the code for assignment 05 was a little unorganized; hence, it’s a logical progression to organize them with different functions. The only challenge I had was to separate input with processing and how to pass the arguments between functions. I decided to initialize the variable as global and return them in tuple which allows another function to reference it.

# Appendix

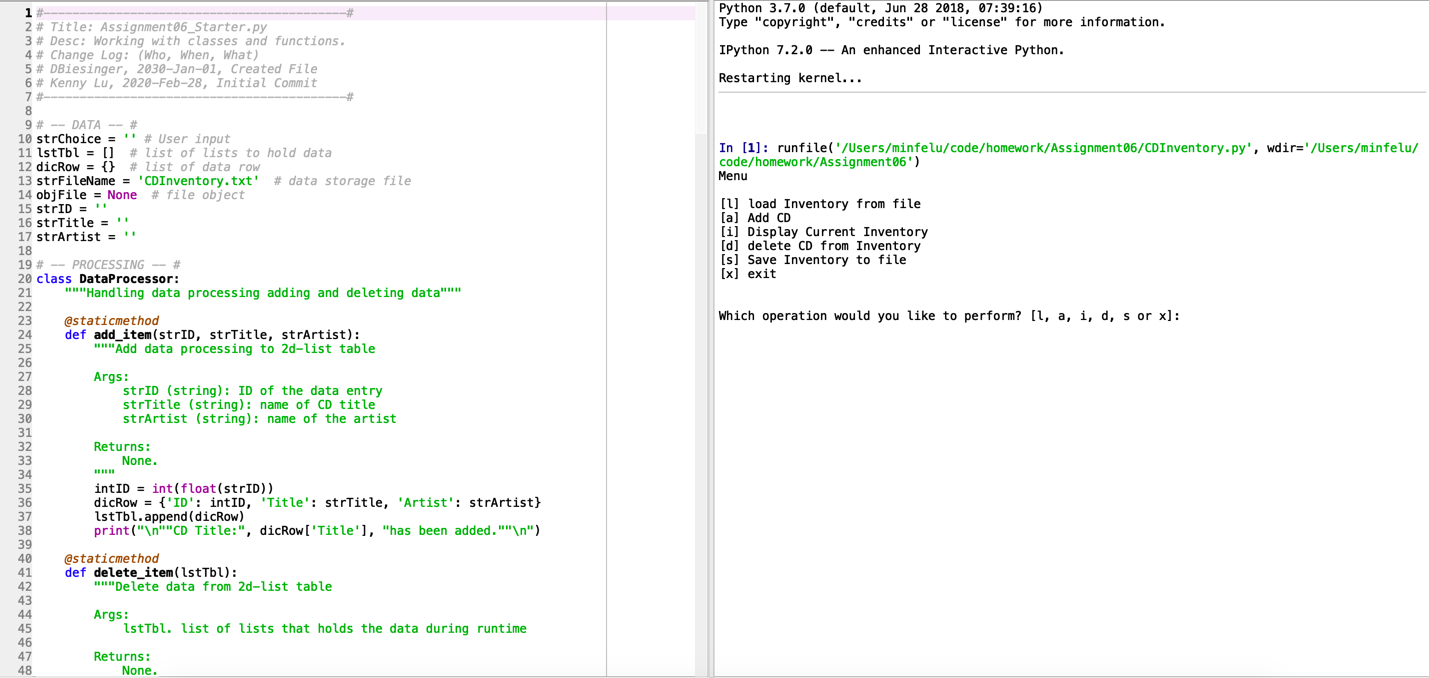


Figure 1 – Example output from script initial loaded in Spyder

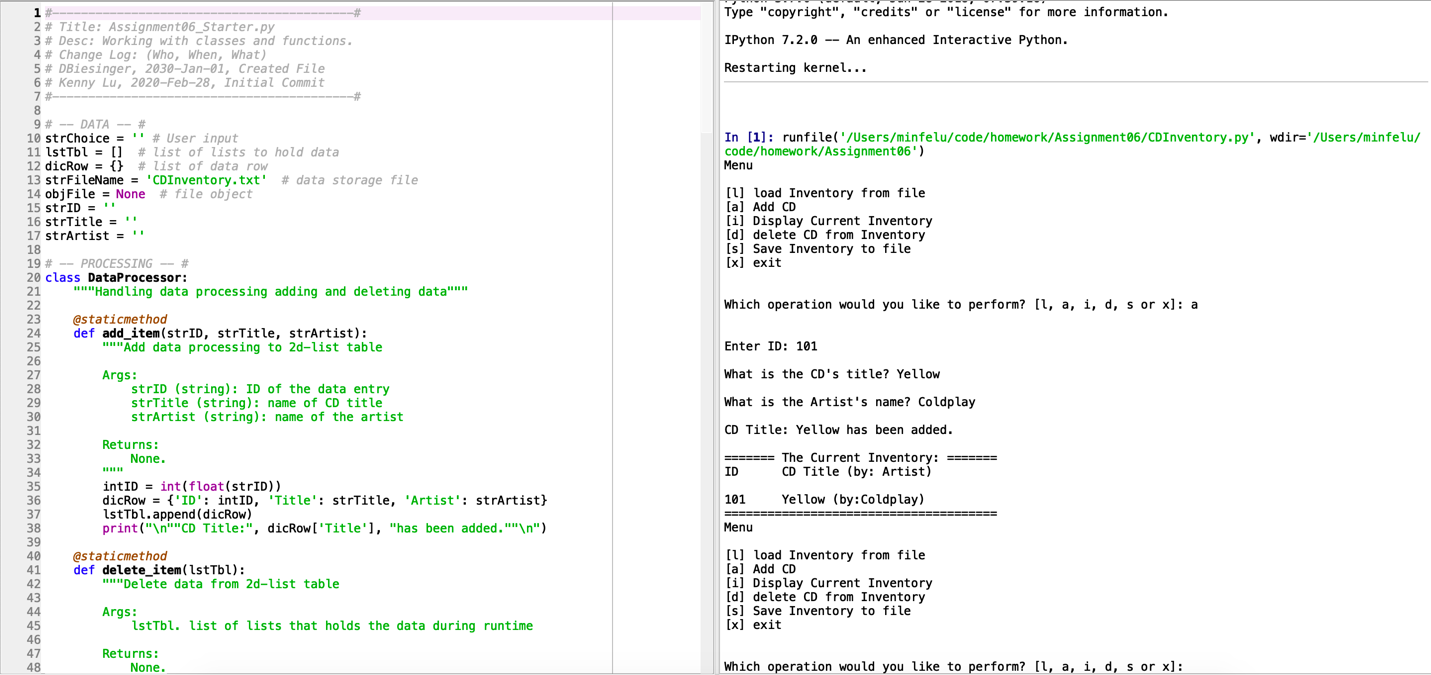


Figure 2 – Example output from script add data and display inventory

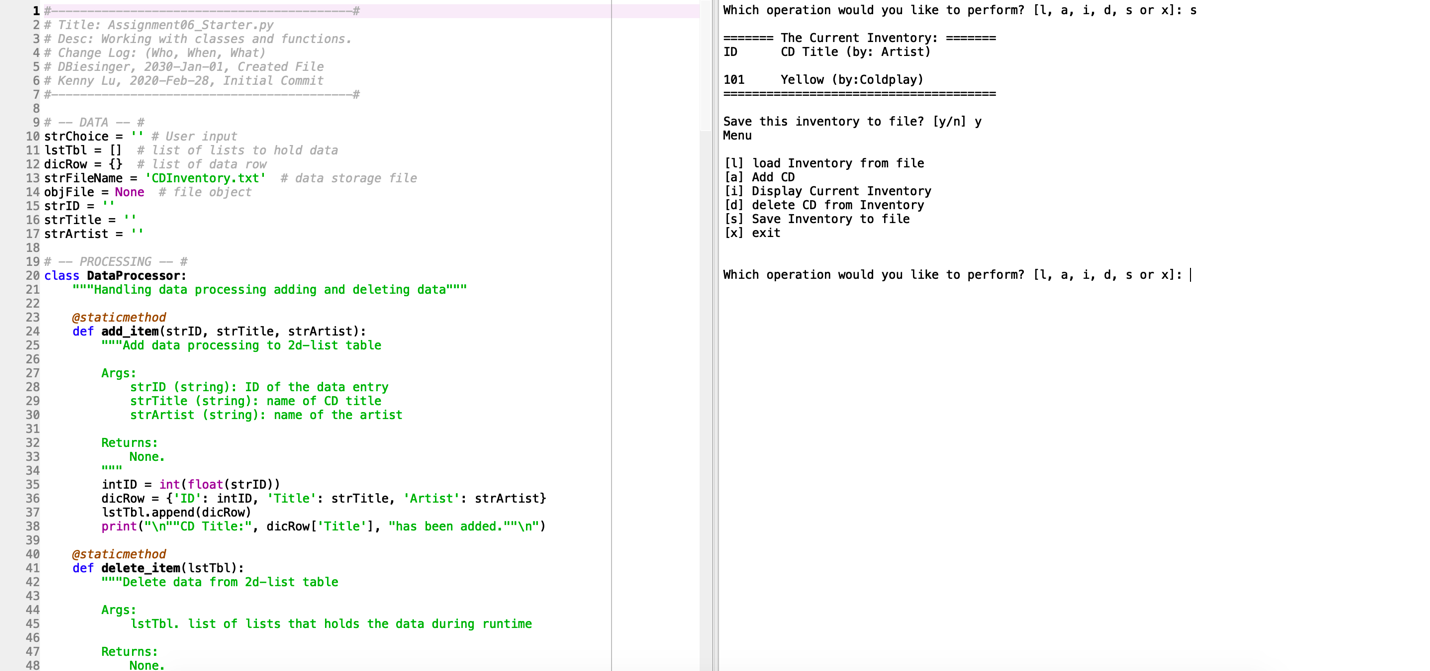


Figure 3 – Example output from script save inventory

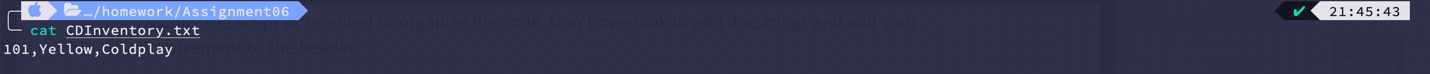
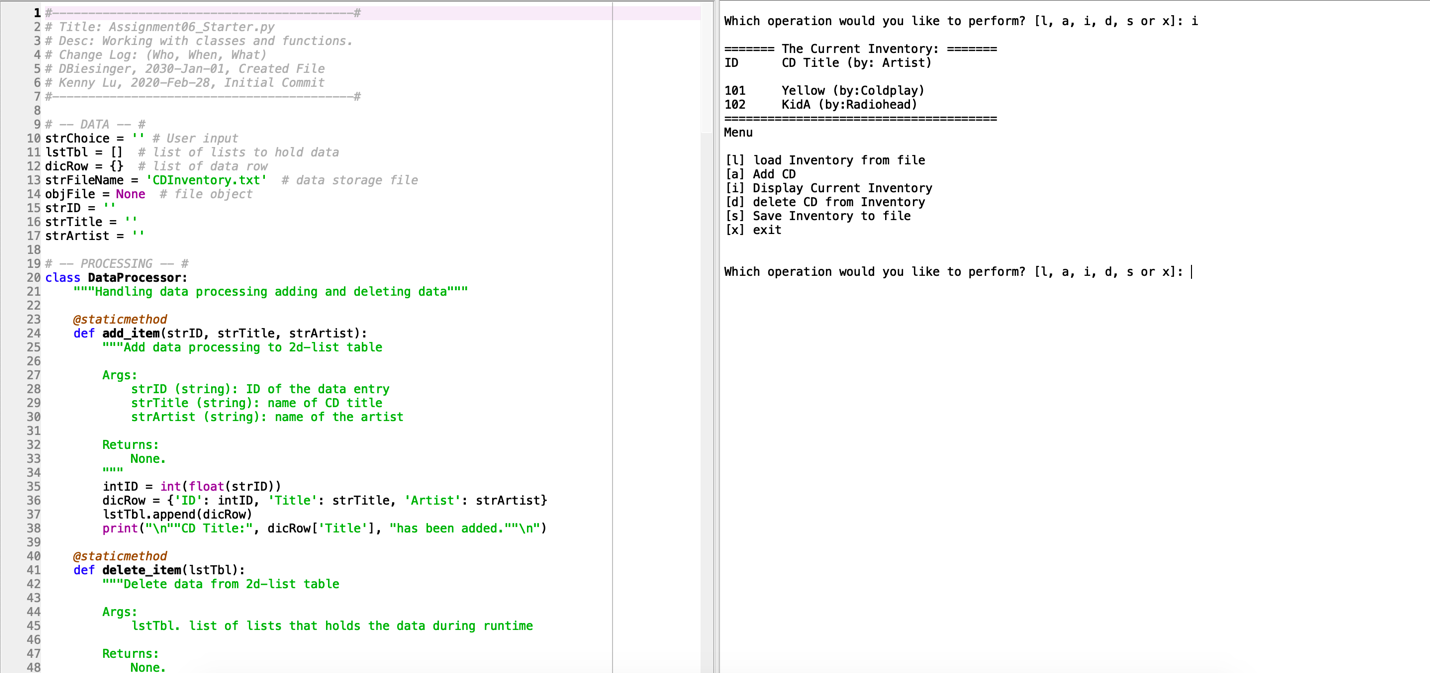


Figure 4 – Verify CDInventory.txt has saved inventory

  
Figure 5 – Example output from script display CDInventory in memory

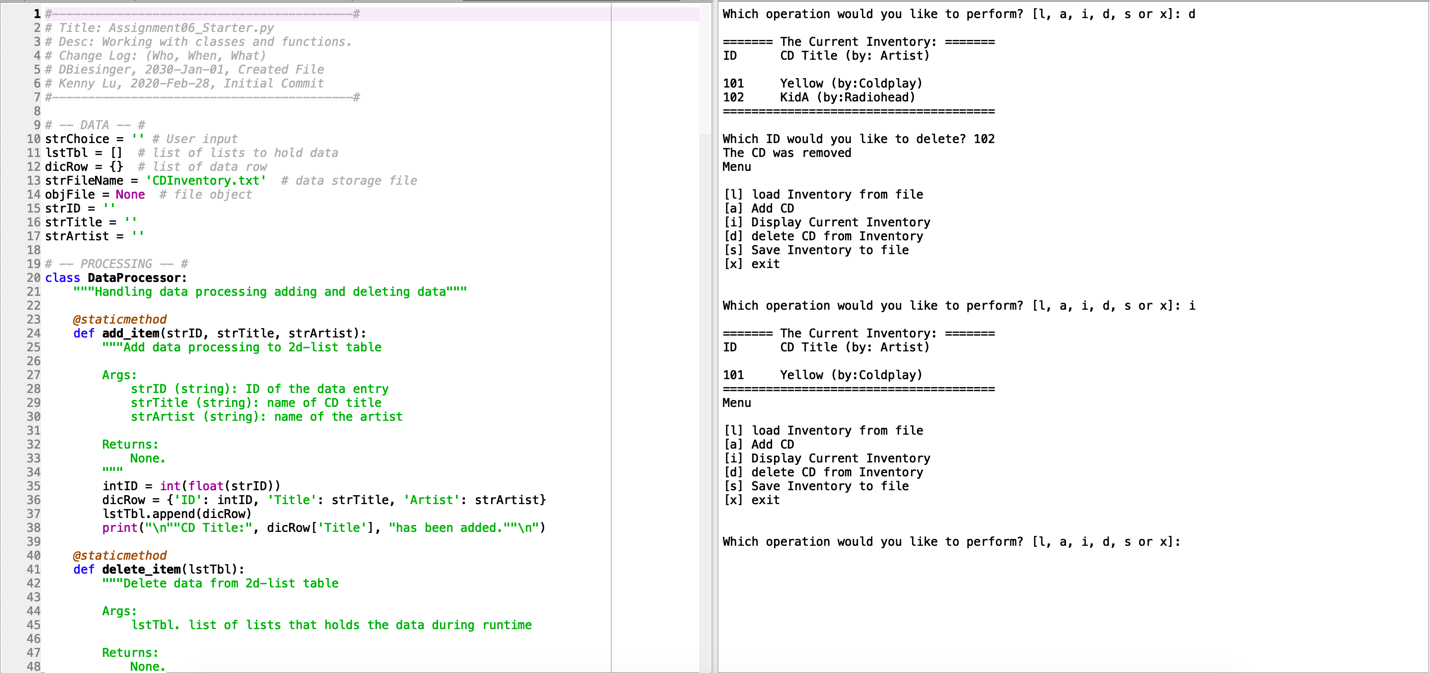


Figure 6 – Example output from script delete data from memory

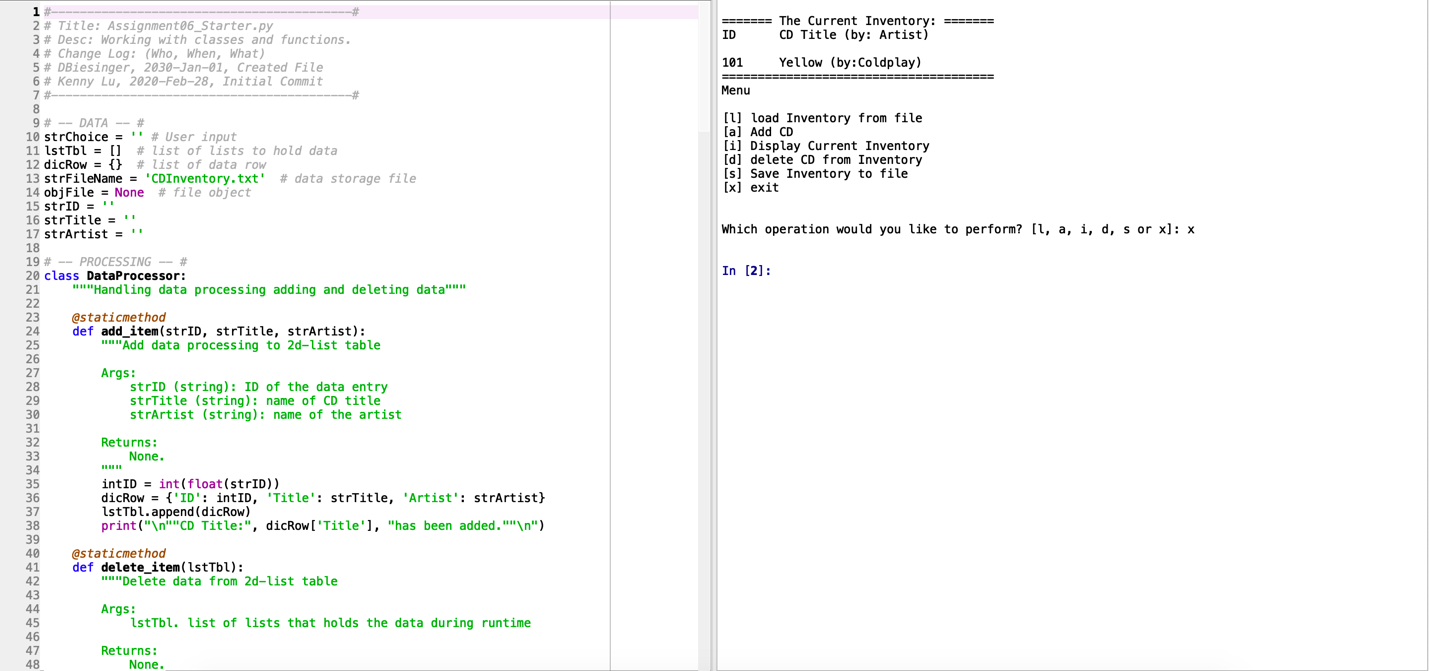


Figure 7 – Example output from script exit program

1. #------------------------------------------#
2. # Title: Assignment06\_Starter.py
3. # Desc: Working with classes and functions.
4. # Change Log: (Who, When, What)
5. # DBiesinger, 2030-Jan-01, Created File
6. # Kenny Lu, 2020-Feb-28, Initial Commit
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
15. strID = ''
16. strTitle = ''
17. strArtist = ''
19. # -- PROCESSING -- #
20. **class** DataProcessor:
21. """Handling data processing adding and deleting data"""
23. @staticmethod
24. **def** add\_item(strID, strTitle, strArtist):
25. """Add data processing to 2d-list table
27. Args:
28. strID (string): ID of the data entry
29. strTitle (string): name of CD title
30. strArtist (string): name of the artist
32. Returns:
33. None.
34. """
35. intID = int(float(strID))
36. dicRow = {'ID': intID, 'Title': strTitle, 'Artist': strArtist}
37. lstTbl.append(dicRow)
38. **print**("\n""CD Title:", dicRow['Title'], "has been added.""\n")
40. @staticmethod
41. **def** delete\_item(lstTbl):
42. """Delete data from 2d-list table
44. Args:
45. lstTbl. list of lists that holds the data during runtime
47. Returns:
48. None.
50. """
51. intRowNr = -1
52. blnCDRemoved = False
53. **for** row **in** lstTbl:
54. intRowNr += 1
55. **if** row['ID'] == intIDDel:
56. **del** lstTbl[intRowNr]
57. blnCDRemoved = True
58. **break**
59. **if** blnCDRemoved:
60. **print**('The CD was removed')
61. **else**:
62. **print**('Could not find this CD!')
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) table 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 dict): 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, 'w+')
83. **for** line **in** objFile:
84. data = line.strip().split(',')
85. dicRow = {'ID': (data[0]), 'Title': data[1], 'Artist': data[2]}
86. table.append(dicRow)
87. objFile.close()
89. @staticmethod
90. **def** write\_file(file\_name, table):
91. """Function to write data in memory to file
93. Reads the data from memory and write to a data storage file
95. Args:
96. file\_name (string): name of file used to read the data from
97. table (list of dict): 2D data structure (list of dicts) that holds the data during runtime
99. Returns:
100. None.
101. """
102. objFile = open(strFileName, 'w')
103. **for** row **in** lstTbl:
104. lstValues = list(row.values())
105. lstValues[0] = str(lstValues[0])
106. objFile.write(','.join(lstValues) + '\n')
107. 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. """
125. **print**('Menu\n\n[l] load Inventory from file\n[a] Add CD\n[i] Display Current Inventory')
126. **print**('[d] delete CD from Inventory\n[s] Save Inventory to file\n[x] exit\n')
128. @staticmethod
129. **def** menu\_choice():
130. """Gets user input for menu selection
132. Args:
133. None.
135. Returns:
136. choice (string): a lower case sting of the users input out of the choices l, a, i, d, s or x
138. """
139. choice = ' '
140. **while** choice **not** **in** ['l', 'a', 'i', 'd', 's', 'x']:
141. choice = input('Which operation would you like to perform? [l, a, i, d, s or x]: ').lower().strip()
142. **print**()  # Add extra space for layout
143. **return** choice
145. @staticmethod
146. **def** show\_inventory(table):
147. """Displays current inventory table
149. Args:
150. table (list of dict): 2D data structure (list of dicts) that holds the data during runtime.
152. Returns:
153. None.
155. """
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**('======================================')
162. @staticmethod
163. **def** add\_data():
164. """Add data to 2d-list table from user input
166. Args:
167. None.
169. Returns:
170. None.
172. """
173. **global** strID
174. strID = input('Enter ID: ').strip()
175. **global** strTitle
176. strTitle = input('What is the CD\'s title? ').strip()
177. **global** strArtist
178. strArtist = input('What is the Artist\'s name? ').strip()
179. **return** (strID, strTitle, strArtist)
181. # 1. When program starts, read in the currently saved Inventory
182. FileProcessor.read\_file(strFileName, lstTbl)
184. # 2. start main loop
185. **while** True:
186. # 2.1 Display Menu to user and get choice
187. IO.print\_menu()
188. strChoice = IO.menu\_choice()
190. # 3. Process menu selection
191. # 3.1 process exit first
192. **if** strChoice == 'x':
193. **break**
194. # 3.2 process load inventory
195. **if** strChoice == 'l':
196. **print**('WARNING: If you continue, all unsaved data will be lost and the Inventory re-loaded from file.')
197. strYesNo = input('type \'yes\' to continue and reload from file. otherwise reload will be canceled')
198. **if** strYesNo.lower() == 'yes':
199. **print**('reloading...')
200. FileProcessor.read\_file(strFileName, lstTbl)
201. IO.show\_inventory(lstTbl)
202. **else**:
203. input('canceling... Inventory data NOT reloaded. Press [ENTER] to continue to the menu.')
204. IO.show\_inventory(lstTbl)
205. **continue**  # start loop back at top.
206. # 3.3 process add a CD
207. **elif** strChoice == 'a':
208. # 3.3.1 Ask user for new ID, CD Title and Artist
209. IO.add\_data()
210. # 3.3.2 Add item to the table
211. DataProcessor.add\_item(strID, strTitle, strArtist)
212. IO.show\_inventory(lstTbl)
213. **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.
218. # 3.5 process delete a CD
219. **elif** strChoice == 'd':
220. # 3.5.1 get Userinput for which CD to delete
221. # 3.5.1.1 display Inventory to user
222. IO.show\_inventory(lstTbl)
223. # 3.5.1.2 ask user which ID to remove
224. intIDDel = int(input('Which ID would you like to delete? ').strip())
225. # 3.5.2 search thru table and delete CD
226. DataProcessor.delete\_item(lstTbl)
227. **continue**  # start loop back at top.
228. # 3.6 process save inventory to file
229. **elif** strChoice == 's':
230. # 3.6.1 Display current inventory and ask user for confirmation to save
231. IO.show\_inventory(lstTbl)
232. strYesNo = input('Save this inventory to file? [y/n] ').strip().lower()
233. # 3.6.2 Process choice
234. **if** strYesNo == 'y':
235. # 3.6.2.1 save data
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.
240. # 3.7 catch-all should not be possible, as user choice gets vetted in IO, but to be save:
241. **else**:
242. **print**('General Error')

Figure 8 – CDInventory.py