Muthulekshmi Sivathanu

02-21-2021

Foundations of programming python

Assignment-06

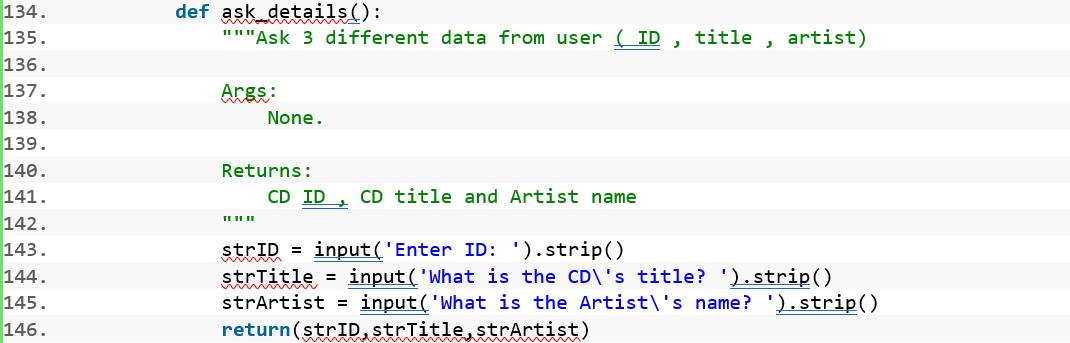
Classes and Functions

# Introduction

In this module I learnt in detail about functions with arguments and without arguments. This module also dealt about passing mutable and immutable objects. I got a brief introduction about classes also.

# Functions

Unlike C, functions can return more than one value as in the below code in Listing 1



Listing 1 Function returning more than one value in CDInventory.py

# Functions and **Arguments**

We can have either mutable or immutable objects as arguments. In python, the system is called, call by object reference, or call by assignment ([External Reference](https://www.geeksforgeeks.org/is-python-call-by-reference-or-call-by-value/)[[1]](#footnote-1)). If you pass arguments like immutable objects (whole numbers, strings, tuples), the passing is like call be value. The immutable objects passed to function does not get affected by any changes in the function definition. The happens, even if they are received in the same name in the function definition. Below Listing 2 explains the behavior.







Listing 2 Output of Program VariablesAndFunctions.py detailing variable mutability

The function definition part creates a local copy of the arguments received when doing any update on it. In the event that we pass arguments like mutable objects (list, dictionary), the passing is like call by reference. The mutable objects passed to the function gets affected, if any changes happen to them in the function definition. This happens even if they are received in another name. Below output Listing 3 of program Variables and Functions in Listing 2 explains this behavior.





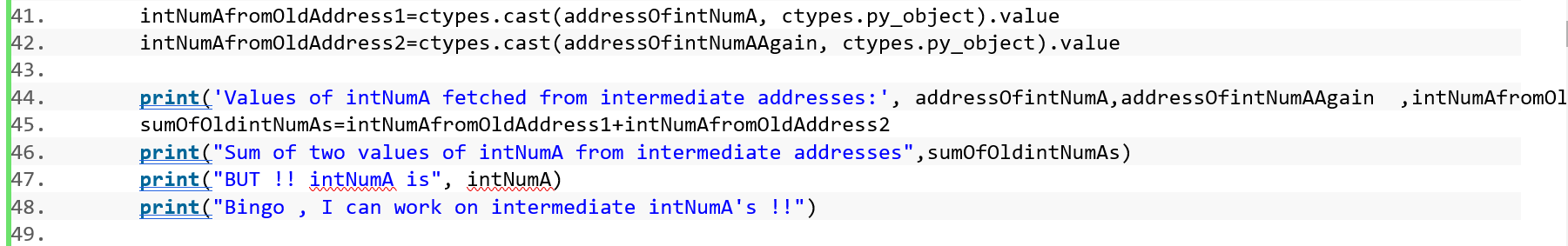
Listing 3 Output of Variables and Functions program detailing the behavior of list with function

# Observations on passing mutable objects and immutable objects.

Addresses of variables which received the values are the same as the addresses of variables that were passed through function call statement.

Address of a variable did not change even when it is used in an assignment statement (in case of no change in the previous value).

Addresses of the variables changed each time when there was an update. I could even retrieve the data from previous addresses and do some calculations on it, as in below Listing 4



Listing 4 Snippet from VariablesAndFunctions.py

 Whenever an update is done, a new copy of the variable is created, and it is updated in a new address. The old address retained the old value of the variable.

Variable automatically becomes local when it is subjected to change and the original parameter that was passed do not get affected.

After function definition, the same address which was there before the function call for the parameter was retained even after the function call.

Address of list which received the values was the same as the addresses of list that was passed through function call statement.

Unlike integer variable, whatever change that was made on the list in the function definition was reflected on the actual list that was passed through function call.

Whenever an update is done, the address of the whole list is never changed, but the address of the values in the list gets changed. Listing 5 shows the output from Listing 6 which shows how list gets changed when passing through function









Listing 5 Snippet from output of VariablesAndFunctions.py which shows list gets changed in function

# Summary

It was interesting to know how functions work in python. I had a detailed study on arguments and return values in a function. Making changes on the starter code was difficult even this time. Adding function document headers was helpful. Actual use of classes is yet to be understood in real time.

# Appendix

## VariablesAndFunctions.py

1. #!/usr/bin/env python3
2. # -\*- coding: utf-8 -\*-
3. """
4. Title: VariablesAndFunctions.py
5. Desc: Program to understand where variables get stored in and out of function
6. Change Log:(who,When,What)
7. Muthu,Fri Feb 19 18:21:18 2021,New File Created
8. """

11. #######passing list to function
12. **import** ctypes
13. intNumA=2
14. intNumB=4
15. lstResult=[1000,2000]

18. **def** Cal\_Result\_List(intNumA,intNumB,results):
19. **print**('Address of intNumA after receiving it in function defnition',id(intNumA))
20. **print**('Address of variable remained unchanged !!!')
22. intNumA=intNumA\*1
24. **print**('Address of intNumA inside function definition after multiplied by 1')
25. **print**(id(intNumA))
26. **print**('Address of variable remained unchanged even now, since data is not changed !!!')

29. intNumA=intNumA\*5
30. **print**('Address of first parameter inside function after first change made',id(intNumA))
31. **print**('Address of variable changed now !!!')
33. #store the address of variable intNumA
34. addressOfintNumA=id(intNumA)
36. intNumA=intNumA\*10
37. **print**('Address of first parameter inside function after second change made',id(intNumA))
38. **print**('Address of variable changed AGAIN now !!!')
39. addressOfintNumAAgain=id(intNumA)
41. intNumAfromOldAddress1=ctypes.cast(addressOfintNumA, ctypes.py\_object).value
42. intNumAfromOldAddress2=ctypes.cast(addressOfintNumAAgain, ctypes.py\_object).value
44. **print**('Values of intNumA fetched from intermediate addresses:', addressOfintNumA,addressOfintNumAAgain  ,intNumAfromOldAddress1,intNumAfromOldAddress2)
45. sumOfOldintNumAs=intNumAfromOldAddress1+intNumAfromOldAddress2
46. **print**("Sum of two values of intNumA from intermediate addresses",sumOfOldintNumAs)
47. **print**("BUT !! intNumA is", intNumA)
48. **print**("Bingo , I can work on intermediate intNumA's !!")

51. **print**('values of intNumA and intNumB before exiting Function Defn')
52. **print**(intNumA,intNumB,'')
54. results[0]=intNumA
55. results[1]=intNumB
56. **print**("Updated list inside funtion is " , results)
58. **print**("========================\n")
59. **print**("intNumA  BEFORE FunctionCALL",intNumA)
60. **print**("intNumB BEFORE FunctionCALL",intNumB)
61. **print**('list value BEFORE FUNCTION CALL:',lstResult)
62. **print**('Address of First element in list before function call',id(lstResult[0]))
63. **print**('Address of intNumA before call is',id(intNumA))

66. **print**("========================\n")
67. Cal\_Result\_List(intNumA,intNumB,lstResult)
68. **print**("========================\n")
69. **print**("intNumA outside function defnition AFTERCALL",intNumA)
70. **print**("intNumB outside function defnition AFTERCALL",intNumB)
71. **print**("All the variables remain unchanged ")
72. **print**('list value AFTER FUNCTION CALL:',lstResult)
73. **print**("List is changed !!")
74. **print**('Address of First element in list After function call',id(lstResult[0]))
75. **print**("Address of element in the list after function call is changed !!")
76. **print**('Address of intNumA after function defnition is',id(intNumA))
77. **print**("Address of intNumA is retained even after we changed with it in function")
78. **print**("========================")

Listing 6 VariablesAndFunctions program

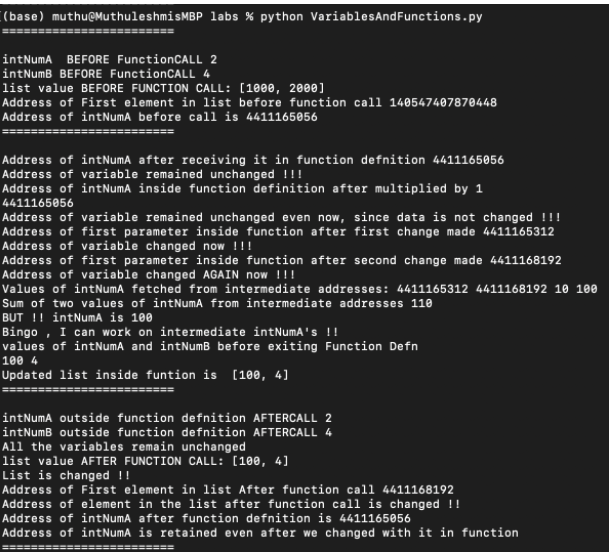


Figure 1 Output of Variables And Functions python script

1. #------------------------------------------#
2. # Title: CDInventory.py
3. # Desc: Working with classes and functions for Assignment 06
4. # Change Log: (Who, When, What)
5. # Muthu, 2021-Feb-19 , Analyzed starter script and decided on new functions to be added
6. # Muthu,2021-Feb-19 , Added new functions add\_cd
7. # Muthu 2021-Feb-20 , Added new function del\_cd
8. # Muthu 2021-Feb-21 , Updated main program and checked whether all functions are being called at right time
9. #------------------------------------------#
11. **import** os

14. # -- DATA -- #
15. strChoice = '' # User input
16. lstTbl = []  # list of lists to hold data
17. dicRow = {}  # list of data row
18. strFileName = 'CDInventory.txt'  # data storage file
19. objFile = None  # file object


23. **if** **not** os.path.exists(strFileName):
24. with open(strFileName, 'w'):**pass**
26. # -- PROCESSING -- #
27. **class** DataProcessor:
29. """Add/Delete new CD Data entered by user to/from the table"""
31. **def** add\_cd(strID,strTitle,strArtist):
32. """Function to add data entered by user to the 2D table
34. Write CD Details as a dictionary entry .
35. Add this dictionary to the 2D list .
37. Args:
38. strID (string ): ID for the CD entered by user
39. strTitle (string ) : Title for the CD entered by user
40. strArtist ( String ) : Artist name for the CD entered by user
42. Returns:
43. None.
44. """
45. intID = int(strID)
46. dicRow = {'ID': intID, 'Title': strTitle, 'Artist': strArtist}
47. lstTbl.append(dicRow)
49. **def** delete\_CD(intIDDel,lstTbl):
50. """Function to delete a row identified by user from 2D table
52. Checks if there is a match between the ID entered by the user and the ID present in the list .
53. If there is a match delete the row that matches the ID .
54. Display the latest content in the table .
56. Args:
57. intIDDel (Integer) : The CD ID entered by user
58. table (list of dict): 2D data structure (list of dicts) that holds the data during runtime
60. Returns:
61. None.
62. """
63. intRowNr = -1
64. blnCDRemoved = False
65. **for** row **in** lstTbl:
66. intRowNr += 1
67. **if** row['ID'] == intIDDel:
68. **del** lstTbl[intRowNr]
69. blnCDRemoved = True
70. **break**
71. **if** blnCDRemoved:
72. **print**('The CD was removed\n\n')
73. **else**:
74. **print**('Could not find this CD!')
75. IO.show\_inventory(lstTbl)


79. **class** FileProcessor:
81. """Processing the data to and from text file"""
83. @staticmethod
84. **def** read\_file(strFileName, lstTbl):
85. """Function to manage data ingestion from file to a list of dictionaries
87. Reads the data from file identified by file\_name into a 2D table
88. (list of dicts) table one line in the file represents one dictionary row in table.
90. Args:
91. file\_name (string): name of file used to read the data from
92. table (list of dict): 2D data structure (list of dicts) that holds the data during runtime
94. Returns:
95. None.
96. """
97. lstTbl.clear()  # this clears existing data and allows to load data from file
98. objFile = open(strFileName, 'r')
99. **for** line **in** objFile:
100. data = line.strip().split(',')
101. dicRow = {'ID': int(data[0]), 'Title': data[1], 'Artist': data[2]}
102. lstTbl.append(dicRow)
103. objFile.close()
105. @staticmethod
106. **def** write\_file(strFileName,lstTbl):
107. """Function to write data from list of dictionaries to a file
109. Reads the data from 2D table row by row
110. Create a list with values from the key value pair of each row
111. Join the contents of this list with a "," seperator
112. Write the list to the file
114. Args:
115. file\_name (string): name of file used to write the data
116. table (list of dict): 2D data structure (list of dicts) that holds the data during runtime
118. Returns:
119. None.
120. """
121. objFile = open(strFileName, 'w')
122. **for** row **in** lstTbl:
123. lstValues = list(row.values())
124. lstValues[0] = str(lstValues[0])
125. objFile.write(','.join(lstValues) + '\n')
126. objFile.close()

129. # -- PRESENTATION (Input/Output) -- #
131. **class** IO:
132. """Handling Input / Output"""
134. **def** ask\_details():
135. """Ask 3 different data from user ( ID , title , artist)
137. Args:
138. None.
140. Returns:
141. CD ID , CD title and Artist name
142. """
143. strID = input('Enter ID: ').strip()
144. strTitle = input('What is the CD\'s title? ').strip()
145. strArtist = input('What is the Artist\'s name? ').strip()
146. **return**(strID,strTitle,strArtist)

149. @staticmethod
150. **def** print\_menu():
151. """Displays a menu of choices to the user
153. Args:
154. None.
156. Returns:
157. None.
158. """
160. **print**('Menu\n\n[l] load Inventory from file\n[a] Add CD\n[i] Display Current Inventory')
161. **print**('[d] delete CD from Inventory\n[s] Save Inventory to file\n[x] exit\n')
163. @staticmethod
164. **def** menu\_choice():
165. """Gets user input for menu selection
167. Args:
168. None.
170. Returns:
171. choice (string): a lower case sting of the users input out of the choices l, a, i, d, s or x
173. """
174. choice = ' '
175. **while** choice **not** **in** ['l', 'a', 'i', 'd', 's', 'x']:
176. choice = input('Which operation would you like to perform? [l, a, i, d, s or x]: ').lower().strip()
177. **print**()  # Add extra space for layout
178. **return** choice
180. @staticmethod
181. **def** show\_inventory(table):
182. """Displays current inventory table

185. Args:
186. table (list of dict): 2D data structure (list of dicts) that holds the data during runtime.
188. Returns:
189. None.
191. """
192. **print**('======= The Current Inventory: =======')
193. **print**('ID\t\t\tCD Title\t\t\t(by: Artist)\n')
194. **for** row **in** table:
195. **print**('{}\t\t\t{}\t\t\t\t(by:{})'.format(\*row.values()))
196. **print**('======================================\n\n')


200. # 1. When program starts, read in the currently saved Inventory
202. FileProcessor.read\_file(strFileName, lstTbl)
204. # 2. start main loop
205. **while** True:
206. # 2.1 Display Menu to user and get choice
207. IO.print\_menu()
208. strChoice = IO.menu\_choice()
210. # 3. Process menu selection
211. # 3.1 process exit first
212. **if** strChoice == 'x':
213. **break**
215. # 3.2 process load inventory
216. **if** strChoice == 'l':
217. **print**('WARNING: If you continue, all unsaved data will be lost and the Inventory re-loaded from file.')
218. strYesNo = input('type \'yes\' to continue and reload from file. otherwise reload will be canceled : ')
219. **if** strYesNo.lower() == 'yes':
220. **print**('reloading...')
221. FileProcessor.read\_file(strFileName, lstTbl)
222. IO.show\_inventory(lstTbl)
223. **else**:
224. input('canceling... Inventory data NOT reloaded. Press [ENTER] to continue to the menu: ')
225. IO.show\_inventory(lstTbl)
226. **continue**  # start loop back at top.
228. # 3.3 process add a CD
229. **elif** strChoice == 'a':
230. # 3.3.1 Ask user for new ID, CD Title and Artist
231. strID,strTitle,strArtist=IO.ask\_details()
233. # 3.3.2 Add item to the table
234. DataProcessor.add\_cd(strID,strTitle,strArtist)
236. IO.show\_inventory(lstTbl)
237. **continue**  # start loop back at top.
239. # 3.4 process display current inventory
240. **elif** strChoice == 'i':
241. IO.show\_inventory(lstTbl)
242. **continue**  # start loop back at top.
244. # 3.5 process delete a CD
245. **elif** strChoice == 'd':
246. # 3.5.1 get Userinput for which CD to delete
247. # 3.5.1.1 display Inventory to user
248. IO.show\_inventory(lstTbl)
249. # 3.5.1.2 ask user which ID to remove
250. intIDDel = int(input('Which ID would you like to delete? ').strip())
251. # 3.5.2 search thru table and delete CD
252. DataProcessor.delete\_CD(intIDDel,lstTbl)
253. **continue**  # start loop back at top.
255. # 3.6 process save inventory to file
256. **elif** strChoice == 's':
257. # 3.6.1 Display current inventory and ask user for confirmation to save
258. IO.show\_inventory(lstTbl)
259. strYesNo = input('Save this inventory to file? [y/n] ').strip().lower()
260. # 3.6.2 Process choice
261. **if** strYesNo == 'y':
262. # 3.6.2.1 save data
263. FileProcessor.write\_file(strFileName, lstTbl)
264. **else**:
265. input('The inventory was NOT saved to file. Press [ENTER] to return to the menu.')
266. **continue**  # start loop back at top.
268. # 3.7 catch-all should not be possible, as user choice gets vetted in IO, but to be save:
269. **else**:
270. **print**('General Error')

Listing 7 CDIntventory.py using functions and class

Menu[l] load Inventory from file[a] Add CD[i] Display Current Inventory[d] delete CD from Inventory[s] Save Inventory to file[x] exitWhich operation would you like to perform? [l, a, i, d, s or x]: lWARNING: If you continue, all unsaved data will be lost and the Inventory re-loaded from file.type 'yes' to continue and reload from file. otherwise reload will be canceled : yesreloading...======= The Current Inventory: =======ID CD Title (by: Artist)======================================Menu[l] load Inventory from file[a] Add CD[i] Display Current Inventory[d] delete CD from Inventory[s] Save Inventory to file[x] exitWhich operation would you like to perform? [l, a, i, d, s or x]: aEnter ID: 1What is the CD's title? oneWhat is the Artist's name? one======= The Current Inventory: =======ID CD Title (by: Artist)1 one (by:one)======================================Menu[l] load Inventory from file[a] Add CD[i] Display Current Inventory[d] delete CD from Inventory[s] Save Inventory to file[x] exitWhich operation would you like to perform? [l, a, i, d, s or x]: i======= The Current Inventory: =======ID CD Title (by: Artist)1 one (by:one)======================================Menu[l] load Inventory from file[a] Add CD[i] Display Current Inventory[d] delete CD from Inventory[s] Save Inventory to file[x] exitWhich operation would you like to perform? [l, a, i, d, s or x]: s======= The Current Inventory: =======ID CD Title (by: Artist)1 one (by:one)======================================Save this inventory to file? [y/n] lThe inventory was NOT saved to file. Press [ENTER] to return to the menu.Menu[l] load Inventory from file[a] Add CD[i] Display Current Inventory[d] delete CD from Inventory[s] Save Inventory to file[x] exitWhich operation would you like to perform? [l, a, i, d, s or x]: s======= The Current Inventory: =======ID CD Title (by: Artist)1 one (by:one)======================================Save this inventory to file? [y/n] yMenu[l] load Inventory from file[a] Add CD[i] Display Current Inventory[d] delete CD from Inventory[s] Save Inventory to file[x] exitWhich operation would you like to perform? [l, a, i, d, s or x]: i======= The Current Inventory: =======ID CD Title (by: Artist)1 one (by:one)======================================Menu[l] load Inventory from file[a] Add CD[i] Display Current Inventory[d] delete CD from Inventory[s] Save Inventory to file[x] exitWhich operation would you like to perform? [l, a, i, d, s or x]:

Figure Output of CDInventory.py

1. Referred on 19-02-2021 [↑](#footnote-ref-1)