PADMAVATHY BHAMIDIPATI

FD IT FDN 110 B Au 22: Foundations Of Programming: Python

11/27/2022

Assignment 07

Files and Exceptions

What we learnt in Module 7

In this module, we learned more about working with ***text files*** and explored ***Binary Files***. We honed the use of *Functions* to organize our code. We learned Structured Error Handling by first of all exploring the Exception class, then we briefly learnt to create custom Exception classes. This latter topic of the learning module was useful to catch exceptions and script a better way for the program to behave in case of unforeseen errors, rather than have the program fail from getting executed! We continued to use *Spyder* as our IDE and stored code in *GitHub*.

A brief of what we learnt in this module is as follows:

- While continuing to work with text files, we went over the read, write and append modes and how they are called in functions. In conjunction with these modes, we learnt to use the **with…as** option.
- The various reading data options such as **readline()** and **readlines()** functions – with **while** and **for** loops, respectively; so, lines of data can be read and stored in an in-memory variable either as a single line of data or as multiple / desired number of lines of data.
- We learnt how to use ***Binary*** *files* and replace text files with these, including 'rb', 'wb', 'ab' as file modes – with 'b' indicating binary format.
- Structured Error Handling: the use of **try…except** code block; using the Exception class, catching specific exceptions and how best to leverage Python's existing exception-handling classes; raising custom errors, and creating custom exception classes.
- We briefly learnt how to create custom github readme files that is better formatted using markup/down language.

Research: Exception Handling and Pickling in Python

While researching the topic, I found the following three websites for information reliability and for the content being simple, easy-to-read and understand, to-the-point and comprehensive! They give brief definitions, syntax, useful examples and sample code

a) https://pythonguides.com/python-exceptions-handling/

b) https://www.geeksforgeeks.org/python-exception-handling/: in the website, there are additional constructs discussed >> Try with Else clause, Finally keyword in Python are useful to know about in the current context of learning. This site has discussion tab that many others from the community have contributed to about the topic, videos etc.
c) https://www.tutorialspoint.com/python/python_exceptions.htm : I liked this website because it seemed to point out the most commonly required (read standard) exceptions that I am likely to come across or want to use at this stage of my programming. Also, the same page teaches about **Assertions** that is also useful to learn / know while trying out exception handling. Nothing about Pickling here but well-covered in the above two sites!

Assignment 07

For assignment 7, we research about Exception handling and pickling, in addition to the contents of the learning module, and modified last week's assignment to add structured error handling wherever there is user interaction, type casting or file access operations. We are to use Python's built-in exception classes. We are also required to modify the permanent data store to use binary data.

Apart from re-testing last week's test cases to ensure that the changes in code did not adversely affect the program's intent, the exceptions were also duly tested. Snips of code execution are listed under the **Code Execution** section below – in both **Spyder** and on the **Terminal**. The code is detailed in the **Appendix** section.

Steps taken:

1. Copied last week's assignment and updated for exception handling and converting code to use binary file instead of text file
2. Exception Handling: since the existing code handled potential exceptions such as "ID does not exist" and unavailable input option for "choice" variable, unless custom exceptions were used, the application of structured error handling was limited for user I/O operations in the code.

Code Execution in Spyder

Code was **tested** for the following:

1. Adding entries to the CD Inventory
2. Saving entries to file
3. Deleting an entry in the list and displaying the edited inventory
4. Attempting to delete an entry that does not exist
5. Loading data
6. Displaying the inventory
7. Exiting the program

Then, the following was further tested for the newly added script:

Upon first execution, if the file did not exist, then the exception was handled - as follows:

```
In [11]: runfile('C:/FP_Python/Mod_07/CDInventory.py', wdir='C:/FP_Python/
File not found.
File not found! Creating a new file.
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: 1

What is the CD's title? Dancing Queen

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

1   Dancing Queen (by:ABBA)
======================================
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]:
```

Figure 1: File not found exception

Input typecast error was tested as follows:

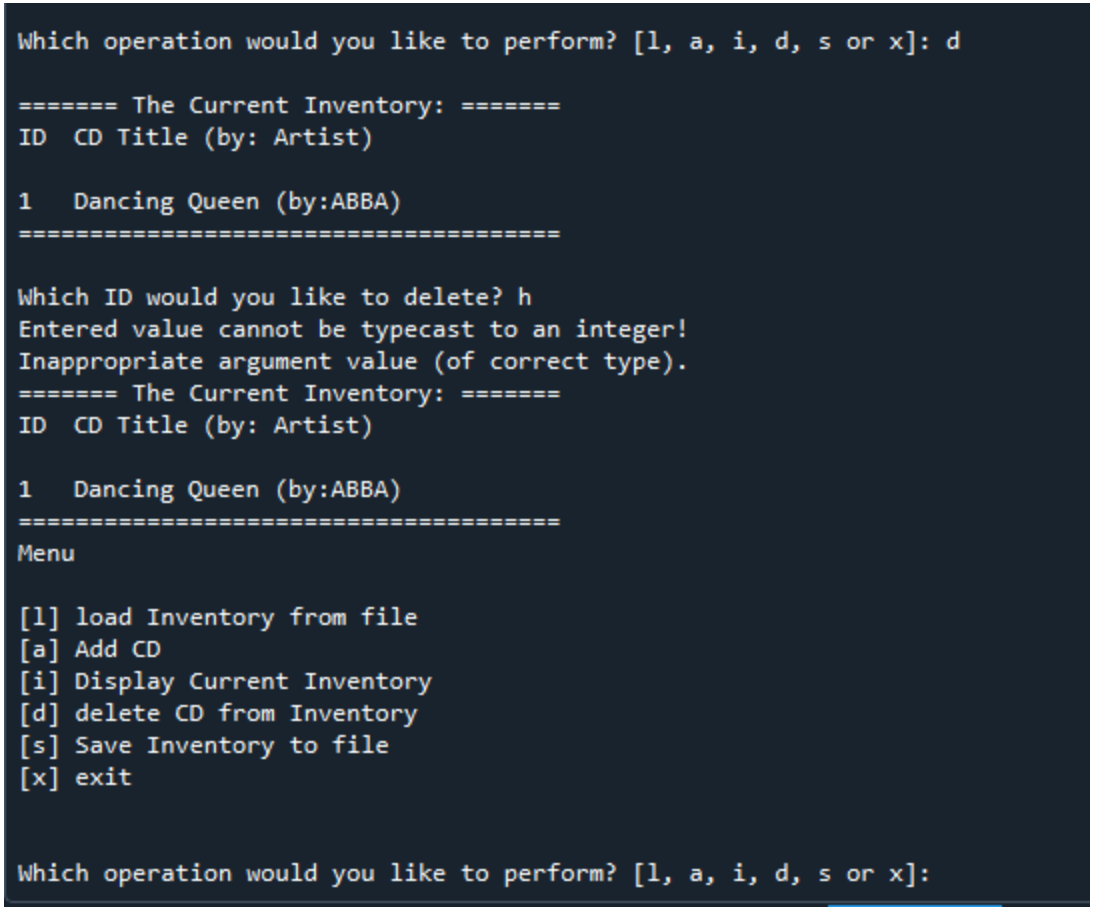

Figure 2: Typecasting error

Adding and saving data:

Figure 3: Add and save data

Loading data:

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

WARNING: If you continue, all unsaved data will be lost and the Inventory re-loaded from file.

Type 'yes' to continue and reload from file. Otherwise, reload will be canceled. yes

Re-loading...

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

1   Dancing Queen (by:ABBA)
2   Brothers in Arms (by:Dire Straits)
======================================
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]:
```

Figure 4: Load data

Execution on Terminal

The following image captures a successful execution of Assignment 07's script on the Terminal.

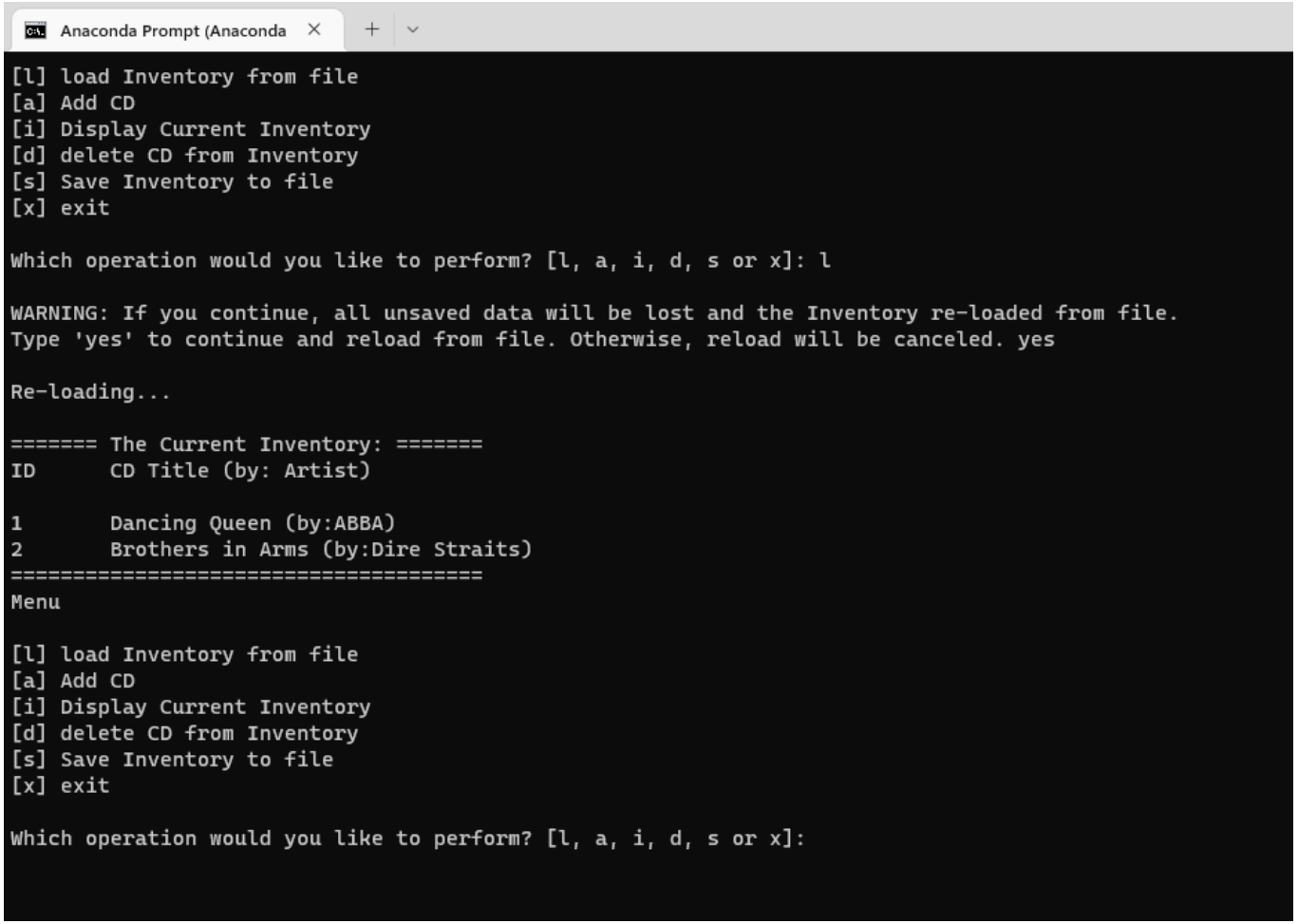

Figure 5: Execution on Terminal

GitHub

The following is the link to my GitHub repository for this assignment:

https://github.com/PadmaBham/Assignment_07

Appendix

1. My **Python script** for the assignment is as follows:

```
#------------------------------------------#
# Title: Assignment06_Starter.py
# Desc: Working with classes and functions.
# Change Log: (Who, When, What)
# DBiesinger, 2030-Jan-01, Created File
# PBhamidipati, 2022-Nov-19, Edited file to address TODOs as part of Assignment 06
# PBhamidipati, 2022-Nov-20, Edited file to add DocStrings, corrected code after testing it and debugging
# PBhamidipati, 2022-Nov-27, Edited file to change text file to binary file, added exceptions for file access
#   and IO operations, and updated corresponding docstrings; changed 'ENTER' to 'any key' in lines #279 and #302
#   to reflect its true functionality
#------------------------------------------#

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


# -- PROCESSING -- #
class DataProcessor:
    """ Manipulating data; contains functions that are typically called from IO functions"""
    # TODONE add functions for processing here

    #   METHOD TO CLEAR IN-MEMORY LIST / VARIABLE
    def list_clear():
        """Clears the in-memory list of lists ahead of new data load and is called only as required

        Args:
            None.

        Returns:
```

```
        None.
    """
    lstTbl.clear() # this clears existing data and allows to load data from file

# METHOD FOR ADD CD
@staticmethod
def data_add(getID, getTitle, getArtist):
    """Adds new entry at the bottom of the CD inventory

    Args:
        getID (string): the unique ID of a CD entry that's received as a string
            but converted to integer within the method
        getTitle (string): the title of the CD being added
        getArtist (string): the name of the artist of the CD

    Returns:
        None.
    """
    intID = int(getID)
    dicRow = {'ID': intID, 'Title': getTitle, 'Artist': getArtist}
    lstTbl.append(dicRow)

# METHOD FOR DELETE
@staticmethod
def delete_row(intIDDel):
    """Deletes an entry from the inventory if the corresponding ID is matched with the specified ID;
    otherwise, due info given to user

    Args:
        intIDDel (integer): this is the CD's ID that the user specifies for the entry's deletion from inventory

    Returns:
        None.
    """
    intRowNr = -1
    blnCDRemoved = False
    for row in lstTbl:
        intRowNr += 1
        if row['ID'] == intIDDel:
            del lstTbl[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 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

        Returns:
            None.

        Raises:
            FileNotFoundError: triggered when file does not exist, most likely during the first execution of program
            IOError: Capture of generic I/O error when reading from file
        """
        table = []
        try:
            objFile = open(file_name, 'rb')
            table = pickle.load(objFile)
        except FileNotFoundError as e:
            print(e.__doc__)
            print('File not found! Creating a new file.')
            objFile = open(file_name, 'wb')
        except IOError as e:
            print(e.__doc__)
            print('Unhandled error while reading file!')
            raise(e)
```

```
        finally:
            objFile.close()

        return table

    @staticmethod
    def write_file(file_name, table):
        """Function that writes data from a list of dictionaries to a file

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

        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.

        Raises:
            IOError: Capture of generic I/O error when writing to file
        """
        # TODONE Add code here
        # METHOD TO SAVE
        try:
            objFile = open(file_name, 'wb')
            '''for row in table:
                lstValues = list(row.values())
                lstValues[0] = str(lstValues[0])
                tempStr += ','.join(lstValues) + '\n' '''
            pickle.dump(table, objFile)
        except IOError as e:
            print(e.__doc__)
            print('Unhandled error while reading file!')
            raise(e)
        finally:
            objFile.close()


# -- PRESENTATION (Input/Output or IO) -- #
```

```
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('=======================================')

# TODONE add I/O functions as needed
# ASK USER FOR INPUT; call DataProcessor.
@staticmethod
def ask_input_addCD():
    """Gets user input for the CD's details that are to be added to the inventory

    Args:
        None.

    Returns:
        None.

    """
    strID = input('Enter ID: ').strip()
    strTitle = input('What is the CD\'s title? ').strip()
    strArtist = input('What is the Artist\'s name? ').strip()

    DataProcessor.data_add(strID, strTitle, strArtist)
    IO.show_inventory(lstTbl)

@staticmethod
def delete_data():
    """Gets CD's ID from user for deletion of the entry from the inventory;
    then calls delete func. from Data Processing section;
    displays the latest inventory after deletion

    Args:
```

```
        None.

    Returns:
        None.

    Raises:
        ValueError: when the value cannot be converted to integer

    """
    # 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
        # TODONE move processing code into function
        DataProcessor.delete_row(intIDDel)
    except ValueError as e:
        print("Entered value cannot be typecast to an integer!")
        print(e.__doc__)
    finally:
        IO.show_inventory(lstTbl)

@staticmethod
def save_data_list():
    """Displays the latest inventory from the in-memory list and confirms that the list is to be saved
        to the file. If confirmed, the list is written to the file, otherwuse it informs that the list
        was NOT saved and asks the user to go back to the menu list

    Args:
        lstTbl (list): is the in-memory list variable that is used to store the list of CDs and their details

    Returns:
        None.

    """
    IO.show_inventory(lstTbl)
    strYesNo = input('Save this inventory to file? [y/n] ').strip().lower()
```

```
        # 3.6.2 Process choice
        if strYesNo == 'y':
            # 3.6.2.1 save data
            # TODONE move processing code into function
            FileProcessor.write_file(strFileName, lstTbl)
        else:
            input('The inventory was NOT saved to file. Press [any key] to return to the menu.')

    @staticmethod
    def load_from_file():
        """Ascertains from the user if the CD inventory is to be loaded from the file or the in-memory list variable,
            displaying suitable warning messages as to what would happen for each of their choices, and proceeds to
            load or cancel loading as may be the user's choice

        Args:
            None.

        Returns:
            None.

        """
        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('\nRe-loading...\n')
            DataProcessor.list_clear()
            lstTbl = FileProcessor.read_file(strFileName)
            IO.show_inventory(lstTbl)
        else:
            input('canceling... Inventory data NOT reloaded. Press [any key] to continue to the menu.\n')
            IO.show_inventory(lstTbl)


# 1. When program starts, read in the currently saved Inventory
DataProcessor.list_clear() # this clears existing data and allows to load data from file
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':
    IO.load_from_file() # call method to load data from file to in-memory list
    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
    # TODONE move IO code into function
    IO.ask_input_addCD()
    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':
    IO.delete_data()
    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.save_data_list()
    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')
```

2. Reference material to learning this module has been all the reading documentation and videos demonstrations that were provided in this course.

3. Search strings: Convert string to binary in Python (delftstack.com)