This piece of work is my submitted response to the module COA122 Introduction To Programming Coursework 2019.

# Use of the System:

My solution uses a combination of both a graphical user interface and the command line. When the menu.py module is first run, the main menu for the library system will appear in a GUI window. However, this GUI only displays the menu, and all input / output is handled by the command line. When generating the graph of book popularity, this graph appears in a new GUI window. Had I of had more time and knowledge of the Tkinter interface, I would have implemented both the input / output and graph into the main menu window.

# Areas of Interest:

When writing my solution, I ensured that I refactored it such that I did not have any repeated code, making my solution more efficient overall. Before refactoring, I had 3 very similar occurences of the WriteData function, so I decided to put this common function into the database.py module and alter it so that it takes parameters of the filename, file heading and line length. I then stored these variables globally in each of the modules that needed to call this function, meaning that this function only needs one occurrence in my solution. Similarly, I have done this for the ExtractData function as well, as before refactoring, this had multiple occurences as well. I have also used a bubble sort in booklist.py, in order to generate the graph in descending popularity order.

# Testing

I have provided tests for a range of different modules and functions within my solution. Most notably, these tests have been provided in the database.py file, although there are tests in other modules too. These tests been provided at the bottom of the modules.

The IsBookAvailable test is an example of a fragile test that I have had to create. When writing my tests, I have assumed that the book with the ID of 1 is checked out, and that the book with the ID of 2 is available. If these two cases remain a constant throughout the runtime of the program, the tests will pass. However, there are no restraints on whether or not the librarian can checkout or return either of these books. Hence, if this happens, the tests will fail. This makes them fragile because somebody using the system could affect the outcome of the tests.

It should also be noted that I have not written any tests for the WriteData function. This is because this is the function responsible for writing data to the text files. Hence, implementing a test for this function would result in creating a destructive test, as this test would change the file every time that it is run.

I have not written tests for any of the functions that are responsible for retrieving user input either. This is because I am unsure on how to implement unit tests for functions responsible for user input. Instead, I carried out several manual tests to ensure that my solution works accordingly to the specification requirements, which it does.