CP1404 – Assignment 2 – Books to Read 2.0

Task:

Create both a console program and a Graphical User Interface (GUI) program similar to your first assignment, using Python 3 and the Kivy toolkit, as described in the following information and accompanying screencast video. This assignment will help you build skills using **classes** and GU[illegible]ptions, lists, JSON file handling and functions.

Everything you need [illegible]signment can be found in the subject materials. You should find the pr[illegible] helpful, and you are expected to follow the provided style guide: h[illegible]CP1404/Starter/wiki

Start your work by clic[illegible]ate a new repository in GitHub classroom:

https://classroom.github.com/a/Pd38u1HJ

Do not use any other repo or a copy of this one... just use this repository!
This will give you a new repo containing starter files including a README for your project, all of which you must use. Do not add any other files in this project, and do not rename anything - just use this as your assignment repo. Do not "download" this repo, but rather *clone* this repo using PyCharm ("Get from VCS"). You will probably need to create and use a **token** to access your private repository. Do not make your repository public.

Classes:

The most important learning outcome of this assignment is using **classes** to create reusable data types that simplify and modularise your programs. Using these classes in both a console and a GUI program highlights this modularity.
***Create these classes first* – before any code that requires them**. This is good practice. You should write and then test each method of each class, one at a time, committing as you develop, e.g., you might commit each time you complete a method and its tests.
We will assess your Git history to see that you do these in an appropriate order, so make sure you write your classes with tests, then the console program, *before* starting the GUI.

The starter code includes two files (test_book.py and test_bookcollection.py) with incomplete code for testing your classes. **Complete** these files with simple tests, that you write as you develop your Book and BookCollection classes.
Do not change the existing tests… write code that makes these tests pass.
You may use **assert** as shown in lectures, or just very simple tests that print the results of calling the methods you are testing with expected and actual results.
Once you have written and tested your classes, use them in your console program.

- Complete the **Book** class in book.py. This should be a simple class with the required attributes for a book and the methods:
  - __init__
  - __str__
  - two (not one) methods to mark the book as completed or unread
  - determine if the book is considered long (long books are >= 500 pages)

- Complete the **BookCollection** class in bookcollection.py. It should contain a *single* attribute: a list of Book objects, and at least the following methods:
  - add book – add a single Book object to the books attribute
  - get number of unread pages
  - get number of completed pages
  - load books (from JSON file into Book objects in the list)
  - save books (from book list into JSON file)
  - sort (by the key passed in, then by title)

Do not store additional attributes in this class, e.g., the number of books, because this information is easily derived from what you do store.

Console Program:

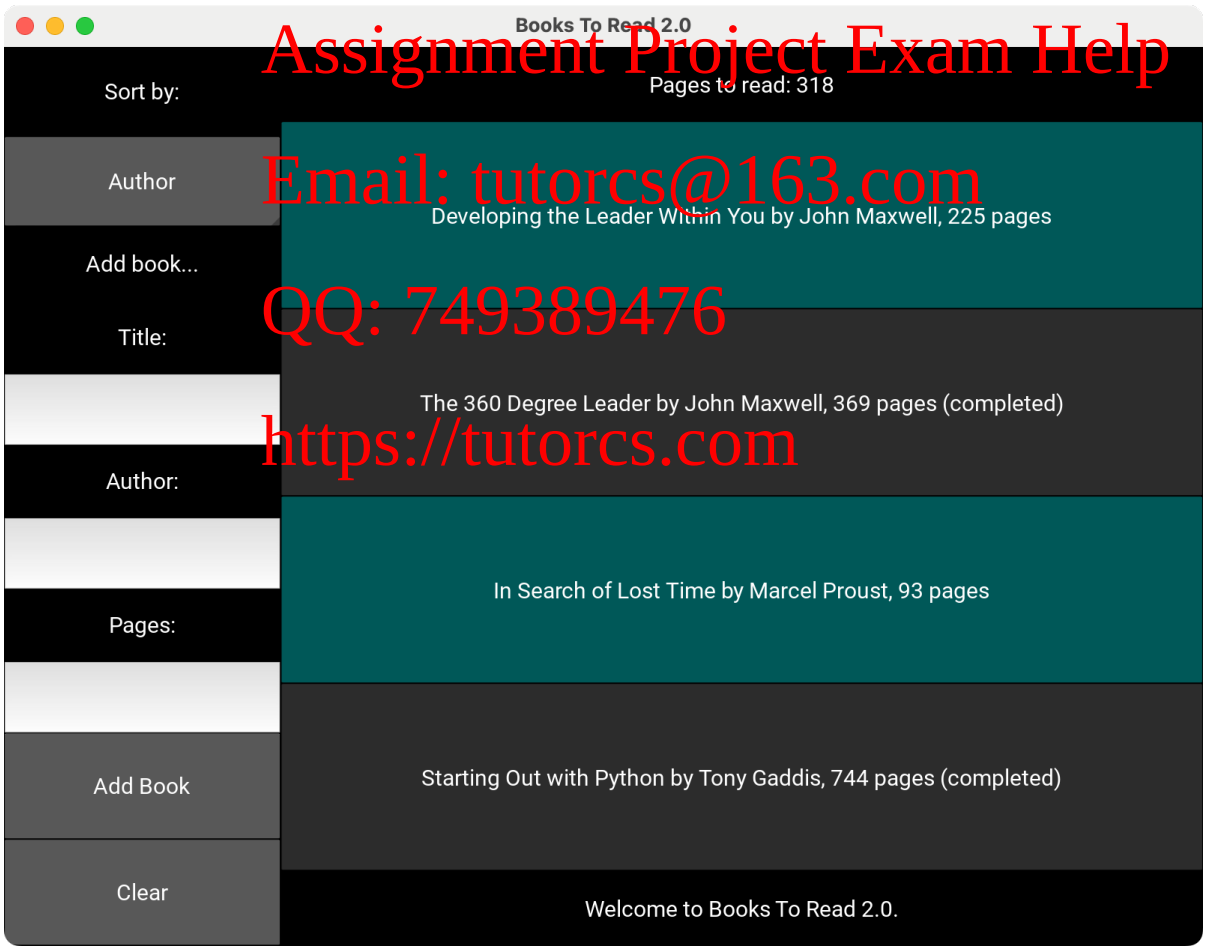
After you have written and tested your classes, rewrite your first assignment to use them. Start by copying the code from your first assignment into the provided a1_classes.py file and committing. In the first book was stored as a list. Modify your code so that each book is stored as new Book class.

You do *not* need to re gnment in any other way, even if it had problems, unless this is needed t e classes properly. We will only evaluate how you use the classes in the

GUI Program:

Complete the classes and console program before you start the GUI.

In the past, students have written a working GUI program without any classes, missing the main point of the assignment, and scoring very low marks.



Ensure that your program GUI has the following features and functionality, as demonstrated in the provided screencast video.

- Complete the main program in a Kivy App subclass in the provided main.py.
- The program should start by loading a JSON file, books.json as provided. This must be done within a method of your main app class using the appropriate method from your BookCollection class.
- The books file must be saved when the program ends, updating any changes made by the user. See the on_stop method from KivyDemos.

- The left side of the GUI screen contains a drop-down "spinner" for the user to choose the book sorting (see spinner_demo from KivyDemos), and text entry fields for inputting information for a new book.
- The right side contains buttons for the books, colour-coded based on whether they are completed or not (the colour scheme is up to you, but the colours must be noticeably different).
- The status label at the top of the right side shows the number of pages still to read.
- The status label at the bottom of the right side shows messages about the state of the program, including when a book or other button is clicked on.
- When the user clicks a book button, the state of the book changes between completed and not completed. The messages are different for long books.
  See guitars_app from KivyDemos for an example of using Kivy with custom objects associated with buttons.
- The user can add a new book by typing in the input fields and clicking "Add Book".

Adding Books (Error Checking):

- All book fields are required. If any field is left blank, the bottom status label should display "**Please complete all fields.**" when "Add Book" is clicked.
- The number of pages field must be a valid integer. If this is invalid, the status label should display "**Please enter a valid number**".
- The number of pages must be greater than zero, otherwise the status label should display "**The book must have some pages!**".
- Pressing the Tab key should move between the text fields. See popup_demo from KivyDemos.
- When the user successfully adds a book, the text fields should be cleared, and the new book button should appear in the books list on the right. See dynamic_widgets from KivyDemos.
- When the user clicks the "Clear" button, all text in the input fields and the status label should be cleared.

See the screencast video for a demo of this in action.

Coding Requirements:

1. At the very top of your main.py file, complete the docstring containing your details.
2. Document all your classes and methods clearly with docstrings. Include inline/block comments as appropriate. You do not need comments in the kv file.
3. Make use of named constants where appropriate. E.g., colours could be constants.
4. Use functions/methods appropriately for each significant part of the program. Remember that functions should follow the Single Responsibility Principle.
5. Use exception handling where appropriate to deal with input errors. When error checking inside functions (e.g., a handler for clicking the Add button), you should consider the "Function with error checking" pattern.
6. Complete your GUI design using the kv language in the app.kv file. Creating the book buttons should be done in main.py, not in the kv file, since this will be dynamic. See dynamic_widgets from KivyDemos.
7. Use the json library to load and save the JSON file, as taught in the subject.
8. Follow the style guide: https://github.com/CP1404/Starter/wiki/Style-Guide

Project Reflection:

It is important and beneficial for you to start developing good coding and working practices, so you will complete a short but thoughtful reflection on this project. Complete the template provided in the README and reflect on what you learned regarding both coding and your development process. **This is worth significant marks, so allocate significant time to it.** We expect answers that show some **detail** and **thought**, not just trivial statements.

Version Control:

You must use Git version control with your project stored in the private repository on GitHub that will be created when you accept the GitHub Classroom invitation above. Please ensure you use Git properly as taught: Do proper commits with good messages locally, then push your changes to GitHub. Do not create commits directly via the GitHub site. You are assessed on your use of version control including commits and commit messages, using the **imperative voice** (like "Add X" not "Added X").

Integrity:

The work you submit f must be your own. Submissions that are detected to be too similar to tha t or other work (e.g., code found online or created with AI) will be dealt w College procedures for handling plagiarism and may result in serious p

The goals of this assignment include helping you gain understanding of fundamental programming concepts and skills and future subjects will build on this learning. Therefore, it is important that you develop these skills to a high level by completing the work and gaining the understanding yourself. You may discuss the assignment with other students and get assistance from your peers, but you may not do any part of anyone else's work for them and you may not get anyone else to do any part of your work. Note that this means you should **never give a copy of your work to anyone or accept a copy of anyone else's work, including looking at another student's work or having a classmate look at your work**. If you require assistance with the assignment, please ask **general** questions in the discussion forum, or get **specific** assistance with your own work by talking with subject staff.

The subject teaching contains all the information you need for this assignment. Do not use online resources or AI, e.g., Google, Stack Overflow, ChatGPT, Copilot, etc. for assistance because this would limit your learning and would mean that you would not achieve the goals of the assignment - mastering fundamental programming concepts and skills.

Submission:

Submit your assignment on LearnJCU under Assessment.
Type your assignment GitHub URL in the submission text box.
Attach the following files to your submission:

- All Python files
- README
- A zip file of your project

Due:

Submit your assignment by the date and time specified on LearnJCU. Submissions received after this date will incur late penalties as described in the subject outline.

Sample Output:

Study the **screencast** provided with this assignment to see how the GUI program should work, including what the messages should be and when they occur.
The console program should look identical to the requirements for Assignment 1; only the implementation has changed.

Marking Scheme:

Ensure that you follow the processes and guidelines taught in class in order to produce high quality work. Do not just focus on getting the program working. This assessment rubric provides you with the characteristics of exemplary down to very limited work in relation to task criteria covering the outcomes:

- SLO1 - develop and utilise best-practice coding techniques to develop solutions
- SLO2 - select and apply appropriate and efficient data structures
- SLO3 - manage software projects using version control

Criteria	Exemplary (9, 10)	Satisfactory (5, 6)	Limited (2, 3, 4)	Very Limited (0, 1)
Project reflection SLO1, 2, 3 ***12%***	The project reflection is complete and describes development and learning well, shows careful thought, highlights insights made during code development.	Project reflection contains some good content but is insufficient in coverage, depth or insight.	Exhibits aspects of exemplary (left) and satisfactory (right)	Many aspects of the project reflection are missing or could be improved.
Use of version control SLO3 **10%**	Git/GitHub has been used effectively and repository contains a good number of commits with good messages that demonstrate incremental code developme **starting with classes and testing then console before GUI**.	Git/GitHub used but several aspects of the use of version control are poor, e.g., not enough commits, or meaningless messages that don't represent valuable incremental development in an appropriate order.		Git/GitHub not used.
Console program SLO1 **8%**	Classes used correctly in console program.	Classes used in console program but not correctly.		Classes not used in console program.
Error handling SLO1 **5%**	Errors are handled correctly and robustly as required.	Some errors are handled but not all, or errors are not handled properly.		No reasonable error handling.
Correctness SLO1 ***16%***	GUI layout is correct and program works correctly for all functionality required.	Aspects of the GUI layout are incomplete or poorly done or there are significant problems with functionality required.		GUI layout is very poor or not done. Program works incorrectly for all functionality required.
Identifier naming SLO1 **10%**	All function, variable and constant names are appropriate, meaningful and consistent, following style guide.	Several function, variable or constant names are not appropriate, meaningful or consistent, not following style guide.		Many function, variable or constant names are not appropriate, meaningful or consistent, not following style guide.
Use of code constructs SLO1, 2 **14%**	Appropriate and efficient code use, including no unnecessary duplication, good logical choices for control and storage, good use of constants, no global variables, good use of functions in main app, etc.	Several problems, e.g., unnecessary duplication, poor control, no use of constants, poor use of functions in main app.		Many problems with code use. **Any use of global variables.**
Use of classes and methods SLO1, 2 **14%**	Classes and methods are used correctly as required. Method inputs and outputs are well designed.	Some aspects of classes and methods are not well used, e.g., methods not used where they should be, problems with method/parameter design, incorrect use of objects.		Classes and methods used very poorly or not used at all.
Commenting SLO1 **6%**	Code contains helpful # block comments, all classes and methods have meaningful docstrings and main module docstring contains all details.	Comments are reasonable, but missing docstrings or block comments, and/or there is some noise or missing details in main module docstrings.		Commenting is very poor or not done.
Formatting SLO1 **5%**	All formatting is appropriate, including indentation, horizontal spacing and vertical line spacing. PyCharm shows no formatting warnings.	Problems with formatting reduces readability of code. PyCharm shows multiple formatting warnings.		Readability is poor due to formatting problems. PyCharm shows many formatting warnings.