Danielle Rodriguez

06 March 2022

IT FDN 110 B

Assignment 05

Improving the CD Inventory

Introduction

This module covers functions and classes as well as docstrings. These tools allow the programmer to organize scripts better and make scripts easier for others to read. Functions allow the programmer to group statements and give the programmer numerous tools, including parameters and return values. Using these tools in the assignment allows the user to modify the provided script to add more functions.

Functions

Functions group statements, making them available to be called by a specific name. The function must be defined to be called. Once the function is defined, the code within the function can be executed. Parameters are useful tools within functions that allow the programmer to pass in values for processing. Return values are another useful tool that can be consumed instantly or assigned to a variable. Using return values allows the programmer to use the results repeatedly without having to call the function every time.

Using these elements, we can improve on the basic math script (Figure 1). The first half of the script defines the variables to be called in followed by several functions to calculate the various math problems using the values chosen by the user. After defining the functions, the script then calls the return values to populate the answers to each math equation.

Text

Description automatically generated

Text

Description automatically generated

**Figure 1: Working with functions**

Functions can also utilize multiple return values bundled into a collection. We can use multiple return values to simplify the basic math script, so all four calculations are done in one function. By doing so, the function calculates and returns all the calculations at once. The output is identical to Figure 1 but accomplishes the result in fewer lines.

Text

Description automatically generated

**Figure 2: Returning tuples**

Classes allow the programmer to group functions and organize the script. In Figure 3, the script expands on the basic math program to utilize classes to group math calculations. This code produces the same output as the other two examples but provides a way to keep data together and organize the script.

Text

Description automatically generated

**Figure 3: Creating a class of functions**

Improving the CD Inventory

In this assignment, I modified a code to include more functions and simplified the while loop to call the return values. The first figure in the appendix outlines the entire code for the CD Inventory. This script utilizes three different classes to group the various functions of the code. Starting at line 205, the code references the classes and the various return values to execute the menu options. The most challenging part of this assignment was identifying the parts of the existing code to add to the functions. Once I organized the script, I found that aligning the names of the functions and the return values caused challenges. I ran into multiple errors because I was not paying close attention when modifying the code. But by correcting the mislabeled functions, I was able to understand the script better.

Summary

This assignment expanded upon functions and introduced classes. These tools allowed me to modify the CD Inventory starter code to create more functions and utilize classes. Classes are a very powerful tool which allows the programmer to organize scripts effectively and group similar functions together.

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 | *#------------------------------------------#*  *# 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*  *#------------------------------------------#*  *# -- 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*  *# -- 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*    *"""*  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!')  **return** IO.show\_inventory(lstTbl)  @staticmethod  **def** write\_file():  *"""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*    *"""*  objFile = open(strFileName, 'w')  **for** row **in** lstTbl:  lstValues = list(row.values())  lstValues[0] = str(lstValues[0])  objFile.write(','.join(lstValues) + '**\n**')  objFile.close()  **return** 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 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*  objFile = open(file\_name, 'r')  **for** line **in** objFile:  data = line.strip().split(',')  dicRow = {'ID': data[0], 'Title': data[1], 'Artist': data[2]}  table.append(dicRow)  objFile.close()  @staticmethod  **def** write\_file(file\_name, table):  *"""Writes data to the file*  *Args:*  *objFile: calls the text file containing CD inventory*  *lstValues: row of values in 2D table*  *objFile.write: writes inputted data to file*  *lstTbl (list of dict): 2D data structure (list of dicts) that holds the data during runtime.*  *Returns:*  *Table of data written to file.*  *"""*  objFile = open(file\_name, 'w')  **for** row **in** lstTbl:  lstValues = list(row.values())  lstValues[0] = str(lstValues[0])  objFile.write(','.join(lstValues) + '**\n**')  objFile.close()  **return** IO.show\_inventory(lstTbl)  *# -- 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*    *"""*  strID = input('Enter ID: ').strip()  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*  intIDDel = int(input('Which ID woudld you like to delete? ').strip())  *# 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()  **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**

Text

Description automatically generated

Text

Description automatically generatedText

Description automatically generated

**Appendix 2: CD Inventory script output in spyder**

Text

Description automatically generatedText

Description automatically generatedText

Description automatically generated

**Appendix 3: CD Inventory script output in terminal**