Andrew Hanson

2020 Jul 31

IT FDN 110 B Su 20

Assignment04

Functions, classes, and docstring

# Intro

This module covered functions, docstring, and classes. Functions were used to implement separation of concerns. The functions of the program were split into data, processing, and presentation. We were given starter code for this assignment and asked to move some of the code into various functions.

# Data

The main data of the program is the CD inventory that is stored as a 2-d list of dictionaries. This data is worked with in memory and stored in a text file. By loading the file into memory, entries can be added and deleted and then the file can be overwritten with the new inventory.

A screen shot of a computer

Description automatically generated

Figure - data loaded from inventory file (Spyder)

# Processing

## [l] load Inventory from file

The function read\_file in the class FileProcessor is responsible for loading data from the inventory file into memory. This allows easier manipulation of the file data. The function requires two arguments: filename and table.

1. objFile = open(strFileName, 'w')
2. **for** row **in** lstTbl:
3. lstValues = list(row.values())
4. lstValues[0] = str(lstValues[0])
5. objFile.write(','.join(lstValues) + '\n')
6. objFile.close()

Listing - code for read\_file function

## [a] Add CD

The function add\_2 in the class DataProcessor is responsible for making the user’s entries accessible by the program. It does this by organizing the user’s entries into a dictionary with keywords related to the purpose of the data. It then adds the dictionary into a list of similar dictionaries of user inputs to comprise the inventory data. The function takes in one argument: userinputs.

1. intID = int(userinputs[0])
2. strTitle = userinputs[1]
3. stArtist = userinputs[2]
4. dicRow = {'ID': intID, 'Title': strTitle, 'Artist': stArtist}
5. lstTbl.append(dicRow)

Listing - code for add\_2 function

## [d] delete CD from Inventory

The function delete\_entry in the class DataProcessor is responsible for removing entries from the inventory table in memory. The function cycles through the dictionaries in the table looking for an entry with a matching ID. If it finds one, it will remove that entry. User feedback is given based on whether a matching ID is found. The inventory table is also displayed before and after the delete function is run to show the user what is available to delete and what the new inventory table looks like. The function takes one argument: table.

1. intRowNr = -1
2. blnCDRemoved = False
3. **for** row **in** table:
4. intRowNr += 1
5. **if** row['ID'] == intIDDel:
6. **del** lstTbl[intRowNr]
7. blnCDRemoved = True
8. **break**
9. **if** blnCDRemoved:
10. **print**('The CD was removed')
11. **else**:
12. **print**('Could not find this CD!')

Listing - code for delete\_entry function

## [s] Save Inventory to file

The function write\_file in the class FileProcessor is responsible for overwriting the inventory file with the table inventory. Overwriting the file allows deleting entries from the file by deleting them from the table inventory which is stored in memory. The function takes one argument: table.

A screenshot of a cell phone

Description automatically generated

Figure - the inventory being saved (console)

1. objFile = open(strFileName, 'w')
2. **for** row **in** lstTbl:
3. lstValues = list(row.values())
4. lstValues[0] = str(lstValues[0])
5. objFile.write(','.join(lstValues) + '\n')
6. objFile.close()

Listing - code for write\_file function

# Presentation

## The menu

The programs user interactivity is handled mostly by the main menu. One function is responsible for displaying the menu choices to the user. Another function is responsible for taking asking for user input and returning their choice. Some parts of the program have localized menus or user choices not handled by the main menu.

### print\_menu

The function print\_menu in the class IO is responsible for making the user aware of their options and how to interact with the program. Placing the menu in a function allows additional functionality to be more easily added and helps with the separation of concerns. The options are: l, a, i, d, s or x.

1. **print**('Menu\n\n[l] load Inventory from file\n[a] Add CD\n[i] Display Current Inventory')
2. **print**('[d] delete CD from Inventory\n[s] Save Inventory to file\n[x] exit\n')

Listing - code for print\_menu function

### menu\_choice

the function menu\_choice in the class IO is responsible for asking for the user’s choice input and returning it to the while loop so that the program can fulfil their desires. This is the main interface by which the user interacts with the program.

1. choice = ' '
2. **while** choice **not** **in** ['l', 'a', 'i', 'd', 's', 'x']:
3. choice = input('Which operation would you like to perform? [l, a, i, d, s or x]: ').lower().strip()
4. **print**()  # Add extra space for layout
5. **return** choice

Listing - code for the menu\_choice function

## [a] Add CD

The function add\_1 in the class IO is responsible for taking user inputs and passing them to the add\_2 function. It asks the user for inputs for the CD’s ID, name, and artist. The function returns a tuple of values: strID, strTitle, and strArtist.

1. strID = input('Enter ID: ').strip()
2. strTitle = input('What is the CD\'s title? ').strip()
3. strArtist = input('What is the Artist\'s name? ').strip()
4. **return** strID,strTitle,strArtist

Listing - code for add\_1 function

## [i] Display Current Inventory

The function show\_inventory in the class IO is responsible for printing the inventory table for the user’s viewing pleasure. This allows the user to know what entries are already on the list. This function takes one argument: table.

1. **print**('======= The Current Inventory: =======')
2. **print**('ID\tCD Title (by: Artist)\n')
3. **for** row **in** table:
4. **print**('{}\t{} (by:{})'.format(\*row.values()))
5. **print**('======================================')

Listing - code for the show\_inventory function

# Challenges

This assignment was mostly a conceptual challenge. It required understanding the functionality of various code blocks within the program so that they may be properly documented and placed into a function under the correct class.

Writing docstring for the new function was a bit of a challenge but the already written docstring for the existing functions provided a good template for the construction of the new docstring.

# Summary

# Questions

## • What is a function?

A function is a block of code indented within a def statement. It can be called with two parentheses().

## • What are parameters?

Parameters are values passed into a function.

## • What are arguments?

arguments are values passed into a function.

## • What is the difference between parameters and arguments?

They are pretty much the same.

## • What are return values?

Return values are the output of a function. If a variable is assigned to a function call then the return values of the function will be saved inside that variable.

## • What is the difference between a global and a local variable?

Global variables are available anywhere inside of the program. Local variables are available within a specific scope usually a specific function or for loop. These are often defined implicitly.

## • What is shadowing?

Shadowing is when you use the same name for a local variable that is already being used by a global variable. It will overwrite the global variable within the scope of the local variable.

## • How do you use functions to organize your code?

Functions allow the position of the code block comprising your function to appear earlier than where it is used. When functions are used extensively this allows all the functions of a program to be grouped by purpose. This helps with organization. Also, docstring is commonly used with functions to further help document code.

## • What is the difference between a function and a class?

Functions are a callable block of code sometimes with implicitly defined variable that are passed in when the function is called. A class allows grouping of functions, data, and variables. They are both helpful for organization, but classes are a higher-level organizational tool.

## • How do functions help you program using the “Separations of Concerns" pattern?

Functions can be used to separate the utility of separate parts of a program. This allows working with the functions on a more individual basis. It can be easier to test a single function that deals with a single problem than dealing with the entire codebase.

# Appendix

## Repository

https://github.com/clichekhfan/Assignment\_06

## 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. # AHanson, 2020-Aug-15, moved code into functions
7. # AHanson, 2020-Aug-15, added docstring to new functions.
8. # AHanson, 2020-Aug-15, added @static method to functions.
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
18. #create file if none exists
19. with open(strFileName, 'a') as objFile:
20. **pass**
22. # -- PROCESSING -- #
23. **class** DataProcessor:
25. @staticmethod
26. **def** add\_2(userinputs):
27. """proccesses user inputs into dictionary entries and puts dictionary in table
28. Appends dictionary of user inputs to LstTbl as global.
30. unpacks tuple of user inputs
31. sorts indiviual user inputs into dictionary entries
32. appends dictionary into table
34. Args:
35. userinputs (tuple): user inputs from IO.add\_1
37. Returns:
38. None.
39. """
40. intID = int(userinputs[0])
41. strTitle = userinputs[1]
42. strArtist = userinputs[2]
43. dicRow = {'ID': intID, 'Title': strTitle, 'Artist': strArtist}
44. lstTbl.append(dicRow)

47. @staticmethod
48. **def** delete\_entry(table):
49. """Deletes an entry from memory.
50. Modifies table to remove selected entry.
52. Args:
53. table (list of dict): 2D data structure (list of dicts) that holds the data during runtime
55. Returns:
56. None.
57. """
58. intRowNr = -1
59. blnCDRemoved = False
60. **for** row **in** table:
61. intRowNr += 1
62. **if** row['ID'] == intIDDel:
63. **del** lstTbl[intRowNr]
64. blnCDRemoved = True
65. **break**
66. **if** blnCDRemoved:
67. **print**('The CD was removed')
68. **else**:
69. **print**('Could not find this CD!')

72. **class** FileProcessor:
73. """Processing the data to and from text file"""
75. @staticmethod
76. **def** read\_file(file\_name, table):
77. """Function to manage data ingestion from file to a list of dictionaries
79. Reads the data from file identified by file\_name into a 2D table
80. (list of dicts) table one line in the file represents one dictionary row in table.
82. Args:
83. file\_name (string): name of file used to read the data from
84. table (list of dict): 2D data structure (list of dicts) that holds the data during runtime
86. Returns:
87. None.
88. """
89. table.clear()  # this clears existing data and allows to load data from file
90. objFile = open(file\_name, 'r')
91. **for** line **in** objFile:
92. data = line.strip().split(',')
93. dicRow = {'ID': int(data[0]), 'Title': data[1], 'Artist': data[2]}
94. table.append(dicRow)
95. objFile.close()
97. @staticmethod
98. **def** write\_file(file\_name, table):
99. """Function writes inventory to file
101. opens the file
102. takes in the inventory table
103. writes each entry in table on seperate line
104. closes file
106. Args:
107. file\_name (string): name of file used to read the data from
108. table (list of dict): 2D data structure (list of dicts) that holds the data during runtime
110. Returns:
111. None.
112. """
113. objFile = open(strFileName, 'w')
114. **for** row **in** lstTbl:
115. lstValues = list(row.values())
116. lstValues[0] = str(lstValues[0])
117. objFile.write(','.join(lstValues) + '\n')
118. objFile.close()

121. # -- PRESENTATION (Input/Output) -- #
123. **class** IO:
124. """Handling Input / Output"""
126. @staticmethod
127. **def** print\_menu():
128. """Displays a menu of choices to the user
130. Args:
131. None.
133. Returns:
134. None.
135. """
137. **print**('Menu\n\n[l] load Inventory from file\n[a] Add CD\n[i] Display Current Inventory')
138. **print**('[d] delete CD from Inventory\n[s] Save Inventory to file\n[x] exit\n')
140. @staticmethod
141. **def** menu\_choice():
142. """Gets user input for menu selection
144. Args:
145. None.
147. Returns:
148. choice (string): a lower case sting of the users input out of the choices l, a, i, d, s or x
150. """
151. choice = ' '
152. **while** choice **not** **in** ['l', 'a', 'i', 'd', 's', 'x']:
153. choice = input('Which operation would you like to perform? [l, a, i, d, s or x]: ').lower().strip()
154. **print**()  # Add extra space for layout
155. **return** choice
157. @staticmethod
158. **def** show\_inventory(table):
159. """Displays current inventory table

162. Args:
163. table (list of dict): 2D data structure (list of dicts) that holds the data during runtime.
165. Returns:
166. None.
168. """
169. **print**('======= The Current Inventory: =======')
170. **print**('ID\tCD Title (by: Artist)\n')
171. **for** row **in** table:
172. **print**('{}\t{} (by:{})'.format(\*row.values()))
173. **print**('======================================')
175. @staticmethod
176. **def** add\_1():
177. """takes initial user input
178. returns the user's inputs as a tuple
180. Args:
181. None.
183. Returns:
184. (strID,strTitle,strArtist)
186. """
187. strID = input('Enter ID: ').strip()
188. strTitle = input('What is the CD\'s title? ').strip()
189. strArtist = input('What is the Artist\'s name? ').strip()
190. **return** strID,strTitle,strArtist
192. # 1. When program starts, read in the currently saved Inventory
193. FileProcessor.read\_file(strFileName, lstTbl)
195. # 2. start main loop
196. **while** True:
197. # 2.1 Display Menu to user and get choice
198. IO.print\_menu()
199. strChoice = IO.menu\_choice()
200. # 3. Process menu selection
201. # 3.1 process exit first
202. **if** strChoice == 'x':
203. **break**
204. # 3.2 process load inventory
205. **if** strChoice == 'l':
206. **print**('WARNING: If you continue, all unsaved data will be lost and the Inventory re-loaded from file.')
207. strYesNo = input('type \'yes\' to continue and reload from file. otherwise reload will be canceled: ')
208. **if** strYesNo.lower() == 'yes':
209. **print**('reloading...')
210. FileProcessor.read\_file(strFileName, lstTbl)
211. IO.show\_inventory(lstTbl)
212. **else**:
213. input('canceling... Inventory data NOT reloaded. Press [ENTER] to continue to the menu.')
214. IO.show\_inventory(lstTbl)
215. **continue**  # start loop back at top.
216. # 3.3 process add a CD
217. **elif** strChoice == 'a':
218. # 3.3.1 Ask user for new ID, CD Title and Artist
219. tplUserInputs = IO.add\_1()
220. # 3.3.2 Add item to the table
221. DataProcessor.add\_2(tplUserInputs)
222. IO.show\_inventory(lstTbl)
223. **continue**  # start loop back at top.
224. # 3.4 process display current inventory
225. **elif** strChoice == 'i':
226. IO.show\_inventory(lstTbl)
227. **continue**  # start loop back at top.
228. # 3.5 process delete a CD
229. **elif** strChoice == 'd':
230. # 3.5.1 get Userinput for which CD to delete
231. # 3.5.1.1 display Inventory to user
232. IO.show\_inventory(lstTbl)
233. # 3.5.1.2 ask user which ID to remove
234. intIDDel = int(input('Which ID would you like to delete? ').strip())
235. # 3.5.2 search thru table and delete CD
236. DataProcessor.delete\_entry(lstTbl)
237. IO.show\_inventory(lstTbl)
238. **continue**  # start loop back at top.
239. # 3.6 process save inventory to file
240. **elif** strChoice == 's':
241. # 3.6.1 Display current inventory and ask user for confirmation to save
242. IO.show\_inventory(lstTbl)
243. strYesNo = input('Save this inventory to file? [y/n] ').strip().lower()
244. # 3.6.2 Process choice
245. **if** strYesNo == 'y':
246. # 3.6.2.1 save data
247. FileProcessor.write\_file(strFileName, lstTbl)
248. **else**:
249. input('The inventory was NOT saved to file. Press [ENTER] to return to the menu.')
250. **continue**  # start loop back at top.
251. # 3.7 catch-all should not be possible, as user choice gets vetted in IO, but to be save:
252. **else**:
253. **print**('General Error')