Britta Anson

August 19, 2020

Foundations of Programming: Python

Assignment 06

Modified CD Inventory Script:

Now with functions!

# Introduction

This week we were introduced to functions and updated the CD Inventory script to move a lot of the code within the main menu while loop out to functions in a variety of classes. This allowed us to organize the code by the principle of separation of concerns.

# General background

## What is a function?

A function is a group of code statements that are defined by a specific name that can be used to later “call” the full group of code statements.

## Parameters and Arguments

The difference between parameters and arguments was confusing to me at first because the module document said “Generally, parameters are called arguments.”[[1]](#footnote-1) This made it sound like they were synonymous so I was unsure how to differentiate them.

I turned to Google and found an explanation on MDN Web Docs that helped to clarify. It’s glossary entry for “parameter” explained it as follows:

“A **parameter** is a named variable passed into a function. Parameter variables are used to import **arguments** into functions. Note the difference between parameters and arguments:

* Function parameters are the names listed in the function's definition.
* Function arguments are the real values passed to the function.
* Parameters are initialized to the values of the arguments supplied.”[[2]](#footnote-2)

So, using Listing 4 from the Module 06 document as an example:

A screenshot of a cell phone

Description automatically generated

Figure 1 - screenshot of Listing 4 in Module 06 Doc, p. 4

In this example, value1 and value2 are parameters, while strData and strAnswer are arguments in the function call.

## What are return values?

Return values are the output of a function. They can be saved as a variable or printed, etc.

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

Global variables are defined outside of functions and can be accessed both within and outside of functions without needing to be defined in a given function. Local variables are defined within a function block and hold true within that block but not outside of it.[[3]](#footnote-3)

## What is shadowing?

Shadowing a global variable is defining a new variable within a function, but giving it the same name as a global variable. This creates two separate variables with the same name, rather than changing the value of the global variable. This is not recommended.

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

Creating functions not only allows for separation of concerns, but reduces instances of repetitive code, thereby making the code more organized and concise. By moving certain code sequences into functions we can store them sorted by concern rather than by the order in which they will be called within a program.

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

A function is one specific block of code that does a specific thing and is assigned a specific variable. A class is a group of functions that share some kind of theme, like Data Processing functions or Input/Output functions. To call a function, you must call both the class and individual function name.

# Modifying the CD Inventory script

To start modifying the inventory script, the first thing I did was try to run the starter script as it was to see how it worked. I immediately encountered an error due to the fact that I didn’t have a pre-existing CDInventory.txt file in the Assignment 6 folder. Apparently the first thing the program does is try to read data from a file, so I need to add a way for the script to create an empty file if one doesn’t already exist.

A screenshot of a cell phone

Description automatically generated

Figure 2 - Error due to no 'CDInventory.txt' file in directory

## Avoiding Error: No such file

My first thought it to complete the TODO to create the function FileProcessor.write\_file first, and then to add code at the very start of the program where it starts by attempting to read data from a file, to first check if the file exists before calling the read\_file function, and if not, to create an empty file by calling the write\_file function.

The first step was to create the write\_file function.

### Moving the while loop code to function

The starter code gave us an elif block for the ‘s’ (save to file) menu selection that looked like this:

A screenshot of a cell phone

Description automatically generated

Listing 1 - lines 177-190 of 'Assignment06\_Starter.py' – showing code for writing to file within the main while loop for user input/output.

To clear the TODO item, I moved the code for writing to the file to the write\_file function within the FileProcessor class in the data processing section of the script, and then replace the code in the elif statement with a simple function call.

A screenshot of a cell phone

Description automatically generated

Listing 2 – From Assignment06.py: defined write\_file function using code moved from the main loop elif statement for 's' (save to file) menu choice

A screenshot of a cell phone

Description automatically generated

Listing 3 – From Assignment06.py: elif statement calling the write\_file function defined above

### Checking if the data file exists

I Googled “how to check if file exists in python” and found a suggestion online to use a try-except block to accomplish this.[[4]](#footnote-4)

The starter code starts off the program by reading data from the file CDInventory.txt.



Listing 4 - First step in running the starter code

I added a try-except block using the new write\_file function that creates an empty text file and names is “CDInventory.txt” in the event that such a file doesn’t already exist.

A close up of a screen

Description automatically generated

Listing 5 - try-except block in Assignment06.py preventing program crash if no file exists

## General Formatting Clean-up

I noticed the starter code had a few sections where there were spaces or newlines missing (in my opinion), so I added those.

For example, these little things bug me:

A close up of text on a black background

Description automatically generated

Figure 3 - Original script running. Notice space after "by: " in header but not in inventory rows; and lack of space between inventory and menu.

I added a newline before the string ‘Menu’ in the print\_menu function, and added the extra space in the show\_inventory function before the {} placeholder.

I find this version much easier to read:

A close up of text on a black background

Description automatically generated

Figure 4 - Updated display with added spacing

It might seem trivial but adding white space for the eye really helps me read any document, including code or output, and makes testing much easier.

## Adding user input code to IO function

At first I didn’t realize how to move all three input statements into one function so I created three different functions: new\_id(), new\_title(), and new\_artist().

A screenshot of text

Description automatically generated

Listing 6 - First attempt to move user input to function

But in class on Tuesday I was reminded about using multiple values to return a tuple, so I attempted to combine all three into one function “get\_data()” instead, that returns a tuple “tplUserData.”

I was able to replace three functions with just one:

A screenshot of a cell phone

Description automatically generated

Listing 7 - New function to collect all user input in one place

Now in the program, I can just call the IO.get\_data() function in the “add” menu routine, and set the tuple it returns equal to the variables strID, strTitle, and strArtist in the interface, before passing those variables as arguments into the DataProcessor.add\_data() function to add them as a dictionary row to the 2D Table.

A screenshot of a cell phone

Description automatically generated

Listing 8 - Using an IO class function and DataProcessor class function to execute the "add" routine in the menu

Whereas I had previously tried using my original IO.new\_ID(), IO.new\_title(), and IO.new\_artist() functions within the DataProcessor.add\_data() function, I learned in class not to call functions from other classes within a function (in case you want to copy a full class to another script, it should stand alone). I learned that I needed to define the variable within the interface as the output of one function before feeding those variable values into another function in another class (as I do in the example above).

## Adding a few features

I added a few features I had had in my previous versions of the script. Namely, the check to make sure a user really wants to exit when they select ‘x’ (requesting a y/n input), and a verification printout of the new data added in the ‘a’ menu choice and a y/n input to confirm that the user wants the new data added to the table. I did this to give an out if there was a data entry error. This is a different variation of the “press ‘c’ to clear” option I added for every data input line in the old program. I couldn’t figure out how to add this feature within a function that is outside of the while loop – the “continue” keyword that I used previously, wouldn’t work in the function. So, this was a workaround.

Unfortunately, I didn’t find a solution for how to ensure that the ID entered by the user is not already existing in the table. This version of the script does therefore allow for duplicate ID entries (which I don’t like).

# RUNNING the FINAL code

## Running in Spyder

## A screenshot of a cell phone Description automatically generated

Figure 5 - CDInventory.py running in Spyder part 1

A close up of text on a black background

Description automatically generated

Figure 6 - CDInventory.py running in Spyder part 2

A screenshot of text

Description automatically generated

Figure 7 - CDInventory.py running in Spyder part 3

A close up of text on a black background

Description automatically generated

Figure 8 - CDInventory.py running in Spyder part 4

The resulting text file CDInventory.txt:

A screenshot of a cell phone

Description automatically generated

Figure 9 - Text file CDInventory.txt after running script in Spyder

## Running in the Terminal

A screenshot of a cell phone

Description automatically generated

Figure 10 - CDInventory.py running in Terminal part 1

A screenshot of text

Description automatically generated

Figure 11 - CDInventory running in Terminal part 2

A screenshot of text

Description automatically generated

Figure 12 - CDInventory.py running in Terminal part 3

A screenshot of a cell phone

Description automatically generated

Figure 13 - CDInventory.txt after running script in Terminal

# Posted to github

The assignment documents are posted to my GitHub repository “Assignment\_06” at: <https://github.com/bricolagerie/Assignment_06>.

# Summary

I learned that organizing code in functions makes it a lot cleaner to sort code by concern, but I ran into more challenges to create the same functionality I was able to achieve in my previous version where all the code was within the main while loop.

# Appendix

## Listing CDInventory.py

1. #------------------------------------------#
2. # Title: CDInventory.py
3. # Desc: Assignment 06 - Working with classes and functions.
4. # Change Log: (Who, When, What)
5. # BAnson, 2020-Aug-16, Created File
6. # BAnson, 2020-Aug-16, moved write to file code to write\_file function
7. #   Added try-except block to create new file if none exists
8. #   Cleaned up formatting in I/O presentation
9. #   Added data processing functions add\_data and del\_id
10. #   Added IO functions to collect user inputs of new\_id, new\_title, new\_artist
11. # BAnson, 2020-Aug-18, removed calls to IO class from DataProcessor class in add\_data()
12. #   Consolidated 3 user input IO functions into 1 get\_data()
13. #   Added:
14. #       Review of entered data and y/n choice to add to table
15. #       "are you sure?" to exit sequence
16. #------------------------------------------#
18. # -- DATA -------------------------------------------------------------------- #
20. strChoice = '' # User input
21. lstTbl = []  # list of lists to hold data
22. dicRow = {}  # list of data row
23. strFileName = 'CDInventory.txt'  # data storage file
24. objFile = None  # file object


28. # -- PROCESSING FUNCTIONS ---------------------------------------------------- #
30. **class** DataProcessor:
31. # Added functions for processing
32. @staticmethod
33. **def** add\_data(strID, strTitle, strArtist):
34. """Appends Inventory with values assigned to strID, strTitle, and strArtist
36. Args:
37. strID = user input ID
38. strTitle = user input Title
39. strArtist = user input Artist
41. Returns:
42. Integer version of strID (intID)
43. A dictionary row (dictRow) containing intID, strTitle, strArtist
45. """
47. intID = int(strID)
48. dicRow = {'ID': intID, 'Title': strTitle, 'Artist': strArtist}
49. lstTbl.append(dicRow)
51. @staticmethod
52. **def** del\_id():
53. """Deletes CD inventory item based on ID
55. Args:
56. None
58. Returns:
59. None
61. """
62. intRowNr = -1
63. blnCDRemoved = False
64. **for** row **in** lstTbl:
65. intRowNr += 1
66. **if** row['ID'] == intIDDel:
67. **del** lstTbl[intRowNr]
68. blnCDRemoved = True
69. **break**
70. **if** blnCDRemoved:
71. **print**('The CD was removed')
72. **else**:
73. **print**('Could not find this CD!')

76. **class** FileProcessor:
77. """Processing the data to and from text file"""
79. @staticmethod
80. **def** read\_file(file\_name, table):
81. """Function to manage data ingestion from file to a list of dictionaries
83. Reads the data from file identified by file\_name into a 2D table
84. (list of dicts) table one line in the file represents one dictionary row in table.
86. Args:
87. file\_name (string): name of file used to read the data from
88. table (list of dict): 2D data structure (list of dicts) that holds the data during runtime
90. Returns:
91. None.
92. """
93. table.clear()  # this clears existing data and allows to load data from file
94. objFile = open(file\_name, 'r')
95. **for** line **in** objFile:
96. data = line.strip().split(',')
97. dicRow = {'ID': int(data[0]), 'Title': data[1], 'Artist': data[2]}
98. table.append(dicRow)
99. objFile.close()
101. @staticmethod
102. **def** write\_file(file\_name, table):
103. """Writes data in memory table to a text file (or creates empty text file if none exists
105. Args:
106. file\_name = name of the file to open and write data to (or create)
107. table (list of dict): 2D data structure (list of dicts) that holds the data during runtime.
109. Returns:
110. Text file containing data of current table in memory
111. """
113. # Moved this code from while loop menu 's'
114. objFile = open(strFileName, 'w')
115. **for** row **in** lstTbl:
116. lstValues = list(row.values())
117. lstValues[0] = str(lstValues[0])
118. objFile.write(','.join(lstValues) + '\n')
119. objFile.close()

122. # -- PRESENTATION (Input/Output) FUNCTIONS ----------------------------------- #
124. **class** IO:
125. """Handling Input / Output"""
127. @staticmethod
128. **def** print\_menu():
129. """Displays a menu of choices to the user
131. Args:
132. None.
134. Returns:
135. None.
136. """
138. **print**('\nMenu\n\n[l] load Inventory from file\n[a] Add CD\n[i] Display Current Inventory')
139. **print**('[d] delete CD from Inventory\n[s] Save Inventory to file\n[x] exit\n')
141. @staticmethod
142. **def** menu\_choice():
143. """Gets user input for menu selectioni
145. Args:
146. None.
148. Returns:
149. choice (string): a lower case sting of the users input out of the choices l, a, i, d, s or x
151. """
152. choice = ' '
153. **while** choice **not** **in** ['l', 'a', 'i', 'd', 's', 'x']:
154. choice = input('Which operation would you like to perform? [l, a, i, d, s or x]: ').lower().strip()
155. **print**()  # Add extra space for layout
156. **return** choice
158. @staticmethod
159. **def** show\_inventory(table):
160. """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**('\n======= 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. # added I/O functions:
177. @staticmethod
178. **def** get\_data():
179. """Collects three pieces of data from user: ID, title, and artist
181. Args:
182. None
184. Returns:
185. Tuple containing values corresponding to ID (strID), title (strTitle), and artist (strArtist)
187. """
189. strID = input('Enter ID: ').strip()
190. strTitle = input('What is the CD\'s title? ').strip()
191. strArtist = input('What is the Artist\'s name? ').strip()
192. tplUserData = (strID, strTitle, strArtist)
193. **return** tplUserData

196. # -- MAIN PROGRAM ------------------------------------------------------------ #

199. # 1. When program starts, read in the currently saved Inventory, or create empty file
200. **try**:
201. FileProcessor.read\_file(strFileName, lstTbl)
202. **except**:
203. FileProcessor.write\_file(strFileName, lstTbl)

206. # 2. start main loop
207. **while** True:
208. # 2.1 Display Menu to user and get choice
209. IO.print\_menu()
210. strChoice = IO.menu\_choice()
212. # 3. Process menu selection
213. # 3.1 process exit first
214. **if** strChoice == 'x':
215. choice = input('Are you sure you want to exit? [y/n]: ') # prevent exiting accidentally
216. **if** choice.lower() == 'y':
217. **break**
218. **else**:
219. **continue**
221. # 3.2 process load inventory
222. **if** strChoice == 'l':
223. **print**('WARNING: If you continue, all unsaved data will be lost and the Inventory re-loaded from file.')
224. strYesNo = input('type \'yes\' to continue and reload from file. otherwise reload will be canceled')
225. **if** strYesNo.lower() == 'yes':
226. **print**('reloading...')
227. FileProcessor.read\_file(strFileName, lstTbl)
228. IO.show\_inventory(lstTbl)
229. **else**:
230. input('canceling... Inventory data NOT reloaded. Press [ENTER] to continue to the menu.')
231. IO.show\_inventory(lstTbl)
232. **continue**  # start loop back at top.
234. # 3.3 process add a CD
235. **elif** strChoice == 'a':
236. # 3.3.1 Ask user for new ID, CD Title and Artist
237. # Moved IO code into function
238. tplUserData = IO.get\_data()
239. # Review entry
240. **print**('\nYou entered: ', tplUserData)
241. choice = input('Continue to add data to table? [y/n]: ')
242. **if** choice.lower() == 'y':
243. strID, strTitle, strArtist = tplUserData
244. # 3.3.2 Add item to the table
245. DataProcessor.add\_data(strID, strTitle, strArtist)
246. IO.show\_inventory(lstTbl)
247. **else**:
248. **print**('Data not added to table.')
249. **continue**  # start loop back at top.
251. # 3.4 process display current inventory
252. **elif** strChoice == 'i':
253. IO.show\_inventory(lstTbl)
254. **continue**  # start loop back at top.
256. # 3.5 process delete a CD
257. **elif** strChoice == 'd':
258. # 3.5.1 get Userinput for which CD to delete
259. # 3.5.1.1 display Inventory to user
260. IO.show\_inventory(lstTbl)
261. # 3.5.1.2 ask user which ID to remove
262. intIDDel = int(input('Which ID would you like to delete? ').strip())
263. # 3.5.2 search thru table and delete CD
264. # Moved processing code into function
265. DataProcessor.del\_id()
266. IO.show\_inventory(lstTbl)
267. **continue**  # start loop back at top.
269. # 3.6 process save inventory to file
270. **elif** strChoice == 's':
271. # 3.6.1 Display current inventory and ask user for confirmation to save
272. IO.show\_inventory(lstTbl)
273. strYesNo = input('Save this inventory to file? [y/n] ').strip().lower()
274. # 3.6.2 Process choice
275. **if** strYesNo == 'y':
276. # 3.6.2.1 save data
277. FileProcessor.write\_file(strFileName, lstTbl) # function call replaces previous code
278. **else**:
279. input('The inventory was NOT saved to file. Press [ENTER] to return to the menu.')
280. **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')

1. Dirk Biesinger, Module 06 Document, p. 3 [↑](#footnote-ref-1)
2. MDN Web Docs Glossary, “Parameter,” <https://developer.mozilla.org/en-US/docs/Glossary/Parameter>, retrieved 2020-Aug-15 [↑](#footnote-ref-2)
3. Module 6 Doc, p. 17 [↑](#footnote-ref-3)
4. “How to Check if a File or Directory Exists in Python, <https://linuxize.com/post/python-check-if-file-exists/>, retrieved 2020-Aug-16 [↑](#footnote-ref-4)