Name: Fabio Sacca

Date: Nov-27, 2022

Course: Foundations of Programming (Python)

Title: Assignment 07

# Assignment 07

## Introduction:

In this document, I will describe the steps I took to modify the ‘CD Inventory’ program from Assignment 06 in order to: address areas of improvement from last week, modify the permanent data store to use binary data, and add structured error handling where needed. I will then show the script running in Spyder and Terminal Mac window.

## Topic 1: Improving program

In this section, I describe how I improved my ‘CD Inventory’ program by addressing the opportunities identified in last week’s assignment:

* Need separate out IO functions from Data Processing functions (e.g., In week 6, my program lumped get\_CD\_input IO function with add\_CD Data Processing function)
* Need to use local and global variables as intended (e.g., in week 6, some parts of my program used global variables within functions).

**NOTE**: if you prefer to skip to the next sections covering the requirements of Assignment 07, you can go to Page 4 or [click here](#_Topic_2:_Modify).

### 1.1. Solutions to improve my program – part 1

To address the first area of opportunity, I created distinct functions for collecting user input and processing data:

* IO.get\_user\_input()

**class** **IO**:

*"""Handling Input / Output"""*

@staticmethod

**def** get\_user\_input():

*""" Function to get the user input for adding a CD entry*

*The entry will be returned to be used by a DataProcessor function that will Add it to inventory.*

*Args:*

*None*

*Returns:*

*intID (int): User supplied ID for entry*

*strTitle (string): Title of CD*

*stArtist (string): Name of artist*

*"""*

*#TODone: Add structured error handling for type casting (string to int)*

strID = input('**\n**Enter ID: ').strip()

strTitle = input('What is the CD**\'**s title? ').strip()

strArtist = input('What is the Artist**\'**s name? ').strip()

intID = int(strID)

**return** intID, strTitle, strArtist

* IO.del\_CD\_input()

**class** **IO**:

*"""Handling Input / Output"""*

**def** del\_cd\_input():

*"""Function to get the user input for deleting a CD*

*The entry will be returned to be used by a DataProcessor function that will Remove the chosen entry.*

*Args:*

*None*

*Returns:*

*ID (int): ID to be deleted*

*"""*

*#TODone: Add structured error handling for user interaction*

intIDDel = int(input('**\n**Which ID would you like to delete? ').strip())

**return** intIDDel

* DataProcessor.add\_cd(ID, album, artist, table, row)

**class** **DataProcessor**:

*"""Processing the data held in memory during runtime"""*

@staticmethod

**def** add\_cd(ID, album, artist, table, row):

*"""Function to manage data ingestion from user input to a list of dictionaries*

*Adds data from user entry into a 2D table (list of dicts) in memory during runtime.*

*Args:*

*data (list): values entered by user for ID, CD Title, Artist Name*

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

*Returns:*

*Confirmation message the CD was added to inventory.*

*"""*

row = {'ID': ID, 'Title': album, 'Artist': artist}

table.append(row)

print('The CD was added to Inventory')

* DataProcessor.load\_inventory (file\_name, table)

**def** load\_inventory(file\_name, table):

*"""Function to process user request to load inventory from file*

*Confirms user choice before loading inventory data from runtime and deletes all entries in memory*

*Args:*

*file\_name (string): name of file used to write the data from*

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

*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('reloading...')

FileProcessor.read\_file(file\_name, table)

IO.show\_inventory(table)

**else**:

input('canceling... Inventory data NOT reloaded. Press [ENTER] to continue to the menu.')

IO.show\_inventory(table)

* DataProcessor.save\_inventory (file\_name, table)

**def** save\_inventory(file\_name, table):

*"""Function to process user request to save inventory from file*

*Confirms user choice to daves Inventory data from runtime to permanent memory*

*Args:*

*file\_name (string): name of file used to write the data from*

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

*Returns:*

*None*

*"""*

*# 3.6.2 Process choice*

**while** **True**:

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

**if** strYesNo == 'y':

*# 3.6.2.1 save data*

FileProcessor.write\_file(file\_name, table)

**break**

**elif** strYesNo == 'n':

input('The inventory was NOT saved to file. Press [ENTER] to return to the menu.')

**break**

**else**:

print('Incorrect choice!! Please try again.**\n**')

**continue**

### 1.2. Solutions to improve my program – part 2

To address the second area of opportunity, I looked across all functions where I have used global variables and replaced with local variables as needed. I then tested the program again in Spyder and verified that all program functionalities worked and moved to the next part of my assignment.

## Topic 2: Modify the permanent data store to use binary data

The program uses a strFileName variable, which is assigned to text file ‘CDInventory.txt’ for permanent data store. In order to modify it to use binary data, I had to change the variable value (‘CDInventory.dat’) and modify accordingly the functions in FileProcessor class (read\_file, write\_file) to support reading/writing binary data.

### 2.1. Read File

**class** **FileProcessor**:

*"""Processing the data to and from text file"""*

@staticmethod

**def** read\_file(file\_name, table):

**with** open(file\_name, 'rb') **as** fileObj:

table.clear()

data = pickle.load(fileObj)

**for** i **in** range(len(data)):

table.append(data[i])

**return** table

*Note: I removed function information for the purpose of displaying just the code changes.*

### 2.2. Write File

@staticmethod

**def** write\_file(fileName, table):

**with** open(fileName, 'wb') **as** fileObj:

pickle.dump(table, fileObj)

*Note: I removed function information for the purpose of displaying just the code changes.*

## Topic 3: Adding Structured Error Handling

The requirement is to add structured error handling around the areas where there is user interaction, type casting (string to int) or file access operations.

### 3.1. Handling Errors due to user interaction

The CDInventory.py program uses various methods (e.g, IF statements, WHILE loops) to check that the user is entering an acceptable command/value, which prevents the runtime from raising exceptions which may interrupt the program. For example:

* IO.menu\_choice: uses a WHILE loop to check for any user input that is NOT among the accepted values printing a custom-error message as needed, and continuing the loop until one of the accepted values is entered.
* DataProcessor.load\_inventory: uses a IF statement to check for any user input that is NOT ‘yes’, and prints a custom-error message as needed.
* DataProcessor.save\_inventory: uses a WHILE loop and a IF statement to check whether the user input that is equal to ‘y’ or ‘n’ and prints, a custom-error message as needed, continuing the loop until one of the accepted values is entered.

After testing the program for potential errors, I identified two instances in which user interaction may result in an Exception. In IO.get\_user\_input, and IO.del\_CD\_input functions, a user may enter a value that cannot be converted to a different type (int). In this case, the program raises an exception ‘ValueError’ as the user casts a correct type of value (string) but this cannot be converted to integer. Here is an example of how I handled the error:

**def** get\_user\_input():

*"""Function to collect user input*

*Collects user input*

*Args:*

*None*

*Returns:*

*intID (int): User supplied ID for entry*

*strTitle (string): Title of CD*

*stArtist (string): Name of artist*

*"""*

*#TODone: Add structured error handling for type casting (string to int)*

**while** **True**:

strID = input('**\n**Enter ID: ').strip()

**try**:

intID = int(strID)

**break**

**except** **ValueError** **as** e:

print('**\n**That is not a valid ID number. Please try again.')

print('**\n**Build in error info:', type(e), e, e.\_\_doc\_\_, sep='**\n\t**')

**except** **Exception** **as** e:

print('**\n**There was a general error!')

print('**\n**Build in error info:', type(e), e, e.\_\_doc\_\_, sep='**\n\t**')

strTitle = input('What is the CD**\'**s title? ').strip()

strArtist = input('What is the Artist**\'**s name? ').strip()

**return** intID, strTitle, strArtist

### 3.2. Handling Errors due to type casting

No such exceptions are raised in the program.

### 3.3. Handling Errors due to file access operations.

The only scenarios where a file access error may occur is in the event a file ‘CDInventory.dat’ is not available at the start of the program (which auto-loads the file), or when user chooses to load inventory from file (which both call the FileProcessor.read\_file function. This will raise the exception ‘FileNotFoundError’, which I handled as follows:

**def** read\_file(file\_name, table):

**try**:

**with** open(file\_name, 'rb') **as** fileObj:

table.clear()

data = pickle.load(fileObj)

**for** i **in** range(len(data)):

table.append(data[i])

**return** table

**except** **FileNotFoundError** **as** e:

print('There is no file to open')

print('**\n**Build in error info:', type(e), e, e.\_\_doc\_\_, sep='**\n\t**')

**except** **Exception** **as** e:

print('**\n**There was a general error!')

print('**\n**Build in error info:', type(e), e, e.\_\_doc\_\_, sep='**\n\t**')

*Note: Once again, I removed function information for the purpose of displaying just the code changes.*

The same exception will not be raised in my program when user chooses to save Inventory to file and the file does not exist. This because he FileProcessor.write\_file uses ‘wb’ (write) access mode, which creates a file with filename ‘CDInventory.dat’ even when there is no one.

## Topic 4: Running my script

I saved the script as 'CDInventory.py' in a folder called 'Assignment07' within the course folder '\_FDProgramming', and ran the script in Spyder. The image below shows specifically the error handling responses working on the program.

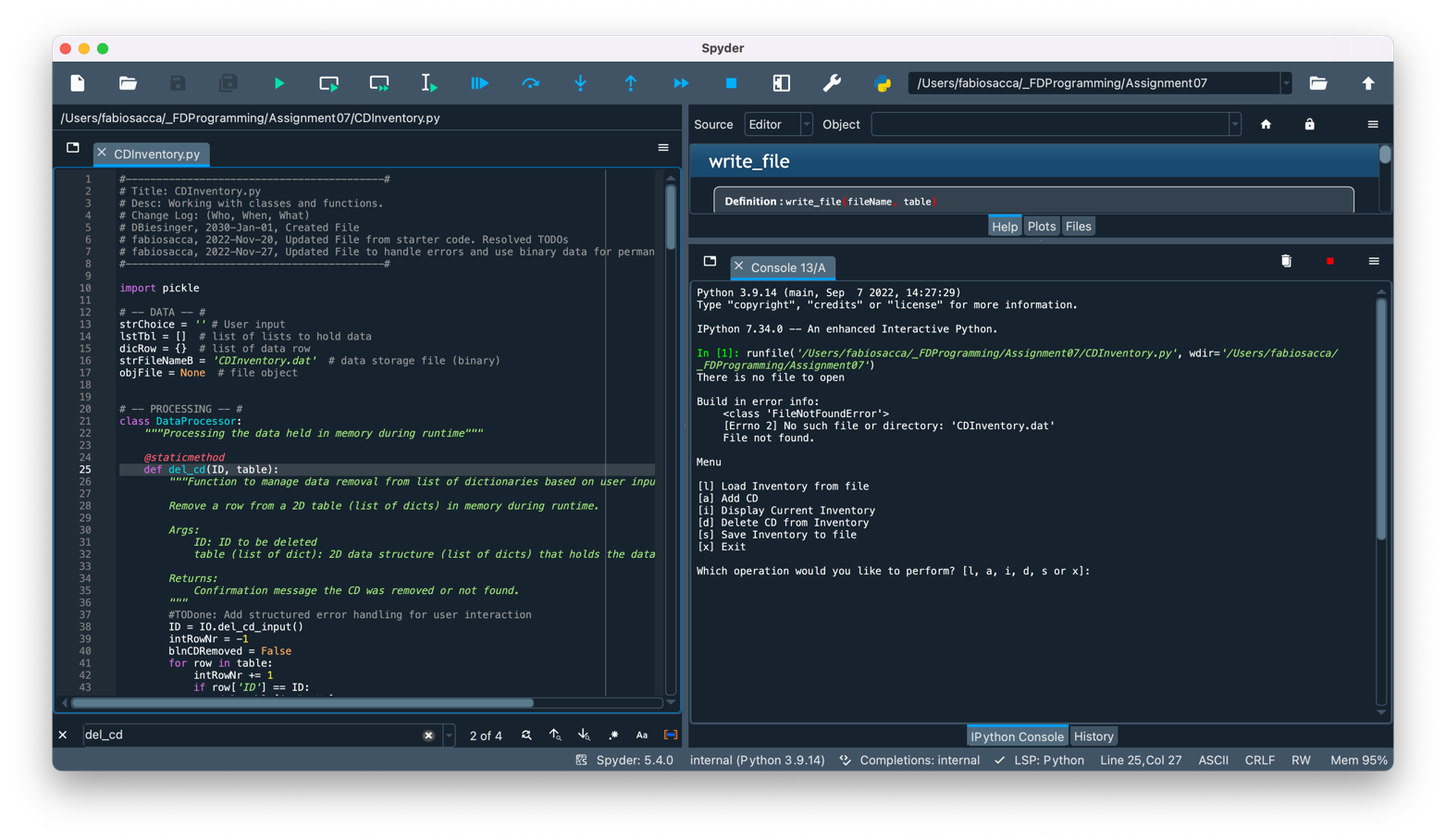


Figure 1 - Screen capture of Spyder returning CDInventory.py script (Exception: FileNotFoundError)

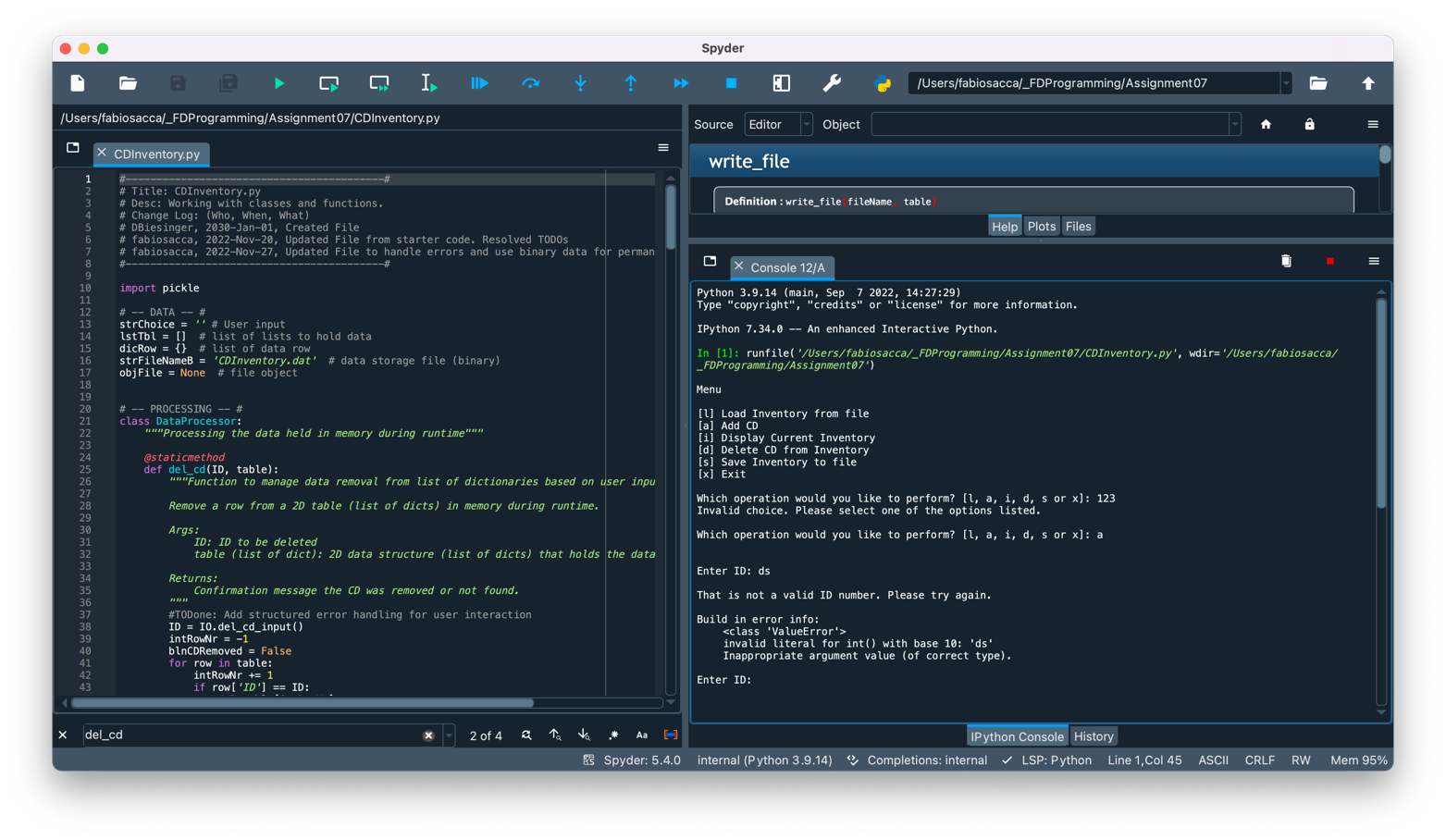


Figure 2 - Screen capture of Spyder returning CDInventory.py script (Custom error handling and ValueError Exception)

A screenshot of a computer

Description automatically generated

Figure 3 - Screen capture of Spyder returning CDInventory.py script (ValueError Exception)

I repeated this step in a terminal window. The image below shows that the script works in my computer.

Graphical user interface, text

Description automatically generated

Figure 4 - Screen capture of OS Terminal returning CDInventory.py script (FileNotFound and ValueError Exceptions)

## Graphical user interface, text, application Description automatically generated

Figure 5 - Screen capture of OS Terminal returning CDInventory.py script (Custom error handling and ValueError Exception)

## Topic 5: Verifying that my script worked

I located the file ‘CDInventory.dat and opened it in a text editor. The image below shows that the data I entered has been written to the file

Text

Description automatically generated with low confidence

Figure 6 - Screen capture of CDInventory.dat displaying data entered.

## Topic 6: Submitting my work on GitHub

Both the CDInventory.py and the present knowledge document are posted on the following public repository

## <https://github.com/fabiosacca/Assignment_07>

## Summary

In this assignment, I covered the steps needed to:

* Modify ‘CD Inventory’ program from last week and address areas of opportunity.
* Modify the permanent data store to use binary data.
* Add structured error handling using python built-in Exceptions.
* Run the script in Spyder and a terminal window.
* Verify that the script in fact wrote in CDInventory.txt the data I entered in the correct format.
* Submit my work on a public GitHub repository.

# Appendix A

## Code of CDInventory.py script presented with [**Syntax Highlighter**](https://saravjishut.org/syntax)**[[1]](#footnote-1)**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120  121  122  123  124  125  126  127  128  129  130  131  132  133  134  135  136  137  138  139  140  141  142  143  144  145  146  147  148  149  150  151  152  153  154  155  156  157  158  159  160  161  162  163  164  165  166  167  168  169  170  171  172  173  174  175  176  177  178  179  180  181  182  183  184  185  186  187  188  189  190  191  192  193  194  195  196  197  198  199  200  201  202  203  204  205  206  207  208  209  210  211  212  213  214  215  216  217  218  219  220  221  222  223  224  225  226  227  228  229  230  231  232  233  234  235  236  237  238  239  240  241  242  243  244  245  246  247  248  249  250  251  252  253  254  255  256  257  258  259  260  261  262  263  264  265  266  267  268  269  270  271  272  273  274  275  276  277  278  279  280  281  282  283  284  285  286  287  288  289  290  291  292  293  294  295  296  297  298  299  300  301  302  303  304  305  306  307  308  309  310  311  312  313  314  315  316  317  318  319  320  321  322  323  324  325  326  327  328  329  330  331  332  333  334  335  336  337  338  339  340  341  342  343  344  345 | *#------------------------------------------#*  *# Title: CDInventory.py*  *# Desc: Working with classes and functions.*  *# Change Log: (Who, When, What)*  *# DBiesinger, 2030-Jan-01, Created File*  *# fabiosacca, 2022-Nov-20, Updated File from starter code. Resolved TODOs*  *# fabiosacca, 2022-Nov-27, Updated File to handle errors and use binary data for permanent storage*  *#------------------------------------------#*  **import** **pickle**  *# -- DATA -- #*  strChoice = '' *# User input*  lstTbl = [] *# list of lists to hold data*  dicRow = {} *# list of data row*  strFileNameB = 'CDInventory.dat' *# data storage file (binary)*  objFile = **None** *# file object*  *# -- PROCESSING -- #*  **class** **DataProcessor**:  *"""Processing the data held in memory during runtime"""*    @staticmethod  **def** del\_cd(ID, table):  *"""Function to manage data removal from list of dictionaries based on user input*  *Remove a row from a 2D table (list of dicts) in memory during runtime.*  *Args:*  *ID: ID to be deleted*  *table (list of dict): 2D data structure (list of dicts) that holds the data during runtime*  *Returns:*  *Confirmation message the CD was removed or not found.*  *"""*  *#TODone: Add structured error handling for user interaction*  ID = IO.del\_cd\_input()  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 in inventory. Try again!')    **def** add\_cd(ID, album, artist, table, row):  *"""Function to manage data ingestion from user input to a list of dictionaries*  *Adds data from user entry into a 2D table (list of dicts) in memory during runtime.*  *Args:*  *data (list): values entered by user for ID, CD Title, Artist Name*  *table (list of dict): 2D data structure (list of dicts) that holds the data during runtime*  *Returns:*  *Confirmation message the CD was added to inventory.*  *"""*  row = {'ID': ID, 'Title': album, 'Artist': artist}  table.append(row)  print('The CD was added to Inventory')    **def** load\_inventory(file\_name, table):    *"""Function to process user request to load inventory from file*  *Confirms user choice before loading inventory data from runtime and deletes all entries in memory*  *Args:*  *file\_name (string): name of file used to write the data from*  *table (list of dict): 2D data structure (list of dicts) that holds the data during runtime*  *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('reloading...')  FileProcessor.read\_file(file\_name, table)  IO.show\_inventory(table)  **else**:  input('canceling... Inventory data NOT reloaded. Press [ENTER] to continue to the menu.')  IO.show\_inventory(table)      **def** save\_inventory(file\_name, table):    *"""Function to process user request to save inventory from file*  *Confirms user choice to daves Inventory data from runtime to permanent memory*  *Args:*  *file\_name (string): name of file used to write the data from*  *table (list of dict): 2D data structure (list of dicts) that holds the data during runtime*  *Returns:*  *None*  *"""*  *# 3.6.2 Process choice*  **while** **True**:  strYesNo = input('Save this inventory to file? [y/n] ').strip().lower()  **if** strYesNo == 'y':  *# 3.6.2.1 save data*  FileProcessor.write\_file(file\_name, table)  **break**  **elif** strYesNo == 'n':  input('The inventory was NOT saved to file. Press [ENTER] to return to the menu.')  **break**  **else**:  print('Incorrect choice!! Please try again.**\n**')  **continue**    **class** **FileProcessor**:  *"""Processing the data to and from text file"""*    @staticmethod  *# TODone: Modify the permanent data store to use binary data.*  *# TODone: Add structured error handling for file access operations.*  **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*  *Args:*  *fileName (string): name of file used to read the data from*    *Returns:*  *table (list): results from file fileName*  *"""*  **try**:  **with** open(file\_name, 'rb') **as** fileObj:  table.clear()  data = pickle.load(fileObj)  **for** i **in** range(len(data)):  table.append(data[i])  **return** table  **except** **FileNotFoundError** **as** e:  print('There is no file to open')  print('**\n**Build in error info:', type(e), e, e.\_\_doc\_\_, sep='**\n\t**')  **except** **Exception** **as** e:  print('**\n**There was a general error!')  print('**\n**Build in error info:', type(e), e, e.\_\_doc\_\_, sep='**\n\t**')    @staticmethod  **def** write\_file(fileName, table):  *# TODone: Modify the permanent data store to use binary data.*  *# ToDone: Add structured error handling for file access operations.*  *"""Function to manage data storage from a list of dictionaries to a binary file*  *Saves the data to file identified by file\_name from a 2D table*  *(list of lists).*  *Args:*  *file\_name (string): name of file used to write the data from*  *table (list of dict): 2D data structure (list of dicts) that holds the data during runtime*  *Returns:*  *None.*  *"""*  **try**:  **with** open(fileName, 'wb') **as** fileObj:  pickle.dump(table, fileObj)  **except** **Exception** **as** e:  print('**\n**There was a general error!')  print('**\n**Build in error info:', type(e), e, e.\_\_doc\_\_, sep='**\n\t**')  *# Structured error handling for FileNotFound not needed in this case as saving creates a new file.*  *# -- 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('**\n**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 = input('Which operation would you like to perform? [l, a, i, d, s or x]: ').lower().strip()  **while** choice **not** **in** ['l', 'a', 'i', 'd', 's', 'x']:  print('Invalid choice. Please select one of the options listed.**\n**')  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** get\_user\_input():  *""" Function to get the user input for adding a CD entry*  *The entry will be returned to be used by a DataProcessor function that will Add it to inventory.*  *Args:*  *None*    *Returns:*  *intID (int): User supplied ID for entry*  *strTitle (string): Title of CD*  *stArtist (string): Name of artist*  *"""*  *#TODone: Add structured error handling for type casting (string to int)*  **while** **True**:  strID = input('**\n**Enter ID: ').strip()  **try**:  intID = int(strID)  **break**  **except** **ValueError** **as** e:  print('**\n**That is not a valid ID number. Please try again.')  print('**\n**Build in error info:', type(e), e, e.\_\_doc\_\_, sep='**\n\t**')  **except** **Exception** **as** e:  print('**\n**There was a general error!')  print('**\n**Build in error info:', type(e), e, e.\_\_doc\_\_, sep='**\n\t**')  strTitle = input('What is the CD**\'**s title? ').strip()  strArtist = input('What is the Artist**\'**s name? ').strip()  **return** intID, strTitle, strArtist    @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('**\n**======= The Current Inventory: =======')  print('ID**\t**CD Title (by: Artist)**\n**')  **for** row **in** table:  print('**{}\t{}** (by:**{}**)'.format(\*row.values()))  print('======================================')  **def** del\_cd\_input():  *"""Function to get the user input for deleting a CD*  *The entry will be returned to be used by a DataProcessor function that will Remove the chosen entry.*  *Args:*  *None*    *Returns:*  *ID (int): ID to be deleted*  *"""*  *#TODone: Add structured error handling for user interaction*  **while** **True**:  **try**:  intIDDel = int(input('**\n**Which ID would you like to delete? ').strip())  **return** intIDDel  **break**  **except** **ValueError** **as** e:  print('**\n**That is not a valid ID number. Please try again.')  print('**\n**Build in error info:', type(e), e, e.\_\_doc\_\_, sep='**\n\t**')  **except** **Exception** **as** e:  print('**\n**There was a general error!')  print('**\n**Build in error info:', type(e), e, e.\_\_doc\_\_, sep='**\n\t**')  *# 1. When program starts, read in the currently saved Inventory*  FileProcessor.read\_file(strFileNameB, lstTbl)  *# 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':  DataProcessor.load\_inventory(strFileNameB, 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*  strID, strTitle, stArtist = IO.get\_user\_input()  *# 3.3.2 Add item to the table*  DataProcessor.add\_cd(strID, strTitle, stArtist, lstTbl, dicRow)  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*  *# 3.5.2 search thru table and delete CD*  DataProcessor.del\_cd(dicRow, lstTbl)  IO.show\_inventory(lstTbl)  **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)  DataProcessor.save\_inventory(strFileNameB, lstTbl)  **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('Invalid choice. Please select one of the options listed.')  **continue** *# start loop back at top.* |

1. Last viewed: Nov-27, 2022 (11:04am PT) [↑](#footnote-ref-1)