

# IFB104 — Building IT Systems

## Topic 8 — Online Documents & How to Remember Things

School of Computer Science  
Semester 1, 2025

# Housekeeping

- Assignment 1B marks are live
  - You can check final marks for this on Canvas
  - You can access criteria breakdowns on Gradescope
  - Note your Gradescope marks may be inaccurate



# Housekeeping

- Keep working on Assignment 2A
  - The deadline has been extended to the **end of this week** (Friday, May 2<sup>nd</sup>)!
  - But this task is worth only 7% of your final grade
  - You may wish to upload 2A sooner to begin work on 2B
  - For safety, upload **multiple drafts** before close-of-business on Friday
  - Remember to press the 'Submit' button in Canvas after uploading the file
  - **DON'T** leave it until the last minute!
  - **DON'T** attempt to email code to teaching staff – most email clients block Python attachments, so we won't receive them!



# A plagiarism reminder

- Your submissions throughout the unit will be submitted to a software plagiarism analyser
- Submit your own work, not someone else's, even if it's incomplete
  - Don't put your code where someone else can see it, especially publicly-accessible online sites such as *Github*
  - Don't show your code to anyone else *even after the deadline* – they may have an extension!
  - And *never* believe *anyone* when they say they “just want to have a look” at your code – they *will* copy it and you will get the blame!

## Finalisation of Misconduct - IFB104

Faculty of Science Appeals and Misconduct Committee

### Penalty

a 6 month  
exclusion and a  
fail recorded for  
the unit IFB104  
Building IT  
Systems

This is the  
outcome of  
a case in a  
previous  
semester!

# Need help?

- STIMulate drop-in consultation on-campus, weekdays, 10am–3pm:

<https://stimulate.qut.edu.au/>

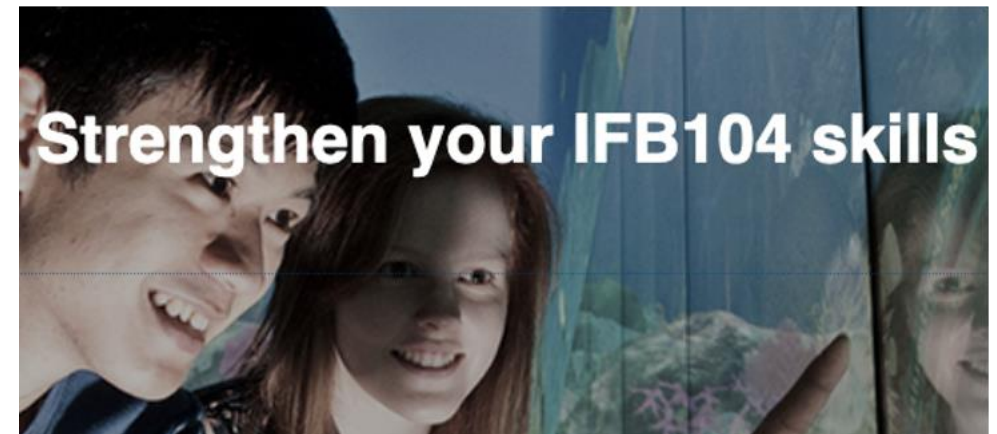
- STIMulate online by request, for students who can't get to campus:

<https://qut.to/q826v>



# STIMulate

Facilitating real success in maths, science and IT across QUT



# Housekeeping

- Workshops this week will focus on today's lecture material
- Full details for Assignment 2B will be released tomorrow (Tuesday, 29<sup>th</sup> April)
- Today, we get a sneak peek!





# Brisbane gamers, we need you!

Are you between 16-25? Passionate about mental health and gaming? Active on Discord?

Join our design and development research for improving gamers mental health and get compensated for your time.

SCAN  
ME



HEALTHY  
GAMERS



# Aims of this Week's Lecture

- Basics of HTML
- Introducing databases
- Introducing SQLite
- A sneak peek of Assignment 2B



# Assignment 2B

## SCENARIO

Part A introduced you to **Tkinter** by asking you to construct a simple window that displayed details drawn from a hard-coded list of dictionaries. Part B moves the same interface into a realistic production setting. MoviVision Media has supplied a fully-populated **SQLite** database file called `movies.db`, which sits alongside your Python code in the assignment download. This file contains all movie records collected by the company's web-scraping system, including titles, lead actors, directors, and the original HTML-rich synopses.

Your task is to connect your existing interface to this file, issue searches against its contents, and present your users with a cleaned synopsis that is free from HTML tags and special character codes.

# Assignment 2B

## TASK BREAKDOWN

- Adapt your code so that all references to the dummy *movies\_data* list are removed
- Every press of the **Search** button must perform a **parameterised SQL** query against movies.db
- A search should match a keyword that appears in the **title**, **lead actor**, **director**, or **synopsis** fields, returning all eligible records
- Movie details should be **stripped of HTML markup** and **entity codes**
  - You can now add the re module to perform this cleaning
  - Use of regex will be evaluated

HTML entity	Character shown after cleaning
&amp;	& (ampersand)

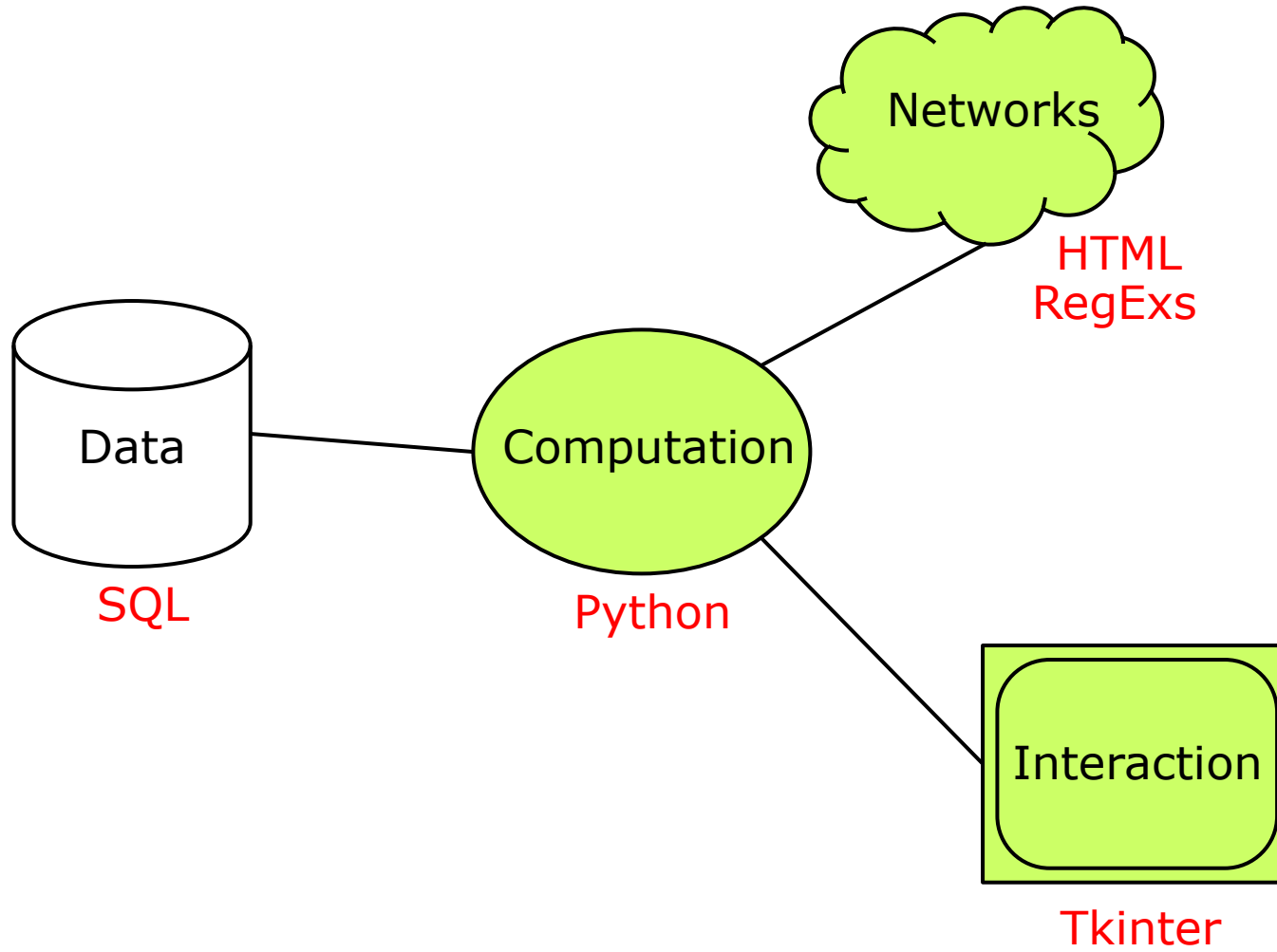
# Assignment 2B

## AMBIGUITIES

- Client briefs rarely include solutions for every ambiguity
- Situations such as searching with a blank keyword, entering purely whitespace characters, or requesting a description from an empty result set are left open
- It is up to you as a developer to include **reasonable accommodations** for these ambiguities in your program
  - Remember that uses value clarity and efficiency
- If you feel that you need to make a stylistic choice, you can and should document those choices in the comments of your code.



# Where are we now?



# Part A — The Hyper-Text Markup Language

# What is the World Wide Web?

- “A system of interlinked hypertext documents accessed via the Internet” [Wikipedia]
  - The Web is not the Internet; it is built on top of the Internet
- Web pages are formatted text documents written in the Hyper-Text Markup Language (HTML)
- Web browsers, such as *Chrome* or *Firefox*, access and display web pages that are linked to each other via hyperlinks





# HTML

- The Hyper-Text Markup Language:
  - Is a *markup* language, not a programming language
    - It has no internal logic
  - HTML builds the basic structure of a website
  - It uses *markup tags* (or *HTML tags*) to categorise the various elements of the document

```
<!DOCTYPE html>
<html>

  <head>
    <title>My First Web
  </head>

  <body>
    <h1>
      My First Webpag
    </h1>
    <p>This is a paragr
  </body>

</html>
```

# HTML

- Web pages are HTML documents
  - They are plain text files that can be created and edited using any text editor such as TextEdit or Wordpad
  - They usually have a '.html' or '.htm' filename extension
  - They consist of elements such as headers, titles, a body, paragraphs, images, etc, marked up using *HTML tags*

```
<!DOCTYPE html>
<html>

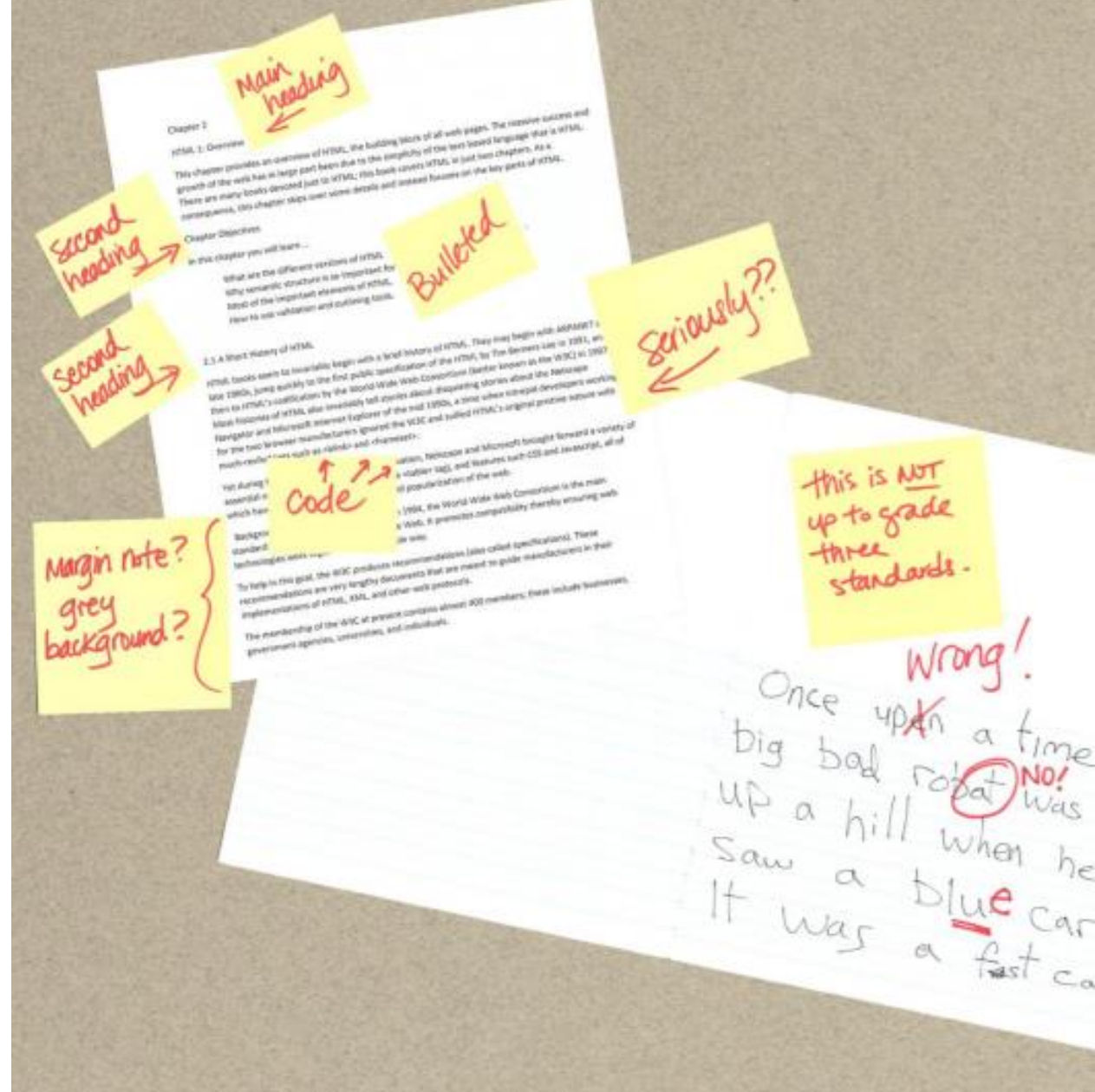
    <head>
        <title>My First Web
    </head>

    <body>
        <h1>
            My First Webpage
        </h1>
        <p>This is a paragr
    </body>

</html>
```

# Markup

- Markup is something we've seen before!
- As HTML, this markup now provides the structure for your webpage.



# Why do we care?

- Assignment 2B will require your program to **identify** and use RegEx to **remove** HTML elements in the dataset provided to you.

# HTML tags

- HTML tags are keywords surrounded by angled brackets, e.g.:

`<title>`

- Tags usually come in start/end pairs, e.g.:

`<title> ... </title>`

`<body> ... </body>`

- Web browsers do not display the HTML tags, but instead use them to interpret the content of the document

# Basic tags

- Any HTML document should contain certain elements:
  - `<!DOCTYPE html>`
    - Identifies the document as HTML5
  - `<html> ... </html>`
    - Marks the HTML portion of the document
  - `<head> ... </head>`
    - Defines header information about the document, e.g., author, overall style, etc

- `<title> ... </title>`
  - Defines the document's title displayed in the browser window frame (not on the web page itself)
  - Usually placed inside the `<head>` element
  - Used as the title of the page when bookmarked, and as the title for the page in search-engine results
- `<body> ... </body>`
  - The displayed content of the document



# Some elements of the document body

- The following elements typically appear in the document's body:
  - `<h1> ... </h1>` to `<h6> ... </h6>`
    - Headings (from largest to smallest)
  - `<p> ... </p>`
    - A paragraph of text
  - `<!-- ... -->`
    - Comments, ignored by the web browser
  - `<br>`
    - A hard-coded line break
  - `<a href="..."> ... </a>`
    - Hyperlinks that connect this document to others
    - The "href" target address attribute can be a URL or a local file
  - ``
    - A visual image
    - The "src" source attribute can be a URL or a local file
    - The "alt" alternative text is displayed if the source image file cannot be found

# Simple formatting

- You can enhance your text with bold, italics and other character formatting options
  - `<b> ... </b>` bold text
  - `<i> ... </i>` italics
  - `<big> ... </big>` big text
  - `<small> ... </small>` small text
- Rather than “hardwiring” the font, you can also let the browser choose an appropriate font for the current context
  - `<em> ... </em>` emphasised text (usually italics when it appears in plain text)
  - `<strong> ... </strong>` standout text (usually boldface when in plain text)

# Special characters (entities)

- Because certain characters have a special meaning in HTML we have to use an “entity” markup to display them
- Entities can be described by numbers or names
  - `&lt;` or `&#60;`; a less-than sign,  $<$
  - `&amp;`; an ampersand,  $&$
  - `&copy;`; copyright,  $\copyright$
- Other special characters such as maths symbols and accented letters can also be specified using entity markups
  - `&#8704;` or `&forall;`; for all,  $\forall$
  - `&#8707;` or `&exist;`; there exists,  $\exists$
  - `&#8713;` or `&isin;`; element of,  $\in$
  - `&#8709;` or `&empty;`; empty set,  $\emptyset$
  - `a&#768;` and `a&#769;`; accented letter a, à and á

# Web documents are just text files

- Web documents are really just text files containing HTML and CSS annotations, so we can download them and manipulate them as character strings
- Python's `urllib` module allows us to read web pages as (long) sequences of text
  - However, what a web server delivers to a Python program may not be the same as what it shows to a standard web browser!
  - And some servers will block attempts by Python programs to access web documents!

```
# Define an address
url = 'http://www.wikipedia.org/'

# Import the function for opening online documents
from urllib.request import urlopen

# Open the web document for reading
web_page = urlopen(url)
```

# Writing to text files

- To create an HTML document using a Python program we just need to open a text file and write the marked-up text into it
- Optionally you can choose the character set used
  - HTML documents would normally be stored as Unicode text
- Writing lines of text into a file is easy in Python using the file `write` method
  - Warning: If the file already exists its contents will be overwritten

```
# Create a new Unicode 8 text file
# in 'write' mode
text_file = open('beatles.txt', 'w',
                  encoding = 'UTF-8')

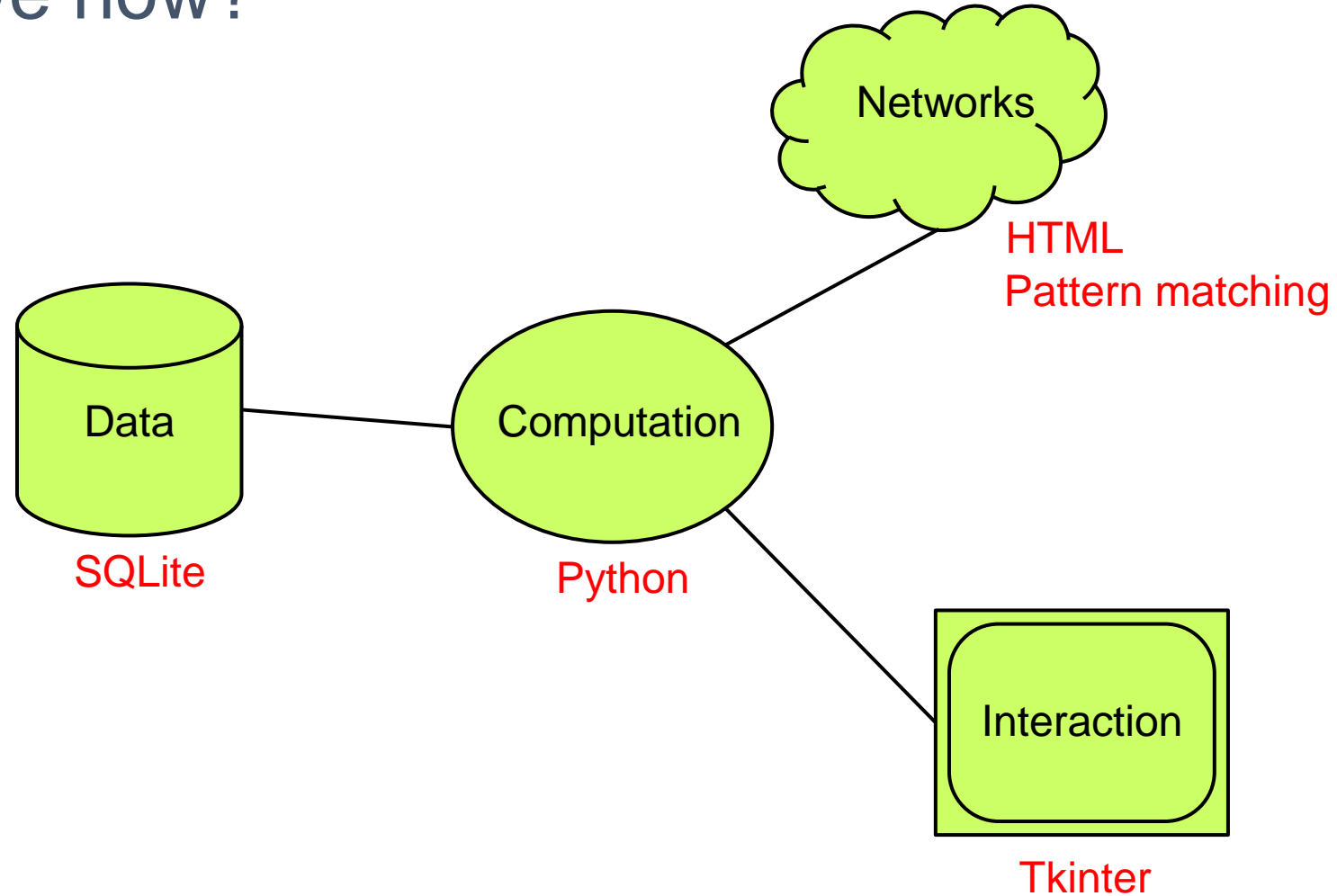
# Write some names into the file, one
# line each
for name in ['John', 'Paul', 'George', 'Ringo']:
    text_file.write(name + '\n')

# Close the file
text_file.close()
```

# Part B — Databases



# Where are we now?



# In this section, we will...

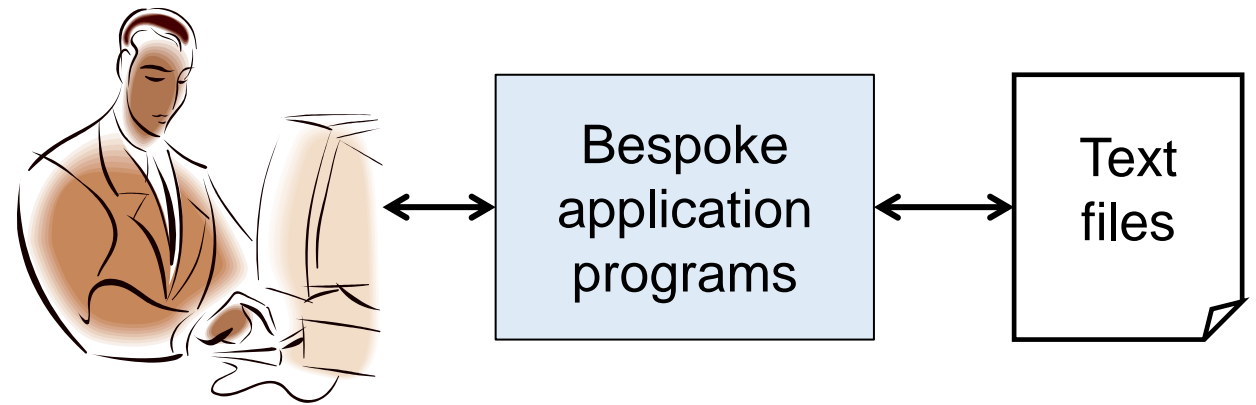
- Introduce relational databases as an example of structured, permanent storage
  - To introduce SQLite's Data Manipulation Language as a way of querying and modifying data in such a database
- Show how to connect our Python programs to an SQLite database
- Some general references:
  - Thomas M. Connolly, Carolyn E. Begg, *Database systems: A practical approach to design, implementation, and management*, available from QUT library, 005.74 343 /5 or 005.74 343 /4
  - Alan Beaulieu, *Learning SQL*, 2009. Electronic resource available from QUT library catalog, 005.7565

# Why do we care?

- Assignment 2B will require you to get information out of a database and display it to the user.
  - In Assignment 2A, we used **dummy data** (strings in your template).
  - In Assignment 2B, you will need to **read from a database**.

# Why use a database?

- We have seen that we can store data in Python lists, but ...
  - This data only exists while our program is running
- To save data permanently we have also seen that we can write data to, and read data from, text files in Python, but ...
  - This requires us to write our own application programs to manage the data
  - The format for the data is not **standardised**
  - The data is **unsafe** because anyone can access the contents of a plain text file



Demonstration files: `text_file_writer.py`, `text_file_reader.py`

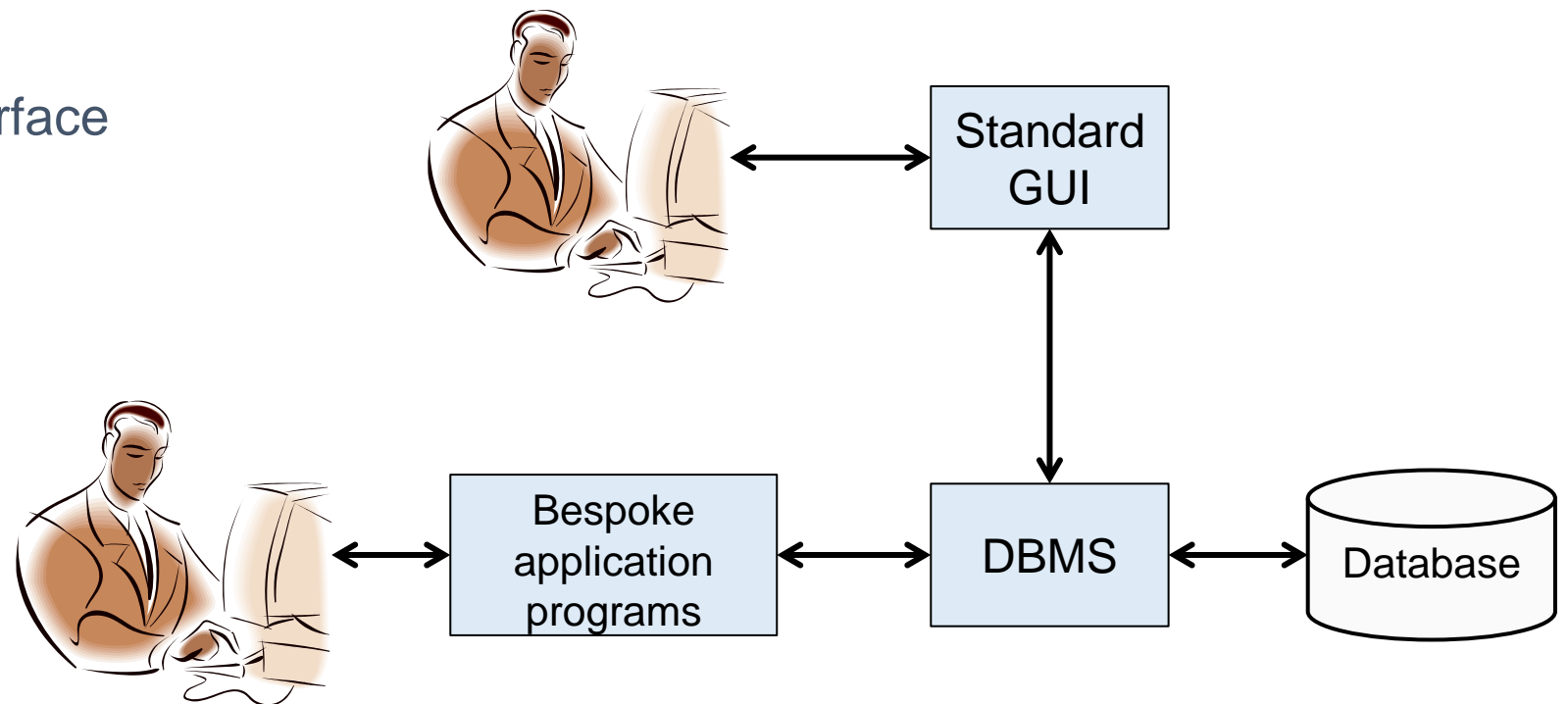
# What is a database?

- Database technology can provide us with program and data independence, plus access control
  - A database is a collection of logically related data with inherent meaning, describing entities (i.e., objects such as people, places, concepts, events, etc.), attributes (properties of objects), and relationships between entities
  - Application programs can manipulate data in the database via a Database Management System (DBMS)



# Using a Database Management System

- We can access a database in two different ways:
  - Via an off-the-shelf GUI
  - By creating our own interface





# What is a database management system?

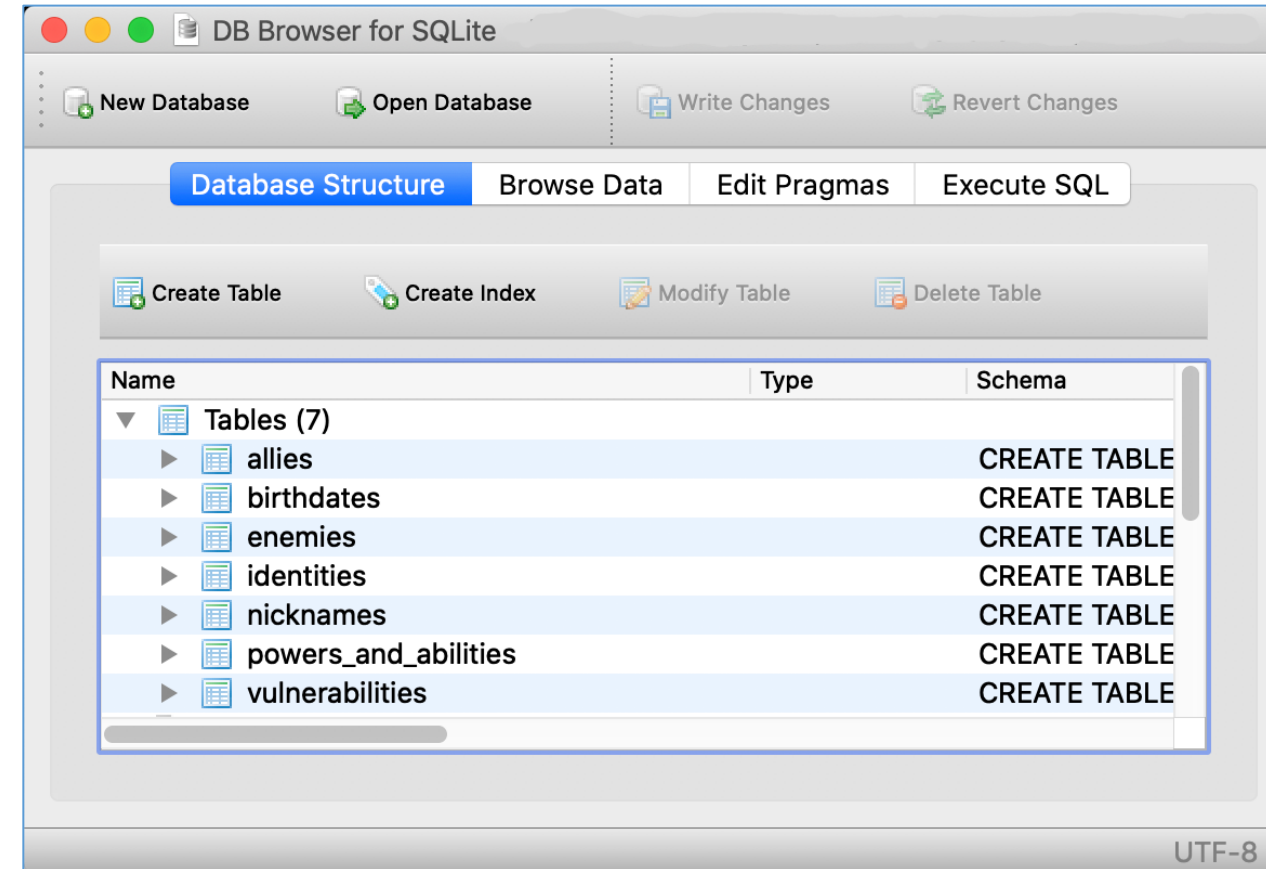
- A database management system (DBMS) is a software system that:
  - enables users to define the database, i.e., to specify the data types and structures, and the constraints on the data to be stored in the database
  - allows users to insert, update, delete, and retrieve data from the database
  - provides controlled access to the database, to prevent unauthorized users from accessing the data
- Structural Query Language (SQL) is designed for managing data via database management systems, specifically for relational databases

## Database Management System



# SQLite and the DB Browser for SQLite

- Here we will use *SQLite* as our query language and the *DB Browser for SQLite* as its user interface
  - As the name suggests, SQLite is a subset of the full SQL language, but is sufficient for our needs
  - Unlike a “full” database system, SQLite does not require a separate DBMS or run-time server; the “database” is just a local file on your computer



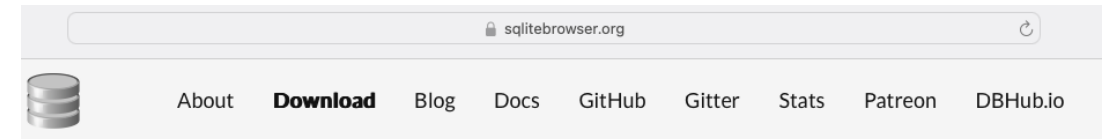
# Structural Query Language

- SQL is a standard relational database language used to:
  - Create databases and tables
  - Insert, modify and delete data from tables
  - Perform queries to retrieve data from tables
- SQLite is a lightweight subset of SQL
- Sequel vs. S-Q-L: doesn't really matter!
  - (gif vs. gif *does* matter)



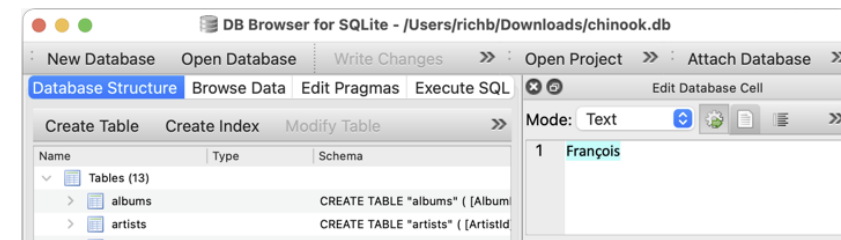
# Install the DB Browser for SQLite

- To complete this week's workshop exercises you need to have a database editor installed compatible with query language *SQLite*
- We recommend the *DB Browser for SQLite* which is free, easy to install and has a very simple user interface
- Just go to <http://sqlitebrowser.org/> and follow the instructions for your particular computing platform

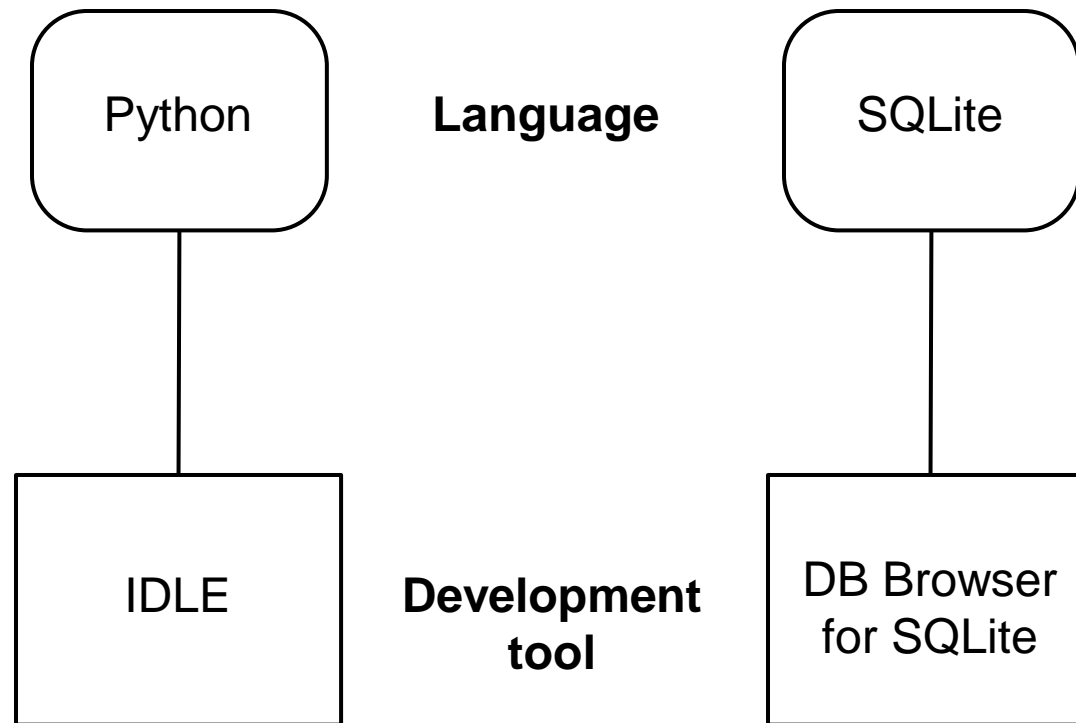


## DB Browser for SQLite

*DB Browser for SQLite* (DB4S) is a high quality, visual, open source tool designed for people who want to create, search, and edit *SQLite* database files. DB4S gives a familiar spreadsheet-like interface on the database in addition to providing a full SQL query facility. It works with *Windows*, *macOS*, and most versions of *Linux* and *Unix*. Documentation for the program is on the *wiki*.



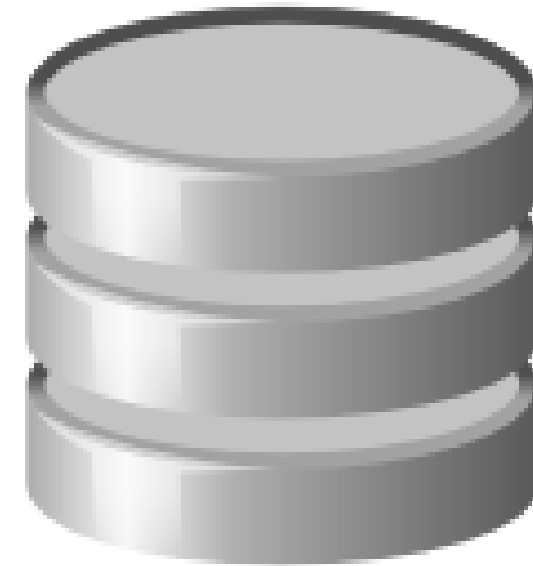
# An analogy



# Database Concepts

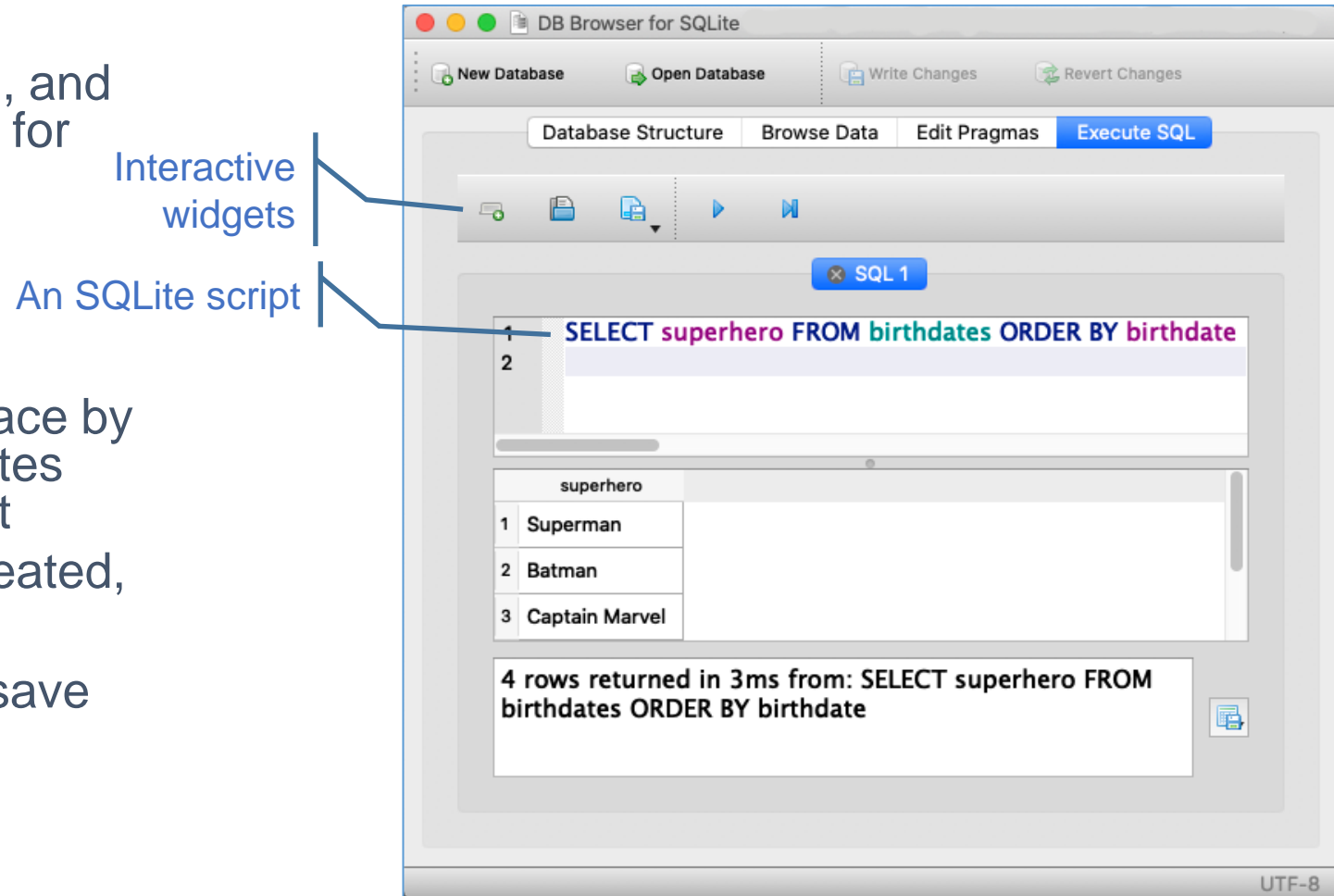
# The *DB Browser for SQLite* interface

- A generic Graphical User Interface (GUI) for accessing a relational database
- Provides all the operations we need to create and maintain a database
- But it does *not* incorporate any knowledge of our specific application domain, so the user has to *manually* perform all necessary operations, either by writing SQLite scripts or by doing sequences of GUI operations



# What the “DB Browser” does for us

- The *DB Browser for SQLite* application, and other such tools like the SQLite plug-in for Firefox, provide a convenient GUI for creating and running SQLite scripts
  - Editing new scripts
  - Loading existing scripts
- When you perform actions in the interface by activating widgets, the tool simply creates and runs a corresponding SQLite script
  - Sometimes it displays the script created, e.g., when defining tables
- In reality, therefore, such tools merely save us a bit of typing





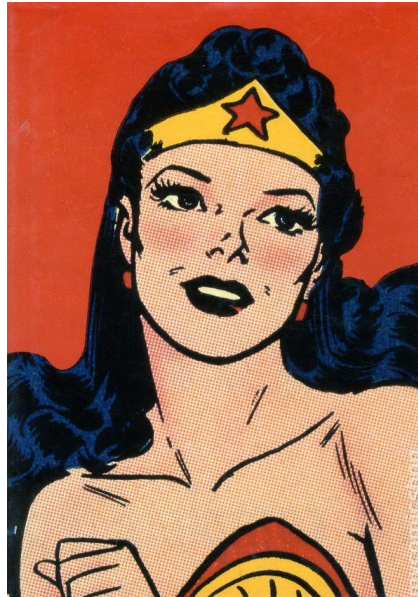
# Example: A DC Golden Age superhero database



Superman



Batman



Wonder Woman



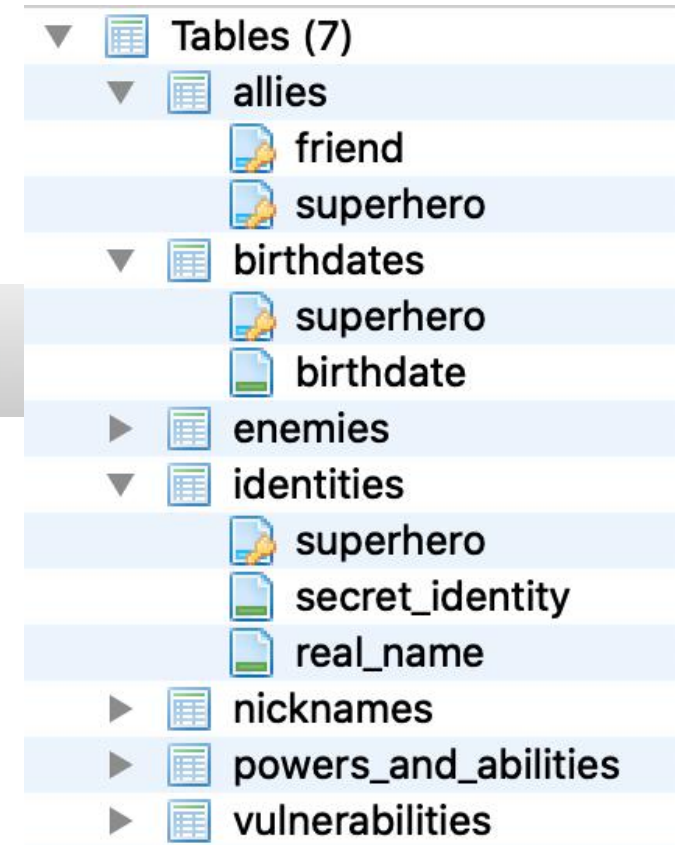
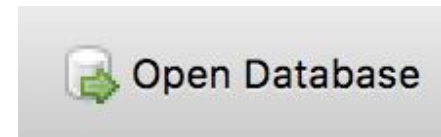
Captain Marvel  
(‘Golden Age’ version)

The Flash (‘Golden Age’  
version)



# Relational databases

- All the data in a *relational database* is stored in a collection of two dimensional *tables* made up of *columns* and *rows*, where each column contains data of a specific *type*
- To open an existing database in the *DB Browser for SQLite* either:
  - Select a “db” file via the Open Database button; or
  - Execute a “dump” script via File > Import > Database from SQL file
  - (Don’t try to drag-and-drop the database into the application because it may challenge you for a password)



# Relational databases

- Database terminology:
  - Tables are sometimes called “relations” or “schemas” (correctly “schemata”)
  - Rows are sometimes called “tuples” or “records”
  - Columns are sometimes called “fields”
  - Types are sometimes called “domains”

rows

	actor	superhero	years
	Filter	Filter	Filter
1	Tom Tyler	Captain Marvel	1941
2	Lewis Wilson	Batman	1943
3	Kirk Alyn	Superman	1948 & 1950
4	Robert Lowery	Batman	1949
5	George Reeves	Superman	1952-58

columns

43

# Restrictions on the format of tables

- Valid database tables usually obey certain restrictions:
  - No two rows are exactly the same, i.e., there are no duplicate rows
  - The order of the rows/columns has no significance to the meaning of the data
  - Each cell in the table contains only a single value, i.e., you can't put several values in the same cell
  - Each column in the table has a distinct name
  - The value in each cell must be compatible with the column's type

Although there are cells with the same value, no two rows are the same





Table: allies		
	friend	superhero
	Filter	Filter
1	Lois Lane	Superman
2	Jimmy Olsen	Superman
3	Perry White	Superman
4	Steve Trevor	Wonder Woman
5	Etta Candy	Wonder Woman
6	Alfred Pennyworth	Batman



# Null values

- When populating the database we sometimes don't have all the data needed to complete a row
  - The data may not yet be available
  - The data may not be necessary
- Cells left empty have the special value "null"

If a hero doesn't have a birth name distinct from their secret identity we can leave it blank

Table:  identities   

	superhero	secret_identity	real_name
	Filter	Filter	Filter
1	Superman	Clark Kent	Kal El
2	Batman	Bruce Wayne	NULL
3	Captain Marvel	Billy Batson	NULL
4	Wonder Woman	Diana Prince	Princess Diana





# Primary keys

- When searching for data in a table we need to be able to uniquely identify each row

In a database, a primary key makes sure each row is unique - like how every student has a different Student ID:

- If the key is just one column (like Student ID), it's called **simple**.
- If it uses two or more columns (like First Name + Birth Date), it's called **composite**.

Each actor has only played one hero, so the "actor" column can be used as the unique primary key in this table

Table:  actors   

	actor	superhero ▼	years
	Filter	Filter	Filter
1	Kirk Alyn	Superman	1948 & 1950
2	George Reeves	Superman	1952-58
3	Tom Tyler	Captain Marvel	1941
4	Lewis Wilson	Batman	1943
5	Robert Lowery	Batman	1949

Some heroes have been played by more than one actor, so "superhero" cannot be used as this table's unique key

# Primary keys

- If no single column contains unique values then the primary key must be created using two or more columns

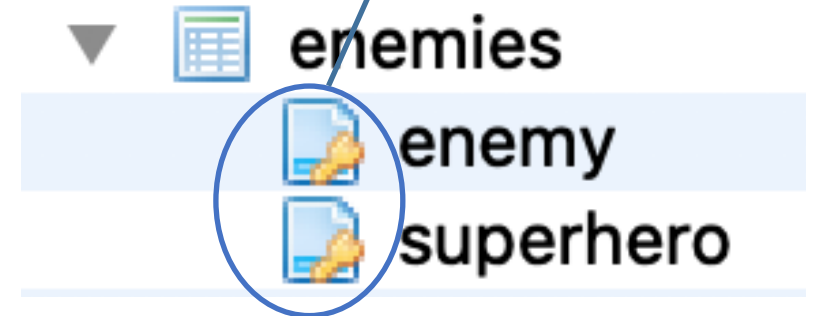


Table: enemies

	enemy	superhero
	Filter	Filter
1	The Riddler	Batman
2	The Penguin	Batman
3	The Joker	Superman
4	The Joker	Batman
5	Mr. Mind	Captain Marvel
6	Lex Luthor	Superman
7	Lex Luthor	Batman
8	Dr. Sivana	Captain Marvel
9	Catwoman	Batman
10	Brainiac	Superman

In DC's shared universe it's possible for the same supervillain to be an enemy of more than one superhero and vice versa

So both columns must be used to create a unique key in this table







# Retrieving data from a database

- In the GUI we can browse the contents of tables interactively
  - The values displayed can be filtered by specific numeric values in a column

Browse Data

This tab in the GUI will show you all the contents of a table

Table:  actors  

	actor	superhero	years
	Filter	Filter	< 1945 
1	Tom Tyler	Captain Marvel	1941
2	Lewis Wilson	Batman	1943

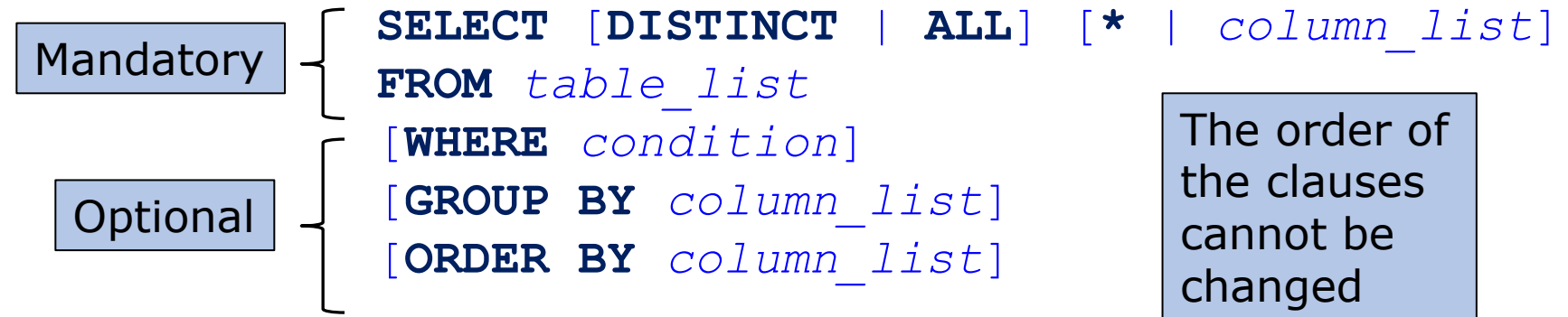
Only show actors who played superheroes before 1945



# Retrieving data from a database

- By far the most common operation we perform on databases is “querying” their contents
- For this reason SQL provides a rich “query” statement, called **SELECT**

- General form of the select statement:



- **WHERE**: filters the rows subject to some condition
- **GROUP BY**: forms groups of rows with the same column value
- **ORDER BY**: specifies the order of the rows
- **DISTINCT** or **ALL**: specifies whether duplicate rows should be returned or not

# The anatomy of a SELECT statement

**SELECT** [**DISTINCT** or **ALL**] [**\*** or *column\_list*]

When you ask a database for information, you can use a **SELECT** statement. It has a few parts you can choose:

- **DISTINCT** or **ALL** (optional):
  - **DISTINCT** means “Only return unique rows”
  - **ALL** means “Return all matching rows, including duplicates” (default)
- **\*** or **column\_list**:
  - **\*** means “show all columns”
  - A **column\_list** means “only show these specific columns” (like StudentID, FirstName)

**SELECT \*** -> show everything.

**SELECT DISTINCT FirstName** -> show each unique name only once.

# An Example

Make	Model	Year	Colour
Toyota	Avalon	2001	Silver
Toyota	Camry	1997	Red
Toyota	Avalon	2002	Blue
Hyundai	Elantra	2018	Blue
BMW	330i	2018	Silver

**SELECT DISTINCT colour**

will return:

Colour
Silver
Red
Blue

**SELECT DISTINCT make, model**

will return:

Make	Model
Toyota	Avalon
Toyota	Camry
Hyundai	Elantra
BMW	330i

# More Examples

```
/* Who are Superman's enemies? */  
SELECT enemy FROM enemies  
WHERE superhero = 'Superman';
```

enemy	
1	Lex Luthor
2	Brainiac
3	The Joker

Demo: Execute these queries from `query_superheroes.sql`

# Even More Examples

```
/* Which superheroes were created in the 1940s? */  
SELECT superhero FROM birthdates  
WHERE birthdate BETWEEN 1940 AND 1949;
```

superhero	
1	Captain Marvel
2	Wonder Woman

A variety of Boolean operators  
can be used (some with  
syntax different from Python)

Demo: Execute these queries from `query_superheroes.sql`

# Even More Examples

```
/* How many enemies does each superhero have? */  
SELECT superhero, COUNT(*) AS 'num_enemies'  
FROM enemies GROUP BY superhero;
```

	superhero	num_enemies
1	Batman	5
2	Captain Marvel	2
3	Superman	3

We can name  
columns in the  
result set

Demo: Execute these queries from `query_superheroes.sql`

# Even More Examples

*/\* Which superheroes have super speed as one of their powers? \*/*

```
SELECT * FROM powers_and_abilities  
WHERE power LIKE '%speed%';
```

	superhero	power
1	Superman	Faster than a speeding bullet
2	Captain Marvel	The speed of Mercury

Demo: Execute these queries from `query_superheroes.sql`

# Manipulating a database from a program



# A Python module for SQLite

- To interact with a database from within a Python program we need an appropriate API module
- Fortunately the `sqlite3` module comes pre-packaged with Python implementations
- It provides Python functions that can do each of the steps we did manually *DB Browser for SQLite* GUI to examine the contents of the database

```
# Import the SQL functions
from sqlite3 import *

# Create a connection to the database
connection = connect(database = 'superheroes.db')

# Get a local view of the database
superheroes_db = connection.cursor()

# Execute an SQL script, a query in this case
superheroes_db.execute('''SELECT superhero, secret_identity
                        FROM identities
                        ORDER BY superhero ASC''')

# Fetch and print the list of heroes returned
for hero, identity in superheroes_db.fetchall():
    print(hero + ' (' + identity + ')')

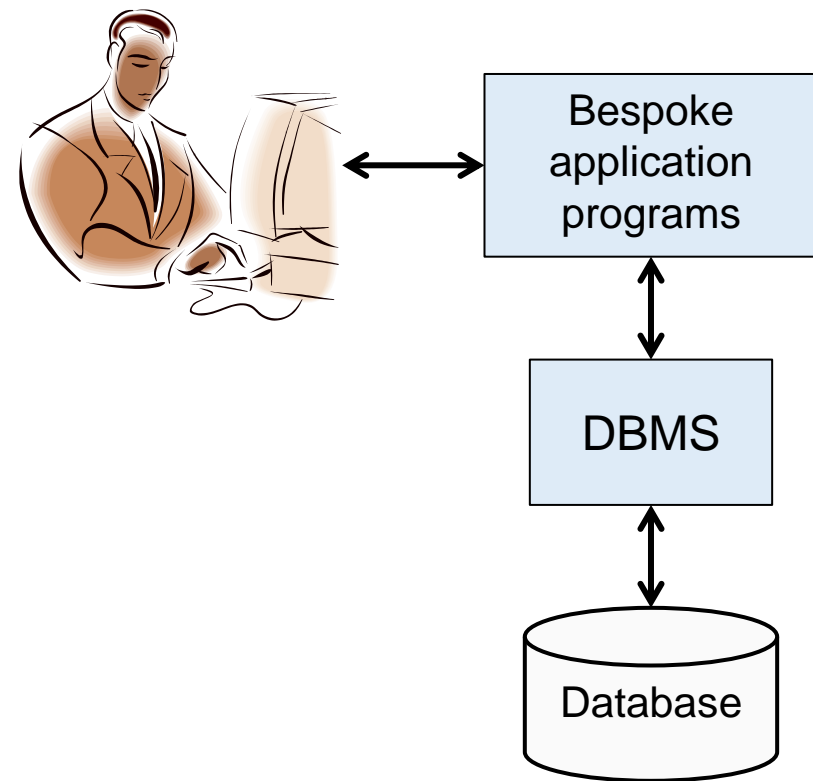
# Close the database
superheroes_db.close()
connection.close()
```

57

Demonstration files: `access_superheroes.py`, `print_superheroes.py`

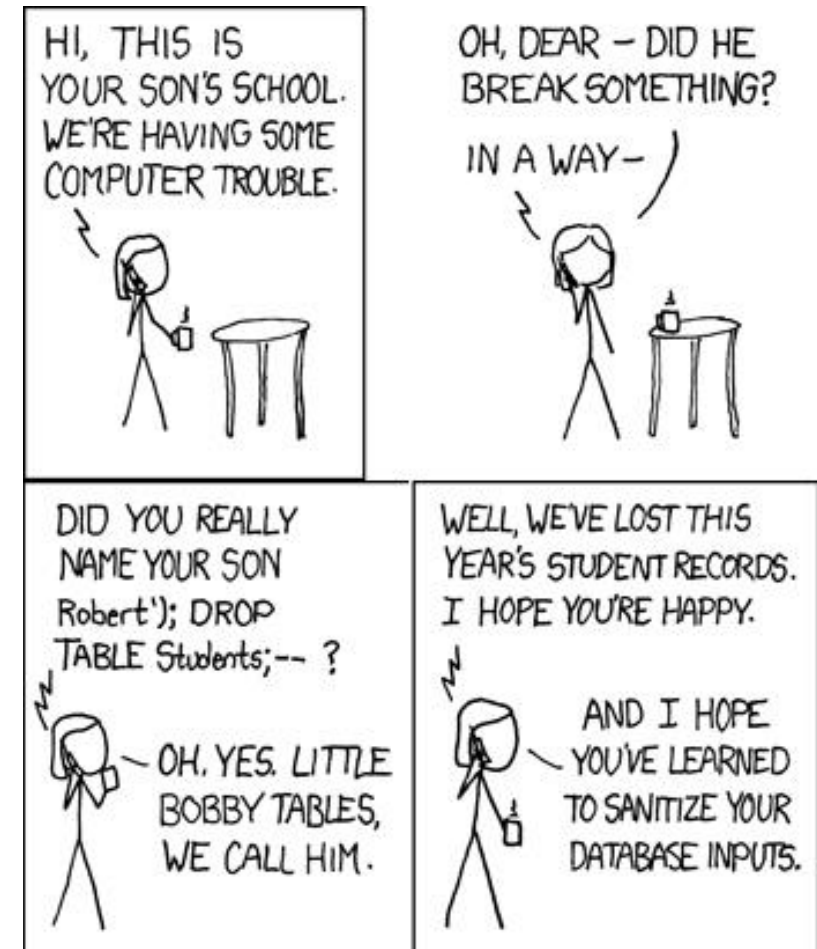
# Creating our own interactive programs to access databases

- The Python demonstrations so far have accessed a database, but haven't involved any user interaction
- To be truly useful, we want to build an IT system in which the user can interact in an efficient way with the database via a Python program



# Security warning!

- The preceding demos show that it's possible to get a Python program to construct and execute an SQL script at run time
- This means we can't know beforehand what the script will do!
- This is the source of one of the biggest cyber security vulnerabilities in large IT systems
- Bad guys can exploit this capability to upload and run their own code to access and manipulate a database!
- This is known as an "SQL injection attack"



# Before next week ...

- Submit Assessment Task 2A!
  - Do so early and often!
- Complete this week's workshop exercises