Kimberly Brock

February 21, 2021

Foundations of Programming (Python) – IT FDN 110 A

Assignment06

GitHub: [kb1981/Assignment\_06 (github.com)](https://github.com/kb1981/Assignment_06)

Introduction

This week was mostly focused on Functions. We went over the modules that went into depth on the uses of functions and how to create them. We read in the book about how to use functions and had a sassy Tic-Tac-Toe game as a demonstration. We went through a function tutorial and watched a video on functions in a couple of supplemental websites. Finally, we put all our knowledge together and updated a starter script to include functions.

Module 06 – Functions ():

Module 06 started out by recalling Functions and how to define and call them in python. We learned the parameter allows you to pass in a value to processing a function (parameter = argument). The return expression can be used to return a value of a variable processed in a function. This is useful if you want to use this value again, but do not necessarily want to keep calling the function. We also discussed naming conventions and what is commonly used. We typically use naming conventions as a standard across programming so other programmers can recognize and quickly understand your code. An interesting fact about functions in python is that they are automatically packages into tuples. So, when working with functions of multiple return values, you can use packing and unpacking how we learned in our tuple sections. We also learned the difference between a positional argument and a name / keyword argument. With positional arguments, it matters the order in which you call them in the function. In name / keyword arguments, the order in which you call them in the function does not matter. You can use both positional and keyword arguments in the same function, but the positional argument must always be listed before the keyword arguments. We learned that setting default values can be useful in some situations, so that the program does not crash if it receives something that ‘doesn’t work’. Must be careful with default values and make sure you are still getting the solutions you are expecting. Next, we went over Overload functions and how to use these in Python using default values. We learned that the keyword None means that the variable has no value, whether it be a number or a string, nothing it there. We reviewed reference type data, and how they can be useful since they are faster and use less memory. We went over variable scopes and the differences between local and global variables. Basically, unless you label your variable as global, if it is used during a function, it is only used within the block of that function, not outside. It is not good practice to have changing global variables or to shadow global variables, can get very confusing. Docstrings, however, is a good practice. Docstrings are used to give a description of your function and how it can be used. Lastly, we went over classes, which we will learn more about in Module8.

*Module 06 can be found at the link:* [*https://saravji.github.io/saravjis\_hut/FDN\_Prog/Modules.html*](https://saravji.github.io/saravjis_hut/FDN_Prog/Modules.html) *and the Lab06\_A, B, C are shown in Appendix A.*

Python Programming for the Absolute Beginner, Third Edition

This week in the book *Python Programming for the Absolute Beginner, Third Edition*, By Michael Dawson I read through. “Chapter 6–Functions: The Tic-Tac-Toe Game.” The purpose of this chapter was to introduce Functions. Functions help organize your code manageable chunks, and then call on that code as needed. We learned about Abstraction and Encapsulation. Abstraction is seeing the forest without worrying about the trees, and encapsulation is knowing there are trees there but not how many. The chapter discussed positional parameters and positional arguments. In the positional parameter and arguments, the position matters of your variables matter when you call the functions. Also discussed were positional parameters with keyword arguments. Using positional parameters used with keyword arguments allows you to call a function with variables in any order but will display (work) in the right places, making the code a little clearer. Global variables were discussed and the uses for them. We learned Global variables are best used when they are used as constants. In addition to the text, I really liked the author’s Tic-Tac-Toe program and the sassy AI.

Additional Webpages

1. [Functions - Learn Python - Free Interactive Python Tutorial](https://www.learnpython.org/en/Functions)

The link above goes to a online hands-on tutorial showing how to build and call simple functions. I really like the hands-on tutorials, learn a lot from those!

Additional Videos

1. Reading from a text file: <https://www.youtube.com/watch?v=_ypAw_pCOt8&feature=youtu.be>

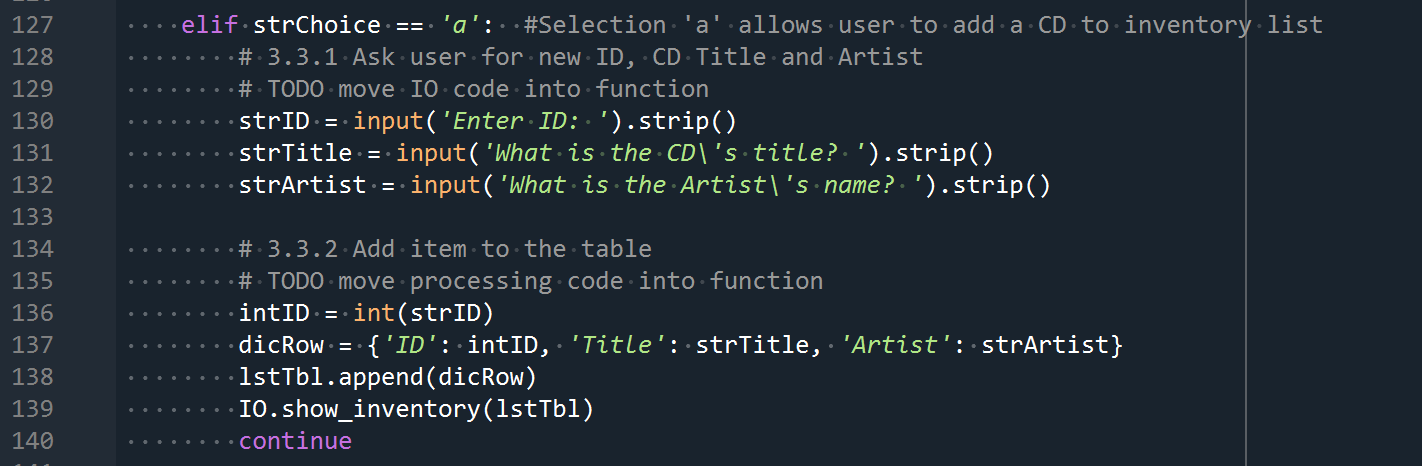
This tutorial went through the basics of creating a function in Python. He walks though how using functions can makes it easy to reuse code. He also walks through how you can make your function custom for the information you are trying to display. *TIP: non-default variable cannot be after default variable!*

Applying Knowledge – CDInventory.py, *enhanced*

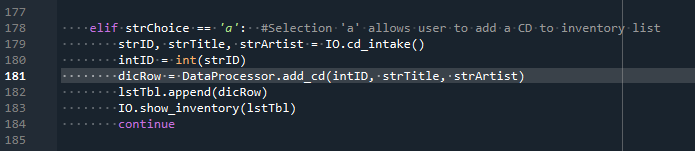
This assignment took CDINventory.py from executing actions in the While loop to calling functions outside the loop to execute actions and return them. We had three different classes to put the functions in: DataProcessor, File Processor, and IO. The TODO items we had to update included adding a CD to the inventory, deleting a CD from inventory, and saving the CD Inventory to a text file. The program was then tested in both Spyder and Terminal

***Updating Adding a CD to inventory***

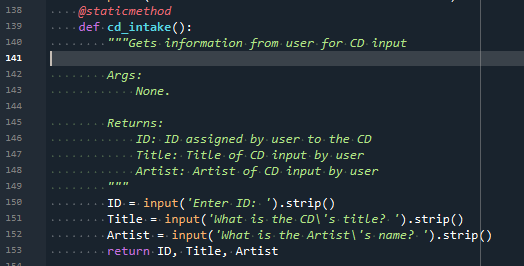
The TODO was to take the script of adding a CD to the CD Inventory from done in the loop to done in a function.



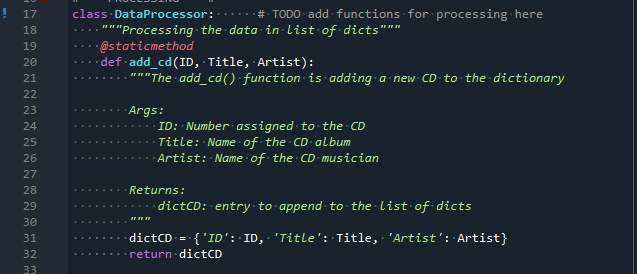
Listing . Adding a CD to Inventory in the Loop (original script)



Listing . Adding a CD to Inventory calling Functions IO.cd\_intake and DataProcessor.add\_cd



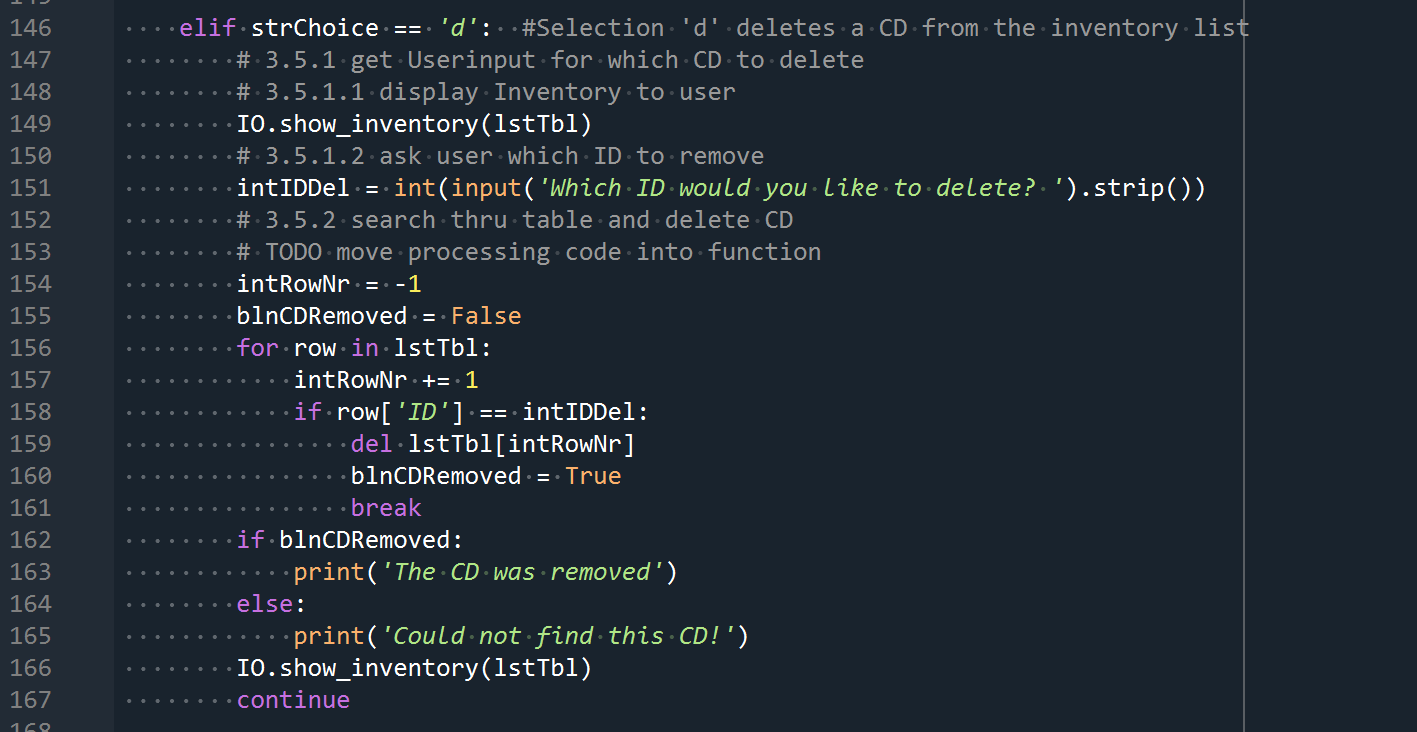
Listing . IO Function to Get User Input (cd\_intake)



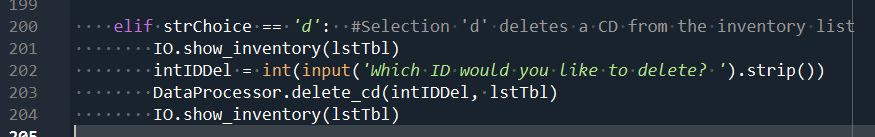
Listing . Class DataProcessor Function to Add CD to List of Dictionaries (add\_cd)

***Deleting a CD from the Inventory***

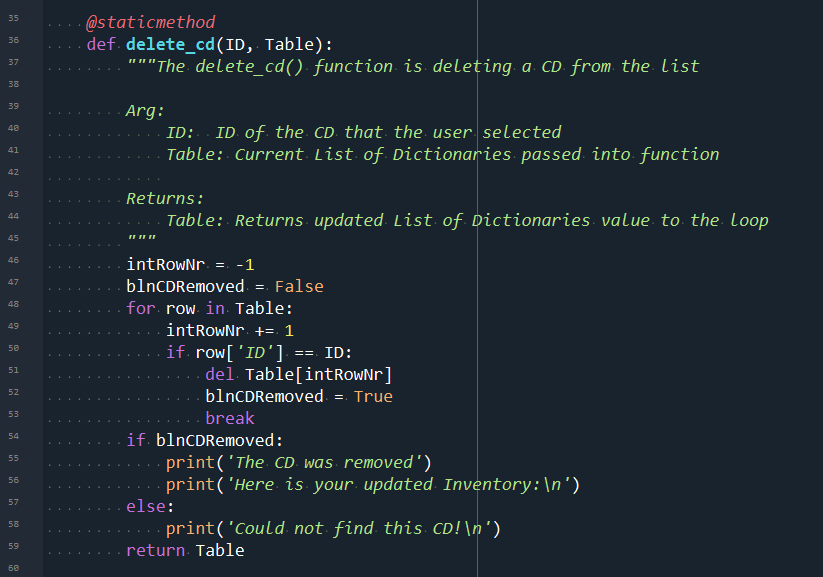
The TODO was to move the action to delete a CD from the inventory to a function in the class DataProcessor



Listing . Deleting CD Inventory in the Loop (original script)



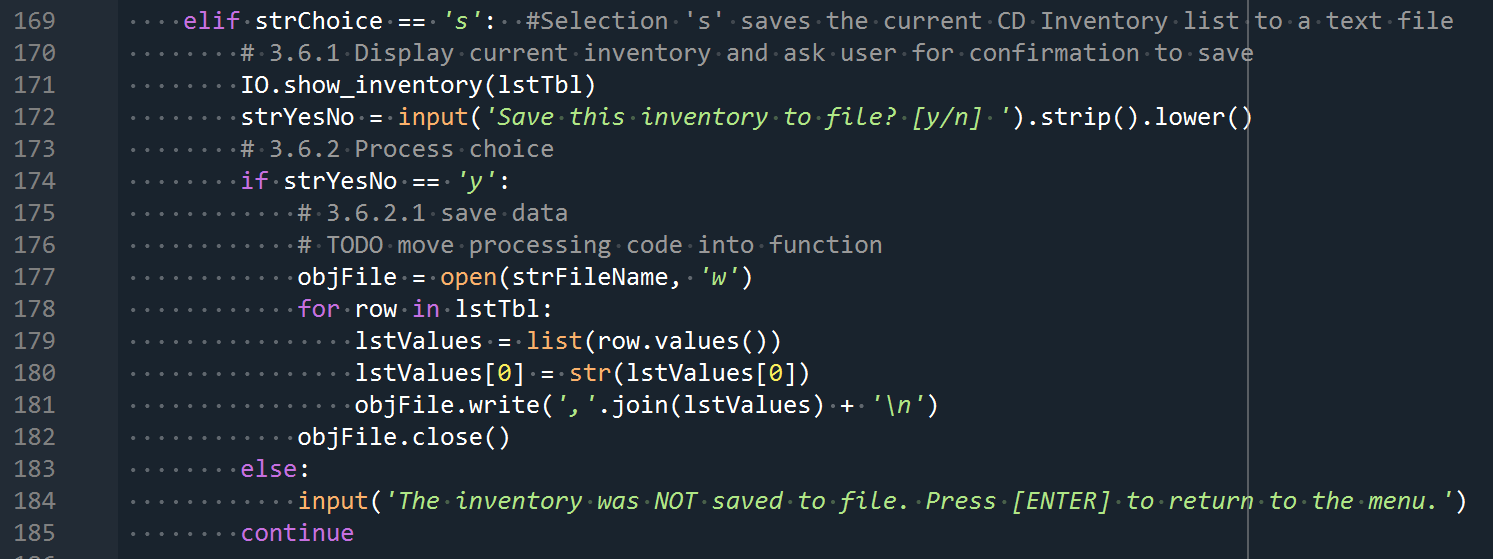
Listing . Deleting a CD from Inventory Calling Function DataProcessor.delete\_cd



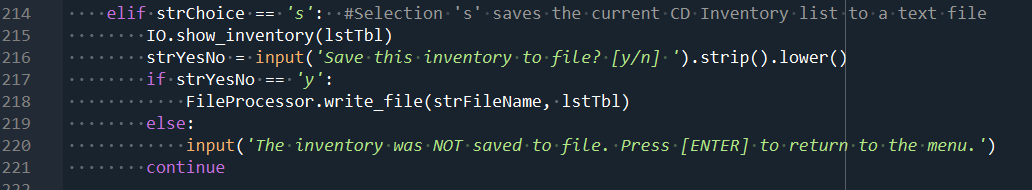
Listing . Class DateProcessor function to Delete a CD from the List of Dictonaries (delete\_cd)

***Saving the CD Inventory to a Text File***

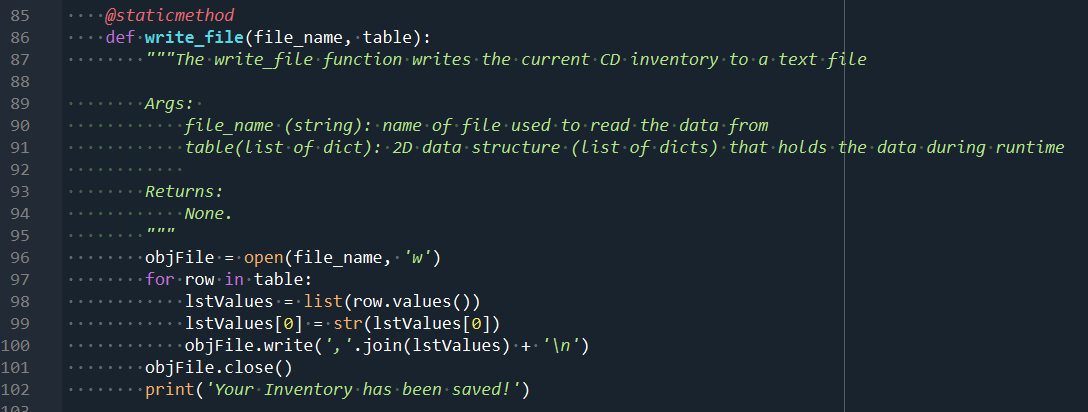
The TODO for this was to take the action of saving a file from the loop into its own function in the class FileProcessor



Listing . Saving CD Inventory in the Loop (original script)



Listing . Saving a CD to Inventory Calling Function FileProcessor.write\_file



Listing . Class FileProcessor function to Save the CD Invenotry to a Text Document (write\_file)

***CDInventory.py Working in Spyder***

|  |  |
| --- | --- |
| Figure . [l] Load Inventory from file option (Spyder) | Figure . [a] Add CD (Spyder) |

|  |  |
| --- | --- |
| Figure . [i] Display Current Inventory (Spyder) | Figure . [d] Delete CD from Inventory (Spyder) |

|  |  |
| --- | --- |
| Figure . [s] Save Inventory to File (Spyder) | Figure . CDInventory.txt File Updated |

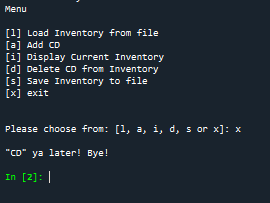


Figure . [x] Exit (Spyder)

***CDInventory.py Working in the Terminal***

|  |  |
| --- | --- |
| Figure . [l] Load Inventory from file (Terminal) | Figure . [a] Add CD (Terminal) |

|  |  |
| --- | --- |
| Figure . [i] Display Current Inventory (Terminal) | Figure . [d] Delete CD from Inventory (Terminal\_ |

|  |  |
| --- | --- |
| Figure . [s] Save Inventory to file (Terminal) | Figure . CDInventory.txt File Updated |

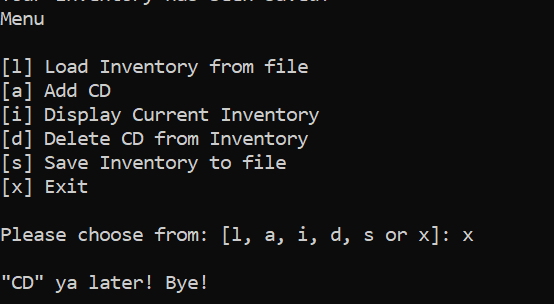


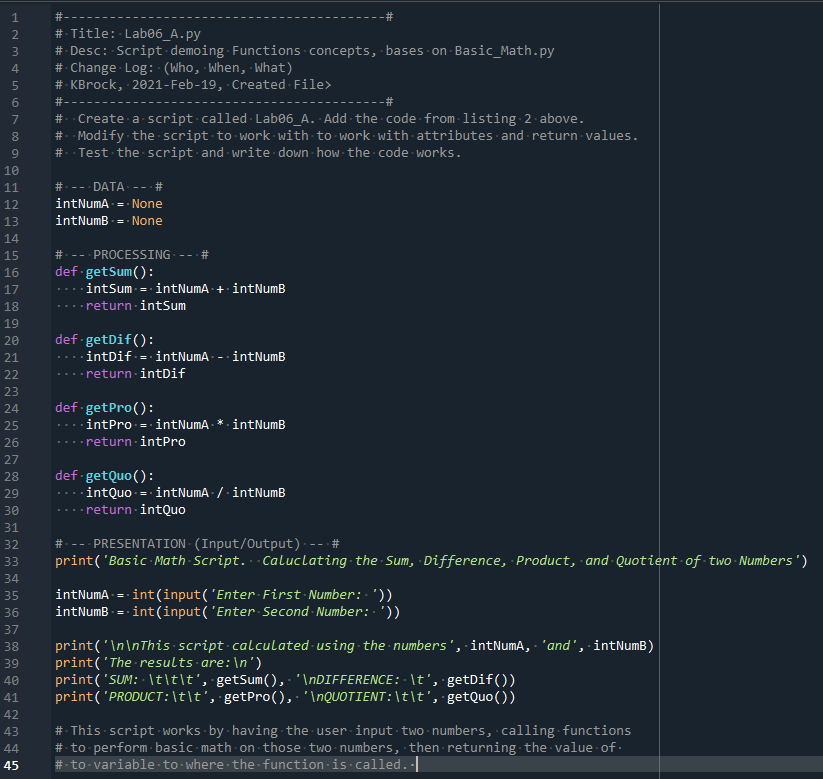
Figure . [x] Exit (Terminal)

Summary

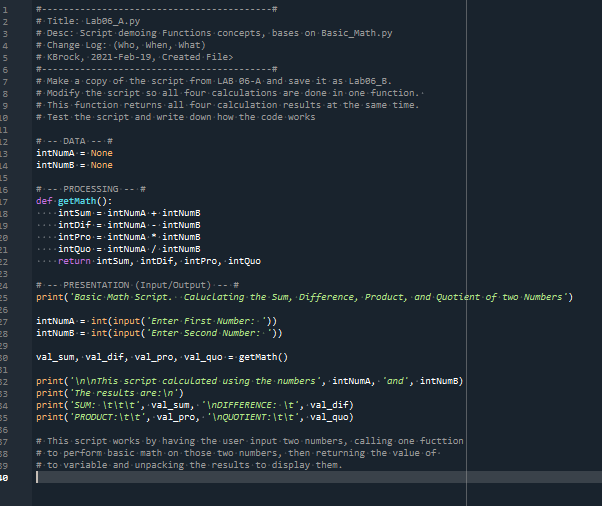
For week six, we learned about how to create and call a function. We learned that functions are blocks of code that can be reused. Because of the reusability, docstrings should be used to help other programmers understand the purpose of the function. This week seemed to be a little less complicated than last week (fingers crossed that is reflected in my script). I do understand the basics of how to pass values to a function for processing and then returning those values back to the main loop.

Appendix

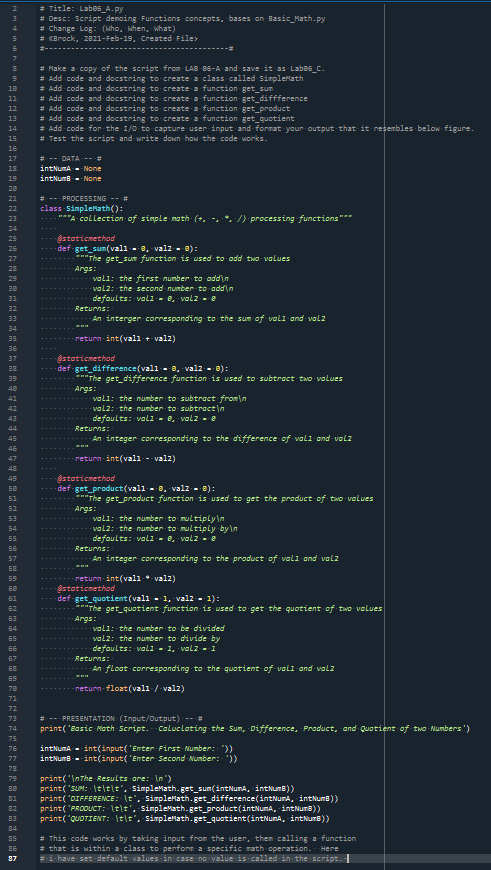
*Appendix A. Module 06 Lab Listings*



Appendix A Listing . Lab06\_A.py



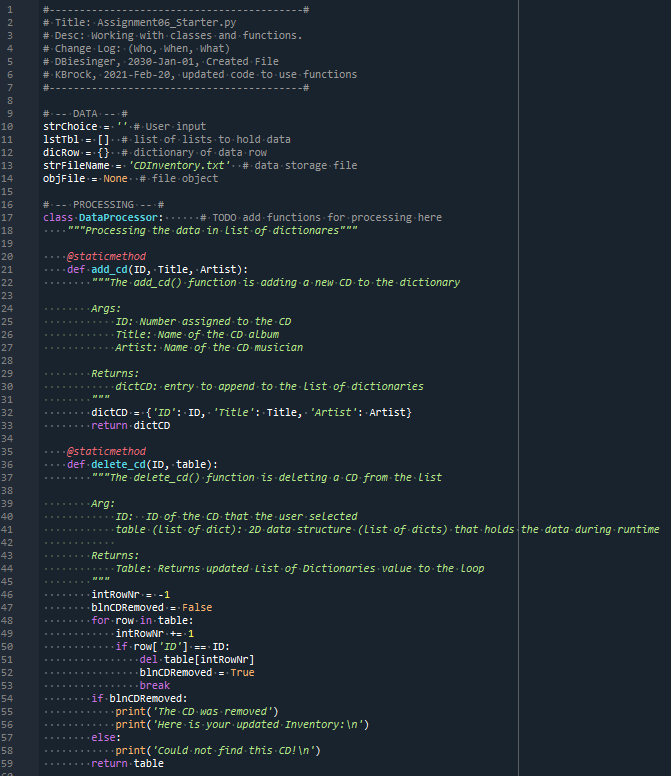
Appendix A Listing . Lab06\_B.py



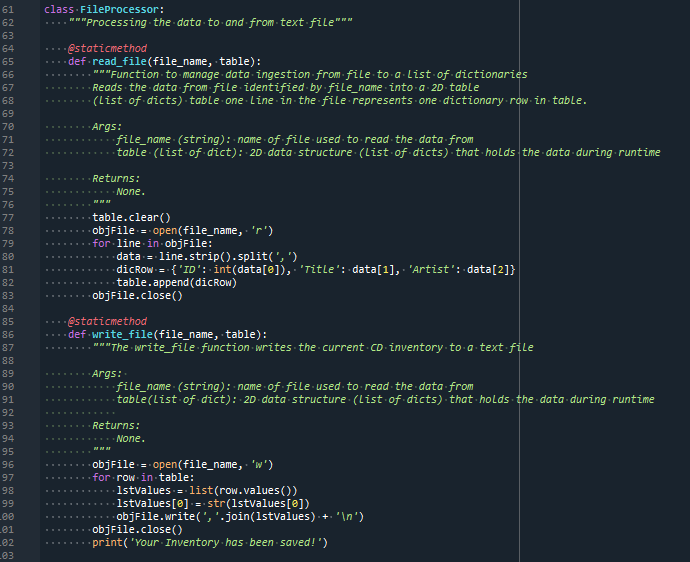
Appendix A Listing 3. Lab06\_C.py

*Appendix B. Assignment04 – CDInventory.py*

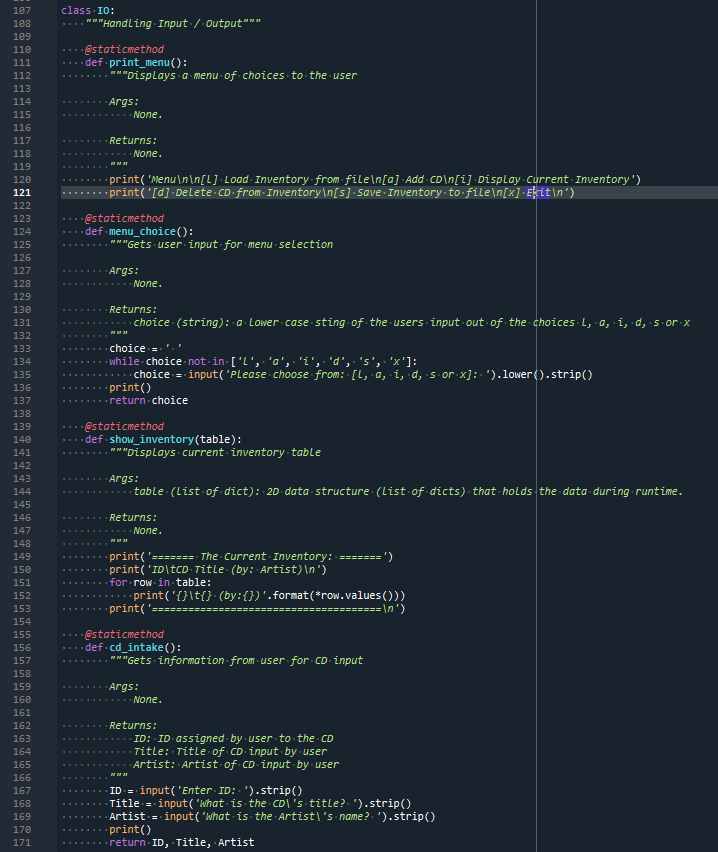
Assignment uploaded to GitHub: <https://github.com/kb1981/Assignment_06>



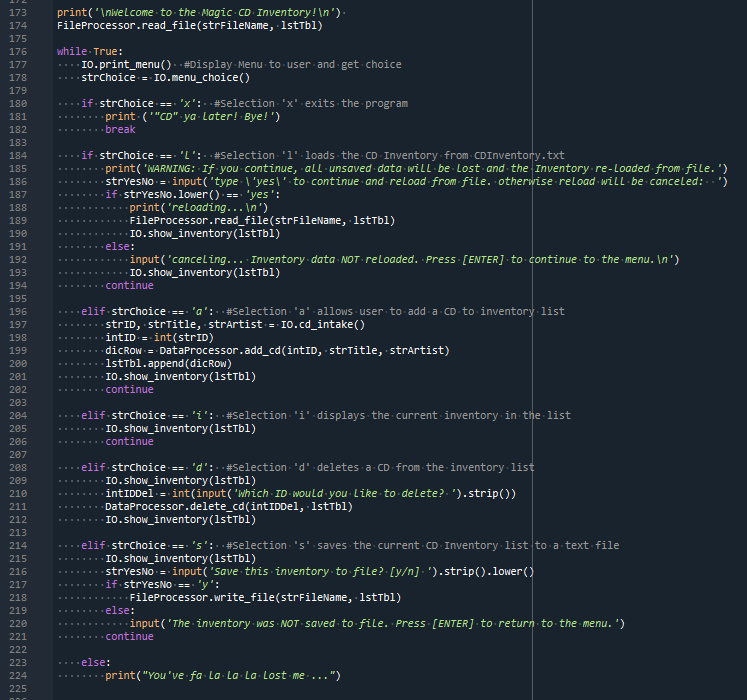
Appendix B Listing . Class DataProcessor and internal functions



Appendix B Listing . Class FileProcessor and interal functions



Appendix B Listing . Class IO and internal functions



Appendix B Listing . While Loop script and Function calls