**Dimitrios Tsakalos**

Sunday, February 21, 2021

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

[Assignment 06](https://github.com/tsakdimi/Assignment_06)

**CD Inventory program using classes and functions in Python**

# Introduction

In Assignment 06 for this course, I reviewed the material for Module 06 which included watching the module videos, reading book chapter 06, reading the suggested webpages and watching the additional videos. 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 and organizes the code in classes containing defined functions. The knowledge document and Python file were uploaded to [GitHub](https://github.com/tsakdimi/Assignment_06) (external reference)[[1]](#footnote-1).

# Reviewing the material

The material I reviewed for Module 06 included watching the module videos, reading book chapter 06, looking at the suggested web pages and searching online for extra information for how to utilize functions and classes. I first looked at the Foundations of Programming (Python) PDF provided in module 06, which seemed a progression from last week’s Assignment 05. 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, although I still do not feel I have mastered all of this week’s material. This week’s challenges were:

* Understanding attributes and return values
* Getting the program for Assignment06 to run without having a text file in the folder
* Paying attention to scope and global variables
* Navigating through code with many lines (many lines for my level) and working off of another person’s code

# Working on the Foundations of Programming (Python) Module 06

After reviewing the above materials, I proceeded to work on the Foundations of Programming Module 06 Labs. I started with LAB 06-A. I first added the code from Module 06 listing 2, and then modified the script to work with attributes and return value calculations (see Figure 1). The program allows the user to enter two numbers and returns four basic math calculations (see Figure 2). The code performs the following:

* Lines 9 and 10 declare the global variables
* Lines 14 to 24 define the 4 functions, taking value1 and value2 as attributes and returning values of the corresponding mathematical operation to be processed
* Lines 28 to 30 collect user input
* Lines 33 to 34 display the input the user entered in a formatted way using positional arguments
* Lines 36 to 39 assign a variable to the functions after changing the parameters of the function to user input intNumA and intNumB
* Lines 41 to 44 execute print statements that format the variables assigned in lines 36 to 39 using positional arguments

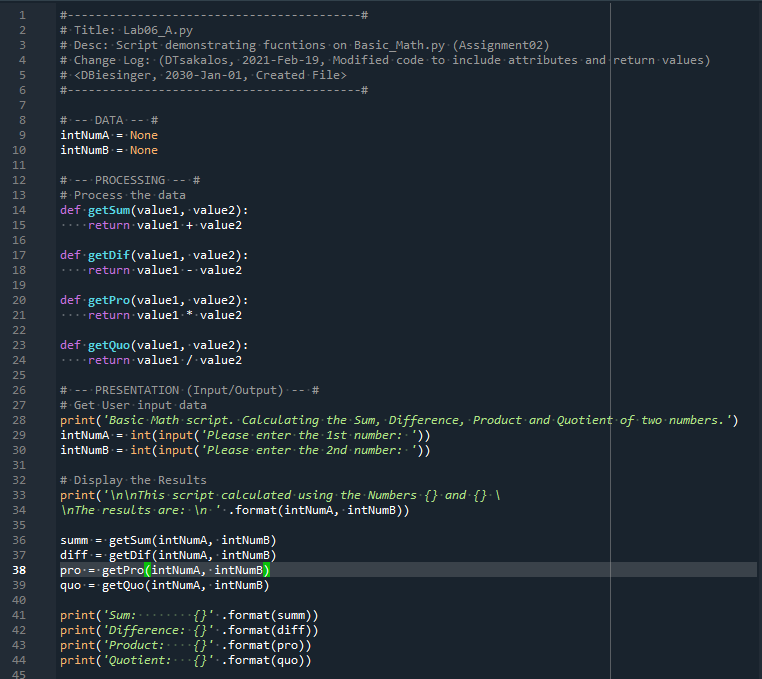


Figure 1 - LAB06-A code

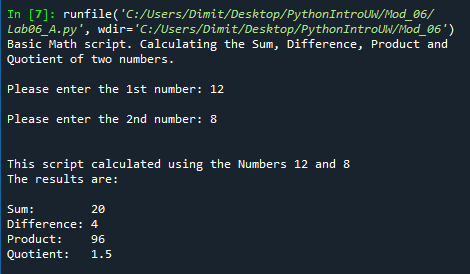


Figure 2 - LAB06-A console

I used the code in LAB-06A as input to LAB-06B and modified it to create a function that returns multiple values (see Figure 3). I then confirmed the code should work as desired (see Figure 4). The code performs the following:

* Lines 9 and 10 declare the global variables
* Lines 14 to 23 define the 4 functions, taking value1 and value2 as attributes, assign a local variable to the corresponding mathematical operation and returning values of the local variable
* Lines 28 to 30 collect user input
* Line 33 assigns the return call of the function to the variables summAns, diffAns, proAns, quoAns, after changing the parameters of the function to the captured user input
* Lines 34 and 35 print the numbers entered by the user, followed by the variables assigned in line 33 using positional arguments

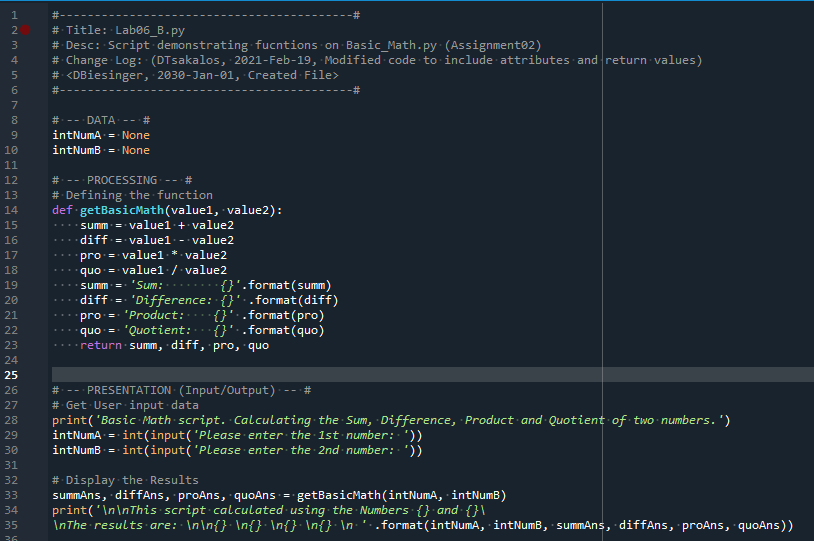


Figure 3 - LAB06-B code

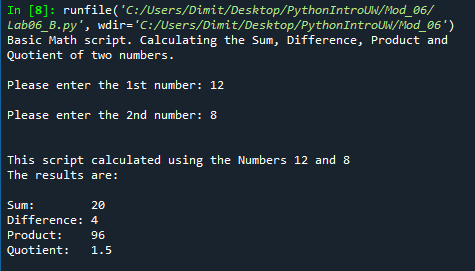


Figure 4 - LAB06-B console

For LAB-06C I made a copy of the script from LAB 06-A then added code and the docstring to create a class called SimpleMath, a function get\_sum, a function get\_diffference, a function get\_product and a function get\_quotient all within the class (see Figure 5). I also added code for the I/O to capture user input and format the output to follow instructions (see Figure 6). The code performs the following:

* Lines 9 and 10 declare the global variables
* Lines 14 to 15 create the class SimpleMath and relevant docstring
* Lines 17 to 59 define the functions of the class with two parameters and return the corresponding mathematical operation
* Lines 63 to 65 collect user input
* Lines 68 to 69 execute print statements that format the variables assigned in lines 64 to 65 using positional arguments
* Lines 70 to 73 execute print statements that call the functions within the class and assign the variables created in lines 64 and 65 to produce the relevant mathematical outcomes

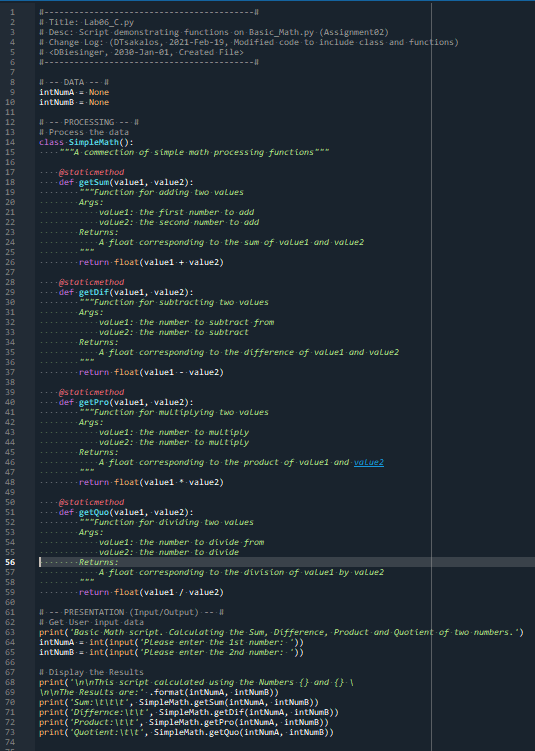


Figure 5 - LAB06-C code

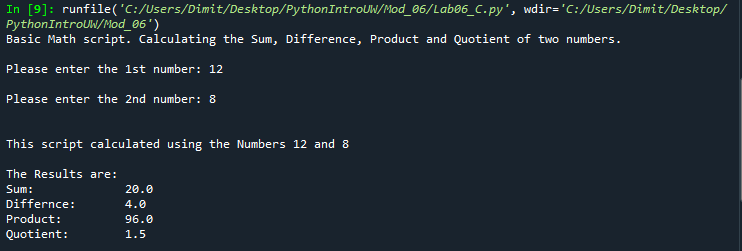


Figure 6 - LAB06 - C console

# Creating a CD Inventory program using classes and functions in Python

After completing the labs, I moved onto Assignment06. 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:

* Lines 9 to 13 declare the global variables
* Lines 16 and 17 create the DataProcessor class that processes data within memory and its docstring
* Lines 19 to 44 add the append\_data function to the class DataProcessor. The function collects user input and appends the data to a list of dictionaries and finishes by displaying current inventory by calling to function in another class
* Lines 46 to 77 add the delete\_data function to the class DataProcessor. The function collects user input and deletes the dictionary that holds the ID that matches the number the user typed in. The function breaks after one match is made, and I didn’t use my previous assignment code and kept the starter code because this is more efficient for programming power since it doesn’t make a copy of the whole list of dictionaries to run through every entry. When in the future we check for double ID entries and reject them it will be error-proof.
* Lines 79 to 80 introduces the class of FileProcessor that processes data to and from text file and its docstring.
* Lines 82 to 107 add the read\_file function to the class FileProcessor. The function reads the data from file to memory, adding the data to a 2D list of dictionaries. For the program to be able to run first time without a text file at hand, I created a while, try, except statement in lines 97 to 102 to account for FileNotFoundError and create a first-time text file.
* Lines 109 to 137 add the write\_file function to the class FileProcessor. The function writes to file the data from memory by converting the data from a 2D list of dictionaries to a string in a comma separated values format with each row a dictionary entry
* Lines 140 to 141 create the IO class that handles user input and output and its docstring
* Lines 143 to 156 add the print\_menu function to the class IO. The function prints the main menu
* Lines 158 to 172 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 174 to 188 add the show\_inventory function to the class IO. The function prints the current in-memory inventory to the user by accessing the list of dictionaries and formatting the data to display for easier user readability
* Lines 190 to 210 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 located in lines 82 to 107 of the code and operates as described above
* Lines 212 to 213 execute first when the program starts and point to the FileProcessor.read\_file function
* Lines 215 to 219 execute second which point to the IO.print\_menu function and the IO.menu\_choice function
* Lines 221 to 245 check for the user input in the IO.menu\_choice function and run the appropriate function.
* Lines 247 to 249 should not be possible as within the IO.menu\_choice function we have a while, not in, return statement.

After that, I ensured the program ran correctly in the Spyder console (see Figure 7). I then confirmed the data saved correctly in the CDInventory.txt file (see Figure 8). I ran a last check in the Anaconda Prompt (see Figure 9) and made sure the last table entry saved correctly to file (see Figure 10).

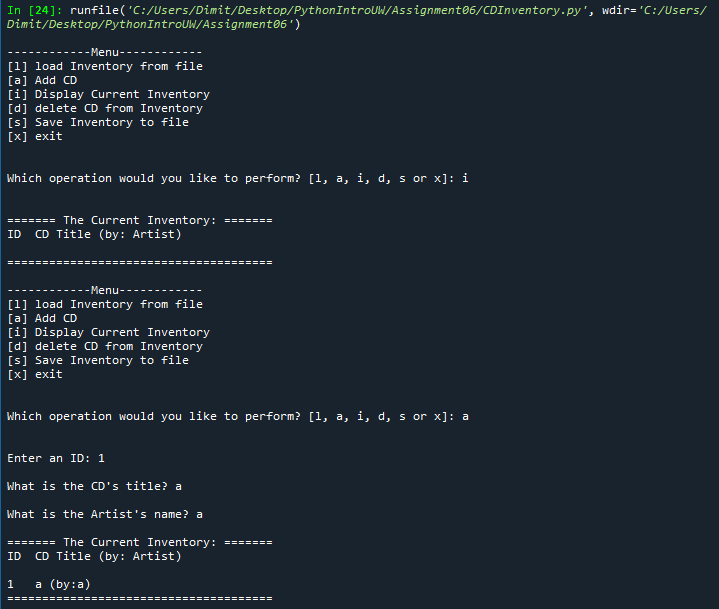


Figure 7 – CDInventory.py console

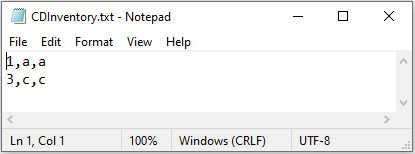


Figure 8 - CDInventory.txt saving in correct format

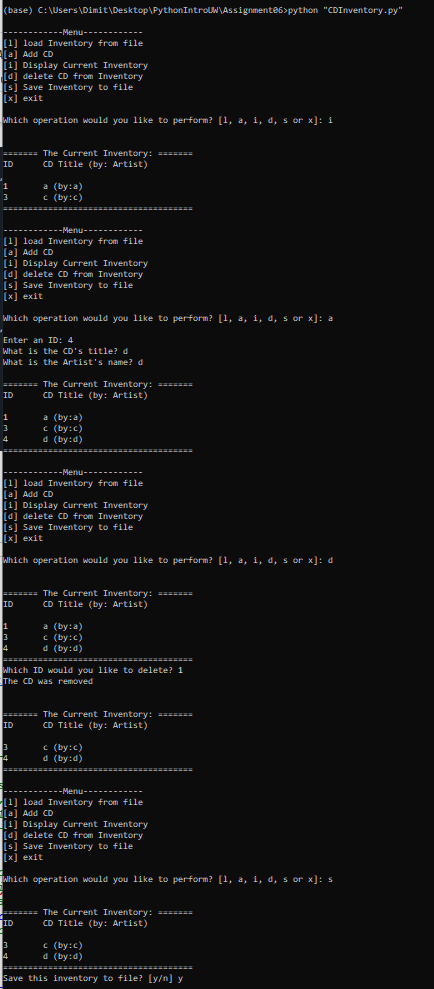


Figure 9 – CDInventory.py Anaconda Prompt

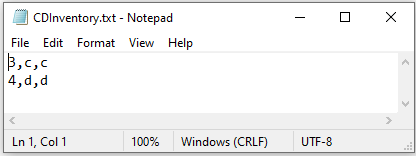


Figure 10 - CDInventory.txt saving in correct format after Anaconda Prompt run

# Summary

After reviewing the material and going through the Labs, I worked on Assignment 06. While it seemed a bit easier than I first thought, I am still in a process of digesting the information and will be revisiting it throughout the week. I again enjoyed the challenge of working on someone else’s code, and kept weighing how much to modify the existing code and if I should go beyond the TODOs. The challenges this week were understanding the differences between parameters, attributes and arguments, and trying to avoid calling global variables in my classes and functions. While I am sure organizing the code in classes and including docstrings under the functions is great practice, working with them for the first time and having a script with the length I had was a challenge. Small tweaks for how to get the program to run with no text file, how to organize my code and if it is a good practice to call between classes was challenging. On a last note, it is rewarding to see something that comes close to 10kb in size for a Python file, although I am sure this is still very small!

# Appendix

## Listing Github

<https://github.com/tsakdimi/Assignment_06>

## 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 | *#------------------------------------------#*  *# Title: CDInventory.py*  *# Desc: Working with classes and functions.*  *# Change Log: (DTsakalos, 2021-Feb-21, Added code to complete program)*  *# DBiesinger, 2030-Jan-01, Created File*  *#------------------------------------------#*  *# -- DATA -- #*  strChoice = '' *# User input*  lstTbl = [] *# list of dicts to hold data*  dicRow = {} *# dict of data row*  strFileName = 'CDInventory.txt' *# data storage file*  objFile = None *# file object*  *# -- PROCESSING -- #*  **class** DataProcessor:  """Processing the data within memory"""  @staticmethod  **def** append\_data(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:  table (list of dict): 2D data structure (list of dicts) that holds the data during runtime  Returns:  None.  """  *# 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.')  strTitle = input('What is the CD\'s title? ').strip()  strArtist = input('What is the Artist\'s name? ').strip()  *# 3.3.2 Add item to the table*  dicRow = {'ID': intID, 'Title': strTitle, 'Artist': strArtist}  table.append(dicRow)  IO.show\_inventory(table)  @staticmethod  **def** delete\_data(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:  table (list of dict): 2D data structure (list of dicts) that holds the data during runtime  Returns:  None.  """  *# 3.5.1 get Userinput for which CD to delete*  *# 3.5.1.1 display Inventory to user*  IO.show\_inventory(table)  *# 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*  intRowNr = -1  blnCDRemoved = False  **for** row **in** table:  intRowNr += 1  **if** row['ID'] == intIDDel:  **del** table[intRowNr]  blnCDRemoved = True  **break** *# Cannot remove more than one entry. When we have the same ID numbers only first is removed*  **if** blnCDRemoved:  **print**('The CD was removed\n')  **else**:  **print**('Could not find this CD!\n')  IO.show\_inventory(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:  None.  """  table.clear() *# this clears existing data and allows to load data from file*  **while** True:  **try**:  objFile = open(file\_name, 'r')  **break**  **except** FileNotFoundError:  objFile = open(file\_name, 'w')  **for** line **in** objFile:  data = line.strip().split(',')  dicRow = {'ID': int(data[0]), 'Title': data[1], 'Artist': data[2]}  table.append(dicRow)  objFile.close()  @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.  """  *# 3.6.1 Display current inventory and ask user for confirmation to save*  IO.show\_inventory(table)  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*  objFile = open(file\_name, 'w')  **for** row **in** table:  lstValues = list(row.values())  lstValues[0] = str(lstValues[0])  objFile.write(','.join(lstValues) + '\n')  objFile.close()  **else**:  input('The inventory was NOT saved to file. Press [ENTER] to return to the menu.')  *# -- 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(file\_name, table):  """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:  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.  """  **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)  **else**:  input('canceling... Inventory data NOT reloaded. Press [ENTER] to continue to the menu.')  *# 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':  IO.load\_choice(strFileName, lstTbl)  IO.show\_inventory(lstTbl)  *# 3.3 process add a CD*  **elif** strChoice == 'a':  DataProcessor.append\_data(lstTbl)  *# 3.4 process display current inventory*  **elif** strChoice == 'i':  IO.show\_inventory(lstTbl)  *# 3.5 process delete a CD*  **elif** strChoice == 'd':  DataProcessor.delete\_data(lstTbl)  *# 3.6 process save inventory to file*  **elif** strChoice == 's':  FileProcessor.write\_file(strFileName, lstTbl)  *# 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-21 [↑](#footnote-ref-1)