Beemnet Workeneh

02/24/20

FDN 100

Assignment 06

Title

# Introduction

In this assignment we will learn about classes, functions, parameters, arguments, return values, local variables, global variables, variable scope, DocString, and shadowing. We will see how we can use functions to organize our code and what the role of functions is for SoC. Afterwards we will use the new learned concepts to improve upon the CDInventory code of the past weeks assignments.

# Exploration Questions

*What is a function?*

Functions are used to make code more clean and compact. When called, functions go off and perform a task and then return control to your program.

A function is defined the following way: def titleName(). It’s possible to write a function that does not take in a parameter or argument.

Functions allow for abstraction, which is not working about the small detains and rather the big picture of the script. Functions have a special mechanism that allows you to document them with docstring or documentation string—a docstring is typically a triple-quoted string that goes as the first line of the code. It is great for displaying tips about how the function works when writing code. Functions are great because they can be used in other programs. Functions can be saved as a module and imported to future programs. In python, the function must be defined before the call of the function.

*What are parameters?*

*Parameters are used in the definition of a function.*

Parameters catch the values sent to the function from a function call. Parameters are essentially variable names inside the parentheses of a function header. For example let’s consider an imaginary function that controls the speed of a spaceship with a velocity parameter:

def speedControl(velocity):

In the above line *velocity* is the parameter.

Functions can have many comma separated parameters. For example the above made up function can be redefined to take a second parameter.

def speedControl(velocity, acceleration):

In the above function definition velocity and acceleration are positional parameters.

We could also define the speedControl() function with default parameter values

def speedControl(velocity = 1000, acceleration = 10):

With the above definition if the function is called without any values, the default values 1000 and 10 will be used. This is how python makes use of overloaded functions.

*What are arguments?*

*Arguments are used in the calling of the function.*

Arguments are used to pass values to parameters when a function is called. Let’s say we call the made up function to control the speed of a spaceship by passing velocity and acceleration .

speedControl(1000, 10)

In the above line *1000* is the *argument* that gets passed to the *parameter velocity* and *10* is the value that gets assigned to acceleration*.*

Since the definition of the function was:

def speedControl(velocity, acceleration):

in the function definition velocity and acceleration are positional parameters.

When we call the function with the following line

speedControl(1000, 10)

in the calling of the function 1000 and 10 are positional arguments.

We could also call the function using keyword arguments and positional parameters

Values with immutable DataTypes pass to a function as Values and mutable DataTypes pass to a function as references.

*What is the difference between parameters and arguments?*

*Parameters are used in the definition of a function and arguments are used in the calling of a function. But in practice arguments and parameters are used interchangeably so it is up to the individual to recognize how the terms parameters and arguments are being used. .*

Parameters are the variables included in the header of a function and they catch the values that are sent to them by arguments when the function is called.

*What are return values?*

A return value is the value returned to the main body of the code after the execution of the function. For example let’s say the function speedControl() function returns a Boolean value of True when the desired velocity is attained. The return value could look like:

def speedControl(velocity, acceleration)

  speedAttained = None

  # some fancy calculation here

  return speedAttained

A function can return multiple comma separated return values however it’s important to note to have enough variables to catch all the return values of a function to avoid throwing an error.

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

Local variables are only valid only in the specific scope block in which they are defined. For example if a variable is defined in a function it is only valid in the function. If you have the same variable called in the main body of the script it can have a different value depending on what the code is doing.

Global variables are valid in a larger scope block. For example if we have a variable called out in the main body of the script and it is used (but not defined) in a function, it’s a global variable and it retains its value. However functions cannot modify the value of a global variable without special permission.

*What is shadowing?*

Shadowing is when we have two distinct variables that happen to have the same name. If we give a variable inside a function the same name as a global variable we “shadow” the global or we hide it with our new variable. However we cannot alter the global variable this way. We should avoid creating shadow variable as this can lead to confusion.

We can gain complete access to a global variable inside a function by using the keyword *global[[1]](#footnote-1)*.

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

Since all functions have to be defined before being called, we can group the functions together regardless of where they are called in the text. This allows us to display the data and presentation sections of our codes more distinctly.

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

Classes are a way of grouping functions, variables and constants.

For example let’s say we write a class for our incredibly simple made up spaceship.

class SpaceShipControl():

  @staticmethod

  def speedControl(velocity, acceleration):

    #some fancy math here

  @staticmethod

  def navigation( x, y, z):

    # more fancy math

The class contains two functions for speed and navigation.

If we define our functions within the class we can call them the following way:

SpaceShipControl.speedControl(1000, 10)

We cannot call a class within the program. But we can call a function inside the function.

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

Functions allow you to break up your code into manageable chunks. You can put a group of functions together to be in the processing section of the code.

# CDInventory.py Revision for Assignment 06

For this version of CDInventory program I once again modify the instructor’s code, this time to apply what I’ve learned about classes and functions.

Functions Added under class DataProcessor: add\_cd(), delete\_cd()

Functions Added under class FileProcessor: write\_file(),

***Defining function add\_cd()***

I define the add\_cd function to use take the user input for ID, CD title, and Artist’s name and append it to the table that gets passed to it when the function is called. After appending, the function returns the modified table. I do not define lstTbl in this function definition as I do not want global variables, instead I work with the generic parameter table.

1. @staticmethod
2. **def** add\_cd(table):
3. """ function to add a CD to the inventory
5. Args:
6. table: the list of dictionaries containing the CD entries
8. Returns:
9. the modified list of dictionaries with new etries of CDs
10. """
11. strID = input('Enter ID: ').strip()
12. strTitle = input('What is the CD\'s title? ').strip()
13. stArtist = input('What is the Artist\'s name? ').strip()
14. # Add item to the table
15. intID = int(strID)
16. dicRow = {'ID': intID, 'Title': strTitle, 'Artist': stArtist}
17. table.append(dicRow)
18. **return** table

Listing defining function add\_cd()

***Calling function add\_cd()***

I call the add\_cd() function by passing my list of dictionaries lstTbl. I make the current lstTbl equal to the table modified by the add\_cd() function. I opted to not include the show\_inventory() function within the add\_cd() function.

1. **elif** strChoice == 'a':
2. lstTbl = DataProcessor.add\_cd(lstTbl) # pass list of dictionaries to the add\_cd function to have new entries appended
3. IO.show\_inventory(lstTbl)
4. **continue**  # start loop back at top.

Listing Calling function add\_cd()

***Defining function delete\_cd()***

I define this function to take positional parameters table and ID.

1. @staticmethod
2. **def** delete\_cd(table, ID):
3. """ function to delete an entry from the inventory
5. Args:
6. table: the list of dictionaries containing the CD entries
7. ID: the integer ID of the CD to be deleted
9. Returns:
10. the modified list of dictionaries with the entry containing the ID removed
11. """
13. # search thru table and delete CD
14. intRowNr = -1
15. blnCDRemoved = False
16. **for** row **in** table:
17. intRowNr += 1
18. **if** row['ID'] == ID:
19. **del** table[intRowNr]
20. blnCDRemoved = True
21. **break**
22. **if** blnCDRemoved:
23. **print**('The CD was removed')
24. **else**:
25. **print**('Could not find this CD!')
26. **return** table

Listing defining function delete\_cd()

***Calling function delete\_cd()***

Similar to the function that adds CDs, this function also modifies the current lstTbl by passing itself to the delete CD function. One difference is that I have an additional argument intIDDel for the CD entry to be deleted. I opted to not include the show\_inventory() function within the delete\_cd() function.

1. **elif** strChoice == 'd':
2. # get Userinput for which CD to delete
3. # display Inventory to user
4. IO.show\_inventory(lstTbl)
5. # ask user which ID to remove
6. intIDDel = int(input('Which ID would you like to delete? ').strip())
7. lstTbl = DataProcessor.delete\_cd(lstTbl, intIDDel) # pass the list of dictionaries and ID to the delete\_cd function to have entries removed
8. IO.show\_inventory(lstTbl)
9. **continue**  # start loop back at top.

Listing calling function delete\_cd()

***Defining function write\_file()***

1. **def** write\_file(file\_name, table):
2. """Function to write data that is in a list of dictionaries to a text file
4. Reads the data from the list of dictionaries and writes into a file identified by file\_name ad
5. rows of comma delimited items.
7. Args:
8. file\_name(string): name of file used to write the data to
9. table( list of dict): 2D data structure (list of dicts) that holds the data during runtime
11. Returns:
12. None.
13. """
14. objFile = open(file\_name, 'w')# write mode instead of append mode
15. **for** row **in** table:
16. lstValues = list(row.values())
17. lstValues[0] = str(lstValues[0])
18. objFile.write(','.join(lstValues) + '\n')
19. objFile.close()

Listing defining function write\_file()

***Calling function write\_file()***

1. **elif** strChoice == 's':
2. # 3.6.1 Display current inventory and ask user for confirmation to save
3. # IO.show\_inventory(lstTbl)
4. strYesNo = input('Save this inventory to file? [y/n] ').strip().lower()
5. # 3.6.2 Process choice
6. **if** strYesNo == 'y':
7. # 3.6.2.1 save data
8. FileProcessor.write\_file(strFileName, lstTbl)
9. **else**:
10. input('The inventory was NOT saved to file. Press [ENTER] to return to the menu.')
11. **continue**  # start loop back at top.

Listing - calling function wirte\_file()

**Going through the options (l, a, i, d, s)**

Table Starting with an empty text file

|  |  |
| --- | --- |
| Executed in sypder | Executed in anaconda prompt |
|  |  |

Table Add CD

|  |  |
| --- | --- |
| Executed in sypder | Executed in anaconda prompt |
|  |  |

Table Delete CD from inventory

|  |  |
| --- | --- |
| Executed in sypder | Executed in anaconda prompt |
|  |  |

Table Display current inventory

|  |  |
| --- | --- |
| Executed in sypder | Executed in anaconda prompt |
|  |  |

Table Save inventory to file

|  |  |
| --- | --- |
| Executed in sypder | Executed in anaconda prompt |
|  |  |

Table load inventory from file

|  |  |
| --- | --- |
| Executed in sypder | Executed in anaconda prompt |
|  |  |

# Summary

In this assignment we learned that functions are used to make code clean and compact. Functions are defined before they are called and they help organize the processing part of code together. By definition parameters are used inside the definition of the function, and arguments are used inside the calling of a function however in practice parameters and arguments are used interchangeably. We saw that return values are values returned from the function. We saw that local variables are limited to the scope that they are defined whereas global variables are valid in a larger scope block. We learned that shadowing is when we have two distinct variables that happen to have the same name(one variable inside a function and one outside). We learned that this is not good practice. We learned that classes are used to group functions, variables and constants. Classes are not called to be executed instead individual functions within the class would have to be called. Functions allow putting the processing sections of a code into one location, allowing our script to be organized using SoC. The challenging part about working on the instructors code this time was trying to understand the additional I/Os as well as some syntax for deleting and reading file that I did not use in my previous version of the CD inventory code.

# Appendix

## CDInventory.py

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. # BWorkeneh, 2020-Feb-26, completed the TODOs in the starter 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. """ Processing the information in memory """
20. @staticmethod
21. **def** add\_cd(table):
22. """ function to add a CD to the inventory
24. Args:
25. table: the list of dictionaries containing the CD entries
27. Returns:
28. the modified list of dictionaries with new etries of CDs
29. """
30. strID = input('Enter ID: ').strip()
31. strTitle = input('What is the CD\'s title? ').strip()
32. stArtist = input('What is the Artist\'s name? ').strip()
33. # Add item to the table
34. intID = int(strID)
35. dicRow = {'ID': intID, 'Title': strTitle, 'Artist': stArtist}
36. table.append(dicRow)
37. **return** table
38. @staticmethod
39. **def** delete\_cd(table, ID):
40. """ function to delete an entry from the inventory
42. Args:
43. table: the list of dictionaries containing the CD entries
44. ID: the integer ID of the CD to be deleted
46. Returns:
47. the modified list of dictionaries with the entry containing the ID removed
48. """
50. # search thru table and delete CD
51. intRowNr = -1
52. blnCDRemoved = False
53. **for** row **in** table:
54. intRowNr += 1
55. **if** row['ID'] == ID:
56. **del** table[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. **return** table
65. **class** FileProcessor:
66. """Processing the data to and from text file"""
68. @staticmethod
69. **def** read\_file(file\_name, table):
70. """Function to manage data ingestion from file to a list of dictionaries
72. Reads the data from file identified by file\_name into a 2D table
73. (list of dicts) table one line in the file represents one dictionary row in table.
75. Args:
76. file\_name (string): name of file used to read the data from
77. table (list of dict): 2D data structure (list of dicts) that holds the data during runtime
79. Returns:
80. None.
81. """
82. table.clear()  # this clears existing data and allows to load data from file
83. objFile = open(file\_name, 'r')
84. **for** line **in** objFile:
85. data = line.strip().split(',')
86. dicRow = {'ID': int(data[0]), 'Title': data[1], 'Artist': data[2]}
87. table.append(dicRow)
88. objFile.close()
90. @staticmethod
91. **def** write\_file(file\_name, table):
92. """Function to write data that is in a list of dictionaries to a text file
94. Reads the data from the list of dictionaries and writes into a file identified by file\_name ad
95. rows of comma delimited items.
97. Args:
98. file\_name(string): name of file used to write the data to
99. table( list of dict): 2D data structure (list of dicts) that holds the data during runtime
101. Returns:
102. None.
103. """
104. objFile = open(file\_name, 'w')# write mode instead of append mode
105. **for** row **in** table:
106. lstValues = list(row.values())
107. lstValues[0] = str(lstValues[0])
108. objFile.write(','.join(lstValues) + '\n')
109. objFile.close()


113. # -- PRESENTATION (Input/Output) -- #
115. **class** IO:
116. """Handling Input / Output"""
118. @staticmethod
119. **def** print\_menu():
120. """Displays a menu of choices to the user
122. Args:
123. None.
125. Returns:
126. None.
127. """
129. **print**('Menu\n\n[l] load Inventory from file\n[a] Add CD\n[i] Display Current Inventory')
130. **print**('[d] delete CD from Inventory\n[s] Save Inventory to file\n[x] exit\n')
132. @staticmethod
133. **def** menu\_choice():
134. """Gets user input for menu selection
136. Args:
137. None.
139. Returns:
140. choice (string): a lower case string of the users input out of the choices l, a, i, d, s or x
142. """
143. choice = ' '
144. **while** choice **not** **in** ['l', 'a', 'i', 'd', 's', 'x']:
145. choice = input('Which operation would you like to perform? [l, a, i, d, s or x]: ').lower().strip()
146. **print**()  # Add extra space for layout
147. **return** choice
149. @staticmethod
150. **def** show\_inventory(table):
151. """Displays current inventory table
153. Args:
154. table (list of dict): 2D data structure (list of dicts) that holds the data during runtime.
156. Returns:
157. None.
159. """
160. **print**('======= The Current Inventory: =======')
161. **print**('ID\tCD Title (by: Artist)\n')
162. **for** row **in** table:
163. **print**('{}\t{} (by:{})'.format(\*row.values()))
164. **print**('======================================')

167. # 1. When program starts, read in the currently saved Inventory
168. FileProcessor.read\_file(strFileName, lstTbl)
170. # 2. start main loop
171. **while** True:
172. # 2.1 Display Menu to user and get choice
173. IO.print\_menu()
174. strChoice = IO.menu\_choice()
175. # 3. Process menu selection
176. # 3.1 process exit first
177. **if** strChoice == 'x':
178. **break**
179. # 3.2 process load inventory
180. **if** strChoice == 'l':
181. **print**('WARNING: If you continue, all unsaved data will be lost and the Inventory re-loaded from file.')
182. strYesNo = input('type \'yes\' to continue and reload from file. otherwise reload will be canceled: ')
183. **if** strYesNo.lower() == 'yes':
184. **print**('reloading...')
185. FileProcessor.read\_file(strFileName, lstTbl)
186. IO.show\_inventory(lstTbl)
187. **else**:
188. input('canceling... Inventory data NOT reloaded. Press [ENTER] to continue to the menu.')
189. IO.show\_inventory(lstTbl)
190. **continue**  # start loop back at top.
191. # 3.3 process add a CD
192. **elif** strChoice == 'a':
193. lstTbl = DataProcessor.add\_cd(lstTbl) # pass list of dictionaries to the add\_cd function to have new entries appended
194. IO.show\_inventory(lstTbl)
195. **continue**  # start loop back at top.
196. # 3.4 process display current inventory
197. **elif** strChoice == 'i':
198. IO.show\_inventory(lstTbl)
199. **continue**  # start loop back at top.
200. # 3.5 process delete a CD
201. **elif** strChoice == 'd':
202. # get Userinput for which CD to delete
203. # display Inventory to user
204. IO.show\_inventory(lstTbl)
205. # ask user which ID to remove
206. intIDDel = int(input('Which ID would you like to delete? ').strip())
207. lstTbl = DataProcessor.delete\_cd(lstTbl, intIDDel) # pass the list of dictionaries and ID to the delete\_cd function to have entries removed
208. IO.show\_inventory(lstTbl)
209. **continue**  # start loop back at top.
210. # 3.6 process save inventory to file
211. **elif** strChoice == 's':
212. # 3.6.1 Display current inventory and ask user for confirmation to save
213. # IO.show\_inventory(lstTbl)
214. strYesNo = input('Save this inventory to file? [y/n] ').strip().lower()
215. # 3.6.2 Process choice
216. **if** strYesNo == 'y':
217. # 3.6.2.1 save data
218. FileProcessor.write\_file(strFileName, lstTbl)
219. **else**:
220. input('The inventory was NOT saved to file. Press [ENTER] to return to the menu.')
221. **continue**  # start loop back at top.
222. # 3.7 catch-all should not be possible, as user choice gets vetted in IO, but to be safe:
223. **else**:
224. **print**('General Error')

1. Dawson, Michael. Python® Programming for the Absolute Beginner, Third Edition. Course Technology PTR, 2009. [↑](#footnote-ref-1)