**Dimitrios Tsakalos**

Sunday, February 28, 2021

IT FDN 110 A Wi 21: Foundations of Programming: Python

[Assignment 07](https://github.com/tsakdimi/Assignment_07)

**CD Inventory program included structured error handling and storing data in binary format**

# Introduction

In Assignment 07 for this course, I reviewed the material for Module 07 which included watching the module videos, reading book chapter 07 and reading webpages explaining structured error handling and pickling. I read through the Foundations of Programming (Python) PDF and completed the assignment by applying my knowledge to create a script in Python that collects input from the user, organizes the code in classes containing defined functions and contains script that saves the data in binary mode as well as accounts for handling errors. The knowledge document and Python file were uploaded to [GitHub](https://github.com/tsakdimi/Assignment_07) (external reference)[[1]](#footnote-1).

# Reviewing the material

The material I reviewed for Module 07 included reading book chapter 07 and searching online for information about pickling in Python and structured error handling. The best website explaining pickling in Python was on [realpython.com](https://realpython.com/python-pickle-module/) (external reference)[[2]](#footnote-2). My main take-aways from here are:

* Pickling is Python specific
* Pickling serializes data to store it, and is best used when we do not need it to be human-readable
* Not everything is picklable. The list of unpicklable objects includes database connections, opened network sockets, running threads, and others.
* Pickling is not secure

The best website for reviewing structured error handling for me was on [realpython.com](https://realpython.com/python-exceptions/) (external reference)[[3]](#footnote-3). My main take-aways are:

* Errors can occur in two ways; incorrect syntax and when correct syntax script results in an error when it is used
* It is always better to catch an error than have your program terminate uncontrollably
* Raise is a powerful tool which you can use to throw an exception any time
* Using the try clause executes all statements until an exception is encountered

After looking up information online I then worked on the Foundations of Programming (Python) PDF provided in module 07, which built on Assignment 06. After reviewing the materials and the assignment requirements, I started working on the labs step by step. The labs helped and I gained an understanding of the new concepts. This week’s challenges were:

* Understanding pickling
* Getting the program to run in binary
* Understanding classes and how arguments and returns feed into the main

# Working on the Foundations of Programming (Python) Module 07

After reviewing the above materials, I proceeded to work on the Foundations of Programming Module 07 Labs. I started with LAB 07-A. I first added the code from LAB07\_A\_Starter.py, and then modified the script to read the numbers to and from a text file (see Figure 1). The program reads the numbers from a text file, makes the calculation and returns the results (see Figures 2 and 3). The code performs the following:

* Lines 9 and 10 declare the global variables
* Lines 13 to 66 define the 4 functions, taking val1 and val2 as attributes and returning values of the corresponding mathematical operation to be processed
* Lines 83 to 88 define the read\_file function in the IO class and read the data from file
* Lines 101 to 106 define the write\_file function in the IO class and write the data to file
* Line 109 prints a header, line 110 assigns variables and calls IO.read\_file function
* Line 111 defines the variable lstResults, followed by lines 112 to 115 that append the data to lstResults after calling each function
* Line 116 writes the lstResults appended table to the file given as a first argument in the function.

In order to handle multiple lines and to write multiple lines to file we should utilize the readlines function.

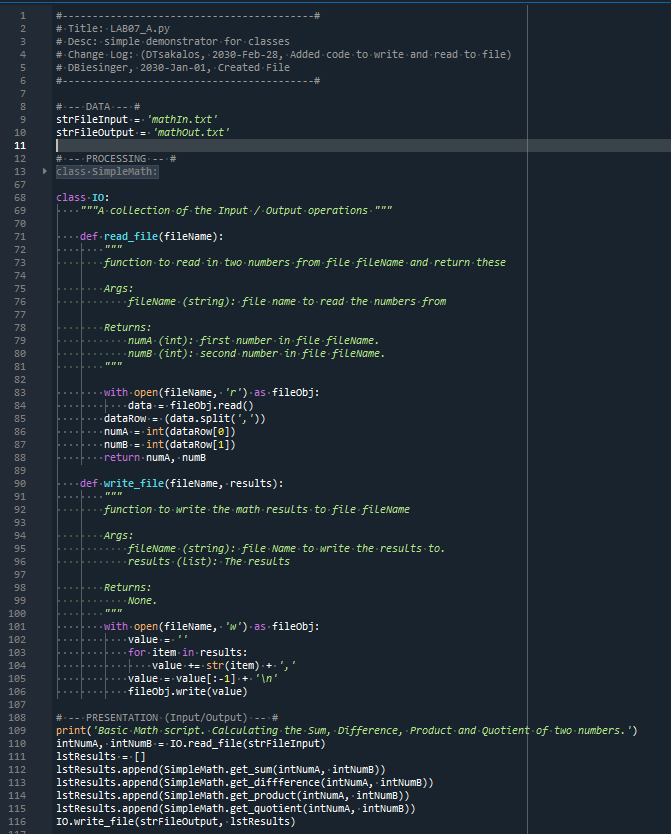


Figure 1 - LAB07-A code



Figure 2 - LAB07-A console

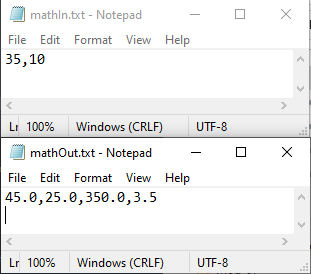


Figure 3 - LAB07-A mathIn.txt and mathOut.txt data

I used the code in LAB-07A as input to LAB-07B and modified it to read in distinct operational paths using the arguments calc and IO. I changed the functions to access binary files (see Figure 4). I then confirmed the program functioned correctly (see figure 5). The code performs the following:

* Lines 8 and 9 import the sys and pickle modules
* Lines 12 and 13 declare the global variables
* Lines 16 to 69 define the 4 functions, taking val1 and val2 as attributes and returning values of the corresponding mathematical operation to be processed
* Lines 86 to 88 define the read\_file function in the IO class and read the data from file using the pickle module
* Lines 102 to 103 define the write\_file function in the IO class and write the data to file using the pickle module
* Line 106 defines arg and lines 107 and 108 are there to allow the program to function when a user does not enter a command line argument and notifies the user with a print function
* Lines 110 to 116 use the sys.argv function and check in list index 1 if the argument matches the string name. If it matches the data the results file is printed out, followed by prompting for user input and then writing to file.
* Lines 119 to 128 use the sys.argv function and check in list index 1 if the argument matches the string name. If it matches the data is read using the argument strFileInput. In lines 121 and 122 intNumA and intNumB are assigned to specific list indices and the results are appended to the lstResults table and then written to file by calling the function IO.write\_file
* Lines 130 and 131 activate if a user selects a command line argument that is unsupported.
* Line 133 prints the header with the program information.

Saving to file using the pickle module is useful because it is easier to store the data without converting it to strings and working with the correct format. We can just dump it and serialize it while still retaining all the functionality we have built into the code that stores the data in memory when the program runs.

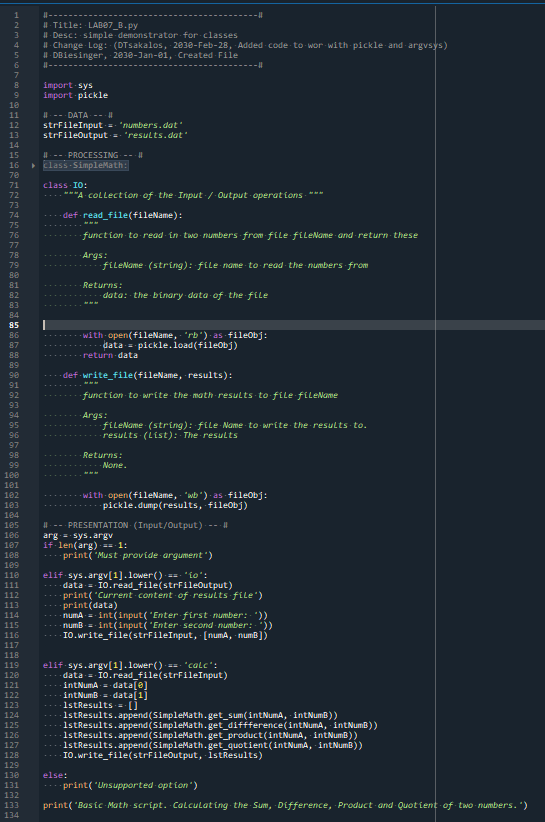


Figure 4 - LAB07-B code

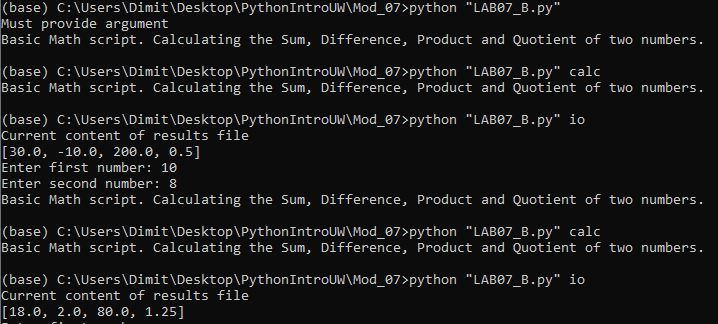


Figure 5 - LAB07-B in Anaconda prompt

For LAB-07C I made a copy of the script from LAB-07B. I then added code to catch errors around file operations and code to catch errors around the user input operations. I also handled non-numeric inputs and value zero input for the number used as a denominator separataly (see Figure 6). I then tested the script (see Figure 7). The code performs the following:

* Lines 9 and 10 import the sys and pickle modules
* Lines 13 and 14 declare the global variables
* Lines 17 to 70 define the 4 functions, taking val1 and val2 as attributes and returning values of the corresponding mathematical operation to be processed.
* In lines 72 to 85 the read\_file function is created and explained with docstrings
* In lines 86 to 93 a while, try, except statement is used to account for a FileNotFoundError. The script returns None in the case of no file present in the folder
* Lines 96 to 112 define the write\_file function in the IO class and write the data to file using the pickle module. In lines 107 to 112 I use a try, except statement to account for a missing file and create a file with an empty list to dump the data in
* Line 116 contains the program title
* Lines 121 to 124 operate the program when a sys.argv argument IO is entered.
* Lines 125 to 130 use a try, except statement in a while loop to catch a ValueError when a user enters a non-integer.
* Lines 131 to 140 use a try, except statement in a while loop to control for a ZeroDivisionError, raising that error when numB is 0 and also accounting later for a non-integer being entered
* Line 145 makes use of the None returned in line 93 and informs the user using the code in line 155 that they must run IO before calc
* Lines 143 to 153 do the calculation based on the data the previous program run completed in the IO mode

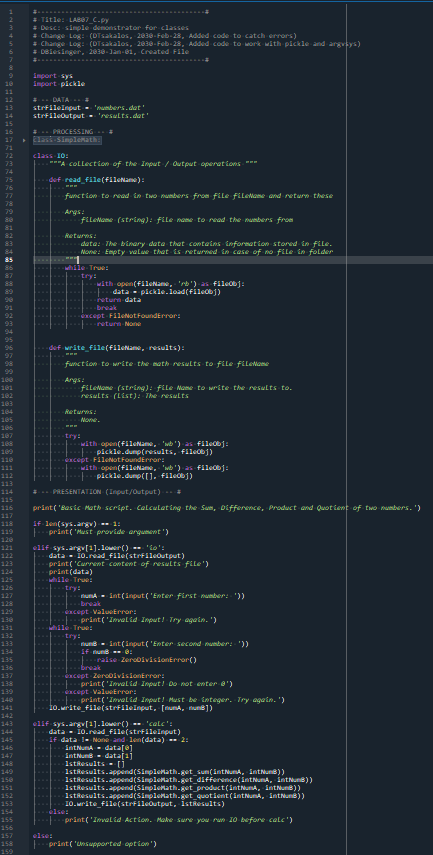


Figure 6 - LAB07-C code

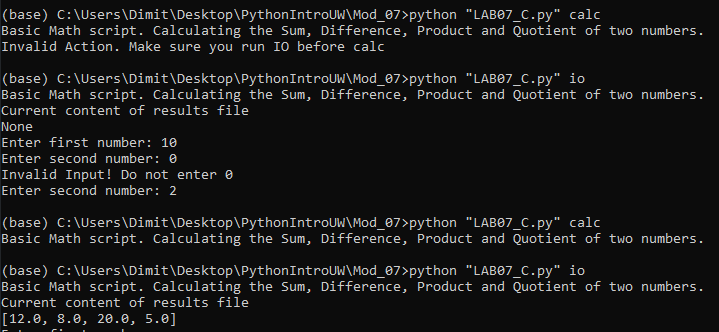


Figure 7 - LAB07-C in Anaconda console

# Creating a CD Inventory program using binary data storing and structured error handling

After completing the labs, I moved onto Assignment07. After going through all the steps of folder creation and header modifications I started working on the code (see Appendix Listing CDInventory.py). The code performs the following:

* Line 8 imports the pickle module
* Lines 11 to 15 declare the global variables
* Lines 18 and 19 create the DataProcessor class that processes data within memory and its docstring
* Lines 21 to 39 add the append\_data function to the class DataProcessor. The function appends data to the table
* Lines 43 to 65 add the delete\_data function to the class DataProcessor. The function has the arguments id\_to\_remove and table. It checks for a dictionary that has a match in ID and removes it
* Lines 67 to 68 introduce the class of FileProcessor that processes data to and from DAT file and its docstring.
* Lines 70 to 97 add the read\_file function to the class FileProcessor. The function reads the data from file to memory using the pickle module. For the program to be able to run the first time without a DAT file at hand, I created a while, try, except statement in lines 86 to 94 to account for FileNotFoundError and create a first-time DAT file.
* Lines 100 to 117 add the write\_file function to the class FileProcessor. The function writes to file the data from memory using the pickle module.
* Lines 121 to 122 create the IO class that handles user input, output and its docstring
* Lines 125 to 137 add the print\_menu function to the class IO. The function prints the main menu
* Lines 140 to 153 add the menu\_choice function to the class IO. The function prints the choice menu that collects user input. The user chooses amongst load, add, inventory display, delete entry, saving to file and exiting.
* Lines 156 to 169 add the show\_inventory function to the class IO. The function prints the current in-memory inventory for the user by accessing the list of dictionaries and formatting the data to display for easier user readability
* Lines 172 to 192 add the load\_choice function to the class IO. The function warns the user that if they proceed, they will lose unsaved data and runs if the user selects yes by calling the FileProcessor.read\_file function
* Lines 196 to 232 add the get\_new\_cd\_data function to the class IO. Within the code at each input statement there is a ValueError in a while loop in case there is an integer or empty string
* Lines 235 to 298 contain the main loop and run when the user selects the equivalent string choice.

After that, I ensured the program ran correctly in the Spyder console (see Figure 8), then I ran a last check in the Anaconda Prompt (see Figure 9).

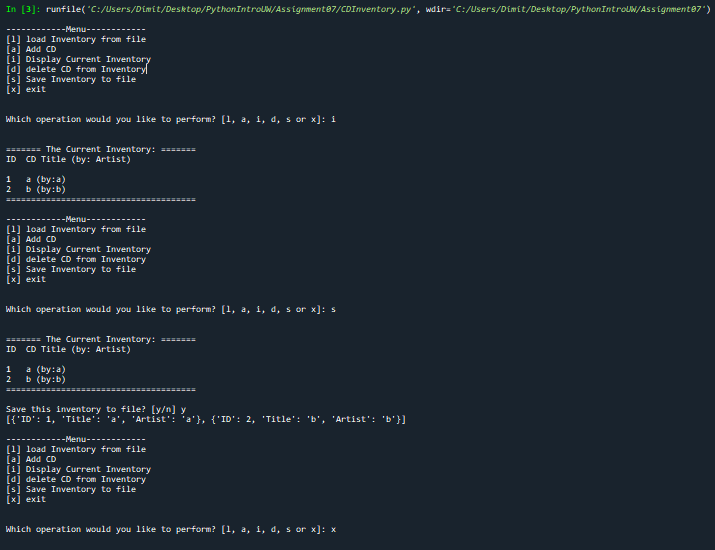


Figure 8 - CDInventory.py console

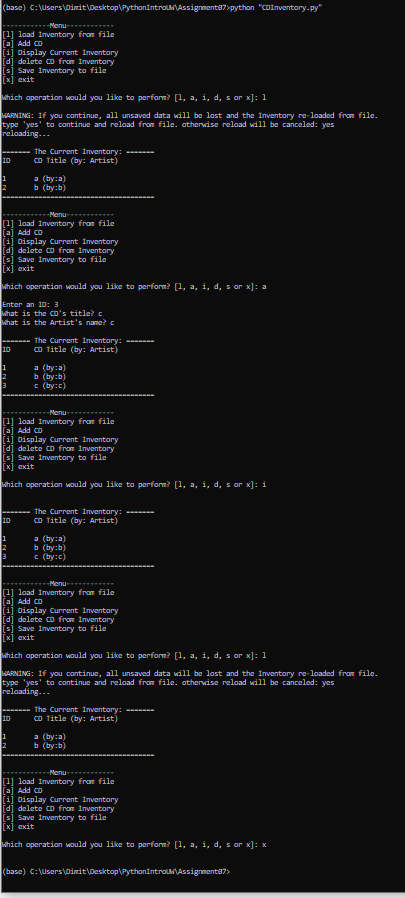


Figure 9 - CDInventory.py in Anaconda Prompt

# Summary

After reviewing the material and going through the Labs, I worked on Assignment 07. The challenges this week were thinking binary and making sure to find ways to break my program so that I could incorporate error handling for proofing against any scenario. I feel structured error handling is very important for a program, but it seems hard to think of ways to break a program when your knowledge is not that extensive. Searching online resources helped me understand the potential scenarios that a user may input better, although some of them seemed excessive for this assignment. I look forward to seeing how my classmates handled these errors!

# Appendix

## Listing Github

<https://github.com/tsakdimi/Assignment_07>

## Listing CDInventory.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  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 | *#------------------------------------------#*  *# Title: CDInventory.py*  *# Desc: Working with classes and functions.*  *# Change Log: (DTsakalos, 2021-Feb-28, Added code for error handling and writing/reading binary to file)*  *# DBiesinger, 2030-Jan-01, Created File*  *#------------------------------------------#*  **import** pickle  *# -- DATA -- #*  strChoice = '' *# User input*  lstTbl = [] *# list of dicts to hold data*  dicRow = {} *# dict of data row*  strFileName = 'CDInventory.dat' *# data storage file*  objFile = None *# file object*  *# -- PROCESSING -- #*  **class** DataProcessor:  """Processing the data within memory"""  @staticmethod  **def** append\_data(intID, strTitle, strArtist, table):  """Function to manage data addition within program memory  Strips the data the user inputs and then appends them into a 2D table  (list of dicts) table one line in the file represents one dictionary row in table  Args:  intID (integer): contains that ID number of CD  strTitle (string): Contains the Title of song name  strArtist (string): Contains the name of the Artist  table (list of dict): 2D data structure (list of dicts) that holds the data during runtime  Returns:  None.  """  dicRow = {'ID': intID, 'Title': strTitle, 'Artist': strArtist}  table.append(dicRow)  @staticmethod  **def** delete\_data(id\_to\_remove, table):  """ Function to manage data deletion within program memory  Asks user for an ID number and checks to find the appropriate listing in a  (list of dicts) table and removes the row (dict) that holds that ID number  Args:  id\_to\_remove (integer): the id the user wants the containing dictionairy to be removed  table (list of dict): 2D data structure (list of dicts) that holds the data during runtime  Returns:  blnCDRemoved (boolean): Checks if CD is removed  """  intRowNr = -1  blnCDRemoved = False  **for** row **in** table:  intRowNr += 1  **if** row['ID'] == id\_to\_remove:  **del** table[intRowNr]  blnCDRemoved = True  **break** *# Cannot remove more than one entry. When we have the same ID numbers only first is removed*  **return** blnCDRemoved  **class** FileProcessor:  """Processing the data to and from dat file"""  @staticmethod  **def** read\_file(file\_name, tablex):  """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  tablex (list of dict): 2D data structure (list of dicts) that holds the data during runtime  Returns:  None.  """  tablex.clear() *# this clears existing data and allows to load data from file*  table = []  **while** True:  **try**:  **with** open(file\_name, 'rb') **as** objFile:  table = pickle.load(objFile)  **break**  **except** FileNotFoundError:  **with** open(file\_name, 'wb') **as** objFile:  pickle.dump(table, objFile)  **print**('No previous file found. Created an empty file!')  **for** row **in** table:  tablex.append(row)  @staticmethod  **def** write\_file(file\_name, table):  """Function to save data from a list of dictionaries to file in csv formating  Saves the data from a 2D table in memory in current program  (list if dicts) and saves it to file, with each file line representing a row  of the 2D table, and each comma within a row separating the columns  Args:  file\_name, (string): name of file used to save the data to  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**('\n------------Menu------------')  **print**('[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 string 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**('\n======= The Current Inventory: =======')  **print**('ID\tCD Title (by: Artist)\n')  **for** row **in** table:  **print**('{}\t{} (by:{})'.format(\*row.values()))  **print**('======================================')  @staticmethod  **def** load\_choice():  """Gets user input after selecting load on main menu  Warns the user unsaved data will be lost if they type yes and runs FileProcessor.read\_file  Args:  None.  Returns:  load\_file (Boolean): checks if user wants to load file  """  load\_file = False  **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...')  load\_file = True  **else**:  input('canceling... Inventory data NOT reloaded. Press [ENTER] to continue to the menu.')  **return** load\_file  @staticmethod  **def** get\_new\_cd\_data():  """Gets user input after selecting load on main menu  Warns the user unsaved data will be lost if they type yes and runs FileProcessor.read\_file  Args:  None.  Returns:  intID (integer): contains that ID number of CD  strTitle (string): Contains the Title of song name  strArtist (string): Contains the name of the Artist  """  *# 3.3.1 Ask user for new ID, CD Title and Artist*  **while** True:  **try**:  intID = int(input('Enter an ID: ').strip())  **break**  **except** ValueError: *#making sure the program does not crash with string as input*  **print**('Invalid Input! Try again.')  **while** True:  **try**:  strTitle = input('What is the CD\'s title? ').strip()  **if** strTitle == '':  **raise** ValueError()  **break**  **except** ValueError: *#* *making sure the program does not contain empty string* **print**('Invalid Input! Try again.')  **while** True:  **try**:  strArtist = input('What is the Artist\'s name? ').strip()  **if** strArtist == '':  **raise** ValueError()  **break**  **except** ValueError: *#* *making sure the program does not contain empty string* **print**('Invalid Input! Try again.')  **return** intID, strTitle, strArtist  *# 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':  reload\_file = IO.load\_choice()  **if** reload\_file:  FileProcessor.read\_file(strFileName, lstTbl)  IO.show\_inventory(lstTbl)  *# 3.3 process add a CD*  **elif** strChoice == 'a':  int\_id, str\_title, str\_artist = IO.get\_new\_cd\_data()  DataProcessor.append\_data(int\_id, str\_title, str\_artist, lstTbl)  IO.show\_inventory(lstTbl)  *# 3.4 process display current inventory*  **elif** strChoice == 'i':  IO.show\_inventory(lstTbl)  *# 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())  removed = DataProcessor.delete\_data(intIDDel, lstTbl)  **if** removed:  **print**('The CD was removed\n')  **else**:  **print**('Could not find this CD!\n')  IO.show\_inventory(lstTbl)  *# 3.6 process save inventory to file*  **elif** strChoice == 's':  IO.show\_inventory(lstTbl)  strYesNo = input('Save this inventory to file? [y/n] ').strip().lower()  *# 3.6.2 Process choice*  **if** strYesNo == 'y':  FileProcessor.write\_file(strFileName, lstTbl)  **else**:  input('The inventory was NOT saved to file. Press [ENTER] to return to the menu.')  *# 3.7 catch-all should not be possible, as user choice gets vetted in IO, but to be safe:*  **else**:  **print**('General Error') |

1. Retrieved 2021-Feb-28 [↑](#footnote-ref-1)
2. Retrieved 2021-Feb-28 [↑](#footnote-ref-2)
3. Retrieved 2021-Feb-28 [↑](#footnote-ref-3)