Sean McElmurry

2020Feb22

Foundations of Programming, Python

Assignment05

Dictionaries and Development Strategies

# Introduction

Over the fifth week of the Foundation of Programming, Python class, we have learned about dictionaries, and some development practices when writing code. To apply the information (apart from user-define functions) we modified a starting script that mimicked the CDInventory.py script from the prior week’s assignment. The execution of the script in a terminal window can be seen in Figure 1 below.

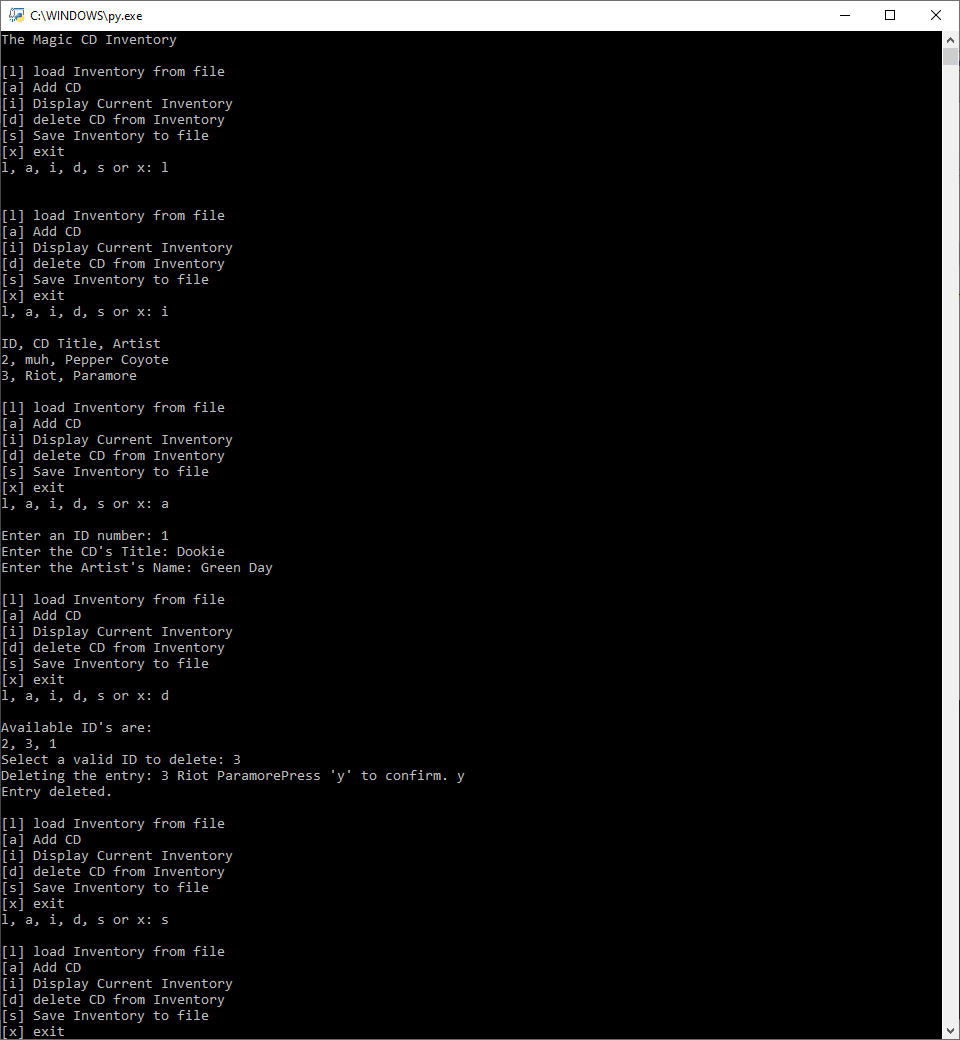
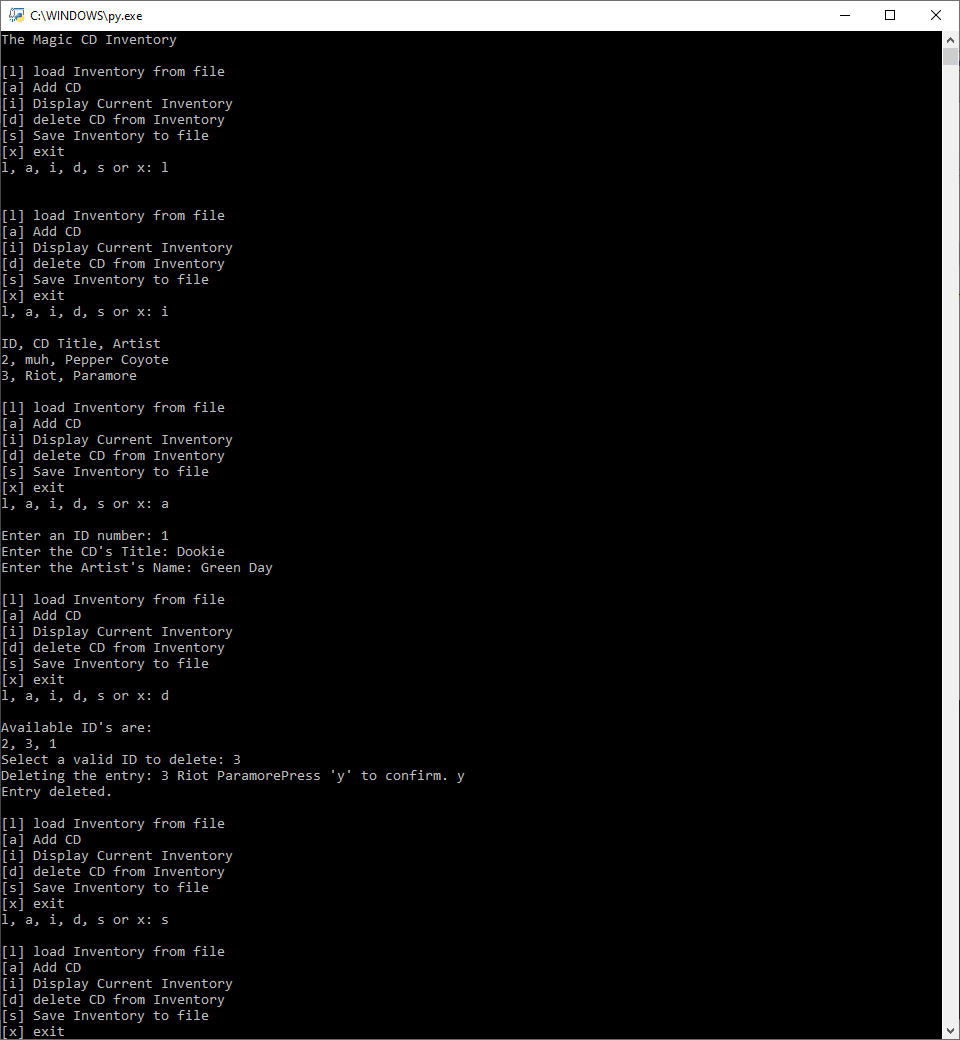


Figure - CDInventory.py When Run from a Terminal Window

# Dictionaries vs Lists

Dictionaries and Lists are both organizational tools to monitor and store information, however, where lists store their values by sequential indices, dictionaries store their information in pairs of keys and values. For instance, if you want to call an item from a list, you call on the index of the list (i.e. *objList.[i]*) which will locate the information at that spot. If you remove an item from that index, any item past that index get shifted one position down, demonstrating that indices are sequential, and can never be greater than the length of the list. For dictionaries, you store a value with an immutable key. Since there is no sequence to the keys, keys can be removed from dictionary objects, and other values will not change keys.

Using the *'r'* parameter of the *open()* function in combination with a for loop, we can read a text file one line at a time and add each line to a list (Foundations of Programming, Dawson). If there are any separator characters in the line being read, we can denote how the data should be delimited by using the *.split()* method. If there are other manipulations to be done to the string, those methods should be called before splitting the string, as it will behave as an array after the *.split()* method. To read a text file into a dictionary, follow the same steps as for a list, but then you can call specific, singular indices of data to be your keys or values as needed.

# Organizational Strategies

Functions are bodies of code that can be called on in the main body of code to reduce unnecessary clutter. They are essentially user-define variables, but instead of simply storing data, they are called on to process data and return some output. This is especially useful for dividing the code with Separation of Concerns, or when a function may need to be called on multiple times in the script.

Separation of Concerns (SoC) is an organizational method that attempts to divide the code into three easily identifiable sections: variables, processing data, and presenting data (or user interaction). It would be difficult to work into our assignment for the week since we were not allowed to use Functions, the main bodies of the processing/user interaction sections were muddled together.

Script templates are helpful for starting to write new projects. Not only do they help users save time writing a header (assuming they have a preferred and consistent style), but it can also pre-load a template of pseudocode or SoC into where the actual code will be written.

I find error handling extremely useful. Not only can it be used to check how forced errors would behave in the code when developing, but it can also be used to present a sort of bypass for those errors during execution *(*[*Foundations of Programming (Python)* *Module 03 Part 4*](https://www.youtube.com/watch?v=uaHH-o83OvQ&feature=youtube)*,* External Site). For instance, I noticed in the starter code for this week that one line read *intID = int(strID)* but if the user had entered anything that could not be converted into an integer an error would stop the code. With try-except, we are able to test that line out, and request the user enters a different one that is appropriate.

# GitHub

We very briefly touched on GitHub this week as well. GitHub is well known as a version control and collaboration tool that is widely used where multiple people are working on a single project. Also, with a mascot like Octocat, who wouldn’t want to use it?

# Writing the Application Script

The script provided a good opportunity to experience the differences between lists and dictionaries, but I also found some extra practice opportunities to manipulate this type of 2-D data structure (a list of dictionaries) within the assignment that I believe deepened my understanding of how those structures interact together. The full script used is quite a bit larger than past projects and can be seen in the appendix. However, its execution in the Spyder terminal window can be seen in Figure 2 below.

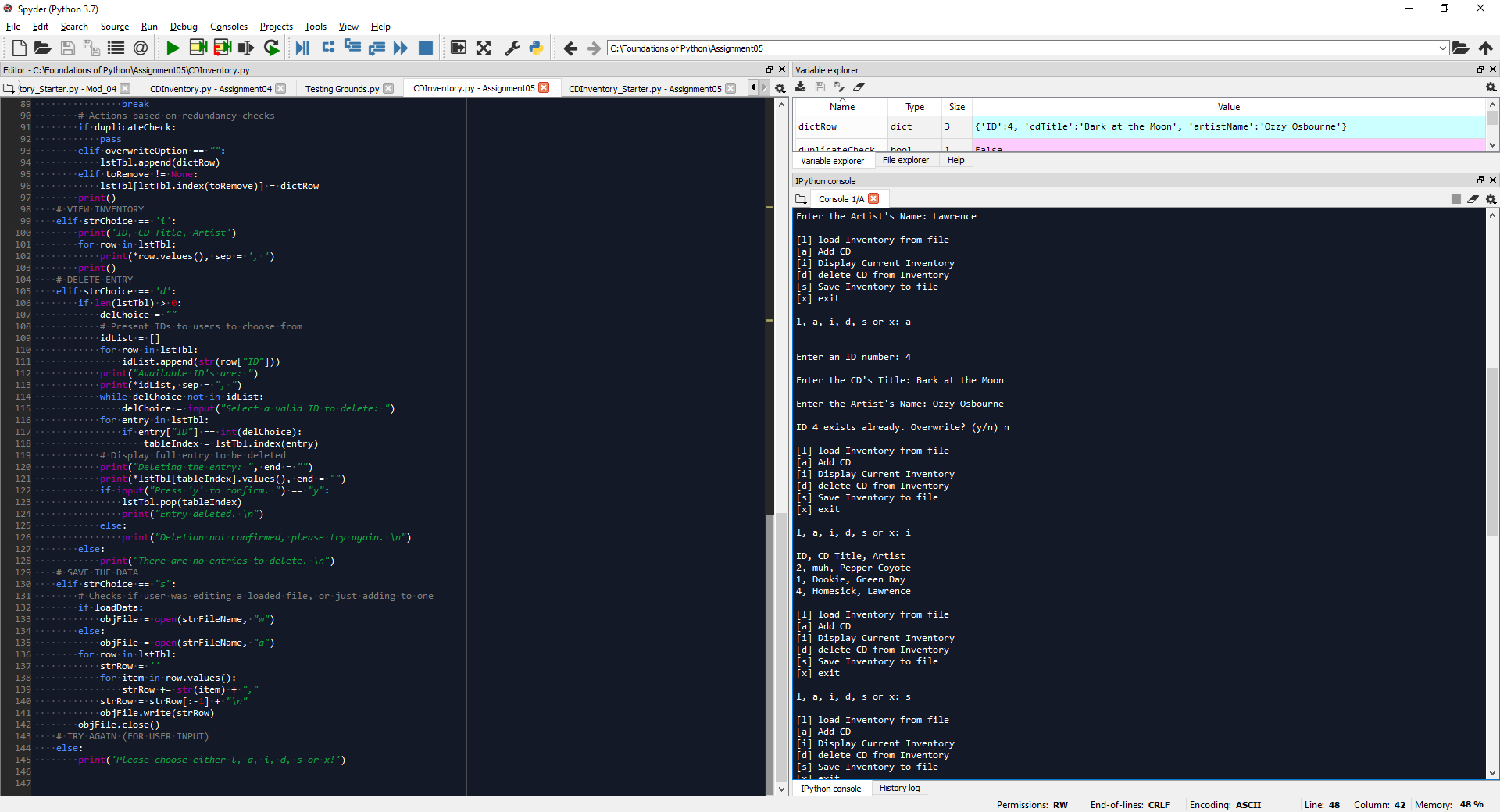
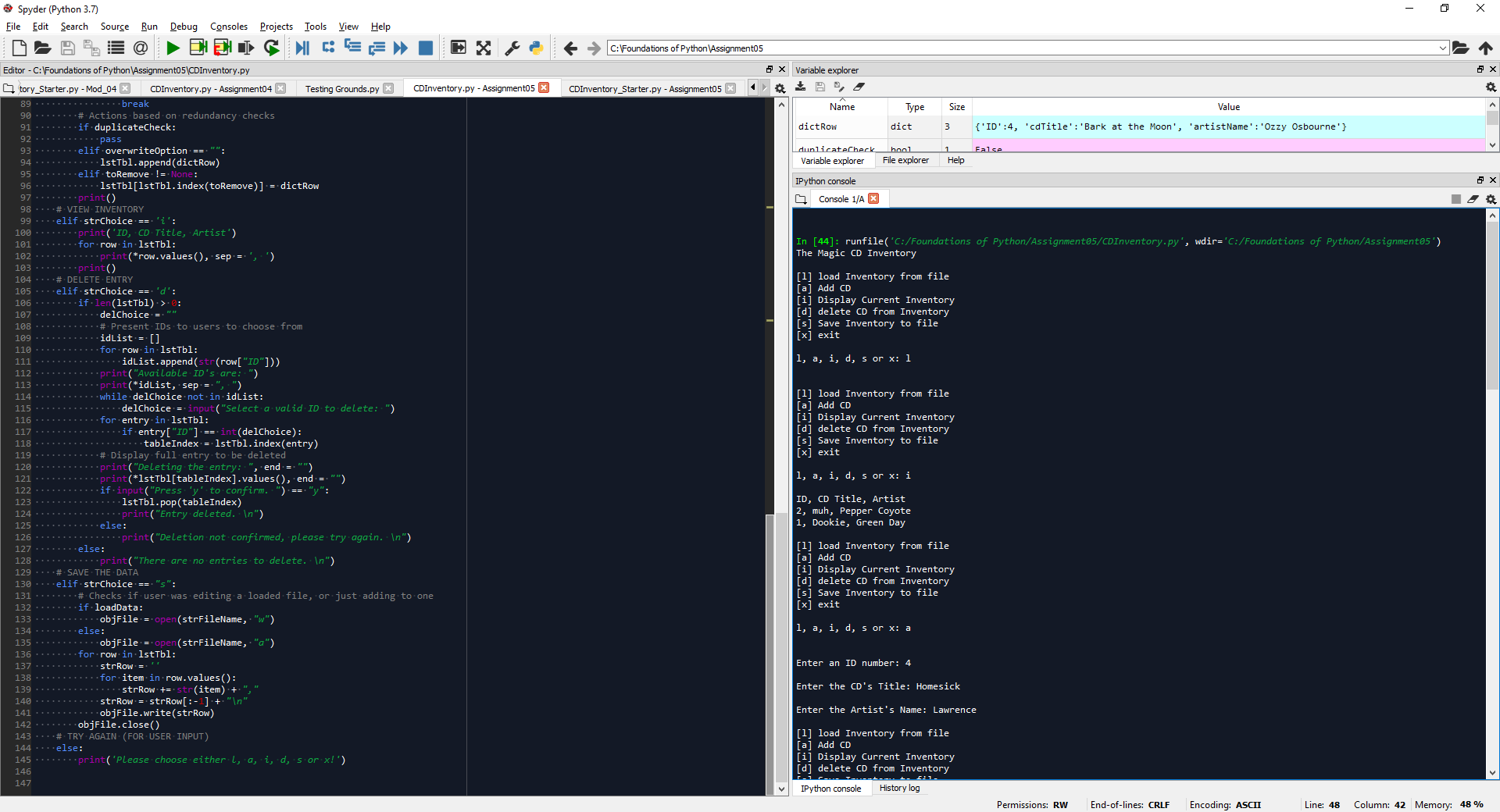


Figure - CDInventory.py Run in Spyder

The first action I took was to open a second editor, and view the different menu options (from last week’s assignment and this week’s starter) side by side to look for any major differences, and try to ‘translate’ my old code into the starter code. For me, I needed to understand the original author’s writing style, and how I could meld mine with it to build off of it.

## Add a CD

I decided to start with this section of the menu as all other menu items could be tested with dictionary items after the Add a CD option was complete. The change that section, we really just needed to change the *lstRow* variable into a dictionary type, and assign the variables being input to keys for an ID, album title, and artist name.

For me, the challenge here because about wanting to ensure unique IDs were being used, avoiding redundant album/artist pairs, and preventing user error. First up was ensuring the user was using an integer as an ID since the *int()* function was being called on the string the user was supposed to enter, but there were no real restrictions. To mitigate the failure mode, I paired a while loop with a try-except statement until a user entered a valid ID. Unique IDs were more of a problem since I wanted to give the user the ability to overwrite another entry, but I needed to wait and check another trait before allowing the overwrite. It did help me understand how to write a for loop to check for matching values within a dictionary as seen in Figure 3 below, and that with a list-dictionary structure it is possible to use a single for loop to parse through entries. Lastly I had decided the code would do nothing if a album/artist pair already existed, since it would be redundant to display the same item twice, and if desired the ID could be changed through other processes.

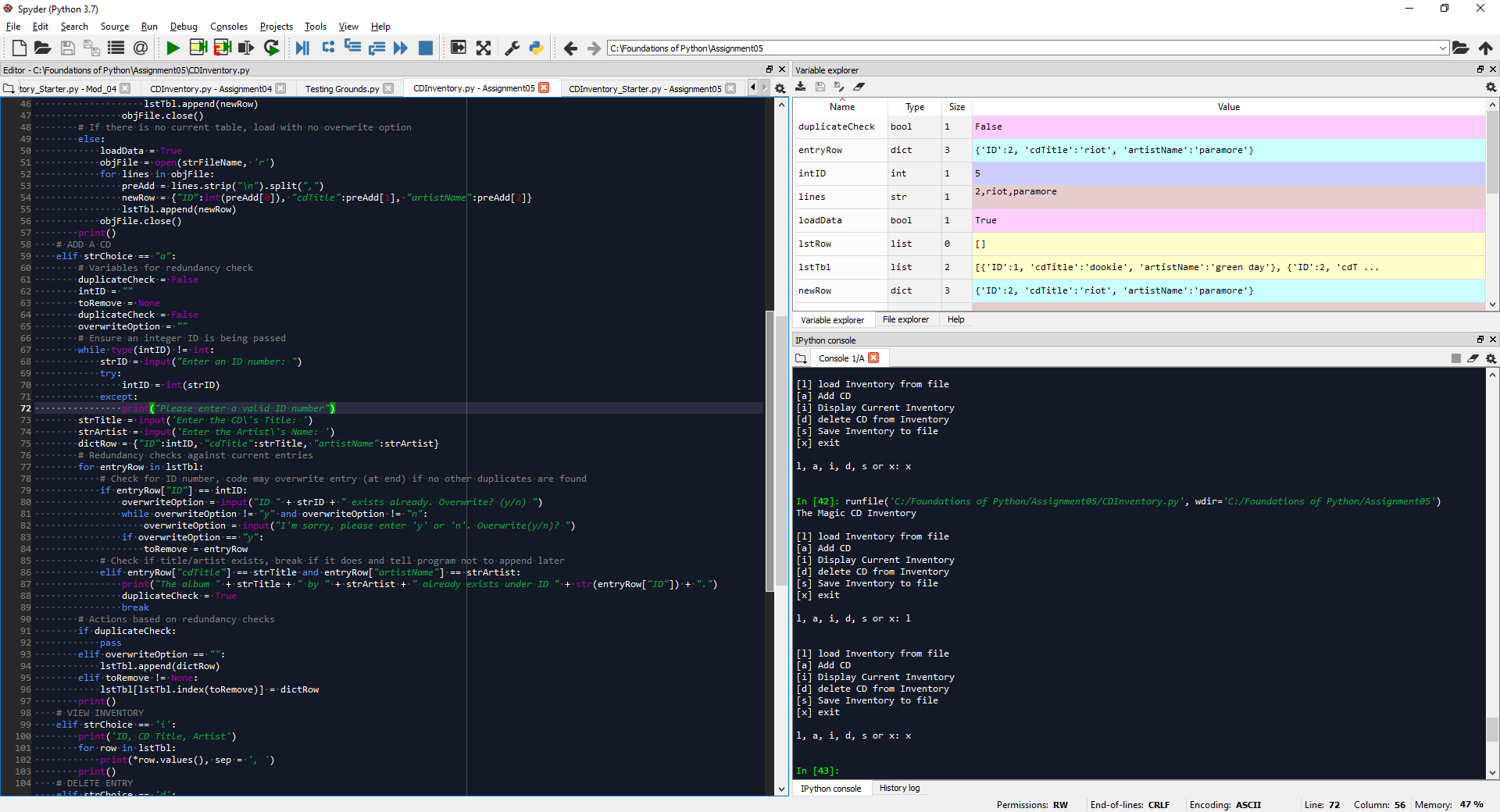


Figure - For Loop Checking Dictionary Values Stored in a List

## View Inventory

The body of code for viewing the inventory was almost identical to the starter code, with the exception that dictionaries behave differently when printed. To present the data, we had to first convert the items of interest (values) into an array with the *.values()* method, and then we could print that list using the existing body.

## Delete Entry

I initially thought that the Delete Entry section would be the easiest, but after realizing that the unique identifier to delete entries had to be the ID field, the challenge became about keeping track of variable types and the proper way to call conversion functions. My biggest challenge was figuring out how to unpack dictionary values in an *input()* function, and honestly it defeated me as nothing I attempted was working. I ended up unpacking the values in a print statement and used a separate *input()* call to interact with the user.

## Save the Data

Similar to the ‘View Inventory’ section, the only real portion of the code that needed altering was updating the row object to a list using the *.values()* method since it then behaved as a dictionary object. I did add the ability for the program to append the data to the file or overwrite the file depending on if a file was loaded in. The idea being to avoid appending the exact same entries to the table if a user loaded, then saved, then saved, then saved, etc.

## Load Data from File

In this section of the code we were able to apply the knowledge we learned of as talked about prior in the section titled “Dictionaries vs Lists.” Outside of how to load text files to a program, one of the most important pieces of information I learned writing this section is that the order of object methods matters. A lot. Anything that’s called to modify the object and keep it as the same data type must be done first, so save the *.split()* methods until the end. I also added extra code to prevent accidental redundancy here which probably could have been cleaner since it opts to erase the entire current inventory if loading a file after adding entries, but it 100% prevents redundancies within this menu option.

# Summary

I like to think of dictionaries as the organizational structure used when the order of the entries is not important and sequences otherwise. SoCs seem like good coding practices that were difficult to implement into our application this week, but that is due more to the lack of functions than the coding assignment itself.

# Appendix

## Sources

Dawson, Michael. Python Programming for the Absolute Beginner: Third ed., Course Technology, Cengage Learning, 2010.

“Foundations of Programming (Python) Module 03 Part 4.” YouTube, 2020Jan07, <https://www.youtube.com/watch?v=uaHH-o83OvQ&feature=youtube>. Accessed 2020Feb07.

## Script

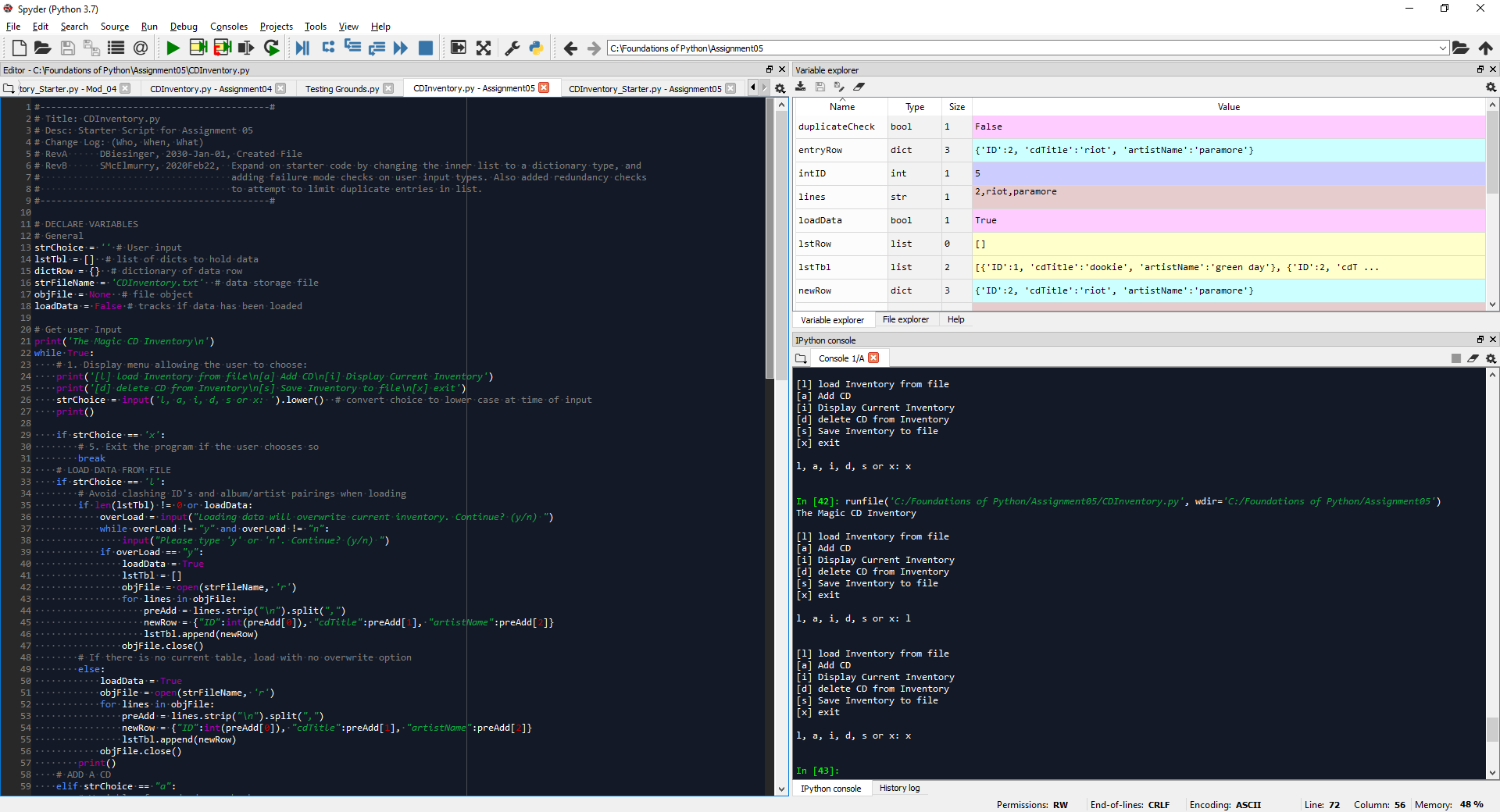


Figure - CDInventory.py Script (Part 1 of 3)

## Script(continued)

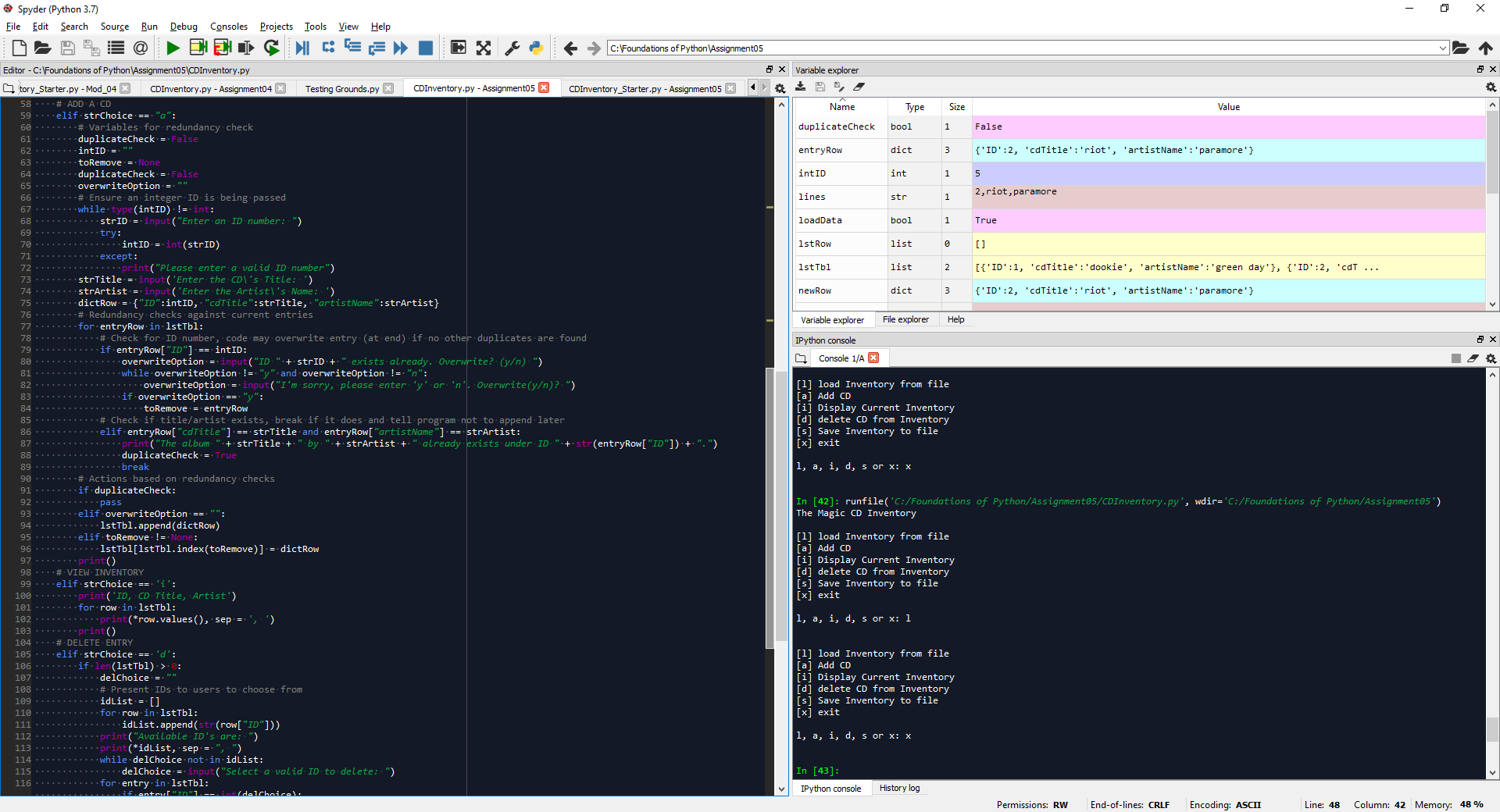


Figure - CDInventory.py Script (Part 2 of 3)

## Script(continued)

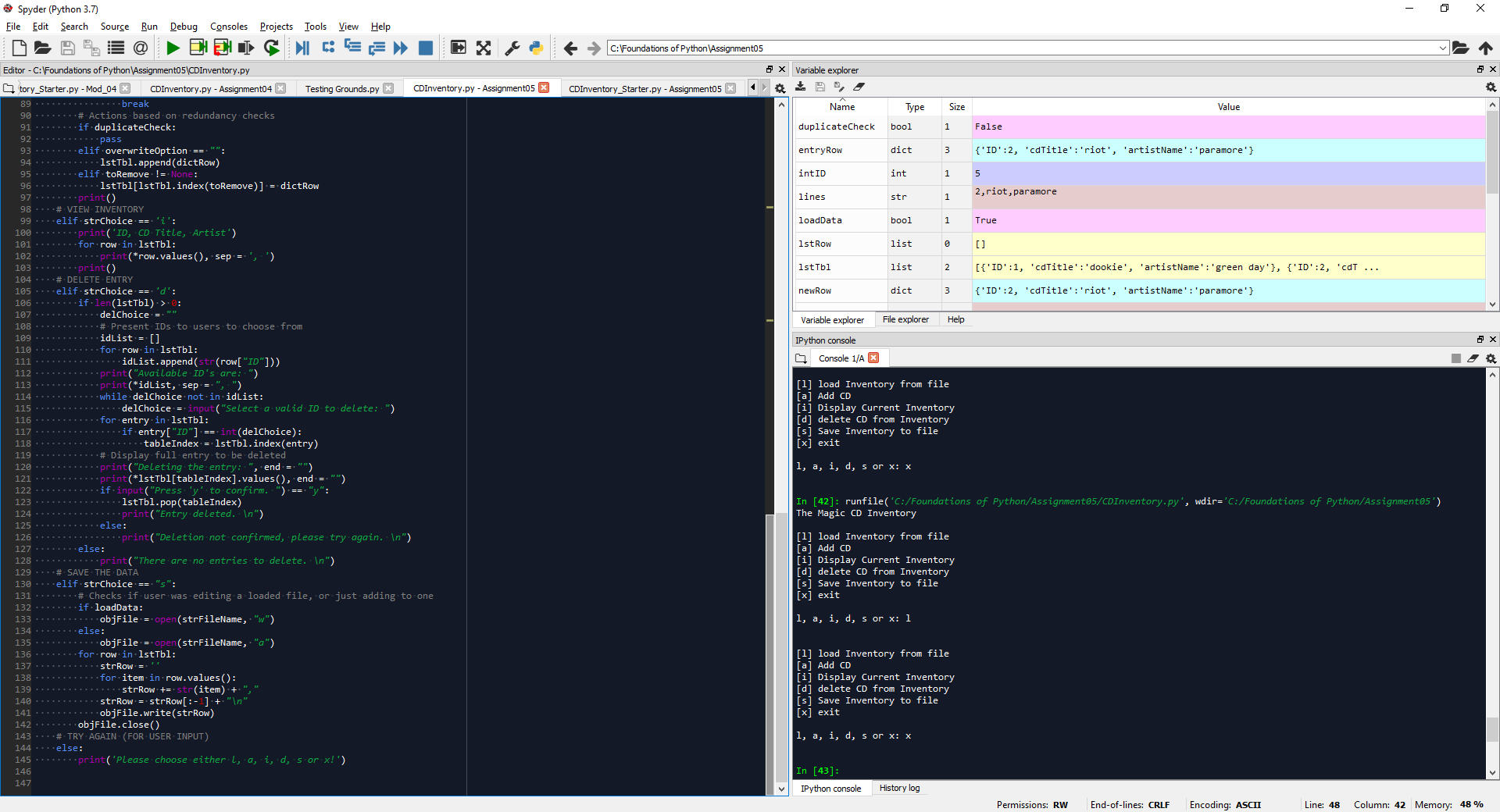


Figure - CDInventory.py Script (Part 3 of 3)