Jurg Huerlimann

Aug 18. 2020

IT FDN 110 B SU 20: Foundations of Programming: Python

Assignment Module 06

Functions,Classes ,Variable Scope,DocString

# Introduction

In the sixth module and assignment of this course, we expanded our knowledge of functions. We learned about passing variables to functions and return values.   
We practiced organizing code with functions and grouping them according the Separation of Concerns (SoC) principles., so that IO functions would be located under the IO Class and data processing functions under the data processing class.

We also learned about variable shadowing, what it is and what its pitfalls are.

Questions we had to investigate and answer:

## What is a function?

A function is a block of code that is reusable, it can be called from anywhere in the program, the execute a specific functionality. If the same set of commands needs to be repeated multiple times, it should be written as a function.

Functions can receive data via parameters and return data back to the main program.

## What are parameters?

Parameters are the values that are passed to functions for processing. Parameters are also called arguments.

## What are arguments?

Generally, arguments are the same as parameters. Arguments that are passed to the function call, have the form of either variables, constants, or not-values.

## What is the difference between parameters and arguments?

Generally when people say parameter/argument they mean the same thing, but the main difference between them is that the parameter is what is declared in the function, while an argument is what is passed through when calling the function. [[1]](#footnote-1)

## What are return values?

A return statement is used to end the execution of the function call and “returns” the result (value of the expression following the return keyword) to the caller. The statements after the return statements are not executed. If the return statement is without any expression, then the special value None is returned. [[2]](#footnote-2)

## What is shadowing?

Variable shadowing occurs when a variable declared within a certain scope (decision block, method, or inner class) has the same name as a variable declared in an outer scope. At the level of identifiers (names, rather than variables), this is known as name masking. This outer variable is said to be shadowed by the inner variable, while the inner identifier is said to mask the outer identifier. This can lead to confusion, as it may be unclear which variable subsequent uses of the shadowed variable name refer to, which depends on the name resolution rules of the language.[[3]](#footnote-3)

How do you use functions to organize your code?  
As functions are blocks of code that are reusable, I plan with pseudo-code which functionality of my script might be called repeatedly. Then I define such functions at the beginning of the program which I then can call throughout the flow of the main program.

What is the difference between a function and a class?  
Classes are a way of grouping functions, variables and constants.

How do functions help you program using the “Separations of Concerns" pattern?  
Functions help to divide the functionality of the script into modules, which then can be grouped according to the roles governing separation of concerns.

# Assignment06: Rearrange Python Script with Functions, Classes

In this sixth assignment we were tasked to modify and rearrange the provided Assignment06\_starter.py script, add more functions and finish organizing the code under the different classes, using the SoC principles.  
  
 Once finished we had to upload it again to GitHub for peer review.

## Resulting script:

The listing of the resulting script can be found in the Appendix. It is the result of doing the lab exercises in Module 06, reading up on the commands and methods, some experimenting and last but not least, feedback from the class forum, and the usual expert advice from Dirk and Doug.

## Testing script in Spyder:

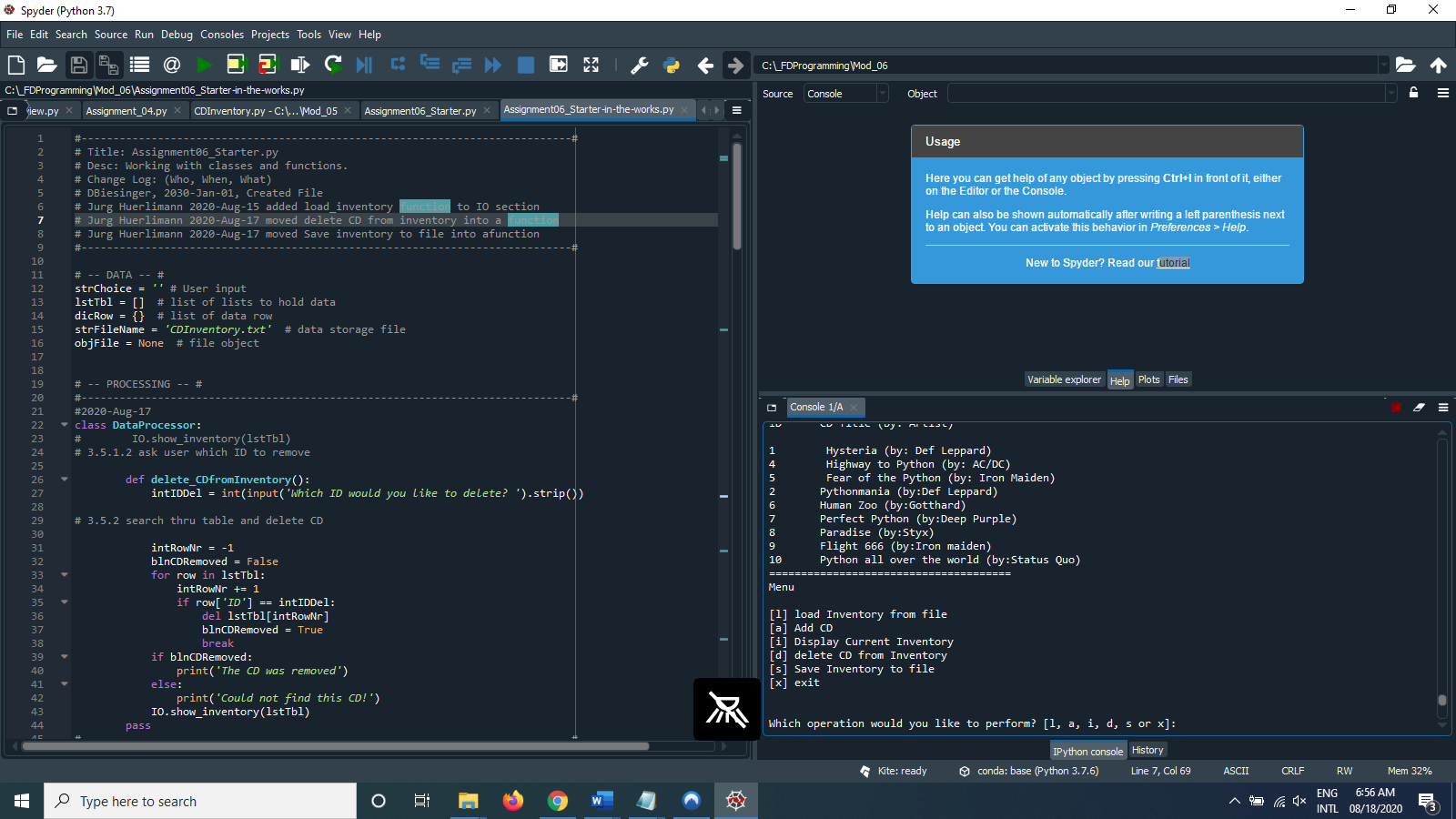


Figure Script running in Spyder

## Testing script in Terminal:

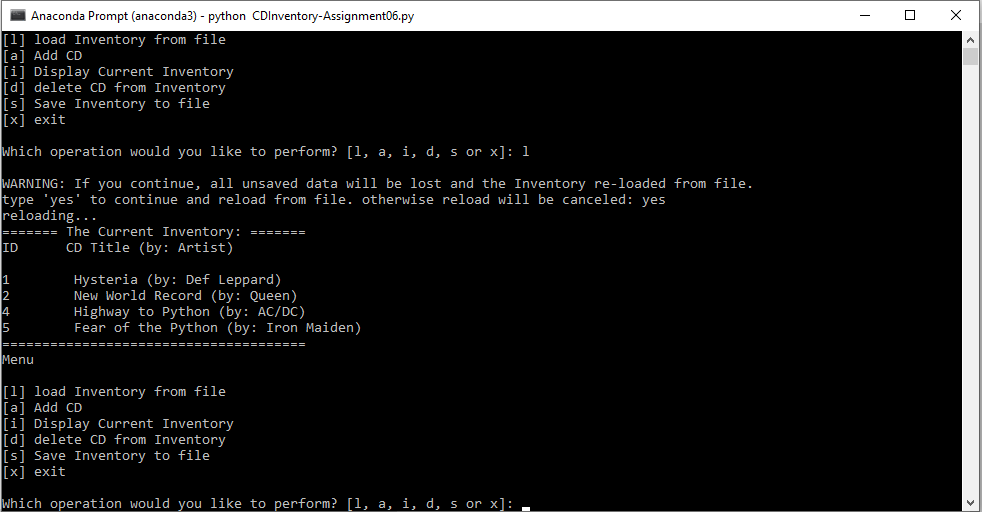


Figure Script running in Terminal

# Summary

This sixth assignment was based on our knowledge of functions, classes and Separation of Concerns (SoC)

During the discussions in class it dawned on me that I may have not fully converted the code as expected, as I just took the existing code and converted into functions, but did not introduce new variables that should be used to pass data to the functions.

However, the instructions in the Assigmment06 and the TODO instructions in the starter script were not very detailed in this regard.

I tested the functionality of my modified code and it seems to work as expected. Due to missing time this week, I am not able to improve the current code, improving how the functions work or adding a check if the CDInventory.txt file exist or not.

Regarding the feedback from Britta I received last week, I think I did a better job this time to implement more detailed footnotes where needed.

# Posting to GitHub:

The CDInventory.py script and knowledge document can be found on GitHub under this link:  
  
<https://github.com/jlhuerlimann/Assignment_06>

Appendix

1. #-----------------------------------------------------------------------------#
2. # Title: CDInventory.py
3. # Desc: Working with classes and functions.
4. # Change Log: (Who, When, What)
5. # DBiesinger, 2030-Jan-01, Created File
6. # Jurg Huerlimann 2020-Aug-15 added load\_inventory function to IO section
7. # Jurg Huerlimann 2020-Aug-17 moved delete CD from inventory into a function
8. # Jurg Huerlimann 2020-Aug-17 moved Save inventory to file into a function
9. # Jurg Huerlimann 2020-Aug-18 added comments, streamlined yes/no questions
10. # Jurg Huerlimann 2020-Aug-18 Saved modified Assignment06\_starter.py to CDInventory.py
11. #-----------------------------------------------------------------------------#
13. # -- DATA -- #
14. strChoice = '' # User input
15. lstTbl = []  # list of lists to hold data
16. dicRow = {}  # list of data row
17. strFileName = 'CDInventory.txt'  # data storage file
18. objFile = None  # file object

21. # -- PROCESSING -- #
23. **class** DataProcessor:
24. #        IO.show\_inventory(lstTbl)
25. # 3.5.1.2 ask user which ID to remove
27. **def** delete\_CDfromInventory():
28. intIDDel = int(input('Which ID would you like to delete? ').strip())
30. # 3.5.2 search thru table and delete CD
32. intRowNr = -1
33. blnCDRemoved = False
34. **for** row **in** lstTbl:
35. intRowNr += 1
36. **if** row['ID'] == intIDDel:
37. **del** lstTbl[intRowNr]
38. blnCDRemoved = True
39. **break**
40. **if** blnCDRemoved:
41. **print**('The CD was removed')
42. **else**:
43. **print**('Could not find this CD!')
44. IO.show\_inventory(lstTbl)
45. **pass**
46. #-----------------------------------------------------------------------------#
48. **class** FileProcessor:
49. """Processing the data to and from text file"""
51. @staticmethod
52. **def** read\_file(file\_name, table):
53. """Function to manage data ingestion from file to a list of dictionaries
55. Reads the data from file identified by file\_name into a 2D table
56. list of dicts) table one line in the file represents one dictionary row in table
58. Args:
59. file\_name (string): name of file used to read the data from
60. table (list of dict): 2D data structure (list of dicts) that holds the data during runtime
62. Returns:
63. None.
64. """
65. table.clear()  # this clears existing data and allows to load data from file
66. objFile = open(file\_name, 'r')
67. **for** line **in** objFile:
68. data = line.strip().split(',')
69. dicRow = {'ID': int(data[0]), 'Title': data[1], 'Artist': data[2]}
70. table.append(dicRow)
71. objFile.close()
73. @staticmethod
74. **def** write\_file(file\_name, table):
75. strYesNo = input('Save this inventory to file? [yes/no] ').strip().lower()
76. **if** strYesNo == 'yes':
77. objFile = open(strFileName, 'w')
78. **for** row **in** lstTbl:
79. lstValues = list(row.values())
80. lstValues[0] = str(lstValues[0])
81. objFile.write(','.join(lstValues) + '\n')
82. objFile.close()
83. **else**:
84. input('The inventory was NOT saved to file. Press [ENTER] to return to the menu.')
86. # -- PRESENTATION (Input/Output) -- #
88. **class** IO:
89. """Handling Input / Output"""
91. @staticmethod
92. **def** print\_menu():
93. """Displays a menu of choices to the user
95. Args:
96. None.
98. Returns:
99. None.
100. """
101. **print**('Menu\n\n[l] load Inventory from file\n[a] Add CD\n[i] Display Current Inventory')
102. **print**('[d] delete CD from Inventory\n[s] Save Inventory to file\n[x] exit\n')
104. @staticmethod
105. **def** menu\_choice():
106. """Gets user input for menu selection
108. Args:
109. None.
111. Returns:
112. choice (string): a lower case sting of the users input out of the
113. choices l, a, i, d, s or x
115. """
116. choice = ' '
117. **while** choice **not** **in** ['l', 'a', 'i', 'd', 's', 'x']:
118. choice = input('Which operation would you like to perform? [l, a, i, d, s or x]: ').lower().strip()
119. **print**()  # Add extra space for layout
120. **return** choice
122. @staticmethod
123. **def** show\_inventory(table):
124. """Displays current inventory table

127. Args:
128. table (list of dict): 2D data structure (list of dicts) that holds
129. the data during runtime.
131. Returns:
132. None.
134. """
135. **print**('======= The Current Inventory: =======')
136. **print**('ID\tCD Title (by: Artist)\n')
137. **for** row **in** table:
138. **print**('{}\t{} (by:{})'.format(\*row.values()))
139. **print**('======================================')
141. @staticmethod
142. **def** load\_inventory():
144. """Loading inventory from file
146. Args:
147. None.
149. Returns:
150. None.
152. """
153. **print**('WARNING: If you continue, all unsaved data will be lost and the Inventory re-loaded from file.')
154. strYesNo = input('type \'yes\' to continue and reload from file. otherwise reload will be canceled: ')
155. **if** strYesNo.lower() == 'yes':
156. **print**('reloading...')
157. FileProcessor.read\_file(strFileName, lstTbl)
158. IO.show\_inventory(lstTbl)
159. **else**:
160. input('canceling... Inventory data NOT reloaded. Press [ENTER] to continue to the menu.')
161. IO.show\_inventory(lstTbl)
163. @staticmethod
164. **def** add\_cdinfo():
165. # 3.3.2 Add item to the table
167. """Adding new CD information to memory
169. Args:
170. None.
172. Returns:
173. None.
175. """
177. strID = input('Enter ID: ').strip()
178. strTitle = input('What is the CD\'s title? ').strip()
179. strArtist = input('What is the Artist\'s name? ').strip()
181. intID = int(strID)
182. dicRow = {'ID': intID, 'Title': strTitle, 'Artist': strArtist}
183. lstTbl.append(dicRow)
184. IO.show\_inventory(lstTbl)
186. #----------------------------------------------------------------------------#
187. # 1. When program starts, read in the currently saved Inventory
188. FileProcessor.read\_file(strFileName, lstTbl)
190. # 2. start main loop
191. **while** True:
192. # 2.1 Display Menu to user and get choice
193. IO.print\_menu()
194. strChoice = IO.menu\_choice()
195. # 3. Process menu selection
196. # 3.1 process exit first
197. **if** strChoice == 'x':
198. **break**
199. # ============================================================================#
200. # 3.2 process load inventory
201. **if** strChoice == 'l':
202. IO.load\_inventory()
203. **continue**  # start loop back at top.
204. # ============================================================================#
205. # 3.3 process add a CD
206. **elif** strChoice == 'a':
207. IO.add\_cdinfo()
208. # 3.3.1 Ask user for new ID, CD Title and Artist
209. **continue**  # start loop back at top.
210. # ============================================================================#
211. # 3.4 process display current inventory
212. **elif** strChoice == 'i':
213. IO.show\_inventory(lstTbl)
214. **continue**  # start loop back at top.
215. # ============================================================================#
216. # 3.5 process delete a CD
217. **elif** strChoice == 'd':
218. # 3.5.1 get Userinput for which CD to delete
219. # 3.5.1.1 display Inventory to user
220. IO.show\_inventory(lstTbl)
221. DataProcessor.delete\_CDfromInventory()
222. **continue** # start loop back at top
223. # ============================================================================#
224. # 3.6 process save inventory to file
225. **elif** strChoice == 's':
226. # 3.6.1 Display current inventory and ask user for confirmation to save
227. IO.show\_inventory(lstTbl)
228. # 3.6.2 Process choice
229. FileProcessor.write\_file(strFileName, lstTbl)
230. **continue**  # start loop back at top.
231. # 3.7 catch-all should not be possible, as user choice gets vetted in IO, but to be save:
232. **else**:
233. **print**('General Error')
234. # ============================================================================#

Listing Code for CDInventory.py

1. Source: <https://stackoverflow.com/questions/47169033/parameter-vs-argument-python#:~:text=Generally%20when%20people%20say%20parameter,through%20when%20calling%20the%20function.> (2020-Aug-14) [↑](#footnote-ref-1)
2. Source: <https://www.geeksforgeeks.org/python-return-statement/> (2020-Aug-14) [↑](#footnote-ref-2)
3. Source: <https://en.wikipedia.org/wiki/Variable_shadowing> (2020-Aug-14) [↑](#footnote-ref-3)