# Introduction

For this assignment we were provided with pre-made script that had only incorporated a few functions. There were several TODO items listed in the code and our objective was to resolve these items and finish organizing the code. In short, we were required to edit the script, organize the code and to document our modifications throughout the script as well as to the header. Lastly, we were required to upload our finished modified code to GitHub for public review. Below are the details taken to complete the assignment.

# Assignment06

At the beginning of this assignment, I must admit I was somewhat intimidated by the sheer volume of lines of code that made up the script. And not just by the volume, but also by the fact I was required to modify this program – which, when ran (without any modification) . . . seem to run perfectly fine, at first.

There was this anxiety of: ‘you break it, you buy it’, staving me off mentally from jumping right into the work. However, after taking time to review the module 6 materials, chapter 6 in our course book and the additional links provided by our instructor beforehand, I began to skim through the code, evaluating each TODO item.

I hadn’t realized at first each item was exhibiting where a function was needed in relation to its class. My initial assumption had been, that where the script needed to be updated, that that section required its own individual and specific separate function. Thankfully, this was not the case.

Text, letter

Description automatically generatedFirst, I moved the processing code into a function defined as write\_file, shown below into the FileProcessor class section – where CDInventory program processes data to and from the stipulated text file: **Fig 1. – write\_file function with docstrings in code**

|  |
| --- |
| 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()  pass |

The function title: write\_file (as displayed above) was in the correct section. So really, all that was required from my end was to shift the embedded code from the script itself into the empty function. This was perhaps the easiest portion of the assignment, as the function write\_file was located below the function: read\_file, which gave me a clue as to how to proceed. In addition, docline notes provided more depth as to the separation of concerns. The next item was to add delete functionality.

I created the delete function much the same way as the write\_file function. However, this is where I ran into a grating and re-occurring error. I hadn’t realized at first each ‘class’ as defined in the script, coincided with separate functionality.

Going back to my initial hesitation due to the volume of code, I began jumping from section to section, stressing out about what I was supposed to do. I wasn’t taking time to read the docstrings of what’d been provided and therefore wasn’t comprehending what each class function does.

Perfect example: there’s two sections in the script provided for data processing – however, because I was jumping about too quickly, it hadn’t sunk in, one section was for Data Processing and the other was for File Processing. An important distinction for sure. More so, considering the fact the delete functionality for dicRows was supposed to be incorporated into the IO class section of the code, considering that was where the program: """Handles Input / Output""".

**Fig 2. – function call for lstTbl deletion with docstring**

Text

Description automatically generatedInstead, I had placed my delete function in the topmost section for DataProcessing – an error that’d take me some time to correct.

Eventually, I realized my mistake and started paying closer attention to the class definitions where I was creating each function, more so than just on functionality.

Again, as detailed in **Fig 2.**, once I determined the correct class, it was just a simple matter of shifting the code into the function so it could be called later.

The last section of the assignment was by far the most difficult, as I attempted created a function that would add a row to the 2D data structure in our list of dictionaries.

Text

Description automatically generatedAt first my assumption in creating this specific functionality had been determining which: ‘class’ the script should fall under.

As an fyi, I’m aware the pseudocode within the starter script had shorn this specific script here in **Fig 3.** into two separate functions, one for IO and the other in one of data processing classes.

**Fig. 3 – newDets() function with docstring detailing functionality**

However, for the sake of brevity and ease of access, I decided to make one function that’d take user input for the dicRow to be appended to the lstTbl. The reason for this is pretty straightforward.

I experimented quite a bit with the placement of what should’ve been two separate functions for adding to the lstTbl. I tried placing one part of the script in the IO class – then another in the FileProcessing and DataProcessing sections. However, each time I placed one section of script as shown in Fig 3. in a different class as part of its own separate function, I would get an error stating that strID and even int(strID) hadn’t yet been defined, as well as warning that although these variables were present in the So if wasn’t possible to have two separate functions contributing together to add a new dictionary to the lstTbl, I did the next best thing. I simply took the whole script where a user could input the ID, Title and Artist of a CD and append it to lstTbl, and just created a function in the DataProcessing class where it could be called in the main body.

And just to show that it works, here’s the completed assignment, working in both Spyder and CMD:

A computer screen capture

Description automatically generated with low confidenceText

Description automatically generated

**Fig 4. – SUCCESS – Assignment 6 code running in Spyder and CMD**

# Conclusion

With the script modified and code organized appropriately, the last thing to be done is to detail my involvement with docstring as well as to the header. Beforehand, dostring and pseudocode had been incorporated - however, trying to determine where the script had been failing was made harder reading through the additional code. Regardless, the program now works in both CMD and Spyder and all prerequisites for the assignment have been reached. At this point, where several beginners would likely quit, I’m very proud of the fact that, even though to an advanced programmer this probably isn’t the most significant script, that I’ve been able to read, understand, edit and execute it.

# Index

**Fig 1. – write\_file function with docstrings in code**

**Fig 2. – function call for lstTbl deletion with docstring**

**Fig. 3 – newDets() function with docstring detailing functionality**

**Fig 4. – SUCCESS – Assignment 6 code running in Spyder and CMD**