Maricha Friedman

August 22, 2021

IT FDN 110: Introduction to Programming (Python)

Assignment 7

CD Inventory Script with Exception Handling and Binary file storage

# Introduction

For this assignment, we had to once again use the CD inventory script, but updating it to use binary files for storing and loading the CD inventory data, as well as adding structured error or exception handling where there are user inputs.

# Online research

## Research on structured error handling

The [Python exception documentation](https://docs.python.org/3/library/exceptions.html)[[1]](#footnote-1) was a useful external resource for this assignment for several reasons. First, I made a mistake with my lab in assuming a non-integer input would be “TypeError” when it’s actually a “ValueError”. Seemed counterintuitive to me, but even though the examples used ValueError, reading the documentation made it clear. Even though this isn’t a third-party academic site, I think this counts as helpful resource.

This [website](https://www.python-course.eu/python3_exception_handling.php)[[2]](#footnote-2) was good because it used separate functions as examples – one for the structured error handling, and one for the example calculation (calculating a dog’s age). This is helpful because in Lab07\_C I wasn’t sure if I (or how) should be using functions for the structured error handling. This is partly because I’m still wrapping my head around how to use functions efficiently.

One thing that is still unclear to me is if I “need” loops to do structured error handling, or if they are just commonly used together because we need the user to keep trying until they input correctly (I used a try/exception block without a loop in this assignment, but I had some issues). In my Lab07\_C, I had trouble figuring out how to test for an integer and a non-zero integer separately, per the instructions. I did it by listing both try cases on separate lines in the try block, and then naming each exception case, which worked.

## Research on Pickling

Once again, the official [Python documentation](https://docs.python.org/3/library/pickle.html)[[3]](#footnote-3) for pickling was my first stop in my research. It was a little too technical for me, but it was helpful to see it compared to json. I have tangentially seen and heard about json conversions used in other settings, so this helped me conceptualize pickling (and saving to binary files) as just putting data into a file with its own formatting/additional data.

This [site](https://realpython.com/python-pickle-module/)[[4]](#footnote-4) was not particularly helpful, it basically gave me the same information as the Python documentation. Even after some research, I am not entirely sure what “serializing” means. Just saving “series” of data? Meaning a variable and its list of values?

This [website](https://www.geeksforgeeks.org/understanding-python-pickling-example/)[[5]](#footnote-5) gave a nice concise overview of pickling, with examples. Particularly helpful was this part:

“Pickling is a way to convert a python object (list, dict, etc.) into a character stream. The idea is that this character stream contains all the information necessary to reconstruct the object in another python script.”

Because it explained in nutshell how I was able to save (dump) and load data in the same format and data type without having to organize, cast, or format it. For example, my list was saved as a list and loaded as a list.

One thing each resource (and the class material) did make clear was that pickling is Python-specific. While I’m sure binary files are not Python-specific, the “pickling” program is. This means that while the .dat file format itself may be readable in other programs, the specific process by with Python “dumped” or saved the data in the binary file is specific to Python, and therefore can only be loaded correctly back into Python.

For the labs (Lab07\_B) this was helpful to just understand why we would do this, if the .dat file can’t be used outside of Python. It seems much easier to use pickle to store data if you’re just going to use it within your Python program, because you don’t have to bother with formatting strings to make the file readable in other programs. It’s not relevant for our small practice programs, but I can imagine pickling being very useful in complicated Python scripts that need to use huge amounts of data. It would make sense to store that data in a binary file to free up the memory.

# The CD inventory script

## Writing to a binary file

I had trouble writing and loading from binary. At first I thought it could be because for the assignment we are using a 2D list (list of dictionaries), whereas in the labs we just had a simple list. Turns out that wasn’t the case, I just needed to better understand how I can use a function to update my global variable “lstTbl”.

I had a temporary code to call the function of loading the inventory from a .txt file (manually copied over from the previous assignment). This worked, but my goal was to save it into a binary file and then remove the .txt file read and instead read the .dat file once I had created it. When I tested saving and re-loading from the binary file, I got and error: “EOFError: Ran out of input”, which, after Googling, seems to mean that the binary file is empty. I thought maybe this meant my “write\_file” (save) function didn’t work. To test, I exited the program after loading the inventory from .txt file and appending a new cd to the in-memory list. After exiting, I did “print(lstTbl)” to see if the inventory was indeed there. It was, so I had to figure out how to save it properly to the binary file.

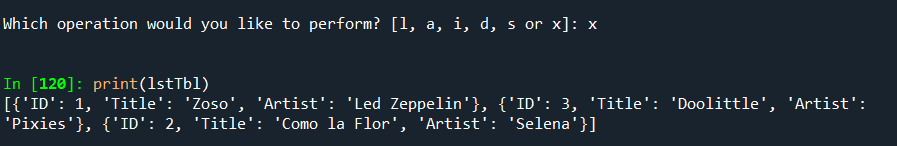
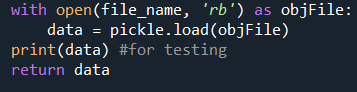


Figure 1 - testing my in-memory list

I realize this is essentially the same as my built-in function “IO.show\_inventory(lstTbl)” but I wanted to be sure where my error was. I checked my file folder, and the script did create a CDInventory.dat file, and it did have something in it, and I could decipher enough to see that it was my same updated list. Logically, problem then had to be in the load (unpickling) function.

I added a print statement in my “read\_file” function, to isolate where the mistakes were happening. After some trial and error and consulting the book and module text, I fixed my mistakes in the unpickling part so that I wasn’t getting the EOFError anymore, but the show\_inventory function kept coming up blank. However, the print statement did show that the pickle.load function worked, it just wasn’t being passed into “lstTbl” at all. I think this because I had some circular logic in my function whereby “table” was one of the arguments and also the result. This may have been either an issue of shadowing a variable, or my confusion in terms of reference v. value types. In any case, I removed “table” as an argument at all, including the table.clear() command, and simply called the resulting data “data”.



Listing 1 - Loading binary data

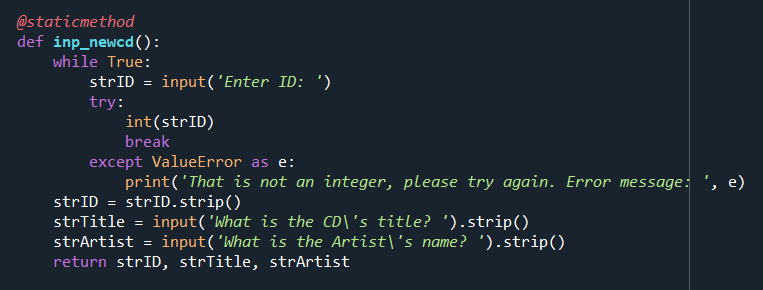
Then, on the line where I call the function in the main script, this is where I updated the value (reference?) of “lstTbl”:

lstTbl **=** FileProcessor**.**read\_file**(**strDatFileName**)**

Once I did that, the show\_inventory function worked great and “lstTbl” was updated.

## Structured Error Handling

I added structured error handling for the ID of the new CD information to be added, as well as for the ID selection of which CD to delete. In both cases, I build the exception handling code into the existing functions of IO**.**inp\_newcd**()**and IO**.**inp\_delete**()**as shown below:



Listing 2 - Error handling for ID # as integer

I also added exception handling for when we read the .dat file in the “load” menu choice option. This was a little more complicated. I created a new function called IO**.**error\_read\_file**(**file\_name**)** for the sole purpose of evaluating if there is a file not found error. It returns True if there is no error, and False if there is an error (this sounds backwards writing it this way, but I though of it as “True, the file is there” vs. “False, the file is not there”).

Then, in the main presentation code (Section 3.2 – Process load inventory), I created another level of “if” nesting that first evaluated if the file was there or not, calling the function I just created. If the result was true, then I proceeded with the existing code that wipes and re-loads the data from the file. If the result was false because the file wasn’t found, I re-printed the warning that the file was not there and no data would be loaded. The continue statement brings the user back to the main menu.

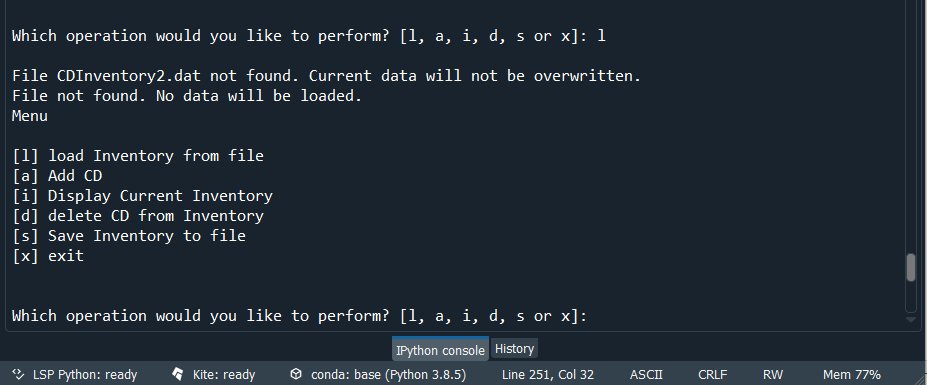
While this was a good exercise, it’s not really necessary because we define the file name in the beginning of the script, and ideally the user already saved a previous file with the same name, so it should be there. For testing, I purposely changed the “strDatFileName” definition at the top of the script to something else. That’s also why I did not add exception handling for the saving process since the ‘wb’ file opening mode creates the file if it doesn’t already exist.

# Summary

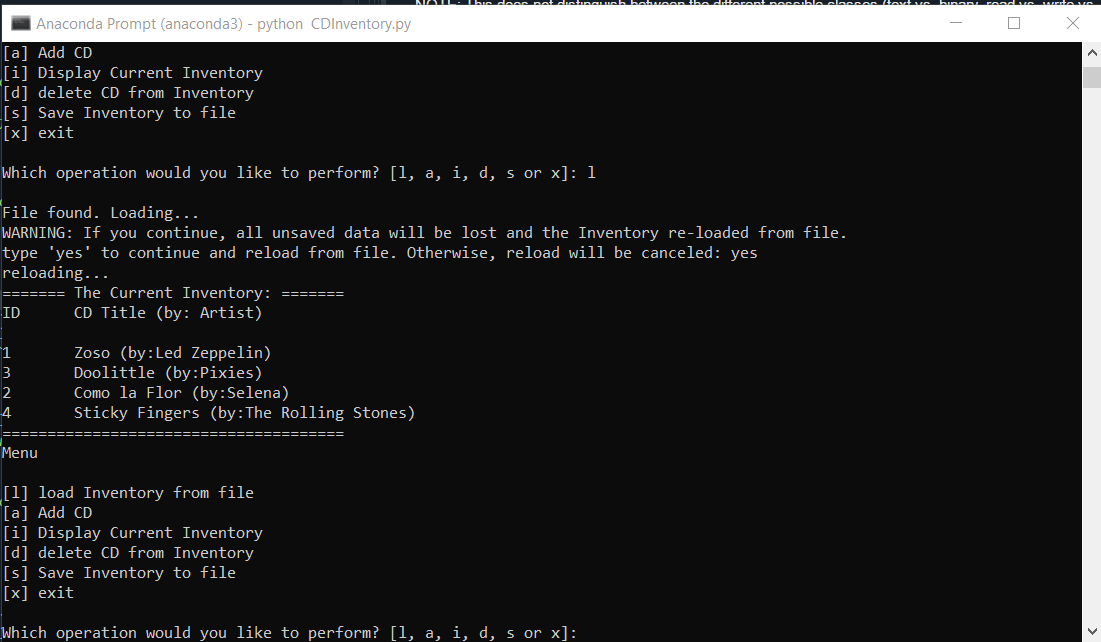
As with the previous assignments, there is always a fair amount of trial an error involved until getting the desired outcome. The external resources (and the StackOverflow forum questions Google inevitably takes me to when I am having a problem) are always helpful to see where I went wrong, or what to try next. Functions, pickling, and structured error handling have made this script cleaner and make more sense.

# Appendix

## Script running in Spyder



## Script running in Terminal



## GitHub Repository

Here is the link for my repository for Assignment 7:

<https://github.com/marichaf/Assignment_07>

1. <https://docs.python.org/3/library/exceptions.html> – Retrieved Aug 21, 2021 [↑](#footnote-ref-1)
2. <https://www.python-course.eu/python3_exception_handling.php> - Retrieved Aug 22, 2021 [↑](#footnote-ref-2)
3. <https://docs.python.org/3/library/pickle.html> - Retrieved Aug 22, 2021 [↑](#footnote-ref-3)
4. <https://realpython.com/python-pickle-module/> - Retrieved Aug 22, 2021 [↑](#footnote-ref-4)
5. <https://www.geeksforgeeks.org/understanding-python-pickling-example/> - Retrieved Aug 22, 2021 [↑](#footnote-ref-5)