Sean McElmurry

2020Mar02

Foundations of Programming, Python

Assignment06

Using Functions

# Introduction

Over the sixth week of the Foundation of Programming, Python class, we have learned about how to write user-defined functions in Python. To apply the information 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 (UPDATE).

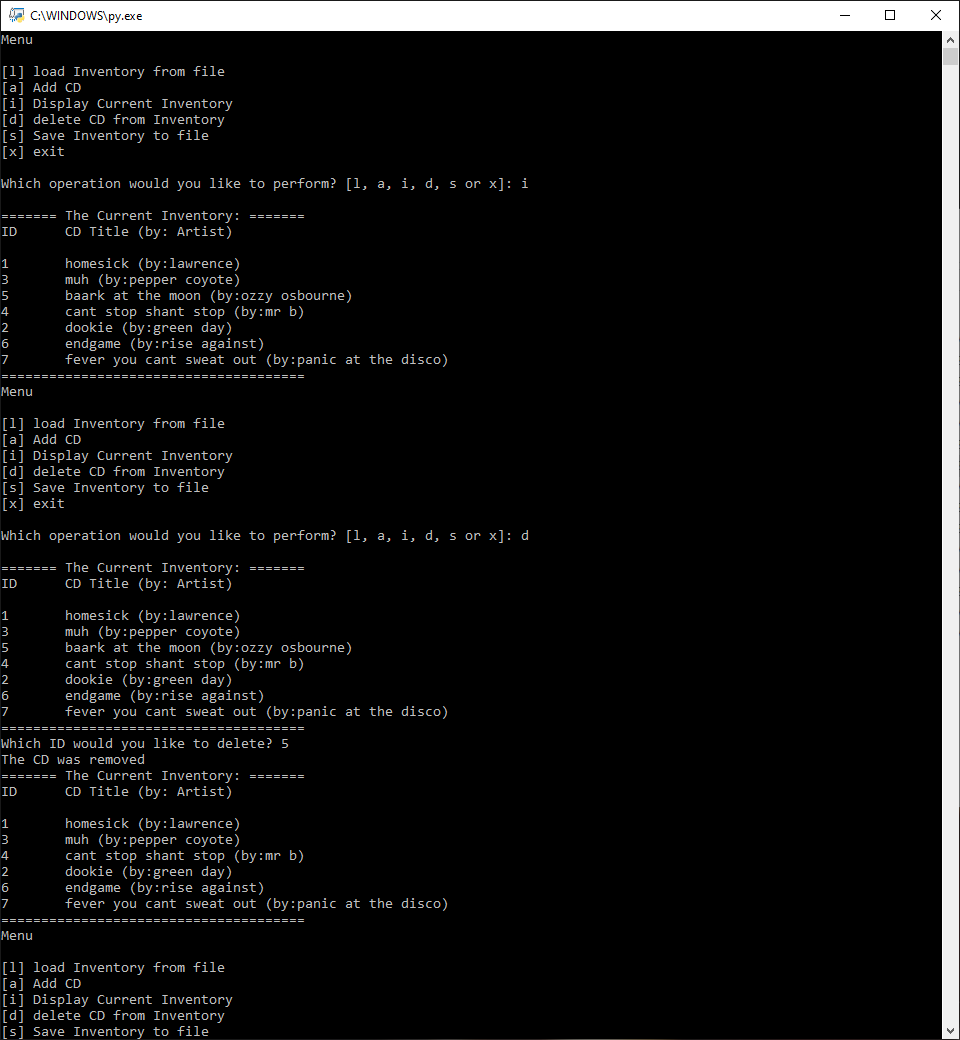


Figure - CDInventory.py When Run from a Terminal Window

# Functions (General)

Simply put, a function is a statement that performs actions and can be used to reduce unnecessary repetition and clutter in a script. Python has many built-in functions that we have used previously (e.g. *len(x)*), but this week we have been focusing on writing our own functions. Functions we have worked with have been comprised of three main sections, the header, the body, and the return values.

Within the header of a function, we have the function name, and the variables passed into the function called parameters. Arguments and parameters of functions are extremely similar, and their differences can be summed up as follows: parameters are variables passed to a function (*def function(param1, param2)*), and arguments are the values assigned to those variables when the function is called (*function( 1, 2)*).

Most functions we have worked with have returned a value. The return value can be any bit of data, but is typically a variable that has been calculated or determined (as opposed to a static datum) within the function. When returning multiple values, the function will return a collection (a tuple by default) whose type can be assigned at the return line.

# Functions (In Relation to an Entire Script)

Defined variables that exist in the main body of the code can be called within a function without being passed as an argument. However, reassigning those variables does not actually update those values unless the variables being changed are reference type variables because they exist locally within the function. In order for the variables to exist or be changed outside of the function when altered within the function, the variables must be defined as a global-type variable.

Shadowing variables is assigning the same name to a variable within a function as one that already exists outside of the function. It is generally frowned upon as it makes it difficult for people to easily distinguish between the two different variables that have the same name in a script.

In a small script where few functions are needed, functions can help organize statements by reducing clutter and removing repetitive sub-bodies of code from the main body making the script more legible to a human interpreter. Classes are groupings of functions,

Where functions are an organizational tool for statements, classes can be used as an organizational tool for grouping functions ([*Foundations of Programming (Python)* *Module 06 Part 4,*](https://www.youtube.com/watch?v=U9wRmnWesRk&feature=youtu.be)External Site). Classes provide a simple way to organize functions, when multiple functions may fall under each section of a Separation of Concerns (SoC) pattern. Similarly to calling methods on objects, to invoke a class function, list the class name, followed by a period, and the function name you wish to call (e.g. *class.function(param)*). An additional perk is that a human reading the main body of code will be able to identify what SoC section a function belongs to (if naming the class by section header) without having to figure out what task the function completes.

# Writing the Application Script

I think the biggest challenge to this week’s script was learning to interpret and alter another person’s code. I did feel that it was useful practice experiencing how return values work (particularly multiple return values), and even got to work with a small amount of recurrence. The full script can be seen in the appendix; and its execution in the Spyder terminal window can be seen in Figure 2 below.

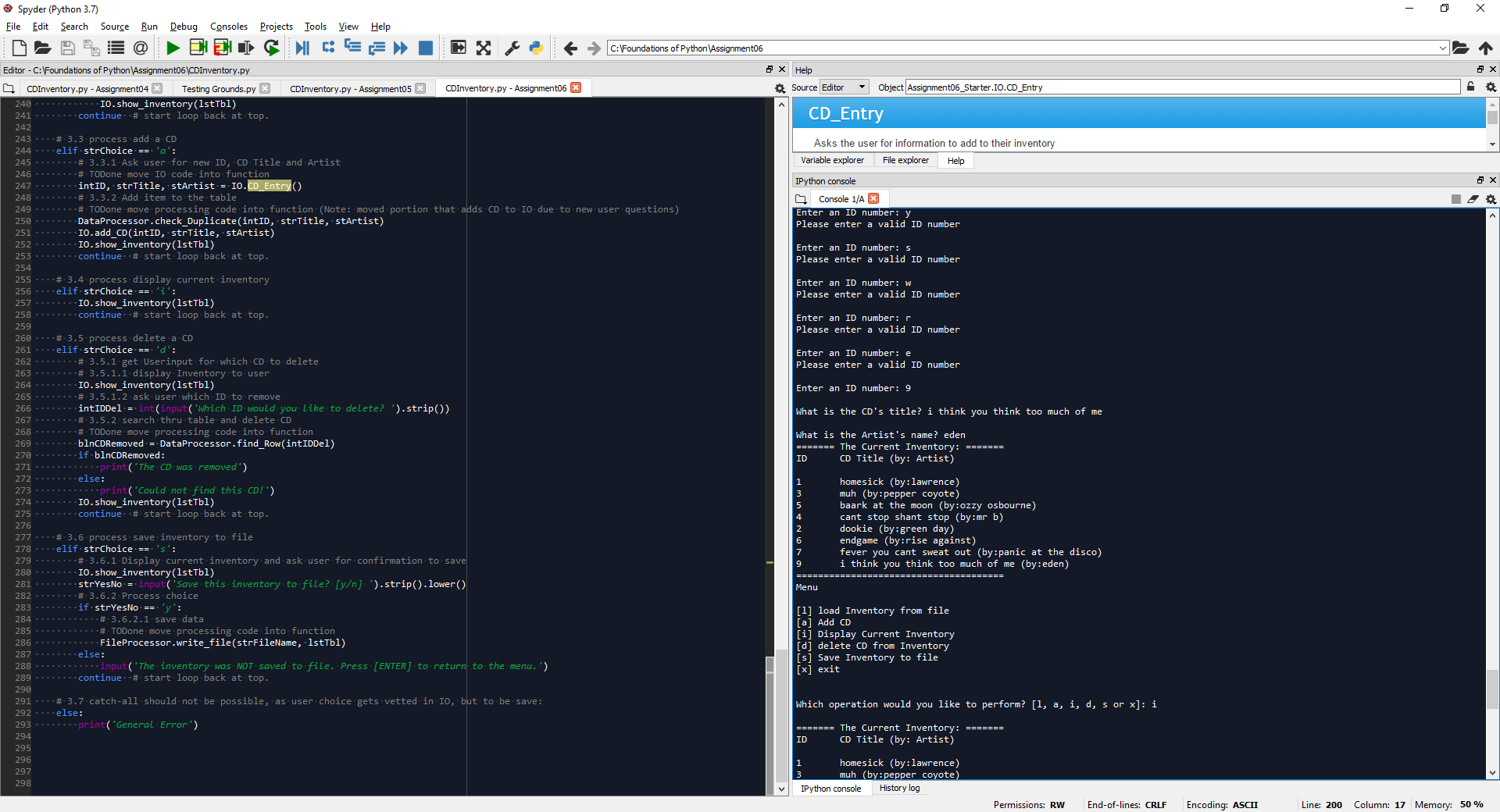
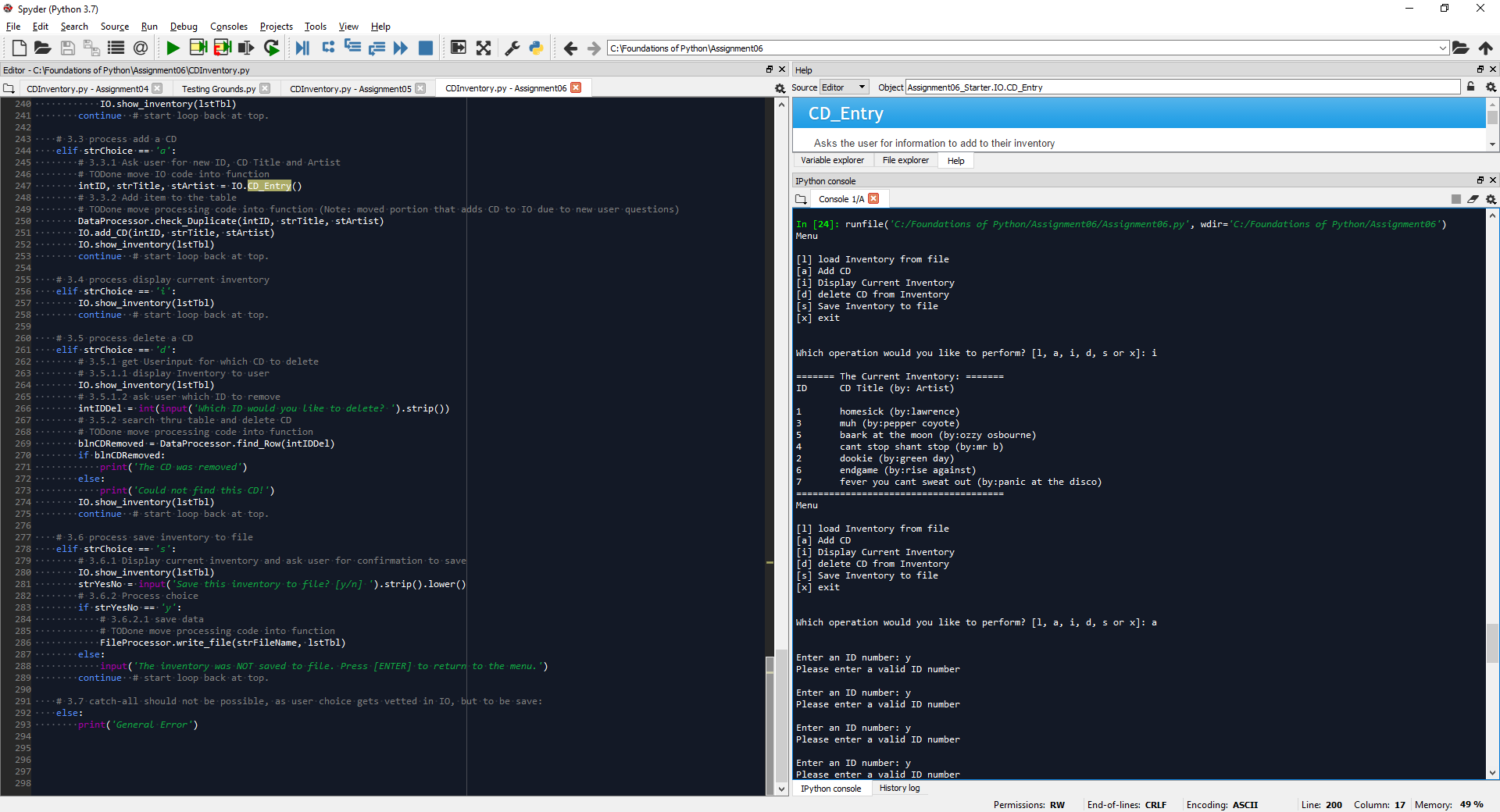


Figure - CDInventory.py Run in Spyder

I decided to prioritize sections the same as last week’s assignment (by menu options: a->d->s) I do think that it was extremely helpful to have worked with this code for the past two weeks since we did not have to spend as much time trying to understand what the output of the code should look like. For each function written, the general process I used was to copy the body from the main script and place it in a function, then look for the variables that would be used within the main body again, and ensure those get returned (and unpacked if necessary).

For sections of code that we needed to add and not just move, I compared the script to what was written for last week’s assignment, and identified where certain bodies from that week’s assignment would be successfully integrated as a function. Lastly, I identified where functions could be called within other functions to further reduce extra lines of code.

## DataProcessor Functions

*DataProcessor.find\_Row(idToFind)* is a function that for the most part was just copied and pasted from starter code. The only real differences were setting up the type int parameter, type Boolean return, and if statement that removes the list entry if the user is using the delete menu option. I decided to add that into this function rather than create its own because I had created another function that does the exact same thing and can use the same method, but just doesn’t delete the entry it finds.

*DataProcessor.check\_Duplicate(dupID, dupTitle, dupArtist)* built off of my redundancy checks from last week’s assignment. After being passed a CD’s values (int, string, string), it calls on the *DataProcessor.find\_Row(dupID)* to determine if the ID has already been used or not. Then it also checks if the artist/title pairing already exists in the inventory. The function results in two Booleans being returned to measure if the unique identifiers have already been input to the inventory.

(potential UPDATE).

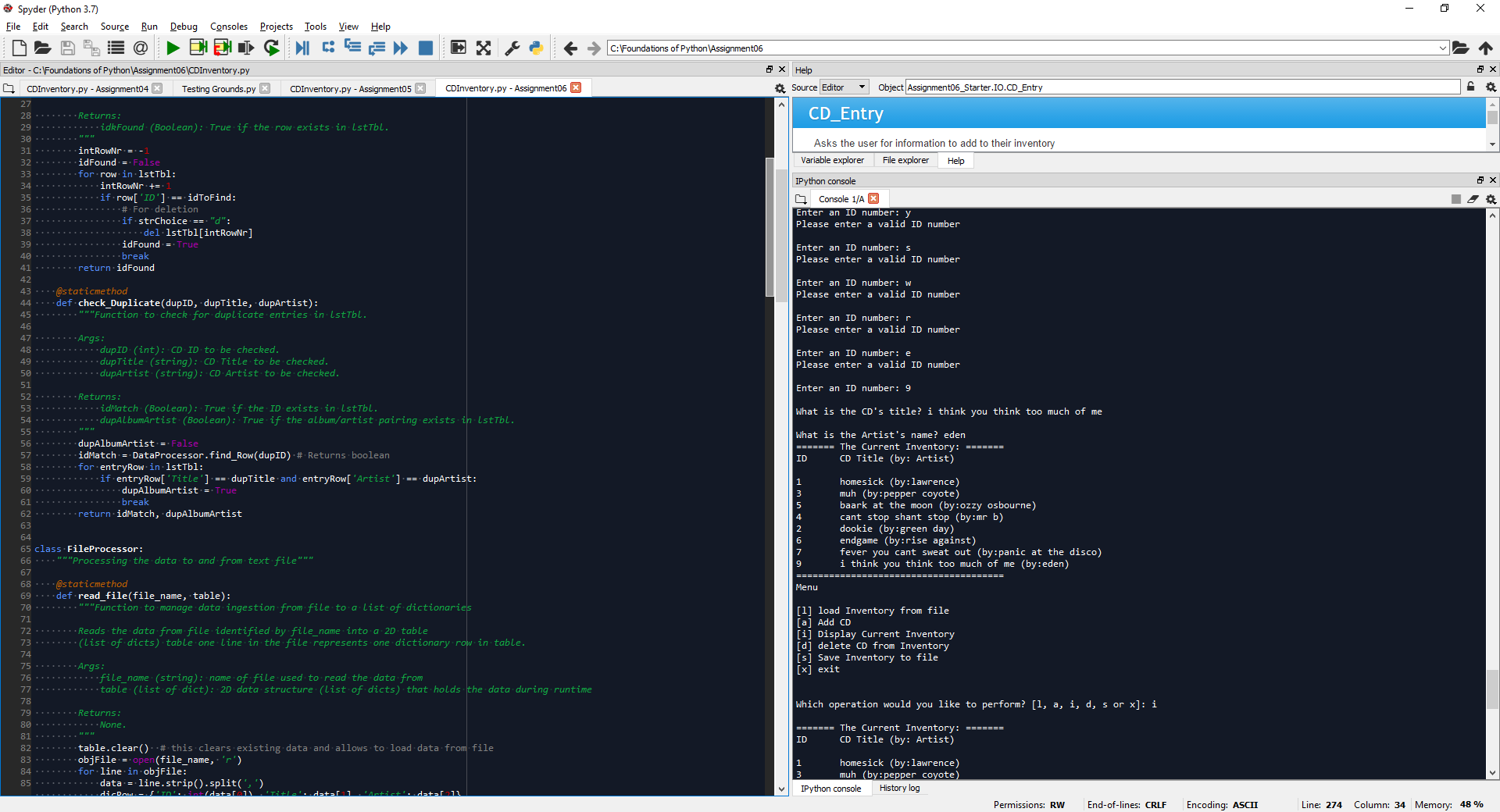


Figure - DataProcessor.check\_Duplicate(dupID, dupTitle, dupArtist)

## FileProcessor Functions

*FileProcessor.write\_file(file\_name, table)* was simple enough as the only thing we had to figure out what parameters to pass into the function.

## IO Classes

*IO.Int\_check(typedInt)* is a function written to error-proof when an ID is asked for to ensure an actual int value is being passed. While simple in its nature, I really liked the applications of using this function since its need was easily identifiable in multiple locations within the script.

*IO.Add\_CD(cdID, cdTitle, cdArtist)* adds the dictionary of CD information to the list. Before adding, it runs *DataProcessor.check\_Duplicate* which results in either informs the user their CD already exists in the inventory, adds the CD, or runs recursively with itself if ID already taken until a new ID that is not taken is entered. Originally intended for DataProcessor but was moved to IO with the user interactions.

*IO.CD\_Entry()* asks the user for CD information and runs *IO.Int\_check* on the ID entered. The function was mostly written, but we did have to add the return statement so that the information would be available outside of the program while avoiding global variables.

# Summary

I think the assignment provided an opportunity for use to see how to use functions in our code. However, I do think it would have been slightly more challenging if the functions we were writing were called on more times during the execution of the code. I think the biggest challenge that I had from the assignment was not shadowing variables.

# Appendix

## Sources

“Foundations of Programming (Python) Module 06 Part 4.” YouTube, 2020Jan24, <https://www.youtube.com/watch?v=U9wRmnWesRk&feature=youtu.be>. Accessed 2020Feb26.

## GitHub Repository

<https://github.com/SMcElmo/Assignment_06>

## Script

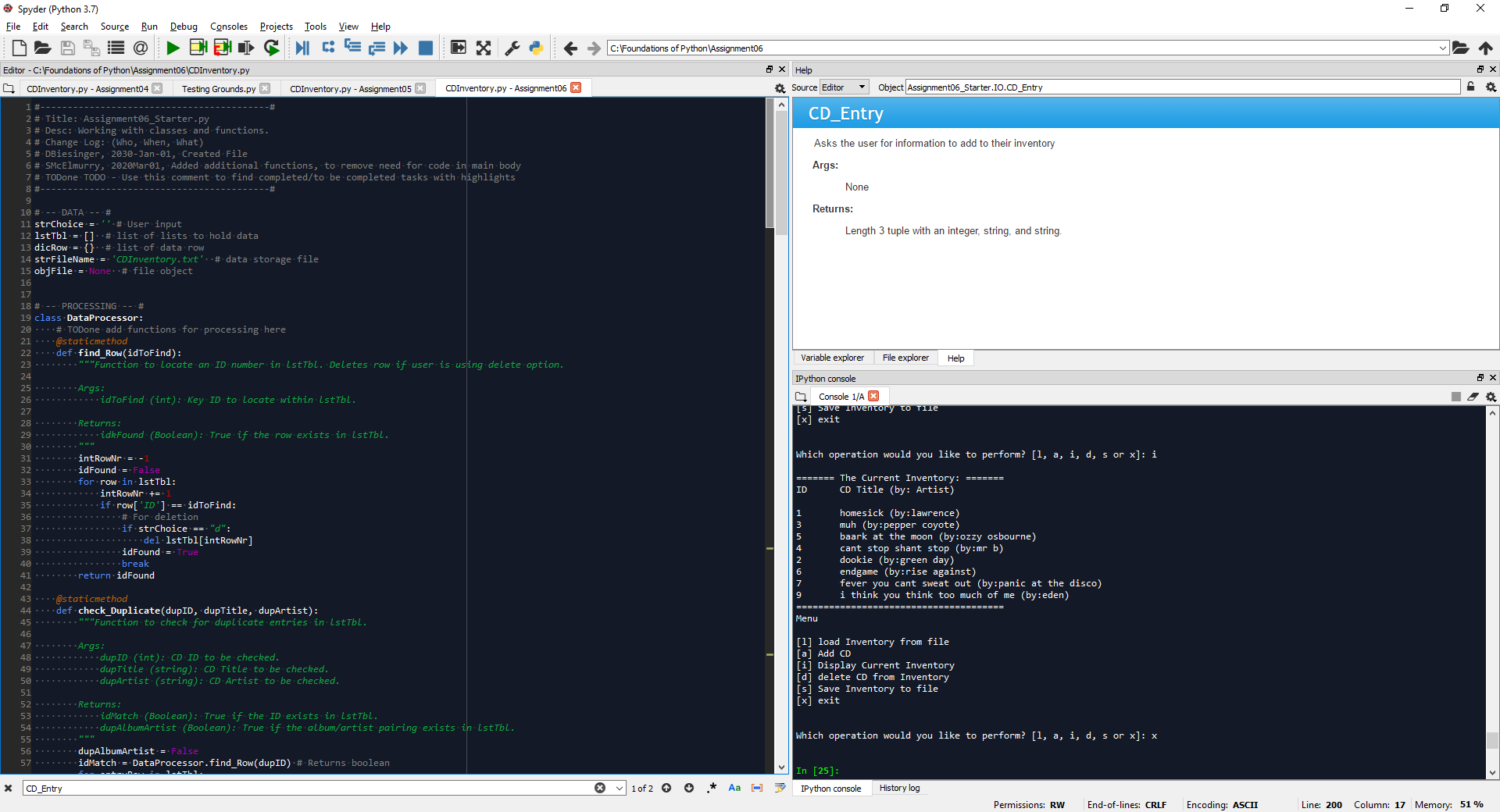


Figure - CDInventory.py Script (Part 1 of 6)

## Script(continued)

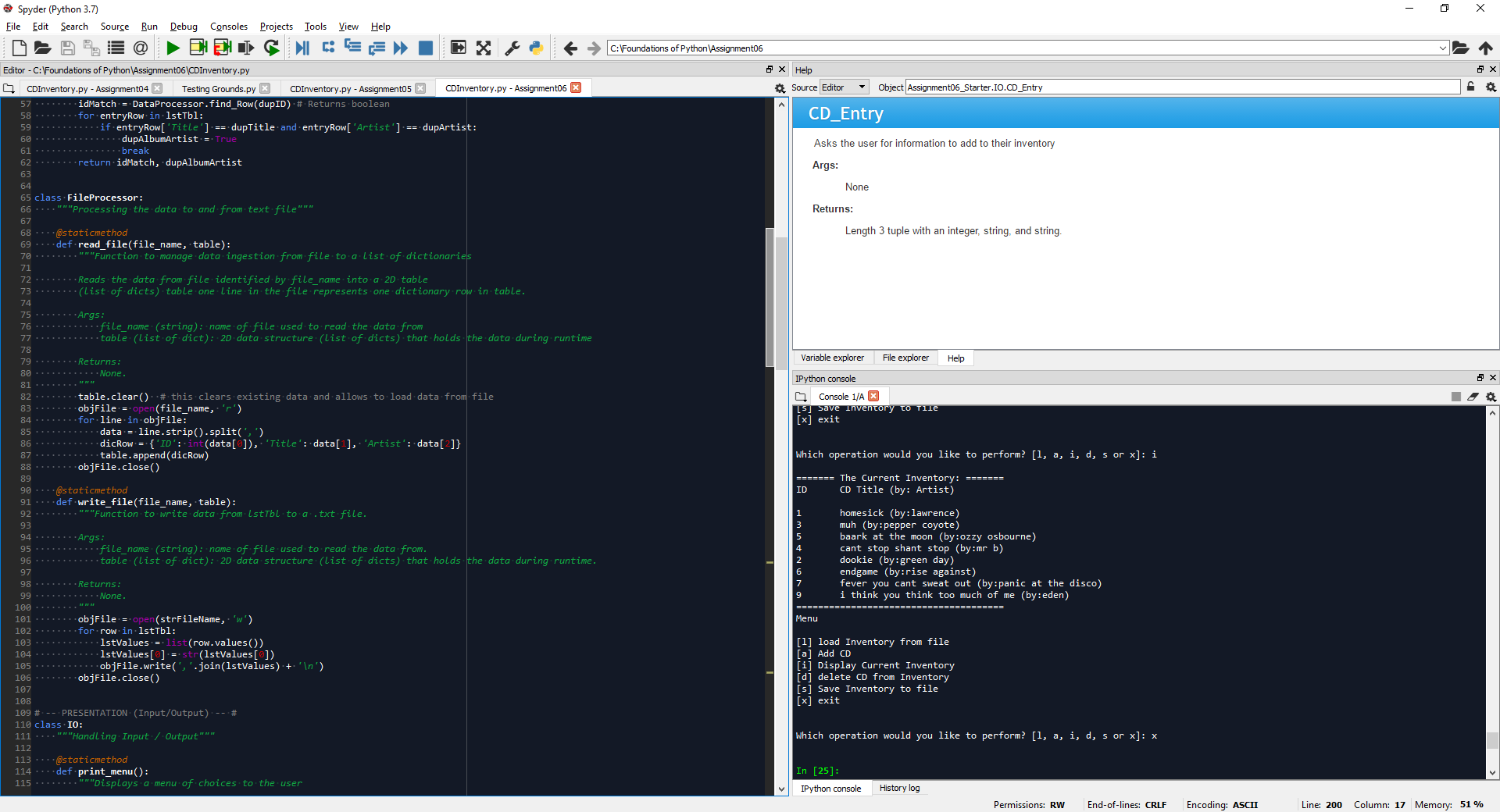


Figure - CDInventory.py Script (Part 2 of 6)

## Script(continued)

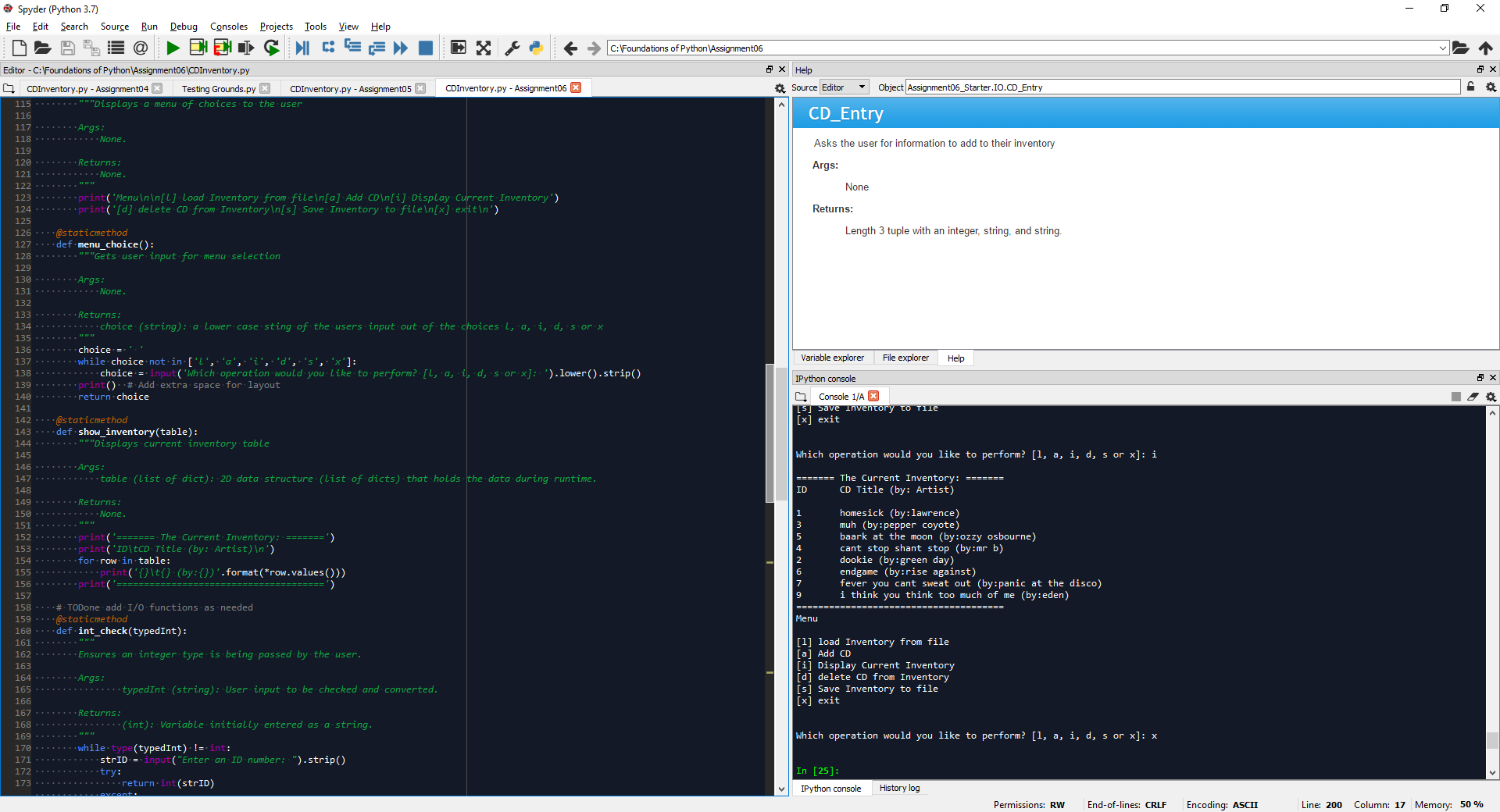


Figure - CDInventory.py Script (Part 3 of 6)

## Script(continued)

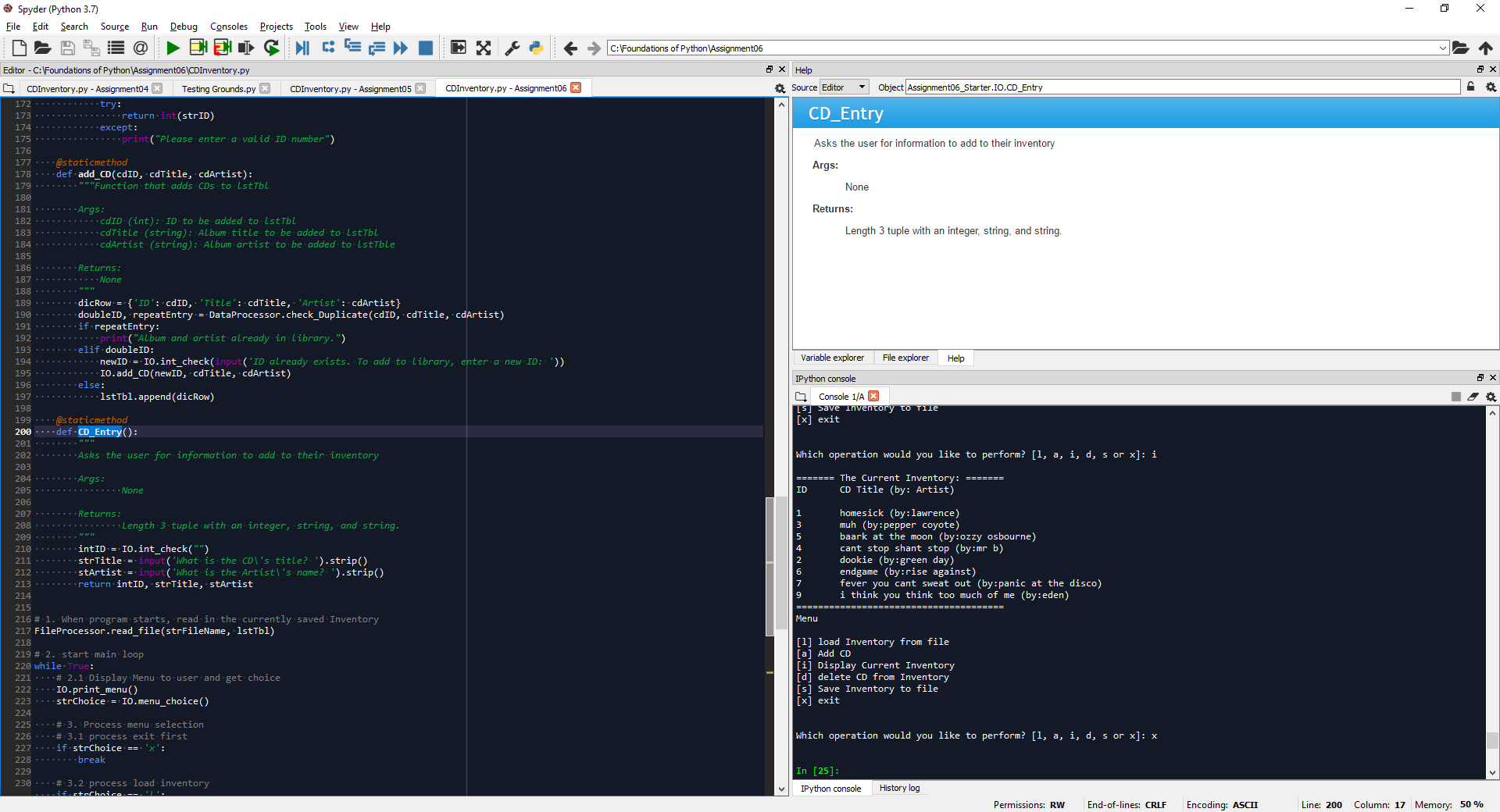


Figure 7- CDInventory.py Script (Part 4 of 6)

## Script(continued)

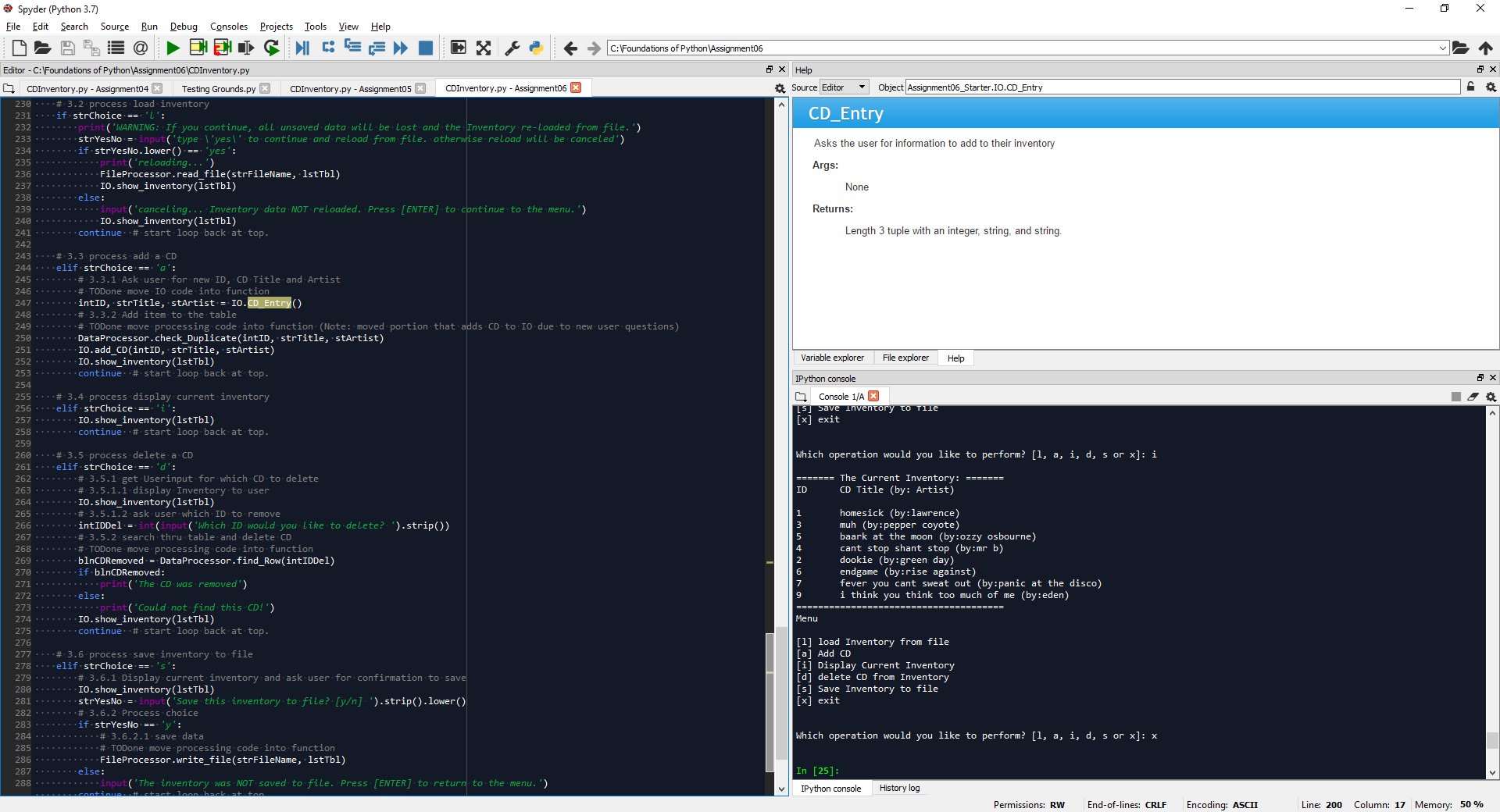


Figure 8- CDInventory.py Script (Part 5 of 6)

## Script(continued)

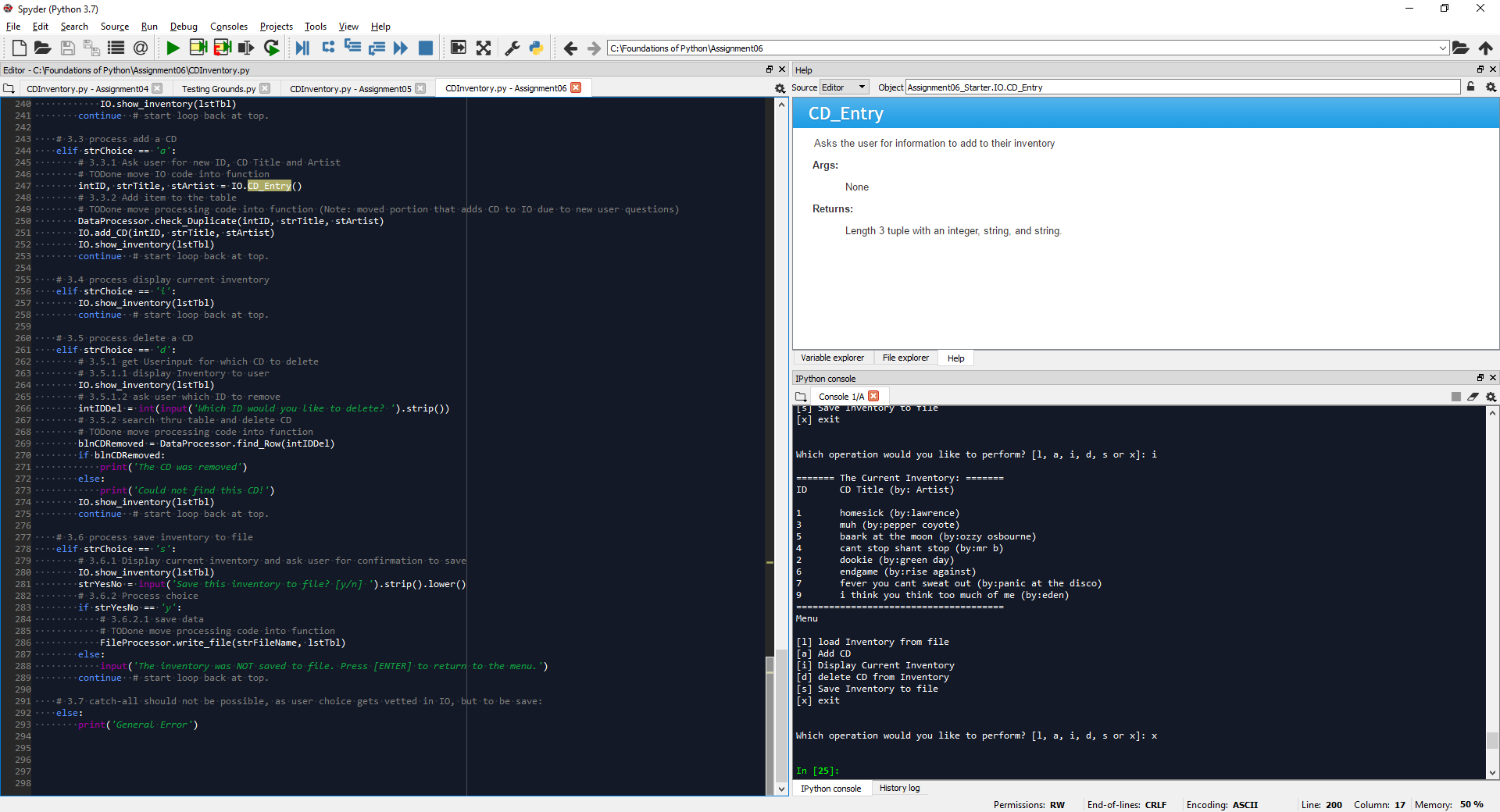
****

Figure 9- CDInventory.py Script (Part 6 of 6)