Tiffany Hou  
2020 March 2  
IT FDN 100B  
Assignment 06

MOdule 6 Submission

# Overview

The goal of this module is to learn about functions and apply the knowledge by adapting an existing script, CDInventory.py.

# Key Concepts

Below is a summary of the key concepts covered by Dirk Biesinger in his Module 6 guide[[1]](#footnote-1).

## Functions

A *function* is a group of a statements that have been assigned a name. This gives the benefit of:

* Helping keep the code more organized by grouping things together logically
* Allowing blocks of code to be re-used since a function can be called at any time after it is defined[[2]](#footnote-2)

Functions should generally be limited to one purpose in scope, making them more flexible and easier to combine with other functions.

Functions can be further organized within a *class[[3]](#footnote-3)*. This works well with the SOC concepts we learned earlier, since we can group functions with related purposes together within a class.

### Arguments and Returns

Since a function can be used in multiple ways, you need to be able to adjust the context and value of the variables being processed. We do this by using *arguments* (also known as *parameters*). We can *pass* arguments into a function and have them be manipulated in there.

Functions can also *return* values, which can be stored within variables to be reused or processed later.

## Variable Scope

Previously, we had worked with *global* *variables* which are available throughout the entire script. With the introduction of functions, we also have access to *local variables* which only exist within the context of a function or class.

It is possible to make a local variable a global variable by using the global keyword. If you want to change the value of a global variable from inside a function, you can also use the keyword[[4]](#footnote-4).

You could use the same variable name for a local and a global variable (*shadowing*). These are different instances with the same name and it is also not recommended.

## DocString

*Docstrings* are a way of providing additional information about a function in within the function itself[[5]](#footnote-5). They can also be used for automation.

A docstring’s format begins with a short description, followed by the *calling conventions* (how the function receives its arguments and returns data)[[6]](#footnote-6).

# CDInventory.Py

Per our discussion in class last week about scope collaborating on someone else’s code, I tried to not make too many changes and stay true to the spirit of the script (keeping the overwriting of existing data when loading from file). Most of the changes made were related to:

1. The TO DOs that asked for functions for:
   1. Adding an entry
   2. Deleting an entry
2. Error processing

The code is set up so that we have 3 main sections:

1. The data processing section, which has 2 classes:
   1. DataProcessor: holds functions for processing data in memory
   2. FileProcessor: holds functions for processing data to and from a save file
2. The presentation section, which has 1 class:
   1. IO: holds functions for processing inputs and outputs
3. The main body of the script, which:
   1. First loads the CDInventory.txt file
   2. Holds the while loop that keeps our script running

## The Code

As usual, the code snippets were formatted using PlanetB’s syntax code highlighter[[7]](#footnote-7).

### Data Processing

The first class in our data processing section is the DataProcessor class, which asks us to move code from the main body of the script to this section.

The first two, add\_entry() and delete\_entry(), had both already existed in the main body of the script and simply had to be moved over, with the proper arguments passed in. I chose to keep the local variable names the same as the global variables passed in for add\_entry() since they were not representing different things in either context. delete\_entry() got a new local variable, which I decided not to make global, since only existed within the context of this function.

A screenshot of a social media post

Description automatically generated

Listing 1 - add\_entry()

A screenshot of a social media post

Description automatically generated

Listing 2 - delete\_entry()

generate\_id() was a function of my own creation. I really dislike giving users the ability to manually create their own PKs since it’s so easy to have user error. This function will generate a CD ID when adding new entries based off the length of the table and checks for unique values. It is not called when loading a table from a saved file since any table in memory is overwritten.

A new local variable, cdID, is used and thus declared at the start of the function.

A screenshot of a social media post

Description automatically generated

Listing 3 - generate\_id()

### File Processing

In the class FileProcessor, we deal with data being loaded or saved to the CDInventory.txt file. One function already existed (read\_file()), while we were asked to move statements from the while to the write\_file() function.

I did decide to add some error processing while doing so (using a try statement). This is so that in case the function runs into an error, the entire script doesn’t crash.

A screenshot of a social media post

Description automatically generated

Listing 4 - write\_file()

### Presentation

The IO class contains the functions responsible for collecting from and displaying to data to the user. 3 functions already existed (print\_menu(), menu\_choice(), and show\_inventory()). I added a fourth function, input\_cd().

input\_cd() is interesting because it calls another function first, generate\_id, which returns an unused CD ID and displays it to the user, and then collects the CD title and artist. It then returns all 3 values back to the main body of the script to be used in the add\_entry() function.

A screenshot of a cell phone

Description automatically generated

Listing 5 - input\_cd()

## OUtput

Screenshots of the outputs for both Spyder and Terminal are available in the Appendix.

# Summary

## Last Thoughts

Our goal was to improve an existing script by moving some of its statements from the main body into functions, while following the best practices and recommendations regarding SoC and collaboration.

Overall, I did not find this to be a terribly challenging assignment. The main question I have is whether global variables (such as lstTbl) should be passed into a function as an argument.

## Github

My documentation and code can be found on GitHub at: <https://github.com/tiffhou/Assignment_06>

# Appendix

## Listing – CDInventory.PY

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. # THou, 2020-Feb-27, added try statement to check for existing save file; added functions for adding entry, deleting entry; updated display
7. # THou, 2020-Feb-29, added auto-generating ID for new CD entries; cleaned up formatting
8. # THou, 2020-Mar-01, added more formatting
9. #------------------------------------------#
11. # -- DATA -- #
12. strChoice = '' # User input
13. lstTbl = []  # list of lists to hold data
14. dicRow = {}  # list of data row
15. strFileName = 'CDInventory.txt'  # data storage file
16. objFile = None  # file object
17. intID = '' #variable to hold a CD ID
18. strTitle = '' #variable to hold a CD title
19. strArtist = '' #variable to hold a CD's artist
21. # -- PROCESSING -- #
22. **class** DataProcessor:
23. """processess data in memory"""
25. @staticmethod
26. **def** add\_entry(intID, strTitle, strArtist):
27. """Adds the CD entry to a dictionary and appends it to the lstTbl.
29. Args:
30. intID (int): CD ID to be added to the dictionary
31. strTitle (str): CD title to be added to the dictionary
32. strArtist (string): CD artist to be added to the dictionary
34. Return:
35. none
36. """
37. dicRow = {'ID': intID, 'Title': strTitle, 'Artist': strArtist}
38. lstTbl.append(dicRow)
39. IO.show\_inventory(lstTbl)
41. @staticmethod
42. **def** delete\_entry(intIDDel):
43. """Deletes a CD entry based on inputted ID (intIDDel).
45. Args:
46. delID (int): the ID inputted by the user for the entry to be deleted
48. Return:
49. 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!')
63. IO.show\_inventory(lstTbl)
65. @staticmethod
66. **def** generate\_id():
67. """Generates a CD ID based on lstTbl length. Checks for and increments until CD ID is a unique value.
69. Args:
70. None
72. Returns:
73. cdID (int): the auto-generated CD ID
74. """
75. cdID = '' #local variable for generating CD ID
76. lenTable = len(lstTbl)
77. **if** lenTable == 0:
78. cdID = 1
79. **else**:
80. cdID = lenTable + 1
81. **while** any(cdID == row['ID'] **for** row **in** lstTbl):
82. cdID = cdID + 1
83. **return** cdID

86. **class** FileProcessor:
87. """Processing the data to and from text file"""
89. @staticmethod
90. **def** read\_file(file\_name, table):
91. """Function to manage data ingestion from file to a list of dictionaries
93. Reads the data from file identified by file\_name into a 2D table
94. (list of dicts) table one line in the file represents one dictionary row in table.
96. Args:
97. file\_name (string): name of file used to read the data from
98. table (list of dict): 2D data structure (list of dicts) that holds the data during runtime
100. Returns:
101. None.
102. """
103. table.clear()  # this clears existing data and allows to load data from file
104. objFile = open(file\_name, 'r')
105. **for** line **in** objFile:
106. data = line.strip().split(',')
107. dicRow = {'ID': int(data[0]), 'Title': data[1], 'Artist': data[2]}
108. table.append(dicRow)
109. objFile.close()
111. @staticmethod
112. **def** write\_file(file\_name, table):
113. """Write the lstTbl in inventory to the text file.
115. Args:
116. None
118. Returns:
119. None
120. """
121. **try**:
122. objFile = open(file\_name, 'w')
123. **for** row **in** table:
124. lstValues = list(row.values())
125. lstValues[0] = str(lstValues[0])
126. objFile.write(','.join(lstValues) + '\n')
127. objFile.close()
128. **except**:
129. **print**('The save attempt was not successful')

132. # -- PRESENTATION (Input/Output) -- #
133. **class** IO:
134. """Handling Input / Output"""
136. @staticmethod
137. **def** print\_menu():
138. """Displays a menu of choices to the user
140. Args:
141. None.
143. Returns:
144. None.
145. """
146. **print**('[[ Menu ]]\n\n[l] load Inventory from file\n[a] Add CD\n[i] Display Current Inventory')
147. **print**('[d] delete CD from Inventory\n[s] Save Inventory to file\n[x] exit\n')
149. @staticmethod
150. **def** menu\_choice():
151. """Gets user input for menu selection.
153. Args:
154. None.
156. Returns:
157. choice (string): a lower case sting of the users input out of the choices l, a, i, d, s or x
158. """
159. choice = ' '
160. **while** choice **not** **in** ['l', 'a', 'i', 'd', 's', 'x']:
161. choice = input('Which operation would you like to perform? [l, a, i, d, s or x]: ').lower().strip()
162. **print**()  # Add extra space for layout
163. **return** choice
165. @staticmethod
166. **def** show\_inventory(table):
167. """Displays current inventory table.
169. Args:
170. table (list of dict): 2D data structure (list of dicts) that holds the data during runtime.
172. Returns:
173. None.
174. """
175. **print**('======= The Current Inventory: =======')
176. **print**('ID\tCD Title (by: Artist)\n')
177. **for** row **in** table:
178. **print**('{}\t{} (by: {})'.format(\*row.values()))
179. **print**('======================================')
181. @staticmethod
182. **def** input\_cd():
183. """Gets the CD entry fields (ID, Title, Artist).
185. Args:
186. None
188. Returns:
189. intID (int): cleaned and validated inputted ID for a CD
190. strTitle (str): cleaned inputted CD title
191. strArtist (string): cleaned inputted CD artist
192. """
193. intID = DataProcessor.generate\_ID()
194. **print**('CD ID: ', intID)
195. strTitle = input('What is the CD\'s title? ').strip()
196. strArtist = input('What is the Artist\'s name? ').strip()
197. **return** intID, strTitle, strArtist

200. # -- MAIN BODY -- #
201. # 1. When program starts, read in the currently saved Inventory
202. **try**:
203. FileProcessor.read\_file(strFileName, lstTbl)
204. **print**('CDInventory.txt loaded.')
205. IO.show\_inventory(lstTbl)
206. **except** FileNotFoundError:
207. **print**('No save file available. Loading menu.\n')
208. **pass**

211. # 2. start main loop
212. **while** True:
213. # 2.1 Display Menu to user and get choice
214. IO.print\_menu()
215. strChoice = IO.menu\_choice()

218. # 3. Process menu selection
219. # 3.1 process exit first
220. **if** strChoice == 'x':
221. **break**

224. # 3.2 process load inventory
225. **if** strChoice == 'l':
226. **print**('WARNING: If you continue, all unsaved data will be lost and the Inventory re-loaded from file.')
227. strYesNo = input('type \'yes\' to continue and reload from file. otherwise reload will be canceled')
228. **if** strYesNo.lower() == 'yes':
229. **print**('reloading...')
230. FileProcessor.read\_file(strFileName, lstTbl)
231. IO.show\_inventory(lstTbl)
232. **else**:
233. input('canceling... Inventory data NOT reloaded. Press [ENTER] to continue to the menu.')
234. IO.show\_inventory(lstTbl)
235. **continue**  # start loop back at top.

238. # 3.3 process add a CD
239. **elif** strChoice == 'a':
240. # 3.3.1 Ask user for new ID, CD Title and Artist
241. intID, strTitle, strArtist = IO.input\_cd()
243. # 3.3.2 Add item to the table
244. DataProcessor.add\_entry(intID, strTitle, strArtist)
245. **continue**  # start loop back at top.

248. # 3.4 process display current inventory
249. **elif** strChoice == 'i':
250. IO.show\_inventory(lstTbl)
251. **continue**  # start loop back at top.

254. # 3.5 process delete a CD
255. **elif** strChoice == 'd':
256. # 3.5.1 get Userinput for which CD to delete
257. # 3.5.1.1 display Inventory to user
258. IO.show\_inventory(lstTbl)
259. # 3.5.1.2 ask user which ID to remove
260. intIDDel = int(input('Which ID would you like to delete? ').strip())
262. # 3.5.2 search thru table and delete CD
263. DataProcessor.delete\_entry(intIDDel)
264. **continue**  # start loop back at top.

267. # 3.6 process save inventory to file
268. **elif** strChoice == 's':
269. # 3.6.1 Display current inventory and ask user for confirmation to save
270. IO.show\_inventory(lstTbl)
271. strYesNo = input('Save this inventory to file? [y/n] ').strip().lower()
273. # 3.6.2 Process choice
274. **if** strYesNo == 'y':
275. # 3.6.2.1 save data
276. FileProcessor.write\_file(strFileName, lstTbl)
277. **else**:
278. input('The inventory was NOT saved to file. Press [ENTER] to return to the menu.')
279. **continue**  # start loop back at top.

282. # 3.7 catch-all should not be possible, as user choice gets vetted in IO, but to be save:
283. **else**:
284. **print**('General Error')

## Output from Spyder

A screenshot of a social media post

Description automatically generated

Spyder Output 1 - Loading from File

A screenshot of a cell phone

Description automatically generated

Spyder Output 2 - Deleting an Entry

A screenshot of a cell phone

Description automatically generated

Spyder Output 3 - Adding an entry

A screenshot of a cell phone

Description automatically generated

Spyder Output 4 - Saving to File

## Output from Terminal

A screenshot of a cell phone

Description automatically generated

Terminal Output 1 - Loading from File

A screenshot of a cell phone

Description automatically generated

Terminal Output 2 - Adding a CD

A screenshot of a cell phone

Description automatically generated

Terminal Output 3 - Deleting a CD

A screenshot of a cell phone

Description automatically generated

Terminal Output 4 - Saving to File

1. Downloaded 2020-Feb-25 from Canvas [↑](#footnote-ref-1)
2. <https://www.w3schools.com/python/python_functions.asp>, accessed 2020-Mar-2 [↑](#footnote-ref-2)
3. <https://docs.python.org/3.7/tutorial/classes.html>, accessed 2020-Mar-2 [↑](#footnote-ref-3)
4. https://www.w3schools.com/python/python\_scope.asp, accessed 2020-Mar-2 [↑](#footnote-ref-4)
5. https://www.geeksforgeeks.org/python-docstrings/, accessed 2020-Mar-2 [↑](#footnote-ref-5)
6. https://en.wikipedia.org/wiki/Calling\_convention, accessed 2020-Mar-2 [↑](#footnote-ref-6)
7. http://www.planetb.ca/syntax-highlight-word, accessed 2020-Mar-01 [↑](#footnote-ref-7)