Name: Salvador Gavonel

Date: November 27, 2021

Course: IT FDN 110 B

Assignment 07

CD Inventory – Pickling and Exception Handling Script

**Introduction:**

Module 07 introduces new methods to read data from external files, ‘pickling’, and exception handling. The former help users store and load data more efficiently with code, and the latter aid users continue their script running while catching errors and displaying them back.

Assignment 07 continues with the CD:Inventory script to introduce these new concepts and put them into practice. The following paragraphs summarize how I managed to introduce them into the script and any insights I had during the exercise.

**Converting to Binary Files:**

By far the hardest coding exercise I’ve had, I struggled since Lab07\_B to understand how pickling works and how to write/load files using them. It was a good exercise, because I feel a lot more confident using them now, it just took a lot longer than I expected and took more patience that I’m used to having.

Using text files and all the different read options shown in the modules were easy to implement in the code, but the pickling introduction did not have as many information as I needed to complete the assignment. I understand that it was intended to have us research online on pickling and exception handling, and it was quite the reality check for me.

The two functions that I needed to modify with pickling were the IO.read and IO.write functions. The IO.write was very simple: I just needed to switch the write function for the pickle.dump() function, from then on the code would work to write the inventory into the file.

The read function was the hardest to develop out of all the modifications I performed. This was because I did not have a good understanding on how pickle files are read. I initially kept the ‘for line in objFile’ loop but I kept getting error messages on ‘unpickling stack underflow’. I was not able to understand what this meant and looked for another way around it.

I realized that for the pickle.load() to get every line in the file, I needed to keep the file open, so I first moved the open() function as early in the function as I could. I was getting to the point of trying that I would just resolve this issue by copying the pickle.load() multiple times and use multiple variables to hold each line, then use the lists to load the memory list. The issue with this of course was that I would only get as many lines back as variables that I created and it would be a very messy function.

I finally found an article on loading multiple objects in stack overflow that gave me the answer I was looking for: a ‘while True’ loop to continue loading the pickle.load() function until there were no more objects in the file. It also had an exception handling code to keep it from terminating the program (2 birds with one stone!)

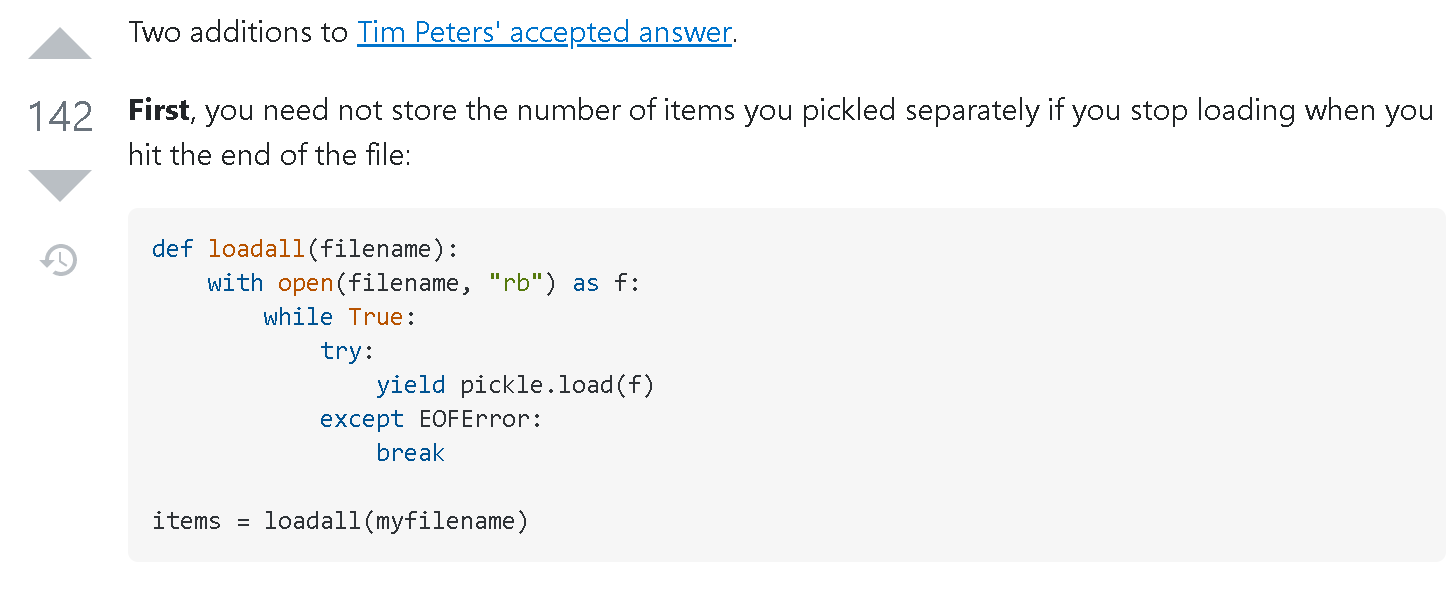
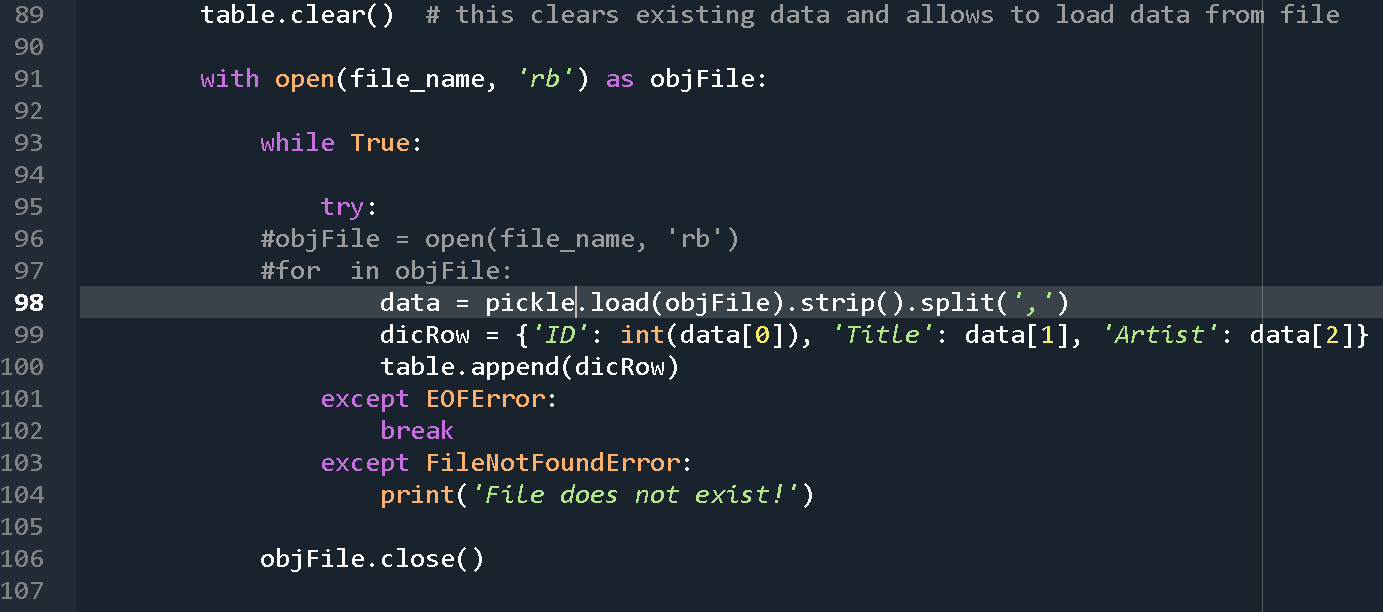


Figure - <https://stackoverflow.com/questions/20716812/saving-and-loading-multiple-objects-in-pickle-file>

Once I used this code in mine, I was able to run both functions with no problem in the script.



Listing - Final IO.read() function code

**Exception Handling:**

This section of the assignment was easier to implement that the binary file conversion. The LAB exercises that we practiced in the module were similar to what I needed to implement here.

**File Processing Errors:**

The first part I modified was the File Processor functions. Here I just added an exception for a ‘filenotfounderror’ in case the file that was called was nonexistent. I place this code on both the ‘read’ and the ‘write’ functions for good measure, from my attempts I know that if the file didn’t exist it would only get caught in the ‘read’ function because it is the first function to check if there is a file.

I mentioned previously about the exception handling code I adapted from the stack overflow webpage, which helps end the read code once all objects have been read from the binary file instead of popping a pickling error.

**Input/Output Errors:**

For the rest of the errors, I used the main body to write the exception handling code. From my different attempts to write exception handling code in the functions I’ve noticed that I get double error messages: one from the function, and another from the main body. I decided to keep the error messages to a minimum and work on the main body instead (I have a feeling this is not the way to handle exceptions and would be better to handle it within a function).

I have to recognize that I had to think for a moment on what types of errors users could run into: a lot of the code handled possible errors through if/else statements such as typing in a wrong character or responding incorrectly to the ‘yes/no’ prompts that the users got. The one instance there would be an error that I recognized was when inputting the ‘ID’ number for the table entry, this affected two options: delete and add new CD. I used a ‘ValueError’ exception to catch this error and display back that the user needed to input an integer rather than any other value type. I also added a ‘continue’ clause to bring the menu back to the user if the answers were incorrect.

**Running Code:**

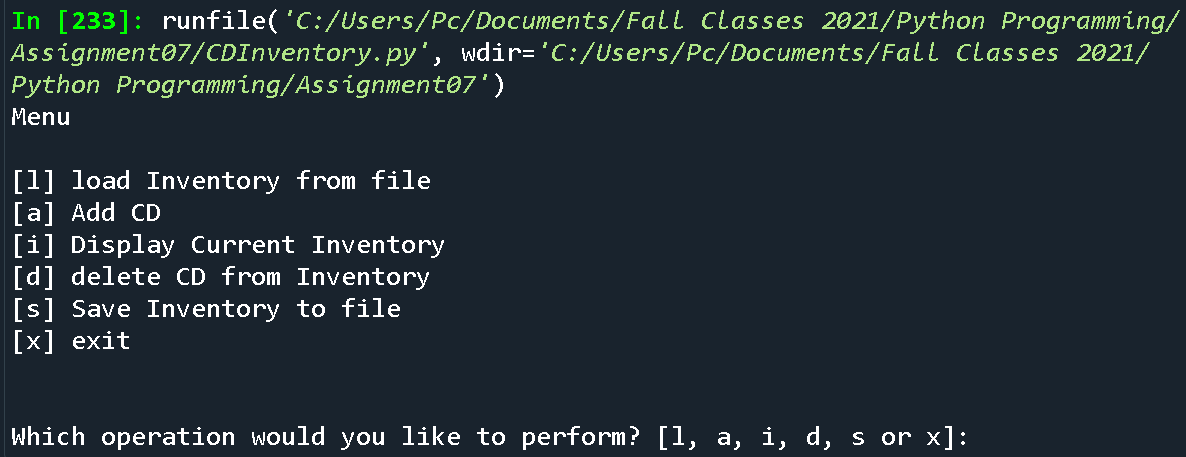


Figure 2 - Running Menu (Spyder)

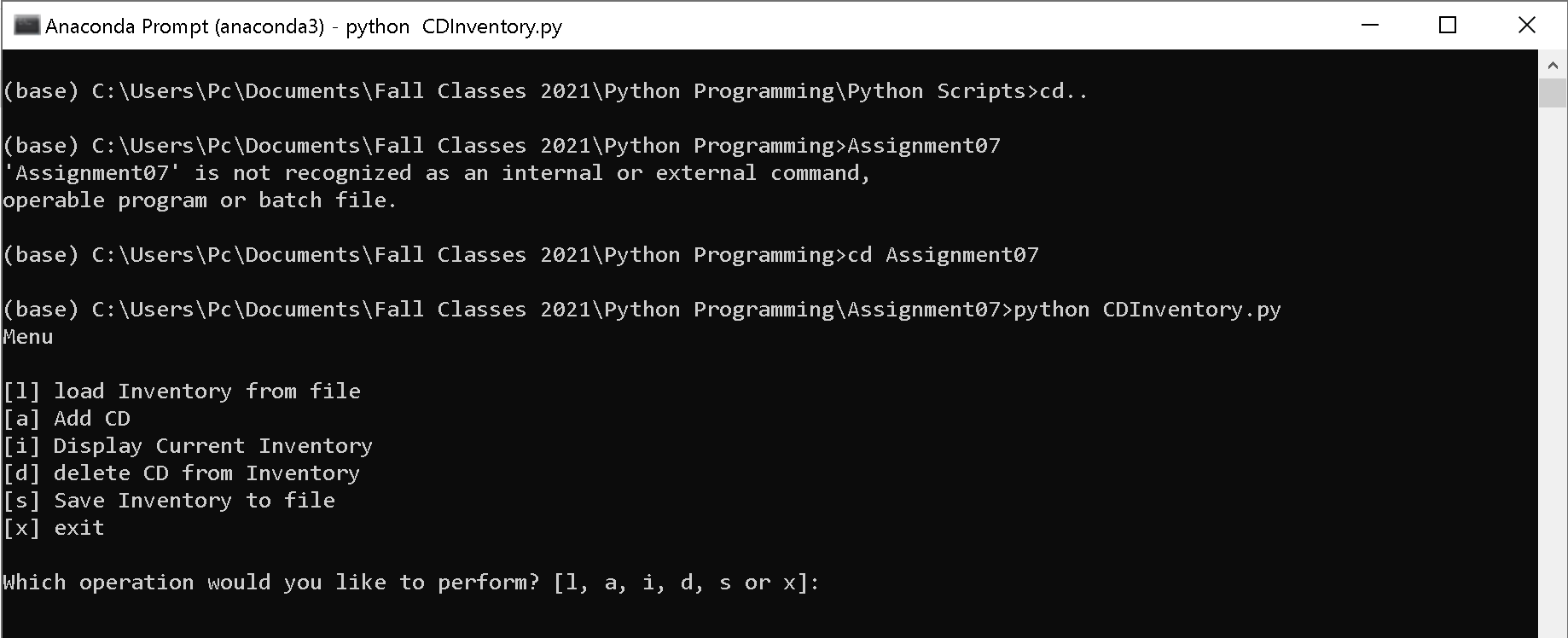


Figure - Running Menu (Terminal)

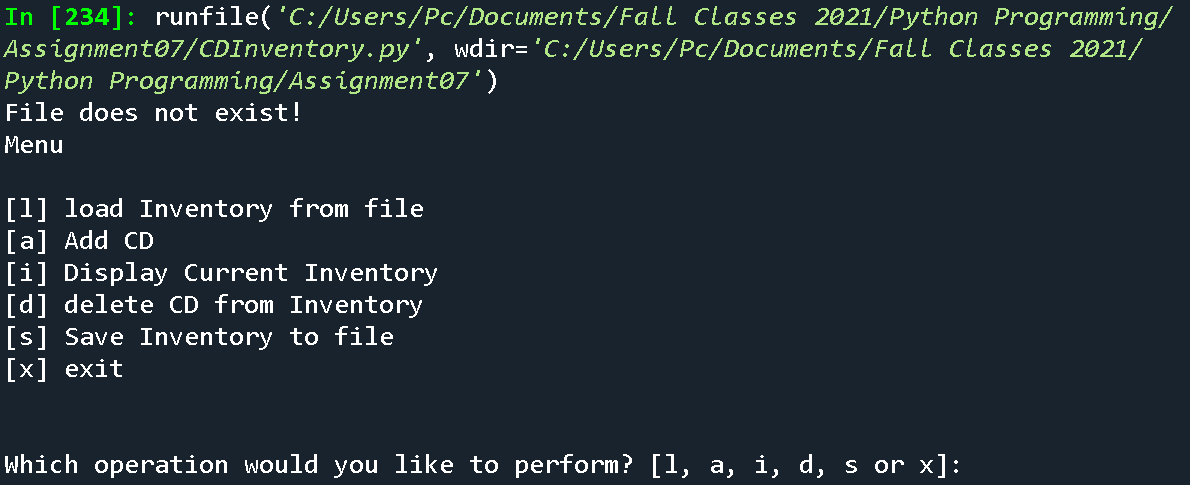


Figure 4 – Error Handling 1 – File not found (Spyder)

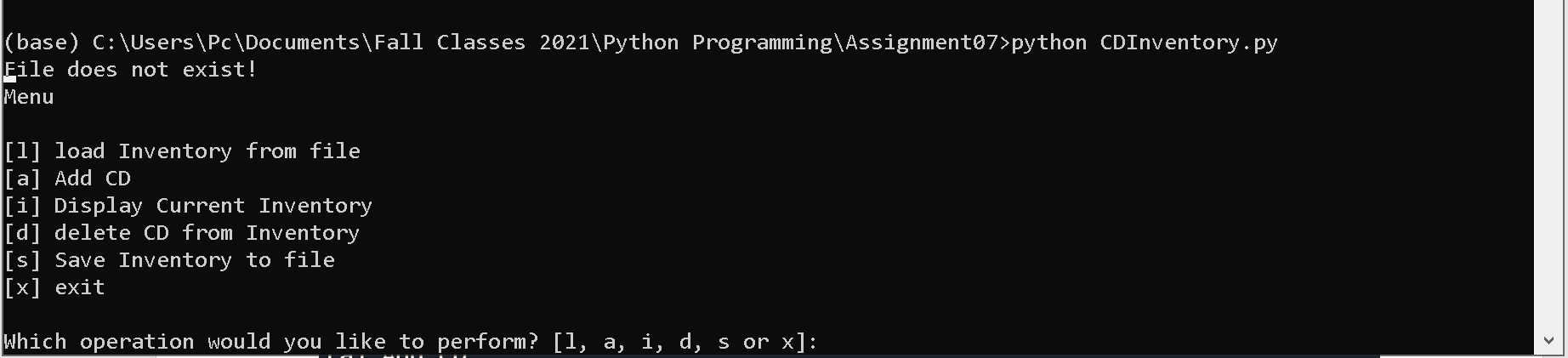


Figure - Error Handling 1 – File not found (Terminal)

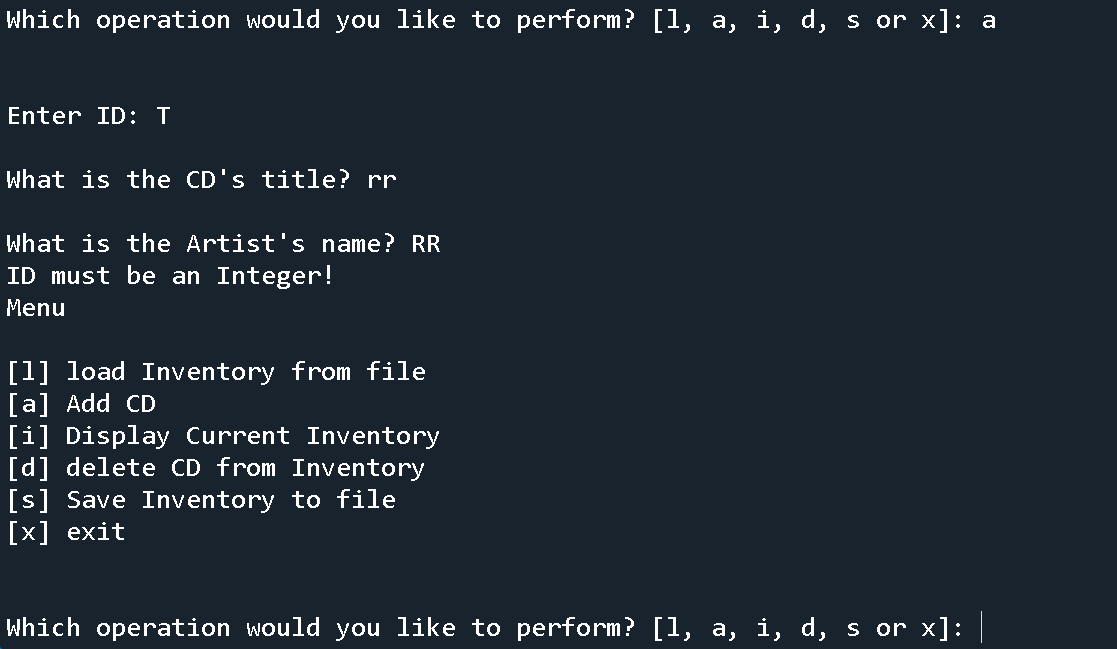


Figure 6 – Error Handling 2 – Invalid ID when adding new entry (Spyder)

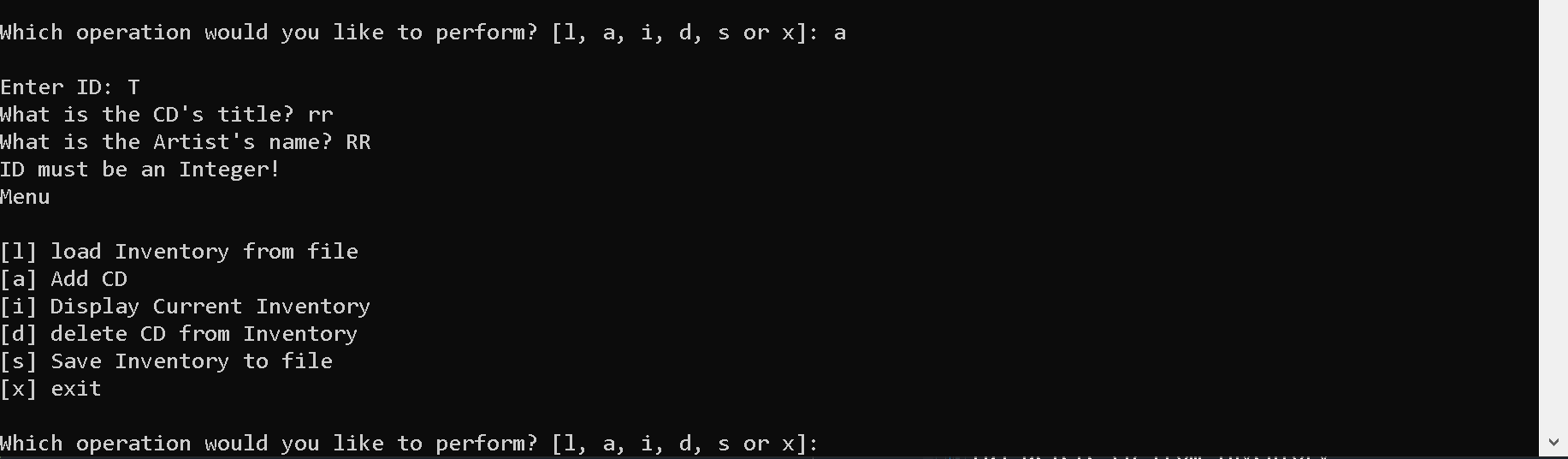


Figure - Error Handling 2 – Invalid ID when adding new entry (Terminal)

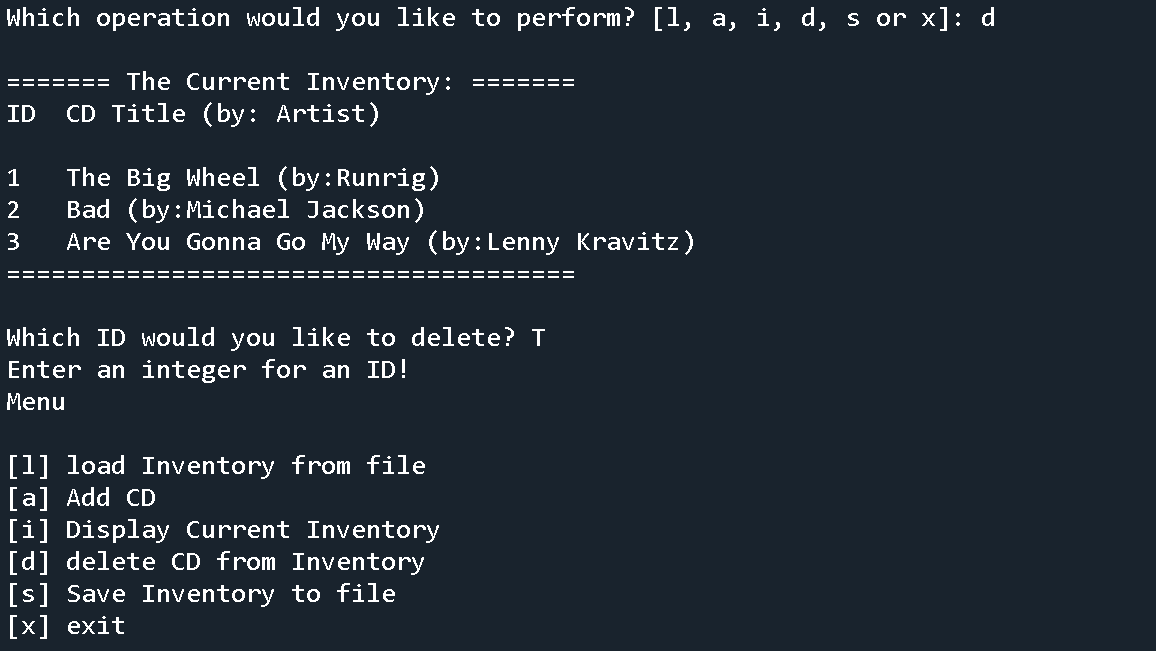


Figure - Error Handling 3 – Invalid ID when deleting entry (Spyder)

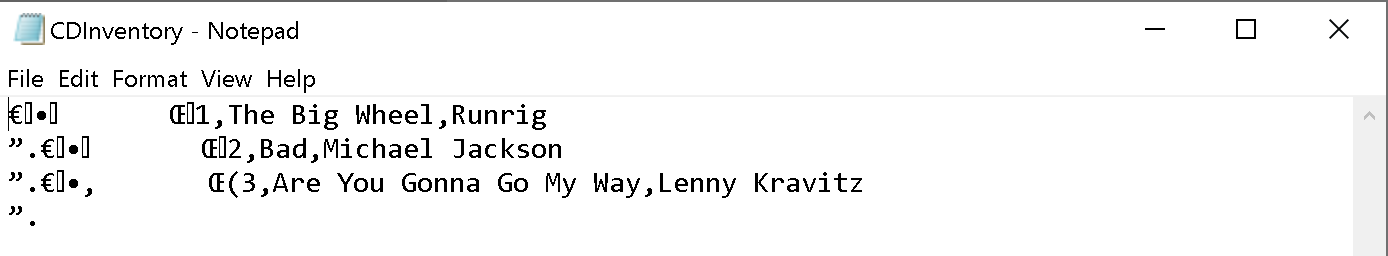


Figure - .dat File with Entry

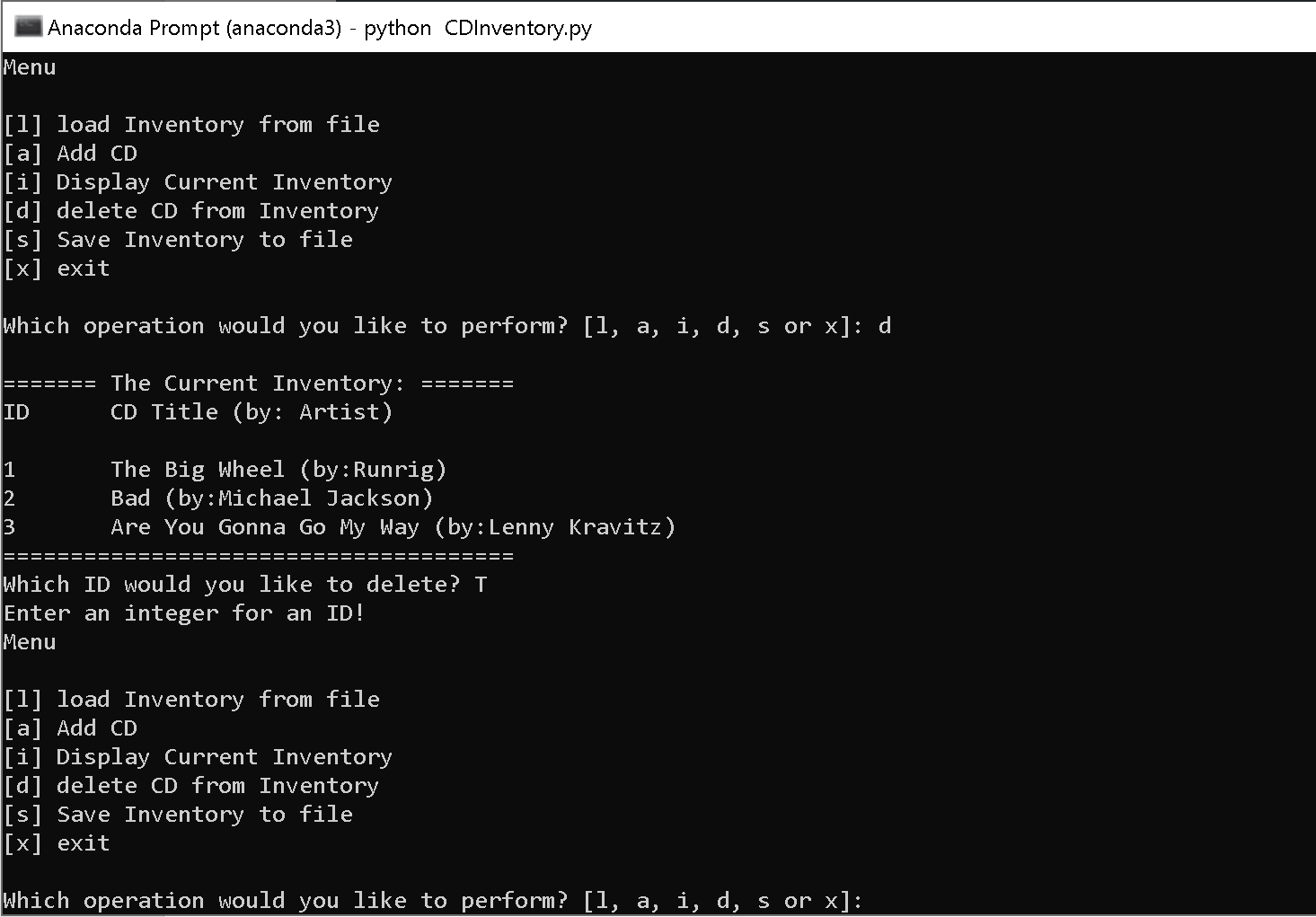


Figure - Error Handling 3 – Invalid ID when deleting entry (Terminal)

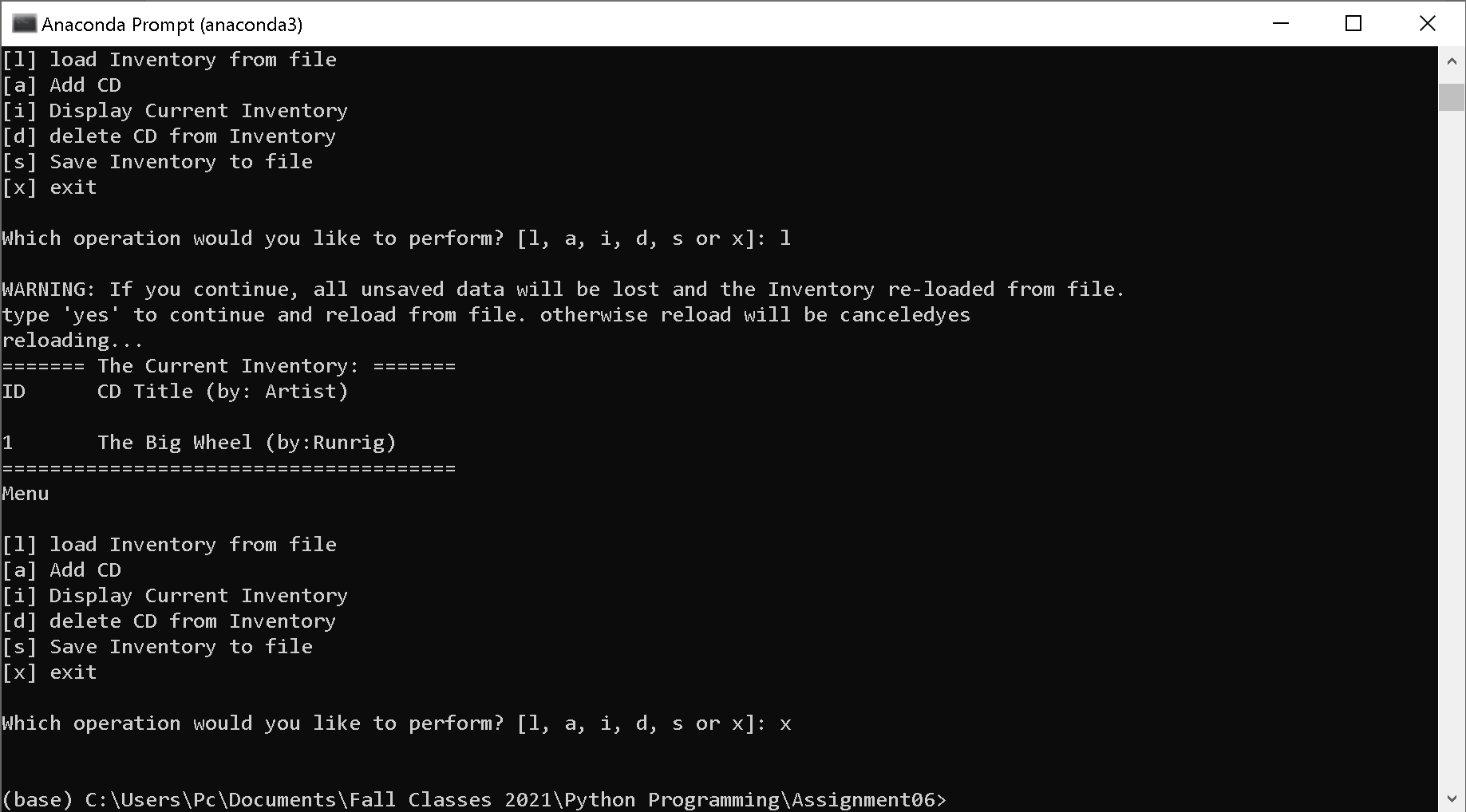


Figure 14- Exiting the File (Terminal)

**GitHub Repository Link:** <https://github.com/Sgavon/Assignment07>

**Summary:**

By far my most challenging assignment, I’ve recognized that I cannot think very many steps ahead (or behind) to understand what is happening with my code. I’ve found myself adding a print() statement every two lines to understand what I’m processing and how the data is transforming. Writing from and into text files is probably my biggest weakness at the moment, so I look forward to any options to simplify my code when handling text files.