Maria Dacutanan

Aug 12, 2020

Foundations of Programming: Python

Assignment 05

Lists and Dictionaries

# Introduction

In module 05, I get to work more extensively on the list sequence type and learn ways of manipulating lists using the different list methods. I am presented with a new data type called dictionaries and how it can work hand in hand with lists.

In the last module, I learned how to write data into text file. But this time around, I get to read the contents of a text file into memory and manipulate it as I wished.

I am got a brief taste of how to define and invoke functions as well as the concept of structured error handling.

Lastly, I get introduced GitHub which is a repository manager and presented with a new concept called separation of concerns which is a design principle of how to structure code.

# Reading Contents of a File and Storing into Lists for Processing

As I have learned in the previous module, lists, like tuples, are a collection of data objects, but unlike tuples, lists are mutable. Part of the fun in this homework is to be able to read contents of the file and load it into memory so that it can be manipulated through lists.

In Figure 1 - Load Data from File Into List, the **open()** function in read mode is used to open CDInventory.txt and store contents into objfile variable. Through the for loop variable, I am able to iterate thru the contents line by line, format each entry with **strip** and **split** methods and then store the formatted data into a list variable.

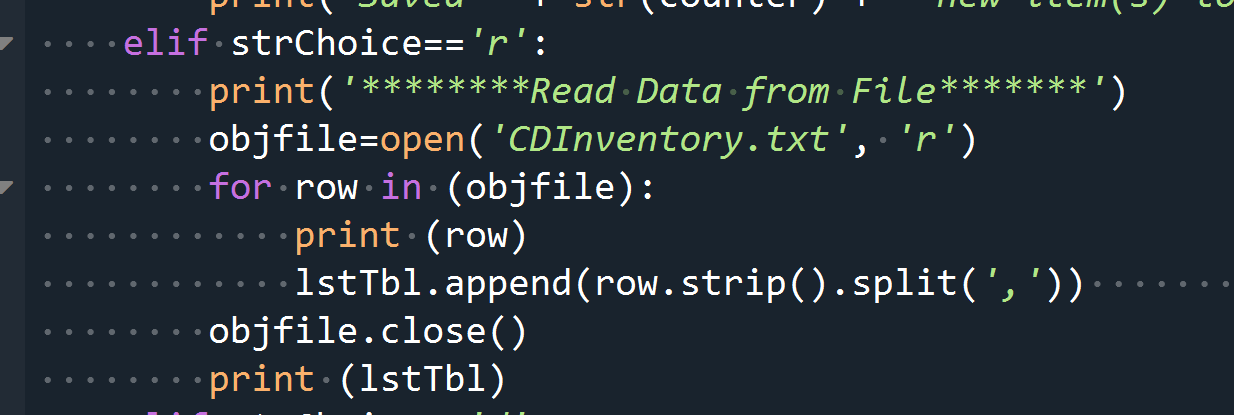


Figure 1 - Load Data from File Into List

Unpacking A List

One can unpack a list using the \* overload operator. Figure 2 - Unpacking A List demonstrates how this is done. Variables to receive the unpacked data are on the left side of the equal operator where one of the variables has an \* in front of it and then the list variable to the right of the equal operator.

This [website](https://note.nkmk.me/en/python-tuple-list-unpack/)[[1]](#footnote-1) explains various ways of unpacking lists, tuples as well as nested tuples and nested lists.

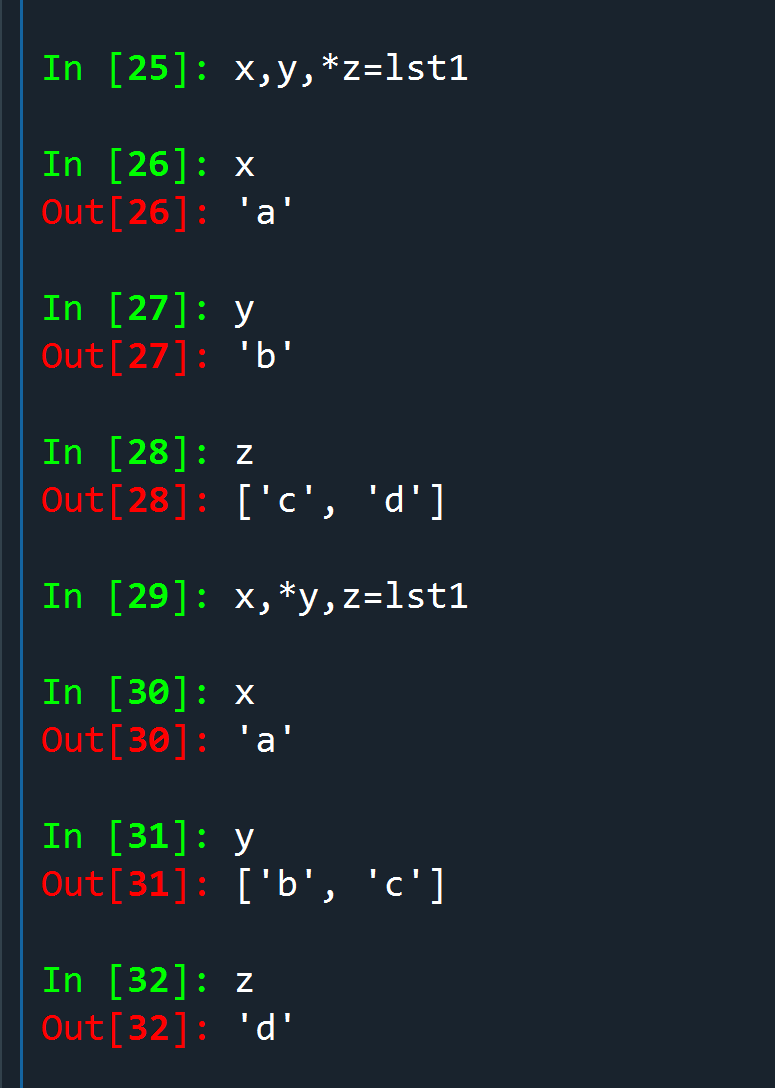


Figure 2 - Unpacking A List

List Methods

Pretty much the same methods that can be used against a string sequence type can also be used for lists. Just like string methods, to use a list method, you specify the list followed by .*<method name>*(). This [website](https://towardsdatascience.com/python-list-methods-fa7c53010300)[[2]](#footnote-2) lists out and explains what and how to use the different list methods in more detail.

For the lab and homework, I got to use a handful of them. I think the most interesting ones that I was presented with in this module were various ways of deleting list elements: **remove()**, **del()** and **pop()**

* *<list>*.**remove**(<*element>*) – The remove() method goes through the sequence elements, starting at position 0 and searches for the value passed on to it. When first occurrence of the values is found, that element is deleted from the list sequence. If value occurs more than once in the sequence, ONLY THE FIRST OCCURRENCE is deleted.
* **del** *<list>*[*<index>*] – While remove() method deletes an element based on value, del() function removes an element based on the index number.
* *<list>*.**pop**(*<index>*) – If no index number is specified, pop() method deletes the last element of the sequence by default. The main difference between del() and pop() is that pop() deletes the element based on index and displays the deleted data as well.

Figure 3 - Removing List Elements illustrates ways of removing list elements.

[111] : 
[112] : 
[113] : 
Out[113] : 
In [114]: 
In [115]: 
Out[115] : 
In [116]: 
Out[116] : 
55] 
1st3. remove(55) 
1st3 
[10, 20, 30, 40] 
1st3[3] 
del 
1st3 
[10, 20, 30] 
1st3. pop(0) 
10 

Figure 3 - Removing List Elements

Accessing Elements of a Nested List Sequence

In the previous homework, I talked about ways of accessing elements of a list sequence via indexing and slicing. I also learned that a list can contain more than one data type like a another list , or sublist. In this module, I gained knowledge of how to access the elements of the sublist inside a list. It is still done thru indexing and slicing but instead of specifying a single index position, you can specify multiple indices to go deeper.

Figure 4 - Accessing Nested List Sequence demonstrates how one can access elements of a sublist within another list.

نرا 
( 
( 
ا ل 

Figure 4 - Accessing Nested List Sequence

# Shared References

There is a section in the textbook[[3]](#footnote-3) which I believe is important to drill down and understand fully. This is understanding that when variables refer to the same mutable value, like lists, they share the same reference. That is to say that a change to the value of one variable result to changes to all other variables that references that same value. I learned this lesson first hand when I was writing the code for the homework.

In Figure 5 - Shared References (a), I have 2 separate lists lstA and lstB. I then appended lstB to lstA. But then I decided to change value for lstB[1] from Taylor Swift to Amy Winehouse. When I checked lstA, Taylor swift was also replaced by Amy Winehouse.

To further drive home the point, in Figure 6 - Shared References (b), I did the reverse where I modified value for lstA[3][0] from Eminem to Def Leppard. When I checked lstB, Eminem is no more and was replaced by Def Leppard, too.

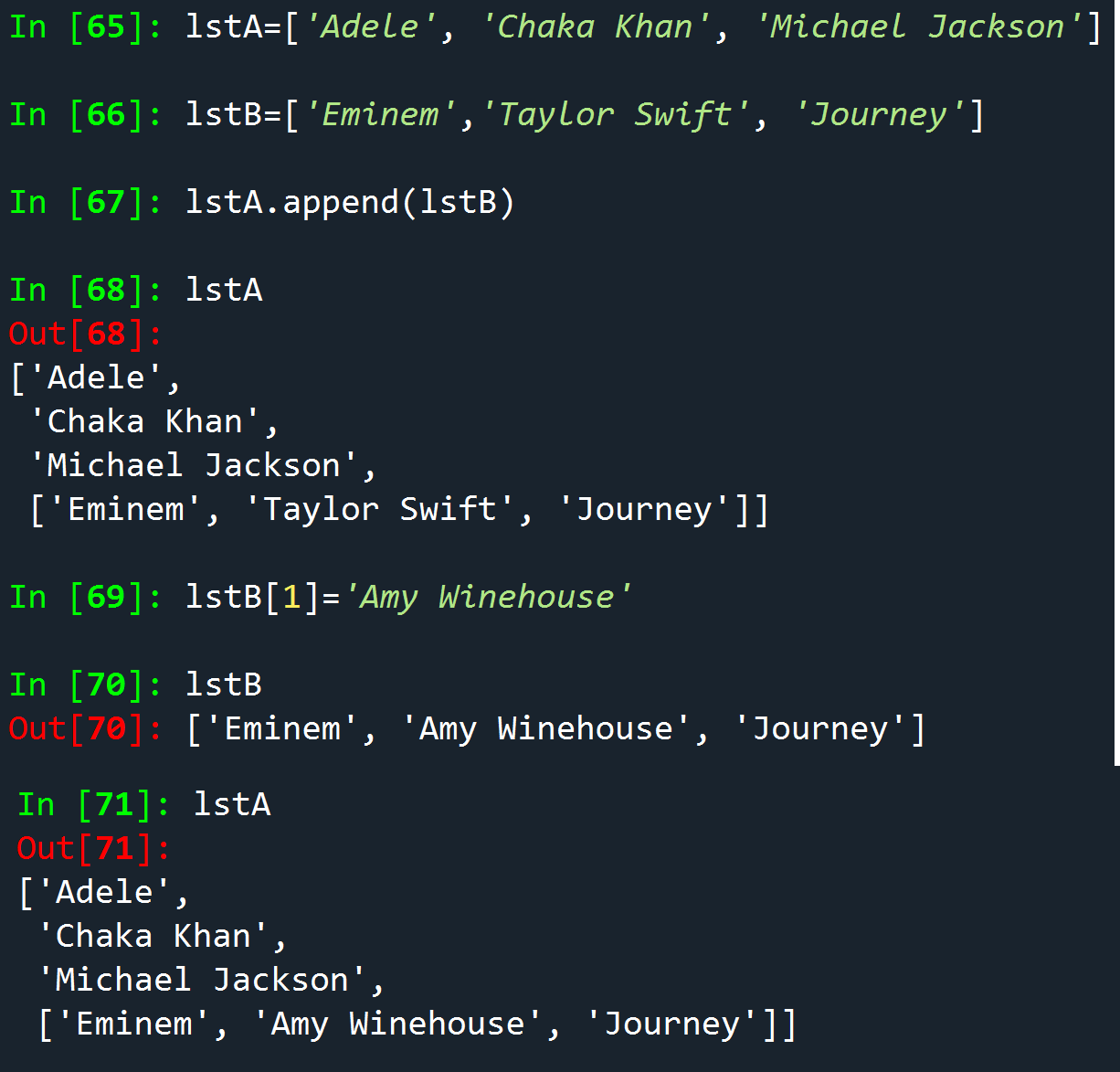


Figure 5 - Shared References (a)

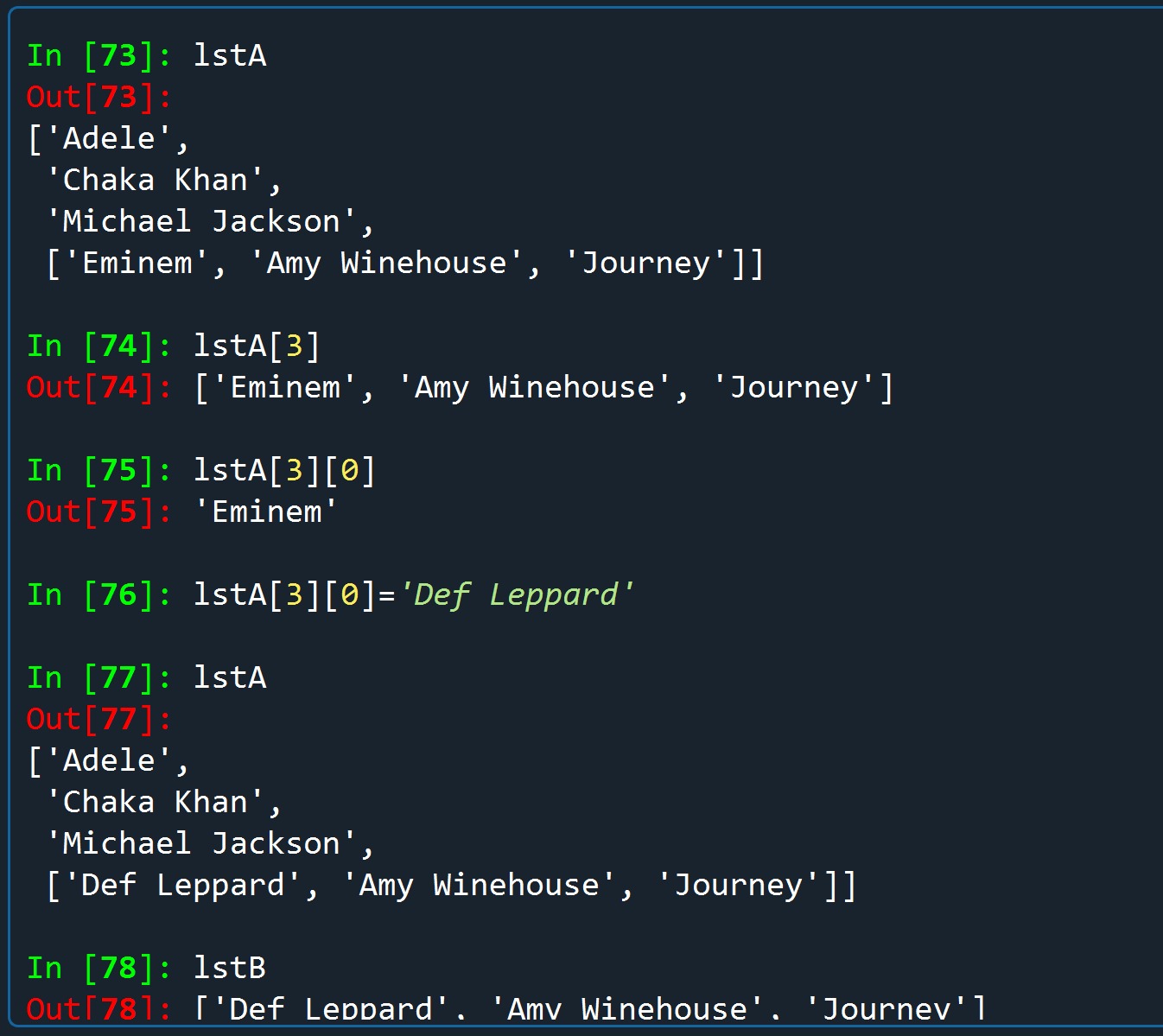


Figure 6 - Shared References (b)

## Dictionary

Dictionaries, like tuples and lists, are a collection of objects. And like lists, dictionaries are dynamic (can grow and shrink as needed) and can be nested.

Defining Dictionary

There a few important key points to remember when creating dictionaries:

1. A dictionary can’t contain multiple items of the same key
2. A key has to be immutable. It can be a string, number and tuples, to name a few.
3. Values does not have to be unique and can be both immutable and mutable data types.

To define a key, type a key name followed by a colon, followed by the key value and enclose everything in curly brackets. Refer to Figure 7 - Defining A Dictionary for an example.

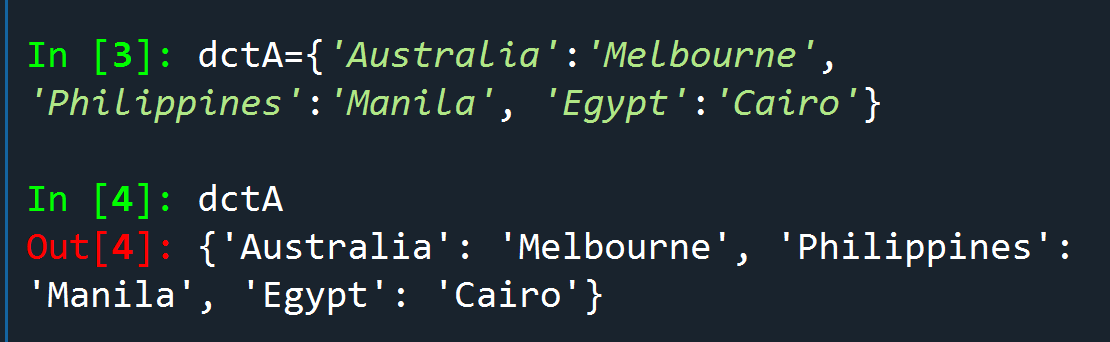


Figure 7 - Defining A Dictionary

Accessing Dictionary Elements

Unlike lists whose elements can be accessed via a position number or indexing, elements of a dictionary are accessed via **keys**.

Dictionaries are composed of key and value pairs; together they can also be called **items.**  A value can be directly accessed by specifying its corresponding key in square brackets (**[]**). See Figure 8 - Accessing via Key for an example

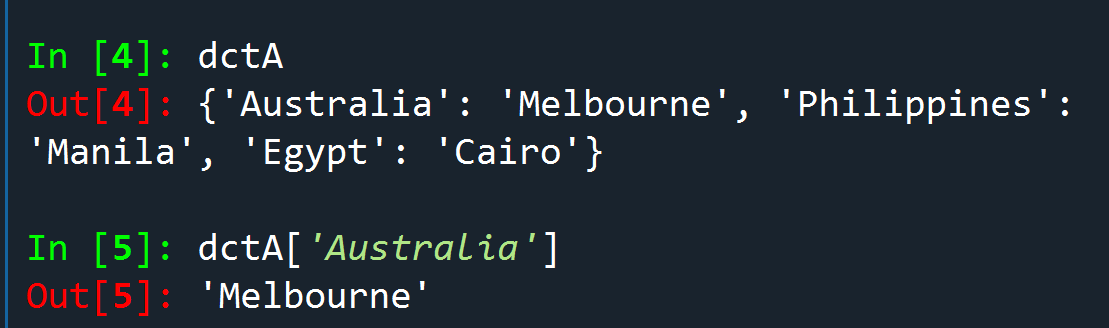


Figure 8 - Accessing via Key

Adding a new entry into the dictionary is just a matter of specifying a new key and value pair as shown in Figure 9 - Adding New Item.

****

Figure 9 - Adding New Item

Updating an existing value for a specific key looks the same as adding a new entry syntax wise but of course you specify an existing key and assign a new value instead.

To delete a item from a dictionary, I can use the **del()** function. Figure 10 - Delete item from dictionary illustrates this.

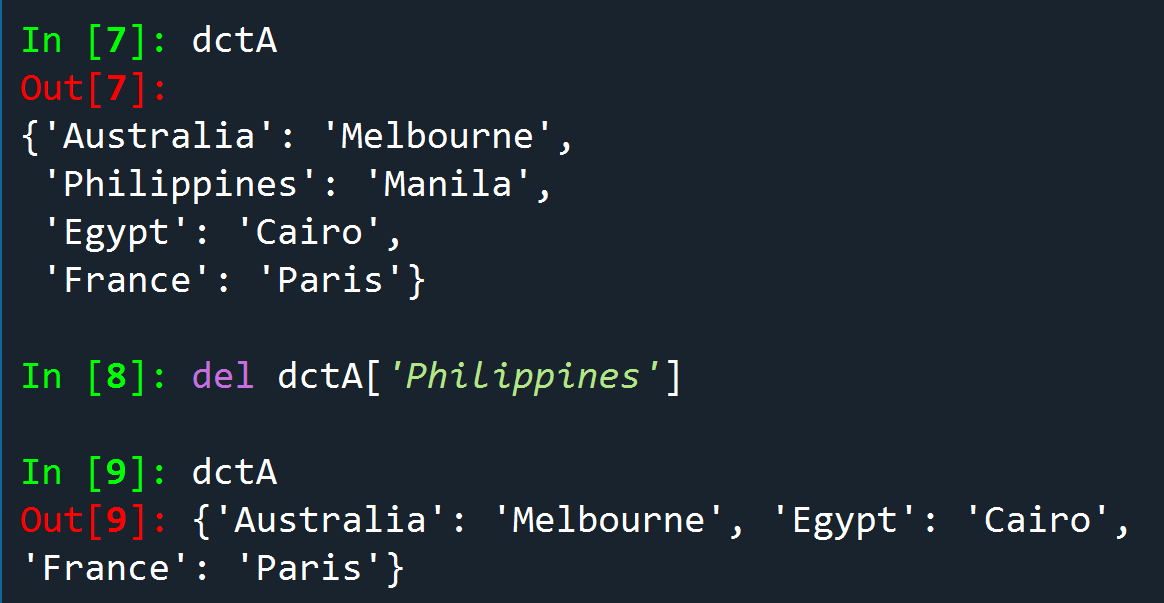


Figure 10 - Delete item from dictionary

Dictionary Views

Dictionary items can also be retrieved via **dictionary views**. Dictionary views are returned by the **keys()**, **values(),** and **items()** methods. Figure 11 - Dictionary Methods[[4]](#footnote-4) gives a brief explanation of the different dictionary methods for retrieving keys or values, or both.

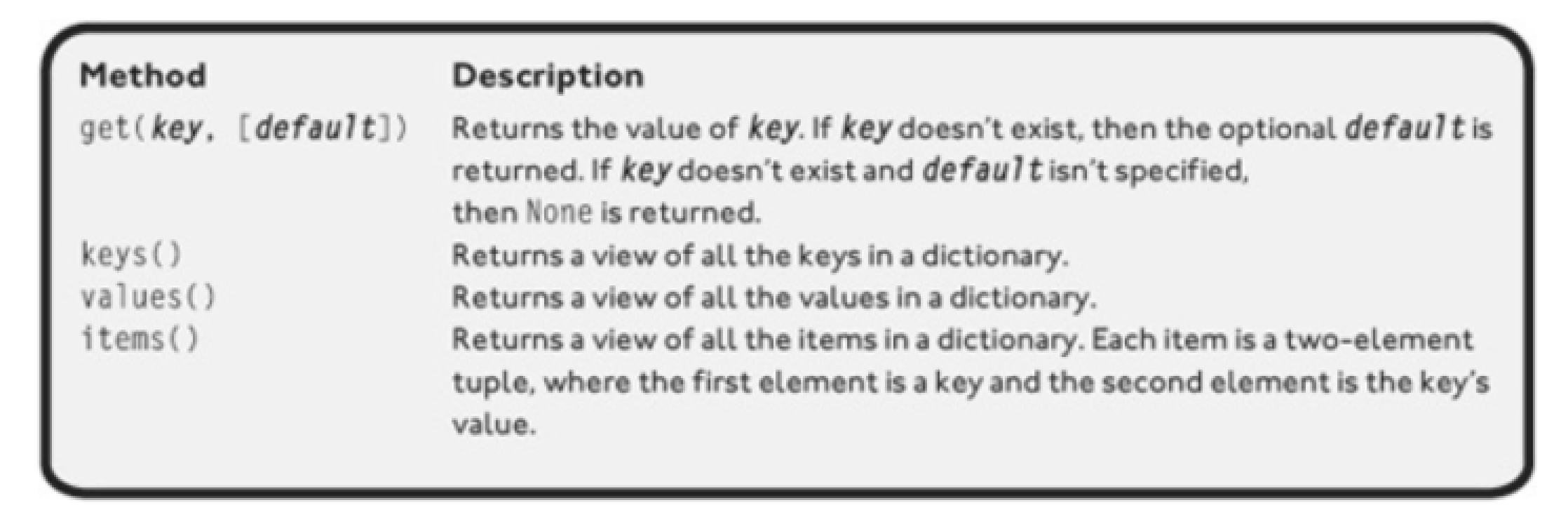


Figure 11 - Dictionary Methods

For a more comprehensive list of other built-in dictionary methods, this [website](https://www.w3schools.com/python/python_ref_dictionary.asp)[[5]](#footnote-5) is a good place to start.

# GitHub

Any decent sized company that has a software development team would have a need for a code repository and version control of their code and documentation. In the current company I work for, we use **GitLab** which is similar **GitHub.** At the heart of GitHub is Git. **Git** is a mature and actively maintained open source project developed by **Linus Trovards** back in 2005 (also creator or linux OS kernel). Go to this [link](https://www.atlassian.com/git/tutorials/what-is-git)[[6]](#footnote-6) for more info regarding Git. While Git is a command line tool, Github is a git repository hosting service that provides web interface for git. For more info regarding github, this [link](https://kinsta.com/knowledgebase/what-is-github/)[[7]](#footnote-7) is worth checking out.

For the homework, I signed up at github and uploaded my code and document for peer review.

# Separation of Concerns (SOC)

In this module, I gained new knowledge about a concept I have not heard before which is **Separations of Concern**, or **SOC**. SOC is a fundamental design principle in software development. The principle is simple – don’t write your program as one solid block; instead, break up the code into distinct sections where each section addresses a separate concern. In the class, the example provided was dividing the code into data (declaration of variables), processing and then presentation.

# Wrapping up the Homework

As expected, for the lab and homework I had to apply dictionary into the CDInventory program. I basically had to store my CD inventory as a dictionary, which in turn is stored inside a list. My dictionary consists of id, song and artist keys with corresponding values. I also had the chance of using structured error handling in the scenario where I am unable to open/load the file into memory. Please observe Figure 12 - Code Snippet. Lines 46-30 is where I included error handling in the event that my program encounters a problem trying to load a file. If it does, it will print a friendly message and go back to main menu. Lines 46 and 47 is where I store each line that was loaded from file as key value pairs into a dictionary. I then append each dictionary item into my main list.

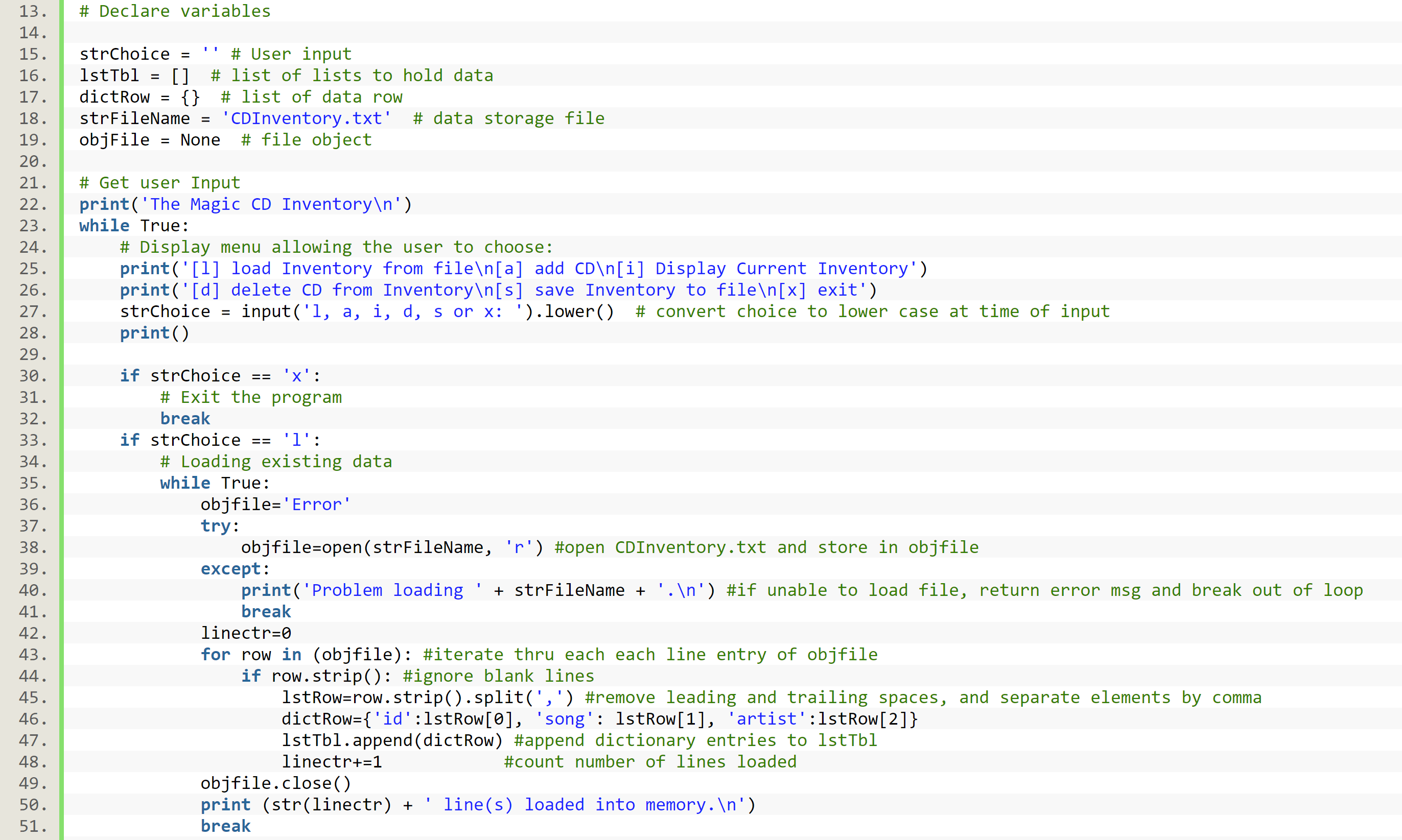


Figure 12 - Code Snippet

# Summary

Another challenging but still fun homework. I am introduced to dictionaries which, like lists, is also a collection of objects. But unlike lists, whose elements can be accessed via indexing or slicing, dictionary values are accessed via their corresponding keys. I learned how to unpack a list thru the asterisk (\*) overloaded operator as well as unpacking nested sequences (lists within lists). And the importance of understanding shared references – that is when severable variables refer to the same mutable value- are further ingrained into my subconscious. I also learned how to include structured error handling so that my program just doesn’t crash and burn on unexpected errors such as issues opening a text file. I gained a little bit more knowledge about a github and its backstory. Lastly, I have been presented a new concept called separation of concern (SOC). SOC is a fundamental design principle in computer science where code is broken into distinct sections such that each section addresses a specific concern.

## Appendix

1. Figure 1 - Load Data from File Into List Code (lab05\_b.py)
2. strChoice=None
3. lstTbl=[]
4. dictRow={}
5. strFilename='CDInventory.txt'
6. objFile=None
8. **print**('Read or Write File data.')
10. **while** strChoice!='q':
11. **print**('\n[a]add data to a list \n[w]write data to file\n[r]read data from file')
12. **print**('[d]display data\n[q]to quit')
13. strChoice=input('Please choose an action to perform: ').lower()
15. **if** strChoice=='q':
16. **break**
17. **if** strChoice=='a':
18. **print**('\*\*\*\*\*\*\*\*Add Data to List\*\*\*\*\*\*\*\*')
19. artist=input('Please enter artist\'s name: ')
20. song=input('Please enter song title: ')
21. dictRow={'arist': artist, 'song':song}
22. lstTbl.append(dictRow)
23. **elif** strChoice=='w':
24. **print**('\*\*\*\*\*\*\*\*Write Data to File\*\*\*\*\*\*\*\*')
25. counter=0
26. **if** (lstTbl):
27. **for** cd **in** (lstTbl):
28. counter+=1
29. strRow=''
30. **for** val **in** cd.values():
31. strRow+=val + ','
32. strRow=strRow[:-1]
33. objFile=open(strFilename,'a') #open file
34. objFile.write(strRow + '\n') # write new inventory stored in strRow
35. objFile.close() #close outFile.txt
36. **else**:
37. **print**('Nothing to save.')
38. **print**('Saved ' + str(counter) + ' new item(s) to the inventory.')
39. **elif** strChoice=='r':
40. **print**('\*\*\*\*\*\*\*\*Read Data from File\*\*\*\*\*\*\*')
41. objfile=open(strFilename, 'r')
42. **for** row **in** (objfile):
43. lstRow=row.strip().split(',')
44. dictRow={'arist': lstRow[0], 'song':lstRow[1]}
45. lstTbl.append(dictRow)
46. objfile.close()
47. **elif** strChoice=='d':
48. **print**('\*\*\*\*\*\*\*\*Display Data from List\*\*\*\*\*\*\*')
49. **print**('Artist,Title')
50. **if** (lstTbl):
51. **for** cd **in** (lstTbl):
52. strRow=''
53. **for** val **in** cd.values():
54. strRow+=str(val) + ','
55. **print** (strRow[:-1])
56. **else**:
57. **print** ('No new inventory.')
58. Figure 12 - Code Snippet (CDInventory.py)
59. #------------------------------------------#
60. # Title: CDInventory.py
61. # Desc: Add, Delete, Load and Display CD Inventory (Assignment 05)
62. # Change Log: (Who, When, What)
63. # Maria Dacutanan, 2020-Aug-09, Drafted pseudo code for the load data from file option
64. # Maria Dacutanan, 2020-Aug-09, Finalized code for loading data from file option
65. # Maria Dacutanan, 2020-Aug-10, Updated code for storing data into dictionary
66. # Maria Dacutanan, 2020-Aug-10, Updated code for displaying inventory - unpacking a dictionary
67. # Maria Dacutanan, 2020-Aug-10, Added code for deleting inventory
68. # Maria Dacutanan, 2020-Aug-10, Refined logic for deleting inventory so that duplicate ID values are also deleted
69. # Maria Dacutanan, 2020-Aug-11, Updated code for saving inventory into file
70. # Maria Dacutanan, 2020-Aug-12, Added check for file not existing
71. # Maria Dacutanan, 2020-Aug-12, Removed Debug Statements
72. # Maria Dacutanan, 2020-Aug-12, Updated comments
73. #------------------------------------------#
75. # Declare variables
77. strChoice = '' # User input
78. lstTbl = []  # list of lists to hold data
79. dictRow = {}  # list of data row
80. strFileName = 'CDInventory.txt'  # data storage file
81. objFile = None  # file object
83. # Get user Input
84. **print**('The Magic CD Inventory\n')
85. **while** True:
86. # Display menu allowing the user to choose:
87. **print**('[l] load Inventory from file\n[a] add CD\n[i] Display Current Inventory')
88. **print**('[d] delete CD from Inventory\n[s] save Inventory to file\n[x] exit')
89. strChoice = input('l, a, i, d, s or x: ').lower()  # convert choice to lower case at time of input
90. **print**()
92. **if** strChoice == 'x':
93. # Exit the program
94. **break**
95. **if** strChoice == 'l':
96. # Loading existing data
97. **while** True:
98. objfile='Error'
99. **try**:
100. objfile=open(strFileName, 'r') #open CDInventory.txt and store in objfile
101. **except**:
102. **print**('Problem loading ' + strFileName + '.\n') #if unable to load file, return error msg and break out of loop
103. **break**
104. linectr=0
105. **for** row **in** (objfile): #iterate thru each each line entry of objfile
106. **if** row.strip(): #ignore blank lines
107. lstRow=row.strip().split(',') #remove leading and trailing spaces, and separate elements by comma
108. dictRow={'id':lstRow[0], 'song': lstRow[1], 'artist':lstRow[2]}
109. lstTbl.append(dictRow) #append dictionary entries to lstTbl
110. linectr+=1            #count number of lines loaded
111. objfile.close()
112. **print** (str(linectr) + ' line(s) loaded into memory.\n')
113. **break**
114. **elif** strChoice == 'a':
115. # Add data to the inventory
116. **while** True: #user is re-prompted for null ID
117. strID = input('Enter an ID: ')
118. **if** (strID):
119. **break**
120. **while** True: #user is re-prompted for null CD Title
121. strTitle = input('Enter the CD\'s Title: ')
122. **if** (strTitle):
123. **break**
124. **while** True: #user is re-prompted for null Artist's Name
125. strArtist = input('Enter the Artist\'s Name: ')
126. **if** (strArtist):
127. **break**
128. dictRow = {'id':strID, 'song':strTitle, 'artist':strArtist} #store new data into dictionary variable
129. lstTbl.append(dictRow) #append to list lstTbl
130. **print** ()
131. **elif** strChoice == 'i':
132. # 3. Display the current data to the user each time the user wants to display the data
133. **print**('ID, CD Title, Artist')
134. **if** (lstTbl):
135. **for** row **in** lstTbl:
136. **print**(\*row.values(), sep = ',') #unpack list elements, separated by comma
137. **else**:
138. **print**('Inventory is empty.\n')
139. **print** ()
140. **elif** strChoice == 'd':
141. # Delete an entry from Inventory
142. **while** True:
143. keyID=input('Please enter an ID number to delete: ')
144. **if** (keyID): #checks for null ID
145. lstID=[]
146. delctr=0
147. **for** cd **in** lstTbl:
148. **for** row **in** cd['id']:
149. lstID.append(row) #store all IDs from lstTbl to a variable
151. **if** lstID.count(keyID) > 0: #check if ID provided by user exists in list of IDs
152. ctr=0
153. **while** ctr < len(lstTbl): #iterate thru lstTbl elements and match against user input
154. var=(lstTbl[ctr]['id']) #store ID from lstTbl
156. **if** str(var)==keyID: #compare value in var against user input
157. **del** lstTbl[int(ctr)] #delete element from lstTbl if found
158. delctr+=1 #counts number of deleted elements from lstTbl
159. ctr+-1 #decrease counter since lstTbl has shifted after deletion
160. **else**:
161. ctr+=1
162. **else**:
163. **print** ('Sorry, either inventory is empty or ID ' + str(keyID) +\
164. ' is not in the inventory.\n')
165. **break**
166. **print** (str(delctr) + ' row(s) deleted.\n')
167. **break**
168. **else**:
169. **continue**
170. **elif** strChoice == 's':
171. # Save the data to CDInventory.txt
172. counter=0
173. **if** (lstTbl): #check for null list
174. objFile = open(strFileName, 'a') #open file
175. **for** row **in** lstTbl: #iterate thru lstTbl
176. counter+=1
177. strRow = ''
178. **for** item **in** row.values(): #capture dictionary values
179. strRow += str(item) + ',' #store dictionary values to variable
180. strRow = strRow[:-1] + '\n'
181. objFile.write(strRow) #write stored dictionary values
182. objFile.close() # close file
183. **else**:
184. **print**('Inventory is empty.')
185. **print**('Saved ' + str(counter) + ' new item(s) to the inventory.\n')
186. **else**:
187. **print**('Please choose either l, a, i, d, s or x!\n')

1. Last retrieved 12-Aug-2020 [↑](#footnote-ref-1)
2. Last retrieved 12-Aug-2020 [↑](#footnote-ref-2)
3. Pages 137-139 of Python Programming for the Absolute Beginner, Third Edition by Michael Dawson [↑](#footnote-ref-3)
4. Screenshot taken from page 148 of Python from Programming for the Absolute Beginner, 3rd Edition by Michael Dawson textbook. [↑](#footnote-ref-4)
5. Last retrieved 12-Aug-2020 [↑](#footnote-ref-5)
6. Last retrieved 12-Aug-2020 [↑](#footnote-ref-6)
7. Last retrieved 12-Aug-2020 [↑](#footnote-ref-7)