Assignment 2, Part B: Storing Events

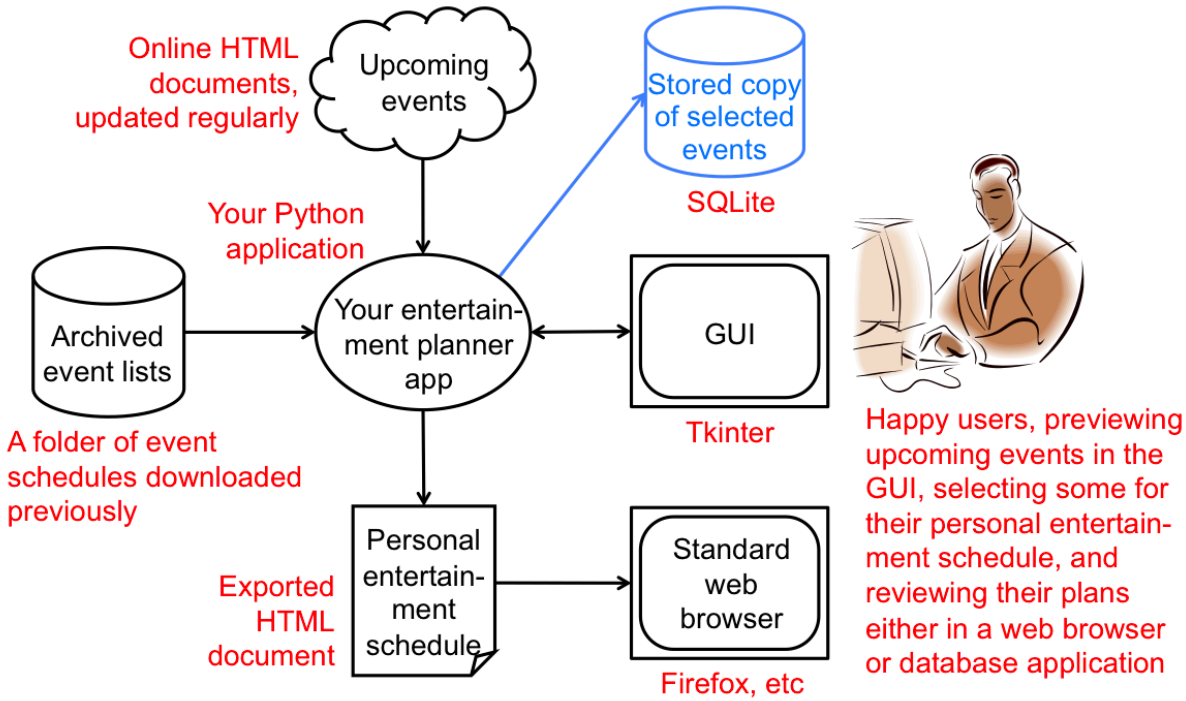
(4%, due 11:59pm Sunday, May 26th, end of Week 12)

Overview

This is the second part of a two-part assignment. This part is worth 4% of your final grade for IFB104. Part A which preceded it was worth 21%. 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 easy. 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 application which allows its users to browse upcoming entertainment events (or review old ones in "offline mode"), select certain events of interest, and export the resulting entertainment schedule as a web document. Here you will extend your application so that it can also save the entertainment schedule in a database. The extension is shown in blue below.



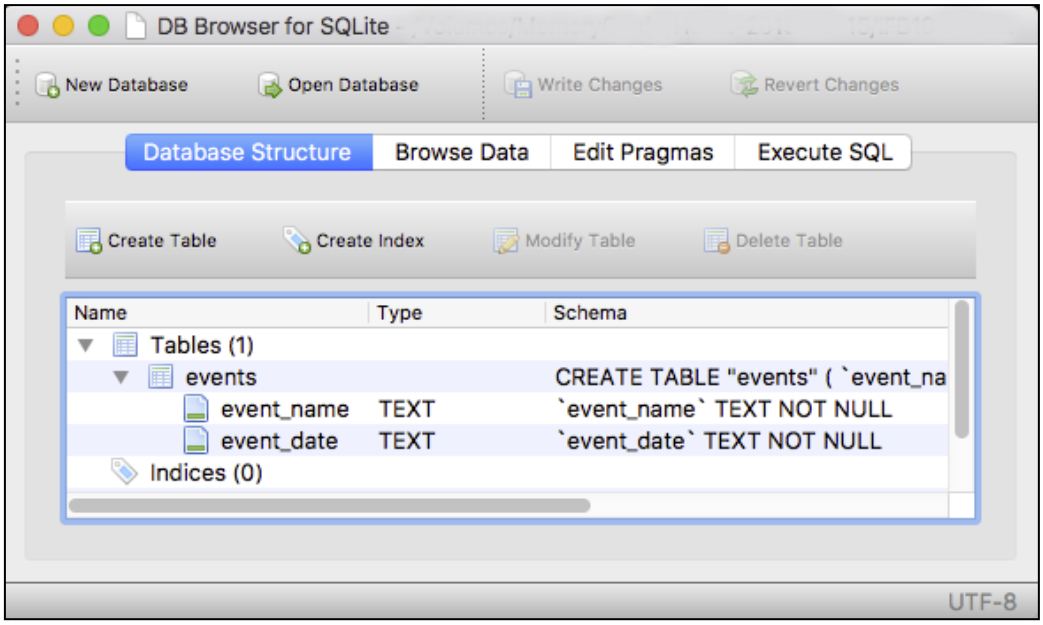
To complete this part of the assignment you must extend the Python code you have written for Part A so that it gives the user the ability to store their currently-selected entertainment events in the database whenever they want. This involves both extending the GUI and linking your application to a database, `entertainment_planner.db`.

The database file accompanies these instructions. It contains a single table, `events`, which has two fields, `event_name` and `event_date`. For each event currently chosen by the

user your program must insert these two values as a new row in the table. Your program should assume the database already exists and the table is empty when it is started. Whenever the user chooses to store the current event data, any previous data in the table is overwritten; only the currently-selected events are stored, not previous selections.

Illustrative example

Supplied with these instructions is an empty SQLite database called `entertainment_planner.db`. If you open it in the *DB Browser for SQLite* or an equivalent tool you will see that it contains a single table, `events`, which has two text fields, `event_name` and `event_date`, as shown below.

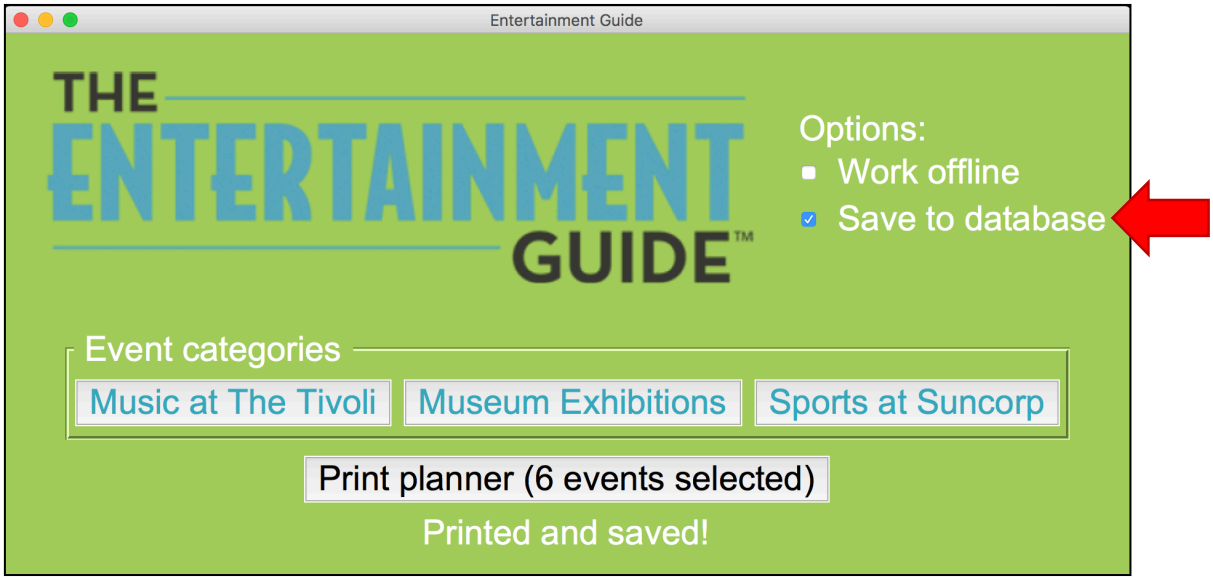


In the instructions for Part A we used a demonstration program which we called "*The Entertainment Guide*". It allowed its user to preview and select upcoming entertainment events in a graphical user interface and export the resulting schedule as an HTML document. Now we extend its capabilities so that it additionally allows the user to store the schedule in this database. To illustrate this, consider the following example.

As explained in the Part A instructions, the most important aspect of your Python program is that it must allow for changes to the source web documents in "online mode". This is why you need to use a general "pattern matching" approach to extract elements from the online web documents. In this example the user decides to check which events are coming up at *Suncorp Stadium* and the *Museum of Brisbane*. Compare the screenshots below, taken on May 15th 2019, with those in the Part A instructions and you will see how the events have changed since those instructions were written.

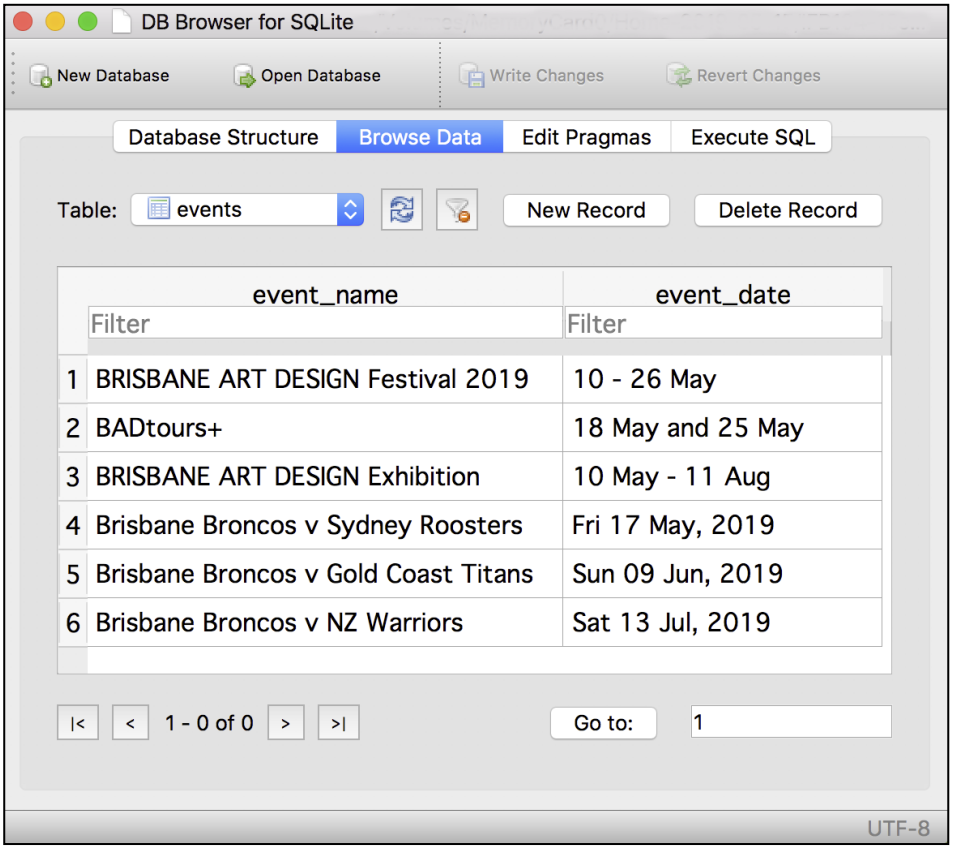
In this case the user is keen to visit all the events in the "Brisbane Art Design" festival and has selected three such events at the Museum of Brisbane. The user is also a Brisbane Broncos fan and has selected three upcoming Broncos games to attend.

At this point our Part A solution allows the user to export their selections as an entertainment schedule in HTML format. However, our Part B extension additionally gives the user the option of storing the selected events in the database, as shown overleaf.



Here we have added a checkbox which allows the user to control whether or not the events are saved in the database when the HTML document is printed. Other GUI solutions are possible, of course, such as a push button which saves the data whenever it is pressed, or linked radio buttons to control whether or not to save.

In the screenshot above the user has just printed their entertainment "planner" and simultaneously saved the chosen events in the database. Opening the database in the *DB Browser for SQLite* reveals the data stored.



Recall that the order of columns and rows in a relational database table is irrelevant. In the screenshot above the items appear in the order they were inserted by our Python program, but in some *DB Browser for SQLite* implementations the order in which rows are displayed may be different.

In this example the user's selections were also exported as an HTML document, containing more detail about the events, as shown below.

Your *Entertainment Guide*

BRISBANE ART DESIGN Festival 2019

When: 10 - 26 May

BADtours+

When: 18 May and 25 May

BRISBANE ART DESIGN Exhibition

When: 10 May - 11 Aug

Brisbane Broncos v Sydney Roosters

When: Fri 17 May, 2019

Brisbane Broncos v Gold Coast Titans

When: Fri 17 May, 2019

Brisbane Broncos v Gold Coast Titans

When: Sun 09 Jun, 2019

Brisbane Broncos v NZ Warriors

When: Sat 13 Jul, 2019

Events sourced from:

- https://thetivoli.com.au/events
- https://www.museumofbrisbane.com.au/whats-on/
- https://suncorpstadium.com.au/what-s-on.aspx

In our sample Part B solution, every time the user chooses to "print" their chosen events, and the "save to database" option is selected, the events are also stored in the database. Any existing data in the database is overwritten when this happens. Only the current selections may appear in the database, not old ones.

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 follow the GUI design above. Creativity is encouraged and any solution that is easy to use and provides the user with equivalent functionality is acceptable.

What about the primary key?

For the purposes of this assignment the definition of the database's `events` table is very simple, merely two fields of type `Text`. No primary key for the table is specified. This is because we cannot know for certain that all the event name/date pairs will be unique, as illustrated by the *Museum of Brisbane* exhibitions in our sample solution above. In this case the source web site lists two different versions of the museum's "Clock Tower Tours" with the same name and date (but with different descriptions and images). Normally in an SQL database duplicate rows such as this are not permitted in a table. Fortunately, however, SQLite allows such duplicates when no primary key is defined, which simplifies your task because you don't need to check for duplicate rows when inserting rows into the database.

Requirements and marking guide

To complete this task you are required to further extend the provided `whats_on.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 events (2%).** Your GUI must provide some simple, intuitive feature to allow the user to control whether or not events are stored in the database. A number of solutions are possible, e.g., a pushbutton widget which the user presses in order to save the currently selected events, or a "save mode" checkbox or radiobuttons which can be toggled so that the currently-selected events are stored automatically whenever an event planner is exported. Either way, the user must have full and clear control over whether or not the events are saved in the database.
- **Storing events in the database (2%).** Whenever the user chooses to do so, all of the currently-selected events must be written into the `entertainment_planner.db` database's `events` table. Each record must consist of the event's name and date. Existing data in the table, if any, should be overwritten, so that only the events currently-selected are stored. 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. 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 in Part A.

The SQLite statements needed to complete this task are quite simple. Similar examples can be found in the relevant lecture demonstrations and workshop exercise solutions.

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.

Supporting material

Accompanying these instructions we have provided a copy of the necessary SQLite database, `entertainment_planner.db`. It contains a single table, `events`, which has only two fields, `event_name` and `event_date`. As supplied the database table is empty. Your solution should assume that this database and its table *already exists in the same folder as your Python program*. Your Python program does not need to create the database or the table. Before you begin you should confirm that you can open this database with the *DB Browser for SQLite* or a similar database tool.

As well as the SQLite database, `entertainment_planner.db`, we have provided a very simple "dump" script, `entertainment_planner.sql`, which re-creates the database's table when executed. You should not need to use this script at all; it should be possible to open the `entertainment_planner.db` file from the *DB Browser for SQLite*'s "Open Database" menu option. (Do not drag-and-drop the database file into the *DB Browser*'s GUI. On some operating systems this causes the *DB Browser* to ask for a password, even though the database is not password-protected.) In the highly unlikely event that you need to create a fresh copy of the database using the dump script you can do so easily using the *DB Browser*'s "Import database from SQL file" menu option. Ensure that you name the resulting database "`entertainment_planner.db`" and remember to write the changes to it after importing the script.

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 functions and modules 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. 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 the Internet!

Deliverables

You should develop your solution by completing and submitting the provided Python template file `whats_on.py`. Submit this in a "zip" archive containing all the files needed to support your application as follows:

1. Your `whats_on.py` Python program. 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. A folder containing the previously-downloaded web documents used for your static "archive" of old events. Again, this folder may contain HTML/XML source code files only. **It must not contain any image files**. All images needed for your exported HTML document must be sourced from online when it is viewed in a web browser.
4. A copy of the `entertainment_planner.db` database as provided with these instructions. Ensure that the database's `events` table is **empty when you submit your solution** so that the marker can clearly see your Part B solution adding data to it.

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 and HTML code must both be well-presented and easy to understand, thanks to (sparse) commenting that explains the *purpose* of significant elements 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, May 26th, 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 files to Blackboard should contact the IT Helpdesk (ithelpdesk@qut.edu.au; 3138 4000) for assistance and advice. 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, May 24th.