Cheat Sheet

Setting Up a Python Virtual Environment Running a Flask Server

If you have no virtual environment setup, or need to re-setup, use the following steps:

1. Open Terminal (Mac) or Command Prompt (Windows)
2. Change (`cd`) to your project directory (i.e. `cd Desktop\212-Burgers`) -- if the directory does not exist, you will have to create it first
3. Create a new virtual environment using: `python3 -m venv env`
4. Activate the new environment using: `source env/bin/activate`
5. Install Flask using pip: `pip install Flask`
6. Specify the main Flask file (in this case, run.py): `export FLASK_APP=run.py`
7. For debugging mode, enter: `export FLASK_DEBUG=1`
8. Start the Flask server: `flask run`

Note: For Windows, use `set` instead of `export` for steps 6 and 7.

If your virtual environment is already setup (i.e. you have already created a `env` directory on the machine you are working on), use the following steps:

1. Open Terminal (Mac) or Command Prompt (Windows)
2. Change (`cd`) to your project directory (i.e. `cd Desktop\212-Burgers`)
3. Activate the environment using: `source env/bin/activate`
4. Specify the main Flask file (in this case, run.py): `export FLASK_APP=run.py`
5. For debugging mode, enter: `export FLASK_DEBUG=1`
6. Start the Flask server: `flask run`

SQLite3 Commands

Starting SQLite3 in the Terminal

To start sqlite3, `cd` to the directory in which your database file resides and type :

`sqlite3 filename`

(where *filename* is the name of your database file)

Dot Commands

There are a number of useful SQLite dot commands you should be aware of:

Command	Description
`.exit`	exits back to command prompt
`.tables`	lists all tables
`.headers on`	displays column headers for queries
`.mode column`	displays left-aligned columns for queries
`.width 5 30 10`	specifies the width of each column in characters (in this case, the first 3 columns)

Creating a New Table

Use `CREATE` to create new tables.

Here is an example table:

```
CREATE TABLE burgers(
  id INTEGER PRIMARY KEY,
  burger TEXT UNIQUE NOT NULL,
  price FLOAT NOT NULL
);
```

- the `id` column holds an integer that serves as the primary key, therefore it will automatically increment with each new record inserted. This ensures that every `id` value is unique.
- the `burger` column accepts a unique text value (i.e. no duplicate burger names are permitted), and will not accept empty/null values.
- the `price` column accepts a floating-point value, and null values are not permitted.

Inserting Values into a Table

Use `INSERT` to insert new rows into a table. Here is an insert command for the burgers table:

```
INSERT INTO burgers(burger, price)
VALUES ('Classic Burger', 4.99);
```

Note: It is not necessary to specify an `id` value for this entry, as this has been defined as an `INTEGER PRIMARY KEY` column.

Deleting a Table

To delete a table, use the `DROP` command:

```
DROP TABLE burgers;
```

Listing Records

Use `SELECT` to query table(s).

For example, to display all records in the burgers table, use:

```
SELECT * FROM burgers;
```

The Where Clause

The `WHERE` clause is used to specify a condition. Below are a few examples of it in action.

Select all burgers that cost more than 5.50:
`SELECT * FROM burgers WHERE price > 5.50;`

Select all burgers that cost 4.99:
`SELECT * FROM burgers WHERE price == 4.99;`

Select all burgers that cost 4.99, or more than 5.50:
`SELECT * FROM burgers WHERE price == 4.99 OR price > 5.50;`

Altering/Editing Entries

Use `UPDATE` to change any values. You will usually need to couple this with a `WHERE` clause.

For example, to change the name of the "Classic Burger" to "El Cheapo", use:

```
UPDATE burgers
```

```
SET burger = 'El Cheapo'
WHERE burger = 'Classic Burger';
```

Note: It is usually best to use the `id` in the `WHERE` clause comparison if you wish to ensure that you affect just a single row. For example:

```
UPDATE burgers
SET burger = 'El Cheapo'
WHERE id = 1;
```

Aggregate Fucntions

Aggregate functions are useful in various scenarios. Below are a few examples.

Display the most expensive burger:
`SELECT burger,max(price) FROM burgers`

Display the cheapest burger:
`SELECT burger,min(price) FROM burgers`

Display the average burger price:
`SELECT avg(price) FROM burgers;`

Display the sum of all burger prices:
`SELECT sum(price) FROM burgers;`

Count how many burgers there are in total:
`SELECT count(burger) FROM burgers;`

SQLite Resources

If you wish to explore your SQLite database visually, try:
http://sqlitebrowser.org/

For more on how to use SQLite, refer to:
https://www.tutorialspoint.com/sqlite/