Greg Eisele

Nov 21, 2022

IT FDN 110 B

Assignment 07

Introduction

Week 7's lesson introduced us to more ways of working with text files, a brief intro to binary files, and structured error handling.

Assignment

Week 7's assignment had two specific actions -

1) Modify week 6's program, adding structured error handling to areas of the code where there is user interaction.

2) Modify week 6's program, changing the data storage type from text to binary.

Error Handling

Error handling was explained quite well in the book, but I found the following website to be additionally helpful -

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

The CD Inventory program already has some built in error proofing. For example, when faced with the main menu, typing anything other than the options will not throw an error, but ask again for another selection. Additionally, the load inventory ('l') selection in the main menu also has built in error proofing. Any entry other than 'yes' will force you back to the main menu.

After playing around with each menu option, intentionally trying to break the program, I could only find two areas where the program would break (Menu Add and Menu Delete); these are the areas where I focused on adding error handling.

Main Menu - Add CD ('a')

Figure 1 shows the main menu - add CD code block from week 6 assignment.

```
elif strChoice == 'a':
    # 3.3.1 Ask user for new ID, CD Title and Artist
    IO.ask_user()
    # 3.3.2 Add item to the table
    DataProcessor.add_row()
    IO.show_inventory(lstTbl)
    continue  # start loop back at top.
```

Figure 1 - Main Menu - Add CD Code Block

Entering a non-integer value when prompted "Enter ID:" will result in an 'ValueError' (figure 2).

```
ValueError: invalid literal for int() with base 10: 'h'
```

Figure 2 - ValueError

To address this, I modified the code block to execute in a 'while' loop, with 'try' and 'except' functions added (figure 3).

```
elif strChoice == 'a':
    # 3.3.1 Ask user for new ID, CD Title and Artist
    while True:
        IO.ask_user()
    # 3.3.2 Add item to the table
        try:
            DataProcessor.add_row()
            break
        except (ValueError):
            print('USER ERROR!\nOnly numbers can be entered for ID!  Try again!\n')
    IO.show_inventory(lstTbl)
    continue  # start loop back at top.
```

Figure 3 - Main Menu - Add CD Code Block with Error Handling

Anytime the user input is not an integer value, the while loop raises an exception for ValueError, and prints "USER ERROR….." When an integer value is entered, the while loop is broken, and the program moves on to the next logic step.

Main Menu - Add CD ('d')

Entering a non-integer value when prompted "Which ID…" will result in an error, similar to the previous example. Figure 4 shows the main menu - delete CD code block from assignment 6.

```
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
    intIDDel = int(input('Which ID would you like to delete? ').strip())
    # 3.5.2 search thru table and delete CD
    DataProcessor.del_row()
    IO.show_inventory(lstTbl)
    continue  # start loop back at top.
```

Figure 4 - Main Menu - Delete CD Code Block

Entering a non-integer value when prompted "Which ID ….." will result in an 'ValueError' (figure 5).

```
ValueError: invalid literal for int() with base 10: 'h'
```

Figure 5 - ValueError

Similar to the last section, we'll capture the input inside a 'while' loop, using the 'try' and 'except' functions (figure 6). If a non integer value is entered, the ValueError exception prints, and the while loop runs again. Once an integer value is entered, the while loop breaks, and the program continues (figure 6).

```
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
    while True:
        try:
            intIDDel = int(input('Which ID would you like to delete? ').strip())
            break
        except (ValueError):
            print('Only numbers can be entered for ID!  Try again!\n')
    # 3.5.2 search thru table and delete CD
    DataProcessor.del_row()
    IO.show_inventory(lstTbl)
    continue  # start loop back at top.
```

Figure 6 - Main Menu - Delete CD Code Block with Error Handling

Binary Data Storage

Binary data storage was a little tricky to understand at first, but this youtube clip helped fill in some of the blanks for me -

https://www.youtube.com/watch?v=YBz3ERXQw_M

Modifying week 6's assignment to use Binary data storage, instead of text, actually simplified the code. We no longer had to worry about formatting into a text file.

First, the most important parts of switching to binary data storage, was changing the file type from '.txt' to '.dat', and importing the built in function 'pickle.' See figures 7 and 8 for code change.

```
# -- DATA -- #
strChoice = '' # User input
lstTbl = []  # list of lists to hold data
dicRow = {}  # list of data row
strFileName = 'CDInventory.txt'  # data storage file
objFile = None  # file object
```

Figure 7 - Text File Name

```
import pickle

# -- DATA -- #
strChoice = '' # User input
lstTbl = []  # list of lists to hold data
dicRow = {}  # list of data row
strFileName = 'CDInventory.dat'  # data storage file
objFile = None  # file object
```

Figure 8 - Binary File Name

The only blocks of code that needed to be revised to support the switch from text to binary storage are the 'read_file' and 'write_file' function blocks. For week 6's 'read_file' code block, see figure 9.

```
@staticmethod
def read_file(file_name, table):
    """Function to manage data ingestion from file to a list of dictionaries.

    Reads the data from file identified by file_name into a 2D table
    (list of dicts) table one line in the file represents one dictionary row in table.

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

    Returns:
        None.
    """
    table.clear()  # this clears existing data and allows to load data from file
    objFile = open(file_name, 'r')
    for line in objFile:
        data = line.strip().split(',')
        dicRow = {'ID': int(data[0]), 'Title': data[1], 'Artist': data[2]}
        table.append(dicRow)
    objFile.close()
```

Figure 9 - read_file Week 6 Code

All of the code between objFile open and objFile close (figure 9) will be replaced. It is no longer necessary because we don't need to read and format the content from the file.

The code is replaced with global variable lstTbl, and the use of the pickle function. See figure 10 (next page). Also, we had to replace 'r' with 'rb' to denote that its reading binary.

```python
@staticmethod
def read_file(file_name, table):
    """Function to manage binary data ingestion from .dat file to a list of dictionaries.

    Reads the data from file identified by file_name into a 2D table.
    (list of dicts) table one line in the file represents one dictionary row in table.

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

    Returns:
        None.
    """
    table.clear()  # this clears existing data and allows to load data from file
    objFile = open(file_name, 'rb')
    global lstTbl
    lstTbl = pickle.load(objFile)
    objFile.close()
```

Figure 10 - read_file Week 7 Code

Similarly, for the write_file code block from week 6 (figure 11), we can delete all of the code between objFile open and objFile close, because formatting is no longer needed.

```python
@staticmethod
def write_file(file_name, table):
    """Function to manage data writing from list of dictionaries to file.

    Writes the data to file identified by file_name in csv format
    (list of dicts) table one line in the table represents one row in file.

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

    Returns:
        None.
    """
    objFile = open(strFileName, 'w')
    for row in lstTbl:
        lstValues = list(row.values())
        lstValues[0] = str(lstValues[0])
        objFile.write(','.join(lstValues) + '\n')
    objFile.close()
    print('file saved')
```

Figure 11 - write_file Week 6 Code

The code is replaced with a single line, using the pickle function to write the data from memory to file in binary. The 'w' was also replaced with 'wb' to denote writing the file in binary.

```python
@staticmethod
def write_file(file_name, table):
    """Function to manage data writing from list of dictionaries to .dat file.

    Writes the data to file identified by file_name in binary format.

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

    Returns:
        None.
    """
    objFile = open(strFileName, 'wb')
    pickle.dump(lstTbl, objFile)
    objFile.close()
    print('file saved')
```

Figure 12 - write_file Week 7 Code

Figure 13 shows the program functioning correctly in Spyder. The error handling for add (‘a’) and delete (‘d) works properly.

```
IPython 7.31.1 -- An enhanced Interactive Python.

In [1]: runfile('/Users/gregeisele/Documents/Python_Class_(IT_FDN_110_B)
Users/gregeisele/Documents/Python_Class_(IT_FDN_110_B)/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 ID: not an integer
What is the CD's title? bla
What is the Artist's name? bla
USER ERROR!
Only numbers can be entered for ID!  Try again!

Enter ID: 7
What is the CD's title? bla bla
What is the Artist's name? bla
======= The Current Inventory: =======
ID  CD Title (by: Artist)

1   Midnights (by:Taylor Swift)
2   A Song for You (by:Luke Evans)
3   Harry's House (by:Harry Styles)
4   The Car (by:Arctic Monkeys)
5   The Highlights (by:Weeknd)
6   Diamonds (by:Elton John)
7   bla bla (by:bla)
======================================
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]: d

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

1   Midnights (by:Taylor Swift)
2   A Song for You (by:Luke Evans)
3   Harry's House (by:Harry Styles)
4   The Car (by:Arctic Monkeys)
5   The Highlights (by:Weeknd)
6   Diamonds (by:Elton John)
======================================
Which ID would you like to delete? integer
Only numbers can be entered for ID!  Try again!

Which ID would you like to delete? 6
The CD was removed
======= The Current Inventory: =======
ID  CD Title (by: Artist)

1   Midnights (by:Taylor Swift)
2   A Song for You (by:Luke Evans)
3   Harry's House (by:Harry Styles)
4   The Car (by:Arctic Monkeys)
5   The Highlights (by:Weeknd)
======================================
```

Figure 13 - Program Running in Spyder

Figure 14 shows the program functioning correctly in Terminal.

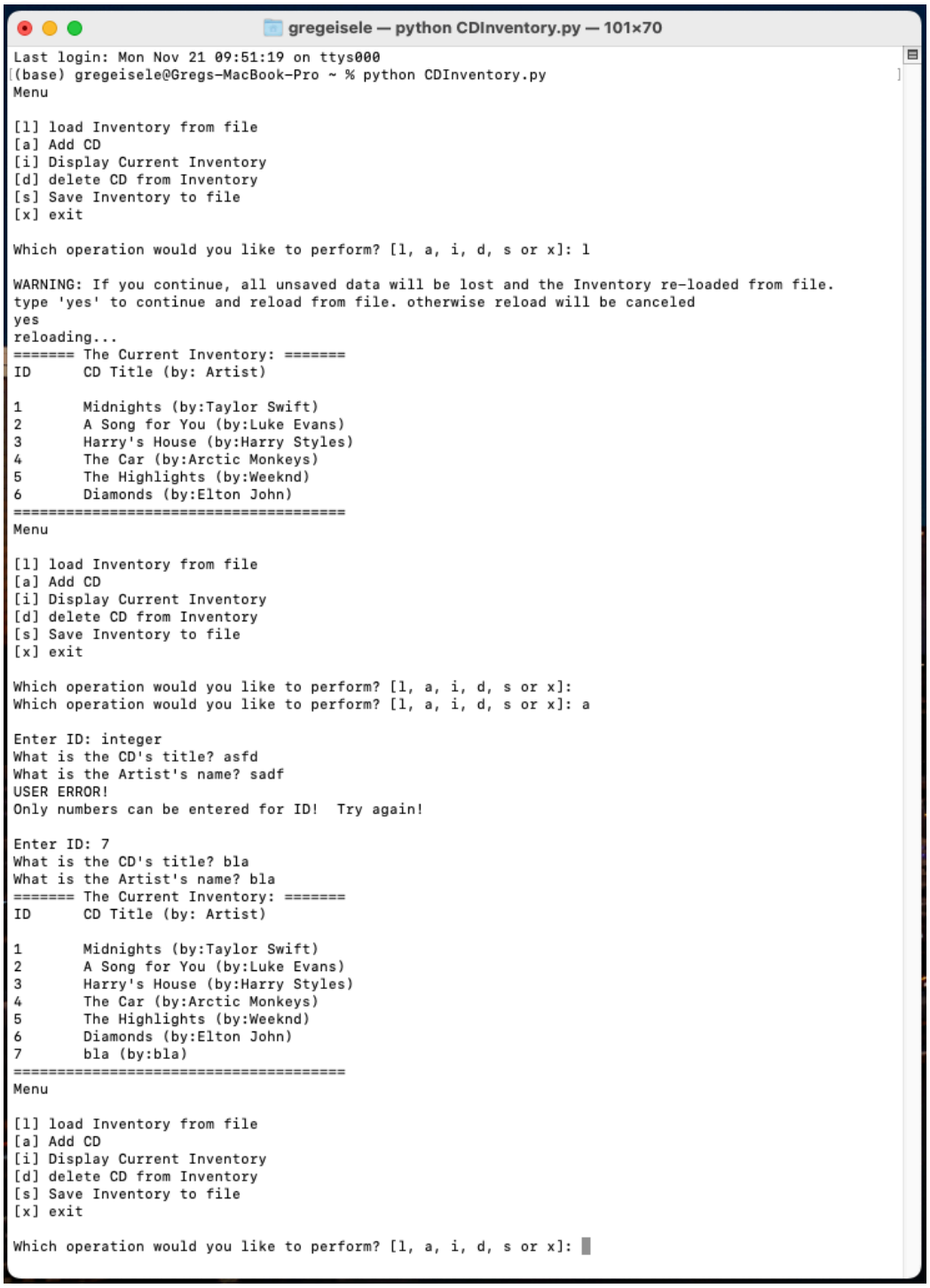

Figure 14 - Program Running in Terminal

Summary

This week's lesson was quite a bit easier for me to grasp compared to the previous few weeks. Error handling seems pretty straightforward to implement, and I appreciate the program continuing to function rather than having to reboot any time an error pops up. I also appreciate the simplicity of binary data storage code, compared to having extra lines for formatting in load-from or save-to text file code blocks.

Git Hub

https://github.com/Greg6874/Assignment_07