Anna Willie

November 27, 2021

IT FDN 110

Assignment 07

Pickling and Error handling

# Introduction

For this week’s assignment we worked on serializing our list data into a file, read it and save it to memory. We also added some error handling to our previous code, so the script doesn’t quit out automatically.

# Topic 1

First, I transferred my previous write and read functions to binary format. I initially had issues with this because my load function wouldn’t read the file and transfer the data into my list table. First, I had to change where I put return; I initially had it after the pickle.load and that wasn’t working so I moved it after the file closes and that allowed my pickle load to read the file. After that I had to change my parameters, because lstTbl was getting confused with a global and a parameter so I changed it to lstTable and that allowed for my pickle unpacking to work.

For my error handling I worked with my pickle files to tell me when my reading and writing functions were working. It will give me feedback if any error occurs. I previously had some error handling with the delete function and if the wrong ID is entered the script will let you know. I also have error handling with automating the ID numbers so a user can’t give a duplicate ID.

To research this project, I looked at a couple of websites to better understand pickling and error handling. For pickling I looked at [synopsis](https://www.synopsys.com/blogs/software-security/python-pickling/) and [tutorialspoint](https://www.tutorialspoint.com/python-pickling).[[1]](#footnote-1) Overall they both had their advantages and disadvantages. Synopsis did a really good job conceptionally explaining pickling, but it did not a do a good job with showing applicable examples. The tutorialspoint did a good job of showing examples, but the overall conceptional explanation wasn’t as good. For Error handling I looked at [tutorialspoint](https://www.tutorialspoint.com/python/python_exceptions.htm) and [Real Python.](https://realpython.com/python-exceptions/)[[2]](#footnote-2) I thought both websites did a nice job conceptually explaining what error handling is, but tutorialspoint showed more information and examples that I found to be useful and easy to understand.

Here are my examples of the script working in Spyder:

Text

Description automatically generated

Figure 1 Add and Delete Functions

Text

Description automatically generated

Figure 2 Save and Load Functions

Here is my code running in a shell:

Graphical user interface, text, application

Description automatically generated

Figure 3 Shell

Here is the link to my [GitHub](https://github.com/awillie32/Assignment07) post.

# Summary

Overall, I though this assignment was interesting to see how we can store data in different formats. I struggled a bit with my use of scope and global variable this time around and I am now aware how important it is to keep track of that.

# Appendix

Generated by [Syntax Highlighter](https://ajblk.github.io/SyntaxHighlightGenerator-v3.0/OnlineGenerator.html).[[3]](#footnote-3)

|  |  |
| --- | --- |
| 001  002  003  004  005  006  007  008  009  010  011  012  013  014  015  016  017  018  019  020  021  022  023  024  025  026  027  028  029  030  031  032  033  034  035  036  037  038  039  040  041  042  043  044  045  046  047  048  049  050  051  052  053  054  055  056  057  058  059  060  061  062  063  064  065  066  067  068  069  070  071  072  073  074  075  076  077  078  079  080  081  082  083  084  085  086  087  088  089  090  091  092  093  094  095  096  097  098  099  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 | #------------------------------------------#  # Title: CDInventory.py  # Desc: Working with classes and functions.  # Change Log: (Who, When, What)  # DBiesinger, 2030-Jan-01, Created File  # AWillie, 2021-Nov-17, Added add function and delete function  # AWillie, 2021-Nov-18, Worked on save and load function  # AWillie, 2021-Nov-21, Added ID naming function  # AWillie, 2021-Nov-26, Added pickling and some error handling  # AWillie, 2021-Nov-27, Fixed load pickling function  #------------------------------------------#    # -- DATA -- #  **import** pickle    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  ID **=** 0    # -- PROCESSING -- #  **class** DataProcessor:      @staticmethod  **def** New\_ID():  **global** ID          ID **+=** 1      @staticmethod  **def** add\_Title(strTitle, strArtist):          """ Function to add additonal CD's to inventory            Uses the input as variable to add to a dictionary to add to the inventory            Args:              strTitle: Variable for the CD title              strArtist: Variable for the CD\'s Artist            Returns:              None.          """  **global** ID          DataProcessor.New\_ID()          dicRow **=** {'ID': ID, 'CD Title': strTitle, 'Artist': strArtist}          lstTbl.append(dicRow)          print('Your CD has been added')          print()      @staticmethod  **def** del\_Title(IDIntDel):          """Function to delete a chosen CD title            User input the desired ID they would like to delete            Args:              IDIntDel: The ID # that is chosen to be deleted            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(FileName, lstTable):          """Function to read a binary file and to return the list data            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.          """          lstTable.clear()          with open(FileName, 'rb+') as objFile:  **try**:                  lstTable **=** pickle.load(objFile)  **except**:                  print('Something has gone wrong!')  **else**:                  print ('Successful read: %s' **%** (objFile))              objFile.close()  **return** list(lstTable)      @staticmethod  **def** write\_file(FileName, lstTable):          # Added code here          """Function to save data to a binary file            Takes the current memory and moves it to a binary file            Args:              file\_name(string): name of file used to copy memory to              table (list of dict): 2D data structure holding the inventory          Returns:              None.          """          with open(FileName, 'wb+') as objFile:  **try**:                  pickle.dump(lstTable, objFile)  **except**:                  print('An error has occured')  **else**:                  print ('Successful write: %s' **%** (objFile))          objFile.close()  # -- 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\tCD Title (by: Artist)\n')  **for** row **in** table:              print('{}\t{} (by:{})'.format(**\***row.values()))          print('======================================')      # 1. When program starts, read in the currently saved Inventory  FileProcessor.read\_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          # moved IO code into function          strTitle **=** input('What is the CD\'s title? ').strip()          strArtist **=** input('What is the Artist\'s name? ').strip()          DataProcessor.add\_Title(strTitle, strArtist)          IO.show\_inventory(lstTbl)          # 3.3.2 Add item to the table          #moved processing code into function  **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          intIDDel **=** int(input('Which ID would you like to delete? ').strip())          # 3.5.2 search thru table and delete CD          # moved processing code into function          DataProcessor.del\_Title(intIDDel)          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)          strYesNo **=** input('Save this inventory to file? [y/n] ').strip().lower()  **if** strYesNo **==** 'y':              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. Accessed November 26, 2021 [↑](#footnote-ref-1)
2. Accessed November 26, 2021 [↑](#footnote-ref-2)
3. Accessed November 27, 2021 [↑](#footnote-ref-3)