Michael List

February 24, 2020

Foundations of Programming: Python

Assignment 05

Tuples, Lists & Dictionaries

# Introduction

This document is an introduction to dictionaries and goes more into depth about lists. It then gives an overview on how to improve scripts with some common programming techniques and concludes with the coding assignment. For this coding assignment a starter script with established TODOs, the main assignment was to modify the script to use dictionaries rather than lists for the inner data type.

# Part 1 – Knowledge

Part 1 of this document explains key concepts learned in this module as outlined in the introduction.

## What is the difference between a Dictionary and a List?

Instead of being a sequence type like lists, dictionaries are mapping types. Rather than storing its data in sequence and using an index to identify the position of a value a dictionary uses a key to reference the values stored in it. Each key in a dictionary needs to be unique and can only be assigned to one value. This means that a dictionary consists of ‘key: value’ pairs. For both datatypes the values are mutable, however, the keys of a dictionary are immutable.

Dictionaries appear to introduce an increased complexity as it is a little trickier to access the values of a dictionary. To do so the values form the dictionary need to be extracted (which results in a view that appears like a list). This view then needs to be accessed or unpacked as well. See example below for Lab05B where the printing code section has been enhanced to display only the values of the stored dictionary:

1. #printing 2D list storing dictionaries
2. **elif** strChoice == 'd':
3. **print**('Artist, Title')
4. **for** row\_dic **in** lstTbl:
5. #reading only the values of the dictionary and storing it in a view variable
6. row\_dic\_values = row\_dic.values()
7. #unpacking the view that consists of the dictionary values
8. **print**(\*row\_dic\_values, sep=', ')

Listing - Accessing values of a dictionary

## What is the difference between an index and a key?

Index and key serve a similar purpose as an index identifies the position of a value in a sequence and a key identifies the value associated to the key stored in a dictionary. The big difference is that an index is automatically assigned by Python and starts with the number 0 while the programmer has the flexibility to choose and assign a key to each value of a dictionary. The one thing the programmer must keep in mind is that keys are immutable and cannot be changed as they are a unique identifier. This automatically excludes lists to be used as a key. However, tuples, strings or constants can be used as keys in a dictionary. If a tuple is used as keys in a dictionary the elements in the tuple itself need to be immutable as well.

## How do you read data from a file into a list?

There are multiple steps for reading data from a file into a list:

1. Read each row form the file,
2. For each row, use the split method to split the read string into individual listing elements (e.g. comma). Remove any white spaces
3. Store each read line of the text file in a list variable
4. Append lists to each other in another list data type (table)

See execution of reading data from a file into a list in the listing below as implemented in Lab05-A:

1. #opening and reading existing file. Storing results as 2D table in lstTbl variable
2. **elif** strChoice == 'r':
3. objFile = open(strFileName, 'r')
4. **for** row **in** objFile:
5. lstRow = row.strip().split(',')
6. lstTbl.append(lstRow)
7. objFile.close()
8. **print**(lstTbl)

Listing - Reading data from file into a list

## How do you read data from a file into a dictionary?

Reading data from a file into a dictionary rather than a list adds an additional layer of complexity as the programmer needs to define keys for each value that needs to be stored in the dictionary. An additional line needs to be added as highlighted in yellow in the below listing:

1. **elif** strChoice == 'r':
2. objFile = open(strFileName, 'r')
3. **for** row **in** objFile:
4. lstRow = row.strip().split(',')
5. dicRow = {'artist': lstRow[0], 'album': lstRow[1]}
6. lstTbl.append(dicRow)
7. objFile.close()
8. **print**(lstTbl)

Listing - Reading data from file into a dictionary

This additional line defines the keys in the dictionary and is referencing the values of each key: value pair by its index location in the list variable ‘lstRow’.

## Why is it making sense to organize data in a 2-dimensional way?

Organizing data in 2-dimensional way allows the programmer to organize more complex data in an easy way. A 2-dimensional data structure can be interpreted as a table with multiple rows.

## What is the programming pattern “Separation of Concerns”?

Separation of Concerns (SOC) is a programming practice that helps the programmer to keep the increasing complexity of scripts organized by separating the program in different sections of concerns. Each of these sections will address an individual concern. SOC can be used for any kind of application and any programming language. A simplistic approach to this concept is splitting the program into three sections:

1. Data (declaring variables and constants)
2. Processing (data manipulation)
3. Presentation (input/output)

An important part to achieve great SOC is the usage of functions, see more below in the functions section. The programmer should always strive for as much as SOC as possible.

## How would you use a function to organize your code?

Functions are a great improvement to coding and can be very powerful. With functions it is possible to group together a set of statements that then can be called by their function name later. Because of this behavior functions can be used to organize code and allow the programmer to achieve a better separation of concerns. For example, the functions themselves can be defined in the processing section of the program and then be called in the presentation section later.

## Why is a script template useful?

Script templates are useful as they allow the programmer to create their own, customized templates for new script files. So far, we have been creating the header of a script file from scratch each time a new .py file has been created. With a customized template the programmer can save time and establish consistency across their script files. Additional comments in the template help to establish a separation of concerns. See the below listing for the new template:

1. #------------------------------------------#
2. # Title: <Enter Title here>.py
3. # Desc: <Enter Description here>
4. # Change Log: (Who, When, What)
5. # MList, 2020-Feb-01, Created File
6. #------------------------------------------#
8. #---------- DATA ----------#
10. #---------- PROCESSING ----------#
12. #---------- PRESENTATION (Input/Output) (I/O) ----------#

Listing - Customized .py Template

## Why is error handling (try-except) useful?

The default behavior for error handling is to stop the program if an error occurs. However, there are scenarios where the programmer can expect errors. Rather than having the program crash the programmer can implement try-except in Python. Using this error handling method allows the program to continue by defining an action if the error occurs. See the below listing for a try-except error handling implemented in this week’s assignment. In this example the try-except is used to open a potentially existing text file. Without error handling the program would crash if the file does not exist. With the implemented error handling the program will instead print out an informational message:

1. **try**:
2. with open(strFileName, 'r') as objFile:
3. **for** line **in** objFile:
4. lstRow = line.strip().split(',')
5. dicRow = {'ID': int(lstRow[0]), 'CD Title': lstRow[1], 'Artist': lstRow[2]}
6. lstTbl.append(dicRow)
7. **print**('\nGood news! There is already a CDinventory.txt file. \nThe existing file has been loaded and any saved changes will overrite the existing file.')
8. **except** IOError: **print**('\nThere is currently no existing inventory file - A new Inventory File will be creating when saving.\n'

Listing - try-except implemented in Assignment 5

## What is GitHub and why is it used?

GitHub is a cloud platform for git. Git itself is a very popular version control system. A version control system tracks code changes by individual programmers and gives programmers the ability to roll back changes. Git makes it easy for multiple programmers to work on the same project and keeps the changes and files in sync by using a central repository. Before Git this was accomplished with shared network drives which caused a lot of administrative work and had a lot of limitation. Git removes the headaches of administrating code files. Git can be just employed on a local network; however, it is very popular to use a cloud-based solution. The most common cloud-based solution for Git is GitHub. The mascot of GitHub is the Octocat.

# Part 2 – Address Book Python Script

Part 2 of this assignment was to modify the CD inventory program from last week’s assignment. However, instead of modifying our own code the students have been provided with a starter code to gain working experience with unfamiliar code. The starter code included the following TODOs:

* # TODO replace list of lists with list of dicts
* # TODO Add the functionality of loading existing data
* # TODO Add functionality of deleting an entry

The students were also instructed to not use functions for this assignment to experience the difference in the Separation of Concerns.

The big challenge for me in this assignment was adding the delete functionality as the task was to delete a full dictionary from a 2D list by referencing the index of the dictionary that needs to be removed. I accomplished this by introducing a counter. However, even after I was able to implement a successful delete option, I encountered the following issue:

1. When deleting the item, I was successfully deleting it from the 2D list that was loaded in the memory. However, my implementation did not delete the requested dictionary from the text file. Even worse, when saving the changes after deleting the dictionary the existing code appended the 2D list in the memory to the existing text file. This did not only not delete the entry in the text file, it also introduced duplicates for the dictionaries that were not deleted.
2. Based on the current flow of the starter script it seemed overly complicated to solve this issue as I got the impression that I can either delete the entry from the text file or delete it from the 2D list stored in memory.
3. A good solution in my opinion was to make the reading of an existing inventory non-optional. As soon as the script starts it will check if an existing text file already exists. If it exists, the content will be loaded into the 2D list in memory. By implementing a try-except error handling in case the file does not exist the script can be run without an existing CDInventory.txt file. I then went ahead and changed the save/writing option from append to write, which will fully overwrite the existing text file with information stored in the 2D list in memory. With this solution the script does not add potential duplicates when saving and the script can manipulate the full 2D table in memory.
4. The only concern I can see is that this might be not the most efficient way to completely rewrite the whole text when saving any edits. However, this should not be a problem with this small file.

The other TODOs were easier to implement as they mirrored the Labs of this module.

The other struggle was to implement a good separation of concerns as the whole program is just a while loop with inputs, processing statements and outputs within the loop. I can see how e.g. defining a save function in the processing section and then calling that function in the presentation section will introduce much needed clarity in the code. For SoC improvement some variables have been declared in the DATA section, however, it is currently unclear how much sense this makes in Python where new variables will be declared while using them. This is something I will raise during class to get a better understanding of the subject. In general, adding SoC to a script without using functions seems difficult.

All functions of the program have been tested and the text file validated to ensure proper functionality of the program. Figure 1 and 2 show the execution in Anaconda Prompt and the result text file.

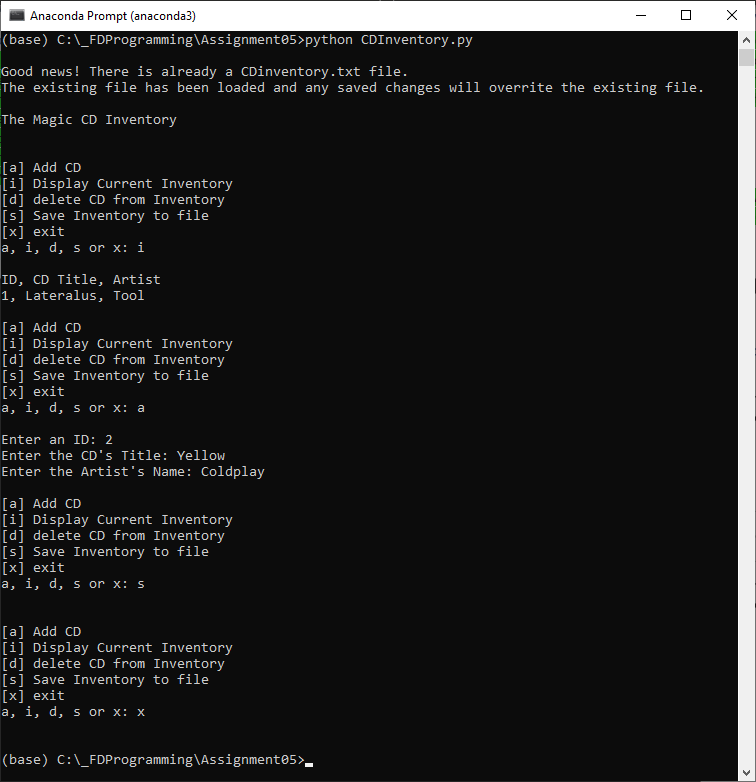


Figure - CDInventory.py in Anaconda Prompt

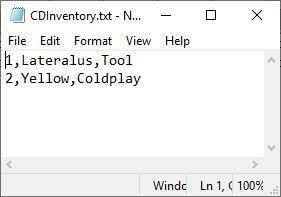


Figure - CDIventory.txt result after Anaconda Prompt execution

Figure 3 and 4 on the next page show the execution of the script in Spyder and the result text file after adding a new CD as well as removing an existing entry.

# 

Figure - Adding CD and removing CD in Spyder

# 

Figure - CDInventory.txt after Spyder execution

# Appendix

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. # MList, 2020-Feb-23, Modified File by replacing inner lists with dictiionaries
7. # MList, 2020-Feb-24 Added delete functionality
8. # Mlist, 2020-Feb-24 Made file loading mandatory to prevent duplicates
9. # Mlist, 2020-Feb-24 Added error message in case deleted CD does not exist
10. # Mlist, 2020-Feb-24 Cleaning up code and adding comments
11. # Mlist, 2020-Feb-24 Improving SoC
12. #------------------------------------------#
14. #---------- DATA ----------#
16. # Declare variabls
17. strChoice = '' # User input
18. lstTbl = []  # list of dictionaries to hold data
19. lstRow = []  # Row list variable for storing text file infomration while reading
20. dicRow = {}  # dictionary of data row
21. strFileName = 'CDInventory.txt'  # data storage file
22. objFile = None  # file object
23. Flag = True # flag to identify no found match for delete functionality
25. #---------- PROCESSING ----------#
27. # Attempting to load current iventory upon startup. Loading inventory from existing CDInventory.txt file has been made non-optional to avoid writing duplicates when deleting/saving edits with append
28. **try**:
29. with open(strFileName, 'r') as objFile:
30. **for** line **in** objFile:
31. lstRow = line.strip().split(',')
32. dicRow = {'ID': int(lstRow[0]), 'CD Title': lstRow[1], 'Artist': lstRow[2]}
33. lstTbl.append(dicRow)
34. **print**('\nGood news! There is already a CDinventory.txt file. \nThe existing file has been loaded and any saved changes will overrite the existing file.')
35. **except** IOError:
36. **print**('\nThere is currently no existing inventory file - A new Inventory File will be creating when saving.\n')
38. #---------- PRESENTATION (Input/Output) (I/O) ----------#
40. **print**('\nThe Magic CD Inventory\n')
41. **while** True:
42. # 1. Display menu allowing the user to choose:
43. **print**('\n[a] Add CD\n[i] Display Current Inventory')
44. **print**('[d] delete CD from Inventory\n[s] Save Inventory to file\n[x] exit')
45. strChoice = input('a, i, d, s or x: ').lower()  # convert choice to lower case at time of input
46. **print**()
48. # Exit the program if the user chooses so
49. **if** strChoice == 'x':
50. **break**
52. # Add data to the table (2d-list) each time the user wants to add data
53. **elif** strChoice == 'a':  # no elif necessary, as this code is only reached if strChoice is not 'exit'
54. # Ask for CD input
55. strID = input('Enter an ID: ')
56. strTitle = input('Enter the CD\'s Title: ')
57. strArtist = input('Enter the Artist\'s Name: ')
58. intID = int(strID)
59. # casting input into dictionary
60. dicRow = {'ID': intID, 'CD Title': strTitle, 'Artist': strArtist}
61. lstTbl.append(dicRow)
62. # Display the current data to the user each time the user wants to display the data
63. **elif** strChoice == 'i':
64. **print**('ID, CD Title, Artist')
65. **for** row\_dic **in** lstTbl:
66. row\_dic\_values = row\_dic.values()
67. **print**(\*row\_dic\_values, sep = ', ')
69. # Delete functionality
70. **elif** strChoice == 'd':
71. # Ask for entry to delete. This will need to the value of the album because artist is not a unique identifier and their could be potential duplicate IDs.
72. CD = input('Which CD would you like to delete?: ')
73. # Counter starting at 0 for cycling through the individual dictionaries. Need to reset counter to 0 in case user alread used delete functionality
74. counter = 0
75. # Defining a flag as false in case no match will be identified to print an error message
76. Flag = False
77. # For each dictionary row in the 2D list
78. **for** row **in** lstTbl:
79. # Assinging values of dictionary row to a variable
80. row\_values = row.values()
81. # Checking if input matches any values of dictionary row
82. **if** CD **in** row\_values:
83. # If match has been indentified use counter variable as index location for delete function in the 2D list
84. **del** lstTbl[counter]
85. **print** ('The requested CD has been deleted. Do not forget to save your changes!')
86. # Setting Flag to true so error message does not get printed
87. Flag = True
88. **break**
89. **else**:
90. # If the input does not match the values of the dictionary row increase the counter by 1 and move to the next dictionary row in the 2D list
91. counter += 1
92. # If no match has been found and deleted the flag will still be False and an error message will be printed
93. **if** Flag **is** False:
94. **print** ('The requested CD does not exist in the inventory txt file')
96. # Save the data to a text file CDInventory.txt if the user chooses so
97. **elif** strChoice == 's':
98. write\_string = ''
99. **for** row\_dic **in** lstTbl:
100. row\_dic\_values = row\_dic.values()
101. **for** item **in** row\_dic\_values:
102. write\_string += str(item) + ','
103. write\_string = write\_string [:-1] + '\n'
104. objFile = open(strFileName, 'w')
105. objFile.write(write\_string)
106. objFile.close()
107. **else**:
108. **print**('Please choose either a, i, d, s or x!')

Listing - CDInventory.py