Bramble Eagle-Jack

2021-02-09

Foundations Of Programming

Assignment 5 Knowledge Document

Dictionaries

# Introduction

In this week’s assignment we’re asked to review materials that cover new topics like dictionaries and also go more in depth on already-covered topics such as lists and reading/writing to file. We’re then tasked with altering an existing script so it can handle two-dimensional data and upload it to GitHub for peer review. The assignment tests us on the new material but also underscores the importance of collaboration in coding.

# Dictionaries

Dictionaries are data types similar to lists in that they are a mutable collection of elements but they are specifically for storing paired data. Each key in a dictionary has a corresponding value and these value-pairs are matched with a colon, while the whole dictionary is surrounded by curly brackets. The function dict() can also be used to create a dictionary. Values and keys do not need to be of the same data type but keys must be immutable data types, so a tuple, string, or integer would work for a key. Using the in operator, I can check for key entries present in a dictionary and the len() function returns the number of key entries.

Where list elements are accessed by index position, dictionary elements do not have positions and instead are accessed using keys. Using the key in brackets, like you would with a list index position, returns the value paired with the key. If a dictionary is nested within another dictionary, two key arguments will be required in order to get to the nested level’s value. A new entry is added in a similar way, assigning a value to the key in brackets, and dictionary entries appear in the order that they were defined so new ones go on the end. A key can only appear once in a dictionary so if I try to add an entry with the same name of an existing key, I’ll just end up reassigning the original.

1. dicty = {'1':'a', '2':'b', '3': 'c'}
2. **print**(dicty['1'], dicty['2'], dicty['3'])
3. dicty['4'] = ['d']
4. **print** (dicty.keys())
5. **print**(len(dicty))

Figure - script demonstrating dictionary assignment, printing values using keys, adding new dictionary entry, printing dictionary keys, and printing length of dictionary – code formatting from [Planet B](http://www.planetb.ca/syntax-highlight-word)[[1]](#footnote-1)

# Assignment Five

In order to modify the script to replace the inner data structure with dictionaries, I’ve changed the user inputs that add new data to create key-value pairs, with each CD inventory line being its own dictionary. The dictionary containing the inventory data is then appended to list data that will be used for displaying the inventory to the user, deleting unwanted inventory, and saving inventory to file.

Loading existing inventory data from the file into data starts with opening the file to read. The rows of strings on the file are then iteratively stripped of their ending new line characters using the strip method and separated into elements in a list at each comma using the split method. Each index position of the new list is then used to create a dictionary where the positions are corresponding values to the keys of id, title and artist. The dictionary is then appended to the same list that is used when the user wants to create new inventory.

Graphical user interface, text, application, chat or text message

Description automatically generated

Figure - CDInventory.py running on Spyder

I rely on one for loop in order to display the data in memory to the user which iterates over the list of CD dictionaries from the main inventory list. Each iteration will print the three values stored in the dictionaries by calling their three corresponding keys. I use a comma to concatenate them so they’ll display separated by white space and the default new line end of the print statement will separate each dictionary in rows.

If the user chooses to delete an entry from data, an if condition will first check if there is data to delete by testing whether the length of the CD data list is longer than zero. If the condition is true, then there is data in the list and a print statement will then display to the user how many lines of CD inventory exist in memory. The len() function returns an integer so I have to use the string function in order to concatenate it within the print statement. The user will then input a number to represent the line of inventory that needs to be deleted which is converted into an integer so that it can be used to reference an index position. Returning an index position that does not exist will cause an error so I added structured error handling that will first try to delete an index position. If there is an error, a print statement will tell the user that the line number does not exist.

Saving the CD data to file first requires converting the cd data to strings before it can be written. I use two nested for loops to first iterate through the list of dictionaries and then iterate through the values of the dictionaries using the values method. The values are concatenated into an empty string separated by commas and then written to file using a string slice that omits the final comma and separates each row of values with a new line.

Text

Description automatically generated

Figure - CDInventory.py running in terminal

I usually like to use underscore naming convention with my variables but this was a good practice in adopting the precedence of someone else’s code and using camel case when assigning variables. I’ve also left Dirk’s first entry in the change log to show that I wasn’t the one to create the script but I did make changes to it.

# Summary

Now that we’ve reached the level of scripting that requires loops nested within loops and multi-dimensional data, the importance of writing clear and concise code has become important. Small quality of life improvements such as script templates will have a big impact moving forward, not just with keeping organized but also reducing time spent on not coding. Keeping my own code organized is enough of a struggle but we’ve added another dimension (again) of working with someone else’s code but it’s a nice way to dip my toe into collaborating with the larger community.

At this point in the course, I’m getting much faster at interpreting syntax errors and I’m beginning to find them more helpful than frustrating. I have tried to use conditional if statements to avoid error messages for the user in past assignments but it will be nice to add structured error handling to my toolbox as scripts continue to get more complex. I look forward to creating functions on future assignments and continue to make my scripts clearer and more organized for myself, the user, and the community.

1. Captured on 2021-Feb-13 [↑](#footnote-ref-1)