Adam Hockemeyer

2020-Feb-28

Introduction to Programming (Python)

Assignment06

**Adding Functionality to CD Inventory Script**

**Introduction**

During this assignment, I took previously written, partially functional code and added sections to improve and add to user options. I also moved much of the code into functions, and organized it all into proper sections based on function. The overall function of the script is to import the contents of a text document into a table in memory, allow the user to manipulate that table, then export the table when ready back into the file.

**Planning the Additions and Changes**

When I got the script to work on, I started by looking at which parts of code could be moved into functions. Broadly, I wanted to use three classes, based on the function of the code. These would be DataProcessor, to handle data within memory; FileProcessor, to handle importing and exporting our table, and IO to handle user input and output. With this decided, I got my list of needed functions and divided them into the classes that make sense. The functionality to add albums and delete albums out of the table was put into DataProcessor, reading and writing to file was put into FileProcessor, and printing the menu, handling the user's input choice, showing the current inventory, and handling the user input part of inputting an album were all put into IO.

**Class IO**

The first function in IO would be print the menu for the user, print\_menu(). This function is one of the most core functions, as it is seen every time the user wants to make a choice, coupled with menu\_choice(), to handle that menu input. Neither of these functions require any arguments, since they either don't need it or generate the input while inside the function.



*Figure 1 - print\_menu() and menu\_choice() in action.*

The third function in IO is show\_inventory(table). This function prints the current table in memory, displaying it in a legible manner. I decided to have this function take the argument of the name of the table saved in memory, because if there was ever a point in the future where functionality for multiple tables would be added,



*Figure 2 - show\_inventory(table) in action.*

The final function in IO is input\_album(). This function is designed to prompt the user for three types of input, each time asking for an ID, an Album Name, and an Artist. These are saved as strings, and immediately used in DataProcessor.add\_album. I chose to have these be two seperate functions because they have two distinct types of functionality; one asks for and handles user input, while the other deals with data manipulation in memory. The functions were built to be used together, though, in that you can nest them and the output of input\_album() is properly formatted for use in add\_album.



*Figure 3 - input\_album() and add\_album in action.*

**Class DataProcessor**

The DataProcessor class has two functions in it, one to handle adding user-inputted albums into the table in memory, while the other is to handle deleting items out of the table. The first, add\_album(lstInput), was touched on earlier. This takes the input from input\_album() and turns it into a dictionary with keys, then appends that dictionary to the bottom of our table in memory.

The second function, delete\_album(intIDDel), searches our table for the user-inputted ID number, then deletes the line it finds that ID on. It searches using a for loop, iterating through the table and searching the ID key in each dictionary for the matched ID number. This function returns a boolean of whether or not it was able to find and delete the line item the user searched for, and back in the main script the user is shown the results based on that boolean.

I was able to handle user input, the delete\_album function, and storing the boolean result in a single line of code by running the delete\_album function with a nested input function, which is subsequently ran through int() before passing the argument to delete\_album.



*Figure 4 - delete\_album() in action.*

**Class FileProcessor**

The final class I built is for reading and writing to file our table in memory. Both of these functions take the arguments file\_name and table name, because they mirror each other, both needing to both read from and write to one of those two locations. The first function, read\_file, opens the specified text document and iterates through it, for each line turning it into a dictionary and appending it the bottom of the table in memory specified by the arguments. Read\_file also clears the table in memory before writing anything, in order to make sure we don't add duplicate versions of our inventory if the user wants to reload the data multiple times.



*Figure 5 - read\_file() in action, being run by giving the user the 'load data' option.*

The second and final function in FileProcessor is write\_file, which works as a mirror to read\_file, by opening a file, iterating through our table in memory, and for each line turning it into a comma-seperated row and writing it into the text file.



*Figure 6 - write\_file() in action, being run by giving the user the 'save data' option.*

**The Main Script**

After defining the classes, the rest of the script is dedicated to putting them to use. Firstly, the script loads data into the table, then goes into a while loop and printing the menu. Each menu option uses IO.menu\_choice() to parse which function to call, then calls that function and loops back to the top of the menu. After putting together all of the functions and setting up the menu, the script ran as intended and I took screenshots.



*Figure 7 - the script being run in Anaconda Prompt.*

**Summary**

I was able to put the code of this script into classes and functions, letting the code be properly seperated into type of functionality. I was successfully able to make a menu system that ran functions cleanly.

GitHub link: <https://github.com/heliotropite/Assignment_06>