Michael List

March 7, 2020

Foundations of Programming: Python

Assignment 07

Files & Exceptions

Introduction

This document is about the seventh module and assignment of the Foundations of Programming: Python course. It is covering a lot of programming basics on how to use binary files rather than text files to store data outside of the program and how to work with structured error handling, how the exception class works and how to define your own exceptions. In the second part the gained knowledge is demonstrated by modifying the CDInventory.py file from Assignment 06.

Part 1 – Knowledge

Part 1 of this document explains key concepts learned in this module as outlined in the introduction.

What are the benefits of using structured error handling?

Structured error handling prevents the program from crashing when an error (exception) occurs. Programming languages generally can catch these exceptions as there are a lot of error producing scenarios where something can go wrong (e.g. incorrect input, files not being found, etc.). With structured error handling the programmer can keep the programming running while an error occurs and is also able to advise what should be done next. Using structured error handling can vastly improve the user experience by preventing the program from not crashing, providing customized error messages that are not too technical. Structured error handling also has a big advantage for the programmer: rather than losing data stored in memory when the program crashes the programmer will not lose any data with structured error handling as it will prevent the program from crashing.

What are the differences between a text file and a binary file?

The main purpose of a text file is to be read by humans. This means that the text file needs to be formatted well enough be read by humans or exchanged with external sources. Formatting the file requires processing before it can be written into a text file. A binary file on the other hand is only intended to be read by the program and does not require to be formatted into a readable format. Instead, the program will serialize the data for storage and deserialize it when the file is read. The big advantage of binary file is that the data structure of the written data will be maintained and does not have to be reconstructed when reading out a text file. This can save memory and processing power and allows the programmer to just ‘dump’ even complex data of the memory into an external file.

How is the Exception class used?

When an exception occurs, it creates an object of the Exception class that contains information about the error. The programmer can store and access this information when the exception occurs. For example, the programmer then can use the type() function on an exception class to return e.g. 'ValueError'. This further helps the programmer to handle specific exceptions in a unique way. The exception class can also be used by the programmer to define their own exceptions.

How do you "derive" a new class from the Exception class and when?

To define a custom exception the programmer can derive a new class from the exception class. This can be done when some unique errors are expected that are not represented by the predefined exceptions. However, if possible, this should be avoided if possible and the programmer should stay with the build-in exceptions as much as possible.

In Lab07_C I went ahead and created a new exception class in case the user input is zero. However, this could have been accomplished with some conditional statements as well.

```python
class ValueIsZero(Exception):
    """Value is zero"""
    def __str__(self):
        return 'Value entered cannot be zero'
```

Listing 1 - Deriving a new exception class

What is the Markdown language?

Github use their own markdown language to format text stored in the readme's in the repositories. The specifications for this markdown language can be found here: https://github.github.com/gfm/. As the readme.md files are the landing pages of our repository we can use the markdown language to make the landing page more visually appealing. The markdown language let's the programmer quickly format the documents that then get converted into HTML by the servers.

Examples of error handling in Python

When researching error handling in Python I came across the Python.org[1] documentation. This documentation mentions two interesting features:

1. Base classes for user-defined exceptions

   Base classes are used when the programmer is defining multiple, custom exceptions. When defining Exceptions that can be grouped together in a module with individual errors the programmer can create a base class and then subclasses for the specific exception classes

2. Clean-up Actions
   The same documentation also mentions the optional clause of a try-except statement: 'finally'. The 'finally' clause will be always executed and is intended to define clean-up actions. Clean-up actions are actions that need to be executed regardless if the try-except statement was successful or not. An example could be network connections.

Examples of pickling in Python

To get a good overview about use cases of pickling in Python I came across a stackoverflow [2]page to get a better impression when pickling is useful. The following stood out the most to me:

- Easily sending data over a network connection (however, this can be a security issue)
- Easily restarting a program and picking of when the program was terminated
- Parallel or distributed computing
- Saving complex datatypes in a database

[1] https://docs.python.org/3/tutorial/errors.html – Retrieved 2020-March-07
[2] https://stackoverflow.com/questions/3438675/common-use-cases-for-pickle-in-python – Retrieved 2020-March-7

Part 2 – Modify CDInventory.py

The coding assignment portion of this assignment was to modify the CDInventory.py file from assignment 06 by changing the CDInventory.txt file to CDInventory.dat. This part of the assignment did not cause any issues for me. The other part of the assignment was to add some structured error handling for user interaction, casting of strings to integers and file access operations using the build in exceptions. My submission of this assignment can also be found on Github: https://github.com/List-Michael/Assignment_07[3]

I already included a try-except statement with structured error handling in my previous assignments, so this portion was not a problem and has been already completed by me. However, I still found a little challenge when I was trying to introduce some structured error handling for user inputs and when casting the input into an integer. In the Assignment 06 file I requested the user input for ID and accepted strings in the cd_user_input() and then casted the input into a string into an integer in append_cd_inventory_memory_list().

To implement proper error handling in this situation I moved the casting of the input string to an integer into the cd_user_input() function. Doing so let me allow to validate directly if the input was an integer or not and prevent any non-integer inputs. After implementing this functionality, I remembered that there is no technically need to cast the input into an integer to use it as an ID, however, this check also prevents any alphabetic characters to be entered as an ID.

It also took me a little while to figure out how to implement loops with structured error handling that is supposed to catch faulty user input. See my solution of this issue in function cd_user_input() in the listing below:

```python
def cd_user_input():
    flag = True
    while flag == True:
        try:
            strID = input('Enter a numerical ID: ').strip()
            intID = int(strID)
            flag = False
        except ValueError:
            print ('The entered ID is not an integer. Please enter a number')
    strTitle = input('What is the CD\'s title? ').strip()
    strArtist = input('What is the Artist\'s name? ').strip()
    return intID, strTitle, strArtist
```

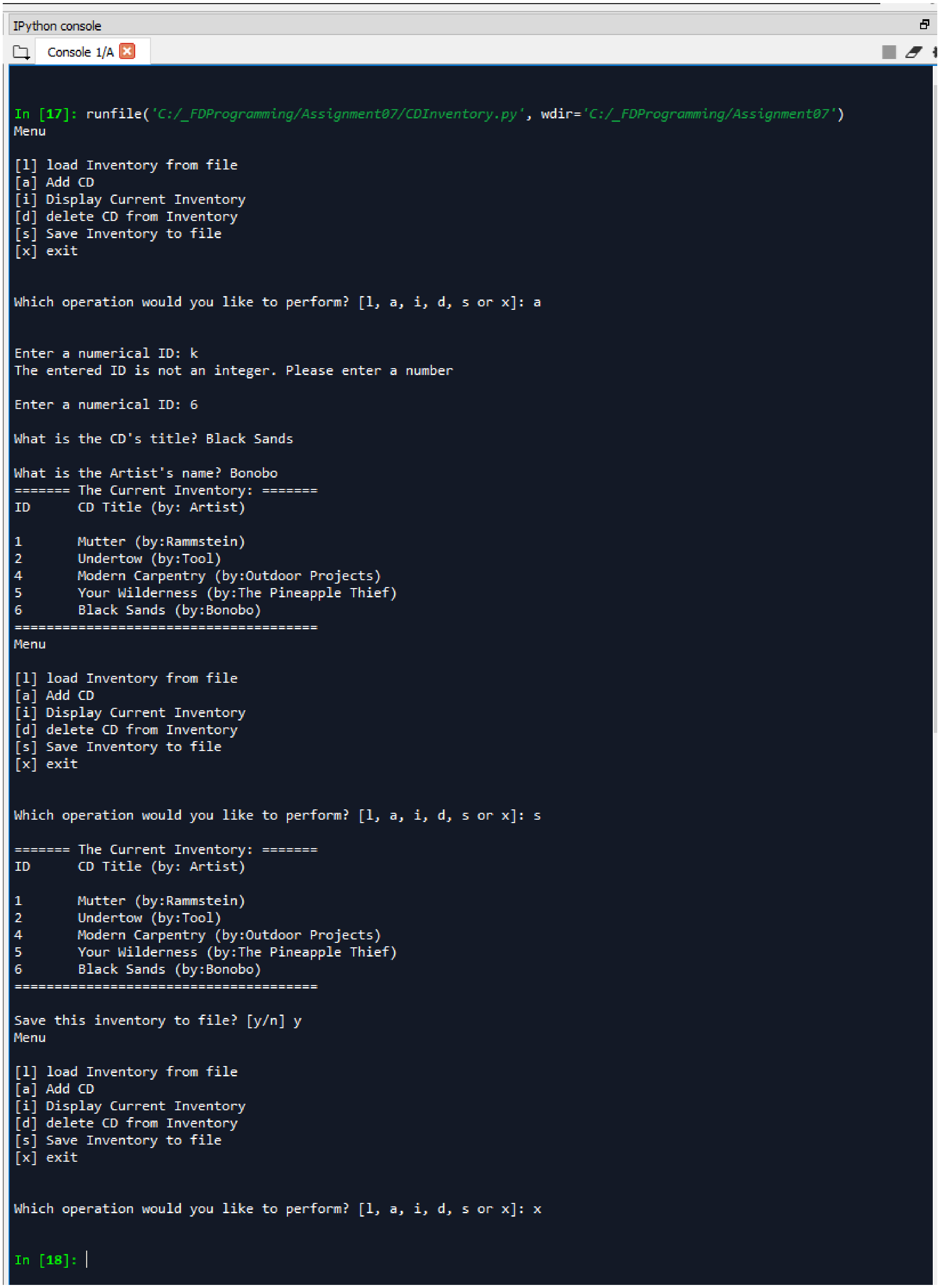
Listing 2 - Input loop with structured error handling

There were two other interesting items I noticed during this assignment:

1. It was also interesting to realize that user input only needs to be restricted by using exception when the input will be used later to do some calculations or casting. For example, there is no need to add exception checking when the input is only used to navigate a menu as any entered options that are not supported by the menu will automatically fall in an else branch
2. It was still surprising to see how much easier it was to edit this week's code as the functions added last week helped a lot with the separation of concerns

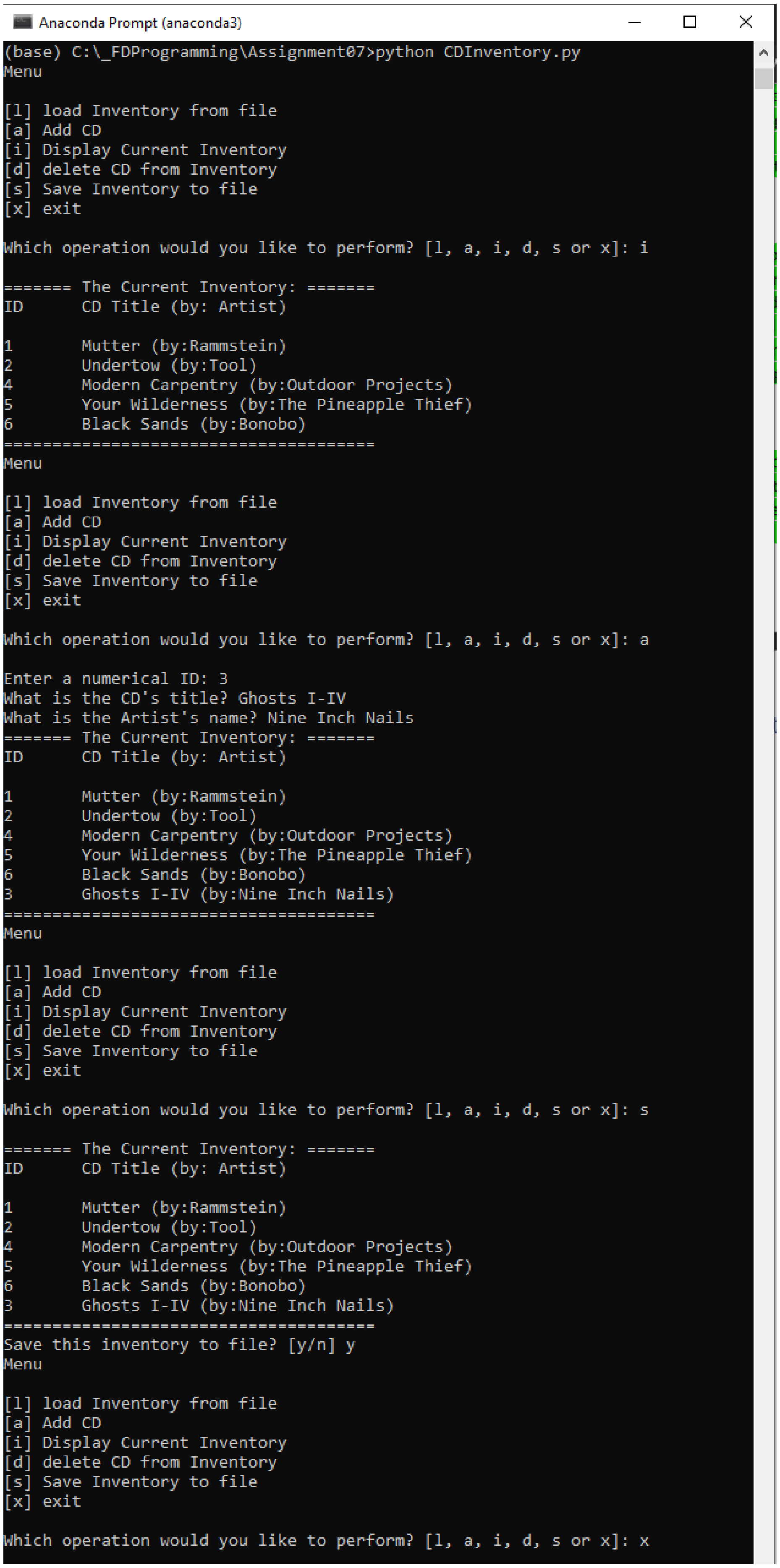
The next two screenshots show the updated CDInventory.py executed in Spyder (first screenshot) and the Anaconda Prompt (second screenshot). In the first screenshot I also entered an letter as ID input to demonstrate the structured error handling.

[3] Created and Retrieved on 2020-Mar-07



```
In [17]: runfile('C:/_FDProgramming/Assignment07/CDInventory.py', wdir='C:/_FDProgramming/Assignment07')
Menu

[l] load Inventory from file
[a] Add CD
[i] Display Current Inventory
[d] delete CD from Inventory
[s] Save Inventory to file
[x] exit


Which operation would you like to perform? [l, a, i, d, s or x]: a


Enter a numerical ID: k
The entered ID is not an integer. Please enter a number

Enter a numerical ID: 6

What is the CD's title? Black Sands

What is the Artist's name? Bonobo
======= The Current Inventory: =======
ID      CD Title (by: Artist)

1       Mutter (by:Rammstein)
2       Undertow (by:Tool)
4       Modern Carpentry (by:Outdoor Projects)
5       Your Wilderness (by:The Pineapple Thief)
6       Black Sands (by:Bonobo)
======================================
Menu

[l] load Inventory from file
[a] Add CD
[i] Display Current Inventory
[d] delete CD from Inventory
[s] Save Inventory to file
[x] exit


Which operation would you like to perform? [l, a, i, d, s or x]: s

======= The Current Inventory: =======
ID      CD Title (by: Artist)

1       Mutter (by:Rammstein)
2       Undertow (by:Tool)
4       Modern Carpentry (by:Outdoor Projects)
5       Your Wilderness (by:The Pineapple Thief)
6       Black Sands (by:Bonobo)
======================================

Save this inventory to file? [y/n] y
Menu

[l] load Inventory from file
[a] Add CD
[i] Display Current Inventory
[d] delete CD from Inventory
[s] Save Inventory to file
[x] exit


Which operation would you like to perform? [l, a, i, d, s or x]: x


In [18]: 
```

Figure 1 - CDInventory.py executed in Spyder



Anaconda Prompt (anaconda3)

```
(base) C:\_FDProgramming\Assignment07>python CDInventory.py
Menu

[l] load Inventory from file
[a] Add CD
[i] Display Current Inventory
[d] delete CD from Inventory
[s] Save Inventory to file
[x] exit

Which operation would you like to perform? [l, a, i, d, s or x]: i

======= The Current Inventory: =======
ID      CD Title (by: Artist)

1       Mutter (by:Rammstein)
2       Undertow (by:Tool)
4       Modern Carpentry (by:Outdoor Projects)
5       Your Wilderness (by:The Pineapple Thief)
6       Black Sands (by:Bonobo)
======================================
Menu

[l] load Inventory from file
[a] Add CD
[i] Display Current Inventory
[d] delete CD from Inventory
[s] Save Inventory to file
[x] exit

Which operation would you like to perform? [l, a, i, d, s or x]: a

Enter a numerical ID: 3
What is the CD's title? Ghosts I-IV
What is the Artist's name? Nine Inch Nails
======= The Current Inventory: =======
ID      CD Title (by: Artist)

1       Mutter (by:Rammstein)
2       Undertow (by:Tool)
4       Modern Carpentry (by:Outdoor Projects)
5       Your Wilderness (by:The Pineapple Thief)
6       Black Sands (by:Bonobo)
3       Ghosts I-IV (by:Nine Inch Nails)
======================================
Menu

[l] load Inventory from file
[a] Add CD
[i] Display Current Inventory
[d] delete CD from Inventory
[s] Save Inventory to file
[x] exit

Which operation would you like to perform? [l, a, i, d, s or x]: s

======= The Current Inventory: =======
ID      CD Title (by: Artist)

1       Mutter (by:Rammstein)
2       Undertow (by:Tool)
4       Modern Carpentry (by:Outdoor Projects)
5       Your Wilderness (by:The Pineapple Thief)
6       Black Sands (by:Bonobo)
3       Ghosts I-IV (by:Nine Inch Nails)
======================================
Save this inventory to file? [y/n] y
Menu

[l] load Inventory from file
[a] Add CD
[i] Display Current Inventory
[d] delete CD from Inventory
[s] Save Inventory to file
[x] exit

Which operation would you like to perform? [l, a, i, d, s or x]: x
```

Figure 2 - CDInventory.py executed in Anaconda Prompt

Summary

In this assignment the students were introduced to binary files and structured error handling with exceptions. The gained knowledge was demonstrated in the coding assignment where last week's assignment file was modified to save program data in a binary CDInventory.dat file rather than the previously used text file CDInventory.txt. The program was also modified to include various structured error handling for user interaction, casting of strings to integers and file access operations using the build in exceptions.

Appendix

Listing 3 - CDInventory.py

```python
#------------------------------------------#
# Title: CDInventory.py
# Desc: Working with classes and functions.
# Change Log: (Who, When, What)
# DBiesinger, 2030-Jan-01, Created File
# MList, 2020-Mar-01, Fixed error in function read_file() as script crashes when no CDInventory.txt is present
# MList, 2020-Mar-01, Completed TODOs when 'a' is called and created functions append_cd_inventory_memory_list() + IO.cd_user_input()
# MList, 2020-Mar-01, Completed TODO for delete function, created delete_CD() and tested functionality
# MList, 2020-Mar-01, Completed TODO for saving function, build out shell function for write_file()
# MList, 2020-Mar-01, Ensured proper commenting in main body of script
# MList, 2020-Mar-01, Added docstrings for newly written functions
# MList, 2020-Mar-06, Changed CDInventory.txt to CDInventory.dat and changed file accesses from text to binary
# Mlist, 2020-Mar-07, Added exceptions for user interactions, type casting (string to int) and file access operations where needed. Updated docstrings
#------------------------------------------#
import pickle
# -- DATA -- #
strChoice = '' # User input
lstTbl = []  # list of lists to hold data
dicRow = {}  # dictionary of data row
strFileName = 'CDInventory.dat'  # data storage file
objFile = None  # file object

# -- PROCESSING -- #
class DataProcessor:

    @staticmethod
    #result is assinged to local variable table which is referencing the same variable as the global lstTbl
    def append_cd_inventory_memory_list(intID, strTitle, strArtist, table):
        """Function to append new user entry as a dictionary within a 2D list

        Uses passed arguements of CD information to create a new CD entry as a dictionary and adds dictionary as a new line to global 2D table

        Args:
            intID (integer): Requested ID of CD the user would like to add
            strTitle (string): Requested CD Title the user would like to add
            strArtist (string): Requested Artist of CD the user would like to add
            table (list of dict): 2D data structure (list of dicts) that holds the data during runtime

        Returns:
            None.
        """
        dicRow = {'ID': intID, 'Title': strTitle, 'Artist': strArtist}
        table.append(dicRow)


    @staticmethod
    def delete_CD(ID, table):
        """Function to delete CD from 2D list based on ID match

        User defines the ID of a CD the user would like to remove. ID gets passed to this function is used to identify a match in global 2D table.
        If match is identified by matching value of ID key, entry dictionary will be removed from global 2D table.

        Args:
            ID (integer): Requested ID of CD the user would like to delete. Casted into integer prior function call
            table (list of dict): 2D data structure (list of dicts) that holds the data during runtime

        Returns:
            None.
        """
        intRowNr = -1
        blnCDRemoved = False
        for row in table:
            intRowNr += 1
            if row['ID'] == ID:
                del table[intRowNr]
                blnCDRemoved = True
                break
        if blnCDRemoved:
            print('The CD was removed')
        else:
            print('Could not find this CD!')

class FileProcessor:
    """Processing the data to and from text file"""
    @staticmethod
    def read_file(file_name):
        """Function to manage data ingestion from file to a list of dictionaries

        Reads the data from file identified by file_name into a global 2D table
        (list of dicts) table one line in the file represents one dictionary row in table.
        Catches IOError in case the file does not exist.

        Args:
            file_name (string): name of file used to read the data from
        Returns:
            None.
        """
        try:
            with open(file_name, 'rb') as objFile:
                table = pickle.load(objFile)
        except IOError:
            print('\nThere is currently no existing inventory file\n')
        return table

    @staticmethod
    def write_file(file_name, table):
        """Function to write added data to a binary file

        Reads the data of dictionaries stored in 2D table.
              Saves data as a binary file with pickle

              Args:
                  file_name (string): name of file used to read the data to
                  table (list of dict): 2D data structure (list of dicts) that holds the data during runtime

              Returns:
                  None.
              """

              with open(file_name, 'wb') as objFile:
                  pickle.dump(table, objFile)

      # -- PRESENTATION (Input/Output) -- #

```

```python
class IO:
    """Handling Input / Output"""

    @staticmethod
    def print_menu():
        """Displays a menu of choices to the user

        Args:
            None.

        Returns:
            None.
        """

        print('Menu\n\n[l] load Inventory from file\n[a] Add CD\n[i] Display Current Inventory')
        print('[d] delete CD from Inventory\n[s] Save Inventory to file\n[x] exit\n')

    @staticmethod
    def menu_choice():
        """Gets user input for menu selection

        Args:
            None.

        Returns:
            choice (string): a lower case sting of the users input out of the choices l, a, i, d, s or x

        """
        choice = ' '
        while choice not in ['l', 'a', 'i', 'd', 's', 'x']:
            choice = input('Which operation would you like to perform? [l, a, i, d, s or x]: ').lower().strip()
        print()  # Add extra space for layout
        return choice

    @staticmethod
    def show_inventory(table):
        """Displays current inventory table


        Args:
            table (list of dict): 2D data structure (list of dicts) that holds the data during runtime.

        Returns:
            None.

        """
        print('======= The Current Inventory: =======')
        print('ID\tCD Title (by: Artist)\n')
        for row in table:
            print('{}\t{} (by:{})'.format(*row.values()))
        print('======================================')

    @staticmethod
    def cd_user_input():
        """Gets user input to add new CD

        Catches ValueError in case the entered ID for the CD is not an integer.
        If this exception is caught the program keeps asking the user to enter a numerical ID
        until an integer has been entered.

        Args:
            None.

        Returns:
            intID, strTitle, strArtist (tuple): Returns tuple with the 3 variables: ID, Title and Artist

        """
        flag = True
        while flag == True:
            try:
                strID = input('Enter a numerical ID: ').strip()
                intID = int(strID)
                flag = False
            except ValueError:
                print ('The entered ID is not an integer. Please enter a number')
        strTitle = input('What is the CD\'s title? ').strip()
        strArtist = input('What is the Artist\'s name? ').strip()
        return intID, strTitle, strArtist


# 1. When program starts, read in the currently saved Inventory
lstTbl = FileProcessor.read_file(strFileName)
# 2. start main loop
while True:
    # 2.1 Display Menu to user and get choice
    IO.print_menu()
    strChoice = IO.menu_choice()
    # 3. Process menu selection
    # 3.1 process exit first
    if strChoice == 'x':
        break
    # 3.2 process load inventory
    if strChoice == 'l':
        print('WARNING: If you continue, all unsaved data will be lost and the Inventory re-loaded from file.')
        strYesNo = input('type \'yes\' to continue and reload from file. otherwise reload will be canceled: ')
        if strYesNo.lower() == 'yes':
            print('reloading...')
            lstTbl = FileProcessor.read_file(strFileName)
            IO.show_inventory(lstTbl)
        else:
            input('canceling... Inventory data NOT reloaded. Press [ENTER] to continue to the menu.')
            IO.show_inventory(lstTbl)
        continue  # start loop back at top.
    # 3.3 process add a CD
    elif strChoice == 'a':
        # 3.3.1 Ask user for new ID, CD Title and Artist and unpacking return tuple
        strIDGlobal, strTitleGlobal, strArtistGlobal = IO.cd_user_input()
        # 3.3.2 Add item to the table
        DataProcessor.append_cd_inventory_memory_list(strIDGlobal, strTitleGlobal, strArtistGlobal, lstTbl)
        # 3.3.3 Displaying current inventory
        IO.show_inventory(lstTbl)
        continue  # start loop back at top.
    # 3.4 process display current inventory
    elif strChoice == 'i':
        IO.show_inventory(lstTbl)
        continue  # start loop back at top.
    # 3.5 process delete a CD
    elif strChoice == 'd':
        # 3.5.1 get Userinput for which CD to delete
        # 3.5.1.1 display Inventory to user
        IO.show_inventory(lstTbl)
        # 3.5.1.2 ask user which ID to remove
        try:
            intIDDel = int(input('Which ID would you like to delete? ').strip())
            # 3.5.2 search thru table and delete CD
            # 3.5.2.1 delete entry
            DataProcessor.delete_CD(intIDDel, lstTbl)
```

```python
                    # 3.5.2.2 display Inventory to user
                    IO.show_inventory(lstTbl)
                except ValueError:
                        print ('Faulty input: The entered ID is not a number (integer)\n')
                continue  # start loop back at top.
            # 3.6 process save inventory to file
            elif strChoice == 's':
                # 3.6.1 Display current inventory and ask user for confirmation to save
                IO.show_inventory(lstTbl)
                strYesNo = input('Save this inventory to file? [y/n] ').strip().lower() #No Error handling required
                # 3.6.2 Process choice
                if strYesNo == 'y':
                    # 3.6.2.1 save data
                    FileProcessor.write_file(strFileName, lstTbl)
                else:
                    input('The inventory was NOT saved to file. Press [ENTER] to return to the menu.')
                continue  # start loop back at top.
            # 3.7 catch-all should not be possible, as user choice gets vetted in IO, but to be save:
            else:
                print('General Error')
```