Name: Matthew Penny

Date: 2021 AUG 06

Course: Foundations Of Programming: Python

Assignment: Assignment 04

**Lists in Depth, Dictionaries, and Code Organization**

**Introduction**

**Lessons Learned and Concepts of Interest**

As a requirement for this module, I reviewed all videos and content from the module 05 section here: https://saravji.github.io/saravjis\_hut/FDN\_Prog/Modules.html (external site)

I also read chapter five of “Python Programming, Third Edition – For the Absolute Beginner”, reviewed three web pages that detailed examples of content we were reviewing and watched another recommended/associated youtube video. After covering these materials, I am able to address points of interest laid out in the course materials:

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

This can easily be answered by describing which each one of these are. A list is a sequence of items; the items are singular and have a corresponding numerical index (though this is hidden by default and is mostly used as a means to access items at a given index). For example:

myLst= [“Gregory”,”Lenna”,”Boris”]

holds three separate values with corresponding index values of 0, 1, and 2. Lists also have their own series of options and functions such as sort(), pop(), clear(), append() and more that work with the specific way it stores data. Dictionaries on the other hand are PAIRS of data that have a “key” and an associated value. For example:

mySongs = {“Moby”:”Everloving”,”Metallica”,”Blackened”}

Holds two entries: “Moby” and “Metallica” as keys, and associated values of “Everloving” and “Blackened” respectively. These values can be called or references by their associated keys.

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

An index is an automatically assigned numerical value that corresponds to an item in a list, whereas a key is manually specified by a user or otherwise. Keys can also be ANY kind of immutable data; tuples, strings, and numbers for example whereas index values are an automatically generated set of numbers that can’t really be substituted with anything else. Like with index numbers though, you can fetch the value paired to a key by simply supplying the key.

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

This is done using the open() function, as we’ve done in the past, except we pass the “r” (read) option to the function. Below is an example pulled from code I wrote for one of our labs in this module:

elif strChoice == 'r':

# File to print

memList=[]

objOne = open(strFileName, "r")

fileLines = objOne.readlines()

for line in fileLines:

line = line.strip()

memList.append(line)

objOne.close()

In this case, I create an object to read from by passing a file stored in the strFileName variable along with the “r” flag. I had trouble getting it to work in the way Professor Besinger described it (it would always error out), so I used the readlines() method to fetch each line of text in the file before looping through the lines, stripping off trailing characters, and using the append() function to attach it to memList.

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

This is again done using the open() method with the read (“r”) option set. However, it works a bit differently compared to a list:

Data contined in the myFile.txt:

AK, Juneau

WA, Seattle

AZ, Phoenix

objFile = open(myFile.txt, ‘r’)

for row in objFile

line = line.strip().split(',')

dicRow = {‘state’ : line[0], ‘city’ : line[1]}

lstTbl.append(dicRow)

This code would create three dictionary objects with the keys AK, WA, AZ and the associated values of Juneau, Seattle, and Phoenix. It then appends it to a list object that holds the list of dictionaries. It does this by iterating through each row in the text file, using strip to remove any trailing characters such as a newline, splits the lines by the “,” character, and then appends the separate objects at line[0] and line[1] to the key and value for each dictionary pair before it’s stored in the main list.

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

This allows a greater breadth of data to be stored an accessed. A simple way to think of a 2d dimensional object like a list of lists or a list of tuples, etc is an excel spreadsheet. A single dimensional data object would only be able to represent a single column or row, limiting the data we could store. By having a two-dimensional data structure, we can have a series of columns or rows. For example, we could indeed create 12 individual lists and work with them. But, this would be cumbersome and difficult. Instead, we could place all 12 of these lists in another list and have a single 2d object to work with and manage.

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

Simply put, separation of concerns involves separating code into distinct sections whereas each section addresses a separate concern. So for example, separating code out into sections like:

'data'

We can declare variables here, for example

'processing'

We can process and manipulate data here for example

'output'

This is where we can display or present the data we have put in and manipulated above.

This makes code not only more readable, but overall more well organized and simple to update.

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

Methods also help in the above regard as well, as they can remove excess code from your main script and store it away from the main script body where it can be called. For example, we could utilize something like a block of code in our main body to complete something, or pass it to a function on a single line:

Def dateFormatter()

dateReturn = strMonth + ‘/’ + strDay + ‘/’ + strYear

return dateReturn

This would return a formatted string (assuming you passed it the correct data) using only a single line and function call.

**Why is a script template useful?**

Script templates save time and can be used to generate headers for example that can include data such as versioning, actions taken by developers, a name, and more as required. Rather than having to manually create or copy it over every time you make a script, IDEs like Spyder have a way of allowing you to create your own to be used whenever you like:

tools>preferences>editor>advanced settings>”edit template for new modules”

Here, you can create a template of your choosing that will be appended to the beginning of any new file opened – saving you time and effort while also providing context and data for others looking at your script.

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

Error handling can be useful to avoid issues such as a program crashing or stopping. Ordinarily, if python encounters an error, the program terminates and an error message is produced. This can be avoided altogether by using a try-except statement where you “try” a line of code or a statement and raise an error/exception message if an actual error occurs. Videos and documentation in this module gave us a simple error one may encounter – a “divide by zero” error. Since this is mathematically impossible, the program will crash and display an error. But, a try statement can catch this beforehand and produce an error while keeping the program functioning:

retValue = Error

try:

retValue = numberA/numberB

except:

print("An error occurred when dividing {} by {}'.format(numberA,numberB)

return retValue

In this case, the demonstration given is a function that tries to divide two numbers. If successful, it assigns and returns a value. If a zero is supplied, a default value of error is returned and the code in the except block is run informing the user that their entry is not valid.

**What is GitHub and why is it used?**

Github is described as “The largest software development control platform”. It implements version control as well and uses GIT. This takes a lot of the work out of software development and integrates it into git software and the github platform. For example, things such as versioning, bug tracking, integration, and sharing code used to be done manually and typically required human intervention and management. Github along with git automate and centralize most of this, enabling easy code sharing, updating, versioning, and any form of issue tracking.

**What is GitHub’s mascot**

Github’s mascot is an anthropomorphic “octopus-cat” designed by illustrator Simon Oxley.

**Labs and Examples Covered**

As noted prior, we were given several labs to cover from module 05, here: https://saravji.github.io/saravjis\_hut/FDN\_Prog/Modules.html (external site)

I will discuss in detail these labs below and show output from them.

For the first Lab, we had to create a full program from a template provided. It can perform four functions:

* Add data to a 2d data structure, namely a musical artist and album name
* Write the 2d data object to a file
* Read data from a file to a 2d data object
* Display the data in the 2d datatable object
* Exit the program.

A picture containing text, screenshot, monitor, screen

Description automatically generated

**Fig01. The program’s first three options.**

This exercise proved a bit challenging and I tried a few things before settling on a method. In the “write” option, I first create an empty string that’ll have data written to it. I then take lstTbl (the list of lists) and write the items at index 0 and 1 (the artist names and albums) to the string with a comma in between and a newline character following. I then pass this set of strings to the “write” option which writes the text to the file as needed in the expected format:

Graphical user interface, text, application

Description automatically generated

**Fig 02. The contents of lstTbl (list of lists) written to a text file**

Next, we have the ‘r’ selection that does the inverse. It loads the above text into a list in memory. First, I declare an empty list that will be populated. I then open the file in “read” mode with the “r” switch and then use the readlines() method to load each line into the fileLines object. From here, I loop through it by first using strip() to get rid of the newline character and then use the append() method to attach it to the list before closing the file.

Lastly, we have the display (“d”) option logic flow. This simply prints a header row and then unloads each row in the list of lists (using \*args) and includes a separator argument.

Text

Description automatically generated

**Fig 03. The top of the script with the “a” option and variable declarations.**

Apologies for presenting the top half later, as I took the screenshots out of order. Anyway, here we have template code that was already provided with added code in the if strChoice == ‘a’ block. This is pretty simple: I use the input function to ask the user for an artist and an album name. I entered “Moby” “Play” for my first choice and “Rush” “2112” as my second. The append() method is used to add these to the lstRow list before that list is appended to the main list (lstTbl). The lstRow list is then set back to empty again before the loop repeats.

Next, we modified this script to use a dictionary for the inner 2d data object instead of a list. Directions were a bit vague on which functions to cover, so I covered options “d” and “a” since those were demonstrated.

A picture containing text, indoor, monitor, screenshot

Description automatically generated

**Fig 04. Re-written display section showing output**

This section had to be redone as the format of a dictionary object is different than that of a list and the \*args unpacking method doesn’t seem to work with this data type. I got around this by replicating the unloading method that Professor Besigner demonstrated in his video documentation. In this case, we print a vertical header, then a separator to match the vertical format of the data. I then use nested loops to access the dictionary object inside the lstTbl list and use the .items() method to retrieve the key/value pairs before printing just the values out. I follow this with a header to separate the contents of each dictionary object I pull from the master list.

Text

Description automatically generated

**Fig 05. Re-written “a” section and some initial variables shown, data is being added on the left pane**

For this rewrite, I needed to throw away all my old .append() statements since they no longer worked.To get around this, I specified keys manually for each value we would store in the dictionary object with input statements on the right being assigned as the associated values. After that, I appendthe dictRow dictionary to the master list and then set dictRow to empty for when the user enters “a” again. I kept a print statement to display the master list containing the dictionary objects.

Lastly, we had the final project/assignment for module 05. This is where things got a bit tricky again, but I managed to revamp the script with a list of dictionaries.

Like the previous exercises, this broke existing functionality and it had to be updated. For the first item, I needed to rewrite the ‘l’ method that reads in a saved file. I did this by creating a file object in conjunction with the “r” operator for read and then manipulated each line in a loop. This required the double usage of the .strip() and .split() methods to give me back three items that I then assign to the dictionary object:

if strChoice == 'l':

objOne = open(strFileName, "r")

for line in objOne:

line = line.strip().split(',')

dicRow = {'ID' : line[0], 'artist' : line[1], 'album' : line[2]}

lstTbl.append(dicRow)

dicRow = {}

objOne.close()

I then append it to the list containing the dictionary objects and clear the row out for future usage.

Next, we have the add option updated to work with dictionaries. This was probably the simplest change and I just created a dicRow dictionary entry directly and appended it to the master list, before clearing it again for future use.

dicRow['ID'] = int(input('Enter an ID: '))

dicRow['artist'] = input('Enter the Artist\'s Name: ')

dicRow['album'] = input('Enter the CD\'s Title: ')

lstTbl.append(dicRow)

dicRow = {}

After that, I updated the “display” option to work again:

elif strChoice == 'i':

# 3. Display the current data to the user each time the user wants to display the data

print('ID,Artist Name,Album')

print('-------')

for row in lstTbl:

strID = row['ID']

strArt = row['artist']

strAlbum = row['album']

strDisp = str(strID) + ',' + strArt + ',' + strAlbum

print(strDisp)

This simply prints a header, a separator, and then extracts all three values from the dictionary, stores them in strings, and then displays them with formatting (as well as converting the ID to a string as well). Next, I wrote the delete option:

print("enter the number ID of the entry you wish to delete: ")

delChoice = int(input('Enter ID: '))

for row in lstTbl:

if row['ID'] == delChoice:

del row['ID']

del row['artist']

del row['album']

while {} in lstTbl:

lstTbl.remove({})

This ended up being the hardest as the “del” command as well as the clear.() command both leave an empty dictionary item after deleting the contents. This previously had a .clear() method usage as well, but they both produced the same issue. That’s where the additional while loop comes in. While an empty dictionary item exists in the dictionary list (which will always be present after the del statements) – then we use the .remove() method to get rid of the empty dictionary object.

Lastly, we had the save to file option to rework:

objFile = open(strFileName, 'a')

for row in lstTbl:

strID = row['ID']

strArt = row['artist']

strAlbum = row['album']

strFinal = str(strID) + ',' + strArt + ',' + strAlbum + '\n'

objFile.write(strFinal)

objFile.close()

We open a file stored in the strFileName variable with the append option and then read each dictionary object from the list with a for loop. Here, I mapped each part of the dictionary to a string and then created a formatted string object to write out to the file before closing it.

**Summary**

This module did a deeper dive into lists and dictionaries as well as how they interact as two-dimensional data objects. Some challenges arose in terms of adding and removing data gracefully from files, but re-checking my work and going over course content a second time was a great help. I struggled far less with the final project of the module versus the labs beforehand and now have a deeper understanding of how they work. Other concepts covered such as Github and code organization were also applicable to my work on the job and will be important in how I write scripts in a structured, understandable way. I can’t wait to cover functions, as that’ll make my code even easier to read.

**Appendix**

Text

Description automatically generated

***Fig 06. Adding two entries using ‘a’ option, running in spyder. The list table is printed out for validation, this line (38) is removed in the final script***

A picture containing text, screenshot, screen, computer

Description automatically generated

***Fig 07. Running the ‘I’ and ‘s’ options in spyder***

A picture containing text, screenshot, monitor, indoor

Description automatically generated

***Fig 08. Delete option being run on index item 1, leaving just the other item added.***

Text

Description automatically generated

***Fig 09. Loading data back in, duplicates are tolerated due to the ID being loaded in as strings instead of int***

Text

Description automatically generated

***Fig 10. Adding data via ‘a’ option in console***

Text

Description automatically generated

***Fig 11. Running I, s, d options in console***

Text

Description automatically generated

***Fig 12. Running l option in console and re-adding data that was saved previously.***