Patrick Danielson

02/28/2021

Foundations of Programming (Python)

Module 07 – Assignment 07

Custom Error Handling

# Introduction

This document summarizes the work performed for Module 06 of the Foundations of Programming course. The purpose of this module was to familiarize myself with the use of class objects to organize functions with a Python program. Additionally, I became comfortable using docstrings as a means of documentation of user defined functions and practiced organizing my programs through applying the separation of concerns philosophy. The problem addressed in this assignment was to take a starter code for the CDInventory program with an already defined class structure and create additional functions that could be called by the main program to integrate the remaining program functionality as it pertained to input/output processing, data processing, and file processing.

# Exception Handling Research

There are a few good articles that exist to help visualize how Python programs handle the try and except blocks of code, however, I found the one from RealPython to be the most helpful in further explaining the use of raisin and calling exceptions.[[1]](#footnote-1) This article also provides a good explanation on why we should be structuring how we handle targeted or specific errors with a likelihood of occurring in our program. It states that, “Catching Exception hides all errors…even those which are completely unexpected. This is why you should avoid bare except clauses in your Python programs. Instead, you’ll want to refer to specific exception classes you want to catch and handlePickling Research” [[2]](#footnote-2)

# Picking Research

There is a good bit of detailed documentation of pickling on RealPython, however, I found that there were other sides that walked through more basic examples of pickling that was more applicable to this course. A GeeksforGeeks article has a good example of pickling and unpickling dictionary objects.[[3]](#footnote-3) Additionally, there is a good datacamp tutorial that summarizes object serialization and pickling process in Python.[[4]](#footnote-4)

# Using a Binary Data File

In order to modify the program from interacting with a plain text CDInventory.txt file to a Binary CDInventory.dat file, every section of the code that interacted with the file had to be updated. Since the program only interacted with the file within the functions of the FileProcessor class, it was simple to identify these as the only blocks of code that needed to be modified.

However, this is not as simple when the user first uses the program, as there is either (1) no CDInventory.dat file exists or (2) the file exists but is empty. The latter scenario is what I had to deal with, because I initially have my program set up to check if the file exists and to create one if none is found. This leaves an empty data file that when loaded using the pickle.load() function, returns an EOFError. This error is raised when the .load() function reaches the end of file condition without reading in data[[5]](#footnote-5). Since this would always occur when a blank file is created the first time the user executes the program, I chose to capture the error in an except block and pass along a statement to the user that the data file initially loaded in by the program was empty.

Now that I could handle the empty binary file, I modified the write\_file() function such that it could save the inventory table stored in runtime memory to the file. This was simply accomplished using the pickle.dump() function.

After the data was stored to the \*.dat file, I realized I did not have an easy way to check in what manner the data was stored and could be read in. To better visualize this when reading in data, I temporarily added a print statement that would show me what the prickle.load() function reads in from the binary file. I tested this by adding two entries and saving the data before trying to load it back in. As shown in Figure 1, the pickle.load() function would read in the entries as a list of dictionary objects for each entry. Therefore, to properly assign them to the inventory table stored in the runtime memory, I used a for loop to append each item of the list read in from the pickled file.

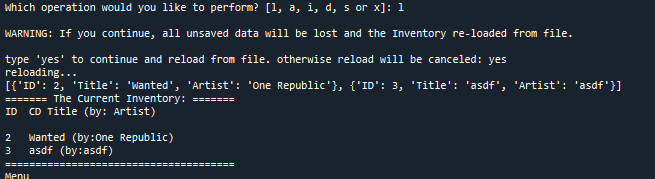


Figure 1: Test Output of pickle.load() Function (shown in Red)

# Error Handling

As previously mentioned, the first case of error handling was to address the EOFError generated by reading in an empty binary file. This, along with other errors, was handled using the try-except structured code blocks. The intended operation or execution of the program is captured in the try block, and any expected errors that it could encounter are handled within the except blocks to prevent the program from simply crashing.

I chose to handle errors by printing back the error type to the user along with a custom message. Within the FileProcessor class, in addition to the EOFError in reading a file, the program could also encounter a FileNotFoundError if the user accidentally deletes the file from the directory while the program is still running. This would require the user to exit the program and restart to create the file, and while the data is lost in the delete file, the user is made aware of what error occurred. This error is handled in both the read\_file() and write\_file() functions.

In the DataProcessor class, I wanted to check that the operation within the add\_entry() function that converts the user entered ID to an integer can handle if the user does not enter an integer value. This was done using the ValueError handling of the except block and would pass back a message warning the user their entry was invalid. Since this was the first piece of data being processed in the try block, if the error was raised, no data would be written to the table as Python jumps out before that line is executed.

A similar error could also occur within the main loop when the user is prompted to enter the ID of the entry they wished to delete from the inventory. The entire section 3.5.1.2 of the program is therefore contained within the try block such that if the user did not enter a valid integer, the program would jump out of the block and display a custom message for handling the ValueError, prompting them to enter an integer value.

# Executing the Program

The execution of the program is summarized in this section. The program is first executed in Spyder where there is no CDInventory.dat file in the directory and one must be created. I then attempted to add an entry with an invalid ID number before correctly adding two entries. This execution is shown in Figure 2 and Figure 3. Figure 4 shows saving the data.

The next step was to delete and entry and reload the data from the binary file. This was done by first entering an invalid ID number for deletion, the correct integer ID, and then reloading to display the previous inventory. This execution is shown in Figure 5.

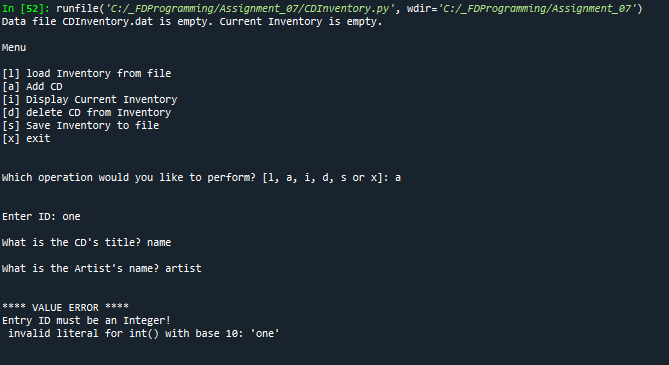


Figure 2: Spyder Execution for Invalid ID Entry

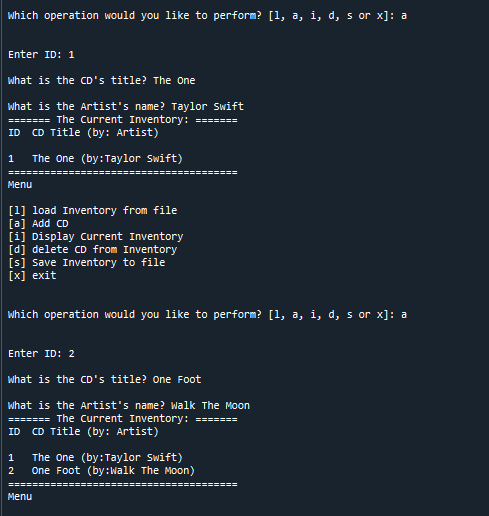


Figure 3: Spyder Execution for Adding Entries

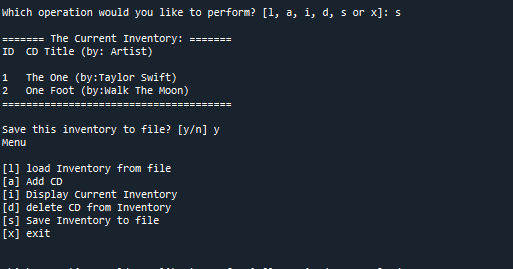


Figure 4: Spyder Execution Saving Data

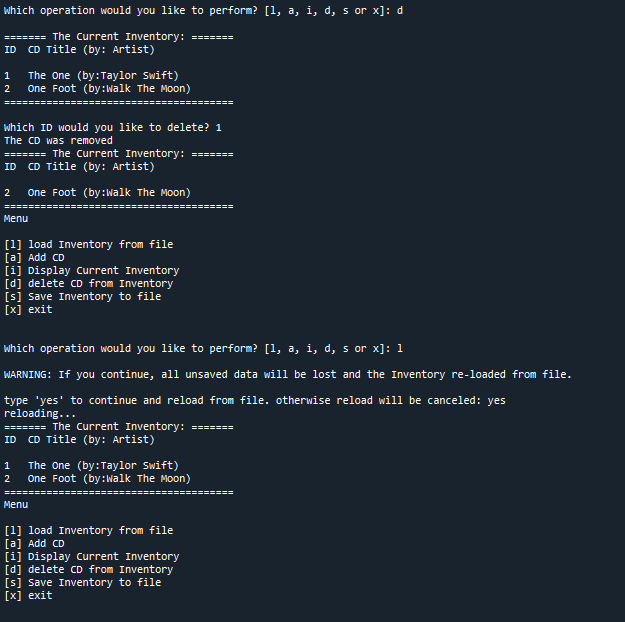


Figure 5: Spyder Execution for Deleting Entries and Loading Data

The terminal execution of the program is shown in Figure 6 for displaying the inventory read in from Spyder execution and deleting an entry. The files submitted for this assignment do not include the final saved CDInventory.dat file.

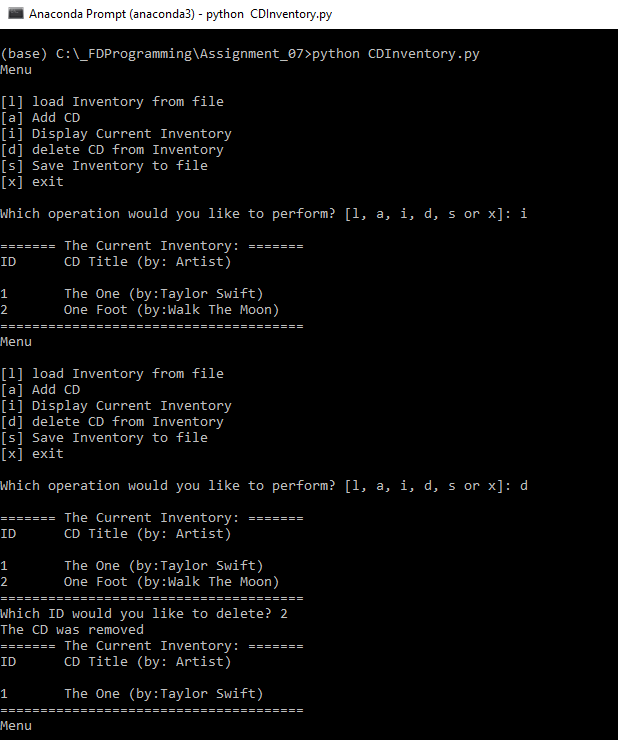


Figure 6: Terminal Execution Reading in Data and Deleting Entry

# Summary

In this assignment, I was able to successfully modify my Assignment 06 program such that the program operated on Binary files rather than plain text files. I also was able to implement structured error handling of all anticipated errors within my program. By doing so, I have learned to work with binary files and unpacking data stored in them, as well as think more intentionally about the way in which my program can be broken. This allowed me to anticipate the errors and handle them by displaying more meaningful messages to the user rather than the program crashing. The Python script and CDInventory.txt file used in this assignment, along with this document, are posted on GIT[[6]](#footnote-6) under the Assignment\_07 repository

1. <https://realpython.com/python-exceptions/> [↑](#footnote-ref-1)
2. <https://realpython.com/python-exceptions/> [↑](#footnote-ref-2)
3. <https://www.geeksforgeeks.org/understanding-python-pickling-example/> [↑](#footnote-ref-3)
4. <https://www.datacamp.com/community/tutorials/pickle-python-tutorial> [↑](#footnote-ref-4)
5. <https://www.geeksforgeeks.org/handling-eoferror-exception-in-python/> [↑](#footnote-ref-5)
6. Patrick Danielson GIT Repositories: <https://github.com/pdaniel441?tab=repositories> [↑](#footnote-ref-6)