**Name: Srirupa Guha**

**Date**: August 19, 2020

**Course**: IT FDN 110: Introduction to Programming (Python)

**Assignment**: 6

**Title**: Knowledge Document – Functions, Global and Local Variables, Classes

**Introduction** –

This week we learned about functions, the various arguments one can pass to a function for processing within the function, as well as return values. We also learned about global and local variables, as well as classes

**Functions**

Functions are groups of code that can be accessed and used by other code and reduce repetition of code in the program. An example could be a calculated result of two numbers. Instead of embedding code to calculate some result in the main program, a separate function is defined with the “def” keyword which contains the code. And this function can be used over and over again by other parts of the main code. Functions can either accept arguments, such as: “def add\_function(intA, intB):”, or not accept any arguments at all. So the programmer using this function can code as follows:

num1 = 2

num2 = 4

result = add\_function(num1, num2)

Default values can be added to the arguments (num1 and num2 in this example) in case there is no user input for the arguments getting into the function. In that case, the first line of the function definition would be, for example: “def add\_function(intA=1, intB=2):”

Return values could be of any type, like string, integer float, Boolean, etc. Usually, if multiple values are returned, the default is a tuple which will need to be unpacked for further processing later on. And it is not necessary to return any value also, such as a function displaying announcements, for example. The learnpython.org [Functions](https://www.learnpython.org/en/Functions) [[1]](#footnote-1) site is very useful in explaining the general anatomy of a function, how to write one, call one, pass arguments to get return values, etc. And to understand the benefit of functions in code, this video demonstrates the use of functions in calculations: [Python Functions](https://www.youtube.com/watch?v=_ypAw_pCOt8&feature=youtu.be) [[2]](#footnote-2)

**Global and Local Variables**

We also learned about global and local. Local variables are local only to the function. They don’t have any meaning to code outside of the function. Usually some arguments are passed to functions for processing, and there could be other variables inside the functions that aid the processing, but those variables are not defined or used outside the functions. Global variables, on the other hand, don’t just belong to the parent code, but can also be used by functions if those functions have the “global” keyword for those variables. Usually, for simplicity’s sake, it is a good practice to keep variables exclusive to functions (i.e., local), and then pass the return value(s) from the functions to the main code for further processing.

**Classes**

Classes are object that contain similar grouping of functions. There could a class for input/output processes that contain the appropriate functions, or another class for calculations that could contain functions that only perform calculations. We will learn more about classes in a later module.

Ultimately, I put all this knowledge to test and wrote and executed my Python program:

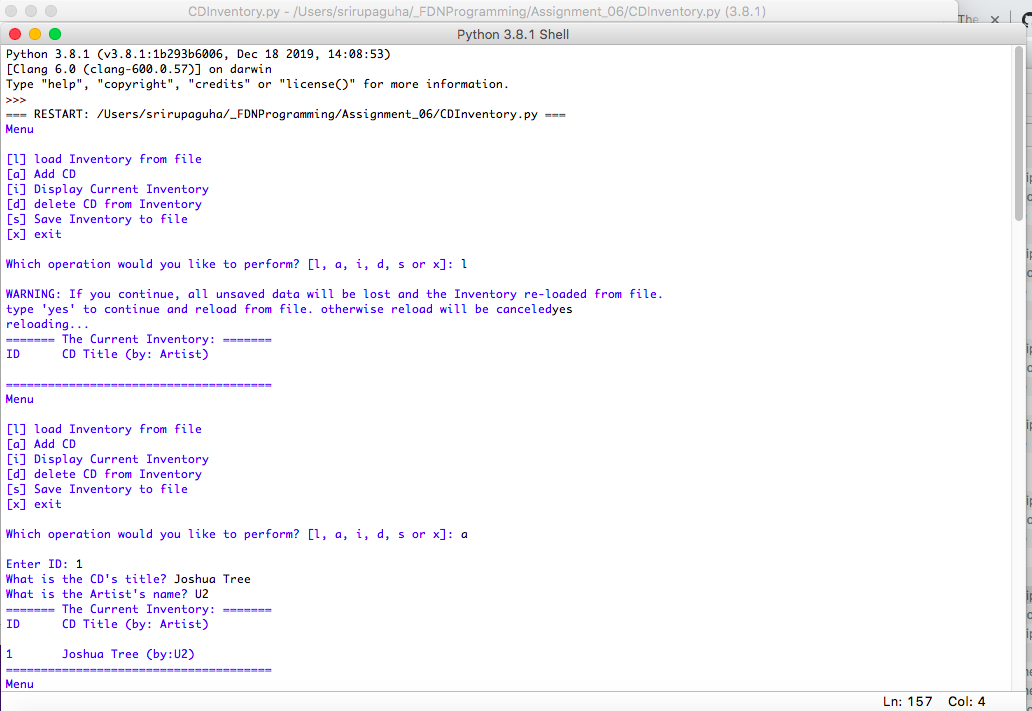


Figure Python program - Adding records

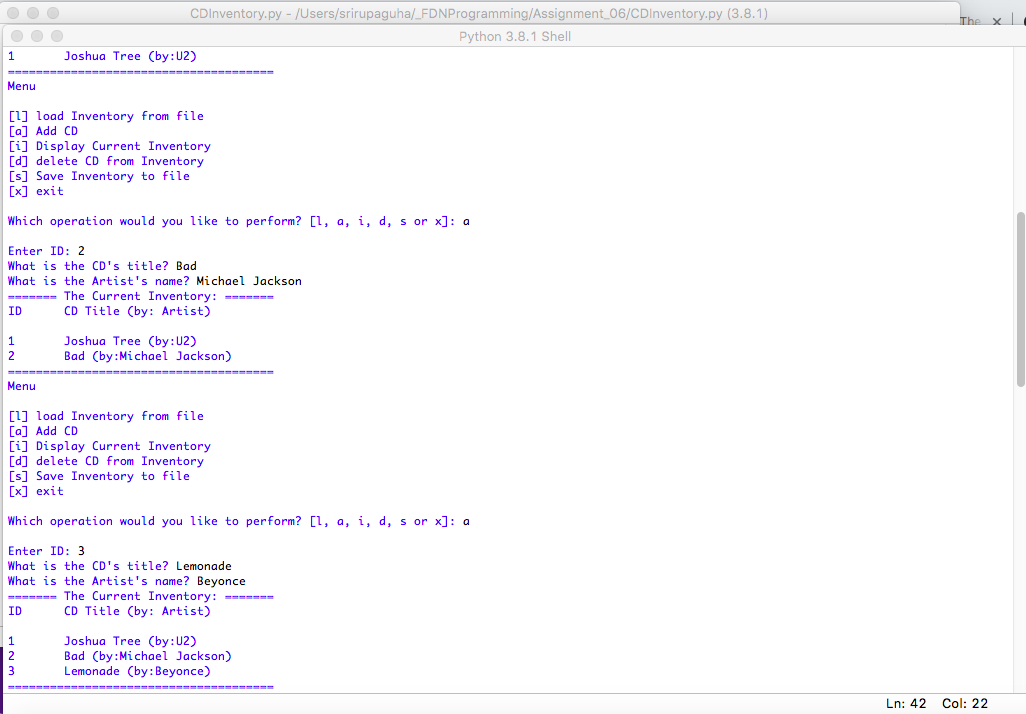


Figure Python program - adding records cont.

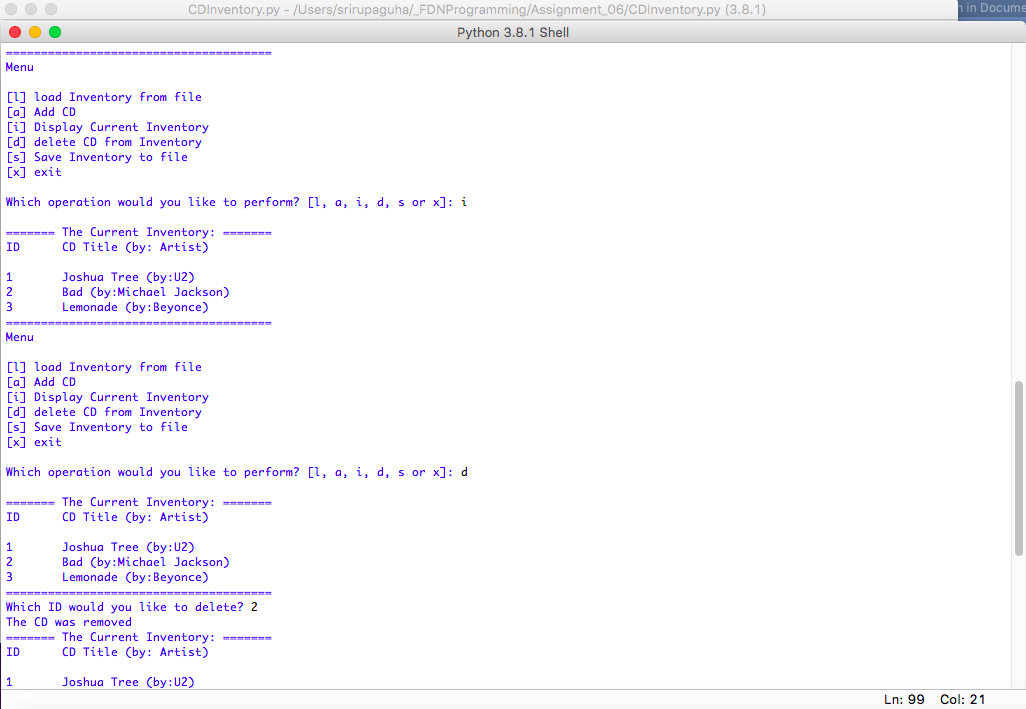


Figure - Displaying records added and deleting one

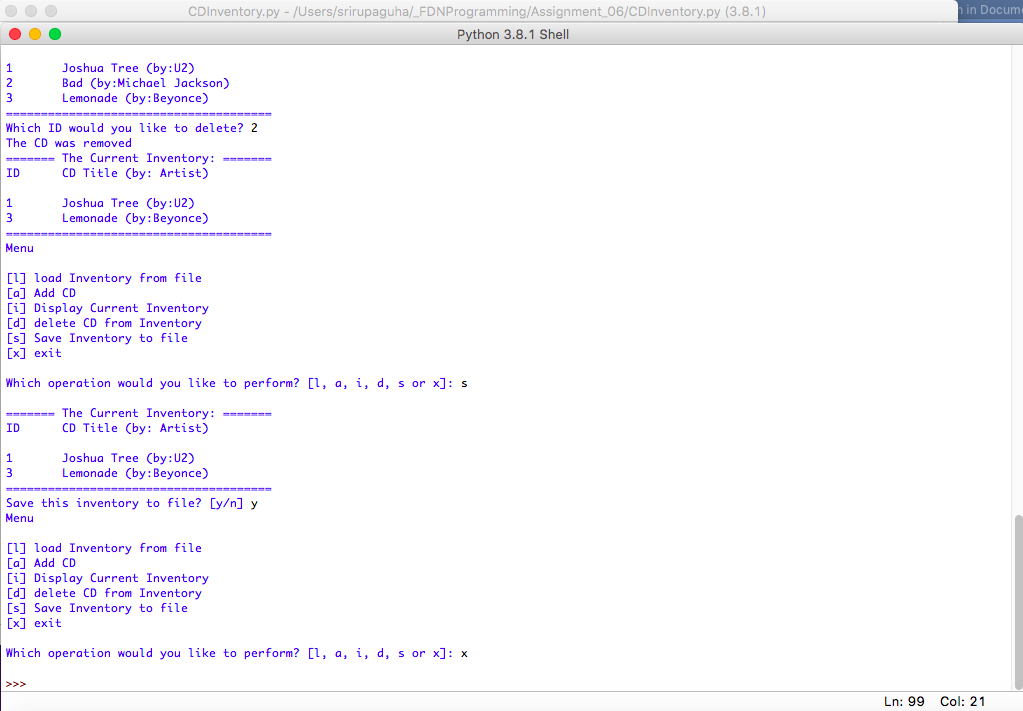


Figure Saving and Exiting

**Summary**

This week we learned more about lists and tuples, as well as a new variable type called the dictionary, as well as using GitHub (creating an account, setting it up, and finally using it for uploading files).

**Appendix**

The Python assignment code for this week:

1. #------------------------------------------#
2. # Title: CDInventory.py
3. # Desc: Working with classes and functions.
4. # Change Log: (Who, When, What)
5. # Rupa Guha, 2020-Aug-16, Created File
6. #------------------------------------------#
8. # -- DATA -- #
9. strChoice = '' # User input
10. lstTbl = []  # list of lists to hold data
11. dicRow = {}  # list 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
19. **def** input\_data\_process(idCd, cdTitle, cdArtist):
20. """Function to add user input data to table
22. Reads the data from file identified by file\_name into a 2D table
23. (list of dicts) table one line in the file represents one dictionary row in table.
25. Args:
26. The ID, Title and Artist newly input by the user
28. Returns:
29. None.
30. """
31. intID = int(idCd)
32. dicRow = {'ID': intID, 'Title': cdTitle, 'Artist': cdArtist}
33. lstTbl.append(dicRow)
35. **def** delete\_row(rowId):
36. """Function to delete row from the inventory
38. Args:
39. The ID of the row intended to be deleted
41. Returns:
42. None.
43. """
44. intRowNr = -1
45. blnCDRemoved = False
46. **for** row **in** lstTbl:
47. intRowNr += 1
48. **if** row['ID'] == rowId:
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!')
56. #pass

59. **class** FileProcessor:
60. """Processing the data to and from text file"""
62. @staticmethod
63. **def** read\_file(file\_name, table):
64. """Function to manage data ingestion from file to a list of dictionaries
66. Reads the data from file identified by file\_name into a 2D table
67. (list of dicts) table one line in the file represents one dictionary row in table.
69. Args:
70. file\_name (string): name of file used to read the data from
71. table (list of dict): 2D data structure (list of dicts) that holds the data during runtime
73. Returns:
74. None.
75. """
76. table.clear()  # this clears existing data and allows to load data from file
77. objFile = open(file\_name, 'r')
78. **try**:
79. **for** line **in** objFile:
80. data = line.strip().split(',')
81. dicRow = {'ID': int(data[0]), 'Title': data[1], 'Artist': data[2]}
82. table.append(dicRow)
83. **except**:
84. ("Something happened here - maybe there is nothing in the file yet?")
85. objFile.close()
87. @staticmethod
88. **def** write\_file(file\_name, table):
89. """Function to manage data from file to a list of dictionaries
91. Writes the data from a 2D table (list of dicts) into a long string and saved into a text file.
93. Args:
94. file\_name (string): name of file used to write the data to
95. table (list of dict): 2D data structure (list of dicts) that holds the data during runtime
97. Returns:
98. None.
99. """
101. # ADDED - TODO Add code here
102. new\_line = ""
103. objFile = None
105. **for** row **in** table:
106. **print**(row)
107. **for** item **in** row.values():
108. new\_line = new\_line + str(item) + ","
109. new\_line = new\_line[:-1] + "\n"
111. objFile = open(strFileName, "w")
112. objFile.write(new\_line)
113. objFile.close()
114. **print**("Data saved!")
116. **def** save\_file(file\_name, table):
117. """Function to save the text file
119. Writes the data from a 2D table (list of dicts) into a long string and saved into a text file.
121. Args:
122. file\_name (string): name of file used to write the data to
123. table (list of dict): 2D data structure (list of dicts) that holds the data during runtime
125. Returns:
126. None.
127. """
128. objFile = open(file\_name, 'w')
129. **for** row **in** table:
130. lstValues = list(row.values())
131. lstValues[0] = str(lstValues[0])
132. objFile.write(','.join(lstValues) + '\n')
133. objFile.close()
134. #pass

137. # -- PRESENTATION (Input/Output) -- #
139. **class** IO:
140. """Handling Input / Output"""
142. @staticmethod
143. **def** print\_menu():
144. """Displays a menu of choices to the user
146. Args:
147. None.
149. Returns:
150. None.
151. """
153. **print**('Menu\n\n[l] load Inventory from file\n[a] Add CD\n[i] Display Current Inventory')
154. **print**('[d] delete CD from Inventory\n[s] Save Inventory to file\n[x] exit\n')
156. @staticmethod
157. **def** menu\_choice():
158. """Gets user input for menu selection
160. Args:
161. None.
163. Returns:
164. choice (string): a lower case sting of the users input out of the choices l, a, i, d, s or x
166. """
167. choice = ' '
168. **while** choice **not** **in** ['l', 'a', 'i', 'd', 's', 'x']:
169. choice = input('Which operation would you like to perform? [l, a, i, d, s or x]: ').lower().strip()
170. **print**()  # Add extra space for layout
171. **return** choice
173. @staticmethod
174. **def** show\_inventory(table):
175. """Displays current inventory table

178. Args:
179. table (list of dict): 2D data structure (list of dicts) that holds the data during runtime.
181. Returns:
182. None.
184. """
185. **print**('======= The Current Inventory: =======')
186. **print**('ID\tCD Title (by: Artist)\n')
187. **for** row **in** table:
188. **print**('{}\t{} (by:{})'.format(\*row.values()))
189. **print**('======================================')
191. # TODO add I/O functions as needed
192. **def** ask\_user\_data():
193. """Asks for user data
195. Args: None
196. Returns: The ID, the CD Title and the Artist of the title
197. """
198. ID = input('Enter ID: ').strip()
199. Title = input('What is the CD\'s title? ').strip()
200. Artist = input('What is the Artist\'s name? ').strip()
201. **return** ID, Title, Artist

204. # 1. When program starts, read in the currently saved Inventory
205. FileProcessor.read\_file(strFileName, lstTbl)
207. # 2. start main loop
208. **while** True:
209. # 2.1 Display Menu to user and get choice
210. IO.print\_menu()
211. strChoice = IO.menu\_choice()
213. # 3. Process menu selection
215. # 3.1 process exit first
216. **if** strChoice == 'x':
217. **break**
219. # 3.2 process load inventory
220. **if** strChoice == 'l':
221. **print**('WARNING: If you continue, all unsaved data will be lost and the Inventory re-loaded from file.')
222. strYesNo = input('type \'yes\' to continue and reload from file. otherwise reload will be canceled')
223. **if** strYesNo.lower() == 'yes':
224. **print**('reloading...')
225. FileProcessor.read\_file(strFileName, lstTbl)
226. IO.show\_inventory(lstTbl)
227. **else**:
228. input('canceling... Inventory data NOT reloaded. Press [ENTER] to continue to the menu.')
229. IO.show\_inventory(lstTbl)
230. **continue**  # start loop back at top.
232. # 3.3 process add a CD
233. **elif** strChoice == 'a':
234. # 3.3.1 Ask user for new ID, CD Title and Artist
235. # ADDED TODO move IO code into function
236. strID, strTitle, stArtist = IO.ask\_user\_data()
238. # 3.3.2 Add item to the table
239. # ADDED TODO move processing code into function
240. DataProcessor.input\_data\_process(strID, strTitle, stArtist)
241. IO.show\_inventory(lstTbl)
242. **continue**  # start loop back at top.
244. # 3.4 process display current inventory
245. **elif** strChoice == 'i':
246. IO.show\_inventory(lstTbl)
247. **continue**  # start loop back at top.
249. # 3.5 process delete a CD
250. **elif** strChoice == 'd':
251. # 3.5.1 get Userinput for which CD to delete
252. # 3.5.1.1 display Inventory to user
253. IO.show\_inventory(lstTbl)
254. # 3.5.1.2 ask user which ID to remove
255. intIDDel = int(input('Which ID would you like to delete? ').strip())
256. # 3.5.2 search thru table and delete CD
257. # ADDED TODO move processing code into function
258. DataProcessor.delete\_row(intIDDel)
260. IO.show\_inventory(lstTbl)
261. **continue**  # start loop back at top.
263. # 3.6 process save inventory to file
264. **elif** strChoice == 's':
265. # 3.6.1 Display current inventory and ask user for confirmation to save
266. IO.show\_inventory(lstTbl)
267. strYesNo = input('Save this inventory to file? [y/n] ').strip().lower()
268. # 3.6.2 Process choice
269. **if** strYesNo == 'y':
270. # 3.6.2.1 save data
271. # ADDED TODO move processing code into function
272. FileProcessor.save\_file(strFileName, lstTbl)
273. **else**:
274. input('The inventory was NOT saved to file. Press [ENTER] to return to the menu.')
275. **continue**  # start loop back at top.
277. # 3.7 catch-all should not be possible, as user choice gets vetted in IO, but to be save:
278. **else**:
279. **print**('General Error')

1. <https://www.learnpython.org/en/Functions> Retrieved Aug 16, 2020 [↑](#footnote-ref-1)
2. <https://www.youtube.com/watch?v=_ypAw_pCOt8&feature=youtu.be> Retrieved Aug 16, 2020 [↑](#footnote-ref-2)