Dan Oranski

3/09/20

IT FDN 100 B

Assignment 07

Observations of Assignment 07

# Introduction

This document covers the work through of Assignment 07. In this assignment, students applied the concepts of binary file reading and writing, and custom error handling via try and except. The concepts were unified by modifying Assignment 06 to work with binary files and incorporating error handling for all user inputs.

# Working with Text Files

There are several different ways to manipulate text files using Python. As it applies to reading text files, a convenient method to use is the readline() function. When the readline() function is called, the next line of data is returned. As long as the file stays open, this position is memorized and calling the readline() function will advance to the next line. This concept is referred to as a cursor in programming. When the file is closed, the cursor is reset to the first line.

For reading multiple lines, the readline() function can be used conveniently with a while loop to fetch multiple lines of data. The readlines() function can also be called to read all of the lines of a file into a list. The read() function returns a string.

LAB 07-A directed students to modify LAB 06-C by adding in functions that read from and wrote to a text file. Below is a screenshot of the results.

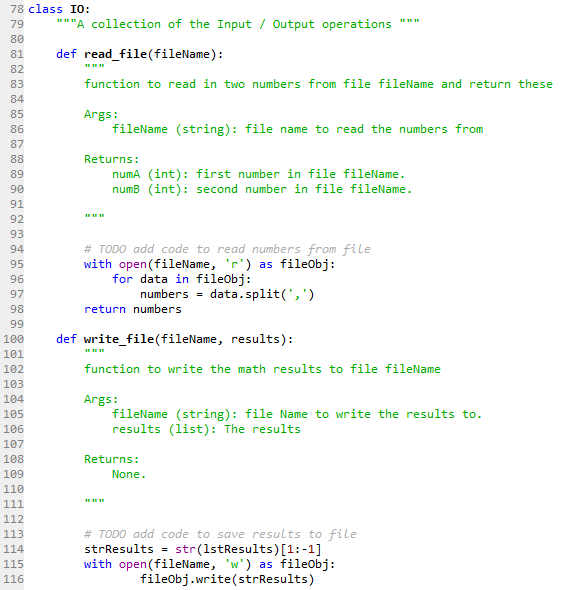


Figure - Read/Write functions

`

# Working with Binary Files

Text files are convenient because they are human readable. However, it is time consuming to format text files, and text files occupy more memory than binary files. This makes storing data in binary convenient. Reading from and writing to binary files requires additional inputs when performing the associated operation. The normal ‘r’ and ‘w’ arguments are appended with ‘b’: ‘rb’, wb’.

Saving data to a binary format in Python is called pickling. The pickle module takes the information associated with an object and serializes it in a way that can be easily stored or loaded as binary information.

LAB 07-B directed students to modify LAB 07-A by: creating two distinct program arguments (‘calc’ and ‘IO’), reading from and writing to file in binary, and adding the required functionality for the calc and IO program arguments. The results of this are shown below.

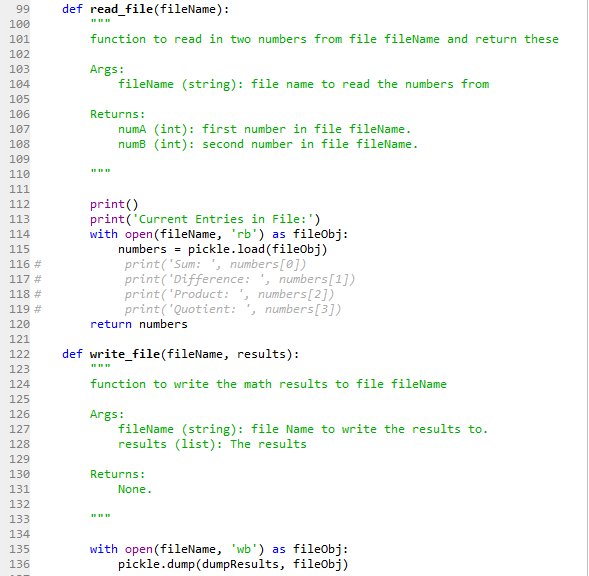


Figure - Binary compatibility

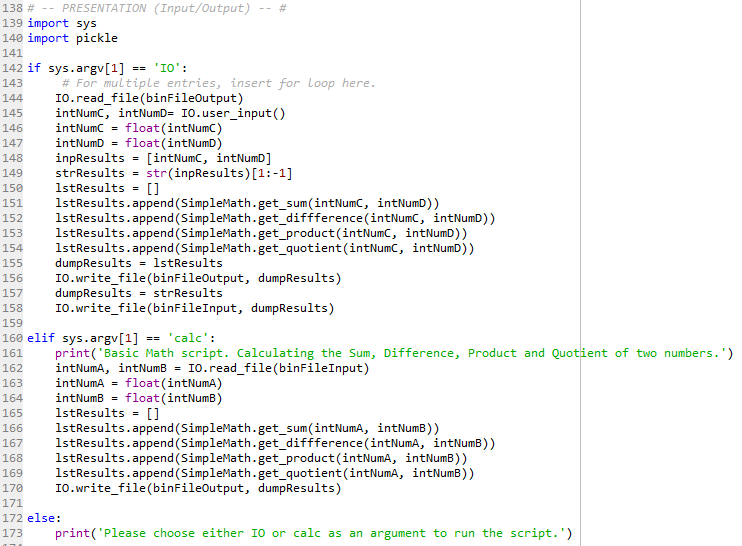


Figure - System path arguments

# Structured Error Handling

Rather than having Python crash each time an error is encountered, it is more appropriate to handle the error such that the program does not crash. Structured error handling allows the programmer to insert procedures that prevent crashing the program in the event that an error has occurred. Some examples of embedded error handling:

* Zero Division Error
* ValueError
* FileNotFound

Error handling should be used anytime there is a foreseeable possibility that something could go wrong. Some examples of when to utilize error handling:

* Interaction with humans
* Interaction with files
* Function calls

Structured error handling can be accomplished by “trapping” the statements likely to encounter problematic situations into try-except blocks. It is also possible to have custom error messages that provide more specificity to the error encountered than what is provided natively in Python. Handling errors with structured error handling and messages provides a better user experience, prevents the program from crashing, and doesn’t clear everything in memory in the process.

Python uses a class “Exception” to hold information about an error. Occurring errors can create an object contained in this class, and the object is filled with information about the error that caused the exception.

Programmer can create custom exception classes. These classes are derived from a base class and the message output comes from the Exceptions docstring. The data and functions can be overwritten and extended/added to in the derived class to customize the derived class.

LAB 07-C directed students to modify LAB 07-B to include structured error handling for various user inputs. The results are shown below.

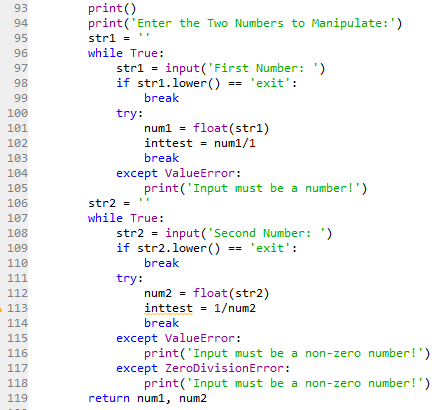


Figure - Structured error handling

# Application of Learning Objectives

Tying everything together, the student was asked to modify the Assignment 06 Python script by utilizing binary files for storing the inventory and adding in structured error handling for the various user inputs. The script was created, troubleshot, and run in the Spyder IDE.

This task proved to be more challenging than initially expected. Functions make integrating data processing a bit harder than processing the data in the program execution stage.

The results of Assignment 07 are shown below.

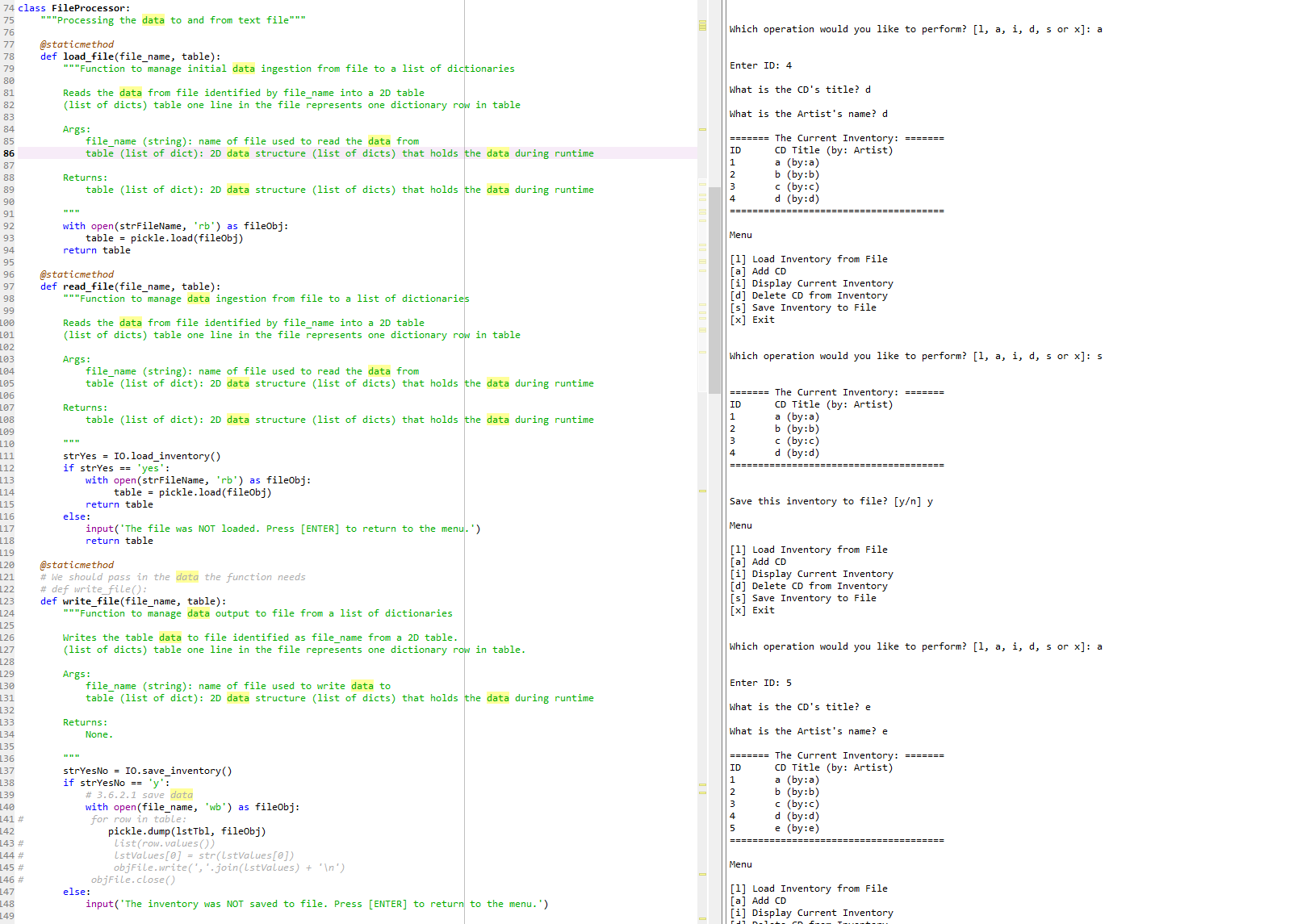


Figure - Assignment07-1

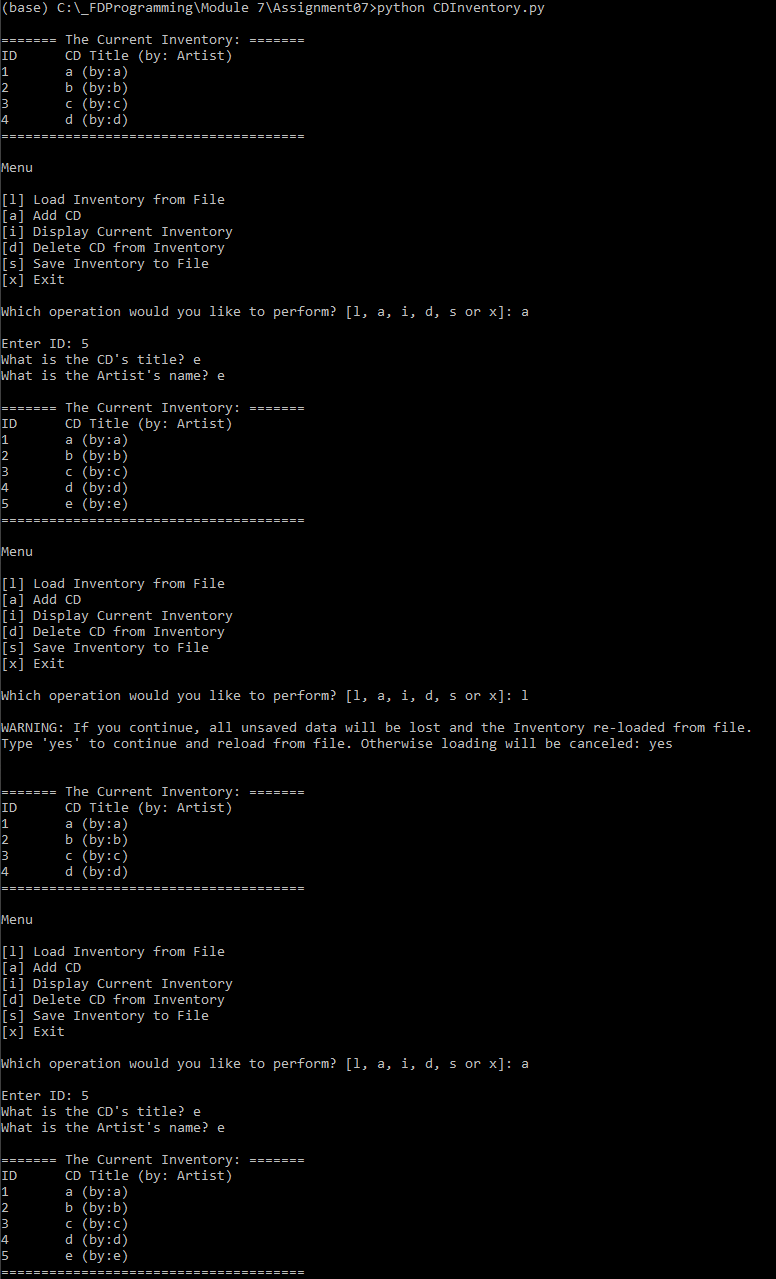


Figure - Terminal run

The files for this Assignment were uploaded to GitHub [here](https://github.com/angryeng/Assignment_07).

# Summary

Assignment 07 was far more challenging than Assignment 06. Working with functions, structured error handling and binary files adds layers of complexity not previously encountered. Working with binary files will be very useful in a professional setting and structured error handling is just plain necessary when writing programs.