Tiffany Hou  
2020 February 23  
IT FDN 100B  
Assignment 05

MOdule 5 Submission

# Overview

In this assignment, we update a starter code from Assignment 4 to use dictionaries. We also apply programming principles around separation of concerns, error handling, and GitHub.

# Summary of Key Concepts

## Dictionaries

*Dictionaries* are another type of sequence where elements are composed of *key:value* pairs. *Keys*cannot be changed, while *values* can.

## Separation of Concerns

*Separation of Concerns* (SoC) refers to the practice of grouping code into different sections so that each section addresses a specific issue/concern. This helps keep the code more organized, especially as the scripts grow more complex.

*Functions* are a way to help organize code. They group a set of statements together under a function name, which can be called at any time during the script (or even multiple times).

## Error Handling

Without *error handling*, our script could terminate if there is an invalid input or if it cannot execute a statement. Error handling builds in safeguards against that. They could be as simple as an if/else statement or a try/except statement[[1]](#footnote-1) (both of which are used in my script).

## Source Control

*GitHub* is just one company that provides hosting for a source control software called *Git*. Git tracks changes between iterations of code, which allows us to roll back the code to an earlier point/version needed, collaborate with teams, create “branches” to develop/test different features separate from the main branch, and so much more[[2]](#footnote-2).

These different points and versions are created using *commits*, which captures a snapshot of the code at that point in time[[3]](#footnote-3). As a general best practice, one should limit a commit to a single purpose. This makes it easier to “revert” or undo only a single change rather than a lot of them[[4]](#footnote-4).

# The Code

## Script Overview

This week, we take an existing starter code for the CD Inventory script (provided by Dirk Biesinger) and update it to use dictionaries as the inner sequence in a 2D-list and add two new features that:

1. load data from a saved file
2. delete an entry

Parts of the script not impacted by changing to a dictionary sequence in the inner list include:

* code for displaying the menu
* code for exiting the script
* code for displaying the current inventory
* code for error handling an invalid input in the menu

These portions of the script are not discussed below.

As usual, the code snippets below have been formatted using [PlanetB’s syntax highlighter](http://www.planetb.ca/projects/syntaxHighlighter/)[[5]](#footnote-5).

## GitHub Repository

My script is available in my GitHub repository here: <https://github.com/tiffhou/Assignment_05>

## Breaking Down The Script

### Header & Variables

A screenshot of a cell phone

Description automatically generated

Figure 1 - Header & Variables

As usual, we begin with our header. A rudimentary change log is here, but now that we have a GitHub repository set up, we can also add more details in our commit messages there for future iterations.

Following the header, we begin to see the principles of SoC coming into play as we declare our variables. Note that instead of a variable for an inner list row, we have a variable for a dictionary sequence (dicRow).

**Question:** What is the recommendation for declaring dictionary strings? Should I be setting up the structure? I’ve tried it with both just dicRow = {} and what I have above and wonder what the best practice is.

### Loading Data from a File

Originally marked with a TO DO comment, code was added so that data can be loaded from a text file.

A screenshot of a cell phone

Description automatically generated

Figure 2 – Code to Auto Generate the CD IF when Loading from a File

It’s important for IDs to be unique, so to guarantee this when loading from memory, we would either need to overwrite any existing CD entries or find a way enforce unique IDs.

To avoid overwriting, I decided to add the ability to autogenerate CD IDs. Existing CD IDs are overwritten with auto-generated ones, but not CD titles and artists. This would prevent duplicate CD IDs and since the CD IDs are not meaningful by themselves, I felt comfortable doing so.

In my script, IDs are generated by checking the length of the inventory table, then adding 1 to it. Just in case the last used ID number does not match the table length (e.g. if an entry had been deleted), we use an any statement[[6]](#footnote-6) within a while True loop to check to see if the ID number exists within each CD entry’s ID key:value pair. If the ID is in use, it increments by 1 until there is a value that is not in use. This also works if there is nothing in the table memory, since the len() function would just return 0, making our first ID 1.

A screenshot of a cell phone

Description automatically generated

Figure 3 - Code to load from a file

Now that we have our first CD ID ready, we can open the file (in read mode) and starting pulling out each line using a for loop. The strip() command removes the trailing newline escape operators at end of each line/row and the and split() command tells Python that each element is separated by commas. A potential bug for this is if a title or artist contains commas (I did not build in any error handling for this possibility).

**QUESTION:** is it generally recommended to reset the dicRow variable?

After we get the row, we can unpack it into our dictionary sequence and append it to the end of our 2D inventory list in memory. We then increment the CD ID by 1 in preparation for the next row. Once all the rows have been loaded, we close the file and alert the user.

### Adding A CD

While this function already existed, we had to update it now that we are using dictionary sequences and for our ID auto-generation.

A screenshot of a cell phone

Description automatically generated

Figure 4 - Code for Generating the CD ID for a New Entry

Here, we generate our ID using the method described above when loading the CD inventory from memory. Once we have the ID, we can assign it as the value for the ID key:value pair.

Note that it’s the same 3 lines of code to auto-generate the ID as above – this makes it a prime candidate for a function in future iterations!  
In case we want to update how we generate IDs, we would only need to update the function itself, rather than updating it in multiple places.

A screenshot of a cell phone

Description automatically generated

Figure 5 - Code for Title & Artist Inputs and appending the Dictionary Sequence to 2D List

Next, we collect and set the title and artist key:value pairs using inputs.

Once we have all 3 elements, we can append the dictionary sequence to our 2D inventory list.

### Deleting an Entry

A new portion of the script, this deals with deleting an entry from the inventory table list in memory.

A screenshot of a cell phone

Description automatically generated

Figure 6 - Code for Deleting an Entry

First, we need to collect the ID of the entry to be deleted via user input. We then use a try statement cast the input as an integer (which all valid IDs must be). If it cannot be cast as one, the except ValueError statement catches the error and alerts the user and returns to the menu.

If the input can be cast an integer, we once again use an any statement to check to see if the ID is in use (if not, it alerts the user and returns to the menu).

If it is a valid ID, it checks each inner dictionary sequence until it finds the one where the value in the ID key:value pair matches the input, informs displays to the user the ID, title, and artist of the entry being deleted, and deletes it from the outer inventory list using the remove() command.

### Saving to File

Our last elif statement, we had to modify it so we could save a dictionary to the file.

A screenshot of a cell phone

Description automatically generated

Figure 7 - Code for Saving to a File

To modify the script, we update line 105 to pull from the *value* portion of each key:value pair in the row (I also renamed the element from item to dicValue to better reflect this).

## Outputs

Screenshots of the output from both Spyder and Terminal are available in the Appendix.

# Summary

In this module, we put what we learned about dictionaries into action. We also started thinking more critically about how we write code by putting ideas around SoC, error handling, and source control into action.

## other Thoughts

### Loading Files

While testing, I noticed that you could repeatedly load from the same file, thus having duplicate CD entries/titles. You could address this by checking for a unique CD title/artist combination before allowing an entry to be imported.

### Saving to File

Currently, our script keeps the file open for quite a period of time while we prepare and write each individual line. I suppose there must be a way to prepare an string that represents the entire table, which we could write to the file in a single statement. However, I did not explore this particular avenue.

### Functions

I must say that it was painful to not use functions in this script. Having to code without using functions really emphasized their value and importance to me, especially when there were certain statements that I wanted to reuse (e.g. auto-generating IDs for both new entries and when loading existing entries). I definitely felt much more disorganized.

### GitHUb

I also regret not setting up my GitHub repository from the get-go. It would have been much easier write this documentation, since I would have made a commit after modifying each elif statement. Using an application like SourceTree would also display exactly what lines were modified in each commit.

# Appendix

## Listing – CDInventory.Py

1. #------------------------------------------#
2. # Title: CDInventory.py
3. # Desc: Starter Script for Assignment 05
4. # Change Log: (Who, When, What)
5. # DBiesinger, 2030-Jan-01, Created File
6. # THou, 2020-Feb-18, replaced lstRow with lstDicRow
7. # THou, 2020-Feb-20, added delete, load functions; modified save function
8. # THou, 2020-Feb-23, added autogenerating for CD ID
9. #------------------------------------------#
11. # Declare variables
12. strChoice = ''  # User input from menu
13. lstTbl = []  # list of dictionaries to hold data
14. dicRow = {'id': '', 'title': '', 'artist':''} # dictionary to hold CD entry
15. strFileName = 'CDInventory.txt'  # data storage file
16. objFile = None  # file object
17. strInputID = ''  # collect the CD ID from user input
18. intCDID = ''  # CD ID variable
19. strRow = '' # string variable for CD row
21. # Get user Input
22. **print**('The Magic CD Inventory\n')
23. **while** True:
24. # 1. Display menu allowing the user to choose:
25. **print**('\n')
26. **print**('[l] load Inventory from file\n[a] Add CD\n[i] Display Current Inventory')
27. **print**('[d] delete CD from Inventory\n[s] Save Inventory to file\n[x] exit')
28. strChoice = input('l, a, i, d, s or x: ').lower()  # convert choice to lower case at time of input
29. **print**('\n\n')

32. **if** strChoice == 'x':
33. # 5. Exit the program if the user chooses so
34. **print**('Goodbye')
35. **break**

38. **if** strChoice == 'l':
39. # load data from file; appends to existing data
40. #generate unique ID based on table length
41. intCDID = len(lstTbl) + 1 #proposed CD ID based on table length
42. **while** any(intCDID == row['id'] **for** row **in** lstTbl) **is** True: #check if ID already exists
43. intCDID = intCDID + 1 #increment until ID is unique
45. #load line from file
46. objFile = open(strFileName, 'r')
47. **for** row **in** objFile:
48. strRow = row.strip().split(sep=',')
49. dicRow = {'id': intCDID, 'title': strRow[1], 'artist': strRow[2]}
50. lstTbl.append(dicRow)
51. dicRow = {'id': '', 'title': '', 'artist':''}
52. intCDID = intCDID + 1
53. objFile.close()
54. **print**('Load complete')

57. **elif** strChoice == 'a':
58. # 2. Add data to the table (2d-list of dictionaries) each time the user wants to add data
59. #generate unique ID based on table length
60. intCDID = len(lstTbl) + 1
61. **while** any(intCDID == row['id'] **for** row **in** lstTbl) **is** True:
62. intCDID = intCDID + 1
63. dicRow['id'] = intCDID
64. **print**('ID: ', intCDID)
66. #get title and artist inputs
67. dicRow['title'] = input('Enter the CD\'s Title: ')
68. dicRow['artist'] = input('Enter the Artist\'s Name: ')
70. #append to 2D list
71. lstTbl.append(dicRow)
72. dicRow = {'id': '', 'title': '', 'artist':''}
73. **print**('Entry saved')

76. **elif** strChoice == 'i':
77. # 3. Display the current data to the user each time the user wants to display the data
78. **print**('ID, CD Title, Artist')
79. **for** row **in** lstTbl:
80. **print**(\*row.values(), sep = ', ')

83. **elif** strChoice == 'd':
84. #get id input from user, find matching entry, delete
85. strInputID = input('Enter the CD ID of the entry you want to delete: ')
86. **try**:
87. intCDID = int(strInputID)
88. **if** any(intCDID == row['id'] **for** row **in** lstTbl) **is** True: #check each row's ID value to see if intID exists
89. **for** row **in** lstTbl: #find and remove entry with matching ID
90. **if** intCDID == row['id']:
91. **print**('Removing entry:')
92. **print**(\*row.values(), sep = ', ')
93. lstTbl.remove(row)
94. **else**:
95. **print**('No such ID')
96. **except** ValueError:
97. **print**('Invalid input')

100. **elif** strChoice == 's':
101. # 4. Save the data to a text file CDInventory.txt if the user chooses so
102. objFile = open(strFileName, 'w') #overwrite file with memory table
103. **for** row **in** lstTbl:
104. strRow = ''
105. **for** dicValue **in** row.values():
106. strRow += str(dicValue) + ','
107. strRow = strRow[:-1] + '\n'
108. objFile.write(str(strRow))
109. objFile.close()
110. **print**('Save complete')

113. **else**:
114. **print**('Please choose either l, a, i, d, s or x!')

## REsults from Spyder

A screenshot of a cell phone

Description automatically generated

Spyder Output 1 - Adding an Entry

A screenshot of a social media post

Description automatically generated

Spyder Output 2 - Displaying and Deleting an Entry

A screenshot of a cell phone

Description automatically generated

Spyder Output 3 - Saving the Inventory to a File

A screenshot of a cell phone

Description automatically generated

Spyder Output 4 - Loading Entries from a File

A screenshot of a cell phone

Description automatically generated

Spyder Output 5 - Exiting the Script

## Results from Terminal

A screenshot of a cell phone

Description automatically generated

Terminal Output 1 - Adding a new Entry

A screenshot of a cell phone

Description automatically generated

Terminal Output 2 - Deleting an Entry

A screenshot of a cell phone

Description automatically generated

Terminal Output 3 - Saving Memory to File

A screenshot of a cell phone

Description automatically generated

Terminal Output 4 - Loading from File

A screenshot of a cell phone

Description automatically generated

Terminal Output 5 - Exiting the Script

1. https://docs.python.org/3.7/tutorial/errors.html, accessed 2020-Feb-23 [↑](#footnote-ref-1)
2. From personal experience using GitHub [↑](#footnote-ref-2)
3. https://developer.github.com/v3/git/commits/, accessed 2020-Feb-23 [↑](#footnote-ref-3)
4. https://blog.beanstalkapp.com/post/147799908084/commit-early-commit-often, accessed 2020-Feb-23 [↑](#footnote-ref-4)
5. Accessed 2020-Feb-23 [↑](#footnote-ref-5)
6. https://docs.python.org/3.7/library/functions.html#any, accessed 2020-Feb-23 [↑](#footnote-ref-6)