Kelly Ros

2020-Feb-29

Foundations of Programming, Python

Assignment 06

Functions and Classes

# Introduction

In this module, we learn how to create and use functions and are introduced to classes, variable scopes, and docstrings.

# Functions and Classes

We have previously used functions in this course, and to reiterate what we’ve learned before, functions are a group of statements that are available for use when called by its defined name in a program. Calling the function executes the group of statements in the function and can be used to return results or process data. When functions return a value (can be none, one, or many), the function ends and the program should have a variable or variables to be set to the function.[[1]](#footnote-1) We can also define our functions to accept arguments or set these arguments to default values in case they aren’t used. These arguments allow the function to perform different actions or return results that would require the values set to those arguments. Variables created inside of a function only exist within that function and cannot be called in the main portion of the progam unless it has the global keyword used.[[2]](#footnote-2) It is common practice to include a document header, also known as docstrings, at the beginning of a function An example of a function with arguments and a docstring can be seen in listing 1.

1. **def** getPro(intA, intB):
2. """Function that multiplies two values
4. Args:
5. intA: first number to multiply
6. intB: second number to multiply
8. Returns:
9. A float that corresponds to the product of intA and intB
10. """
11. **return** intA \* intB

Listing – Lab06\_C.py’s getPro function.

In this module, we covered the basics of classes. Currently we know that classes are a way of grouping functions, variables, and constants. In order to define a class, the keyword “class” is used, it is usually followed by a short documentation string that tells any user of the class what the functions are used for. In the program, we can call a function from a class using the notation seen in listing 2.

1. **print**('Product:\t', SimpleMath.getPro(num1, num2))

Listing – calling the getPro function from the class SimpleMath

# CDinventory.py

In this week’s assignment, we were given a CD Inventory script. Our task is to define some functions and replace the code in the main portion of the code with the new functions. Initially, I ran the code to make sure that all portions worked and took note of what didn’t work so it could be fixed later. Since I didn’t have a CDinventory.txt file in the same folder, I added in a try-except block to create the text file if it didn’t exist. The rest of the program seemed to work just fine, so I added in functions into the predefined classes. The functions that were added were: add\_CD(idno, title, artist, table), delete\_CD(idno, table), write\_file(file\_name, table), and input\_CD().

After we created the functions, we replaced the code underneath each option with the new functions. We can see an example of the add\_CD() function from the DataProcessor class and its use in listings 3 and 4.

1. @staticmethod
2. **def** add\_CD(idno, title, artist, table):
3. """Function that adds CD to list of dictionaries
5. Args:
6. idno (string): ID number of CD, which will be converted to int
7. title (string): CD title
8. artist (string): CD artist
9. table (list of dicts): 2D table that will be appended with new CD entry
11. Returns:
12. None
13. """
14. intID = int(idno)
15. dicRow  = {'ID': intID, 'Title': strTitle, 'Artist': stArtist}
16. table.append(dicRow)

Listing – add\_CD() function from DataProcessor Class

1. # 3.3.2 Add item to the table
2. DataProcessor.add\_CD(strID, strTitle, stArtist, lstTbl)

Listing – Calling add\_CD() function from DataProcessor Class

After we created the functions and replaced parts of the program to use the newly created functions, the CDInventory program was run to make sure all parts of the program worked. We can see the outputs in figures 1 through 6. After finishing the script, I uploaded the code to [GitHub](https://github.com/kellyros12/Assignment06).[[3]](#footnote-3)

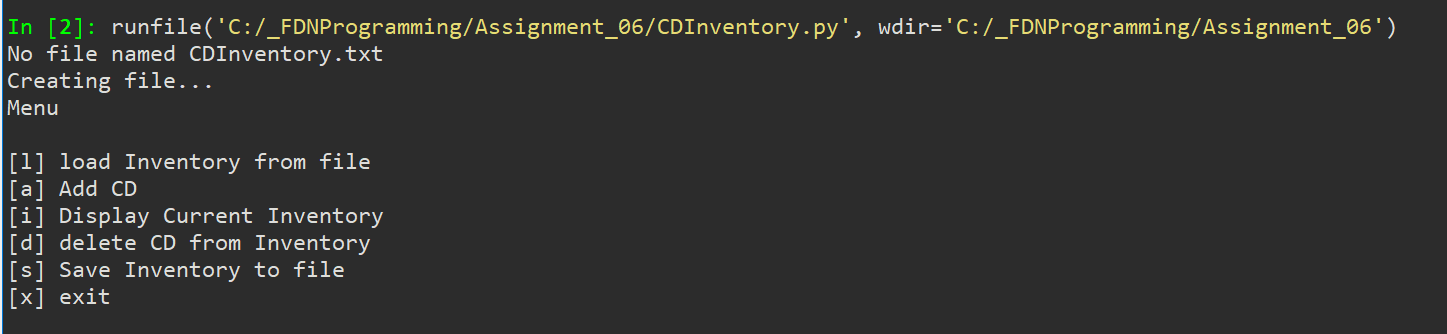


Figure – Initial run of CDInventory program, no CDInventory text file was found.

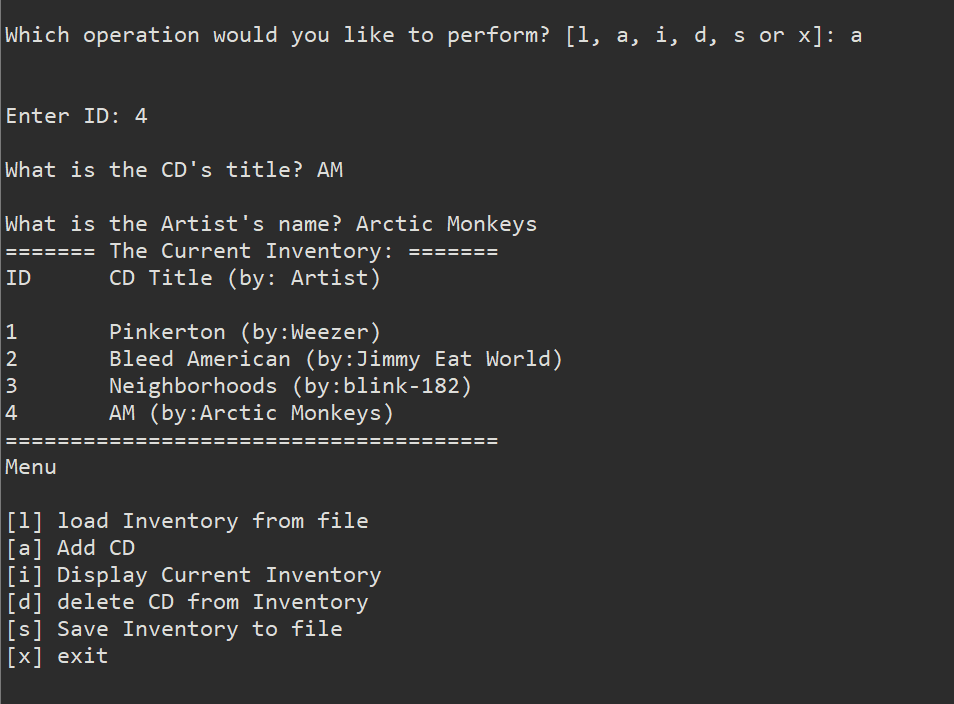


Figure – Add CD function was used

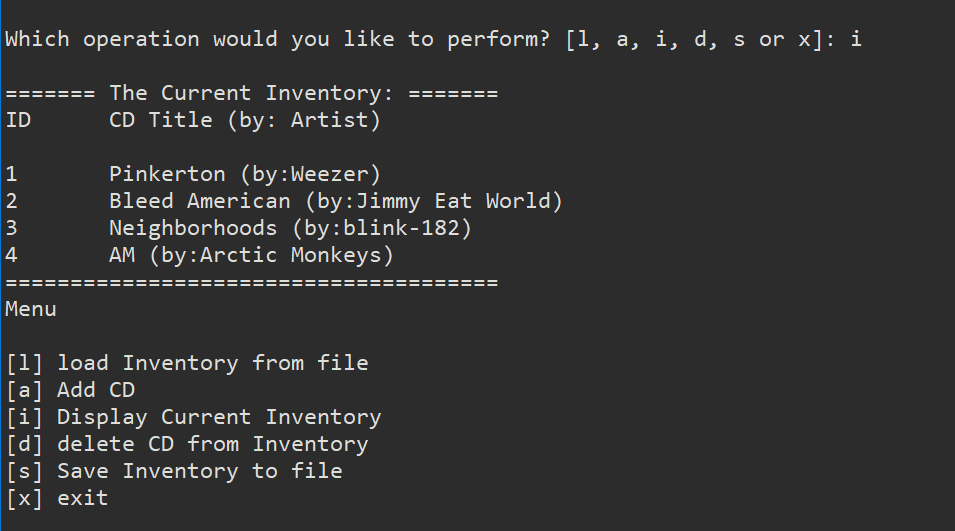


Figure – Inventory display function

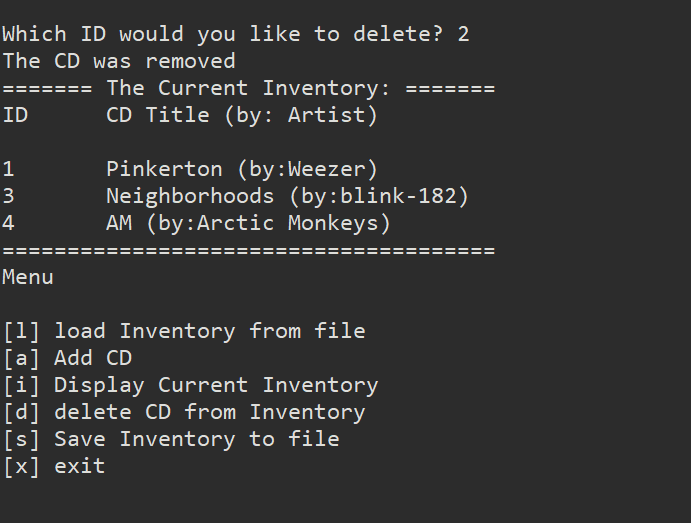


Figure – Delete function used to remove ID 2 from list

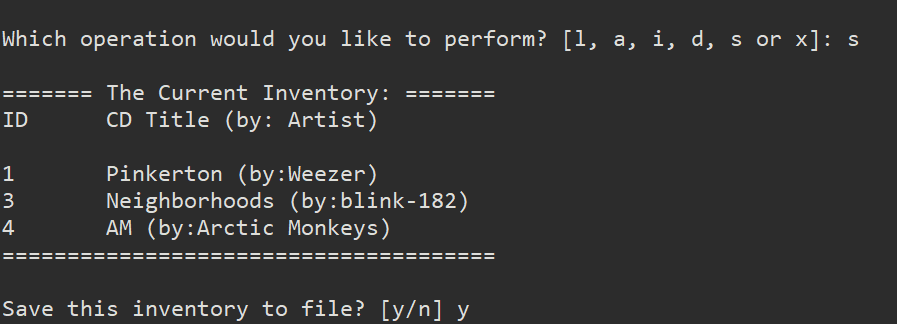


Figure – Save function that saves current inventory to CDInventory text file.

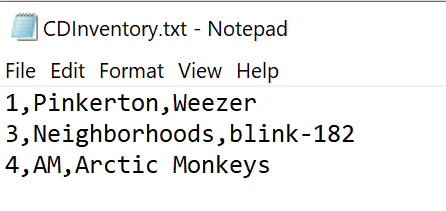


Figure – Test file after this run

# Summary

In this week’s assignment, we covered functions and went over a brief overview of classes. We’ll return to classes in a future module but for now, we use classes to group functions, variables, and constants. Functions can be defined and used in order to make the main portion of the program look cleaner and easier to debug. We also modified someone else’s program in this week’s assignment, similar to last week. Some parts of modifying code is simple, while adding code can be difficult since we don’t know what the original programmer was thinking when it was written. This shows why it’s important to have docstrings and comments that help other programmers who read and modify code in the future.

# Appendix

## Listing Lab06\_C.py

1. #------------------------------------------#
2. # Title: Lab06\_C.py
3. # Desc: Basic Classes
4. # Change Log: (Who, When, What)
5. # KRos, 2020-Feb-26, Copied File and edited code
6. #------------------------------------------#
7. # -- DATA -- #
8. num1 = None
9. num2 = None
11. # -- PROCESSING -- #
12. **class** SimpleMath():
13. """A collection of simple math processing functions"""
14. @staticmethod
15. **def** getSum(intA, intB):
16. """Function that adds two values
18. Args:
19. intA: value 1
20. intB: value 2
22. Returns:
23. A float sum of IntA and IntB
24. """
25. **return** float(intA + intB)
26. @staticmethod
27. **def** getDif(intA, intB):
28. """Function that subracts two values
30. Args:
31. intA: the number subtracted from
32. intB: the number to subtract
34. Returns:
35. A float that corresponds to subtracting intB from intA
36. """
37. **return** float(intA - intB)
38. @staticmethod
39. **def** getPro(intA, intB):
40. """Function that multiplies two values
42. Args:
43. intA: first number to multiply
44. intB: second number to multiply
46. Returns:
47. A float that corresponds to the product of intA and intB
48. """
49. **return** intA \* intB
50. @staticmethod
51. **def** getQuo(intA, intB):
52. """Function that divides two values
54. Args:
55. intA: number to be divided (numerator)
56. intB: number to divide by (denominator)
58. Returns:
59. A float that corresponds to the quotient of intA and intB
60. """
61. intQuo = 'Error'
62. **try**:
63. intQuo = intA / intB
64. **except**:
65. **print**('An error occurred while dividing', intA, 'by', intB)
66. **return** intQuo
67. # -- PRESENTATION (Input/Output) -- #
68. # Get User Data
69. **print**('Basic Math script. Calculates the Sum, Difference, Product, and Quotient of two numbers.')
70. num1 = int(input("Enter the first number: "))
71. num2 = int(input("Enter the second number: "))
72. #Display the results
73. **print**('\n\nThis script calculated using the numbers:', num1, 'and', num2)
74. **print**('\nThe results are:')
75. **print**('Sum:\t\t', SimpleMath.getSum(num1, num2), '\nDifference:\t', SimpleMath.getDif(num1, num2))
76. **print**('Product:\t', SimpleMath.getPro(num1, num2))
77. **print**('Quotient:\t', SimpleMath.getQuo(num1,num2))

## Listing CDInventory.py

1. #------------------------------------------#
2. # Title: Assignment06\_Starter.py
3. # Desc: Working with classes and functions.
4. # Change Log: (Who, When, What)
5. # DBiesinger, 2030-Jan-01, Created File
6. # KRos, 2020-Feb-28, added write\_file function
7. # KRos, 2020-Feb-29, created add\_CD and delete\_CD functions, added Docstrings
8. # KRos, 2020-Mar-01, added in try-except for case where file does not exist
9. #------------------------------------------#
11. # -- DATA -- #
12. strChoice = '' # User input
13. lstTbl = []  # list of lists to hold data
14. dicRow = {}  # list of data row
15. strFileName = 'CDInventory.txt'  # data storage file
16. objFile = None  # file object

19. # -- PROCESSING -- #
20. **class** DataProcessor:
21. """Data Processing Functions"""
23. @staticmethod
24. **def** add\_CD(idno, title, artist, table):
25. """Function that adds CD to list of dictionaries
27. Args:
28. idno (string): ID number of CD, which will be converted to int
29. title (string): CD title
30. artist (string): CD artist
31. table (list of dicts): 2D table that will be appended with new CD entry
33. Returns:
34. None
35. """
36. intID = int(idno)
37. dicRow  = {'ID': intID, 'Title': strTitle, 'Artist': stArtist}
38. table.append(dicRow)
40. @staticmethod
41. **def** delete\_CD(idno, table):
42. """Function that deletes a CD entry from a 2D table
44. Args:
45. idno (int): CD ID number to be deleted
46. table (list of dicts): 2D list that CD should be removed from
48. Returns:
49. None
50. """
51. intRowNr = -1
52. blnCDRemoved = False
53. **for** row **in** lstTbl:
54. intRowNr += 1
55. **if** row['ID'] == intIDDel:
56. **del** lstTbl[intRowNr]
57. blnCDRemoved = True
58. **break**
59. **if** blnCDRemoved:
60. **print**('The CD was removed')
61. **else**:
62. **print**('Could not find this CD!')
64. **class** FileProcessor:
65. """Processing the data to and from text file"""
67. @staticmethod
68. **def** read\_file(file\_name, table):
69. """Function to manage data ingestion from file to a list of dictionaries
71. Reads the data from file identified by file\_name into a 2D table
72. (list of dicts) table one line in the file represents one dictionary row in table.
73. If file does not exist, a file will be created
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. **try**:
84. objFile = open(file\_name, 'r')
85. **for** line **in** objFile:
86. data = line.strip().split(',')
87. dicRow = {'ID': int(data[0]), 'Title': data[1], 'Artist': data[2]}
88. table.append(dicRow)
89. objFile.close()
90. **except**:
91. **print**("No file named", file\_name)
92. **print**("Creating file...")
93. objFile = open(file\_name, 'a')
94. objFile.close()
95. @staticmethod
96. **def** write\_file(file\_name, table):
97. """Function that Saves inventory to text file
98. Reads 2D table and formats line by line to write to a text file
100. Args:
101. file\_name (string): name of file used to save data to
102. table (list of dict): 2D data structure that holds data during runtime
104. Returns:
105. None
106. """
107. objFile = open(file\_name, 'w')
108. **for** row **in** table:
109. lstValues = list(row.values())
110. lstValues[0] = str(lstValues[0])
111. objFile.write(','.join(lstValues) + '\n')
112. objFile.close()
114. # -- PRESENTATION (Input/Output) -- #
116. **class** IO:
117. """Handling Input / Output"""
119. @staticmethod
120. **def** print\_menu():
121. """Displays a menu of choices to the user
123. Args:
124. None.
126. Returns:
127. None.
128. """
130. **print**('Menu\n\n[l] load Inventory from file\n[a] Add CD\n[i] Display Current Inventory')
131. **print**('[d] delete CD from Inventory\n[s] Save Inventory to file\n[x] exit\n')
133. @staticmethod
134. **def** menu\_choice():
135. """Gets user input for menu selection
137. Args:
138. None.
140. Returns:
141. choice (string): a lower case sting of the users input out of the choices l, a, i, d, s or x
143. """
144. choice = ' '
145. **while** choice **not** **in** ['l', 'a', 'i', 'd', 's', 'x']:
146. choice = input('Which operation would you like to perform? [l, a, i, d, s or x]: ').lower().strip()
147. **print**()  # Add extra space for layout
148. **return** choice
150. @staticmethod
151. **def** show\_inventory(table):
152. """Displays current inventory table

155. Args:
156. table (list of dict): 2D data structure (list of dicts) that holds the data during runtime.
158. Returns:
159. None.
161. """
162. **print**('======= The Current Inventory: =======')
163. **print**('ID\tCD Title (by: Artist)\n')
164. **for** row **in** table:
165. **print**('{}\t{} (by:{})'.format(\*row.values()))
166. **print**('======================================')
168. @staticmethod
169. **def** input\_CD():
170. """Asks user for CD inputs
172. Args:
173. None
175. Returns:
176. ID, CD title, Artist
178. """
179. strID = input('Enter ID: ').strip()
180. strTitle = input('What is the CD\'s title? ').strip()
181. stArtist = input('What is the Artist\'s name? ').strip()
182. **return** strID, strTitle, stArtist
184. # 1. When program starts, read in the currently saved Inventory
185. FileProcessor.read\_file(strFileName, lstTbl)
187. # 2. start main loop
188. **while** True:
189. # 2.1 Display Menu to user and get choice
190. IO.print\_menu()
191. strChoice = IO.menu\_choice()
193. # 3. Process menu selection
194. # 3.1 process exit first
195. **if** strChoice == 'x':
196. **break**
197. # 3.2 process load inventory
198. **if** strChoice == 'l':
199. **print**('WARNING: If you continue, all unsaved data will be lost and the Inventory re-loaded from file.')
200. strYesNo = input('type \'yes\' to continue and reload from file. otherwise reload will be canceled \n')
201. **if** strYesNo.lower() == 'yes':
202. **print**('reloading...')
203. FileProcessor.read\_file(strFileName, lstTbl)
204. IO.show\_inventory(lstTbl)
205. **else**:
206. input('canceling... Inventory data NOT reloaded. Press [ENTER] to continue to the menu.')
207. IO.show\_inventory(lstTbl)
208. **continue**  # start loop back at top.
209. # 3.3 process add a CD
210. **elif** strChoice == 'a':
211. # 3.3.1 Ask user for new ID, CD Title and Artist
212. strID, strTitle, stArtist = IO.input\_CD()
213. # 3.3.2 Add item to the table
214. DataProcessor.add\_CD(strID, strTitle, stArtist, lstTbl)
215. IO.show\_inventory(lstTbl)
216. **continue**  # start loop back at top.
217. # 3.4 process display current inventory
218. **elif** strChoice == 'i':
219. IO.show\_inventory(lstTbl)
220. **continue**  # start loop back at top.
221. # 3.5 process delete a CD
222. **elif** strChoice == 'd':
223. # 3.5.1 get Userinput for which CD to delete
224. # 3.5.1.1 display Inventory to user
225. IO.show\_inventory(lstTbl)
226. # 3.5.1.2 ask user which ID to remove
227. intIDDel = int(input('Which ID would you like to delete? ').strip())
228. # 3.5.2 search thru table and delete CD
229. DataProcessor.delete\_CD(intIDDel, lstTbl)
230. IO.show\_inventory(lstTbl)
231. **continue**  # start loop back at top.
232. # 3.6 process save inventory to file
233. **elif** strChoice == 's':
234. # 3.6.1 Display current inventory and ask user for confirmation to save
235. IO.show\_inventory(lstTbl)
236. strYesNo = input('Save this inventory to file? [y/n] ').strip().lower()
237. # 3.6.2 Process choice
238. **if** strYesNo == 'y':
239. # 3.6.2.1 save data
240. FileProcessor.write\_file(strFileName, lstTbl)
241. **else**:
242. input('The inventory was NOT saved to file. Press [ENTER] to return to the menu.')
243. **continue**  # start loop back at top.
244. # 3.7 catch-all should not be possible, as user choice gets vetted in IO, but to be save:
245. **else**:
246. **print**('General Error')

1. From Python Programming for the Absolute Beginner, Third Edition. Pg 164. Retrieved on 2020-Feb-28. [↑](#footnote-ref-1)
2. From Python Programming for the Absolute Beginner, Third Edition. Pg ###. Retrieved on 2020-Feb-28. [↑](#footnote-ref-2)
3. Link to GitHub Repository: <https://github.com/kellyros12/Assignment06> [↑](#footnote-ref-3)