Arianna Morgan

November 27, 2022

Foundations of Programming: Python

Assignment 07

Classes and Functions

Introduction

In this module, we continued to learn about working with text files, and we learned about binary data files and structured error handling. Binary data files save data in binary code, which requires less processing time and memory, so it is ideal to use if it is not necessary to have the data saved in an easily readable format. Structured error handling is the addition of code that allows errors to be handled within the program – giving the user the opportunity to change their inputs so that the code can run successfully – instead of the program terminating with errors. This week, we improved upon last week’s script by adding structured error handling and saving the data to (and loading it from) a binary data file instead of a text file.

Details

There are four situations for which I added structured error handling that were necessary to avoid having the program terminate with errors: when the program loads a file (in case the file does not exist), when the program attempts to display current inventory after trying to load a non-existent file, when the program asks for an ID when adding a CD (in case they input a non-integer), and when the program asks for an when deleting a CD (in case they input a non-integer). I also added structured error handling to prevent the user from using the same ID for multiple CD entries. This situation wouldn’t terminate the program, but it would cause problems if the user wanted to only delete one CD but it shared an ID with another.

There are two places where the program looks to read data from the external file: in the read\_file function and at the beginning of the Processing section in the main code. In both places, I added a try ahead of the code that looks to read the file data and I added two “excepts” – one for a File Not Found error and one for a general error – that provide information on the error. In the main code, I added a pass for these exception cases so that it will move on to the main menu. This will allow the user to go ahead and add CDs, which can be saved down to a file that can be used with the program going forward.

When the user tries to load inventory from a file and the file does not exist, the program terminates when there is no additional structured error handling within the show\_inventory function that is called after the attempt to load file data. This is because we have returned a table with None type when attempting to load nonexistent file data. In the show\_inventory function, we are using a for loop on that NoneType table, which does not work. I added a try/except in show\_inventory that prints that there is no data to display if there is an exception.

In the add\_inventory\_menu section where the user is providing their inputs for the new CD to be added, I added a while loop for the strID input that uses try to see if the input can be converted to an integer type. If it can, the while loop is broken, and it goes on to ask for the strTitle and strArtist. For the case where it cannot, I added an except for the Value Error case that prints that the ID must be an integer and one for a General Error that prints that there was a general error. If there is an error, it stays in the while loop, so it goes back to asking the user for the strID.

The structured error handling for the deletion case occurs in the main code, which is where the user inputs the ID of the CD they want to delete. As with strID above, it uses a while True loop that breaks when the user input can be converted into an integer. It has the same exception handling as well.

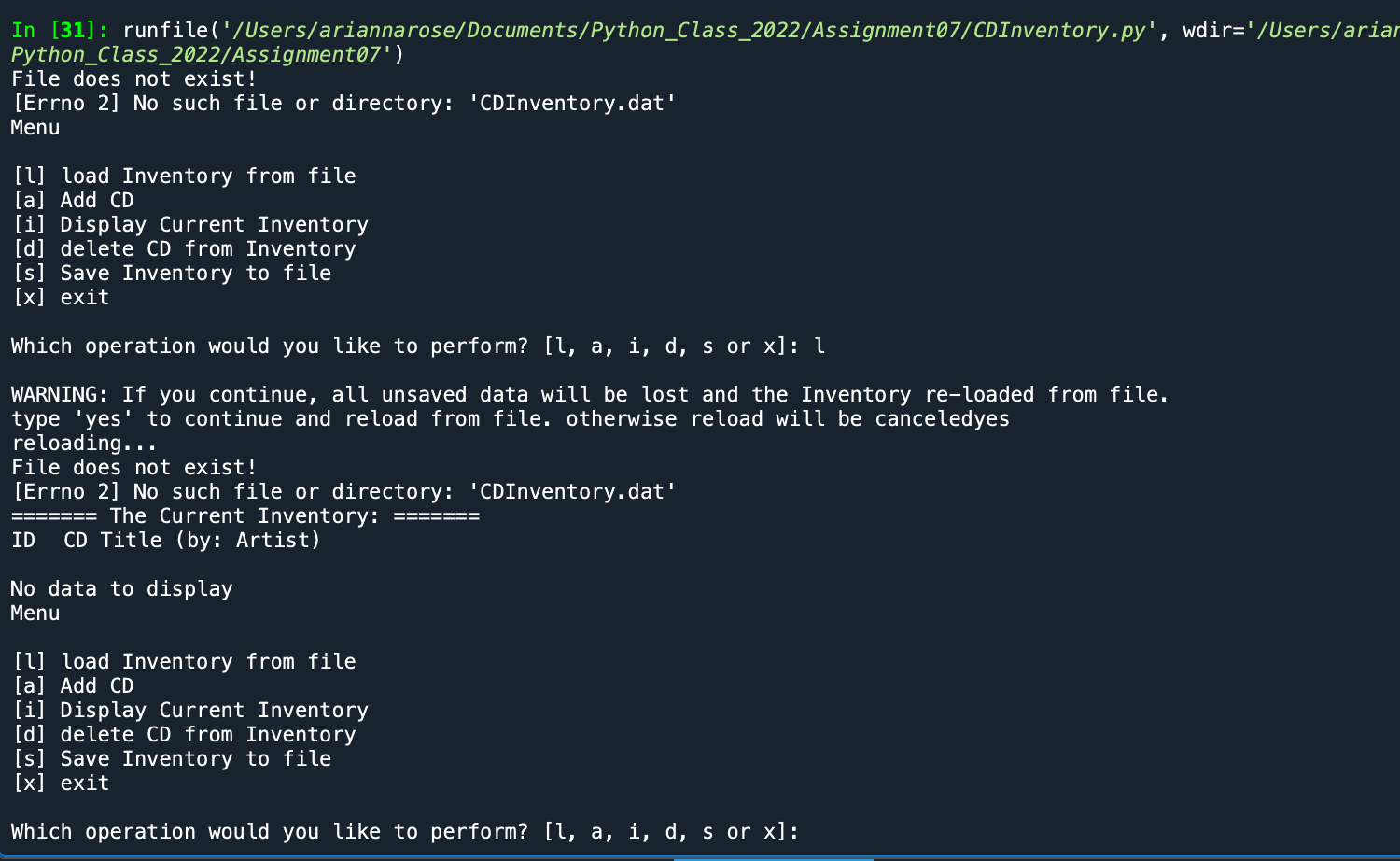
For my additional structured error handling case to prevent the duplicate use of an ID, I added a try/except structure to the add\_inventory function that used two for loops to check if the user-input integer for the new CD matched any existing ID in the table. If it does, it raises an exception and lets the user know that the ID is already in use and that the CD will not be added, bringing the user back to the main menu. Otherwise, it adds the CD to inventory as expected.

To update the code to use a binary data file instead of a text file, I first changed the name of the strFileName that we define in the Data section to a .dat file instead of a .txt file. Then, I used “with open() as fileObj:”, where the arguments of open are file\_name and a string representing how we are interacting with the file, which was ‘rb’ to read the file and ‘wb’ to write to it (overriding existing data). To read, we are using pickle.load(fileObj) and assigning that to table. This saves the data in the file to a table within our function. We return this table, and in order to use it in the main code, we assign it to our lstTbl when we call our read\_file function. To write, we use pickle.dump(table, fileObj). This saves our table parameter to the binary data file.

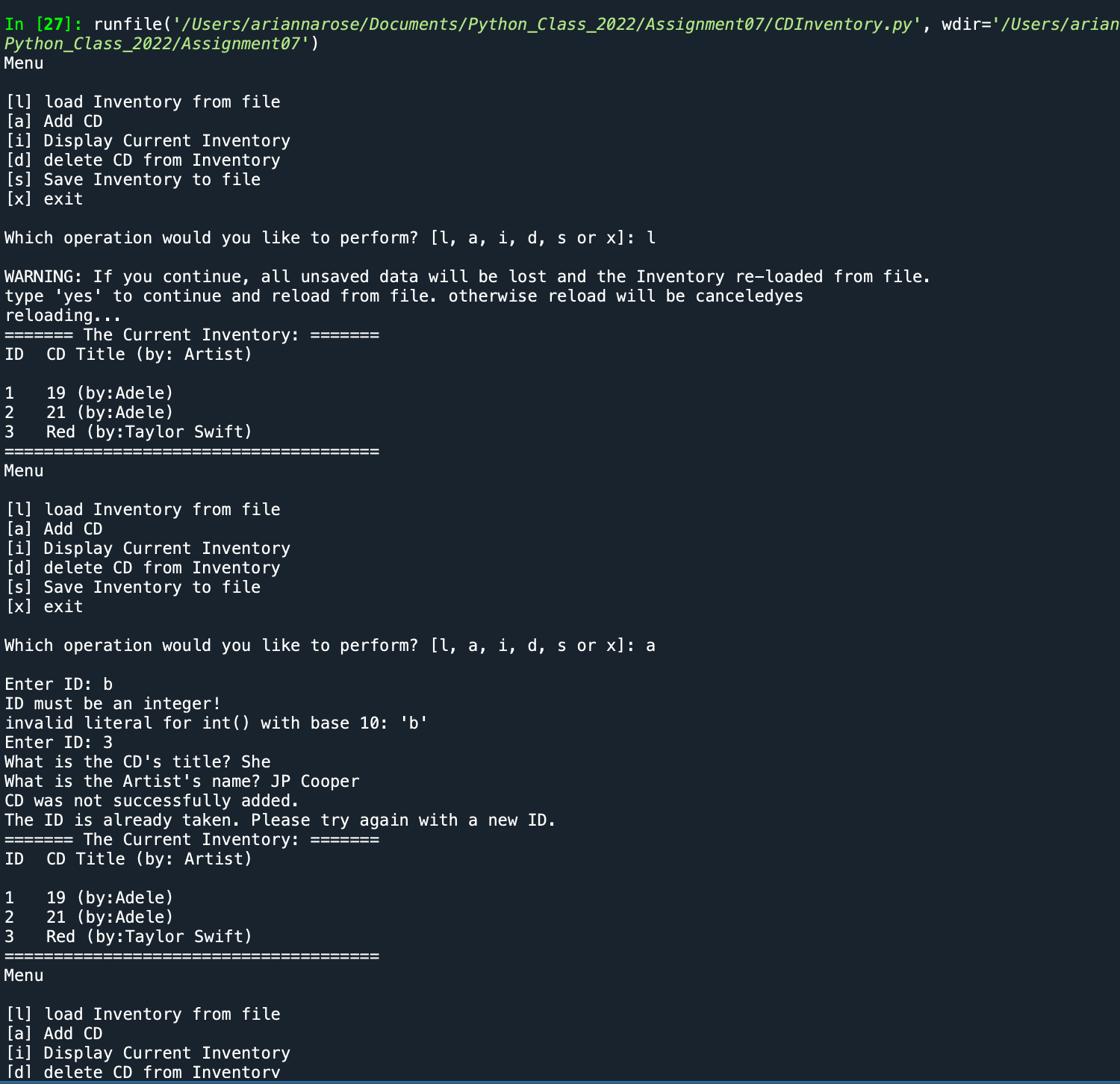
Below are screenshots of my code working:

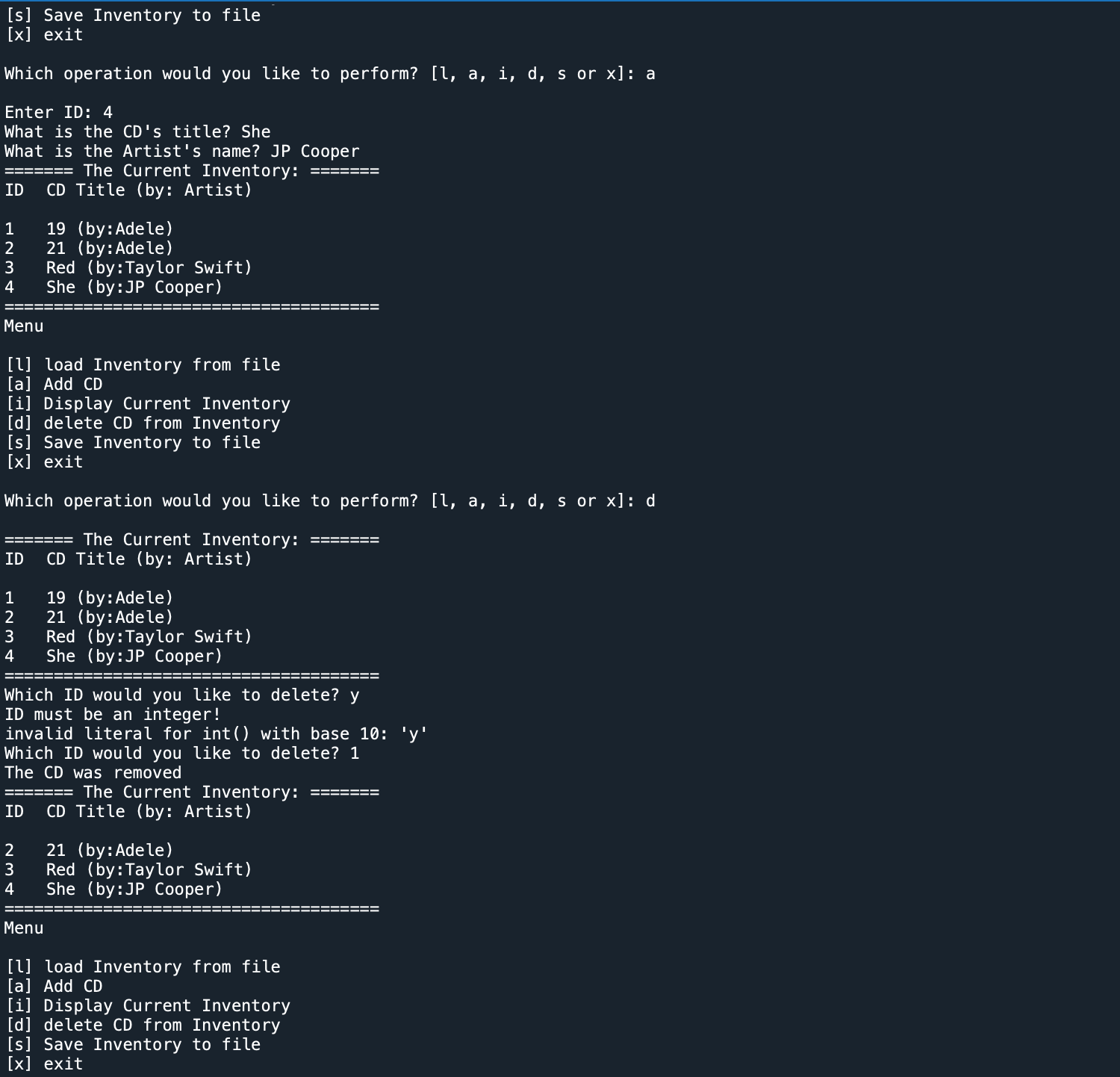
In Spyder:

When the file doesn’t exist:

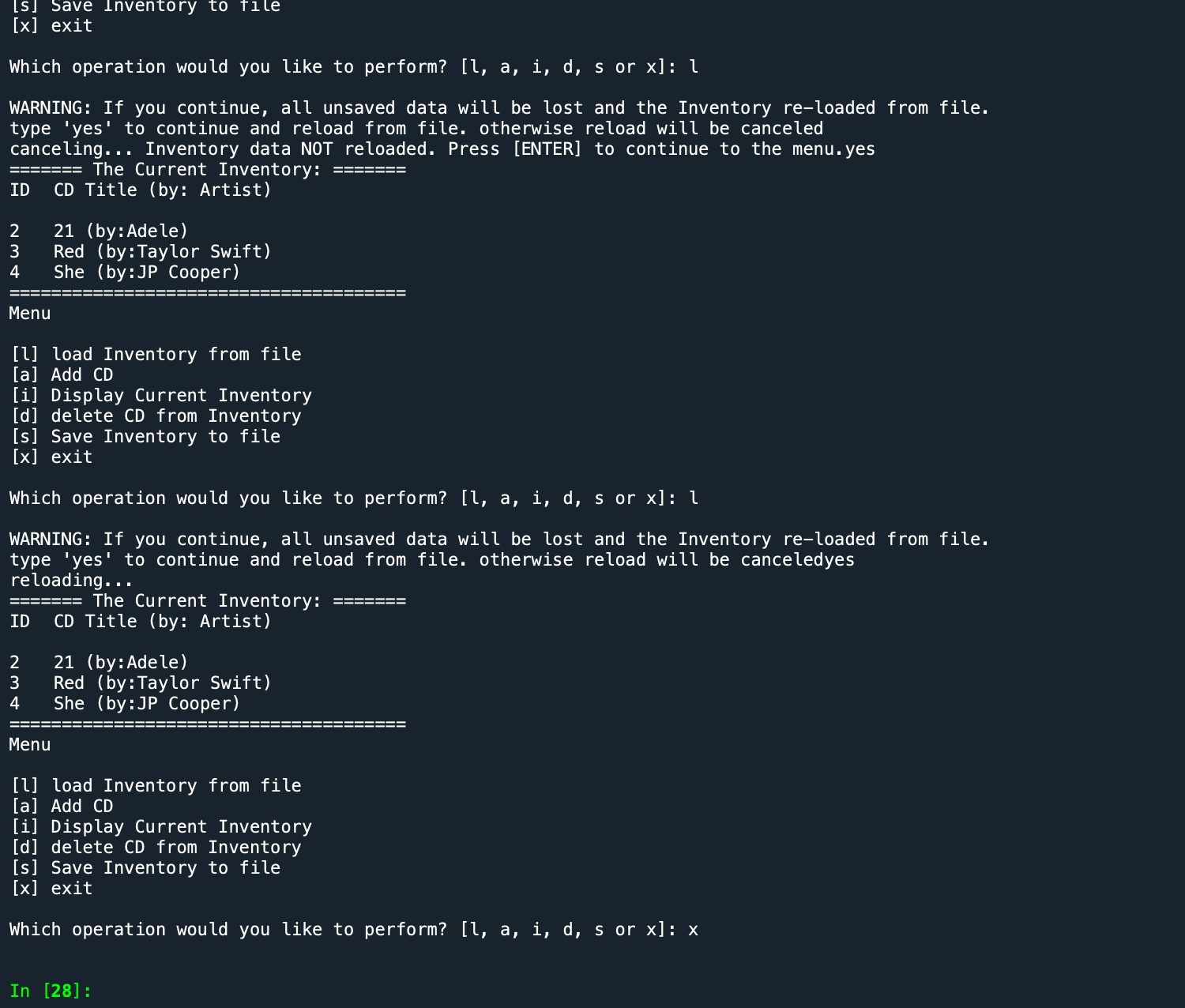


Otherwise:



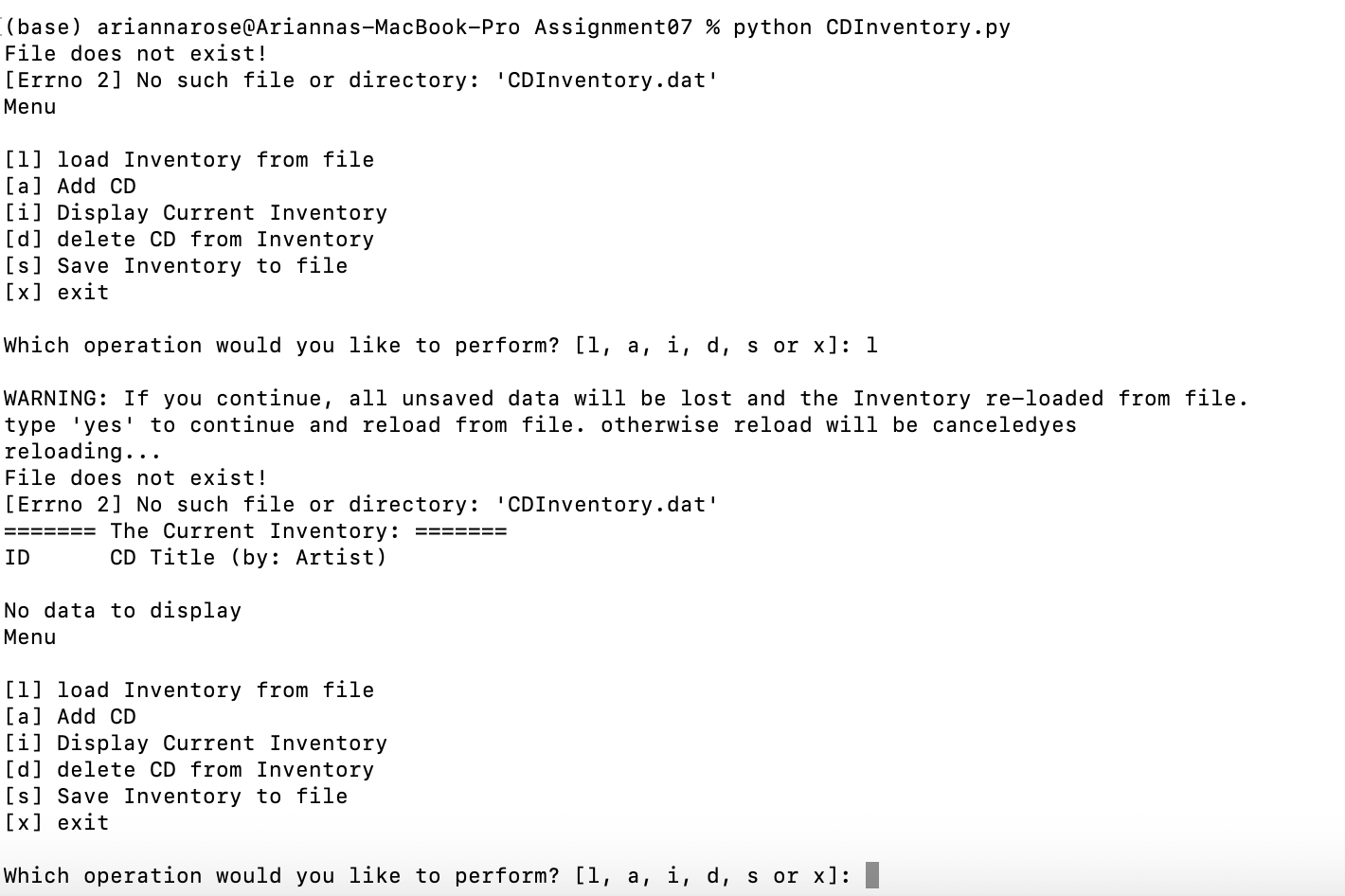






In Terminal:

When the file doesn’t exist:



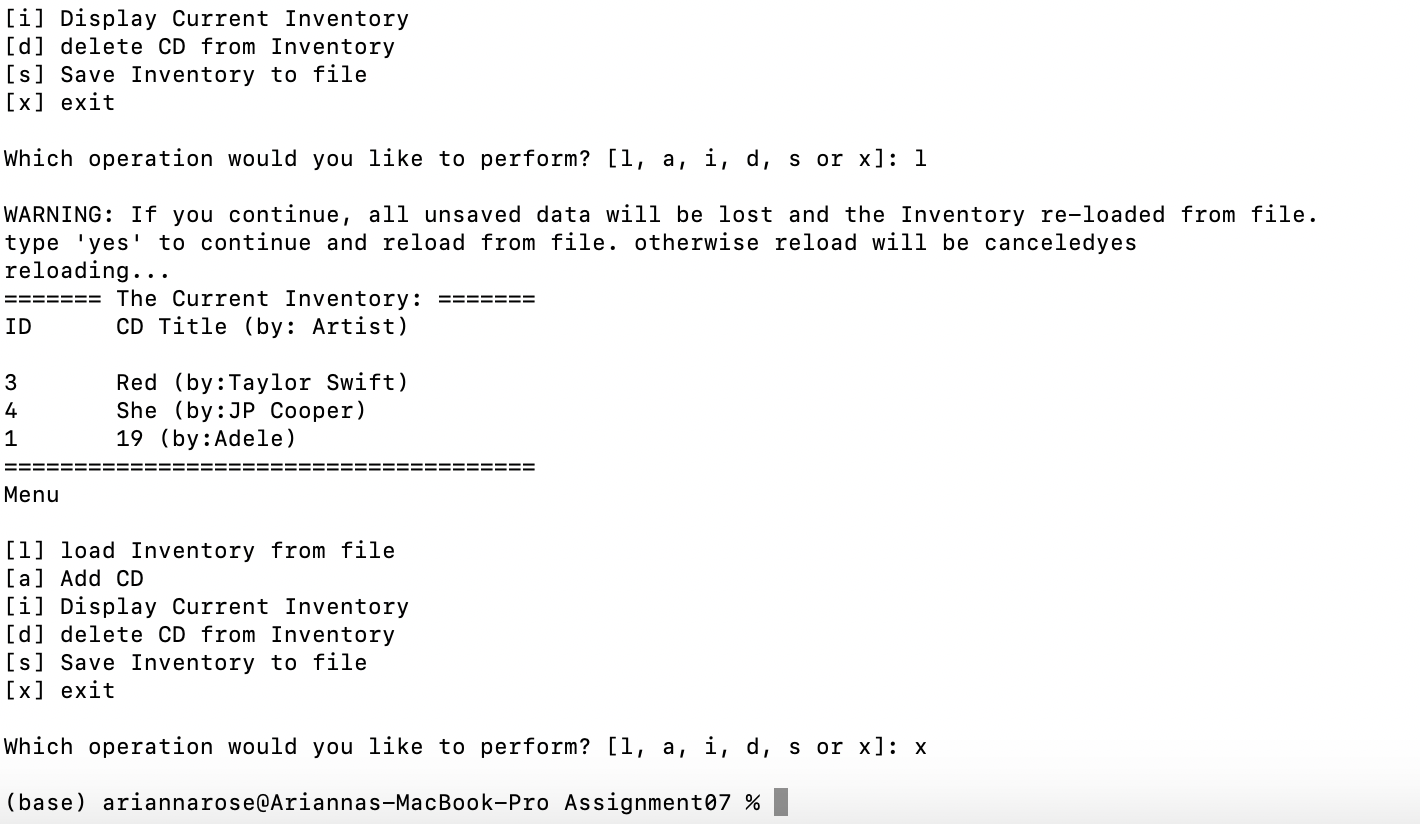
Otherwise:

A black and white document

Description automatically generated with medium confidence

Table

Description automatically generated



Research Findings:

Exception Handling:

1. <https://www.programiz.com/python-programming/exception-handling#:~:text=In%20Python%2C%20exceptions%20can%20be,we%20have%20caught%20the%20exception>.
   1. This resource was easy to follow with simple code examples illustrating the concepts
   2. Taught the use of finally, which was not covered in our module. You can use finally after try and except if there is something that needs to be done regardless of whether there was an exception.
2. <https://www.geeksforgeeks.org/python-exception-handling/>
   1. Also pretty easy to follow with relatively simple code examples. I preferred the formatting of the Programiz one, but this one is not bad.
   2. Added a clarification for the difference between syntax error and exceptions
   3. Also taught the use of finally

Pickling:

1. <https://www.geeksforgeeks.org/pickle-python-object-serialization/>
   1. This one was very detailed and had complex coding examples, so I was doing additional research to better understand the code that was used in the examples (concepts outside of just the pickling part being taught here)
   2. A good resource to come back to when I am working on something more complex with pickling
2. <https://pythonprogramming.net/python-pickle-module-save-objects-serialization/>
   1. A short and sweet one that gives a high-level explanation of picking and its advantages and provides a simple coding example

Summary

This was great practice for structured error handling – it gave us the opportunity to evaluate the code we wrote to see where errors that would terminate the program could occur and then add safeguards for those potential errors to avoid unintended program termination as much as possible. It was also exciting to see how much easier it was to save and load the data to/from a binary data file as opposed to a text file. It reduced the number of lines of code while still performing the same function within our program.