Meredith Bailey

March 2, 2020

Foundations of Programming: Python

Assignment 06

Assignment 6 Knowledge Document

# Introduction

In this module, we learned about functions, how to use arguments within functions, and how to process return values. We also learned about variable scope inside of functions, variable shadowing, and global variables.

# Functions and Variable Scope

In this module, we learned about functions, which are a method of grouping statements of code and making this group available via a programmer-defined name. Functions can pass in parameters, or arguments, which are values for processing. These arguments are usually variables, not values. Functions also have return values, which can be consumed instantly (as in a print statement) or assigned to a variable. Functions can also return ‘None’. When returning multiple values, the values will be bundled into a collection and the collection will be returned. Additionally, at the calling point, the code must accept a collection as a return value. In Python, the implicit packing and unpacking of tuples accomplishes this. However, other languages do not have this feature and must be manually unpacked using something like list indexing. When calling a function, inputting only the argument and not the name of the argument implicitly uses the arguments and assigns them in the same sequence as the arguments defined in the function declaration. However, you could include the name of the argument if desired. Arguments can have default values set in the function definition, and these values will be used if no value for this argument is provided in the function call. Python does not allow programmers to create multiple functions with the same name. Argument values with immutable data types such as strings, integers, floats, and tuples pass to a function as values, while mutable data types pass to a function as a reference. When creating functions, it is helpful to create a docstring within the function, which is a brief description of what the function does, what its arguments are, and what values it returns.

Next, we learned about variable scope. Variables defined within a function are only valid within the scope of that function. This also means that the memory used to hold the value of a variable inside a function is different than the memory used outside the function. Outside variables defined in the main body are valid in child functions. In order to change variables that are defined in the main body within a function, the keyword “global” can be used, though it is considered bad practice. Shadowing of variables occurs when there are two different variables, one inside a function, and one outside of the function, with the same name. It is discouraged as it makes code more difficult to read.

# Adding Functions to a Script File

I only encountered a few problems when creating this program. When I first migrated the processing commands into functions, I forgot to change the variable names inside the function into argument names. However, I realized my mistake before running the program and quickly utilized the argument names inside of the functions instead of the variable names. When I ran the program for the first time, I received an error because the first line of the main body could not be executed. This line loaded the existing CDInventory.txt file, but I received an error because there was no existing file. I then utilized a try-except method to handle the error, in the case that the user is running the program for the first time.

# 

Figure : Try-except used for loading file at program start.

The result of using the error handling is as follows, with Figure 2 showing the outcome of using the program with no inventory file created, and Figure 3 showing the outcome of loading the file upon the program’s start.

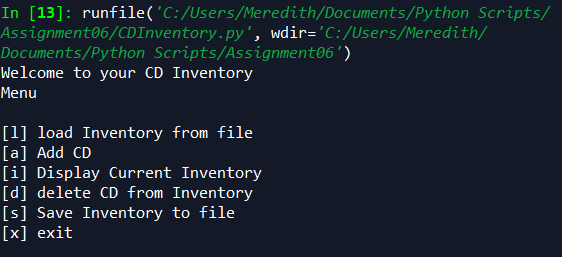


Figure : The "except" branch of the try-except structure.

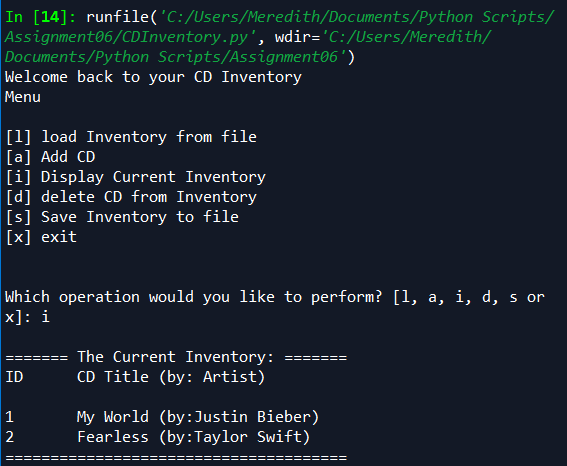


Figure : The "try" branch of the try-except structure.

Finally, I had an issue when I created my docstrings. I got an unindent error, telling me that there was unexpected indentation at the first line of code (not the first line of the docstring) in the function. I was confused because I had run my program before adding docstrings and it worked perfectly, so I knew there wasn’t an issue with the function itself. It turned out that I had one too many spaces before the first line of my docstring. I learned that even though the docstring is a comment, the indentation of it matters. When I resolved the indentation issue, my program ran smoothly, and created the text file as shown in Figure 4.

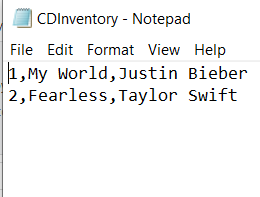


Figure : CDInventory.txt file creation.

The script file can be found on GitHub at <https://github.com/meredithbailey3/Assignment06>

# Summary

In this module, we learned how to use functions, arguments, and return values. We learned about the implicit packing and unpacking of multiple return values in Python, and how to use positional and named arguments. We also learned about variable scope within functions, and how to change variables on a global level.

# Appendix

Source code from this assignment (created using [PlanetB’s](https://planetb.ca/syntax-highlight-word) website):

1. #------------------------------------------#
2. # Title: Assignment06\_Starter.py
3. # Desc: Working with classes and functions.
4. # Change Log: (Who, When, What)
5. # MBailey, 2020-Feb-28, Added Functions, Docstrings, and Try/Except
6. #------------------------------------------#
8. # -- DATA -- #
9. strChoice = '' # User input
10. lstTbl = []  # list of lists to hold data
11. dicRow = {}  # list of data row
12. strFileName = 'CDInventory.txt'  # data storage file
13. objFile = None  # file object

16. # -- PROCESSING -- #
17. **class** DataProcessor:
18. # TODO add functions for processing here - COMPLETE
19. # TODO move IO code into function - COMPLETE
20. @staticmethod
21. **def** user\_input(table):
22. """
23. Function to take in user's CD data, insert into a dictionary, and append dictionary to table.
25. Args: table (list): name of list used to store dictionaries
27. Returns: none
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. # TODO move processing code into function - COMPLETE
34. intID = int(strID)
35. dicRow = {'ID': intID, 'Title': strTitle, 'Artist': stArtist}
36. table.append(dicRow)
38. @staticmethod
39. **def** delete\_row(table, IDnumber):
40. """
41. Function to delete a CD's dictionary based on user's choice of which ID number to delete.
42. Loops through the 'ID' key's value in each dictionary and checks if it is equal to the user's input.
44. Args: table (list): name of list used to store dictionaries
45. IDnumber (int): user's input of which ID to delete
47. Returns: none
49. """
50. # TODO move processing code into function - COMPLETE
51. intRowNr = -1
52. blnCDRemoved = False
53. **for** row **in** table:
54. intRowNr += 1
55. **if** row['ID'] == IDnumber:
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!')
64. @staticmethod
65. **def** save\_file(table, filename):
66. """
67. Function to save table to a file.
69. Args: table (list): name of list used to store dictionaries
70. filename (string): the name of the .txt file where data will be written.
72. Returns: none
74. """
75. # TODO move processing code into function - COMPLETE
76. objFile = open(filename, 'w')
77. **for** row **in** table:
78. lstValues = list(row.values())
79. lstValues[0] = str(lstValues[0])
80. objFile.write(','.join(lstValues) + '\n')
81. objFile.close()

84. **class** FileProcessor:
85. """Processing the data to and from text file"""
87. @staticmethod
88. **def** read\_file(file\_name, table):
89. """Function to manage data ingestion from file to a list of dictionaries
91. Reads the data from file identified by file\_name into a 2D table
92. (list of dicts) table. One line in the file represents one dictionary row in table.
94. Args:
95. file\_name (string): name of file used to read the data from
96. table (list of dict): 2D data structure (list of dict) that holds the data during runtime
98. Returns:
99. None.
100. """
101. table.clear()  # this clears existing data and allows to load data from file
102. objFile = open(file\_name, 'r')
103. **for** line **in** objFile:
104. data = line.strip().split(',')
105. dicRow = {'ID': int(data[0]), 'Title': data[1], 'Artist': data[2]}
106. table.append(dicRow)
107. objFile.close()
109. @staticmethod
110. **def** write\_file(file\_name, table):
111. # TODO Add code here
112. **pass**

115. # -- PRESENTATION (Input/Output) -- #
117. **class** IO:
118. """Handling Input / Output"""
120. @staticmethod
121. **def** print\_menu():
122. """Displays a menu of choices to the user
124. Args:
125. None.
127. Returns:
128. None.
129. """
131. **print**('Menu\n\n[l] load Inventory from file\n[a] Add CD\n[i] Display Current Inventory')
132. **print**('[d] delete CD from Inventory\n[s] Save Inventory to file\n[x] exit\n')
134. @staticmethod
135. **def** menu\_choice():
136. """Gets user input for menu selection
138. Args:
139. None.
141. Returns:
142. choice (string): a lower case sting of the users input out of the choices l, a, i, d, s or x
144. """
145. choice = ' '
146. **while** choice **not** **in** ['l', 'a', 'i', 'd', 's', 'x']:
147. choice = input('Which operation would you like to perform? [l, a, i, d, s or x]: ').lower().strip()
148. **print**()  # Add extra space for layout
149. **return** choice
151. @staticmethod
152. **def** show\_inventory(table):
153. """Displays current inventory table

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