Assignment 2, Part B: Saving the Data

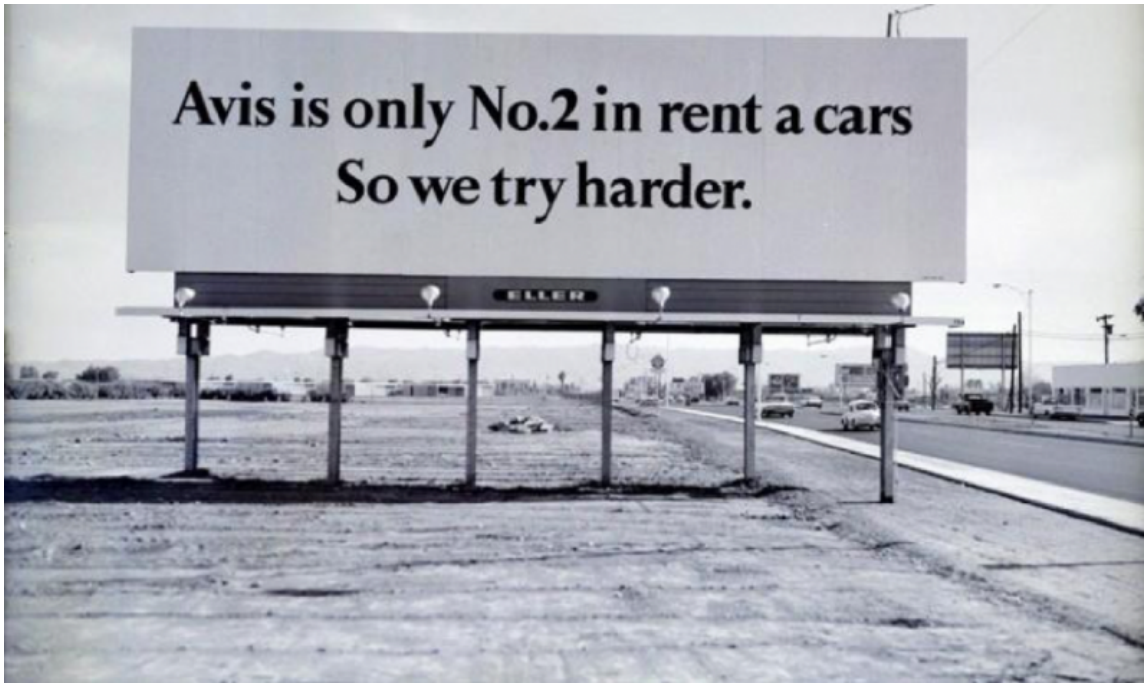
(5%, due 11:59pm Sunday, October 18th, 2020, end of Week 12)

Overview

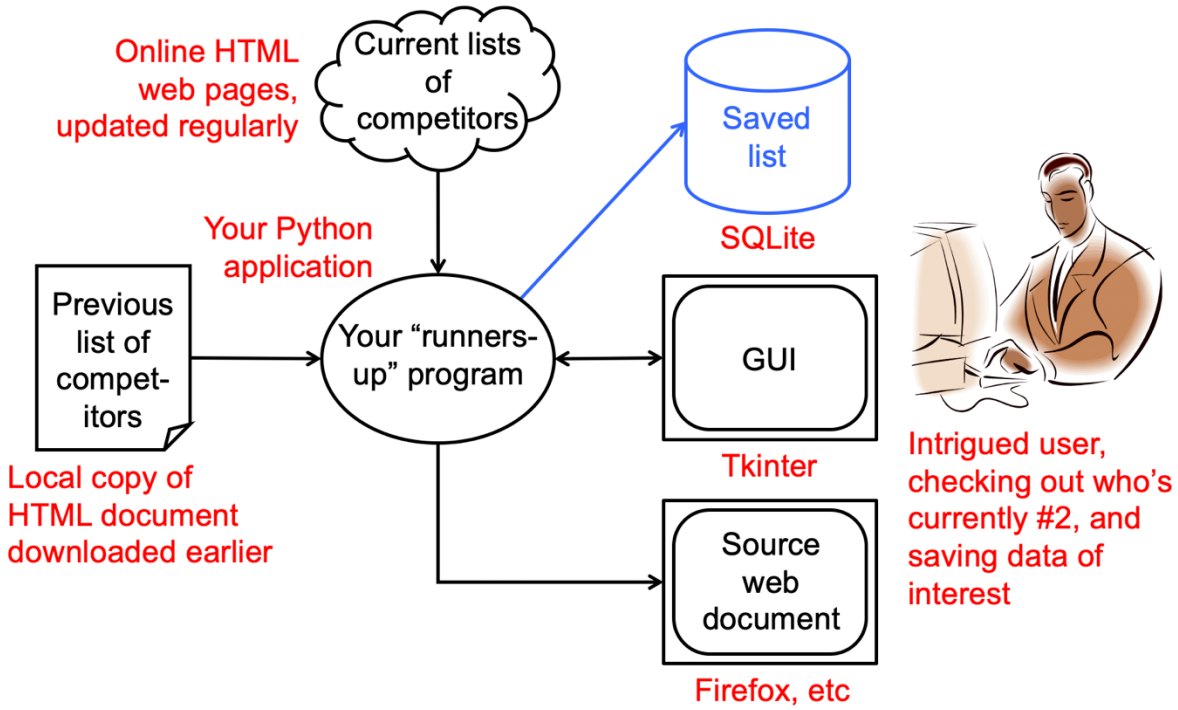
This is the second part of a two-part assignment. This part is worth 5% of your final grade for IFB104. Part A which preceded it was worth 20%. This part is intended as a last-minute extension to the assignment, thereby testing the maintainability of your solution to Part A and your ability to work under time pressure. If you have a neat, clear solution to Part A you will find completing Part B straightforward. For the whole assignment you will submit only one solution, containing your combined response to both Parts A and B, and you will receive one grade for the whole 25% assignment.

Goal

In Part A of this assignment you built a significant "IT system" which provided its users with a convenient way of examining data obtained from online web sites. It was assumed that the user has a particular interest in competitive environments in which the runner-up's position is strategically important. In many situations the competitor coming second is in a position to take the lead at any time, yet avoids the costs and risks associated with being out front. Some organisations even take pride in being second!



Although your solution to Part A allows the user to examine competitive lists "live," the regularly changing contents of these lists means that there may be a need to save a particular list for later study. In this part of the assignment you will extend your application so that the user can choose to save a description of the current runner-up and other competitors in a database. The extension to your software architecture is illustrated in blue below.



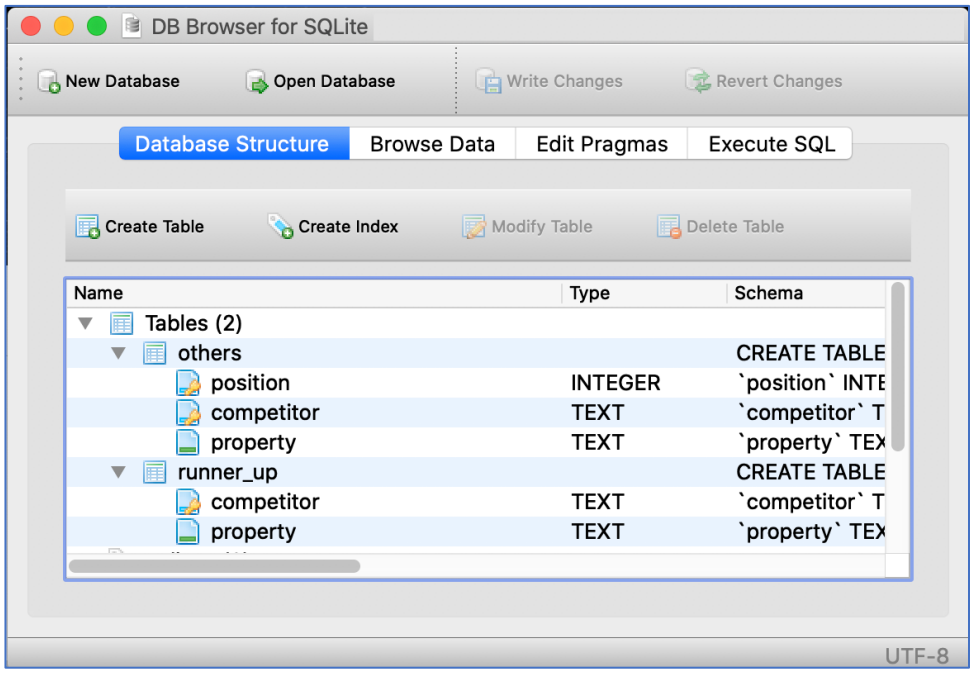
The database can be used to save a copy of all the information currently displayed in the GUI, but persists even after the Python program has terminated and can be accessed by other applications.

To complete this part of the assignment you must extend the Python code you have already written so that the user can choose to update the database with the information displayed in the GUI whenever they like. Your program must save the data in a supplied SQLite database, `runners_up.db` which contains two tables. The first table, called `runner_up`, has two fields, `competitor` and `property`, and is used to save details of the runner-up currently displayed in the GUI. The second table, called `others`, has three fields, `position`, `competitor` and `property`, and is used to save details of the other nine competitors at the top of the currently-selected list. It must be possible to save whichever data set is currently displayed in the GUI, including all three "live" lists and the static one. (Although the user already has a copy of the static list, it is in the form of an HTML source code document, so we assume the user also wants the ability to save it in a more convenient database format.)

You should assume the database already exists when your program is started. Only the current GUI contents should appear in the database, so your program must delete all data in both tables, if any, before inserting new rows. (In effect, the user can keep details of only one list at any time.) The competitors' identities and their other "properties" of interest should appear in the database exactly as they appear in the original web documents. However, no HTML mark-ups or special character codes should appear in the database text.

Illustrative example

Supplied with these instructions is an empty SQLite database called `runners_up.db`. If you open it in the *DB Browser for SQLite* or an equivalent tool you will see that it contains two tables, `runner_up` and `others`, as shown below.

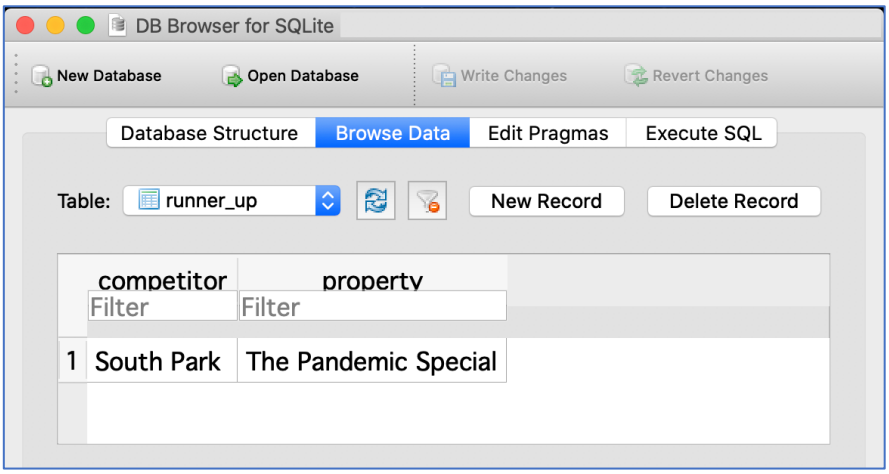


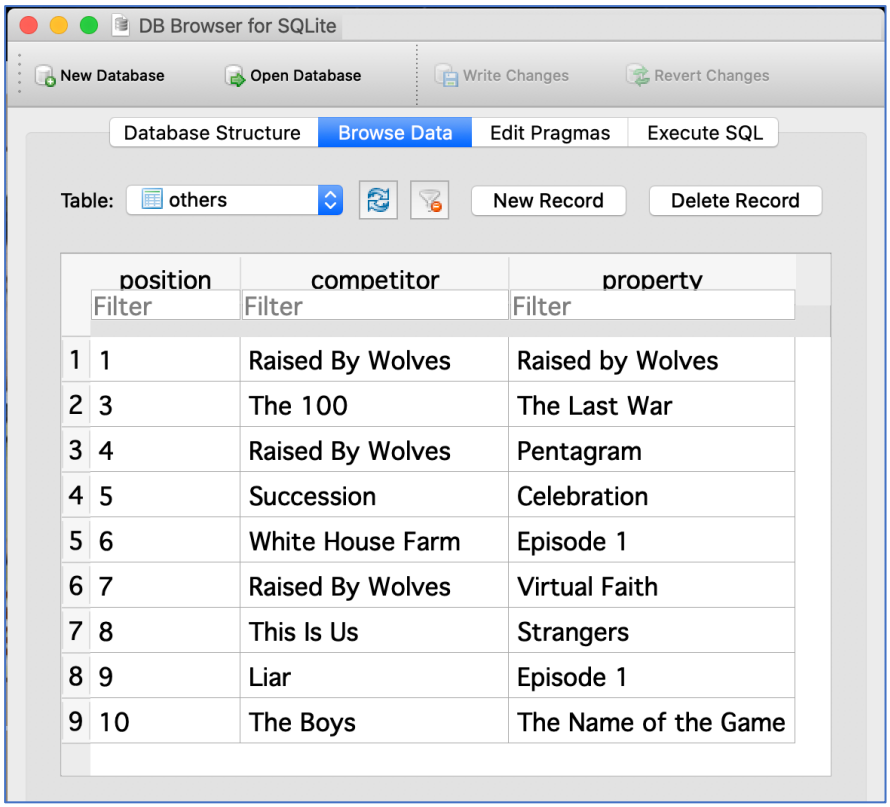
In the instructions for Part A we demonstrated a solution which we called *What's on Second?* Here we extend its capabilities so that the user can save the data currently displayed in the GUI whenever they like.

Assume that the user begins by inspecting the current runner-up TV show on iTunes Australia.

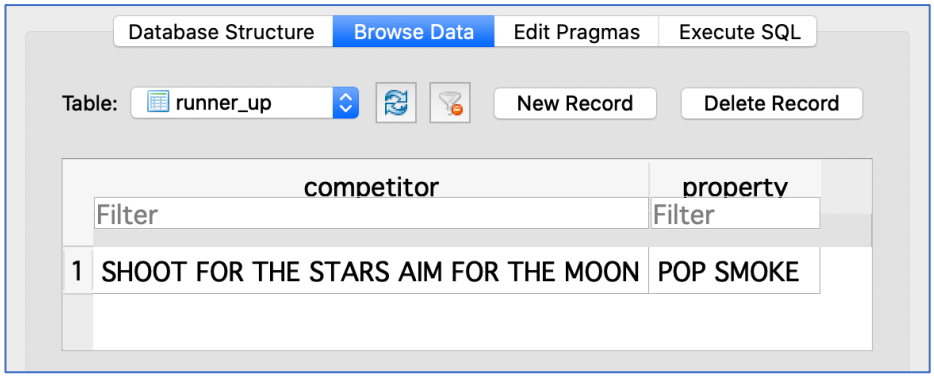
The image above shows the outcome when we ran our program on October 4th. Notice that the data displayed is entirely different from the results shown in the Part A instructions, which were created on August 11th, due to the source web site having been updated in the meantime. Previously the runner-up TV show was *The Real Housewives of Beverly Hills*, but now it is *South Park*, specifically the *Pandemic Special* episode. (The rest of the list is dominated by episodes of the new series *Raised by Wolves*.)

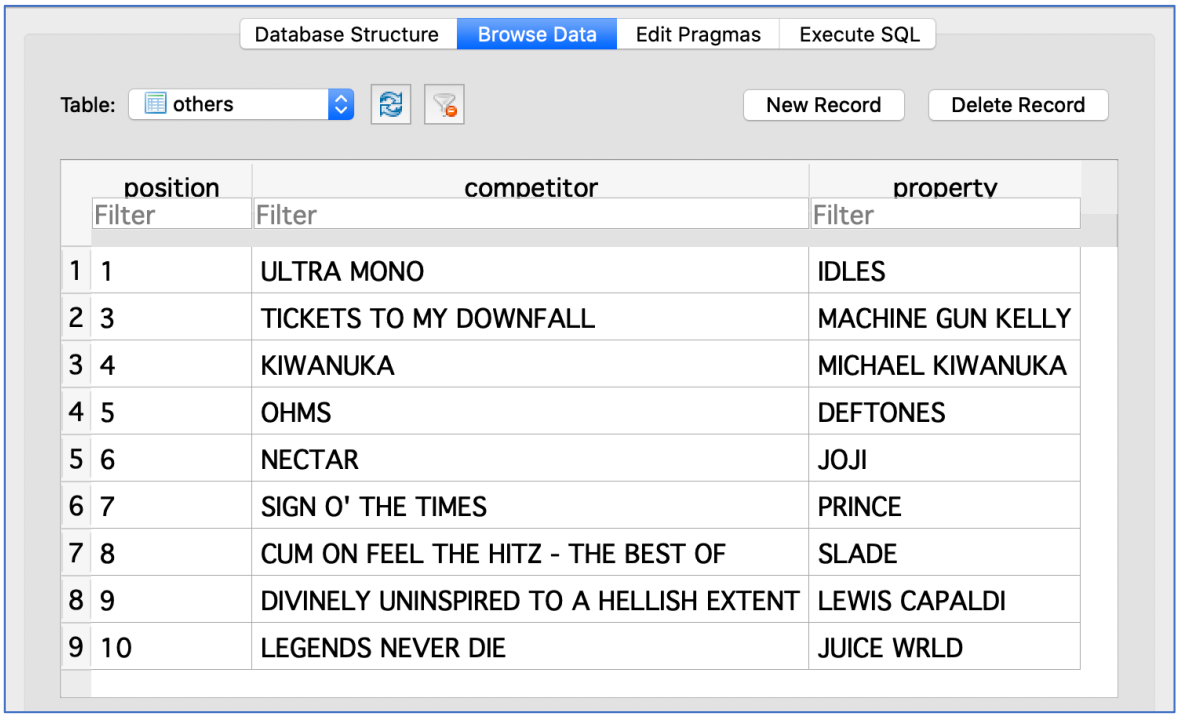
Apart from this, our Part B solution adds a new widget to the GUI, the "Save" button highlighted above. Intrigued by the change to the TV show data, our user now presses this button, which causes the data currently displayed in the GUI to be written into the database's two tables, as shown below. The row numbers on the far left, added by the *DB Browser for SQLite*'s interface, are not related to the competitors' positions, which is why the `position` field is needed in the second table.





The same thing can be done with the other two "live" data sources, as shown below.





	position	competitor	property
	Filter	Filter	Filter
1	1	ULTRA MONO	IDLES
2	3	TICKETS TO MY DOWNFALL	MACHINE GUN KELLY
3	4	KIWANUKA	MICHAEL KIWANUKA
4	5	OHMS	DEFTONES
5	6	NECTAR	JOJI
6	7	SIGN O' THE TIMES	PRINCE
7	8	CUM ON FEEL THE HITZ - THE BEST OF	SLADE
8	9	DIVINELY UNINSPIRED TO A HELLISH EXTENT	LEWIS CAPALDI
9	10	LEGENDS NEVER DIE	JUICE WRLD

Database Structure | Browse Data | Edit Pragmas | Execute SQL

Table: runner_up | New Record | Delete Record

	competitor	property
	Filter	Filter
1	Mulan	5.7

Database Structure | Browse Data | Edit Pragmas | Execute SQL

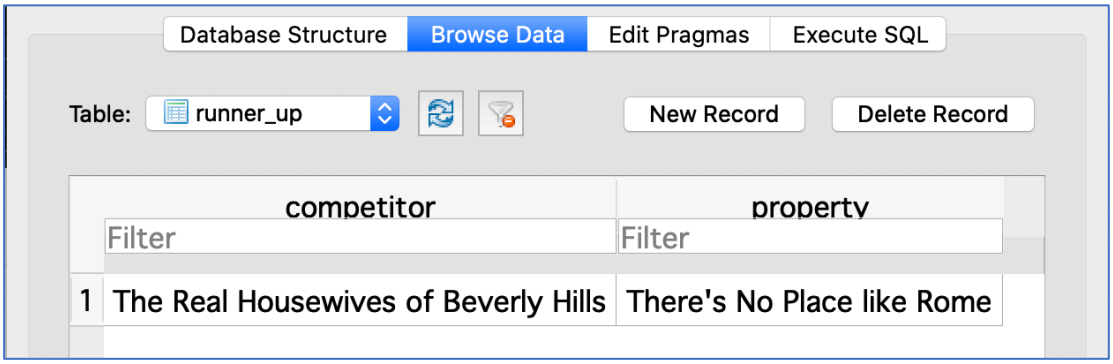
Table: others | New Record | Delete Record

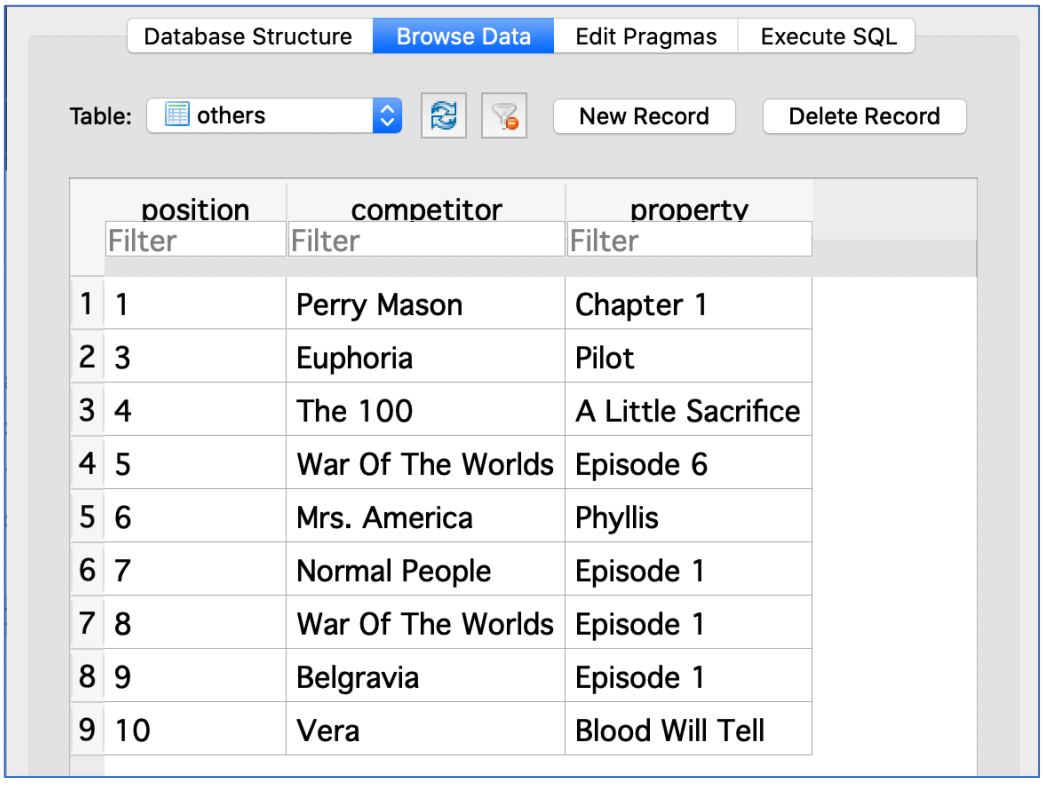
	position	competitor	property
	Filter	Filter	Filter
1	1	Enola Holmes	6.7
2	3	Antebellum	5.5
3	4	Ava	5.4
4	5	The Devil All The Time	7.2
5	6	Greyhound	7.1
6	7	Bill & Ted Face the Music	6.5
7	8	Tenet	7.9
8	9	Project Power	6.1
9	10	Peninsula	5.6

Again this data was obtained by running our solution on October 4th. Compare the screenshots above with those in the Part A instructions to see how the rankings have changed.

Most importantly, notice that each time the user saves the currently-displayed data, the previous contents of the database tables are overwritten. Only one set of data may be saved at any time.

Finally, even though it never changes, we provide the same capability for the "static" list. Its data is, of course, still the same as in the Part A instructions, and appears as follows when written into the database.





	position	competitor	property
	Filter	Filter	Filter
1	1	Perry Mason	Chapter 1
2	3	Euphoria	Pilot
3	4	The 100	A Little Sacrifice
4	5	War Of The Worlds	Episode 6
5	6	Mrs. America	Phyllis
6	7	Normal People	Episode 1
7	8	War Of The Worlds	Episode 1
8	9	Belgravia	Episode 1
9	10	Vera	Blood Will Tell

NB: Recall that the order of columns and rows in a relational database table is irrelevant. In the screenshots above the rows appear in the order they were inserted into the database, but in some *DB Browser for SQLite* implementations the default order in which rows are displayed may be different and, of course, it's always possible to sort the rows differently, e.g., alphabetically by competitors' names. This is another reason why the `position` field is necessary in the second table.

Your task is to extend your solution to Part A of the assignment with a database storage capability equivalent to that above. However, you do not need to duplicate our approach to the GUI. A number of solutions are possible. We used a push button widget which the user presses in order to save the data, but you could equally well implement a "save mode" check box or radio buttons which can be toggled so that the current GUI data is stored automatically whenever it's updated. The main thing is that the user must have clear and full control over what information is stored in the database.

You also need to consider the robustness of your GUI. For instance, with our GUI design above we needed to allow for the possibility that the user presses the "Save" button before selecting one of the radio buttons. One option would be to display an error message in the GUI and leave the database unchanged. However, in this case we chose to implement a default action which was to save the "static" list's data in the database if pressing the "Save" button is the user's first action because that list's radio button is highlighted by default when the program begins.

Supporting material

Accompanying these instructions we have provided a copy of the necessary SQLite database, `runners_up.db`, which contains two tables. The first table, called `runner_up`, has two fields, `competitor` and `property`, both of type `Text`. The second, called `others`, has three fields, `position`, `competitor` and `property`, the first field of type `Integer` and the second two of type `Text`.

As supplied the database table is empty. Your solution should assume that this database and its tables *already exist in the same folder as your Python program*. Your program does not need to create the database or the tables. Before you begin you should confirm that you can open the database with the *DB Browser for SQLite* or a similar database tool. (Do not drag-and-drop the database file into the *DB Browser*'s GUI. On some systems this causes the *DB Browser* to ask for a password, even though the database is not password-protected.)

Development hints

- It should be possible to complete this task merely by *adding* code to your existing solution, with little or no change to the code you have already completed.
- The SQLite statements needed to complete this task are quite simple. Similar examples can be found in the relevant lecture demonstrations and workshop exercises.
- The data written into the database is exactly the same as the data displayed in the GUI, so you should be able to re-use the data extracted from the source web documents in Part A of the assignment.
- If you are unable to complete the whole task, just submit whatever parts you can get working. You will receive *partial marks* for incomplete solutions. Even if incomplete, ensure that your program runs as submitted.

Requirements and marking guide

To complete this task you are required to further extend the provided `runners_up.py` template file with your solution to Part B, on top of your solution to Part A, to provide a data storage capability equivalent to that illustrated above.

The extension for Part B must satisfy the following criteria. Marks available are as shown.

- **Widget(s) for choosing to store the displayed data (1%).** Your GUI must provide some simple, intuitive feature to allow the user to control when the currently-selected runner-up and other competitors are saved in the database. The user must be able to request that the database's contents are updated whenever and as often as they like.
- **All competitor details saved in the database (4%).** Details of the currently-selected runner-up and other competitors can be saved any time the user wants, including all the information required to populate all the fields in both tables. Each time the user chooses to do so the previous database contents, if any, are overwritten. Your Python code should assume that the necessary SQLite database already exists in the same folder as your Python program.

You must complete this part of the assignment using only basic Python features and the `sqlite3` module.

Portability

An important aspect of software development is to ensure that your solution will work correctly on all computing platforms (or at least as many as possible). For this reason you must **complete the assignment using standard Python 3 modules and functions only**. You may *not* import any additional modules or files into your program other than those already imported by the given template file. **In particular, you may not use any Python modules that need to be downloaded and installed separately, such as "Beautiful Soup" or "Pillow." Only modules that are part of a standard Python 3 installation may be used.**

Security warning and plagiarism notice

This is an individual assessment item. All files submitted will be subjected to software plagiarism analysis using the MoSS system (http://theory.stanford.edu/~aiken/moss/). Serious violations of the university's policies regarding plagiarism will be forwarded to the Science and Engineering Faculty's Academic Misconduct Committee for formal prosecution.

As per QUT rules, you are not permitted to copy *or share* solutions to individual assessment items. In serious plagiarism cases SEF's Academic Misconduct Committee prosecutes both the copier and the original author equally. It is your responsibility to keep your solution secure. In particular, **you must not make your solution visible online via cloud-based code development platforms such as GitHub**. Note that free accounts for such platforms are usually public and are thus unsafe. If you wish to use such a resource, do so only if you are certain you have a *private* repository that cannot be seen by anyone else. For instance, university students can apply for a *free* private repository in GitHub to keep their assignments secure (https://education.github.com/pack). However, we recommend that the best way to avoid being prosecuted for plagiarism is to keep your work well away from both the Internet *and your fellow students*!

Deliverables

You must develop your solution by completing and submitting the provided Python template file `runners_up.py`. **Your solution may not rely on any Python modules that need to be downloaded and installed separately, such as "Beautiful Soup." Programs that do not**

run in a standard Python 3 environment will receive 0%. Submit your Python code in a "zip" archive containing all the files needed to support it as follows:

1. Your `runners_up.py` solution. Make sure you have completed the "statement" at the beginning of the Python file to confirm that this is your own individual work by inserting your name and student number in the places indicated. **Submissions without a completed statement will be assumed not to be your own work.**
2. One or more *small* image files needed to support your GUI interface, but **no other image files**.
3. Copies of the previously-downloaded web document used for your static "previous runner-up" list. Include only a single HTML document. **Do not include any image or style files associated with the web document**. Only the HTML source code is needed by your Python program.
4. An empty copy of the `runners_up.db` database, ready to be populated by your Part B solution.

Once you have completed your solution and have zipped up these items submit them to Blackboard as a single file. **Submit your solution compressed as a "zip" archive. Do not use other compression formats such as "rar" or "7z".**

Apart from working correctly your Python code must be well-presented and easy to understand, thanks to (sparse) commenting that explains the *purpose* of significant parts and *helpful* choices of variable and function names. **Professional presentation** of your code will be taken into account when marking this assignment.

If you are unable to solve the whole problem, submit whatever parts you can get working. You will receive **partial marks for incomplete solutions**.

How to submit your solution

A link is available on Blackboard under Assessment for uploading your solution before the deadline (11:59pm Sunday, October 18th, end of Week 12). Note that you can submit as many drafts of your solution as you like. You are strongly encouraged to *submit draft solutions* before the deadline as insurance against computer and network failures. If you are unsure whether or not you have successfully uploaded your file, upload it again!

Students who encounter problems uploading their assignment files to Blackboard should contact *HiQ's Technology Services* (http://qut.to/ithelp; askqut@qut.edu.au; 3138 2000) for assistance and advice. **Do not email assignments to the teaching staff.** They cannot be accepted for marking because we cannot upload assignments to Blackboard on your behalf. Also, teaching staff will not answer email queries on the weekend the assignment is due, so ensure that you have successfully uploaded at least one solution by close-of-business on Friday, October 16th.