## Katrina Hauser

## 22 August 2021

## BIDD Fundamentals of Programming (Python)

## Assignment 07

# Assignment 07: Structured Error Handing and Pickling.

## Introduction

In assignment 07, the focus was on structured error handling and pickling 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 7 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 functionality of error handling and pickling (import/export of data as binary files) was developed. The focus of this assignment is to augment Assignment06 to add structured error handing and picking.

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 new script adds enhanced error handling and functionality to import/export data via pickle libary.

In the previous assignment, the code was 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 script in the last assignment was organized 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 binary data instead of text as before.

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.

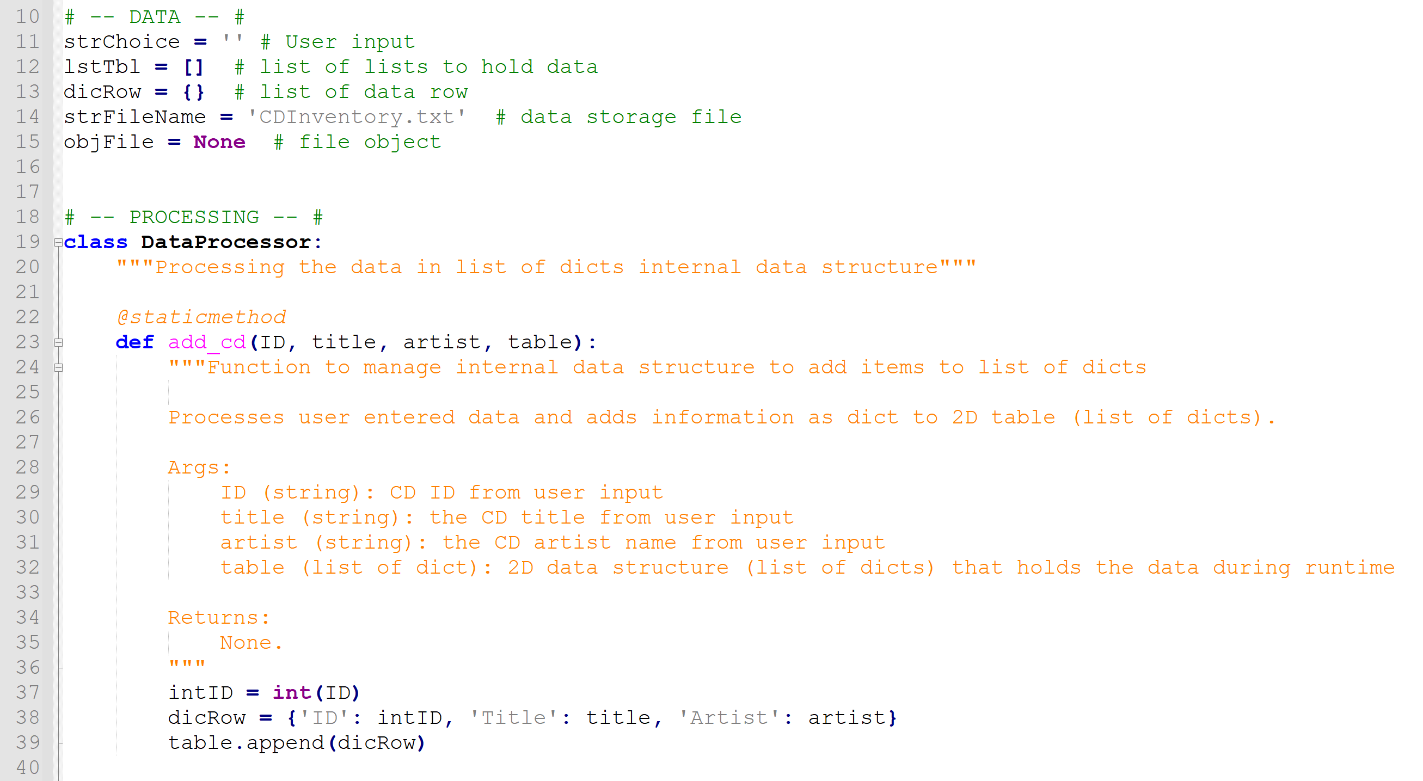


Figure 1: DataProcessor class add\_cd function appends user entered CD data into internal data list of dicts structure.

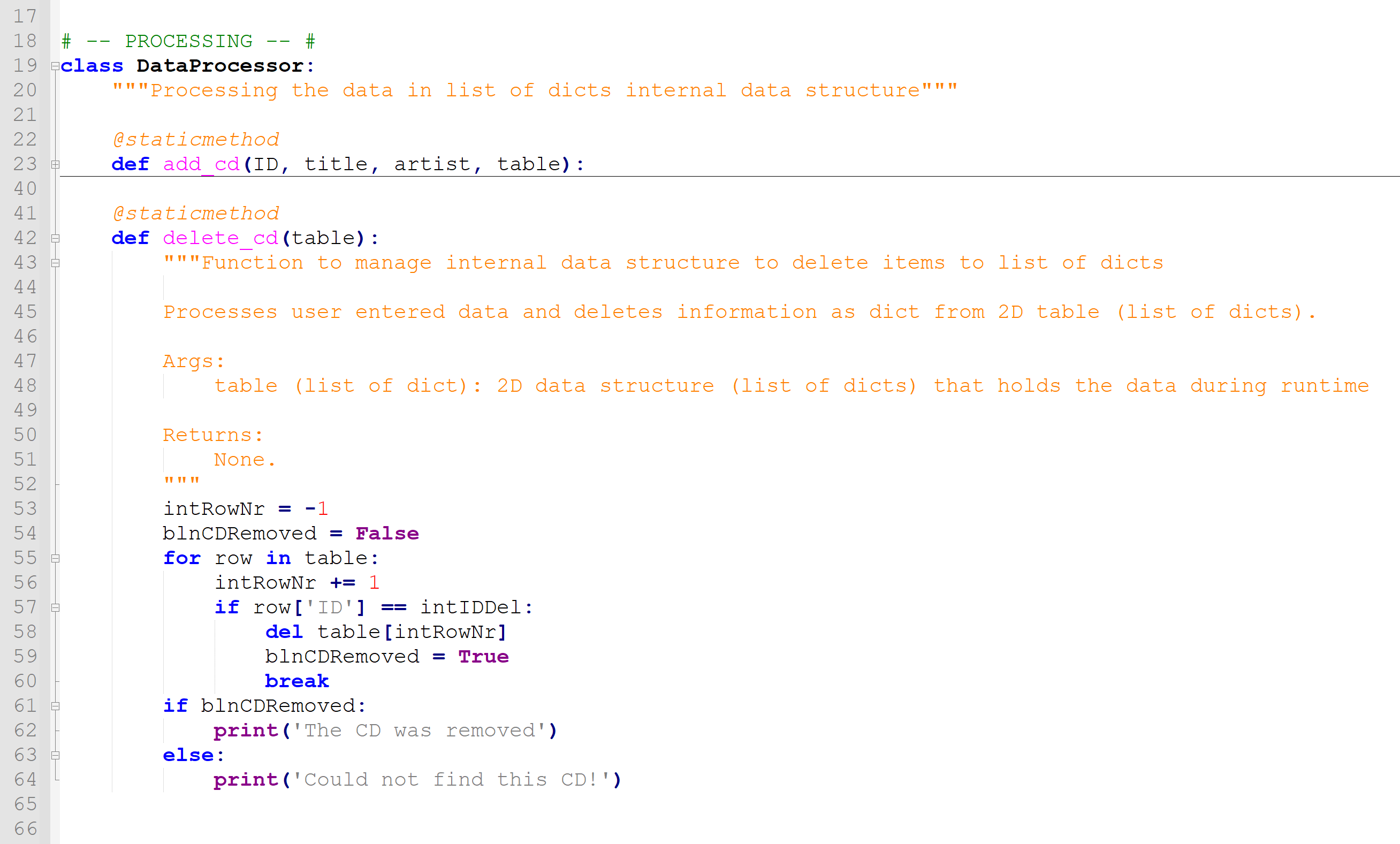


Figure 2: DataProcessor class delete\_cd function removes CD data from internal data list of dicts structure based on ID entered by user.

The FileProcessor class with the read\_file and write\_file functions focus is on reading and writing to the CDInventory.txt file.

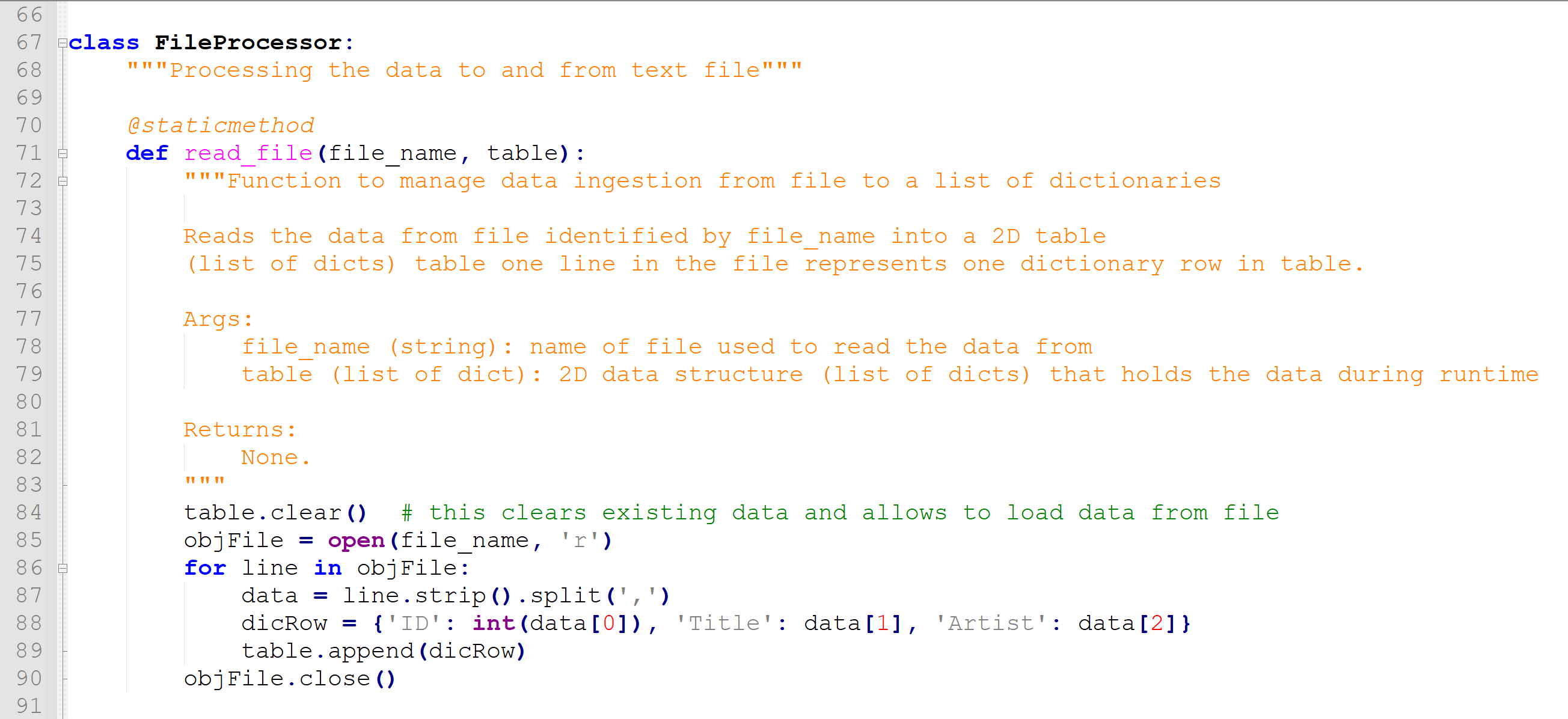


Figure 3: FileProcessor class read\_file function loads data from filename passed to function into internal list of dicts data structure.

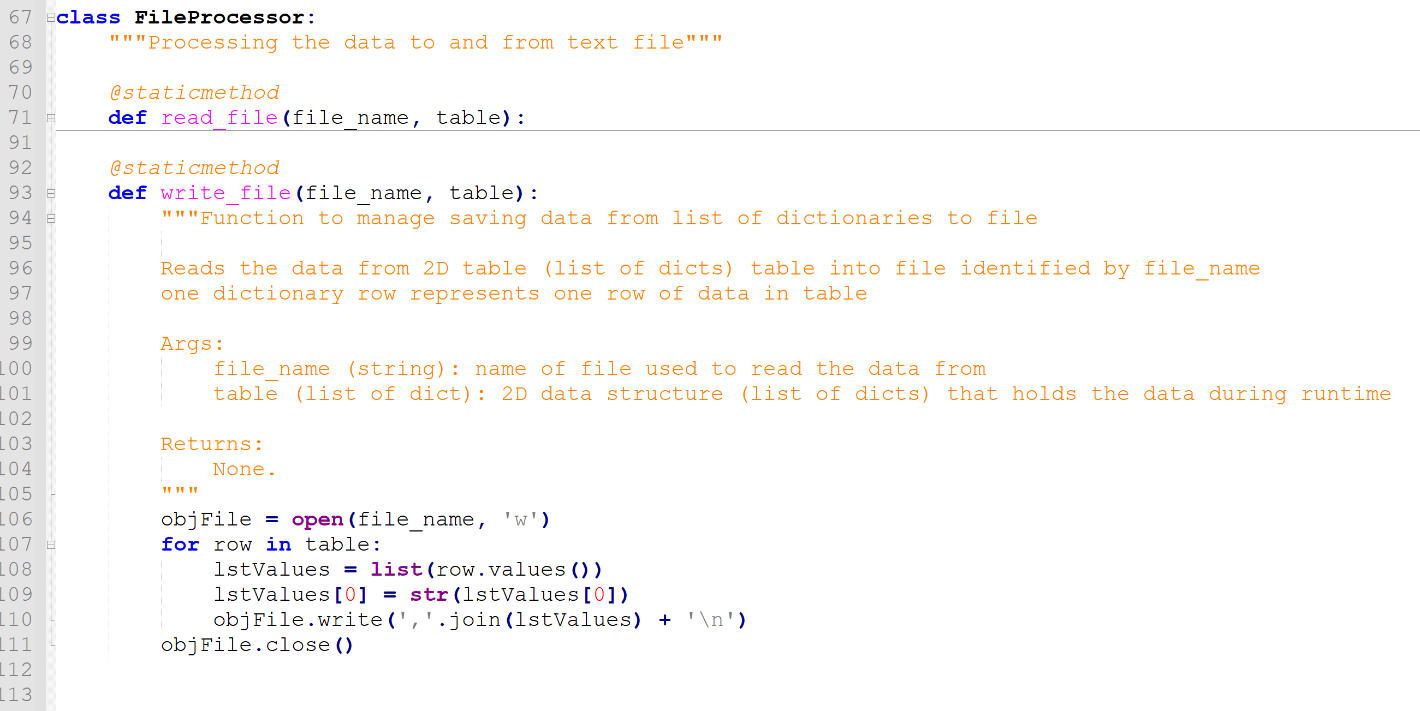


Figure 4: FileProcessor class write\_file function writes data to filename passed to function from internal list of dicts data structure.

The IO class with print\_menu, menu\_choice, show\_inventory and get\_cd\_info functions focus is on input and output to the user.

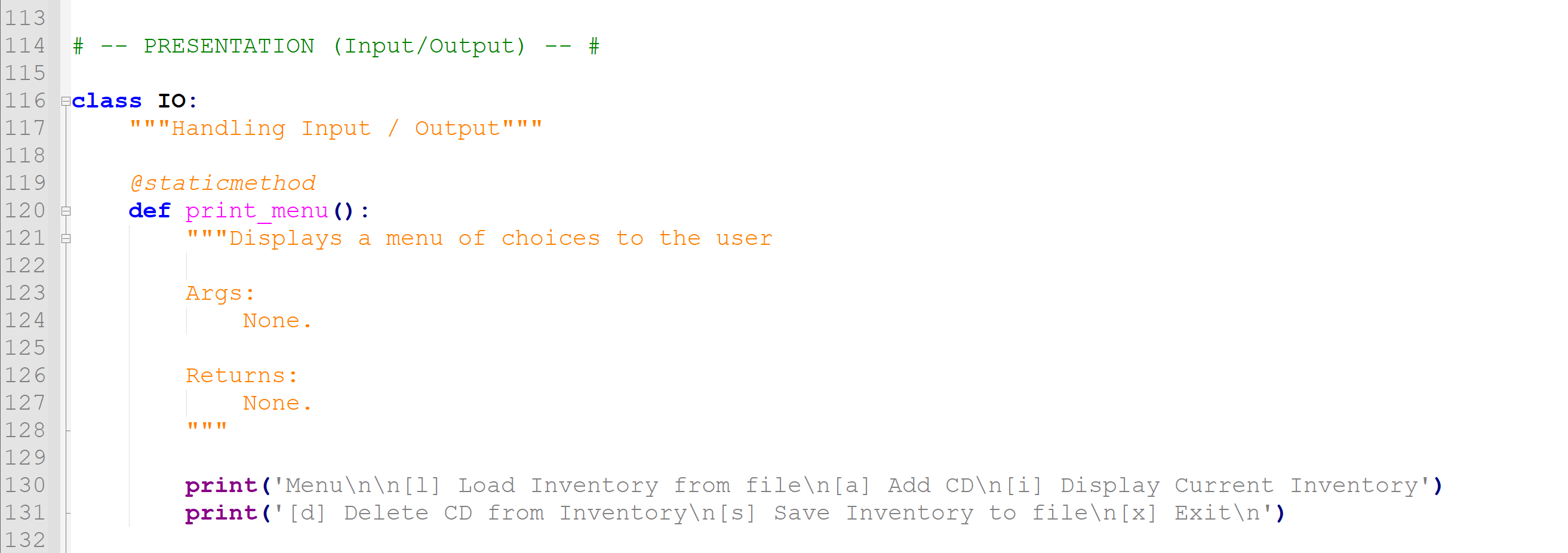


Figure 5: IO class print\_menu function displays directions and menu options to user.

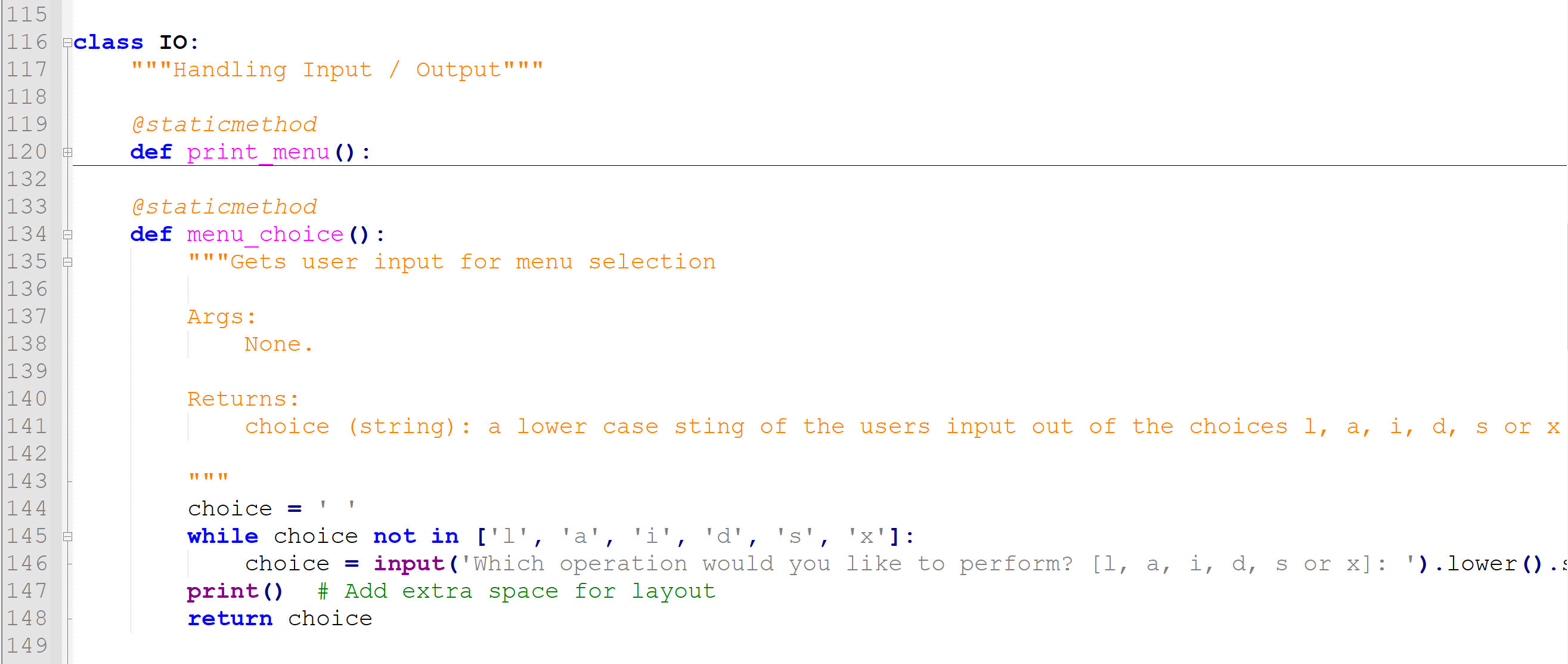


Figure 6: IO class menu\_choice function displays menu and returns user choice selected.

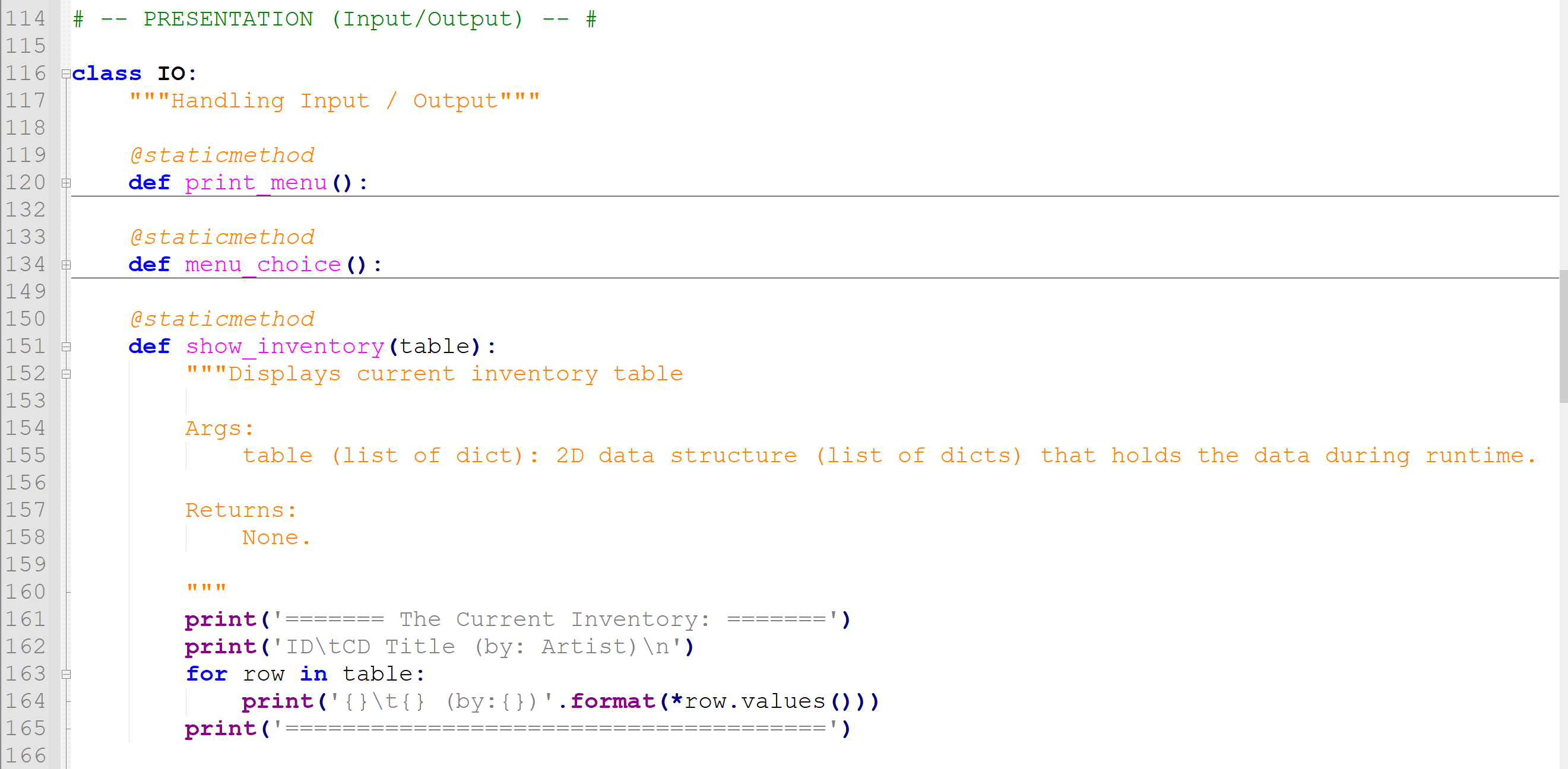


Figure 7: IO class show\_inventory function displays current cd inventory from internal list of dicts data structure.

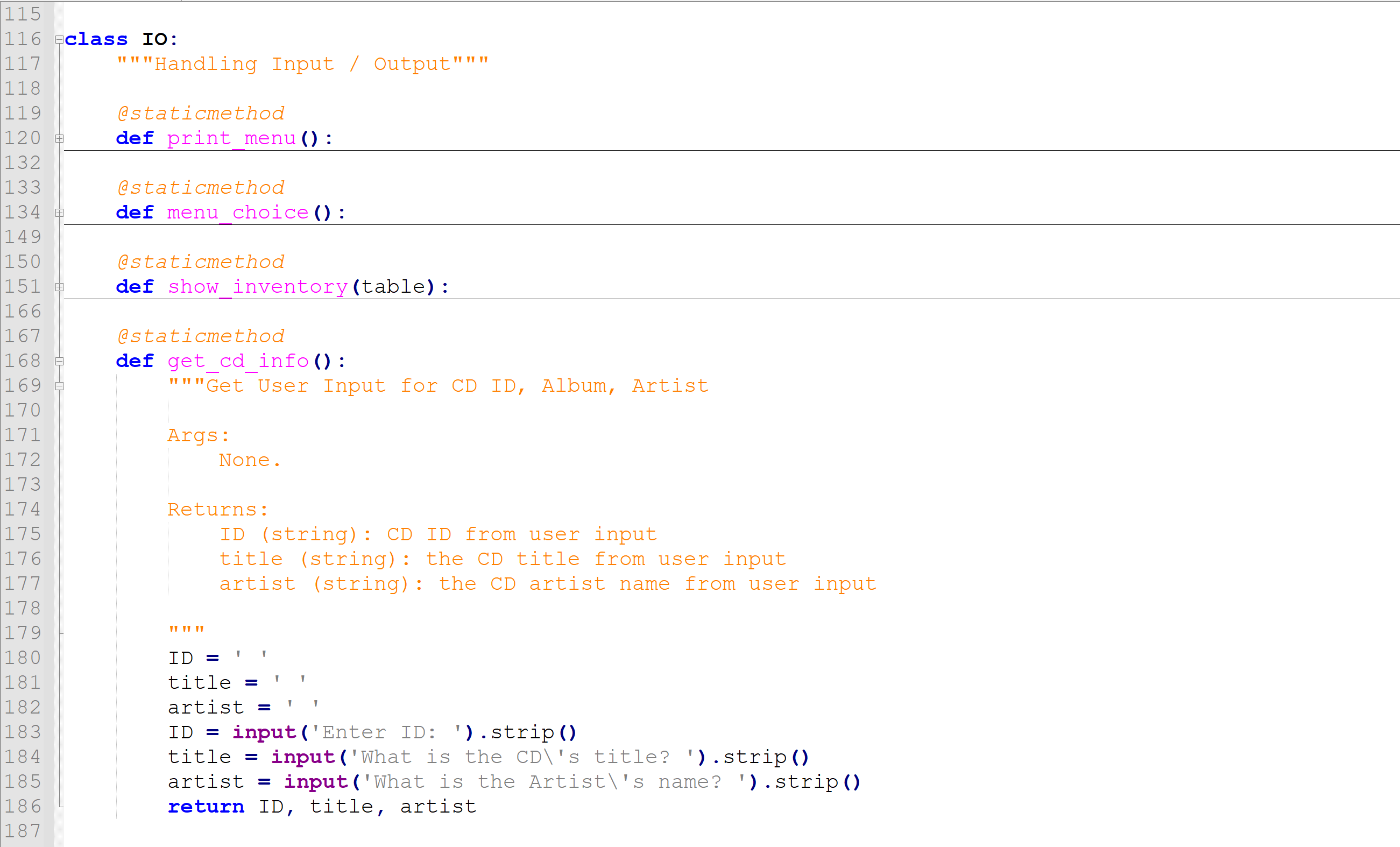


Figure 8: IO class get\_cd\_info function presents questions to user and returns data entered.

To start, variables, lists and dictionaries are initialized. The file name is also assigned to a variable. The loop control is also changed. At the end of this section, the script still presents a menu to the user and the user is asked to choose an option. A WHILE loop is employed to process the user’s choice and allow selecting from the menu multiple times. The WHILE loop continues presenting the menu until it is no longer TRUE or the users chooses to exit (‘x’) the program:

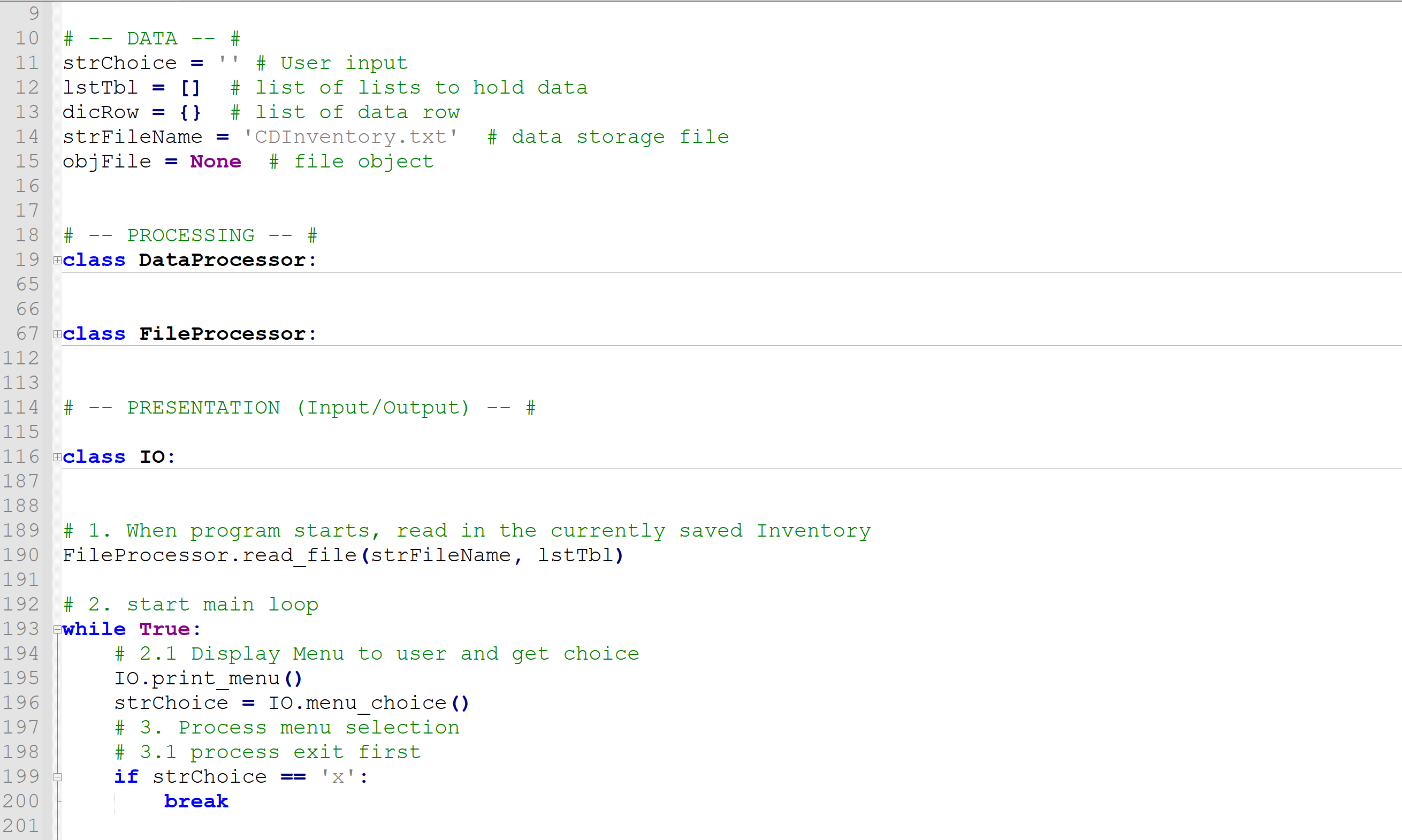


Figure 9: Script CDInventory.py processing details exit (‘x’) and ends script when user chooses this option.

In the processing section, IF statement logic is still used to evaluate the user’s menu selection control the scripts behavior. Structured similar to a CASE statement in other languages, the IF statement routes the script depending on the option selected by the user. The first two user options in this section is exit the script or load data from the file. When the user selects ‘x’ from the menu, the script will terminate.

If the user selects ‘l’, the script load data from the ‘CDInventory.txt’ file. The file is opened in read mode and the script loops through each comma separated line. The script will read each row in the file, split the values from the row and remove the comma separators. The elements in each row will be added as values assigned to specific keys in the dictRow dictionary. The dictionary is then appended to the lstTbl list. This effectively reads the raw data and organizes it into a list of dictionaries. Once EOF is reached, the user is presented with a load complete message and the file is closed.

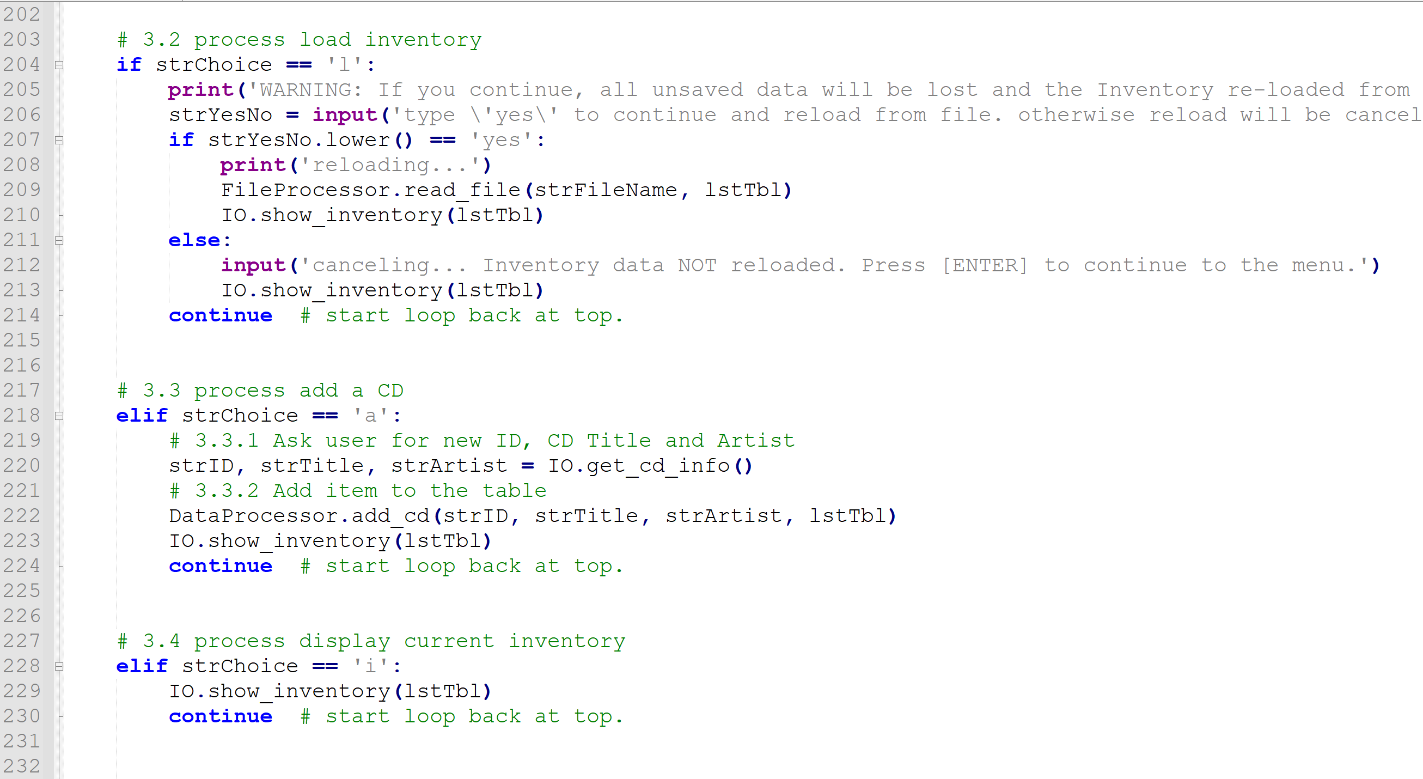


Figure 10: Script CDInventory.py processing details load (‘l’), add cd (‘a’) and show inventory (‘I’). Data is read in a list of dicts.

When the user selects ‘a’ from the menu, the user is able to add a new entry. The script will ask and capture ID, Title and Artist details. The data is capture in the dictRow dictionary and appended to the lstTbl list to append the new dict to the list of dicts.

When the user selects ‘i’ from the menu, the script will list data that has been loaded or appended. The script will print a header first. A loop is then used to loop through the list of dicts to print the values by key and present as ID, Album and Artist to the user.

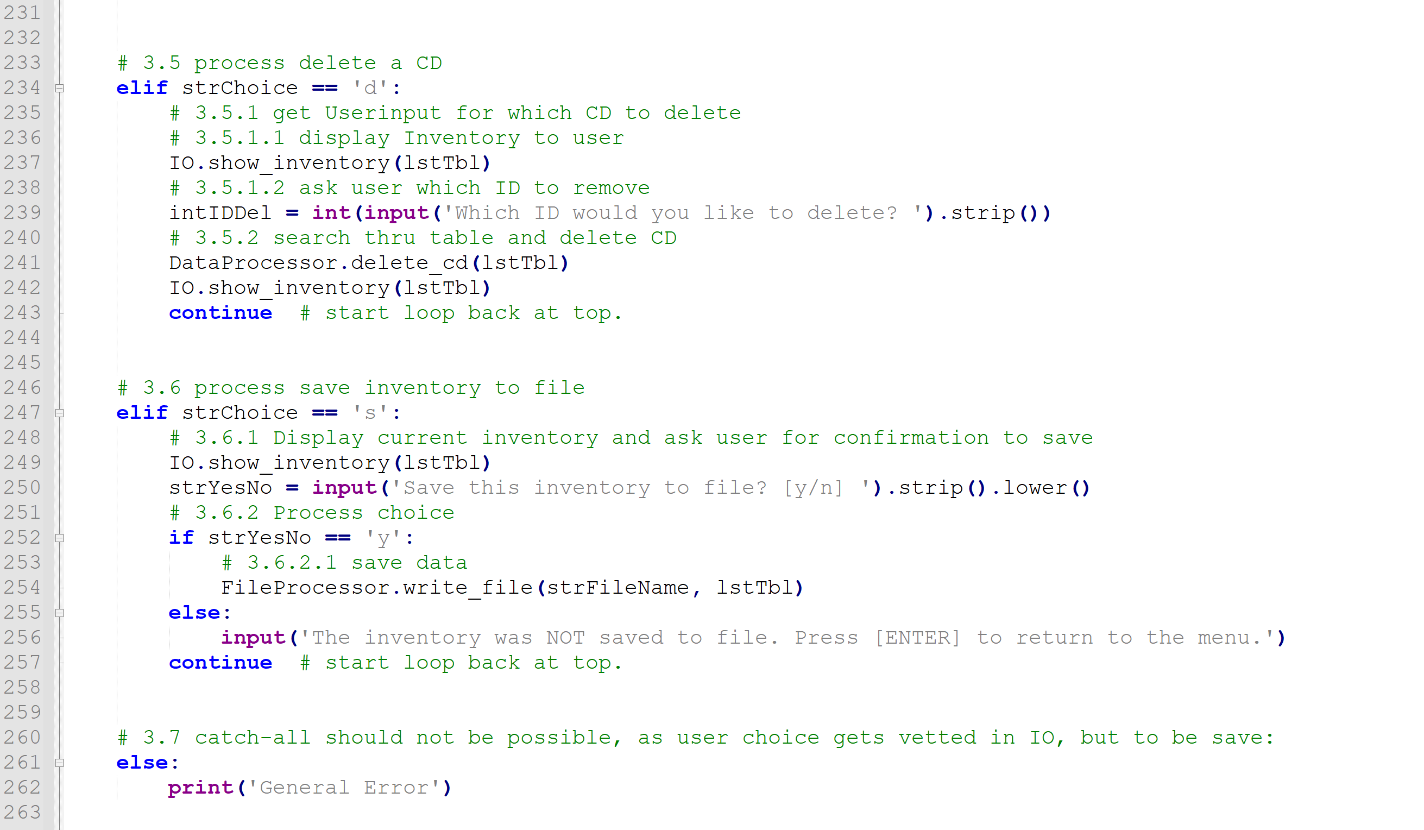


Figure 11: Script CDInventory.py process details Delete (‘d’), Save (‘s’) and error catching. The user can delete data and save current inventory.

When the users selects ‘d’ from the menu, new functionality allows the user to delete rows of data by ID. To process this request, the script asked the user which ID to delete and loops through the list of dicts to find the row for deletion. The loop first determines the number if rows (dicts) in the list and then evaluates ID in each dict. If the user entered value equals the ID key value, the dicts is deleted from the list and the loop exits (“break”).

When the user selects ‘s’ from the menu, the data in the current list of dicts is written to the file. The file is open in write mode (‘w’) which will overwrite the existing file data with the new data. The script starts by opening the ‘CDInventory.txt’ file in write mode. A loop is used to determine the number of rows (dicts) in the list. The loop process each row (dict) and saves the data as a comma separated row of data. Once each dict has been processed and written, a message is displayed to the user and the file is closed.

The final section of the primary loop is the Else statement. This is used for error handling. If the user selects any value that is not expected, it is caught here. The user is presented with a message and returned to the menu for selection an appropriate menu option.

## 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):

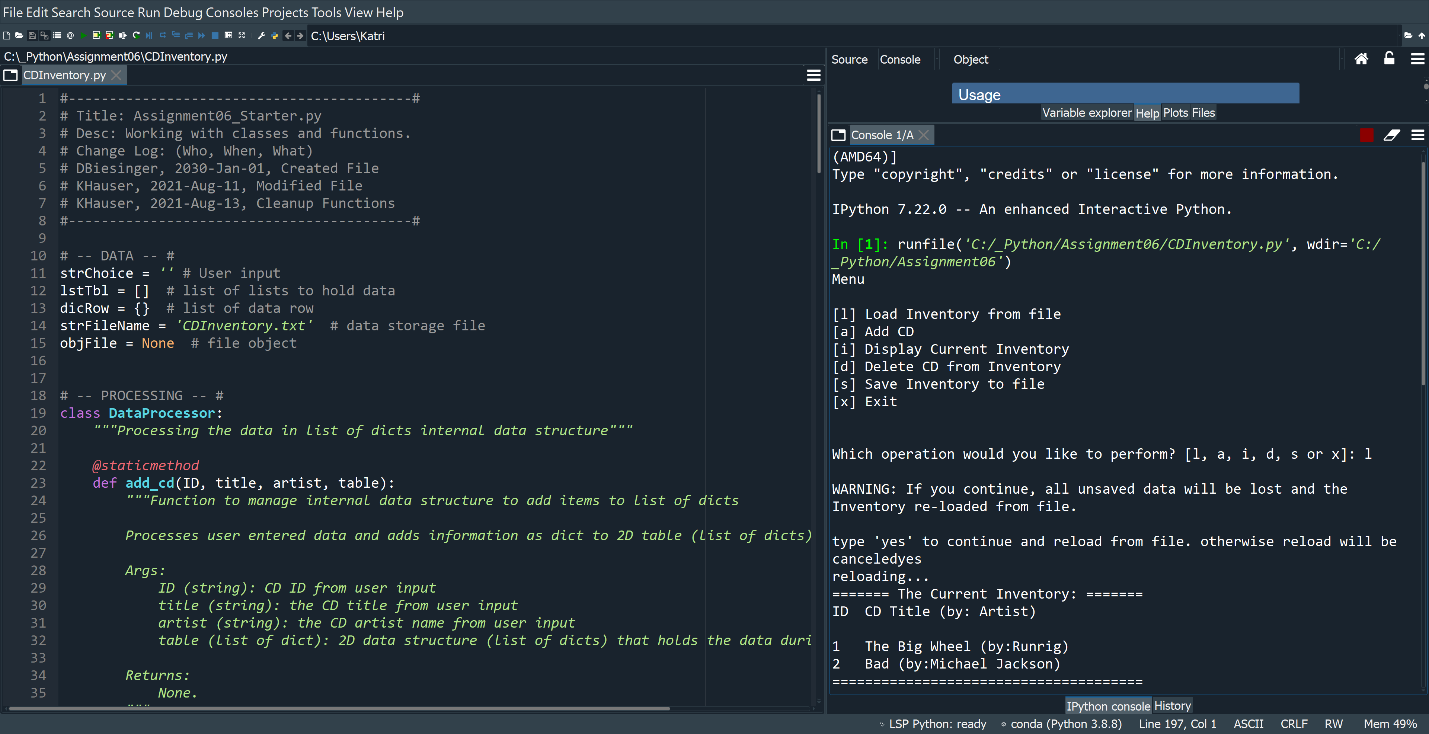


Figure 12: Script CDInventory.py running in Spyder processing load (‘l’) and add (‘a’) commands from user. Program loads from CDInventory.txt file. Adds ID, Title and Album name from user entered data.

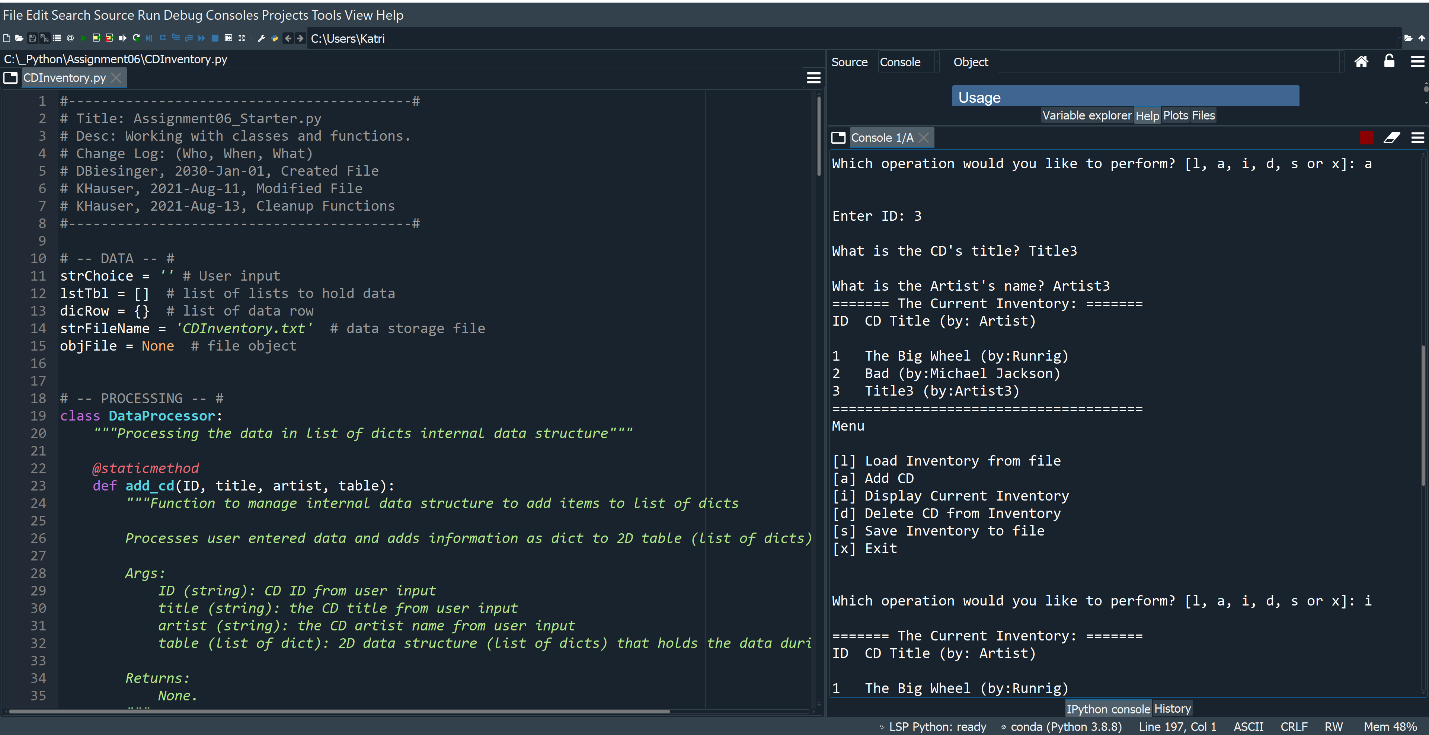


Figure 13: Script CDInventory.py running in Spyder processing display (‘i’) command from user. Program displays data that is stored in inner data structure list of dicts.

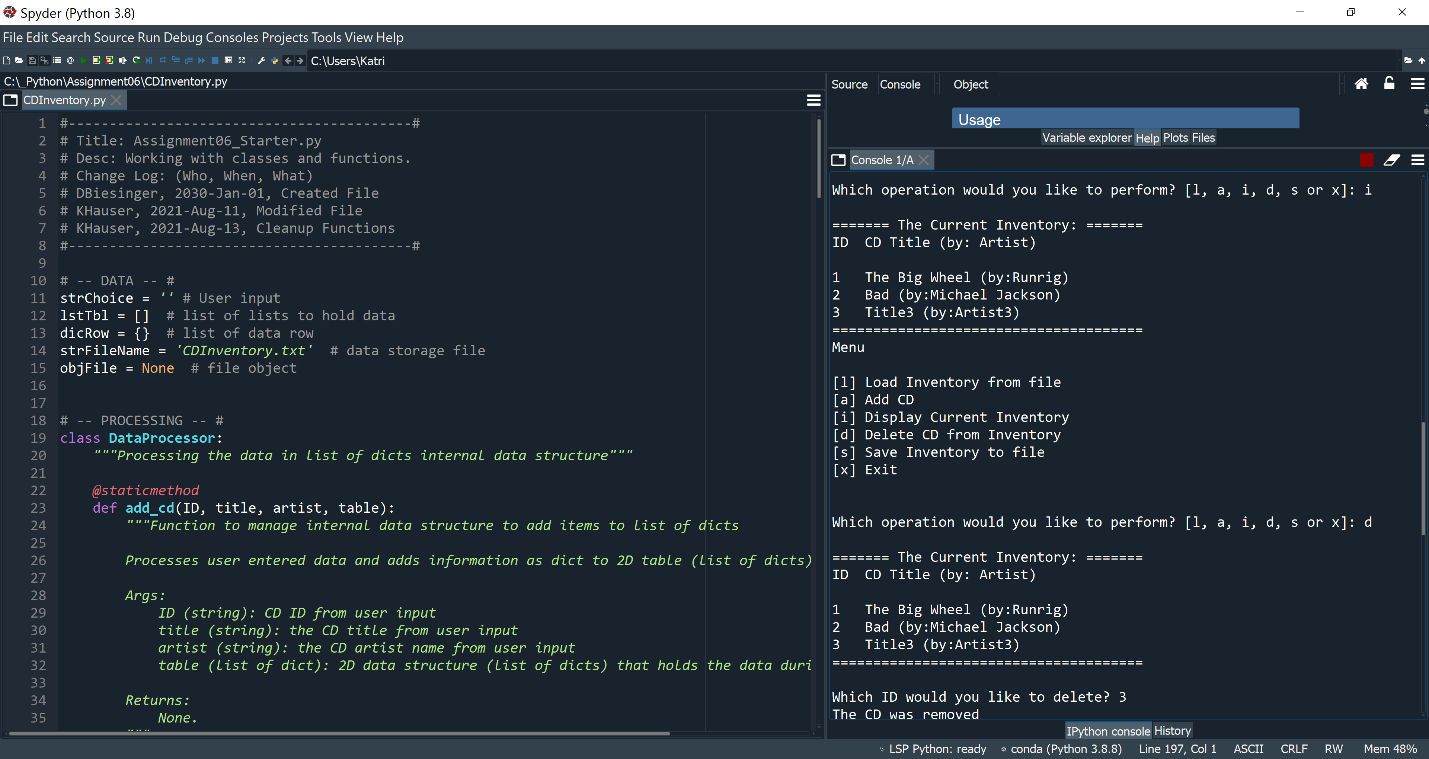


Figure 14: Script CDInventory.py running in Spyder processing delete (‘d’) and display (‘i’) commands from user. Program deletes dict from inner data structure by ID entered by user. Display command showing row deleted.

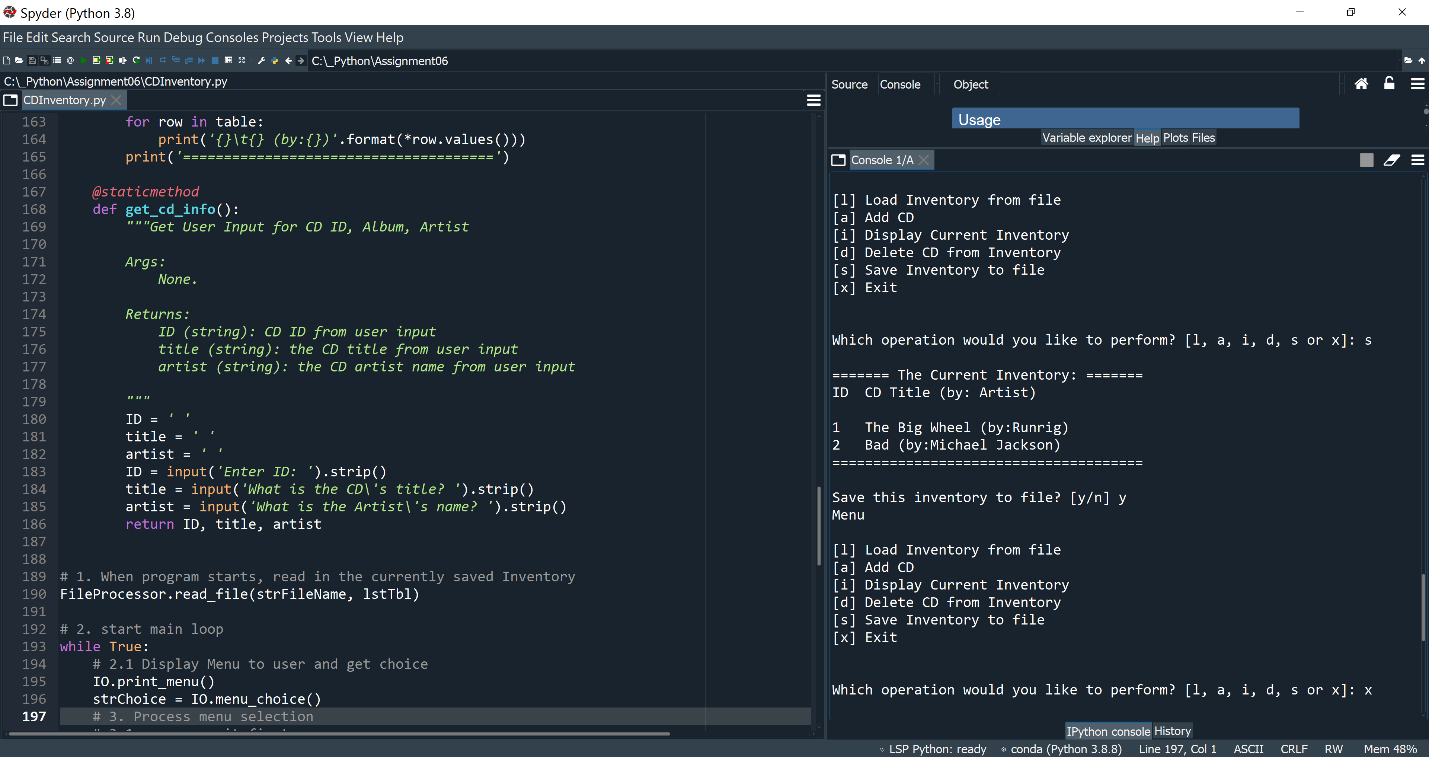


Figure 15: Script CDInventory.py running in Spyder processing save (‘s’) command from user. Program saves data from inner data structure list of dicts to CDInventory.txt file.

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

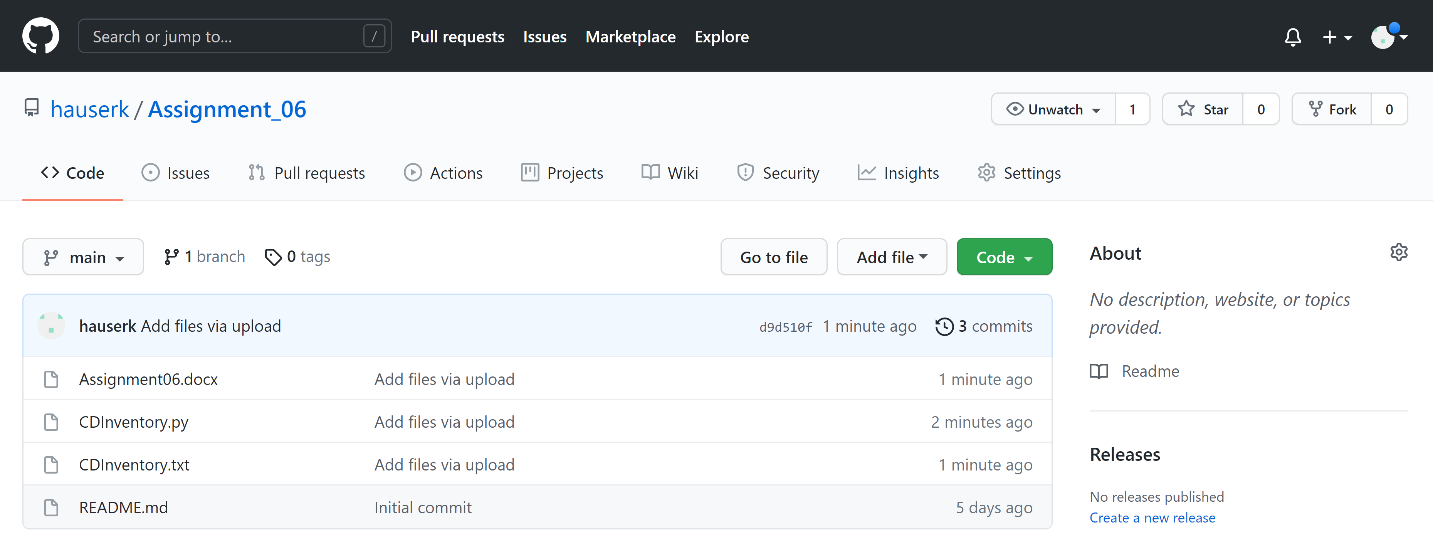


Figure 16: GitHub repository of Assignment07