Kelly Kauffman

August 18, 2020

IT FDN 110: Introduction to Programming: Python

Assignment 06

Module 06: Functions

# Introduction

Module 06 taught me how to define and use custom functions in my code. In addition, I got a better idea of what classes are and how they are beneficial. I applied my new understanding of functions to complete Assignment 06. For this assignment, I was asked to complete TODO items from a starter code. These TODO items included moving functionality from the main code to individual functions. This document will describe the difficulties and steps I took while learning the material and completing my assignment.

# Topic 1: Module 05 Learning and Labs

At this point in the class, we are starting to see longer and longer pieces of code. As a result, I am having more trouble following these pieces of code and understanding how all the pieces fit together. For example, the tic-tac-toe game program on pages 175-185 in the textbook[[1]](#footnote-1) was confusing and overwhelming at first. I did not understand where some variables were coming from and where they were getting redefined with a different value (i.e. a different board). Even after reading the main code, it was unclear to me how the variables and functions work together to run the tic-tac-toe game. I knew I was going to need to get familiar with programs that contain one or two functions at first before I can fully understand programs that are comprised of many functions. The labs and Assignment 06 provided me the practice I needed. Now looking back on the tic-tac-toe game example, I am better able to understand what is happening in the code.

I really enjoyed working on the labs from this module. They were the perfect way to practice functions because I had already written a similar program for an earlier assignment and the functions I was writing were very simple. I had no problems completing Lab A. I just had to spend a little bit of time figuring out what .format did. It will put the values of the variables that come after it into the curly brackets that come before it.

Lab B gave me a little trouble. When I ran the first draft of my code, I got an error that said, “list index out of range”. I had forgotten to call my function in the print statements. Instead, I had typed the variable for my list, ans\_list, which was empty when the program got to the Presentation concern with the print statements. Listing #1 in the Appendix shows the final code for Lab B. The print statements I am referring to are in lines 30, 32, 34, 36 of Figure 1. Next, I swapped all instances of ans\_list in the print statements for my function, math( ). Then I got an error that said, “math( ) takes 0 positional argument but 1 was given”. I did not know what that meant, so I looked through the Module 06 PDF to see what I was doing differently from the examples. Listing 5 in the PDF helped me realize that I was not unpacking the list, ans\_list, that was being returned by my function. Using the unpacking strategy from Listing 6, I was able to successfully display all the answers to the math computations. I listed a new variable for each answer and set those variables equal to my function. Then in my print statements I used these new variables. See Listing #1 in the Appendix for the exact code I used to unpack the returned list.

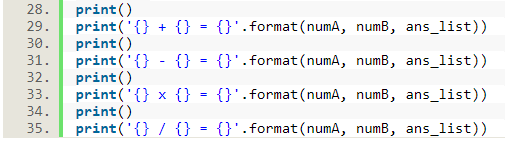


Figure Lab B: Using the variable for an empty instead of calling my function

# Topic 2: Assignment 06 TODOs

I came across fewer difficulties when working on Assignment 06 compared to previous assignments. At first, I was overwhelmed looking at the starter code for Assignment 06. A lot was going on with that code. It took me some time to digest it all and figure out what the TODOs were asking of me. I was glad the code for the menu options was done for me and all I needed to focus on were the functions and formatting. The labs did a great job teaching me how to set up a function, so the TODOs did not take long for me to initially complete. The entire code for my first draft of Assignment 06 is shown in Listing #2 in the Appendix.

As I was working on the TODOs, I thought I was probably going to have a problem with the two functions in the “Add a CD” part of the main code. One function asks a user for their inputs, IO.user\_inputs ( ), and the other puts these inputs into a dictionary and adds the dictionary to a 2D list, DataProcessor.create\_table ( ). I did not think I was successfully transferring the user inputs over to the DataProcessor.create\_table ( ) function.

When I ran my code for the first time, I got a syntax error that was surprisingly not related to my suspicion. I had forgotten the parenthesis after calling my function (see line 204 in Appendix Listing #2). I figured out the problem by using some test print statements and comparing my work to the examples in the Module 06 PDF. Next, I tried the code again and received the error in Figure 2. It seems I was right all along that the user inputs from the IO.user\_inputs ( ) function were not being received by the DataProcessor.create\_table ( ) function. For this first draft, I had nested IO.user\_inputs ( ) inside of DataProcessor.create\_table ( ) (see line 207 in Appendix Listing #2) as a way to transfer the tuple of user inputs between functions. The error I received told me that DataProcessor.create\_table ( ) was only receiving one argument. I had set this function up to receive three arguments (see line 20 in Appendix Listing #2).

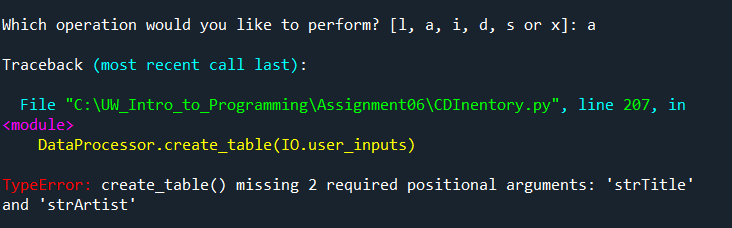


Figure Error in my Assignment 06 first draft telling me DataProcessor.create\_table () was only receiving one argument.

I found that the example in Listing 7 in the Module 06 PDF was close to what I was trying to accomplish, so I tried to mirror my code to match what was in the example. I show this attempt in Figure 3 by only including these new versions of IO.user\_inputs ( ), DataProcessor.create\_table ( ), and the code from the “Add a CD” part of the main loop. Unfortunately, I was still getting an error message. I decided to reach out to our TA, Doug, for help.

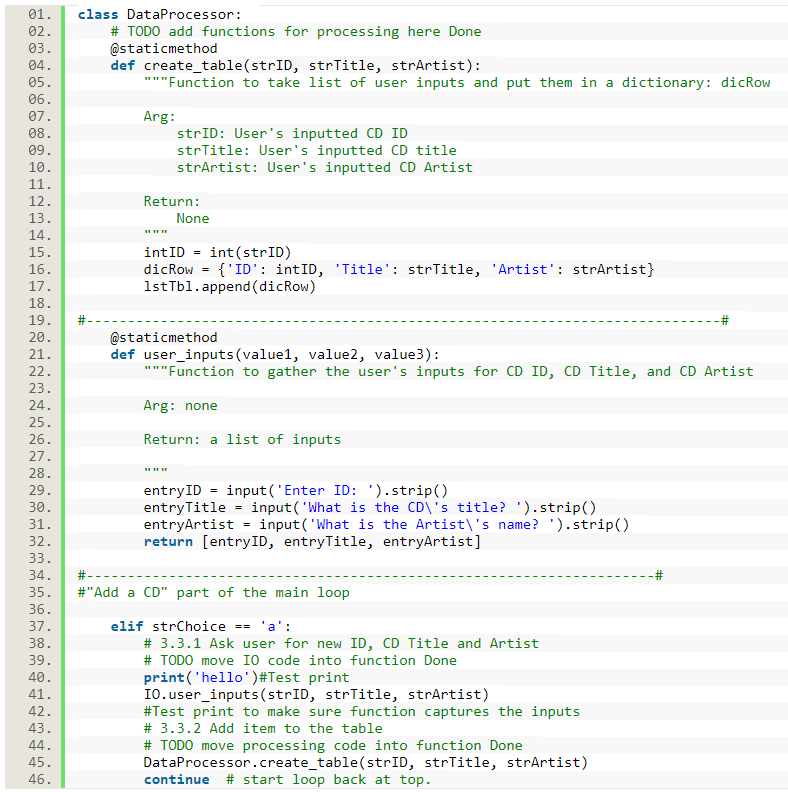


Figure My attempt at transferring user inputs between two functions using Listing 7 of the Module 06 PDF as a guide

Between Doug’s help and the examples we talked about in class, I was able to solve my problem and get DataProcessor.create\_table ( ) to use the list of user inputs returned by IO.user\_inputs ( ). I needed to save the tuple (I decided to go back to a tuple like in my first draft instead of the list in Figure 3) returned by IO.user\_inputs ( ) to a variable before calling DataProcessor.create\_table ( ). I also needed to unpack the tuple of user inputs either before calling the DataProcessor.create\_table ( ) or as a part of this function. I decided to unpack the tuple and assign the return values to a variable all in the same step as I did in Lab B. See Listing #3 in the Appendix for my final code for Assignment 06.

This major difficulty I faced when completing Assignment 06 reinforced valuable concepts for me. I need to make sure the returns of my functions are usable by other functions or pieces of code by assigning the returns to variables. I also need to remember lists, tuples, and dictionaries need to be unpacked before another function can make use of them.

# Topic 3: Final Products

Here is the link to the final code for Assignment 06 on Github: <https://github.com/pythongal6295/Assignment_06.git>.

Below, Figure 4 and 5 show my code working in Spyder. Figure 6 and 7 show my code working in Anaconda.

A screenshot of a cell phone

Description automatically generated

Figure Part1: Assignment 06 working in Spyder

A screenshot of a cell phone

Description automatically generated

Figure Part 2: Assignment 06 working in Spyder

A screenshot of a cell phone

Description automatically generated

Figure Part 1: Assignment 06 working in the Anaconda Prompt

A screenshot of a cell phone

Description automatically generated

Figure Part 2: Assignment 06 working in the Anaconda Prompt

# Summary

Functions were the main focus of Module 06. I learned how to define custom functions, nest them under different classes, and write docstrings to describe the functions. I learned new vocabulary like return, argument, parameter, global/local variables, and class. I used what I learned about functions to complete Assignment 06. For this assignment I was asked to complete the TODOs listed in the starter code, Assignment06\_Starter.py. These TODOs had me move functionalities from the main loop of the code into individual functions and then use these functions in the main loop.

# Appendix

Listing #1: Lab B Final Code

1. #-------------------------------------#
2. #Title: Lab\_B6.py
3. #Desc: Testing functions (math operations)
4. #Change Log: (Who, When, What)
5. #KKauffman, Aug 15 2020, Created File
6. #-------------------------------------#
8. # -- DATA -- #
10. numA = int(input('Please enter an integer. '))
11. numB = int(input('Please enter another integer. '))
12. ans\_list = []
14. # -- PROCESSING -- #
16. **def** math():
17. total = numA + numB
18. diff = numA - numB
19. product = numA \* numB
20. quotient = numA / numB
21. ans\_list = [total, diff, product, quotient]
22. **return** ans\_list
24. # -- PRESENTATION (INPUT/OUTPUT) -- #
26. ans\_1, ans\_2, ans\_3, ans\_4 = math()
28. **print**()
29. **print**('{} + {} = {}'.format(numA, numB, ans\_1))
30. **print**()
31. **print**('{} - {} = {}'.format(numA, numB, ans\_2))
32. **print**()
33. **print**('{} x {} = {}'.format(numA, numB, ans\_3))
34. **print**()
35. **print**('{} / {} = {}'.format(numA, numB, ans\_4))

Listing #2: First Draft of Assignment 06

1. #------------------------------------------#
2. # Title: Assignment06\_Starter.py
3. # Desc: Working with classes and functions.
4. # Change Log: (Who, When, What)
5. # KKauffman, Aug-17-2020, Created File
6. #------------------------------------------#
8. # -- DATA -- #
9. strChoice = '' # User input
10. lstTbl = []  # list of dicts to hold data
11. dicRow = {}  # dict of data row
12. strFileName = 'CDInventory.txt'  # data storage file
13. objFile = None  # file object

16. # -- PROCESSING -- #
17. **class** DataProcessor:
18. # TODO add functions for processing here Done
19. @staticmethod
20. **def** create\_table(strID, strTitle, strArtist):
21. """Function to take list of user inputs and put them in a dictionary: dicRow
23. Arg:
24. strID: User's inputted CD ID
25. strTitle: User's inputted CD title
26. strArtist: User's inputted CD Artist
28. Return:
29. None
30. """
31. intID = int(strID)
32. dicRow = {'ID': intID, 'Title': strTitle, 'Artist': strArtist}
33. lstTbl.append(dicRow)
35. **def** delete\_data(table):
36. """Function to delete CD data from memory
38. Arg: data from lstTbl that is currently in memory
40. Return:
41. None
43. """
44. intRowNr = -1
45. blnCDRemoved = False
46. **for** row **in** lstTbl:
47. intRowNr += 1
48. **if** row['ID'] == intIDDel:
49. **del** lstTbl[intRowNr]
50. blnCDRemoved = True
51. **break**
52. **if** blnCDRemoved:
53. **print**('The CD was removed')
54. **else**:
55. **print**('Could not find this CD!')
57. **class** FileProcessor:
58. """Processing the data to and from text file"""
60. @staticmethod
61. **def** read\_file(file\_name, table):
62. """Function to manage data ingestion from file to a list of dictionaries
64. Reads the data from file identified by file\_name into a 2D table
65. (list of dicts) table one line in the file represents one dictionary row in table.
67. Args:
68. file\_name (string): name of file used to read the data from
69. table (list of dict): 2D data structure (list of dicts) that holds the data during runtime
71. Returns:
72. None.
73. """
74. table.clear()  # this clears existing data and allows to load data from file
75. objFile = open(file\_name, 'r')
76. **for** line **in** objFile:
77. data = line.strip().split(',')
78. dicRow = {'ID': int(data[0]), 'Title': data[1], 'Artist': data[2]}
79. table.append(dicRow)
80. objFile.close()
82. @staticmethod
83. **def** write\_file(file\_name, table):
84. """Function to write data to file
86. Takes each row from lsttable and separates the items in it by a comma
87. and adds a \n at the end
89. Args:
90. file\_name (string): name of file data is saved to
91. table (lists of dicts): 2D data structure that holds the data
93. Returns: print statement confirming save is complete
95. """
96. # TODO Add code here Done
97. objFile = open(strFileName, 'w')
98. **for** row **in** lstTbl:
99. lstValues = list(row.values())
100. lstValues[0] = str(lstValues[0])
101. objFile.write(','.join(lstValues) + '\n')
102. objFile.close()
103. **return** **print**('\nYour data has been saved.')
105. # -- PRESENTATION (Input/Output) -- #
107. **class** IO:
108. """Handling Input / Output"""
110. @staticmethod
111. **def** print\_menu():
112. """Displays a menu of choices to the user
114. Args:
115. None.
117. Returns:
118. None.
119. """
121. **print**('Menu\n\n[l] load Inventory from file\n[a] Add CD\n[i] Display Current Inventory')
122. **print**('[d] delete CD from Inventory\n[s] Save Inventory to file\n[x] exit\n')
124. @staticmethod
125. **def** menu\_choice():
126. """Gets user input for menu selection
128. Args:
129. None.
131. Returns:
132. choice (string): a lower case sting of the users input out of the choices l, a, i, d, s or x
134. """
135. choice = ' '
136. **while** choice **not** **in** ['l', 'a', 'i', 'd', 's', 'x']:
137. choice = input('Which operation would you like to perform? [l, a, i, d, s or x]: ').lower().strip()
138. **print**()  # Add extra space for layout
139. **return** choice
141. @staticmethod
142. **def** show\_inventory(table):
143. """Displays current inventory table

146. Args:
147. table (list of dict): 2D data structure (list of dicts) that holds the data during runtime.
149. Returns:
150. None.
152. """
153. **print**('======= The Current Inventory: =======')
154. **print**('ID\tCD Title (by: Artist)\n')
155. **for** row **in** table:
156. **print**('{}\t{} (by:{})'.format(\*row.values()))
157. **print**('======================================')
159. @staticmethod
160. **def** user\_inputs():
161. """Function to gather the user's inputs for CD ID, CD Title, and CD Artist
163. Arg: none
165. Return: a list of inputs
167. """
168. strID = input('Enter ID: ').strip()
169. strTitle = input('What is the CD\'s title? ').strip()
170. strArtist = input('What is the Artist\'s name? ').strip()
171. **return** (strID, strTitle, strArtist)
173. # TODO add I/O functions as needed
175. # 1. When program starts, read in the currently saved Inventory
176. FileProcessor.read\_file(strFileName, lstTbl)
178. # 2. start main loop
179. **while** True:
180. # 2.1 Display Menu to user and get choice
181. IO.print\_menu()
182. strChoice = IO.menu\_choice()
184. # 3. Process menu selection
185. # 3.1 process exit first
186. **if** strChoice == 'x':
187. **break**
188. # 3.2 process load inventory
189. **if** strChoice == 'l':
190. **print**('WARNING: If you continue, all unsaved data will be lost and the Inventory re-loaded from file.')
191. strYesNo = input('type \'yes\' to continue and reload from file. otherwise reload will be canceled')
192. **if** strYesNo.lower() == 'yes':
193. **print**('reloading...')
194. FileProcessor.read\_file(strFileName, lstTbl)
195. IO.show\_inventory(lstTbl)
196. **else**:
197. input('canceling... Inventory data NOT reloaded. Press [ENTER] to continue to the menu.')
198. IO.show\_inventory(lstTbl)
199. **continue**  # start loop back at top.
200. # 3.3 process add a CD
201. **elif** strChoice == 'a':
202. # 3.3.1 Ask user for new ID, CD Title and Artist
203. # TODO move IO code into function Done
204. IO.user\_inputs
205. # 3.3.2 Add item to the table
206. # TODO move processing code into function Done
207. DataProcessor.create\_table(IO.user\_inputs)
208. **continue**  # start loop back at top.
209. # 3.4 process display current inventory
210. **elif** strChoice == 'i':
211. IO.show\_inventory(lstTbl)
212. **continue**  # start loop back at top.
213. # 3.5 process delete a CD
214. **elif** strChoice == 'd':
215. # 3.5.1 get Userinput for which CD to delete
216. # 3.5.1.1 display Inventory to user
217. IO.show\_inventory(lstTbl)
218. # 3.5.1.2 ask user which ID to remove
219. intIDDel = int(input('Which ID would you like to delete? ').strip())
220. # 3.5.2 search thru table and delete CD
221. # TODO move processing code into function DONE
222. DataProcessor.delete\_data(lstTbl)
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. # TODO move processing code into function DONE
234. FileProcessor.write\_file(strFileName, lstTbl)
235. **else**:
236. input('The inventory was NOT saved to file. Press [ENTER] to return to the menu.')
237. **continue**  # start loop back at top.
238. # 3.7 catch-all should not be possible, as user choice gets vetted in IO, but to be safe:
239. **else**:
240. **print**('General Error')

Listing #3: Final Code for Assignment 06

1. #------------------------------------------#
2. # Title: Assignment06\_Starter.py
3. # Desc: Working with classes and functions.
4. # Change Log: (Who, When, What)
5. # KKauffman, 17 Aug 2020, Created File, wrote code for all TODOs,
6. #debugging 'a' menu functions
7. #KKauffman, 18 Aug 2020, fixed 'a' menu functions, formatting edits,
8. #KKauffman, 19 Aug 2020 formatting edits, added details to docstrings
9. #------------------------------------------#
11. # -- DATA -- #
12. strChoice = '' # User input
13. lstTbl = []  # list of dicts to hold data
14. dicRow = {}  # dict of data row
15. strFileName = 'CDInventory.txt'  # data storage file
16. objFile = None  # file object

19. # -- PROCESSING -- #
20. **class** DataProcessor:
21. """Collect and process user inputs"""
23. @staticmethod
24. **def** create\_table(strID, strTitle, strArtist):
25. """Function to take list of user inputs and put them in a dictionary (dicRow)
27. Arg:
28. Taken from unpacked tuple returned by IO.user\_inputs()
29. strID: User's inputted CD ID
30. strTitle: User's inputted CD title
31. strArtist: User's inputted CD Artist
33. Return:
34. None
36. """
37. intID = int(strID)
38. dicRow = {'ID': intID, 'Title': strTitle, 'Artist': strArtist}
39. lstTbl.append(dicRow)
41. **def** delete\_data(table):
42. """Function to delete CD data from memory
44. Arg: data from lstTbl that is currently in memory
46. Return:
47. None
49. """
50. intRowNr = -1
51. blnCDRemoved = False
52. **for** row **in** lstTbl:
53. intRowNr += 1
54. **if** row['ID'] == intIDDel:
55. **del** lstTbl[intRowNr]
56. blnCDRemoved = True
57. **break**
58. **if** blnCDRemoved:
59. **print**('\nThe CD was removed.\n')
60. **else**:
61. **print**('\nCould not find this CD!\n')
63. **class** FileProcessor:
64. """Processing the data to and from text file"""
66. @staticmethod
67. **def** read\_file(file\_name, table):
68. """Function to manage data ingestion from file to a list of dictionaries
70. Reads the data from file identified by file\_name into a 2D table
71. (list of dicts) table one line in the file represents one dictionary row in table.
73. Args:
74. file\_name (string): name of file used to read the data from
75. table (list of dict): 2D data structure (list of dicts) that holds the data during runtime
77. Returns:
78. None.
79. """
80. table.clear()  # this clears existing data and allows to load data from file
81. objFile = open(file\_name, 'r')
82. **for** line **in** objFile:
83. data = line.strip().split(',')
84. dicRow = {'ID': int(data[0]), 'Title': data[1], 'Artist': data[2]}
85. table.append(dicRow)
86. objFile.close()
88. @staticmethod
89. **def** write\_file(file\_name, table):
90. """Function to write data to file
92. Takes each row from lsttable and separates the items in it by a comma
93. and adds a \n at the end
95. Args:
96. file\_name (string): name of file data is saved to
97. table (lists of dicts): 2D data structure that holds the data
99. Returns: print statement confirming save is complete
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()
108. **return** **print**('\nYour data has been saved.')
110. # -- PRESENTATION (Input/Output) -- #
112. **class** IO:
113. """Handling Input / Output"""
115. @staticmethod
116. **def** print\_menu():
117. """Displays a menu of choices to the user
119. Args:
120. None.
122. Returns:
123. None.
124. """
126. **print**('\nCD Inventory Menu\n\n[l] Load Inventory from file\n[a] Add CD\n[i] Display Current Inventory')
127. **print**('[d] Delete CD from Inventory\n[s] Save Inventory to file\n[x] Exit\n')
129. @staticmethod
130. **def** menu\_choice():
131. """Gets user input for menu selection
133. Args:
134. None.
136. Returns:
137. choice (string): a lower case sting of the users input out of the choices l, a, i, d, s or x
139. """
140. choice = ' '
141. **while** choice **not** **in** ['l', 'a', 'i', 'd', 's', 'x']:
142. choice = input('Which operation would you like to perform? [l, a, i, d, s or x]: ').lower().strip()
143. **print**()  # Add extra space for layout
144. **return** choice
146. @staticmethod
147. **def** show\_inventory(table):
148. """Displays current inventory table

151. Args:
152. table (list of dict): 2D data structure (list of dicts) that holds the data during runtime.
154. Returns:
155. None.
157. """
158. **print**()
159. **print**('======= The Current Inventory: =======')
160. **print**('ID\tCD Title (by: Artist)\n')
161. **for** row **in** table:
162. **print**('{}\t{} (by:{})'.format(\*row.values()))
163. **print**('======================================')
164. **print**()
166. @staticmethod
167. **def** user\_inputs():
168. """Function to gather the user's inputs for CD ID, CD Title, and CD Artist
170. Arg: none
172. Return: a tuple of the three user inputs (entryID, entryTitle, entryArtist)
174. """
175. entryID = input('Enter ID: ').strip()
176. entryTitle = input('What is the CD\'s title? ').strip()
177. entryArtist = input('What is the Artist\'s name? ').strip()
178. **return** (entryID, entryTitle, entryArtist)

181. # 1. When program starts, read in the currently saved Inventory
183. **print**('\nWelcome to your CD Inventory!')
185. #Need to ensure CDInvetory.txt is creaed before running this program
186. FileProcessor.read\_file(strFileName, lstTbl)

189. # 2. start main loop
191. **while** True:
192. # 2.1 Display Menu to user and get choice
193. IO.print\_menu()
194. strChoice = IO.menu\_choice()
196. # 3. Process menu selection
198. # 3.1 process exit first
199. **if** strChoice == 'x':
200. **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\n')
206. **if** strYesNo.lower() == 'yes':
207. **print**('\nreloading...')
208. FileProcessor.read\_file(strFileName, lstTbl)
209. IO.show\_inventory(lstTbl)
210. **else**:
211. input('\ncanceling... Inventory data NOT reloaded. Press [ENTER] to continue to the menu.\n')
212. IO.show\_inventory(lstTbl)
213. **continue**  # start loop back at top.
215. # 3.3 process add a CD
216. **elif** strChoice == 'a':
218. # 3.3.1 Ask user for new ID, CD Title and Artist
219. strID, strTitle, strArtist = IO.user\_inputs() #Assigned return to variables and unpacked this tuple
221. # 3.3.2 Add item to the table
222. DataProcessor.create\_table(strID, strTitle, strArtist) #Arguments are unpacked tuple from IO.user\_inputs()
223. **continue**  # start loop back at top.
225. # 3.4 process display current inventory
226. **elif** strChoice == 'i':
227. IO.show\_inventory(lstTbl)
228. **continue**  # start loop back at top.
230. # 3.5 process delete a CD
231. **elif** strChoice == 'd':
233. # 3.5.1 get Userinput for which CD to delete
234. # 3.5.1.1 display Inventory to user
235. IO.show\_inventory(lstTbl)
237. # 3.5.1.2 ask user which ID to remove
238. intIDDel = int(input('Which ID would you like to delete? ').strip())
240. # 3.5.2 search thru table and delete CD
241. DataProcessor.delete\_data(lstTbl)
242. IO.show\_inventory(lstTbl)
243. **continue**  # start loop back at top.
245. # 3.6 process save inventory to file
246. **elif** strChoice == 's':
248. # 3.6.1 Display current inventory and ask user for confirmation to save
249. IO.show\_inventory(lstTbl)
250. strYesNo = input('Save this inventory to file? [y/n] ').strip().lower()
252. # 3.6.2 Process choice
253. **if** strYesNo == 'y':
255. # 3.6.2.1 save data
256. FileProcessor.write\_file(strFileName, lstTbl)
258. **else**:
259. input('The inventory was NOT saved to file. Press [ENTER] to return to the menu.')
260. **continue**  # start loop back at top.
262. # 3.7 catch-all should not be possible, as user choice gets vetted in IO, but to be safe:
263. **else**:
264. **print**('General Error')

1. (Dawson, 2010) Dawson, M. (2010). *Python programming for the absolute beginner* (3rd ed.). Australia: Course Technology PTR, a part of Cengage Learning.’ [↑](#footnote-ref-1)