## Katrina Hauser

## 29 August 2021

## BIDD Fundamentals of Programming (Python)

## Assignment 08

# Assignment 08: Object Oriented Programing (OOP).

## Introduction

In assignment 08, the focus was on object oriented programming and setting up objects in Python. To understand this functionality, several videos and references were reviewed.

## Topic 1: Python Features Review

The following Python modules were reviewed in addition to reading the second chapter of the course textbook: <https://saravji.github.io/saravjis_hut/FDN_Prog/Modules.html> - Module 8 only.

The following websites were reviewed and videos watched:

<https://docs.python.org/3/tutorial/errors.html>

<https://docs.python.org/3/library/exceptions.html>

<https://www.geeksforgeeks.org/python-exception-handling/>

<https://docs.python.org/3/library/pickle.html>

1. The goal of this module is to understand several questions:
2. What are the benefits of using structured error handling?
3. What are the differences between a text file and a binary file?
4. How is the Exception class used?
5. How do you “derive” a new class from the Exception class?
6. When might you create a class derived from the Exception class?
7. What is the Markdown language?

## Topic 2: Python Script Development

To demonstrate knowledge learned from the modules and references above, the previous version of the CD Inventory program that uses an inner data structure list of dictionaries was modified. In the modified version, structured error handling and pickling (import/export of data as binary files) was added. The focus of this assignment was to augment Assignment06 to add structured error handing where applicable and serialization of the data file read/write.

From the previous script, a menu option is presented to the user to control the CD inventory. The script loops through the options and executes the commands selected by the user. If the user enters an incorrect value, the script will catch this and present an error message. The user will be asked to enter a correct value. The basic looping and structure of the previous script was maintained.

The code is restructured into classes – DataProcessor, FileProcessor and IO - with the following functions – add\_cd, delete\_cd, read\_file, write\_file, print\_menu, menu\_choice, show\_inventory and get\_cd\_info. The focus of the last assignment was the organize the script into classes and functions to implement the design principle for separating a computer program into distinct sections such that each section addresses a separate concern. In this assignment, several functions involving reading and writing file data were modified to utilize serialization in binary file instead of read/write of text previously implemented. Error handling to manage file not found, user input types and validation have been added.

Walking through the script, in the first section the data is imported using the pickle library to serialize the data. Notice that the .dat file replaces the .txt file. The DataProcessor class with the add\_cd and delete\_cd functions focus is on processing the data in the internal data structure that is a list of dicts. The user is provided functionality to delete cds from this list (stored as a dict) and add cds to the list. Structured error handling added to catch type error if user enters value other than integer. Functionality is implemented to add to empty list and save that list to create new CDInventory.dat file. This allows a user to create a new list and save it, if one does not exist.

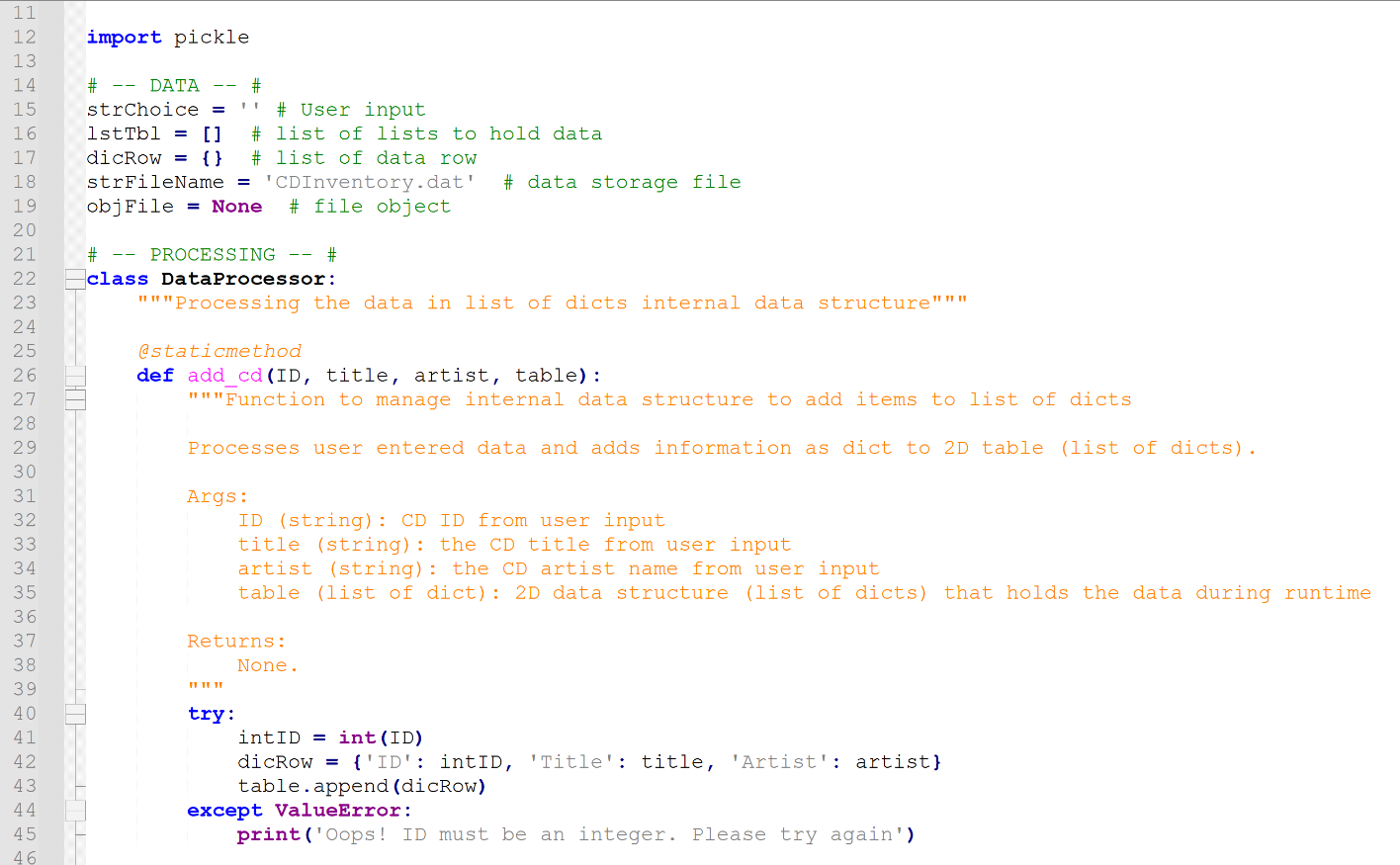


Figure 1: Import pickle, read from .dat and DataProcessor class add\_cd function error handling when user enters CD data into internal data list of dicts structure.

Exception handling manages addition and deletion from an empty table. This can happen if the file does not load and the user tries to add or delete a cd. Since this wasn’t not part of the original requirements, the feature has not been built in and the error is managed by exception.

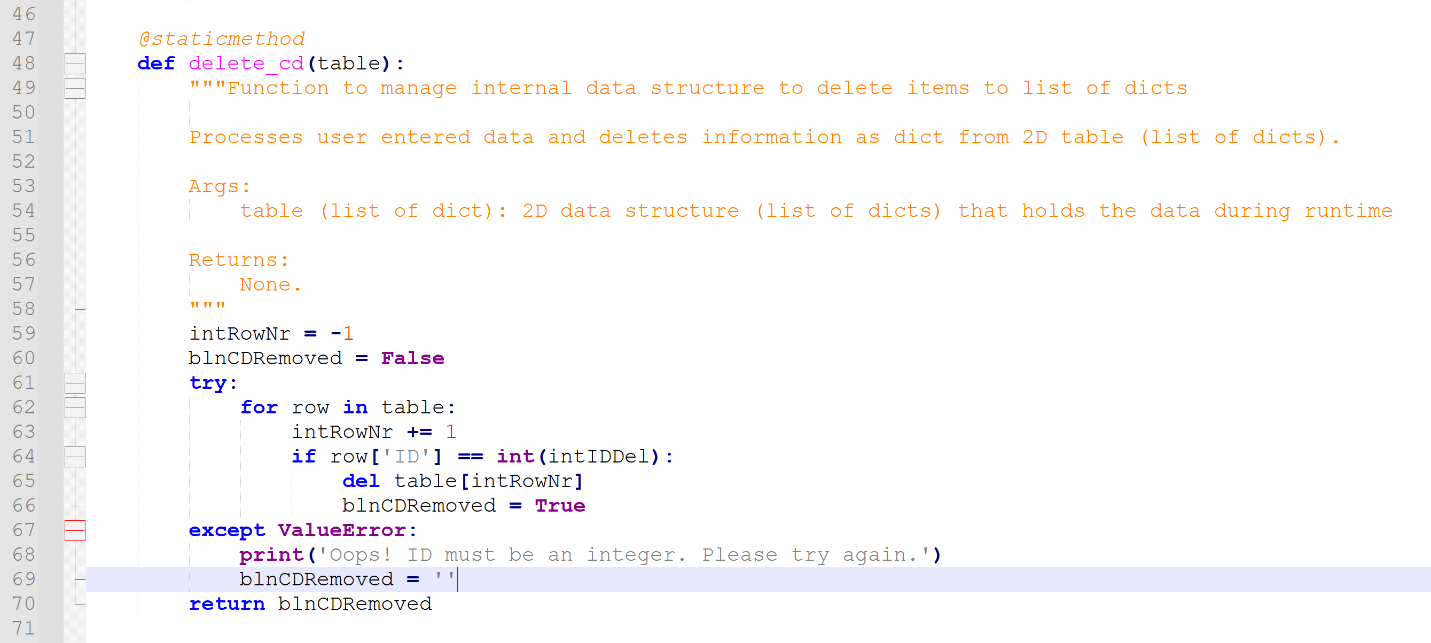


Figure 2: DataProcessor class delete\_cd function removes CD data from internal data list of dicts structure based on ID entered by user and errors are handled with try/except code.

The FileProcessor class with the read\_file and write\_file functions focus is on reading and writing to the CDInventory.txt file. Code modified to serialize the data using the pickle library. To handle errors, file not found exception implemented. Additional error handling added to catch errors when the file doesn’t load initially. If the user then tries to load the file was the script starts, the error will be caught.

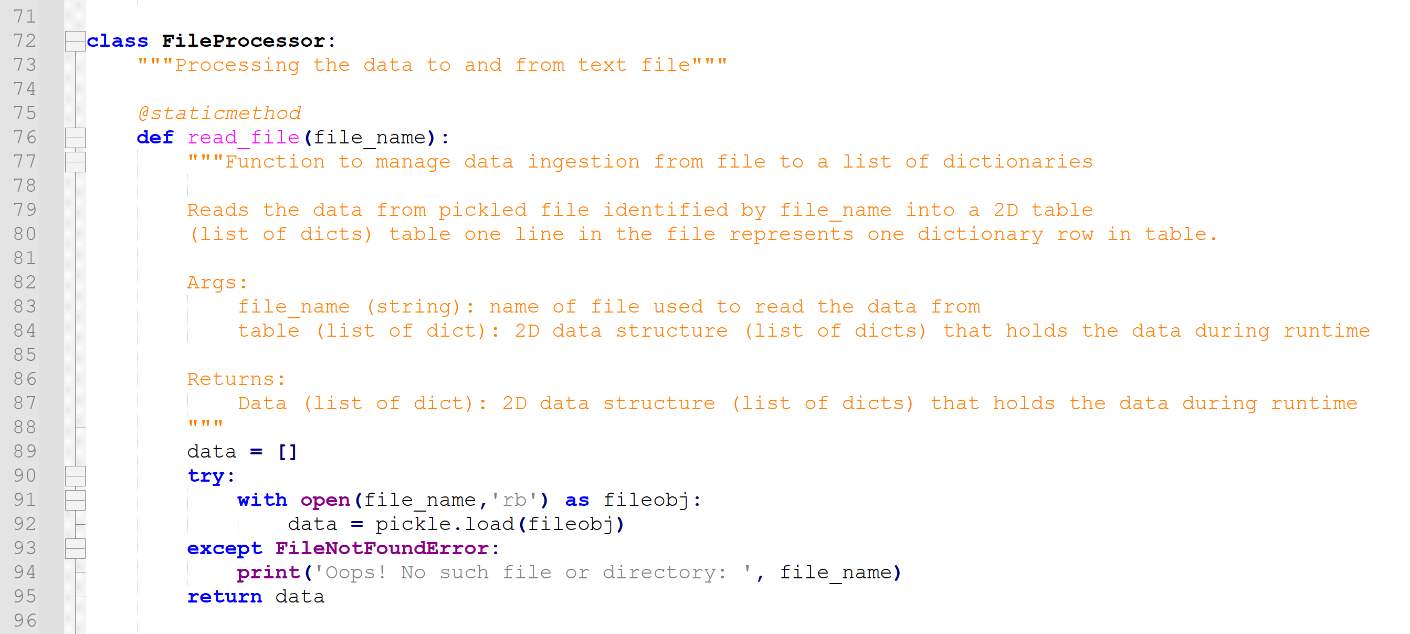


Figure 3: FileProcessor class read\_file function loads data from filename passed to function into internal list of dicts data structure.

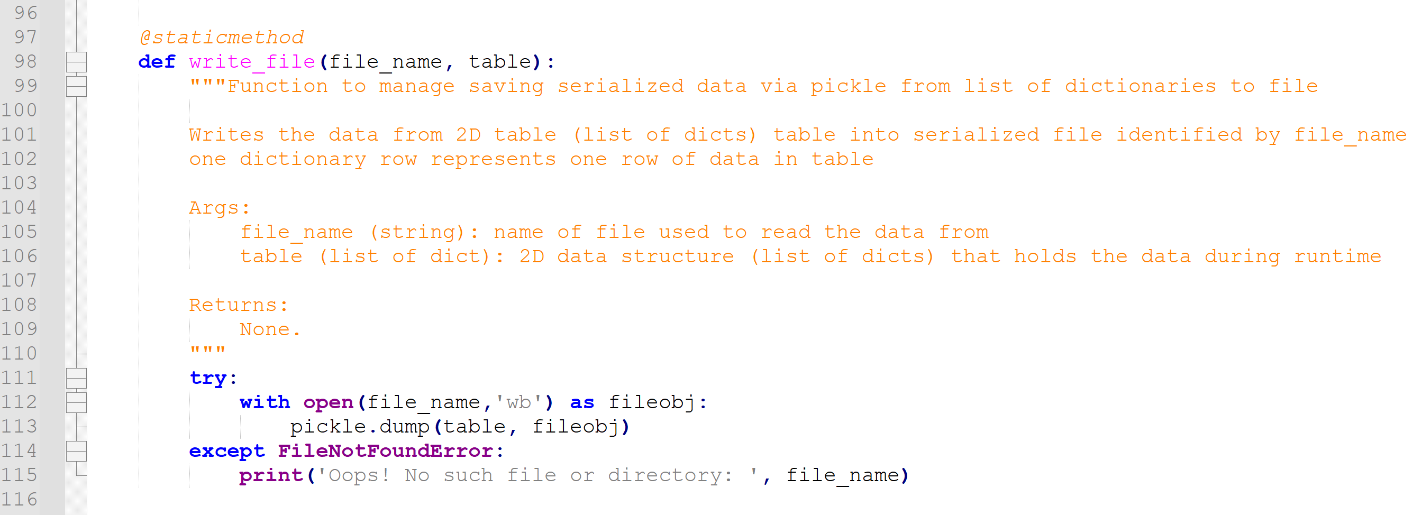


Figure 4: FileProcessor class write\_file function writes data to filename passed to function from internal list of dicts data structure. Error handling to caught file not found issues in addition to other load issue that can arise if the file fails to initially load.

The IO class with print\_menu, menu\_choice, show\_inventory and get\_cd\_info functions focus is on input and output to the user. Error handling added to catch when list is empty and the user chooses to display list.

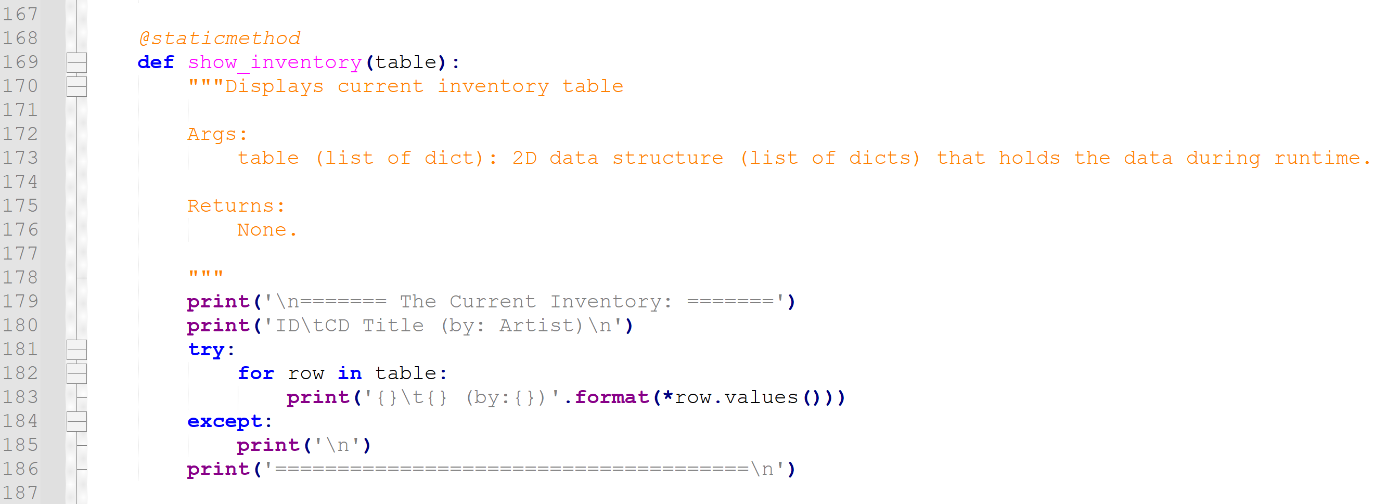


Figure 5: IO class show\_inventory function displays inventory and handles error if list is empty.

To start, variables, lists and dictionaries are initialized. Error handing is managed in the functions. The is serialized into binary files and read in using pickle.

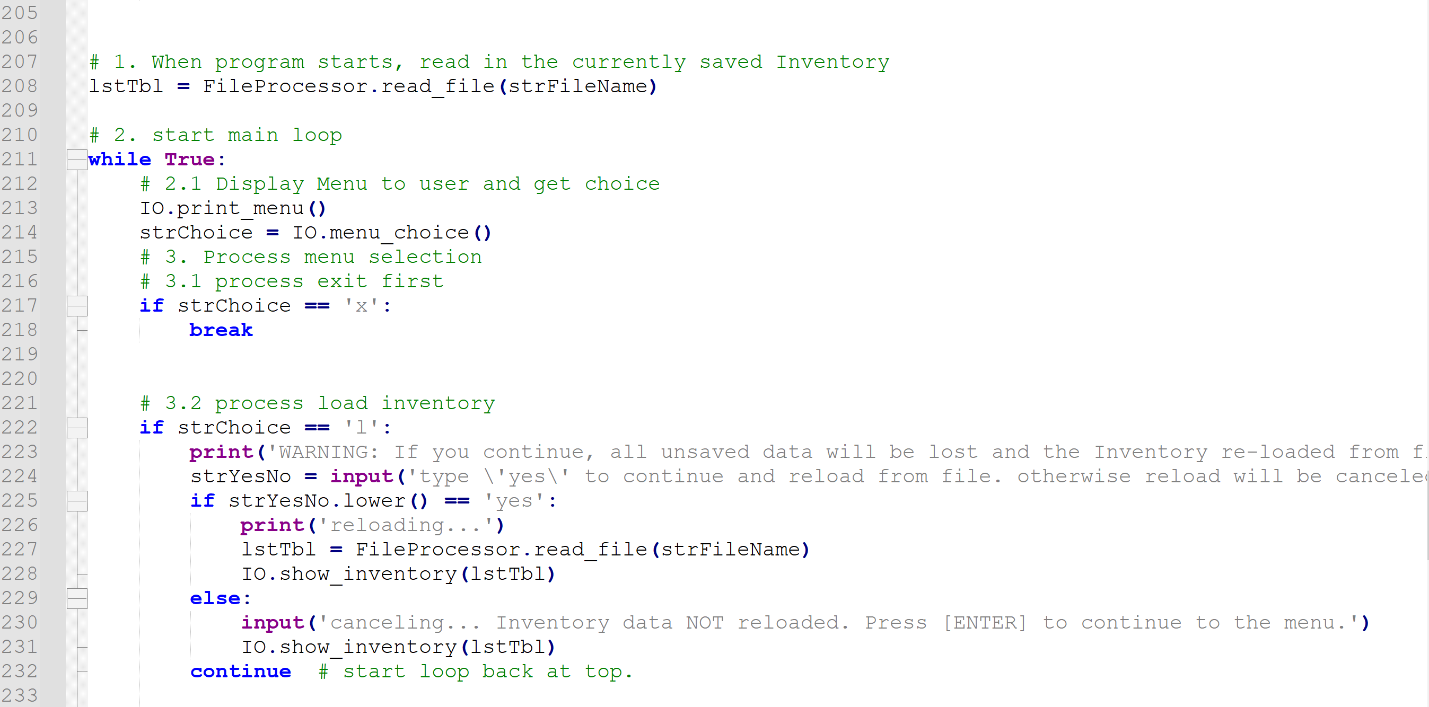


Figure 6: Start loop and read data from serialized binary file. File is read and loaded into list of dicts (lstTbl).

## Summary

In assignment 07, structured error handling and binary data files were reviewed. To demonstrate knowledge of the basics from this assignment, the CDInventory Python script from Assignment 6 was modified to incorporate structured error handling and reading/writing binary files to replace text read/write. The script functionality allows users to load data from a file, add data, delete data, view data and save the data back to a file.

## Appendix

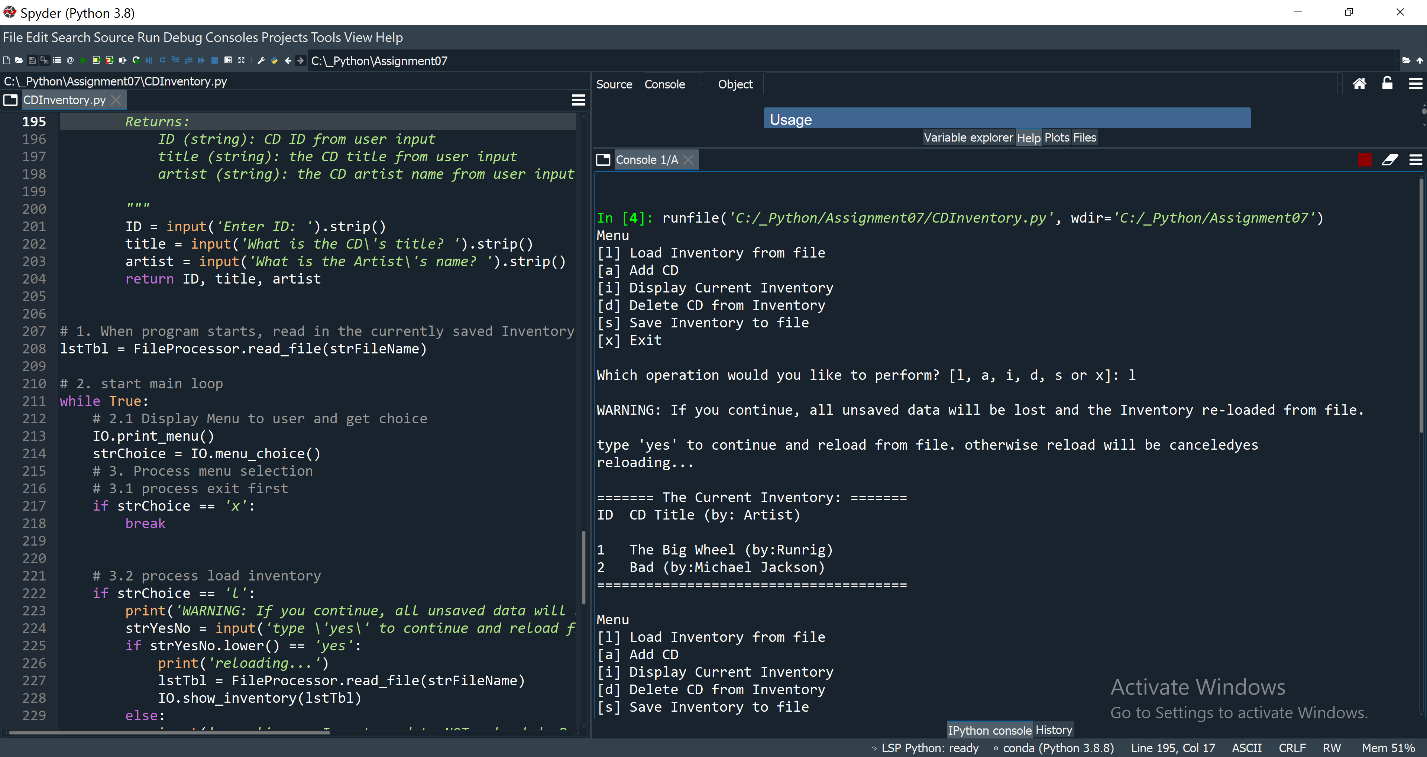
Assignment07 running in Spyder (Python 3.8):

Figure 7: Script CDInventory.py running in Spyder processing load (‘l’) and add (‘a’) commands from user. Program loads from CDInventory.txt file. Adds ID, Title and Album name from user entered data.

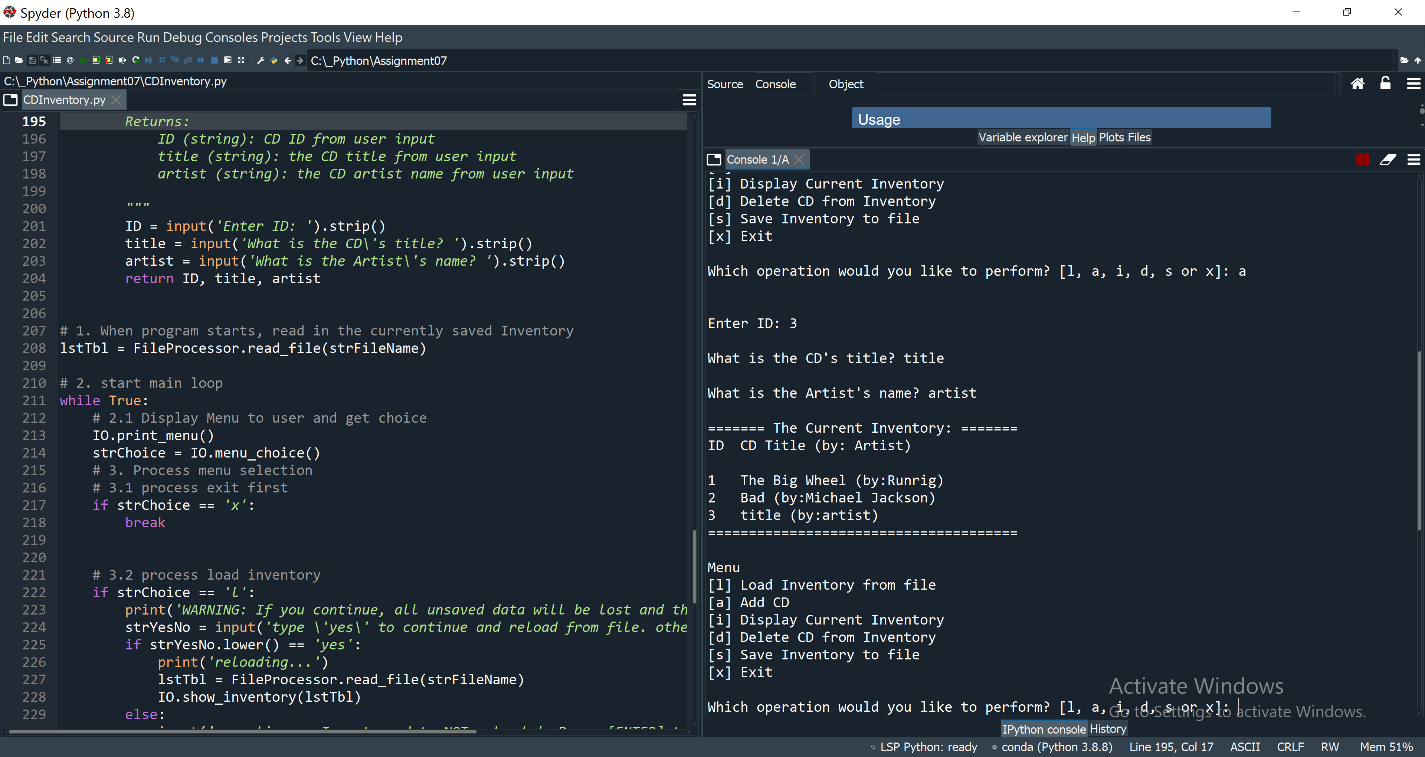


Figure 8: Script CDInventory.py running in Spyder processing display (‘i’) command from user. Program displays data that is stored in inner data structure list of dicts.

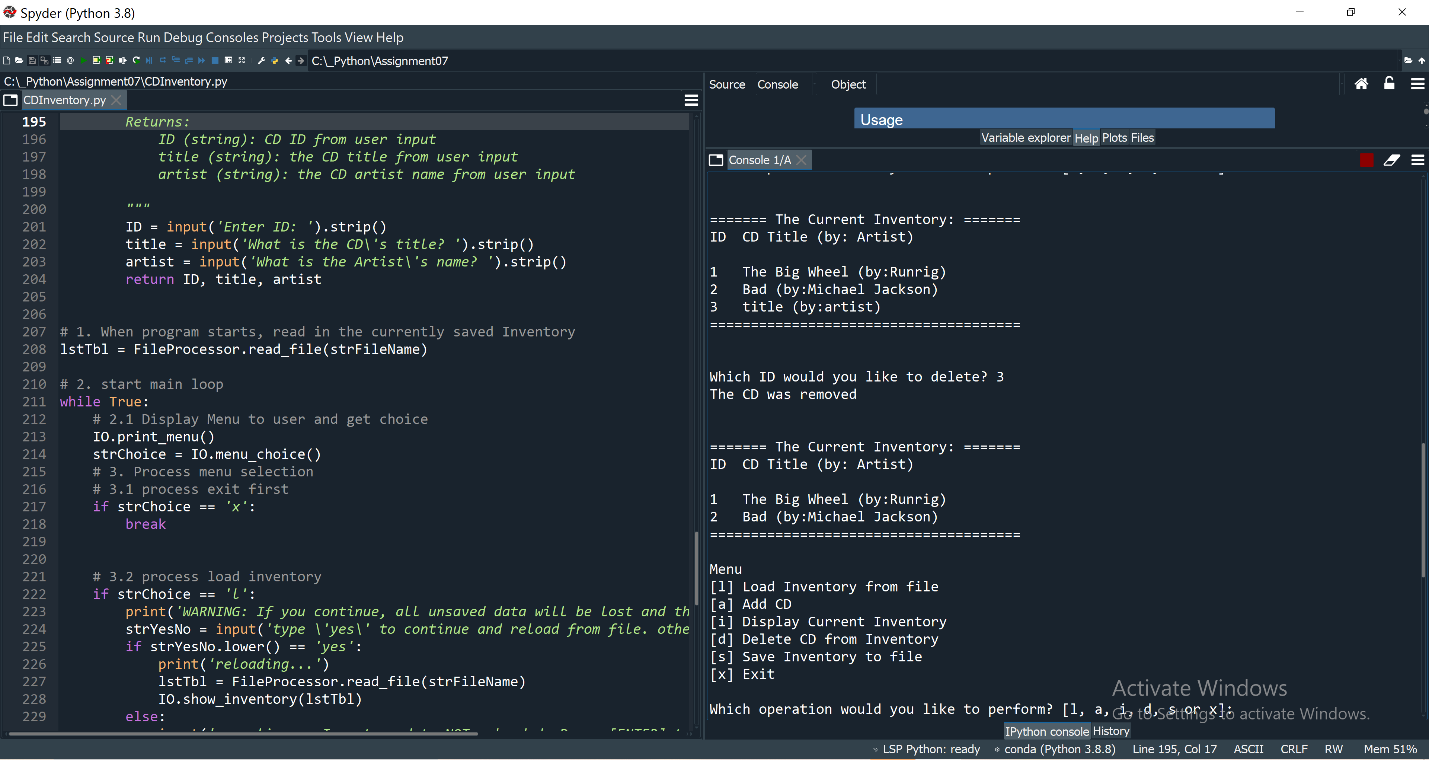


Figure 9: Script CDInventory.py running in Spyder processing delete (‘d’) and display (‘i’) commands from user. Program deletes dict from inner data structure by ID entered by user. Display command showing row deleted.

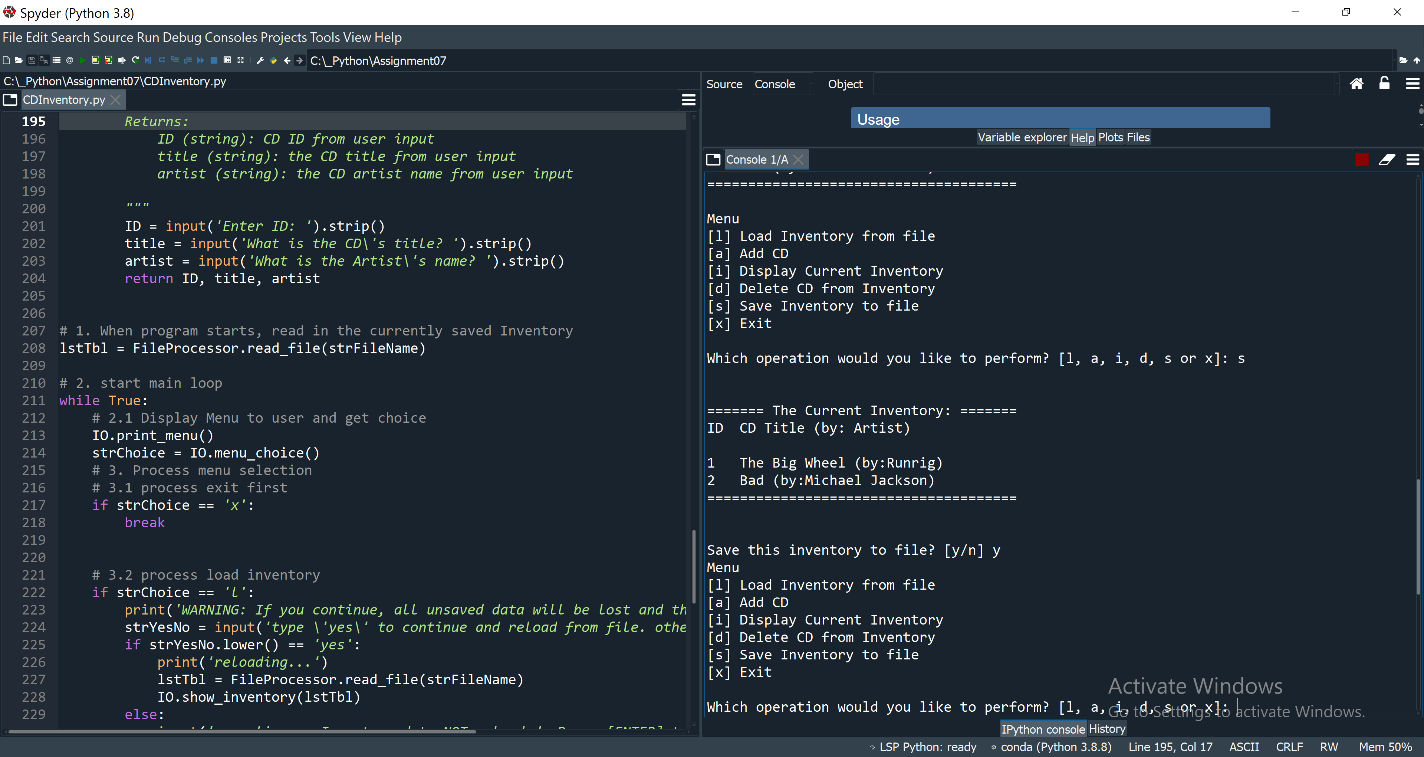


Figure 10: Script CDInventory.py running in Spyder processing save (‘s’) command from user. Program saves data from inner data structure list of dicts to CDInventory.txt file.

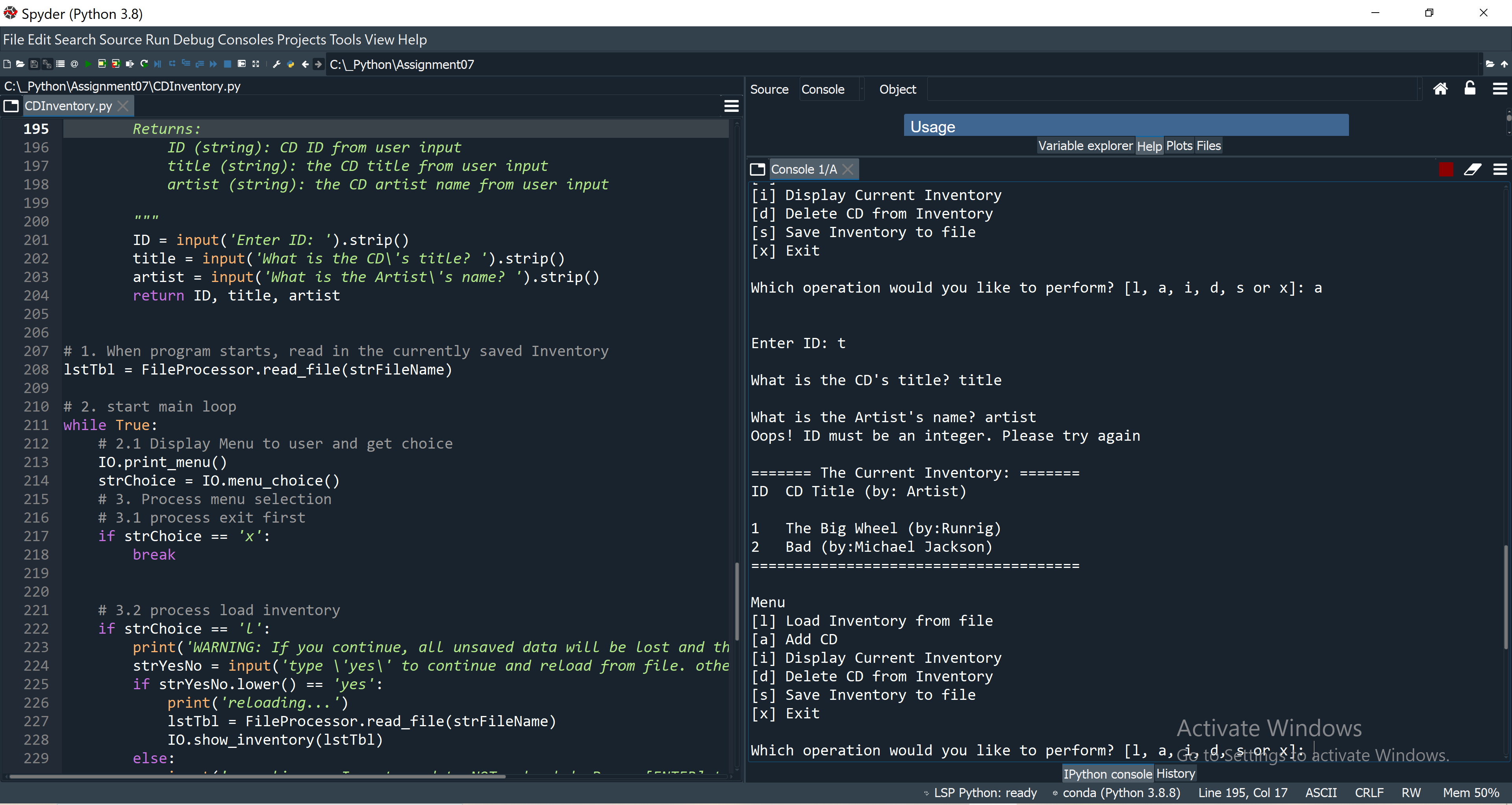


Figure 11: Script CDInventory.py running in Spyder error handling adding ID as text. Integer is required.

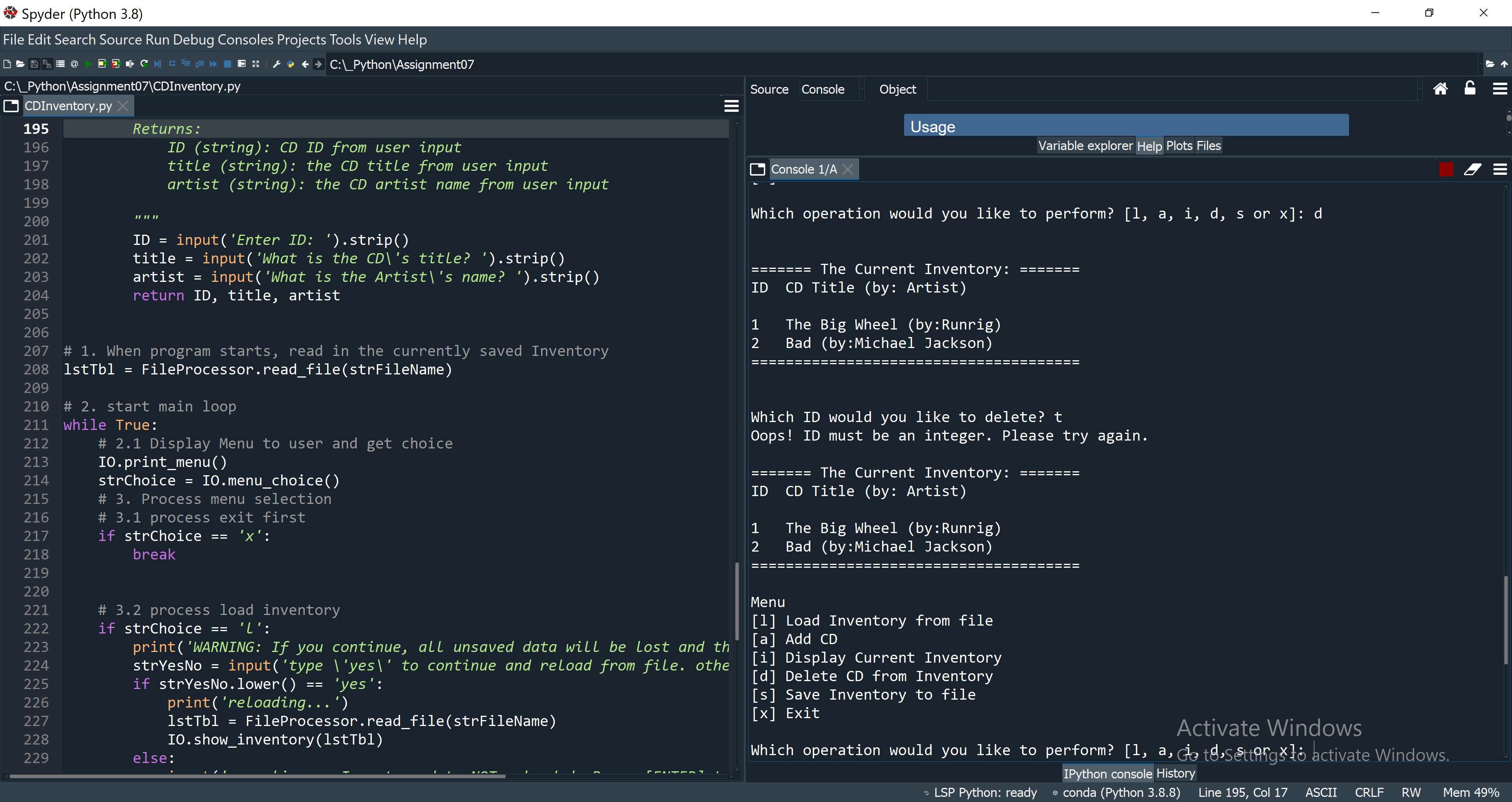


Figure 12: Script CDInventory.py running in Spyder error handling deleting ID as text. Integer is required.

https://github.com/hauserk/Assignment\_07

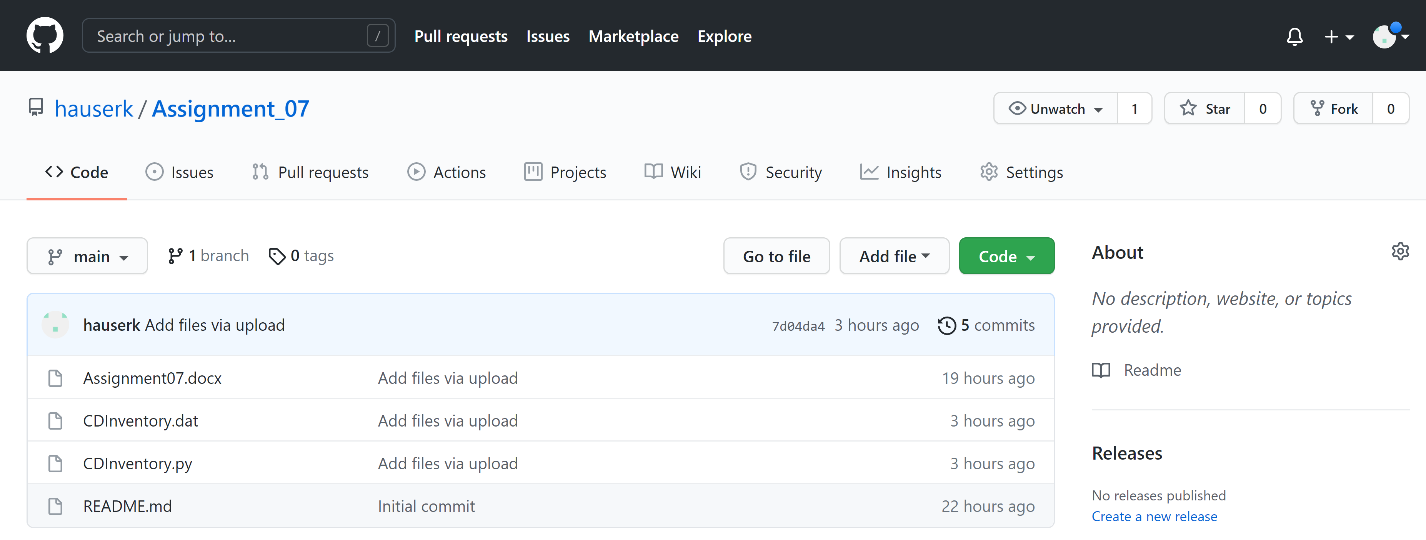


Figure 13: GitHub repository of Assignment07