Brandon Wayward

2020-Aug-18

IT FDN 110 B SU 20

Assignment 06

Continued CDInventory.py- Functions, Classes, Docstrings

Introduction

In this lesson we learned about functions and the variety of items that are involved in their scope. We went over Parameters, working with arguments, Tuples in regard to functions, local vs global variables, and Docstrings.

# Functions

We’ll start with Parameters with this lesson regarding functions. Functions allow the optional use of parameters. These parameters allow you to pass in values to your function for processing. Parameters are usually called arguments, but you’ll see both terms used. There isn’t a limit on how many parameters you can pass through, but you need to ensure you’re concise and only use what is necessary.

Once the arguments are called and a functions return value can be used either instantly in a print function or assigned to a variable. Once captured to a variable, it can be used many times.

Functions can return values ranging from None, a single value, or multiple values. None is returned when the return statement has no attributes, or there is no return statement in the function. When returning multiple values, you’ll want to gather the information into a collection, and return that collection. To access that data, you’ll need to unpack it, which in some instances python can do implicitly. Using tuples [[1]](#footnote-1)is the best way I’ve currently seen to do this.

# Arguments

Arguments allow a function to perform different actions or return situationally specific results. Types of arguments can be Positional or Names. When calling a function you can include the name of the parameters (Declaring them in the function) and fill it with the argument. Otherwise, not naming them uses the arguments and assigns them in sequence. It’s possible to mix positional and named arguments with a few caveats.

In some instances you may need default parameters that will be used in substitution when no other value for that parameter is used in the call. A point to remember here is that default values will cause the function to fall back to default behavior, which may not be expected.

# Overloaded Functions

This concept[[2]](#footnote-2) was kind of strange and not directly called out in the assignment itself so I don’t feel like I grasped it. Basically, you can create the same function with different arguments and datatypes so that different calls can be used for different aspects of your code. As I’ve not ran into a functional example I’m not too sure how to expand on this.

# The ‘None’ Keyword

The None keyword is a special DataType in python with only one value: None. None is used to indicate the absence of values and can also be used as a parameter value. The none keyword is most useful for checking the behavior of functions when used as a default argument.

# Local vs. Global Variables and Variable Scope

Depending on where your variable is, it will have different values. Values within a function are local to that function, and variables outside all functions are global functions. You can access global variables within a function if that variable is not defined within the function. When a variable exists as a global and is directly defined in a function this is called a shadowed variable and can cause issues with your code. Don’t shadow variables.

# Docstrings

Docstrings[[3]](#footnote-3) are a way to add documentation to your function keywords and are activated using triple quotes. Within docstrings, best practice is to call out what the function does, what arguments it needs, and what the return results are. While I understood the basecoat of this lesson, I don’t believe I fully grasped what was needed in my own docstrings and feel this may be something that comes with time and more expertise in the python world. Docstrings are useful because wherever you call your function, someone can hover over the function keywords in your code and get information on what that function is doing.

# Summary

In summary, we learned how functions work, how a collection of functions (a class) works, and how docstrings allow us to document our code for future reference and use. I didn’t run into many actual issues within the code that weren’t fix by taking a second look at it, which excites me because it means I’m learning and improving.

# Appendix

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

CODE:

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. # BWayward, 2020-Aug-15 Created write\_file and remove\_inventory function, added DocStrings to both
7. # BWayward, 2020-Aug-16 Cleaned up write to file and Delete codebase, attempted to create add\_inventory function(Failed)
8. # BWayward, 2020-Aug-17 Revisited add inventory function. added IO.addCD and DataProcessor.Write2Mem
9. # BWayward, 2020-Aug-18 After class, added file check to ensure text file exists.
10. #------------------------------------------#
11. **import** os.path
13. # -- DATA -- #
15. strChoice = '' # User input
16. lstTbl = []  # list of lists to hold data
17. dicRow = {}  # list of data row
18. strFileName = 'CDInventory.txt'  # data storage file
19. objFile = None  # file object
20. strID = None
21. strTitle = None
22. stArtist = None
23. intID = None
25. # -- PROCESSING -- #
26. **class** DataProcessor:
27. # TODO add functions for processing here
28. **def** remove\_inventory():
29. """ Function to manage removal of items from table in memory
31. Args:
32. None
34. Returns:
35. prints 'The CD was removed', and displays current inveotry with item deleted
36. """
37. intRowNr = -1
38. blnCDRemoved = False
39. **for** row **in** lstTbl:
40. intRowNr += 1
41. **if** row['ID'] == intIDDel:
42. **del** lstTbl[intRowNr]
43. blnCDRemoved = True
44. **break**
45. **if** blnCDRemoved:
46. **print**('The CD was removed')
47. **else**:
48. **print**('Could not find this CD!')
50. **def** Write2Mem(userInput, dictionary, table):
51. """ Processing user input into list of dicts
53. Args:
54. userInput: Conversion of strID from string to intenger.
55. dictionary: dictionary where the user input data is stored
56. table: Table where dictionary is written
57. Returns:
58. None """
59. intID = int(strID)
60. dicRow = {'ID': intID, 'Title': strTitle, 'Artist': stArtist}
61. lstTbl.append(dicRow)

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, 'r')
83. **for** line **in** objFile:
84. data = line.strip().split(',')
85. dicRow = {'ID': int(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 to database.
93. Writes the data to a specified file identified by file\_name and 2d table.
95. Args:
96. file\_name (string): name of file used to read the data from
97. table (list of dict): 2D data structure tha holds data during runtime
99. Returns:
100. None.
101. """
102. objFile = open(strFileName, 'a')
103. **for** row **in** lstTbl:
104. strRow = ''
105. **for** items **in** row.values():
106. strRow += str(items) + ','
107. strRow = strRow[:-1] + '\n'
108. objFile.write(strRow)
109. objFile.close()


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

154. Args:
155. table (list of dict): 2D data structure (list of dicts) that holds the data during runtime.
157. Returns:
158. 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**('======================================')
167. **def** addCD(userInput1, userInput2, userInput3):
168. """ Process user input to add data
170. Args:
171. userInput1: The ID of the CD
172. userInput2: The Title of the CD
173. userInput3: The Artist of the CD
174. Returns: I feel like this should say more than "None"... but not sure what
175. """
177. strID = input('Enter ID: ').strip()
178. strTitle = input('What is the CD\'s title? ').strip()
179. stArtist = input('What is the Artist\'s name? ').strip()
180. **return** strID, strTitle, stArtist
182. # 1. When program starts, read in the currently saved Inventory
183. **if** os.path.isfile("./" + strFileName) != True:
184. txtFile = open(strFileName,"a")
185. txtFile.close()
187. FileProcessor.read\_file(strFileName, lstTbl)
189. # 2. start main loop
190. **while** True:
191. # 2.1 Display Menu to user and get choice
192. IO.print\_menu()
193. strChoice = IO.menu\_choice()
195. # 3. Process menu selection
196. # 3.1 process exit first
197. **if** strChoice == 'x':
198. **break**
199. # 3.2 process load inventory
200. **if** strChoice == 'l':
201. **print**('WARNING: If you continue, all unsaved data will be lost and the Inventory re-loaded from file.')
202. strYesNo = input('type \'yes\' to continue and reload from file. otherwise reload will be canceled')
203. **if** strYesNo.lower() == 'yes':
204. **print**('reloading...')
205. FileProcessor.read\_file(strFileName, lstTbl)
206. IO.show\_inventory(lstTbl)
207. **else**:
208. input('canceling... Inventory data NOT reloaded. Press [ENTER] to continue to the menu.')
209. IO.show\_inventory(lstTbl)
210. **continue**  # start loop back at top.
211. # 3.3 process add a CD
212. **elif** strChoice == 'a':
213. # 3.3.1 Ask user for new ID, CD Title and Artist
214. (strID,strTitle,stArtist) = IO.addCD(strID,strTitle,stArtist)
215. # 3.3.2 Add item to the table
216. DataProcessor.Write2Mem(intID,dicRow,lstTbl)
217. IO.show\_inventory(lstTbl)
218. **continue**  # start loop back at top.
219. # 3.4 process display current inventory
220. **elif** strChoice == 'i':
221. IO.show\_inventory(lstTbl)
222. **continue**  # start loop back at top.
223. # 3.5 process delete a CD
224. **elif** strChoice == 'd':
225. # 3.5.1 get Userinput for which CD to delete
226. # 3.5.1.1 display Inventory to user
227. IO.show\_inventory(lstTbl)
228. # 3.5.1.2 ask user which ID to remove
229. intIDDel = int(input('Which ID would you like to delete? ').strip())
230. # 3.5.2 search thru table and delete CD
231. DataProcessor.remove\_inventory()
232. IO.show\_inventory(lstTbl)
233. **continue**  # start loop back at top.
234. # 3.6 process save inventory to file
235. **elif** strChoice == 's':
236. # 3.6.1 Display current inventory and ask user for confirmation to save
237. IO.show\_inventory(lstTbl)
238. strYesNo = input('Save this inventory to file? [y/n] ').strip().lower()
239. # 3.6.2 Process choice
240. **if** strYesNo == 'y':
241. # 3.6.2.1 save data
242. FileProcessor.write\_file(strFileName,lstTbl)
243. **else**:
244. input('The inventory was NOT saved to file. Press [ENTER] to return to the menu.')
245. **continue**  # start loop back at top.
246. # 3.7 catch-all should not be possible, as user choice gets vetted in IO, but to be safe:
247. **else**:
248. **print**('General Error')

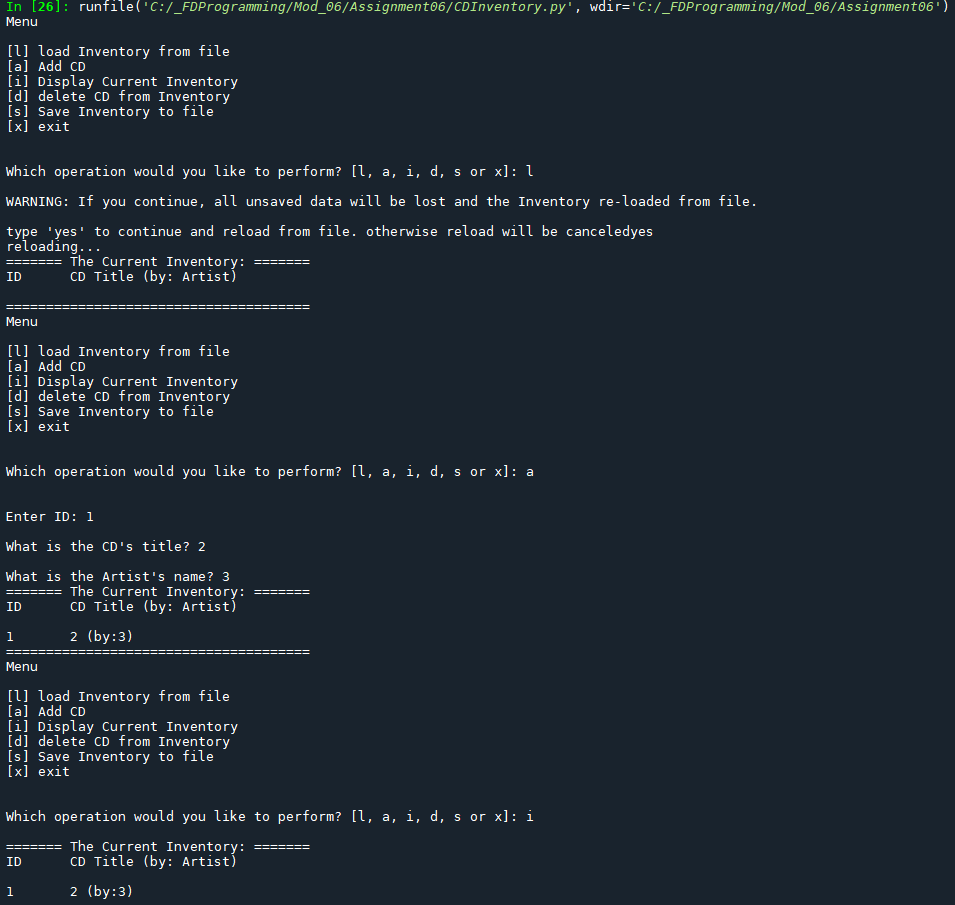


Figure - Running in spyder 1

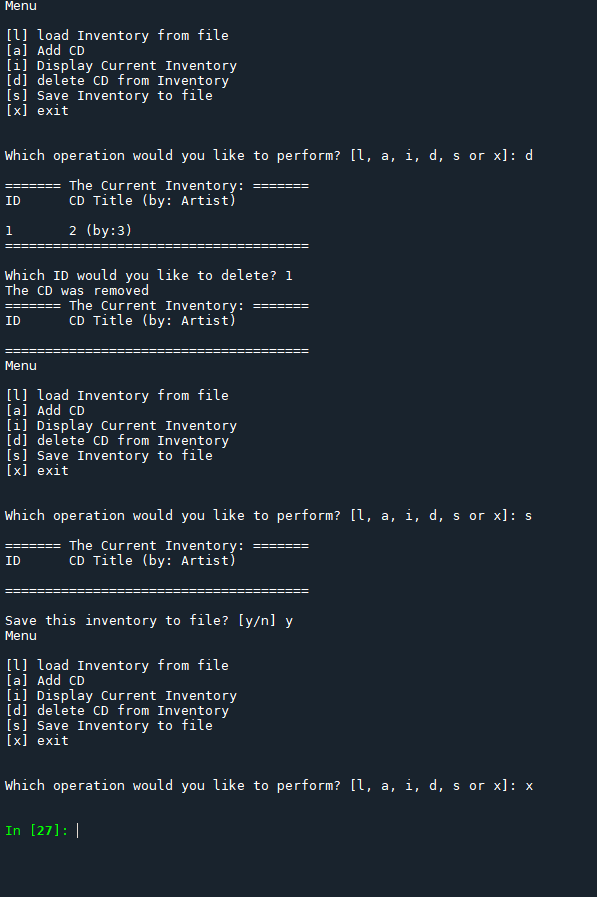


Figure - running in spyder 2

Terminal

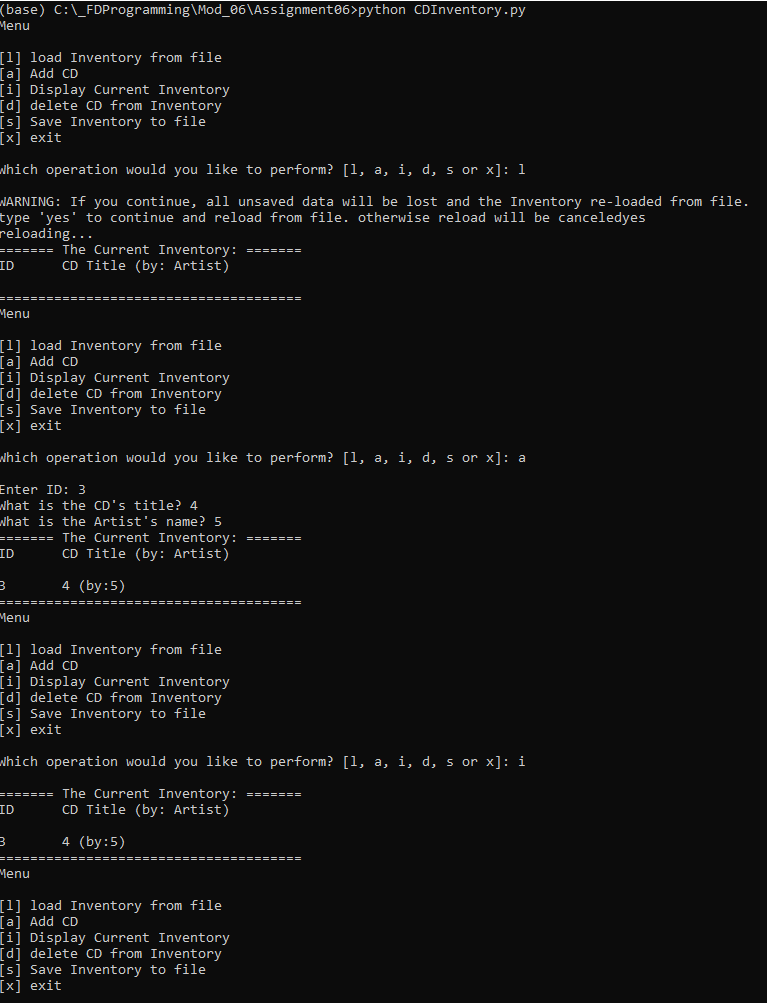


Figure - terminal 1

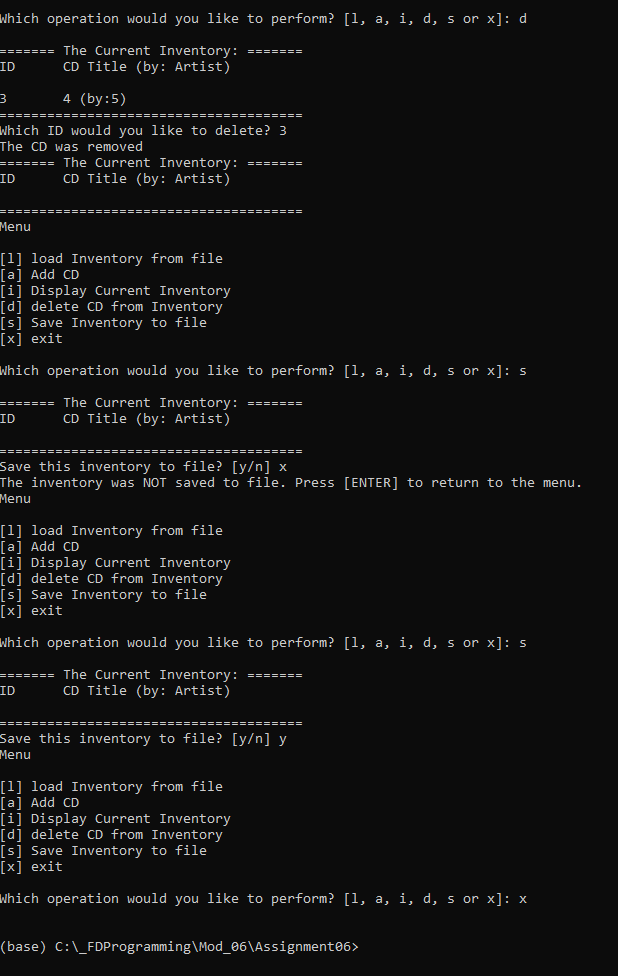


Figure - Terminal 2

1. <https://www.codespeedy.com/how-to-unpack-a-tuple-in-python/#:~:text=Unpacking%20a%20Tuple%20in%20Python%20While%20unpacking%20a,the%20tuple%20and%20put%20them%20into%20normal%20variables.> Accessed 8/15/2020 [↑](#footnote-ref-1)
2. <https://realpython.com/operator-function-overloading/#:~:text=%20Operator%20and%20Function%20Overloading%20in%20Custom%20Python,Many%20of%20the%20special%20methods%20defined...%20More%20> accessed 8/16/2020 [↑](#footnote-ref-2)
3. <https://www.geeksforgeeks.org/python-docstrings/> Accessed 8/18/2020 [↑](#footnote-ref-3)