Danielle Rodriguez

13 March 2022

IT FDN 110 B

Assignment 07

Continuing to Improve the CD Inventory

Introduction

This week’s module covered structured error handling and working with text and binary files. We have worked with text files in previous modules but have not added write, read and append functions. Binary files allow data to be stored in memory to be retrieved later. In order to create and read binary files, the data must be pickled and converted to binary. Structure error handling allows for the user to avoid a program crash while still alerting users of any errors that may have occurred. These functions are useful tools to create an effective python script.

Text Files

Text files allow the user to write data to a file. We can create custom functions that wrap open, write, and close calls into one function when using text files. As seen in the figure below, we can use add read and append operations. Functions like readline() allow the script to return the next line of data following the function. As long as the file remains open, the location of the next line to read is memorized. Once the file is closed, the readline() function returns to the first line. Additionally, the readlines() function reads all the lines in the text file and returns a list.

Text

Description automatically generated

*Figure 1: Working with text files*

Pickling

In Python, pickling allows the coder to work with binary files. Binary files store information in memory and format the data in a way that can be saved and retrieved. The pickling module stored the data as binary information. Figure 2 outlines a script using the pickle function. There are some similarities to working data to text files, but the data transfer to files is different with pickling. Additionally, the script uses ‘b’ to signal binary data.

Text

Description automatically generated

*Figure 2: Pickling data*

Structured Error Handling

Often, Python crashes because of small mistakes in the script. But structured error handling allows the program to handle foreseeable errors without causing the program to crash. When creating structure error handling scripts, the exception class holds information about errors. We can create an object within the class that can extract details about the error. Objects can also be used to catch specific errors. The figure below outlines two ways to use structured error handling to prevent the program from crashing.

Text

Description automatically generated

*Figure 3: Structure Error handling*

Putting it all together

Using the functions and concepts introduced in this module, I modified the CD Inventory code to use structured error handling and the pickling module. Appendix 1 outlines the full script for the CD Inventory. Structured error handling was added to the areas that called for user input, limiting crashes caused by errors. The pickling module was also added, which changed the delete, write and read functions. Additionally, the file source change from .txt to .dat, allowing for the storage of binary data.

This process was not too difficult, but I had a few errors in my code which prevented the script from running the pickling functions. Once I figured out the line that was causing the error, I was able to run the code. Below is a snapshot of the code and an example of the error handling output when the user attempts to delete an entry by entering a non-valid ID.

Text

Description automatically generated

*Figure 4: CD Inventory script*

Summary

This module introduced functions to use with text files as well as pickling modules and structured error handling. In the process of learning these topics, I did external research on pickling and error handling. In doing so, I found sites that were both helpful[[1]](#footnote-1)[[2]](#footnote-2) and unhelpful[[3]](#footnote-3)[[4]](#footnote-4). The helpful files were filled with examples and approached the topics in a straightforward manner. By doing external research and applying the topics covered in the module, I modified the CD Inventory script.

Appendix

|  |  |
| --- | --- |
| 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 | *#------------------------------------------#*  *# Title: CDInventory.py*  *# Desc: Working with classes and functions.*  *# Change Log: (Who, When, What)*  *# DBiesinger, 2030-Jan-01, Created File*  *# DRodriguez, 2022-Feb-06, Modify file*  *# DRodriguez, 2022-Mar-13, added structured error handling and binary data*  *#------------------------------------------#*  **import** **pickle**  *# -- DATA -- #*  strChoice = '' *# User input*  lstTbl = [] *# list of lists to hold data*  dicRow = {} *# list of data row*  strFileName = 'CDInventory.dat' *# data storage file*  pickle\_out = **None** *# file object*  *# -- PROCESSING -- #*  **class** **DataProcessor**:  @staticmethod  **def** add\_CD():  *"""Function to add a new item to table*    *Args:*  *dicRow: creates dictionary*  *lstTbl: adds new values to dictionary*  *show\_inventory: displays data*    *Returns:*  *Table showing inventory in lstTbl*    *"""*  intID, strTitle, strArtist = IO.add\_CD()  dicRow = {'ID': intID, 'Title': strTitle, 'Artist': strArtist}  lstTbl.append(dicRow)  **return** IO.show\_inventory(lstTbl)    @staticmethod  **def** del\_CD():  *"""Function to search thru table and delete CD*    *Args:*  *intRowNr:*  *blnCDRemoved:*  *stTbl (list of dict): 2D data structure (list of dicts) that holds the data during runtime.*    *Returns:*  *Table showing inventory in lstTbl*    *"""*  **try**:  intRowNr = -1  blnCDRemoved = **False**  **for** row **in** lstTbl:  intRowNr += 1  **if** row['ID'] == intIDDel:  **del** lstTbl[intRowNr]  blnCDRemoved = **True**  **break**  **except** **ValueError**:  print('Please enter a valid CD ID.')  **except** **Exception** **as** e:  print('Request could not be completed. There was an error.')  print(type(e), e, e.\_\_doc\_\_, sep='**\n**')  **else**:  blnCDRemoved == **True**  print('The CD was removed')  **return** IO.show\_inventory(lstTbl)  @staticmethod  **def** write\_file(file\_name, table):  *"""Function to save data*    *Args:*  *objFile: calls the text file containing CD inventory*  *lstValues: row of values in 2D table*  *objFile.write: writes inputted data to file*  *stTbl (list of dict): 2D data structure (list of dicts) that holds the data during runtime.*    *Returns:*  *Table showing inventory in lstTbl if no errors occur*    *"""*  **try**:  **with** open(file\_name, 'wb') **as** readfileObj:  pickle.dump(table, readfileObj)  **except** **FileNotFoundError**:  print('File not found.')  **except** **Exception** **as** e:  print('Request could not be completed. There was an error.')  print(type(e), e, e.\_\_doc\_\_, sep='**\n**')  **else**:  **return** table  **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 if no errors occur*  *"""*  table.clear() *# this clears existing data and allows to load data from file*  **try**:  **with** open(file\_name, 'rb') **as** writefileObj:  table = pickle.load(writefileObj)  **except** **FileNotFoundError**:  print('File not found.')  **except** **Exception** **as** e:  print('Request could not be completed. There was an error.')  print(type(e), e, e.\_\_doc\_\_, sep='**\n**')  **else**:  **return** table  *# -- 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** add\_CD():  *"""Ask user for new ID, CD Title and Artist*  *Args:*  *strID: name of ID input*  *strTitle: name of title input*  *strArtist: name of artist input*  *intID: Converts strID to integer*  *Returns:*  *values defined in arguments if no errors occur*    *"""*  **while** **True**:  strID = input('Enter ID: ').strip()  **try**:  intID = int(strID)  **break**  **except** **ValueError**:  print('Please enter a number for CD ID.')  **except** **Exception** **as** e:  print('Request could not be completed. There was an error.')  print(type(e), e, e.\_\_doc\_\_, sep='**\n**')  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  *# When program starts, read in the currently saved Inventory*  FileProcessor.read\_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**  *# 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)  **else**:  input('canceling... Inventory data NOT reloaded. Press [ENTER] to continue to the menu.')  IO.show\_inventory(lstTbl)  **continue** *# start loop back at top.*  *# Add a CD*  **elif** strChoice == 'a':  *# Ask user for new ID, CD Title and Artist and add item to table*  DataProcessor.add\_CD()  **continue** *# start loop back at top.*  *# Display current inventory*  **elif** strChoice == 'i':  IO.show\_inventory(lstTbl)  **continue** *# start loop back at top.*  *# Delete a CD*  **elif** strChoice == 'd':  *# Get Userinput for which CD to delete*  *# Display Inventory to user*  IO.show\_inventory(lstTbl)  *# Ask user which ID to remove*  **try**:  intIDDel = int(input('Which ID woudld you like to delete? ').strip())  **except** **ValueError**:  print('Please enter a valid CD ID.')  **except** **Exception** **as** e:  print('Request could not be completed. There was an error.')  print(type(e), e, e.\_\_doc\_\_, sep='**\n**')  *# Search thru table and delete CD*  DataProcessor.del\_CD()  **continue** *# start loop back at top.*  *# 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':  *# Save data*  DataProcessor.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.*  *#Catch-all should not be possible, as user choice gets vetted in IO, but to be save:*  **else**:  print('General Error') |

***Appendix 1: CD Inventory Script***

A picture containing text

Description automatically generated

***Appendix 2: CD Inventory Script running in terminal***

Text

Description automatically generated

***Appendix 3: CD Inventory Script running in Spyder***

1. Helpful site for pickling: <https://www.tutorialspoint.com/python-pickling> (external) [↑](#footnote-ref-1)
2. Helpful site for structured error handling: <https://www.programiz.com/python-programming/exception-handling> (external) [↑](#footnote-ref-2)
3. Unhelpful site for pickling: <https://www.datacamp.com/community/tutorials/pickle-python-tutorial> (external) [↑](#footnote-ref-3)
4. Unhelpful site for structured error handling: <https://www.datacamp.com/community/tutorials/exception-handling-python> (external) [↑](#footnote-ref-4)