Laurel Warner

11/21/2021

Foundations of Programming

Assignment06

Functions and Classes

# Introduction

In this module we continued learning about functions and also discovered that Classes exist but not really what they are. (Because we will learn that later, apparently.) Also Doc Strings were introduced as well as the different scopes like Global Scope and Local Scope.

# Functions

A function is a way to group statements together and then make that group available later on in a program, even multiple times via a name that we can choose. Parameters are also called arguments and they are the values that we pass to a function that the function then does some sort of processing on. Parameters and arguments are the same thing and the names are used interchangeably. A return value is any result that the function ends up with after performing the processing task. It can be captured via a print statement or assigned to a variable. You use functions to organize your code by defining them first in either the Processing or Presentation section. If the function has anything to do with getting input from a user to displaying output to a user then it can go in the Presentation (IO) section. Then, when you have the main body of your code you just call the function again by typing it in right the spot where it needs to be called, along with the parameters or arguments that need to be a part of it. Using the “Separation of Concerns” pattern you can group functions according to what sort of thing they do in your code by determining what the purpose of the function is in your code. Is it processing data in some way? Or is it helping to present the data and gather input or show output? That will help you to figure out where to put it.

# Global and Local Scope

When we assign a variable as part of a function, then that variable is only valid within that function. Python won’t recognize it if you try to use it in another part of your program. This is called “scope”, and this kind of variable is a local variable. In a program, when you’re not specifically running a function then you are in the global scope. Any variable that you create outside of a function is a global variable and can be used throughout your program that you write. Shadowing is when you give a variable inside a function the same name as a global variable. Even though the variable names are the same, Python doesn’t recognize them that way. They are still totally different.

# Classes

Classes are a way of grouping functions, variables and constants. A function can be PART of a class. Any set of functions that help to do the same thing can all be in the same class together. In Lab C we grouped the math functions together into 1 class because they all were performing some math together. In our assignment we are grouping functions together in the same class that read and write data from the text file, as an example.

# Writing Code

In this module we were given a copy of some starter code which contained a possible solution to last weeks assignment. Given our new emphasis on Separation of Concerns we needed to move code into the Processing and Presentation sections, and define functions for each task the program should do, whether that be loading the data, adding data, or deleting data. Then we needed to call the function in the main portion of the code in the spot that used to contain the code that was moved and became a function. I was able to move my code and define the functions, and when I was done, I was left with only 1 error at the very end of the code in the final 2 lines, which was a syntax error. One of my classmates thought it might be due to indentation issues, so I changed the indentation, which caused a whole bunch more syntax errors to crop up in the earlier portion of the code. All the syntax errors at that point were “Undefined name” for variables that were defined in the functions. The classmate and I both had the same error, so neither of us could help the other out. So, once again I had to call on my tutor to help. We determined that yes, the errors were due to indentation issues, so we fixed those. And the function that was to gather user input to add a new row needed a new line telling it what to return. With my tutor’s help I determined that my other errors when I tried to run the code were mostly caused by me pushing the wrong buttons. I guess that will be a case for error handling in the next module. Here are the screenshots of the code working in Spyder:

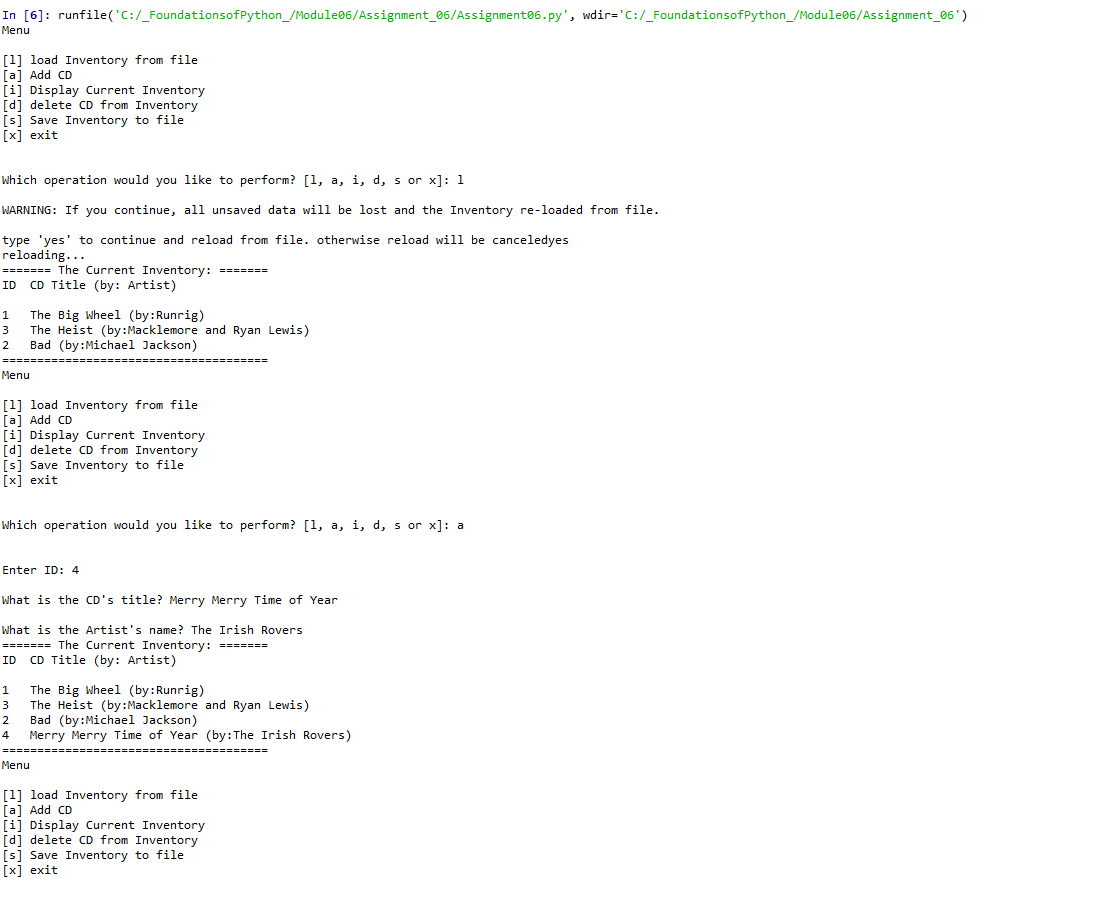


Figure 1

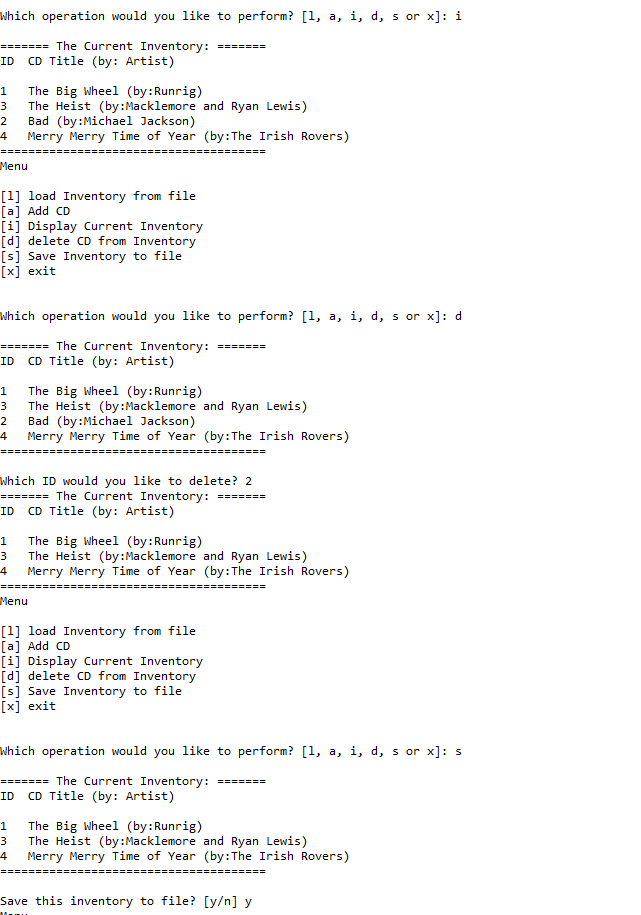


Figure 2

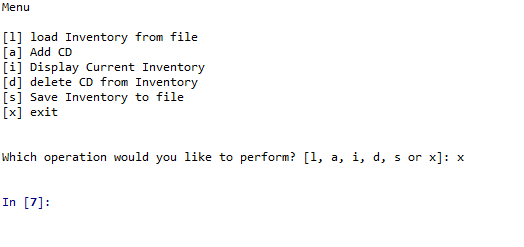


Figure 3

Here are the screenshots of the code working in the terminal window:

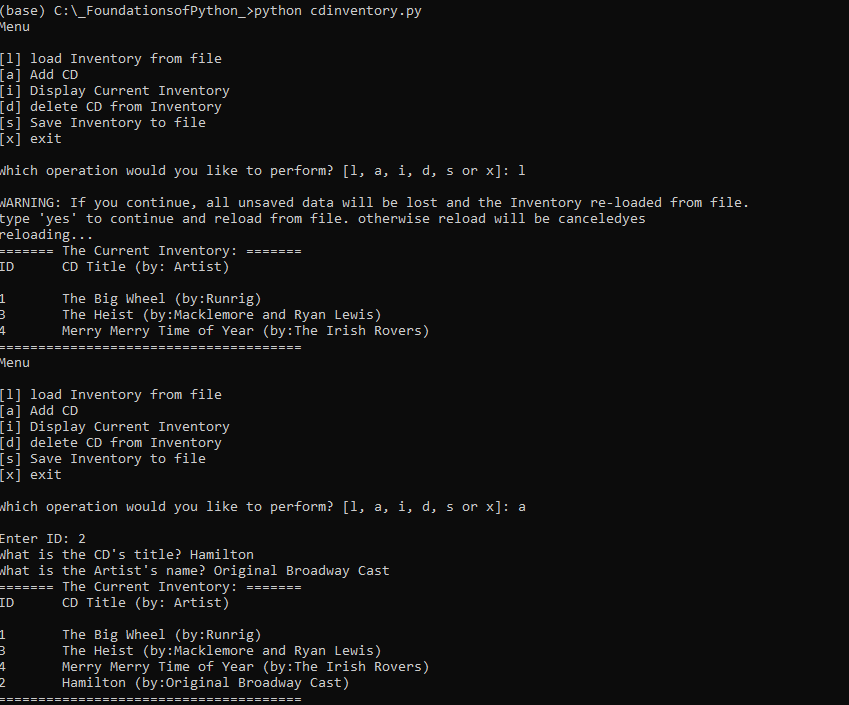


Figure 4

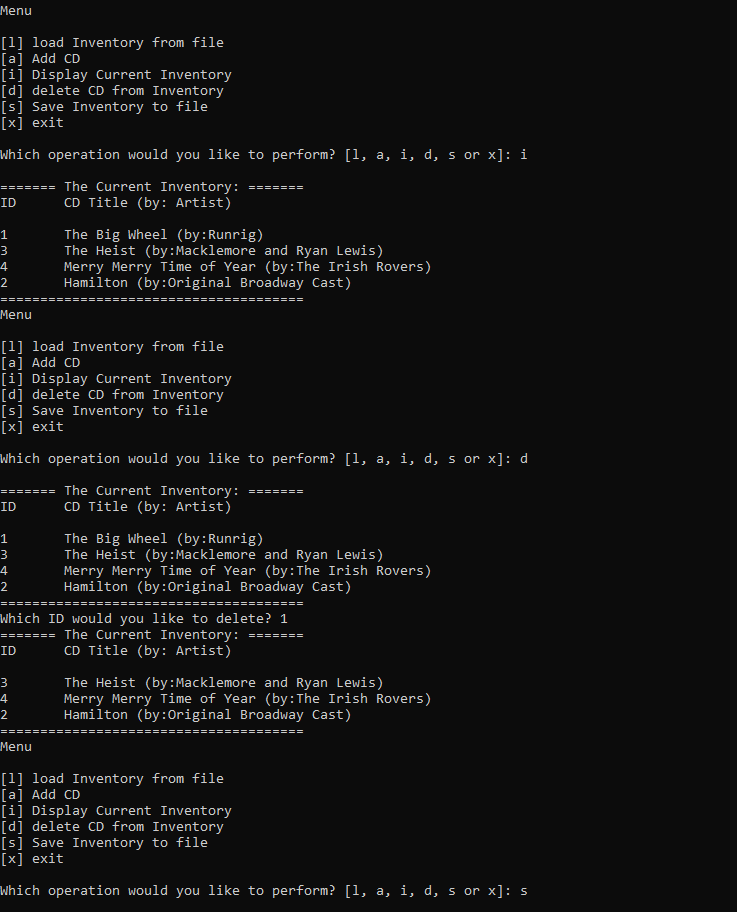


Figure 5

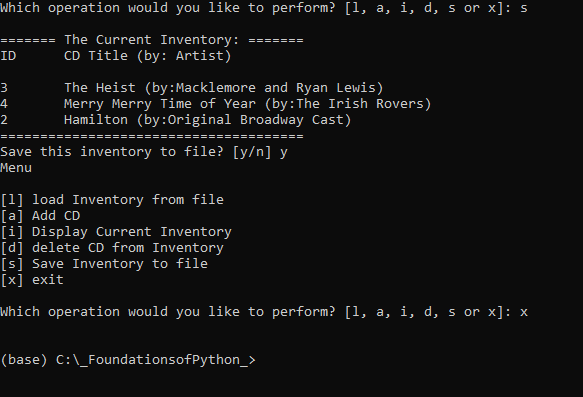


Figure 6

This code was also uploaded to my GitHub Repository: https://github.com/lellenn/Assignment\_06

# Summary

I learned about functions and classes and the differences between them. I learned about functions and parameters and arguments that get fed to a function so that it can do it’s processing work. I learned about what a return value is, what default values are, and I learned about global and local scopes and global and local variables. But I’m still fuzzy on that. I also learned how to define functions in my code and also how to create docstrings.

# Appendix

1. *#------------------------------------------#*
2. *# Title: Assignment06.py*
3. *# Desc: Working with classes and functions.*
4. *# Change Log: (Who, When, What)*
5. ***# DBiesinger, 2030-Jan-01, Created File***
6. *# LWarner, 2021-Nov-20, Added Code and Reorganized Code*
7. *#------------------------------------------#*
9. *# -- DATA -- #*
10. **strChoice = '' *# User input***
11. lstTbl = [] *# list of lists to hold data*
12. dicRow = {} *# list of data row*
13. strFileName = 'CDInventory.txt' *# data storage file*
14. objFile = None *# file object*

17. *# -- PROCESSING -- #*
18. **class** DataProcessor:
19. """ Functions to add, delete, and save data in the table"""
21. @staticmethod
22. **def** add\_row(dicRow):
23. """Function to append a row to the table
24. Gathers the user input for ID, Title and Artist and appends it to the table in memory.
26. Args:
27. row: the row of ID, Title, and Artist
28. table: the 2D data structure that holds the data during runtime
30. **Returns:**
31. Shows inventory
32. """
33. dicRow["ID"] = int(dicRow["ID"])
34. lstTbl.append(dicRow)

37. @staticmethod
38. **def** delete\_row(intRowNr, lstTbl):
39. """ Function to delete a row from the table if the user wants.
41. Args:
42. intRowNr: the ID number for the row to be deleted
43. lstTbl: the table with the data
45. **Returns:**
46. The information showing whether or not the row was successfully deleted.
47. """
48. intRowNr = -1
49. blnCDRemoved = False
50. **for row in lstTbl:**
51. intRowNr += 1
52. **if** row['ID'] == intIDDel:
53. **del** lstTbl[intRowNr]
54. blnCDRemoved = True
55. **break**
56. **if** blnCDRemoved:
57. **print**('The CD was removed')
58. **else**:
59. **print**('Could not find this CD!')

62. **class** FileProcessor:
63. """Processing the data to and from text file"""
65. **@staticmethod**
66. **def** read\_file(file\_name, table):
67. """Function to manage data ingestion from file to a list of dictionaries
69. Reads the data from file identified by file\_name into a 2D table
70. **(list of dicts) table one line in the file represents one dictionary row in table.**
72. Args:
73. file\_name (string): name of file used to read the data from
74. table (list of dict): 2D data structure (list of dicts) that holds the data during runtime
76. Returns:
77. None.
78. """
79. table.clear() *# this clears existing data and allows to load data from file*
80. **objFile = open(file\_name, 'r')**
81. **for** line **in** objFile:
82. data = line.strip().split(',')
83. dicRow = {'ID': int(data[0]), 'Title': data[1], 'Artist': data[2]}
84. table.append(dicRow)
85. **objFile.close()**
87. @staticmethod
88. **def** write\_file(strFileName, lstTbl): *# this allows the data to be written to the .txt file*
89. """ Function to allow data to be written to the .txt file that has previously been created
91. Args:
92. strFileName (string): name of file used to write the data to
93. lstTbl (list of dict): 2D data structure (list of dicts) that holds the data during runtime
95. **Returns:**
96. None.
97. """
98. objFile = open(strFileName, 'w')
99. **for** row **in** lstTbl:
100. **lstValues = list(row.values())**
101. lstValues[0] = str(lstValues[0])
102. objFile.write(','.join(lstValues) + '**\n**')
103. objFile.close()
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**\t**CD Title (by: Artist)**\n**')
155. **for row in table:**
156. **print**('{}**\t**{} (by:{})'.format(\*row.values()))
157. **print**('======================================')
159. @staticmethod
160. **def enter\_row():**
161. """ Gets user input for adding a new CD to the inventory
163. Args:
164. strID: The ID of the CD to be added
165. **strTitle: the Title of the CD to be added**
166. strArtist: the Artist of the CD to be added
168. Returns:
169. The ID, Title, and Artist info for each CD rendered as a string.
170. **"""**
171. strID = input('Enter ID: ').strip()
172. strTitle = input('What is the CD**\'**s title? ').strip()
173. strArtist = input('What is the Artist**\'**s name? ').strip()
174. **return** { 'ID': strID, 'Title': strTitle, 'Artist': strArtist }

177. *# 1. When program starts, read in the currently saved Inventory*
178. FileProcessor.read\_file(strFileName, lstTbl)
180. ***# 2. start main loop***
181. **while** True:
182. *# 2.1 Display Menu to user and get choice*
183. IO.print\_menu()
184. strChoice = IO.menu\_choice()
186. *# 3. Process menu selection*
187. *# 3.1 process exit first*
188. **if** strChoice == 'x':
189. **break**
190. ***# 3.2 process load inventory***
191. **if** strChoice == 'l':
192. **print**('WARNING: If you continue, all unsaved data will be lost and the Inventory re-loaded from file.')
193. strYesNo = input('type **\'**yes**\'** to continue and reload from file. otherwise reload will be canceled')
194. **if** strYesNo.lower() == 'yes':
195. **print('reloading...')**
196. FileProcessor.read\_file(strFileName, lstTbl)
197. IO.show\_inventory(lstTbl)
198. **else**:
199. input('canceling... Inventory data NOT reloaded. Press [ENTER] to continue to the menu.')
200. **IO.show\_inventory(lstTbl)**
201. **continue** *# start loop back at top.*
202. *# 3.3 process add a CD*
203. **elif** strChoice == 'a':
204. *# 3.3.1 Ask user for new ID, CD Title and Artist*
205. **DataProcessor.add\_row(IO.enter\_row())**
206. IO.show\_inventory(lstTbl)
207. **continue** *# start loop back at top.*
208. *# 3.4 process display current inventory*
209. **elif** strChoice == 'i':
210. **IO.show\_inventory(lstTbl)**
211. **continue** *# start loop back at top.*
212. *# 3.5 process delete a CD*
213. **elif** strChoice == 'd':
214. *# 3.5.1 get Userinput for which CD to delete*
215. ***# 3.5.1.1 display Inventory to user***
216. IO.show\_inventory(lstTbl)
217. *# 3.5.1.2 ask user which ID to remove*
218. intIDDel = int(input('Which ID would you like to delete? ').strip())
219. *# 3.5.2 search thru table and delete CD*
220. **DataProcessor.delete\_row(intIDDel, lstTbl)**
221. IO.show\_inventory(lstTbl)
222. **continue** *# start loop back at top.*
223. *# 3.6 process save inventory to file*
224. **elif** strChoice == 's':
225. ***# 3.6.1 Display current inventory and ask user for confirmation to save***
226. IO.show\_inventory(lstTbl)
227. strYesNo = input('Save this inventory to file? [y/n] ').strip().lower()
228. *# 3.6.2 Process choice*
229. **if** strYesNo == 'y':
230. ***# 3.6.2.1 save data***
231. FileProcessor.write\_file(strFileName, lstTbl)
232. **else**:
233. input('The inventory was NOT saved to file. Press [ENTER] to return to the menu.')
234. **continue** *# start loop back at top.*
235. ***# 3.7 catch-all should not be possible, as user choice gets vetted in IO, but to be save:***
236. **else**:
237. **print**('General Error')