XUT CP1404 Assignment 2 – Song List 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 **GUIs** as well as giving you more practice using techniques like selection, repetition, exceptions, lists, file I/O and functions.

Everything you need to complete this assignment can be found in the subject teaching.

You will be given starter code files, which you must use. A substantial amount of code has been provided to get you started. All of the starter code is good and should be used.

Classes:

The most important learning outcome of this assignment is to be able to use **classes** to create reusable data types that simplify and modularise your programs. The fact that you can use these classes in both console and GUI programs highlights this modularity.

It is important that you ***create these classes first* – before any code that requires them**. This is good coding practice. You should write and then test each method of each class – **one at a time**.

The starter code includes two files (test_song.py and test_songcollection.py) with incomplete code for testing your classes. **Complete these files with simple tests**, that you write as you develop your Song and SongCollection 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, you can then use them in your console program.

- Complete the **Song** class in song.py. This should be a simple class with the required attributes for a song and the standard methods: __init__ (constructor), __str__ (used when displaying song details in the status message), and:
    - two (not one or zero) methods to set the song as learned or unlearned.

- Complete the **SongCollection** class in songcollection.py.
  It should contain a *single* attribute: a list of Song objects, and the following methods:
    - add song – add a single Song object to the song list
    - get number of unlearned songs
    - get number of learned songs
    - load songs (from JSON file into the list of Song objects)
    - save songs (from song list into JSON file)
    - sort (by the key passed in, then by title)

  Do not store additional attributes like the number of songs, because this information is easily derived from the list of songs.

Console Program:

After you have written and tested your classes, rewrite your first assignment to use your new Song and SongCollection classes.
Start by copying the code from your first assignment into the existing a1_classes.py file.
In the first assignment, each song was stored as a list. Modify your code so that each song is stored as an object of your new Song class.
In the first assignment, the songs data file was a CSV. Modify your code to use the JSON file.

You do *not* need to rewrite your first assignment in any other way, but you do need to get it working. We will only evaluate how you use your classes in the console program.

GUI Program:

(The display and functionality explained here is also shown in the screencast video demo.)

- Complete the main program in the provided Kivy App in main.py.
- Your main program will contain the 'model' in a single SongCollection instance that you will use in your program.
- The program should start by loading a JSON file of songs.
- The songs file must be saved when the program ends, updating any changes made with the app by the user (use the on_stop method from KivyDemos).

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

- The left side of the screen contains a drop-down "spinner" for the user to choose the song sorting (see spinner_demo from KivyDemos), and TextInput fields for entering information for a new song.
- The right side contains buttons for the songs, colour-coded based on whether they are learned or not.
- The status label at the top of the right side shows the number of songs learned and still to learn.

- The status label at the bottom of the right side shows messages about the state of the program, including when a song or other button is clicked on.
- When the user clicks on a song button, the state of the song changes between learned and unlearned (see guitars_app.py from KivyDemos for an example of using Kivy with custom objects associated with buttons).
- The user can add a new song by typing in the input fields and clicking “Add Song”.
- The “Clear” button clears the input fields and the status label at the bottom.

Adding Songs (Error Checking):

- All song fields are required. If any field is left blank, the bottom status label should display “**All fields must be completed**” when “Add Song” is clicked. Note that this message appears first even if an invalid year exists in the year field.
- The year field must be a valid integer. If this is invalid, the status label should display “**Please enter a valid number**”.
  If a valid but non-positive integer is entered, the status label should display “**Year must be > 0**”.
- Pressing the Tab key should move between the text fields (use multiline: False and write_tab: False as in the provided starter code)
- When the user successfully adds a song, the text fields should be cleared and the new song's button should appear in the songs list on the right (see dynamic_widgets from KivyDemos).

Coding Requirements:

- At the very top of your main.py file, complete the comment containing your details.
- Document your classes and methods clearly with docstrings. Include inline/block comments as appropriate. You do not need comments in the kv file.
- Use functions/methods appropriately for each significant part of the program. Remember that functions should follow the Single Responsibility Principle.
- 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.
- Complete your GUI design using the kv language in the app.kv file. Most of this is done for you in the starter code.
  Creating the song buttons should be done in main.py, not in the kv file, since this will be dynamic. (dynamic_widgets from KivyDemos)

Submission:

Submit your assignment as a zip file of your project containing all the files.

Integrity:

The work you submit for this assignment must be your own. Submissions that are detected to be too similar to that of another student or other work will be dealt with according to the College procedures for handling plagiarism and may result in serious penalties.

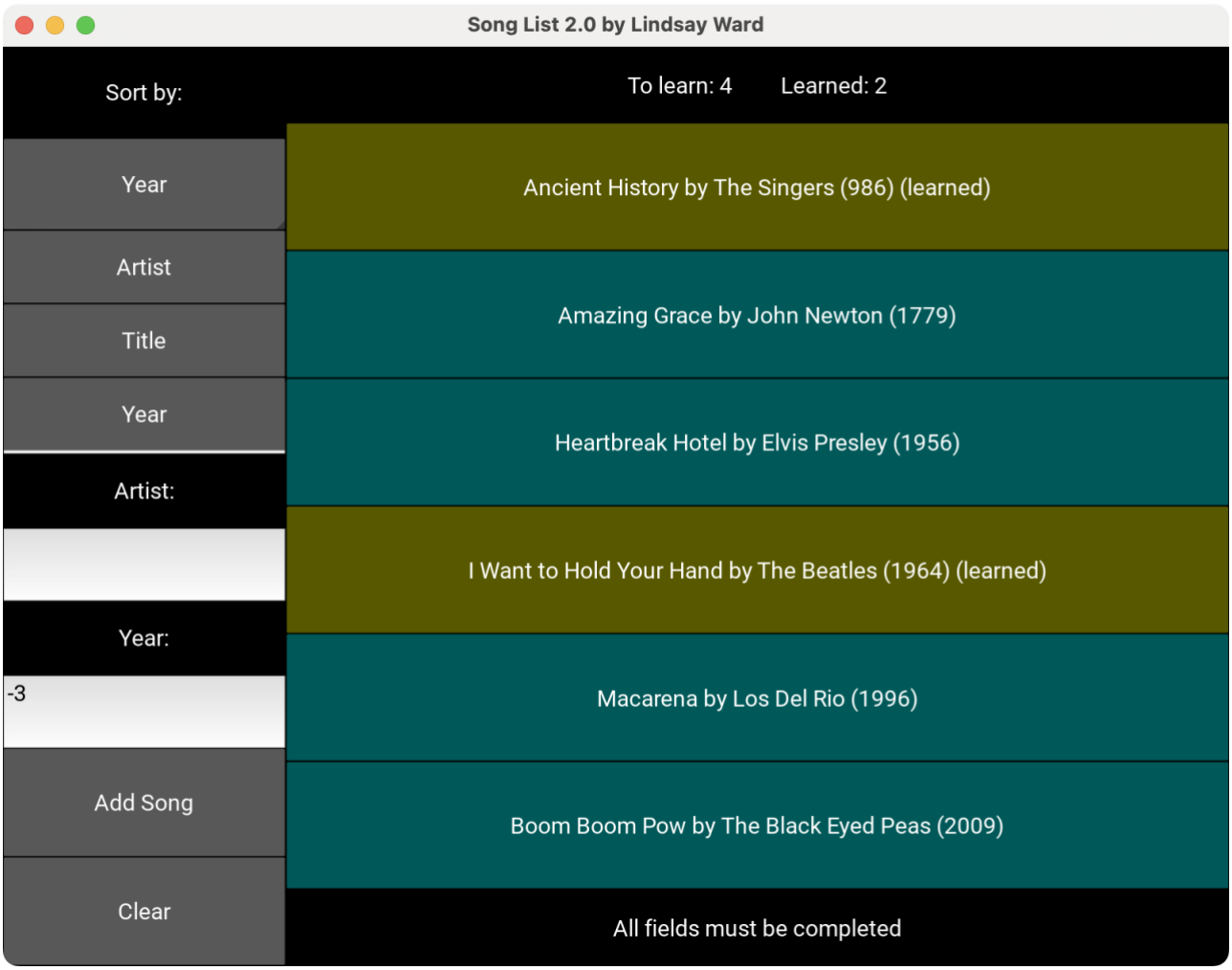
The goals of this assignment include helping you gain understanding of fundamental programming concepts and skills, and future subjects will build on this learning. Therefore, it is important that you develop these skills to a high level by completing the work and gaining the understanding yourself. 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 questions by talking with your lecturer or tutor.

The subject teaching contains all the information you need for this assignment. Do not use online resources (e.g., search, Stack Overflow, ChatGPT) to find resources or 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.

Sample Output:

Screenshots have been provided here. In addition, you should study the **screencast** recording provided with this assignment to see how the GUI program should work, including what the messages should be and when they occur.



Marking Scheme:

Ensure that you follow the processes and guidelines taught in class 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.

Criteria	Exemplary (9, 10)	Good (7, 8)	Satisfactory (5, 6)	Limited (2, 3, 4)	Very Limited (0, 1)
Tests **5%**	All methods in classes are tested appropriately.	Exhibits aspects of exemplary (left) and satisfactory (right)	Some methods in classes are tested appropriately.	Exhibits aspects of satisfactory (left) and very limited (right)	No testing is completed.
Console program **9%**	Class(es) used correctly in console program.		Class(es) used in console program but not correctly.		Class(es) not used in console program.
Error handling **7%**	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 ***20%***	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 **12%**	All function, variable and constant names are appropriate, meaningful and consistent.		Several function, variable or constant names are not appropriate, meaningful or consistent.		Many function, variable or constant names are not appropriate, meaningful or consistent.
Use of code constructs **15%**	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 **20%**	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 **7%**	Code contains helpful # block comments, all classes and methods have meaningful docstrings and main module docstring contains all details.		Comments are reasonable, but some classes and methods have no docstrings, and/or there is some noise (too many comments), and/or missing details in main module docstrings.		Commenting is very poor or not done.
Formatting **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.