< Jayr Padilla >

< 11-27-22 >

< Foundations of Programming; Python >

< Assignment 07>

Working with error handling and binary data

# Introduction

The purpose of this assignment was to modify last week’s assignment. The modifications included changing the permanent data store to use binary data and add structured handling around areas such as user interaction, type casting or file access operations. This was to be completed with the information learned in module 7 such as pickling and error handling.

# Binary data (Pickling)

**Understanding how to work with binary files was important for this assignment. It also is a great tool for many reasons and led to less lines of code. With plain text files it was needed to format the data in a way that we can save it and then retrieve it back. This causes additional steps and more processing time. Working with binary data we can ignore the need to have readable format in a text file. Because of this, the information can be saved into its memory as binary code. As mentioned before, not having to worry about specific formatting to and from a text file allowed less lines of code. This can be seen when comparing Listing 1 to Listing 2. In Listing 1 we are using pickling to load any information from the CDInventory.dat file and assigning and returning it as a variable. Listing 2 shows the formatting that was needed when using text files to save the data. A website that helped me further understand pickling was the python.org site which can be found here:** [Pickle Reference](https://docs.python.org/3/library/pickle.html)**. With this website it did a great job of explaining the module interface of pickling going through the various functions that can be used and showing all the arguments that are available. Additionally, it explained the different types of errors that may come when using pickling such as PickleError, PicklingError, and UnpicklingError.**

Text

Description automatically generated

Listing - read\_file function using binary data (.dat files)

Text

Description automatically generated

Listing - read\_file function using .txt files

# Structured Error Handling

**Learning about structured error handling was interesting and easy to learn in my opinion as it was more of learning where does this “puzzle piece” fit in with everything else we have learned. It fit perfectly into this module since we have already learned about the different types of errors that have terminated our program. These errors included errors such as ValueError and FileNotFound. These were the two main errors that were added to the error handling within my script. The key takeaway of using structured error handling is that using this allows us to protect the parts of our programs that have a possibility of an error occurring. It does seem that this is something that should always be in a script moving forward. Especially if you have having user and file interaction. Looking at Listing 3, error handling is added for the option to delete an entry. This is needed as the user might not enter an integer for the ID they would like to delete. If this is the case the error class Exception creates an object e and spits back information on the type of error. This can be seen being executed and statements being printed in Figure 1. Another place error handling was included was at the beginning of the main code the file was needed to be read. If the file did not already exist an error was printed telling the user and then went onto create the .dat file. This can be seen in Figure 2. A website that explained structured handling well can be found here:** [Exception Handling Python](https://www.geeksforgeeks.org/python-exception-handling/)**. It went on to explain things we covered in this module with different examples which included catching specific exceptions and errors and going over try and except formatting and features. I also learned from looking at this website there is a try-except-else feature that can be used. The code can enter the else block if the try block does not raise an exception.**

Text

Description automatically generated

Listing -error handling for deleting entry

Graphical user interface, text

Description automatically generated

Figure 1 - Deleting entry error handling (Spyder)

Text

Description automatically generated

Figure 2 - Read file error handling (Terminal)

Using [Saravijust Syntax](https://saravjishut.org/syntax) (external reference)[[1]](#footnote-1) web page

# Summary

In this assignment I learned various to help me build this script. The main part of this module focused on how to properly use error handling. This included using the exception class and exception objects to print out arguments and details about the error. Additionally looking at the child classes of the exception class were used. Pickling and using binary data structures allowed data to save information in memory for easier accessing and less additional steps needed for formatting. Looking at Figure 1 and Figure 2, you can see parts of the script (deleting entry with error handling, reading file with error handling) being executed in Spyder and in Terminal, respectively.

# Appendix

Web link to GitHub Dipository: <https://github.com/jayrpad/assignment07>

## Listing Demo\_function.py

|  |  |
| --- | --- |
| 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 | *#------------------------------------------#*  *# Title: CDInventory.py*  *# Desc: Working with error handling and binary data.*  *# Change Log: (Who, When, What)*  *# DBiesinger, 2030-Jan-01, Created File*  *#JPadilla, 2022-Nov-20, Moved appropiate functions into classes*  *#JPadilla, 2022-Nov-20, Created add\_data function under DataProcessor class w/ DocString*  *#JPadilla, 2022-Nov-20, Created delete\_data function under DataProcessor class w/ DocString*  *#JPadilla, 2022-Nov-20, Added function calls to the main code (Lines 251-253, 255, 270)*  *#JPadilla, 2022-Nov-27, changed description title*  *#JPadilla, 2022-Nov-27, combined all input functions into one function (get\_inputs())*  *#JPadilla, 2022-Nov-27, changed read\_data, and write\_data to binary files using pickle syntax*  *#JPadilla, 2022-Nov-27, added error handling to functions and main body of script*  *#------------------------------------------#*  **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*  intID = '' *#user input ID number*  strTitle = '' *#user input CD title*  stArtist = '' *#user input artist name*  *# -- PROCESSING -- #*  **class** **DataProcessor**:  *"""Function to process/ save / delete any data entry made by user"""*  @staticmethod  **def** add\_data(ID, title, artist):  *"""Function that adds user input of ID, CD Title, Artist Name to a dictionary and then adds to table*  *Args:*  *ID: identification number for entry*  *Title: user inputted CD title*  *Artist: user inputted artist name*  *Returns:*  *None*  *"""*  **try**:  intID = int(strID)  dicRow = {'ID': intID, 'Title': title, 'Artist': artist}  lstTbl.append(dicRow)  **except** **ValueError** **as** e:  print('The ID entered is NOT an integer!. **\n**Entry not saved - Please enter ID as an integer! **\n**')  print(type(e), e, e.\_\_doc\_\_, sep = '**\n**')  @staticmethod  **def** delete\_data(ID):  *"""Function that finds desired entry to delete based on ID number*  *Args:*  *ID: identification number for entry*  *Returns:*  *None*  *"""*  intRowNr = -1  blnCDRemoved = **False**  **for** row **in** lstTbl:  intRowNr += 1  **if** row['ID'] == ID:  **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, 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:*  *table: the table of data (ID, CD Title, Artist name) pulled from the file*  *"""*  table.clear() *# this clears existing data and allows to load data from file*  **with** open(file\_name, 'rb') **as** objFile:  table = pickle.load(objFile)  **return** table  @staticmethod  **def** write\_file(file\_name, table):  *"""Function that writes string data to file*  *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*  *"""*  **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.*  *"""*  print('======= The Current Inventory: =======')  print('ID**\t**CD Title (by: Artist)**\n**')  **for** row **in** table:  print('**{}\t{}** (by:**{}**)'.format(\*row.values()))  print('======================================')  @staticmethod  **def** user\_inputs():  *"""Function that prompts user for ID number for entry, CD title, and Artist*  *Args:*  *None.*  *Returns:*  *strID: string ID entered by user*  *strTitle: string CD title entered by user*  *stArtist: string Artist name entered by user*  *"""*  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  *# 1. When program starts, read in the currently saved Inventory*  **try**:  lstTbl = FileProcessor.read\_file(strFileName, lstTbl)  **except** **FileNotFoundError** **as** e:  print('File not found! File has now been created.')  print(type(e), e, e.\_\_doc\_\_, sep = '**\n**')  FileProcessor.write\_file(strFileName, 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':  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, lstTbl)  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*  strID, strTitle, stArtist = IO.user\_inputs()  *# 3.3.2 Add item to the table*  DataProcessor.add\_data(strID, strTitle, stArtist)  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**:  *# 3.5.2 search thru table and delete CD*  intIDDel = int(input('Which ID would you like to delete? ').strip())  DataProcessor.delete\_data(intIDDel)  IO.show\_inventory(lstTbl)  **except** **ValueError** **as** e:  print('ID entered is not an integer. Please try again.')  print(type(e), e, e.\_\_doc\_\_, sep = '**\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()  *# 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') |

1. Retrieved 2022-Nov-27 [↑](#footnote-ref-1)