Miroslava Meza

03/06/2022

IT FDN 110B Wi 22: Foundations of Programming Python

Assignment 06, Module 06

# Functions, Classes, Variable Scope, and DocString

## Introduction

Based on the starter script provided, based on last week’s assignment, I modified the script by adding functions that could be called. Those were divided in three classes (1) data processing, (2) file processing, and (3) input/output. This version allowed the use of functions to load existing data, to erase an entry, and to add new entries while saving the information into a .txt file.

## Github repository

This assignment is stored in two different places, (1) <https://github.com/Mirka3m-Sea/Assignment_06.git> and (2) CANVAS.

## Adding Classes and Functions

Classes are useful to organize functions that have similar characteristics. It took me a little of time to align the class name and functions with their final destination on the main script body. It was very useful to rewrite the entire code-line for calling functions and using the “autofill” aids.

*Challenge: Originally, I created orphan “call variables” that did not match with the functions they were related to. This led me to redundant code lines.*

*Solution: I eliminated those redundancies using the search tool and ensured that I was processing from the functions*.

Class DataProcessor: this class contained all the functions related to data entry and data deletion (see figure 1, look for the rows with red marker: 19, 24, 41.

Class FileProcessor: contains all the functions that require processing the txt file (CDInventory.txt) associated with the script.

Class IO: Organized and associated all functions that required user input or displayed outputs.

|  |
| --- |
| Figure . Class DataProcessor. Two functions add data, and delete data |

## Using DocStrings

This function was useful to remember what each function is supposed to accomplish. I went back to the written notes when testing.

*Challenge: Not sure when to use them outside of functions. Is it an acceptable practice to use this type of DocStrings instead of #?*

## Using Lists, Directories, and calling other Functions within a function.

When adding data, I used a directory to append information to a table (lstTbl). It was very useful to learn that functions can be nested within other functions (see figure 2, row 60).

Figure . Example of function that used lists, directories and nested another function

Text

Description automatically generated

## Running the Script in Spider

The first challenge I faced was that the script would yield an error message if I did not have a .txt file on the same repository. I eliminated that issue by opening/closing or creating/closing the txt file at the beginning of the script. That way, regardless of the sequence chosen by the user there will always be a destination file.

Testing sequence.

1. Started using the option [l] from the menu.
2. Option [a] adding one more inventory entry.
3. Option [w] to save the information to the .txt external file.
4. Option [d] to display the information.
5. Option [e] to erase an entry.
6. Option [d] to display the current inventory.
7. Option [exit] to close the script.

Figure 3. Loading data when there was not .txt file

Text

Description automatically generated

This script nested the function of displaying the inventory within the function of adding new entries. On figure 4, you can see that after each new entry, the script displays the final inventory. On figure 5, we can see how the data is displayed using the same function without being nested. Finally in figure 5, we can also see how an entry is deleted and the inventory is updated and displayed.

|  |  |
| --- | --- |
| *Figure 4 Adding new entries and displaying data. (Nested function)* | Figure . Displaying data |

Figure 6. Deleting an entry and saving the changes into a .txt file.

Text

Description automatically generated

## Running the Script in Terminal

Figure 7. Loading, adding, writing, and displaying data using the Script.

Text

Description automatically generated

Figure 8. Erasing an entry, displaying, and exiting the script using the Terminal.

Text

Description automatically generated

Figure . Saving to a txt file and exiting the script

Text

Description automatically generated

## Conclusions

This assignment had the goal of using Classes, Functions, and DocStrings. During this activity, we continued to use 2D data structures, both as lists and dictionaries. Based on the last week’s assignment started script, I modified the script and replaced the inner data structure with Classes and Functions. This script allowed the following functionality:

1. Started using the option [l] from the menu.
2. Option [a] adding one more inventory entry.
3. Option [w] to save the information to the .txt external file.
4. Option [d] to display the information.
5. Option [e] to erase an entry.
6. Option [d] to display the current inventory.
7. Option [exit] to close the script.

The main challenges were to create functions within logical classes and to call them in the appropriate order at the main script. At first, I had to double ensuring the data and file processing was done by the main script calling the functions; once I learned to use the autofill for each class it became easier to tidy up each option.

## Appendix Full Script

|  |  |
| --- | --- |
| 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 | *#------------------------------------------#*  *# Title: CDInventory.py*  *# Desc: Working with classes and functions.*  *# Change Log: (Who, When, What)*  *# DBiesinger, 2030-Jan-01, Created File*  *#Miroslava Meza, 2022-Mar-06, Included classes and functions*  *#Miroslava Meza, 2022-Mar-06, Eliminated TODO marks, pass, and redundant code lines*  *#------------------------------------------#*  *# -- DATA -- #*  strChoice = '' *# User input*  lstTbl = [] *# list of lists to hold data*  dicRow = {} *# list of data row*  strFileName = 'CDInventory.txt' *# data storage file*  objFile = **None** *# file object*  cdData =[**None**, **None**, **None**] *# Saving data in memory as a list, starts with no values.*  dicRow= {'ID': **None**, 'Title':**None**, 'Artist':**None**} *#start the dictionary with no values*  *# -- PROCESSING -- #*  **class** **DataProcessor**:  *#"""*  *# Action-1 add functions for processing here*  *#using the functions identified in the starter code as:*  *# 3.3.2, 3.5.2, 3.6.2.1*  **def** add\_data():  *"""*  *Add and save data entries*  *Using the list type variable named 'cdData'*  *variable names were already in dicRow*    *Args: intID, strTitle, stArtist, IO.cdData, IO.show\_inventory, lstable, dicRow*  *Returns: inventory list with new entry.*    *Note: read and write functions are on the next class.*  *"""*  intID, strTitle,stArtist =IO.data\_input()  dicRow = {'ID': intID, 'Title': strTitle, 'Artist': stArtist}  lstTbl.append(dicRow)  IO.show\_inventory(lstTbl)      **def** delete\_data(intIDDel):  *"""*  *This function allows to erase a data entry.*  *Args: lstTbl, intIDDel, intRowNr*  *IO.show\_inventory*  *"""*  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!')  *#TODO check this line*  IO.show\_inventory(lstTbl)    **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=strFileName into a 2D table*  *(list of dicts) table one line in the file represents one dictionary row in table.*  *Args:*  *strFileName (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.*  *\*\*\* file\_name will be tied to the variable strFileName*  *"""*    table.clear() *# this clears existing data and allows to load data from file*  objFile = open(file\_name, 'r')  **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):  objFile = open(strFileName, 'w')  **for** row **in** lstTbl:  lstValues = list(row.values())  lstValues[0] = str(lstValues[0])  objFile.write(','.join(lstValues) + '**\n**')  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**\t**CD Title (by: Artist)**\n**')  **for** row **in** table:  print('**{}\t{}** (by:**{}**)'.format(\*row.values()))  print('======================================')    **def** data\_input() :  *"""*  *This function request the user to input data for each CD.*  *Arg: strID, intID, strTitle, stArtist. All of these entries are held*  *in the internal memory of this function. THey are no global variables.*  *Returns: None*  *"""*  strID = input('Enter ID: ').strip()  intID= int(strID)  strTitle = input('What is the CD**\'**s title? ').strip()  stArtist = input('What is the Artist**\'**s name? ').strip()  *#TODO Mirka, to verify this*  **return** [intID, strTitle, stArtist]    *# 1. When program starts, read in the currently saved Inventory*  *"""*  *The script did not run without creating the file to store data. To avoid issues, I will start by*  *ensuring CDInventory.txt exists.*  *"""*  file\_name2=open('CDInventory.txt', 'w')  file\_name2.close()  *#Script starts*  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**\t**')  **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.*  *# 3.3 process add a CD*  **elif** strChoice == 'a':  *# 3.3.1 Ask user for new ID, CD Title and Artist*  *# DONE move IO code into function, Calling the add data function from DataProcessor*  DataProcessor.add\_data()  **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*  *# DONE moved processing code into function*  DataProcessor.delete\_data(intIDDel)  **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':  *# DONE Called the function write\_file from the Class "FileProcessor"*  FileProcessor.write\_file(strFileName, lstTbl) *# 3.6.2.1 save data*  **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') |