Shaked Bason

August-25-2020

Foundations of Programming (Python)

Assignment 07

Assignment 07

Instruction

In this assignment, questions such as “What are the benefits of using structured error handling?,” and “How is the Exception class used?” were answered. Throughout the assignment, I learned the difference between a text file and a binary file.

GitHub Link for my Assignment - <https://github.com/shakba/Assignment_07>

Module 7 and Labs – Step by step

Working with Text Files

In the past modules we got familiar with writing data to text files using the open() and close functions which allows us to write, read, append and close. In this module we got to know two additional methodologies. The readline() and the readlines() functions.

**readline() -** when calling the readline() function, the next line of data gets returned. While the access to the file stays open, the location of which line to read next is memorized.

**readlines() -** This function reads all the lines in a file and returns a list.

# Working with Binary Files

Until this point we saved data in text files that are human readable.

Binary files contain data that is saved exactly in the same format as it is saved in computer memory.

This is a practical way to save states of your application or setting the user made.

# 

# 

# Pickling

Saving data to a binary file by preserving data from Python runtime is called “pickling”.

The pickle module takes the information associated with an object and serializes or de-serializes it in a way that it can be easily stored as binary information. pickle.dump() is the method for saving the data out to the designated pickle file. This method preserves data from Python runtime and saves it to a binary file. Loading is done through the pickle.load() method which retrieves data from a binary file and loads it to Python runtime.

Research has been done at - <https://www.pitt.edu/~naraehan/python3/pickling.html>

# Error - Exception Handling

Until this point, every time we coded something wrong, Python crushed. Rather because of small things we missed or bigger. In this module I learned how to handle those errors. This improvement helps to avoid the code from crashing the moment an error is encountered.

The “try” clause, the statement between the “try” and “except” keywords is executed.

If no exception occurs, the except clause is skipped and execution of the try statement is finished.

If an exception occurs during execution of the try clause, the rest of the clause is skipped. Then if its type matches the exception named after the “except” keyword, the except clause is executed, and then execution continues after the try statement.

5.Creating a program - I noted what the code does and added it to this document using the [planet](http://planetb.ca/syntax-highlight-word)-b website.

Finally, I created my code for this assignment (Appendix01), our known and familiar CDInventory. I modify my code to work with some new instructions.

**Appendix #1**

A screenshot of a cell phone

Description automatically generated

Figure - Spyder output for Assignment 1

A screenshot of a cell phone

Description automatically generated

Figure - Spyder output for Assignment 2

A screenshot of a cell phone

Description automatically generated

Figure - Terminal's Output For Assignment

Summary

Module 7 taught me about working with text files, working with binary files, pickling and error exception handling which avoid code from crush. I continued improving my usage of classes and functions from the last modul. I used Spyder as my IDE on this assignment.

Appendix

#1 CDInventory.py

* #------------------------------------------#
* # Title: CDInventory.py
* # Desc: Working with classes and functions.
* # Change Log: Aug 24, MODIFIED MY SCRIPTED TO ASSIGNMENT7
* # shakedbason, 2020-Aug-24, Created File
* #------------------------------------------#
* **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:
* """Adding CD data to the inventory and deleting CD data from inventory"""
* @staticmethod
* **def** addData(strID, strTitle, stArtist, table):
* """Function to add data to the 2D table (list of dictionaries)
* Handles ValueError exception type for negative and non-numeric values
* Args:
* StrID (string)
* Strtitle (string)
* StArtist (string)
* table (list of dict): 2D data structure
* Returns:
* table (list of dict): 2D data structure
* """
* #check if ID srting or int
* **try**:
* intID = int(strID)
* **except** ValueError:
* **print**('\nInvalid input, ID needs to be an integer\n')
* **return**
* # Add item to the table
* dicRow = {'ID': intID, 'Title': strTitle, 'Artist': stArtist}
* table.append(dicRow)
* **return** table
* @staticmethod
* **def** delData(idRemove, table):
* """Function to DELETE existing data from table.
* Accepts the ID to delete and the list table to remove it from
* Args:
* idRemove (int): ID to remove CD data
* table (list of dic): 2D data structure
* Returns:
* None
* """
* # Search thru table and delete CD
* intRowNr = -1
* blnCDRemoved = False
* **for** row **in** table:
* intRowNr += 1
* **if** row['ID'] == idRemove:
* **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, 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.
* """
* # Load data from binary file
* table.clear() # this clears existing data and allows to load data from file
* with open(file\_name, 'rb') as objFile:
* data = pickle.load(objFile)
* table.extend(data)
* @staticmethod
* **def** write\_file(file\_name, table):
* """Function to write data to a binary file
* Writes the data to a binary file identified by file\_name into a 2D table
* Args:
* file\_name(string)
* table(list of dict): 2D data structure
* Returns:
* None
* """
* # Save data to a binary file
* with open(file\_name, 'wb') as objFile:
* pickle.dump(table, objFile)
* # -- PRESENTATION (Input/Output) -- #
* **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.
* """
* # Display current inventory
* **print**('======= The Current Inventory: =======')
* **print**('ID\tCD Title (by: Artist)\n')
* **for** row **in** table:
* **print**('{}\t{} (by:{})'.format(\*row.values()))
* **print**('======================================')
* @staticmethod
* **def** addItem():
* """Function to get user input for ID, title, and artist
* Args:
* None.
* Returns:
* StrID (string)
* Strtitle (string)
* StArtist (string)
* """
* # Ask user for new ID, CD Title and Artist
* strID = input('Enter ID: ').strip()
* strTitle = input('What is the CD\'s title? ').strip()
* stArtist = input('What is the Artist\'s name? ').strip()
* **return** strID, strTitle, stArtist
* # When program starts, read in the currently saved Inventory
* #handle error with FileNotFoundError exception
* **try**:
* FileProcessor.read\_file(strFileName, lstTbl)
* **except** FileNotFoundError:
* FileProcessor.write\_file(strFileName, lstTbl)
* # Start main loop
* **while** True:
* # Display Menu to user and get choice
* IO.print\_menu()
* strChoice = IO.menu\_choice()
* # Process menu selection
* # Process exit first
* **if** strChoice == 'x':
* **break**
* # 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...')
* FileProcessor.read\_file(strFileName, lstTbl)
* IO.show\_inventory(lstTbl) # Display Inventory to user
* **else**:
* input('canceling... Inventory data NOT reloaded. Press [ENTER] to continue to the menu.')
* IO.show\_inventory(lstTbl)
* **continue** # start loop back at top.
* # Process add a CD
* **elif** strChoice == 'a':
* # Store user inputs
* userInputId, userInputTitle, userInputArtist = IO.addItem()
* # Add data to the 2D table (list of dictionaries)
* DataProcessor.addData(userInputId, userInputTitle, userInputArtist, lstTbl)
* IO.show\_inventory(lstTbl)
* **continue** # start loop back at top.
* # Process display current inventory
* **elif** strChoice == 'i':
* IO.show\_inventory(lstTbl)
* **continue** # start loop back at top.
* # Process delete a CD
* **elif** strChoice == 'd':
* IO.show\_inventory(lstTbl)
* # Ask user to remove ID
* **try**:
* intIDDel = int(input('Which ID would you like to delete? ').strip())
* **except** ValueError:
* **print**('Oops! That was not a number.')
* **print**()
* **continue**
* DataProcessor.delData(intIDDel, lstTbl) # Deletes data from inventory
* IO.show\_inventory(lstTbl)
* **continue** # start loop back at top.
* # Process save inventory to file
* **elif** strChoice == 's':
* # 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()
* # Process choice
* **if** strYesNo == 'y':
* FileProcessor.write\_file(strFileName, lstTbl) # Calling write\_file function
* **print**('Data saved to file.')
* **print**()
* **else**:
* input('The inventory was NOT saved to file. Press [ENTER] to return to the menu.')
* **continue** # start loop back at top.
* # Catch-all should not be possible, as user choice gets vetted in IO, but to be save:
* **else**:
* **print**('General Error')