Name: Salvador Gavonel

Date: November 21, 2021

Course: IT FDN 110 B

Assignment 06

CD Inventory – Functions Script

**Introduction:**

Module 06 continues builds on what we have learned so far by developing on functions and on the elements that comprise it. We also take a look at classes and how they can collect functions within them to organize our code and categorize all of the functions that reside in the script.

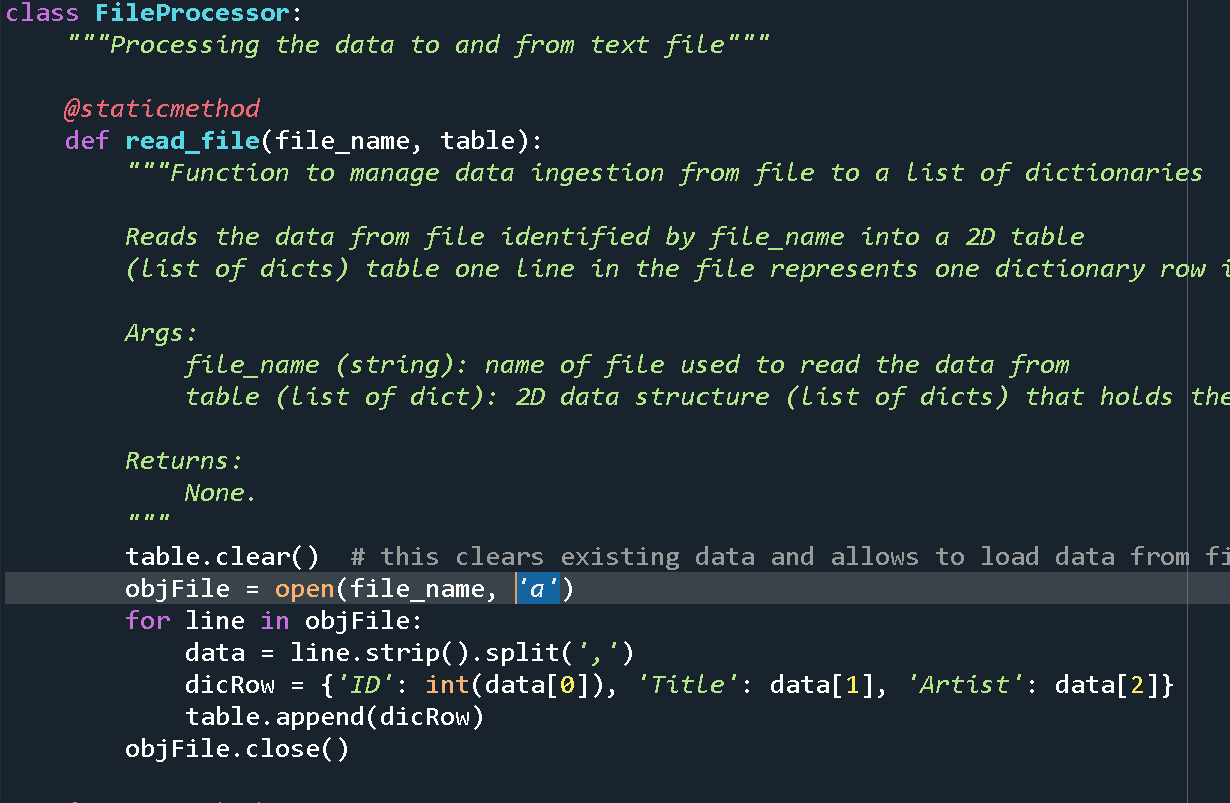
Assignment 06 continues with the CD:Inventory script to introduce functions and classes to put them into practice and organize our codes. The following paragraphs summarize how I managed to introduce them into the script and any insights I had during the exercise.

**Reading New Code:**

Opening the new ‘Assignment06\_Starter.py’, similar to the last assignment, I was even more surprised with the amount of code and text this file had. It was pretty overwhelming and I had to take a step back to take it all in. I was relieved to find that most of it were ‘doctrings’ that summarized what was going on in the classes and functions, but the class and function structure in the script took a while to understand. The TODO sections helped immensely to understand what needed to be done, but I still needed to grasp the whole script before I could start working on it. I started by making sure the menu was showing up, once that was complete, I would move on and start on the TODO (In hindsight I should have made sure that the code worked for every option before I made any additional changes).

**Running the Script Menu:**

I started having issues at the start of the script when I tried to run it and I kept getting error messages about the file not existing. Of course, the text file did not exist and the function to read the file had the read option (‘r’) in the open function, which does not create a new file as opposed as the append option (‘a’). I tried creating a file manually in the folder but the code wasn’t identifying it either, lastly I decided to change the read option to the append option in the FileProcessor.read\_file() function once and run it so that the code would create a text document identical to what it was looking for, if it didn’t exist. That did the trick.



Listing 1 – Running the script for the first time

**Data Processor Class:**

I might have missed critical information on how to separate the functions between DataProcessor and I/O classes, so I decided to place in the DataProcessor class all of the functions that add or delete info from the 2D Table, while the I/O class would hold all functions that took information or displayed information back to the user.

To move the add and delete entry code it was a simple copy and paste under the new DataProcessor class. I would have to name a new function and add a docstring to define what it is doing, but for the most part the code remained intact.

The one thing that did change and made me think for a while was how to setup the arguments for the functions. The delete\_CD function was very straightforward since the only argument needed here is the CD ID, but the add\_item function was a bit tougher, since it was in a way connected to the other IO function that would ask for the user inputs for the new entry. After some thought I decided to have the arguments be simply the string for ID, CD and Artist. I would think about this issue for the IO class function.

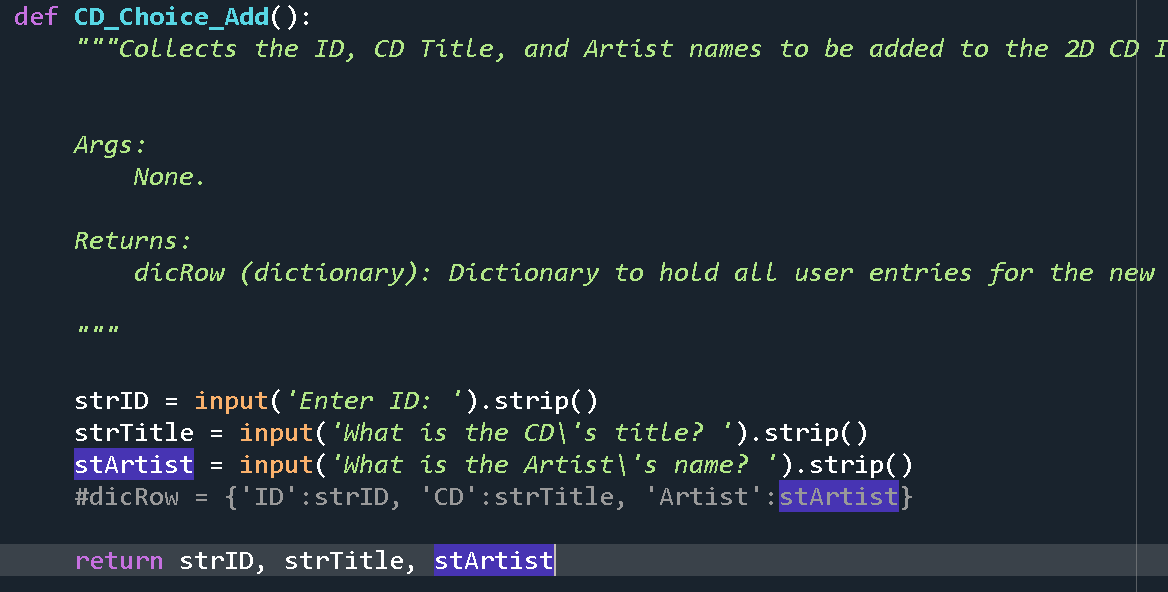
**File Processor Class:**

Again, I am very grateful for having an example of what a docstring and how a function would be correctly formatted with the existing FileProcessor.read\_file function. It gave me a good idea of how the rest of the functions would look after they were relocated under the new classes. Since the read file function uses the same variables as the code to write into the file, I used the same arguments to create the write\_file function, this was all I needed to do to have the new function ready.

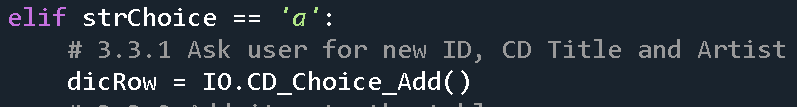
**IO Class:**

This class was the one that had the most functions in them completed and ready to run. The only function I added here was the one used to collect the new entry from the user. For this I copied and pasted the code to collect the info, and initially returned the ID, Title and Artist. In the main body I would then have the dicRow variable collect them, but I realized afterwards that this would convert the dicRow variable into a tuple instead of a dictionary.

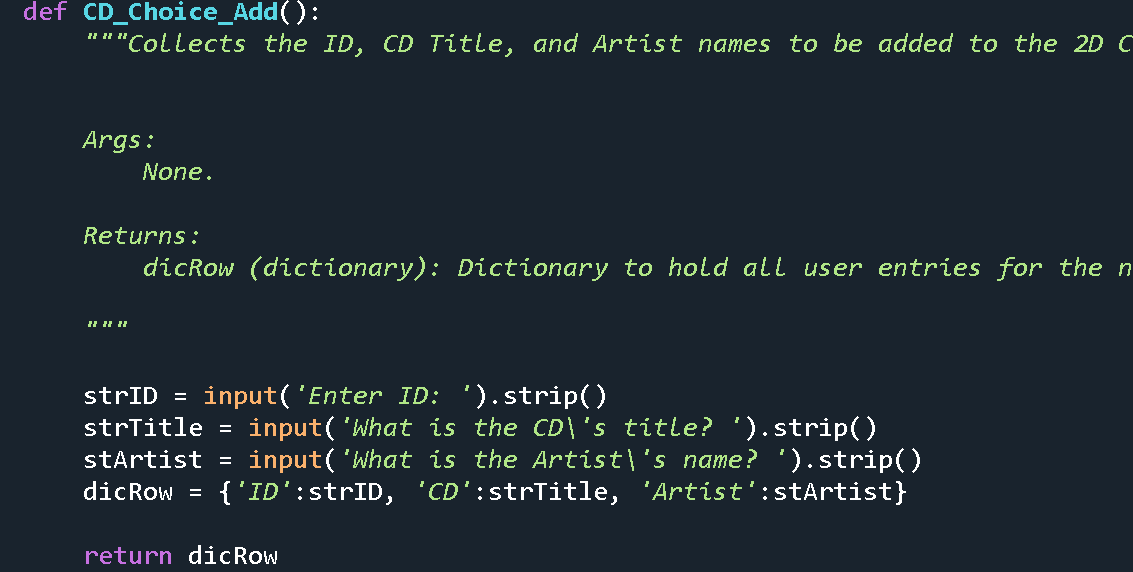
In this light I decided to then to define the dicRow variable in the function with the new entries and return the new dicRow as a dictionary instead of a tuple. Either way would have worked, but this change gave me a little bit of peace to know that each variable is being used as intended.



Listing - CD\_Choice\_Add Function before



Listing - Main Body Calling CD\_Choice\_Add Function



Listing - Return dicRow as a dictionary

**Main Body:**

Everytime I made a change in for a new function I would test it to make sure it worked before moving forward. Once that was complete I came back to a stripped main body that looked like it had been scavenged for code snippets. I cleaned it up to make it easier to read, but it still looks a bit disorganized in my opinion.

Once the functions were complete I then called the function in the main body, defined any arguments that the function would use, and any variables that would hold the return variables in the code. It seems simple but in actuality it took me a while to determine a format that would work in the script and that would make sense to read.

**Running Code:**

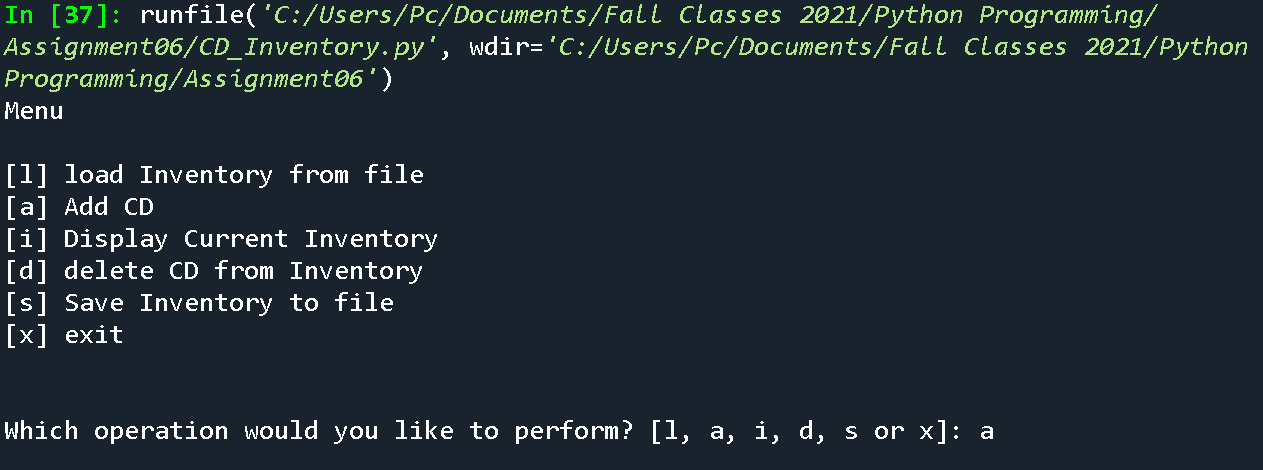


Figure - Running Menu (Spyder)

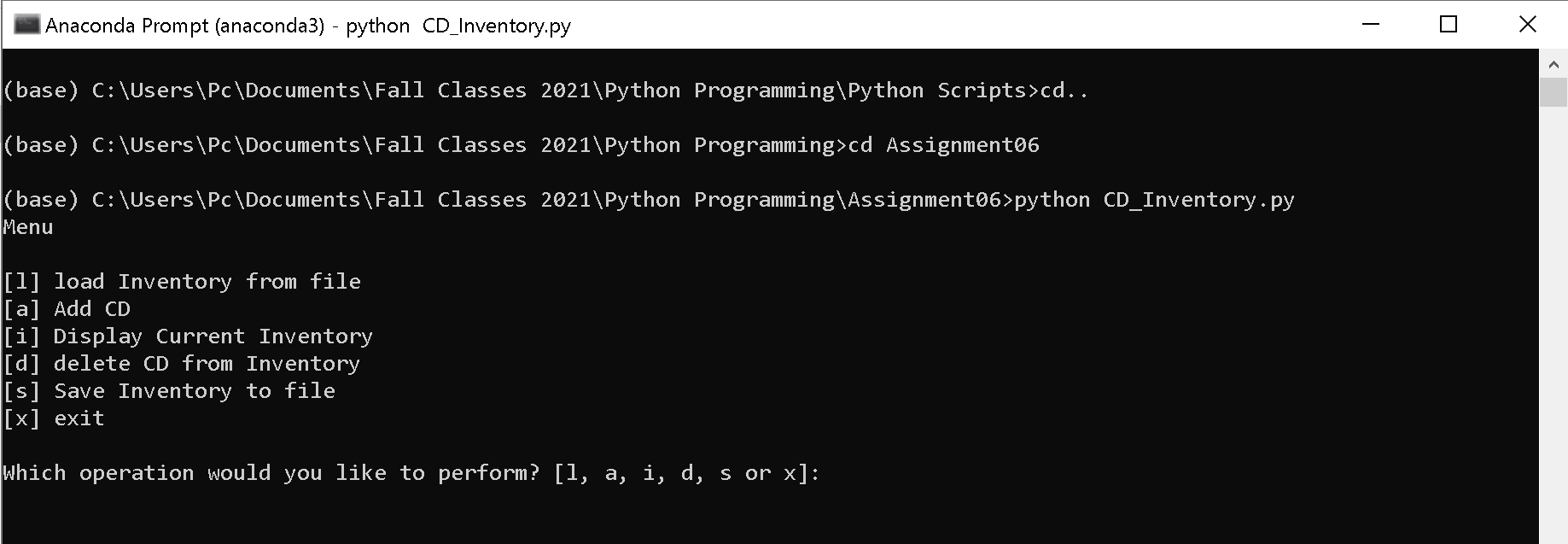


Figure 4 - Running Menu (Terminal)

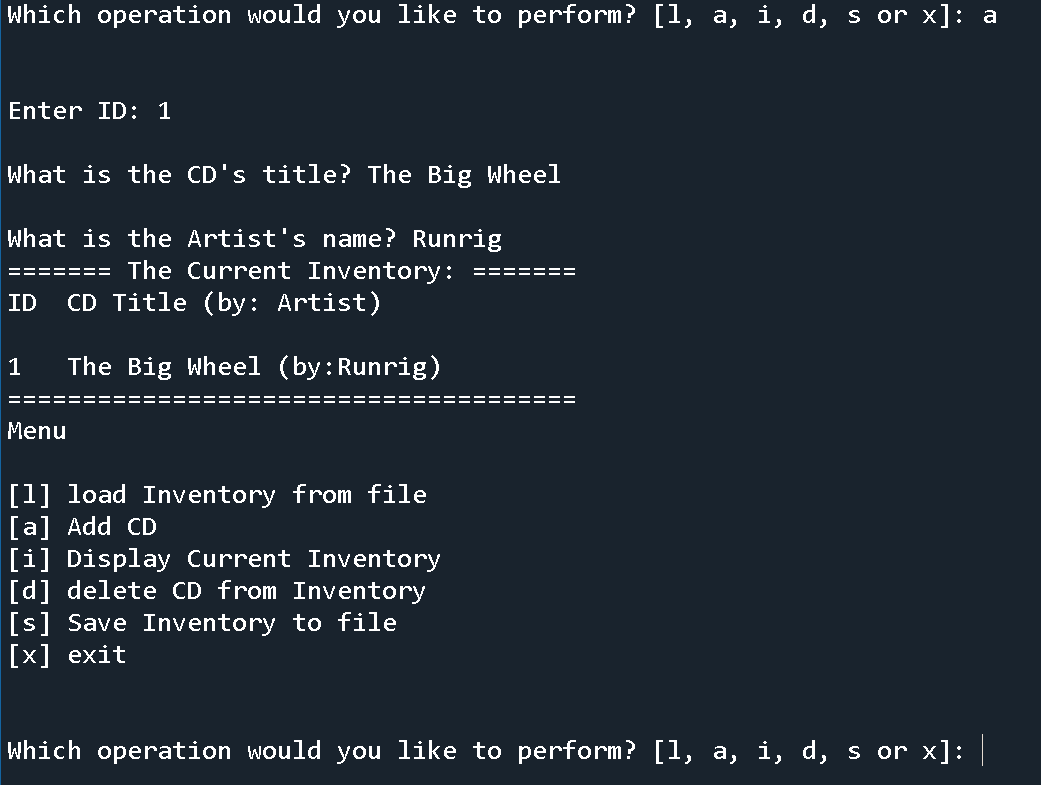


Figure - Entering New Data (Spyder)

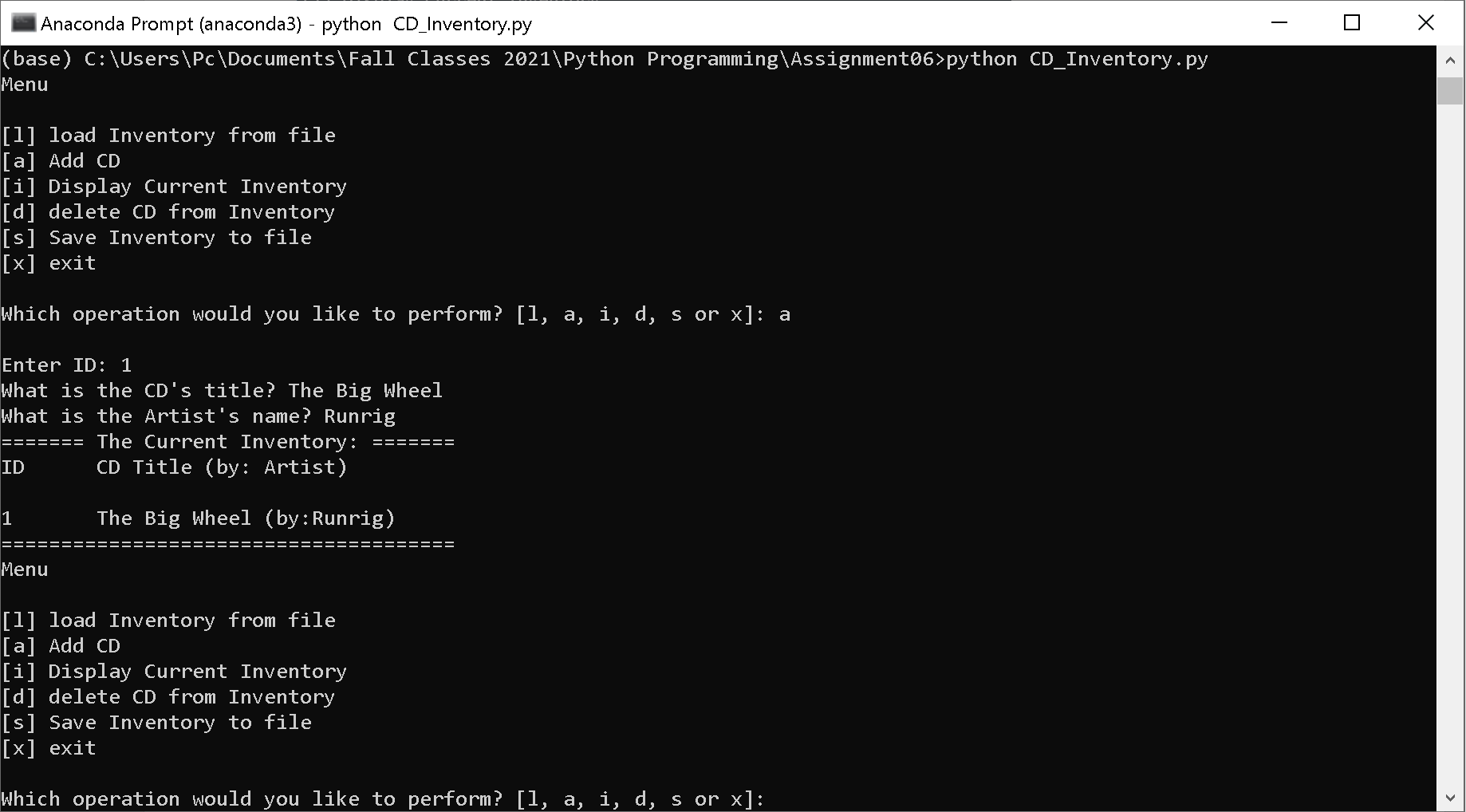


Figure 6 - Entering New Data (Terminal)

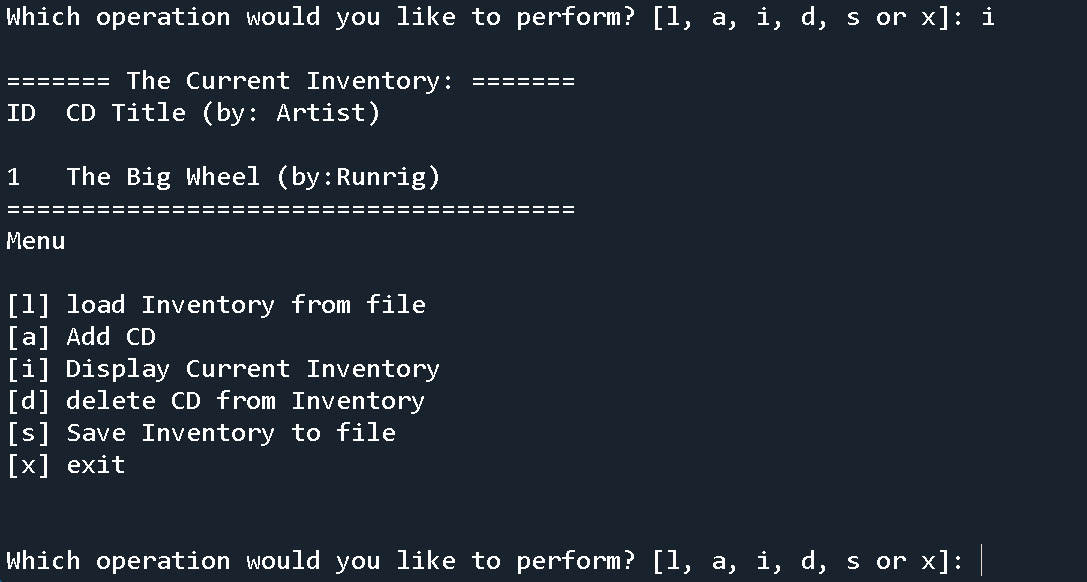


Figure - Displaying Data (Spyder)

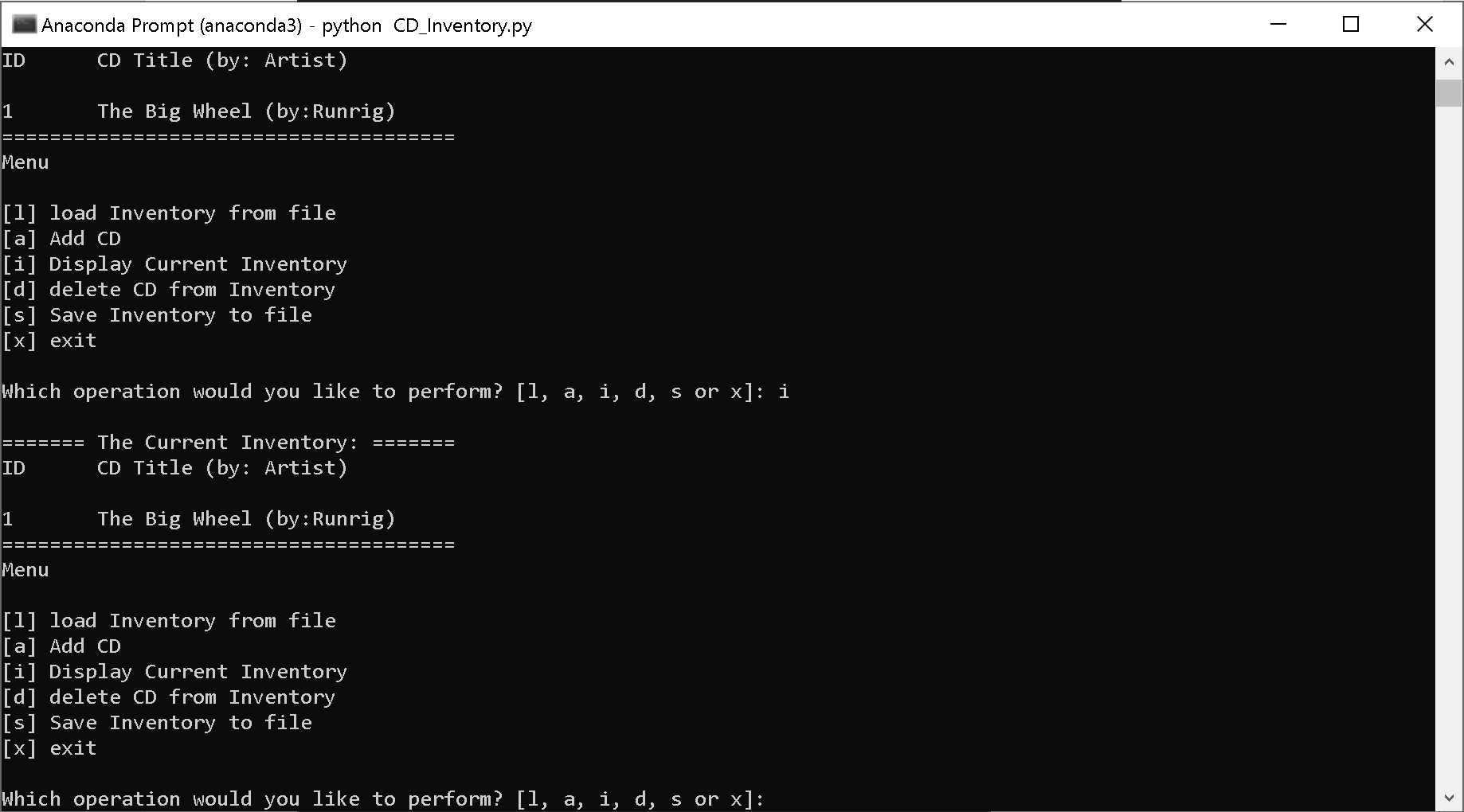


Figure 8 - Displaying Data (Terminal)

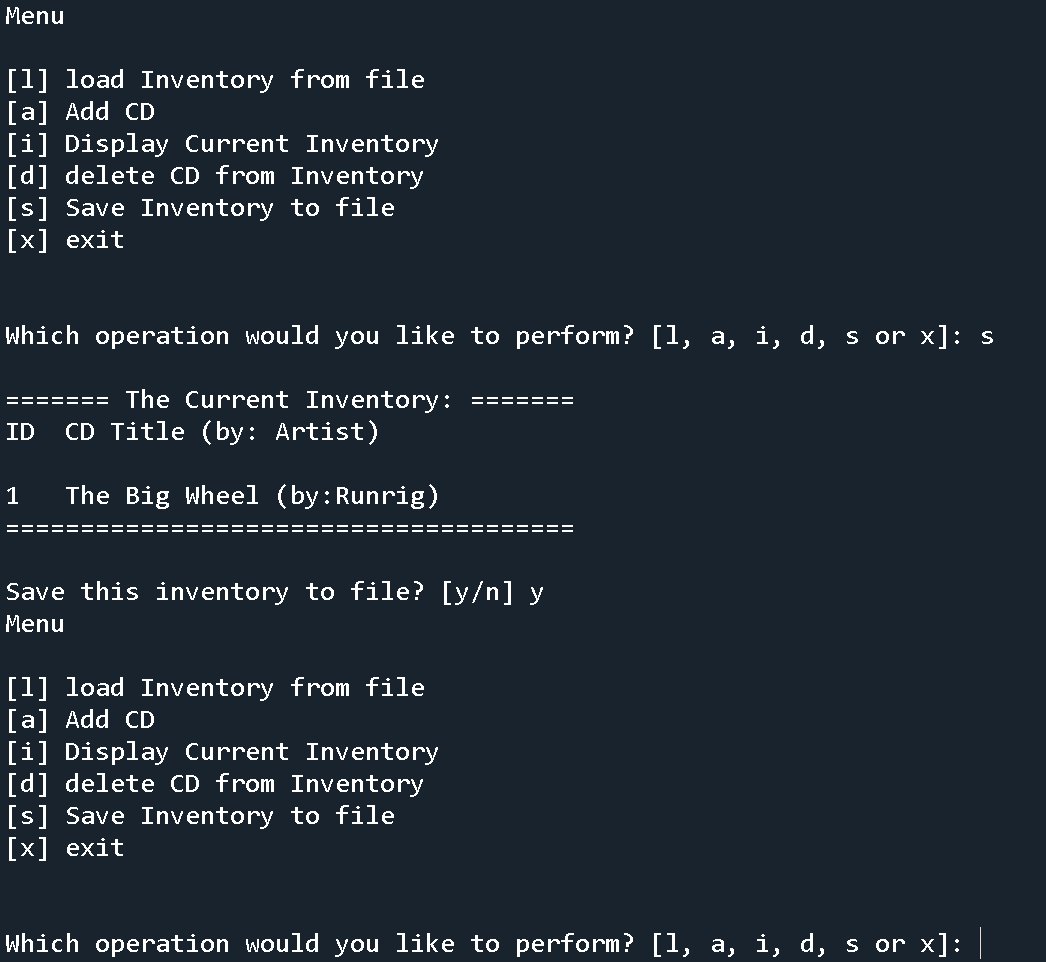


Figure 9 - Writing to .txt file (Spyder)

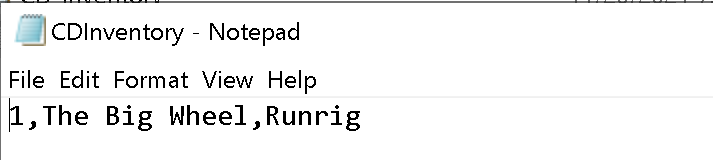


Figure 10 - .txt File with Entry (Spyder)

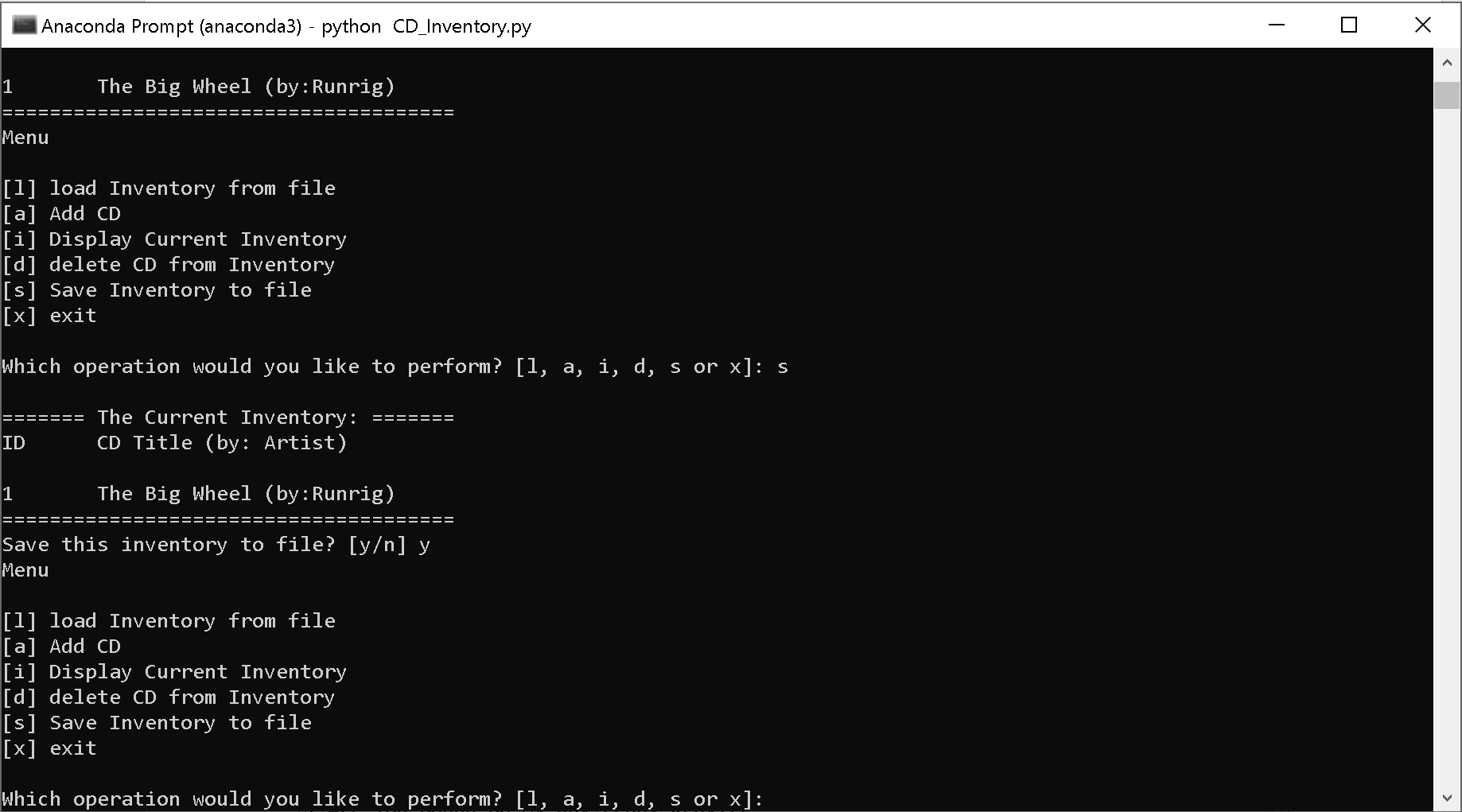


Figure 11 - Writing to .txt File (terminal)

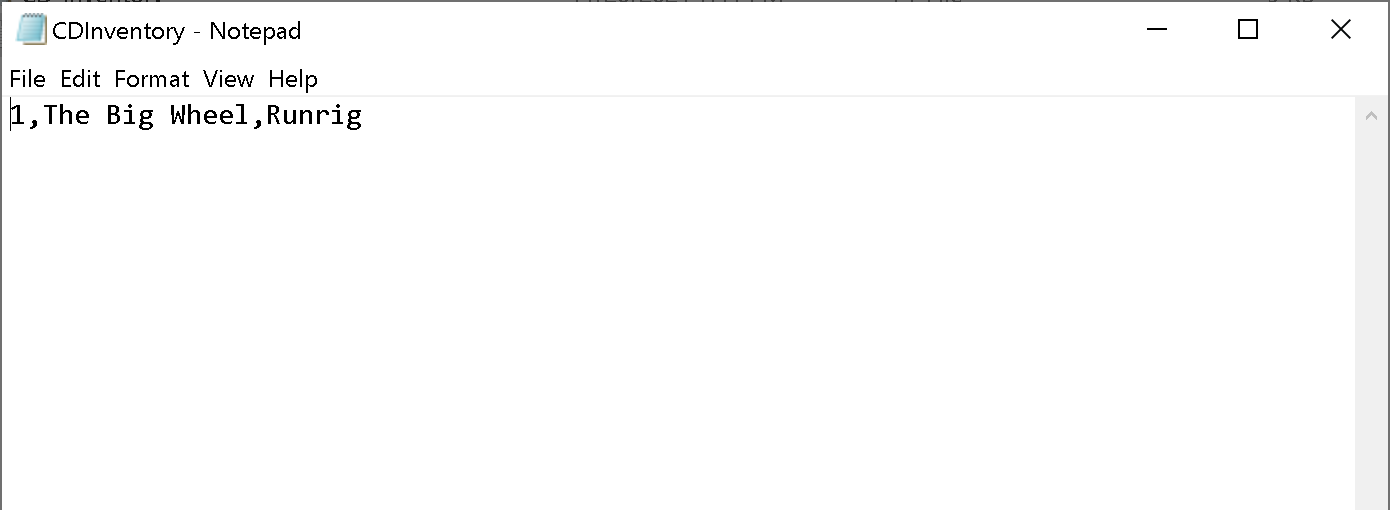


Figure 12 - . txt File with Entry (Terminal)

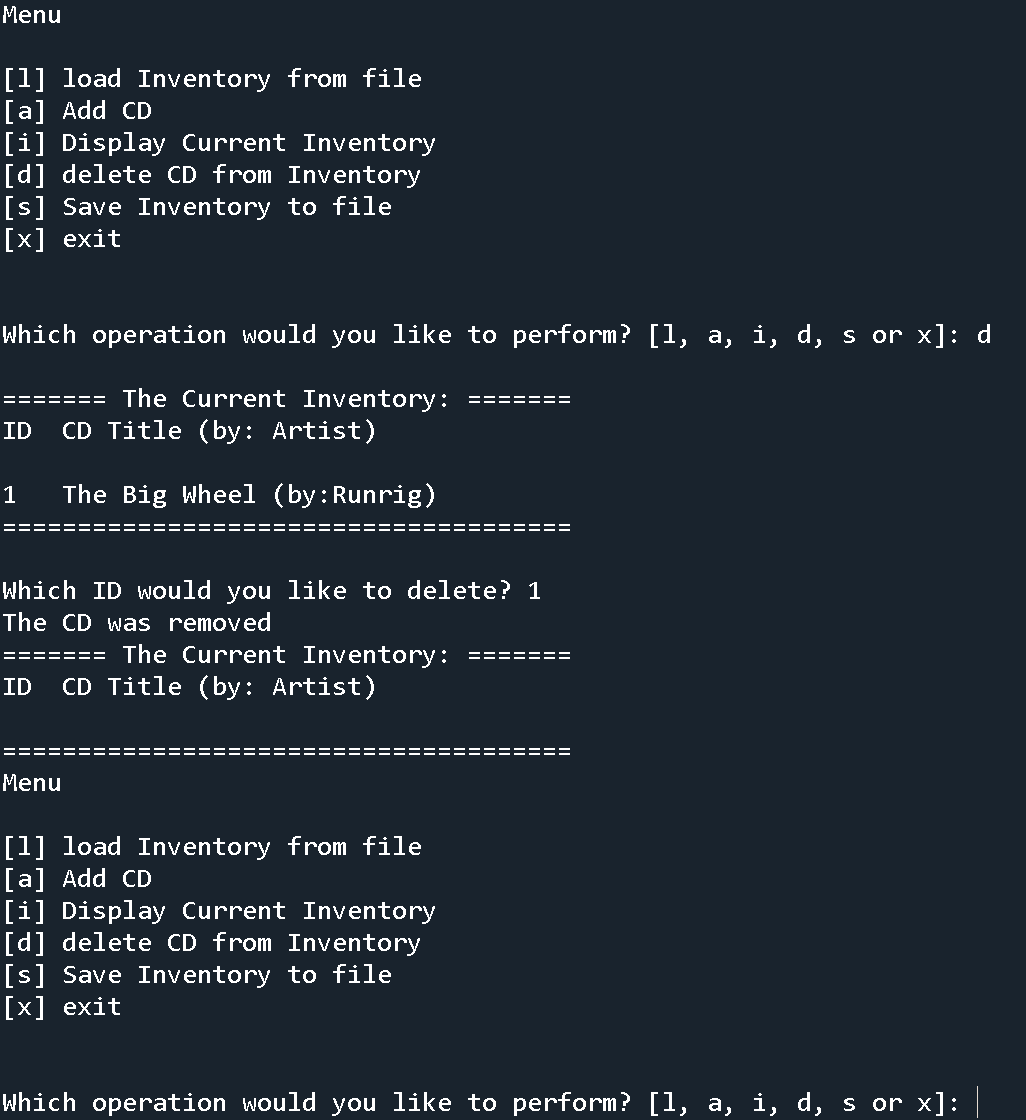


Figure 13 - Deleting Entry from Table (Spyder)

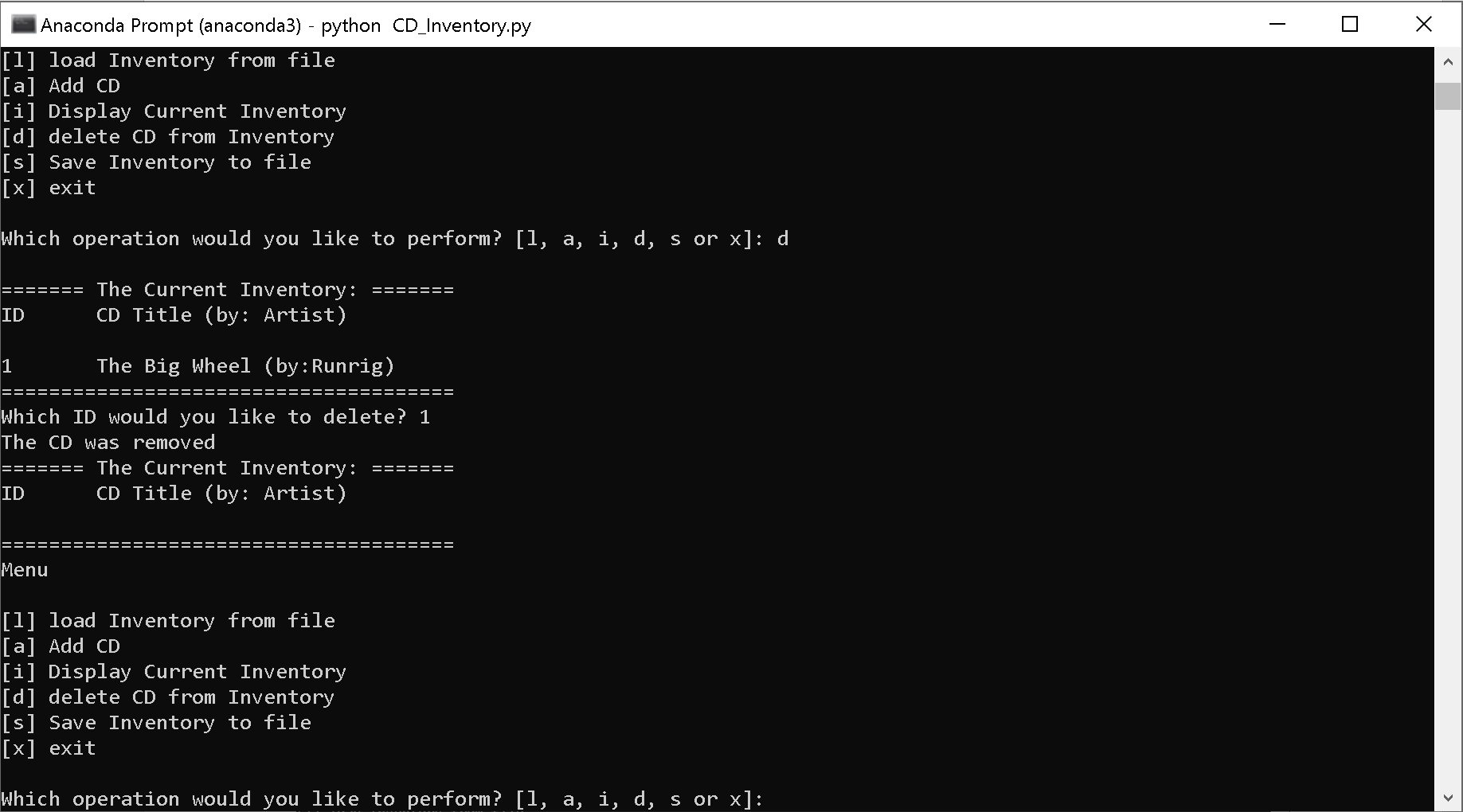


Figure 14 - Deleting Entry from Table (Terminal)

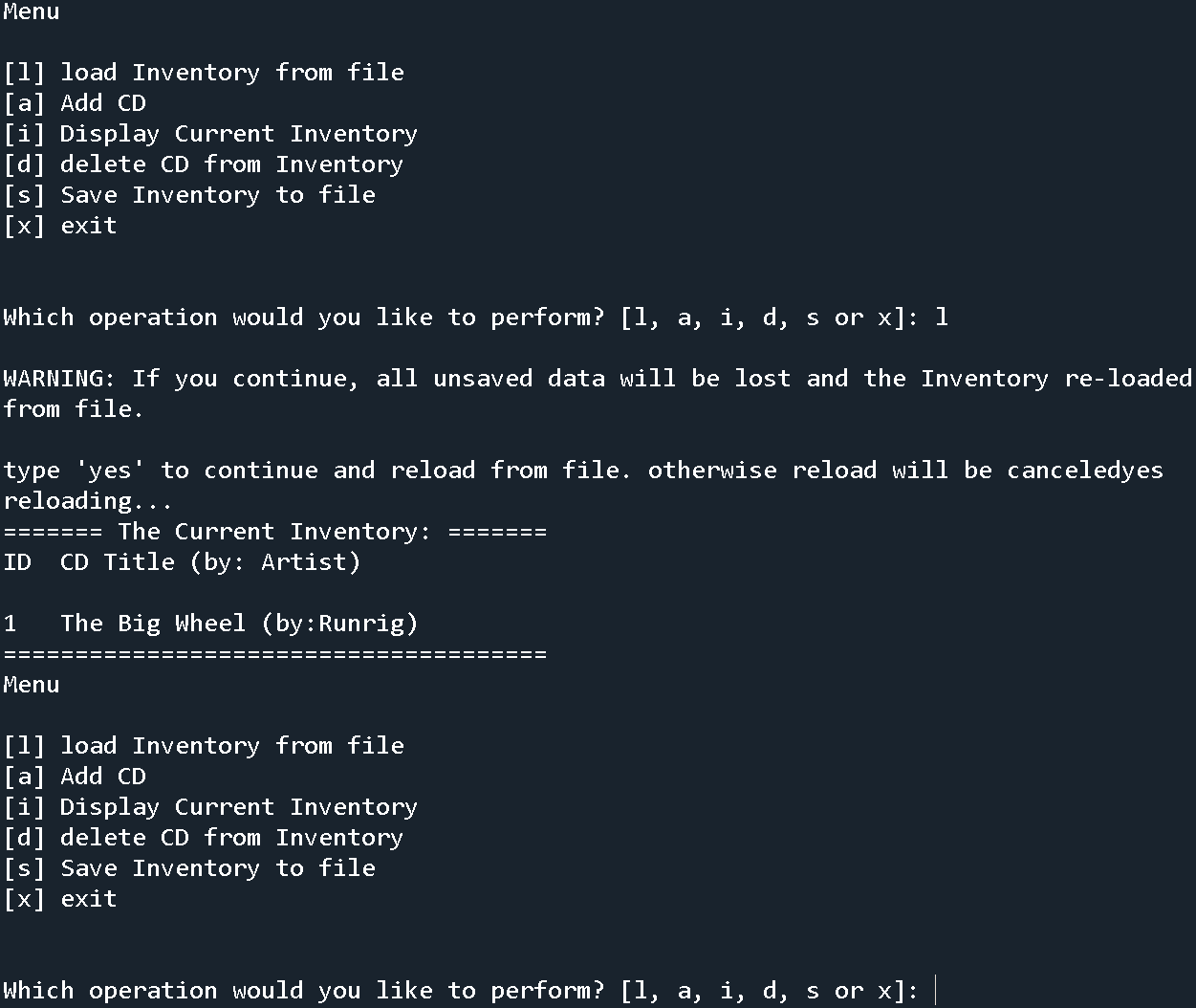


Figure 15 - Loading Data from a File (Spyder)

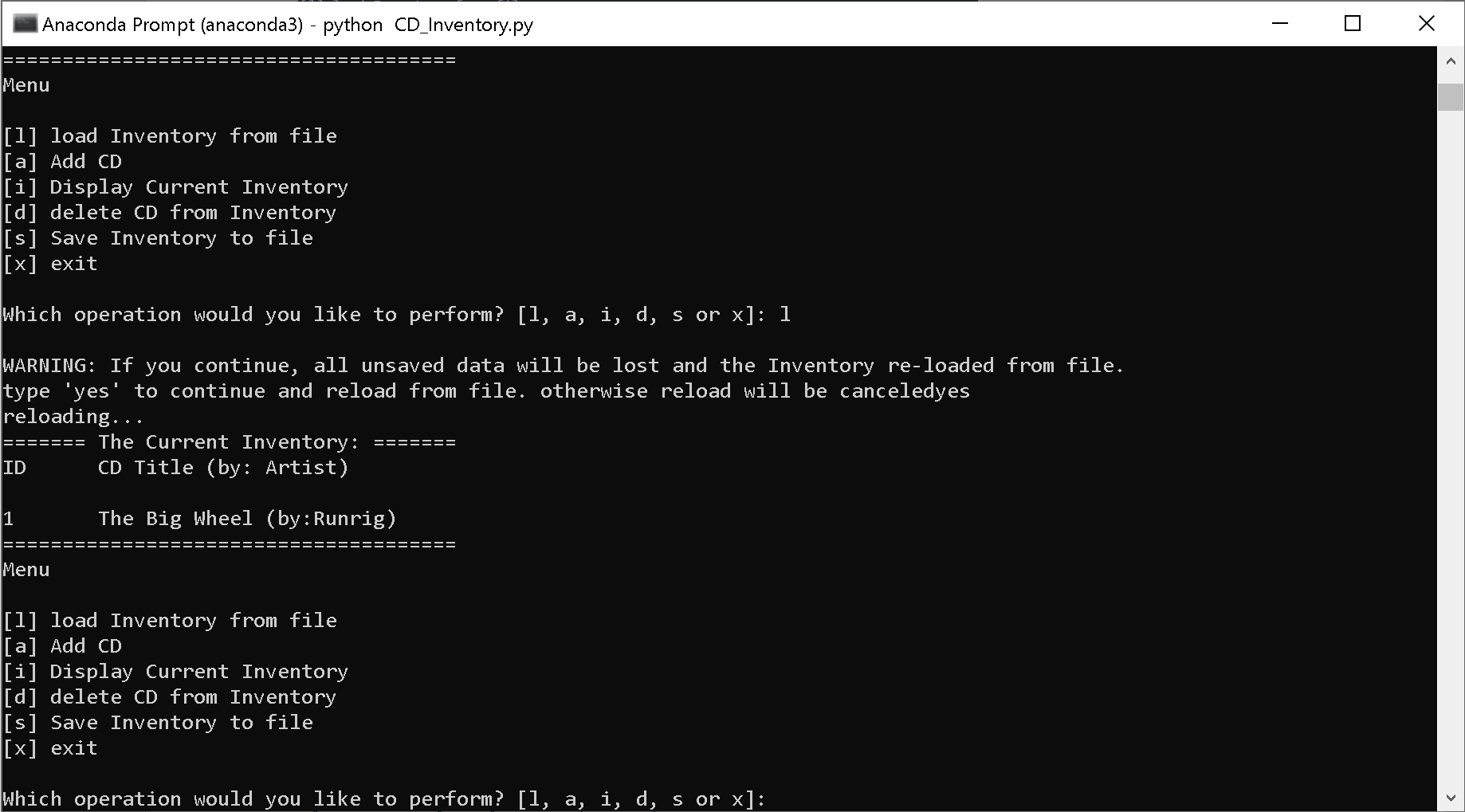


Figure 16 - Loading Data from a File (Terminal)

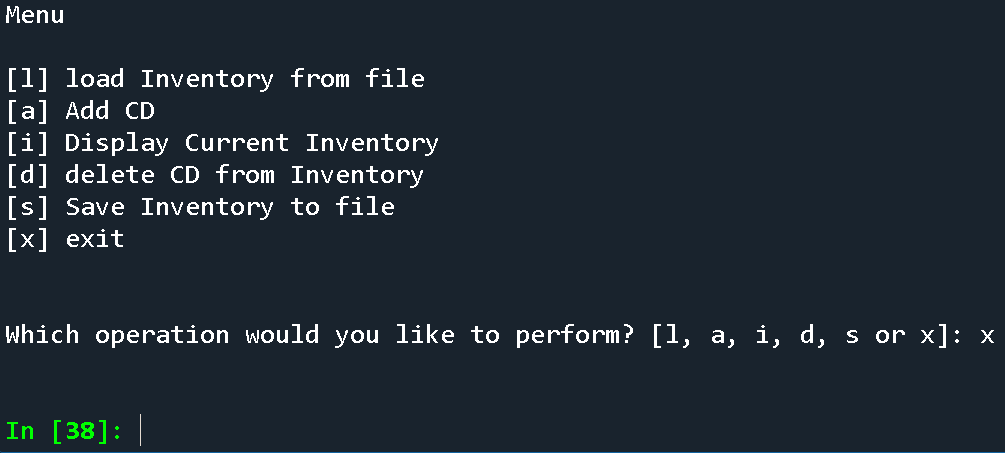


Figure - Exiting the File (Spyder)

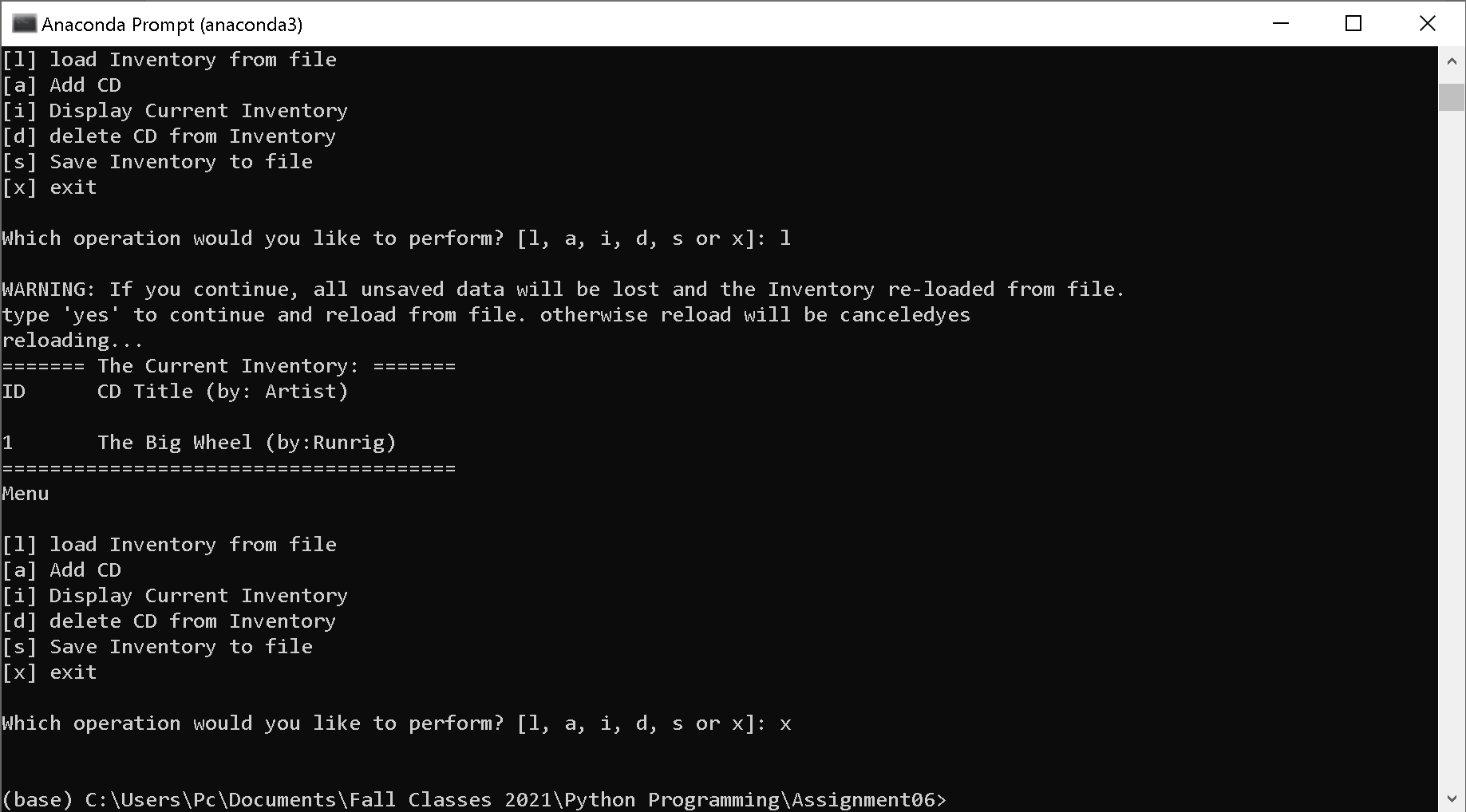


Figure - Exiting the File (Terminal)

**GitHub Repository Link:** [**https://github.com/Sgavon/Assignment06**](https://github.com/Sgavon/Assignment06)

**Summary:**

The CDInventory has evolved since the last assignment, with new functionalities and a new table of dictionaries instead of lists. While the code is better and the script can do additional things, it has also become more cumbersome and harder to read. Separation of concerns and functions(), although they are briefly covered in this assignment, will be needed to create more longer and more complex code.