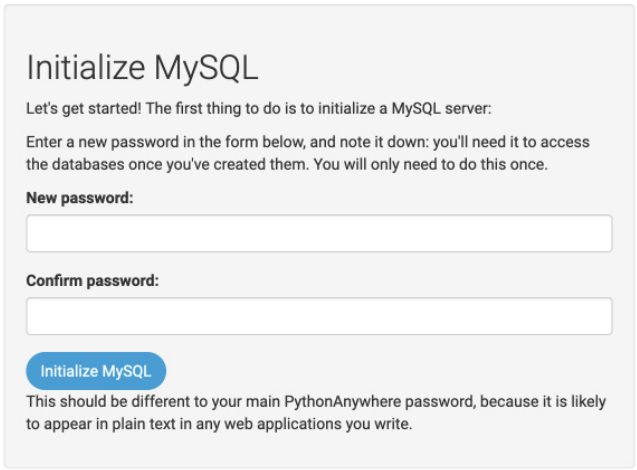
Create a new database on PythonAnywhere

Let's configure your PythonAnywhere account to run the MariaDB-enabled version of your webapp.

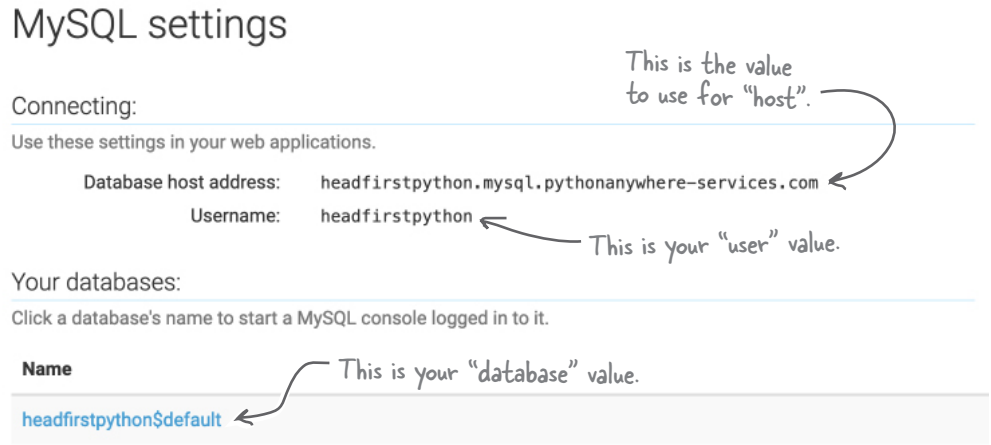
To get going, log into your PythonAnywhere account then click on the **Databases** tab. PythonAnywhere uses MySQL (not MariaDB), but that's OK as MariaDB is designed to be compatible with MySQL to a very high degree. Enter the password you want to use with your database into the input boxes displayed on the **Databases** tab:



Use the same password as for your local database, which is "swimpasswd" for us, but should be something secret for you.

Be sure to read this!

With your password entered, click on the big blue button. After a moment or two, your browser screen refreshes to confirm your database now exists. You already know the password to use, and you're now provided with the three other pieces of credential information needed to establish a connection to your database from your code:



This is the name PythonAnywhere has given to your database. It looks a little strange (what with that embedded dollar sign), but don't let that worry you.

This is the value to use for "host".

This is your "user" value.

This is your "database" value.

Adjust your database credentials dictionary

Now that you know the four pieces of PythonAnywhere database credential information, you may be tempted to adjust your `db_details` dictionary within your *data_utils.py* code to use these cloud-specific values. However, if you do that, your code won't run on your local machine any more…

You could make a mental note to adjust your code depending on what you're doing, but—trust us—you're going to forget at some point.

As luck would have it, the PSL can help here thanks to its included `platform` module. Here's code that exploits `platform`, allowing you maintain a single copy of your code that selects the credentials to use based on the computing platform it's running on: either your local computer or PythonAnywhere.

Once again, the Standard Library comes to our rescue.

Edit data_utils.py to support multiple locations

Let's use VS Code to adjust the top of *data_utils.py* so that it looks like the code shown below, which selects the database credentials after determining the platform your code is running on, either *locally* or on PythonAnywhere.

```
import DBcm

## db_details = "CoachDB.sqlite3"

import platform

if "aws" in platform.uname().release:
    # Running on PythonAnywhere.
    db_details = {
        "host": "headfirstpython.mysql.pythonanywhere-services.com",
        "database": "headfirstpython$default",
        "user": "headfirstpython",
        "password": "swimpasswd",
    }
else:
    # Running locally.
    db_details = {
        "host": "localhost",
        "database": "swimDB",
        "user": "swimuser",
        "password": "swimpasswd",
    }
```

This is a bit of a hack, but does the job. If the string "aws" appears in the release string associated with the platform's name, the assumption is your code is running on PythonAnywhere. Of course, this code will likely break if PythonAnywhere decides to move away from Amazon's cloud platform sometime in the future. That could never happen, could it...?!?

The security conscious among you may well be having a kitten right about now, 'cause here we are not only sharing our passwords in this book but putting them in our code, too!! If we had another 600 pages available to us, we could really dig into how to write security-aware code, but that's not this book's goal.

Copying everything to the cloud

You've created your database on PythonAnywhere, and you've adjusted your *data_utils.py* code *one last time* to support the selection of the correct database credentials to use (based on which platform your code thinks it's running on). You're now ready to copy your code and data to PythonAnywhere.

Preparing your code and data for upload

First up: preparing your code. Use your operating system's ZIP utility to compress your *webapp* folder, giving the compressed file the name *webapp-update.zip*. This will zip all the files in *webapp* and below.

You don't need *all* the files that are in your "webapp" folder, so feel free to move your notebook files (.ipynb) out of there before performing your zip. Only the webapp's code, templates, CSS, and JSON needs to be uploaded.

Now for your data.

You could upload the files your created earlier in this chapter, *schema.sql* and *data.sql*, then use the `source` command on PythonAnywhere to run these commands on your cloud-hosted MySQL server. However, the fact that you're using MariaDB locally, twinned with the fact that MariaDB and MySQL are designed to be compatible, lets you use a MySQL tool to migrate all the details of a database on one machine to another *using a single command*.

The `mysqldump` command (included with MySQL/MariaDB) lets you create a copy of not just the data contained in any database, but also its table definitions. The file produced contains *both* the database table definitions *and* the data.

A typical use of "mysqldump" is to create backups of running databases.

This command can be used to "dump" everything from your local `swimDB` database into a file called *db.sql*:

```
sudo mysqldump swimdb > db.sql
```

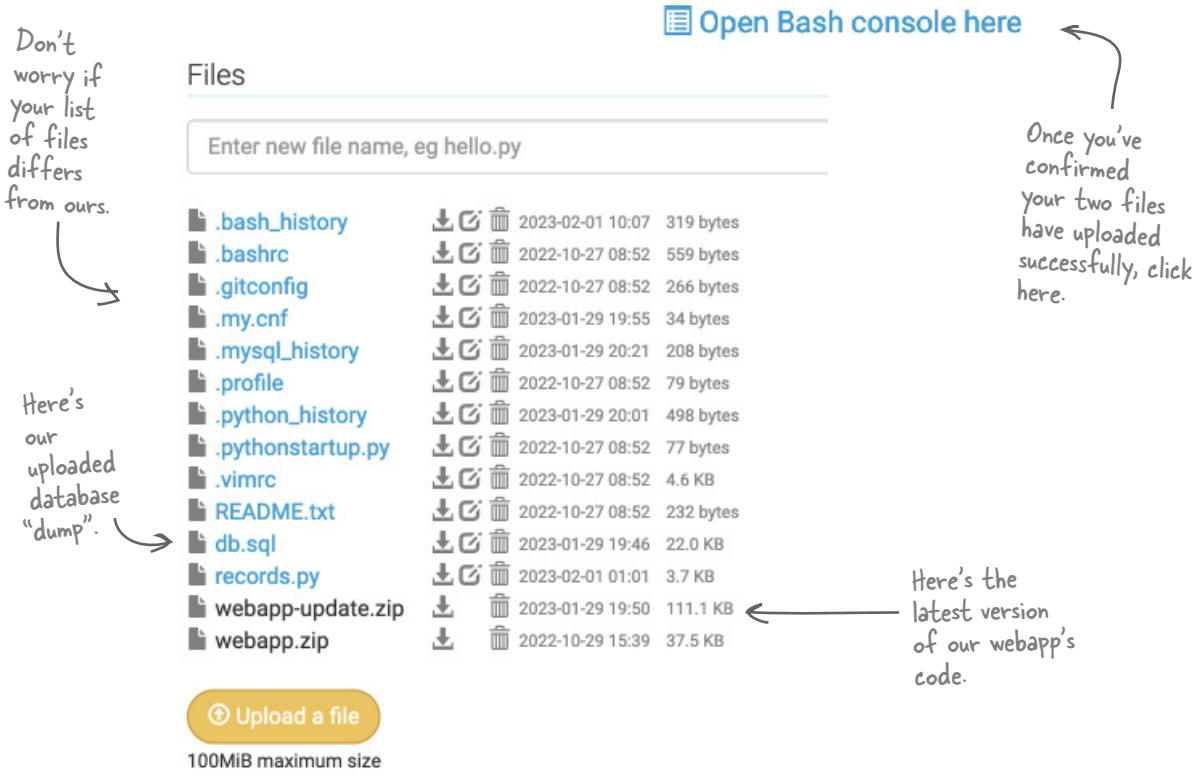
This command should work unchanged if you are using Linux or macOS. Windows users are unlikely to have access to the "sudo" command, so they should check the Windows-specific MariaDB documentation to determine how best to run "mysqldump" on Windows. One suggested workaround is to open the "Maria DB Command Prompt" from your Windows start menu, then run "mysqldump --user=swimuser --password=swimpasswd swimdb > db.sql".

You now have your code in the *webapp-update.zip* file, and all the details of your database in the *db.sql* file. It's time to pop them up on PythonAnywhere…

This could not be any easier: simply use the "Upload a file" button on the PythonAnywhere **Files** tab to copy your two files to the cloud.

Update your webapp with your latest code

With *webapp-update.zip* and *db.sql* uploaded, click on the "Open Bash console here" link:



Once your Bash console opens, type `unzip webapp-update.zip` to extract your uploaded code:

```
19:52 ~ $ unzip webapp-update.zip
Archive:  webapp-update.zip
  inflating: webapp/update_tables.py
  inflating: __MACOSX/webapp/._update_tables.py
  inflating: webapp/app-pre-database.py
  inflating: __MACOSX/webapp/._app-pre-database.py
  inflating: webapp/db.sql
  inflating: webapp/CreateDatabaseTables.ipynb
  inflating: __MACOSX/webapp/._CreateDatabaseTables.ipynb
  inflating: webapp/data.sql
  inflating: webapp/.DS_Store
  inflating: webapp/ReusedCode.ipynb
  inflating: __MACOSX/webapp/._ReusedCode.ipynb
replace webapp/WebappSupport.ipynb? [y]es, [n]o, [A]ll, [N]one, [r]ename:
```

PythonAnywhere's "unzip" command overwrites the files in your existing "webapp" folder. Be sure to enter uppercase "A" when prompted to confirm this is what you mean to do.

Just a few more steps...

You're nearly there.

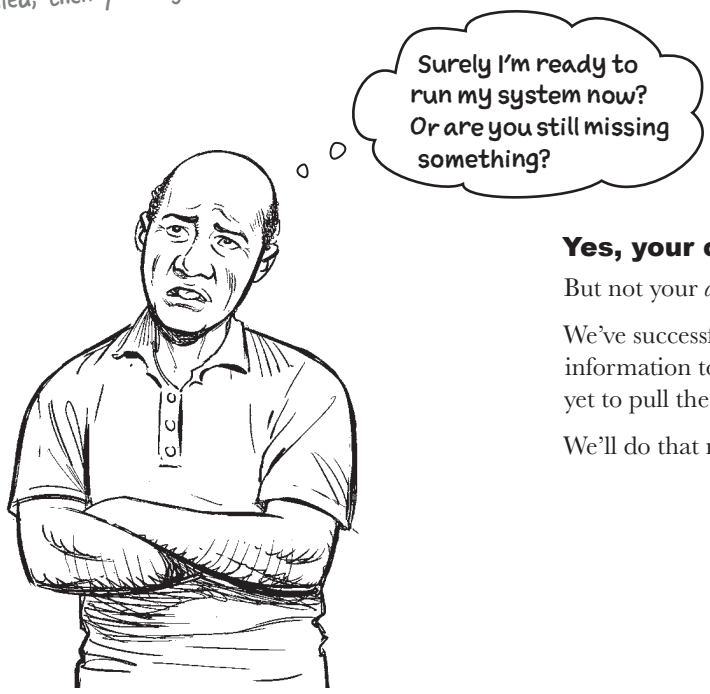
You've uploaded your latest webapp code and unzipped it to your PythonAnywhere *webapp* folder. However, before you run it, you need to install the `DBcm` module as your webapp code now depends on this module being available. It's not installed on PythonAnywhere by default, but can be easily added.

Continuing to work at the Bash console from the last page, ask Python's **pip** command to install `DBcm`:

```
python3 -m pip install DBcm
```

This may well be the first time you've seen this command since this book's Introduction, as you've used the %pip command within your notebooks to do the same thing. This command line installs a named package from PyPI into your Python environment.

Running this command produces a slew of status messages. If you see a message near the bottom of the output telling you "DBcm" has been successfully installed, then you're golden.



Yes, your code's ready, Coach...

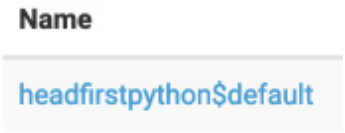
But not your *data*.

We've successfully uploaded your database information to PythonAnywhere, but we've yet to pull the data into their MySQL server.

We'll do that now.

Populate your cloud database with data

Return to your **Databases** tab on PythonAnywhere, then click on the blue link associated with your recently created database:



Clicking this link opens a MySQL console connected to the named database. Note that you are automatically logged in.

As with your local MariaDB database engine, the `source` command is your friend here. Enter `source db.sql` to run all the SQL commands and queries in that file. When the process completes, your three tables have been created *and* populated with a copy of your local data:

You'll see many more "Query OK" messages appear on screen than are being shown here.

```
mysql> source db.sql
Query OK, 0 rows affected (0.00 sec)

Query OK, 0 rows affected (0.00 sec)

Query OK, 0 rows affected (0.00 sec)

Query OK, 0 rows affected (0.00 sec)

Query OK, 0 rows affected (0.00 sec)
```

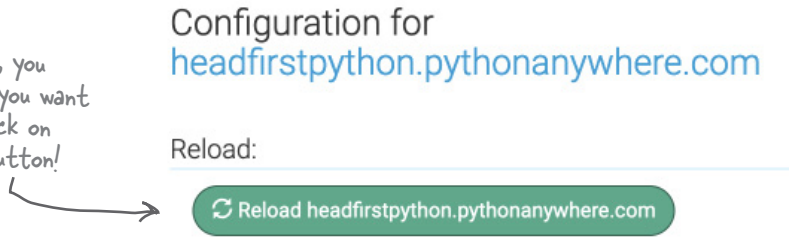
```
mysql> select count(*) from events;
+----------+
| count(*) |
+----------+
|       14 |
+----------+
1 row in set (0.00 sec)

mysql> select count(*) from swimmers;
+----------+
| count(*) |
+----------+
|       23 |
+----------+
1 row in set (0.00 sec)

mysql> select count(*) from times;
+----------+
| count(*) |
+----------+
|      467 |
+----------+
1 row in set (0.00 sec)
```

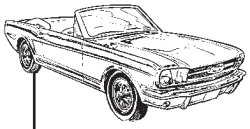
You can check that everything has gone to plan by typing in these three queries. This looks good to us as these row totals match those from earlier.

It's time for a PythonAnywhere Test Drive

Your updated code is loaded into PythonAnywhere, and a copy of your local database is running on PythonAnywhere, too. All that's left to do is take this updated version of your webapp for a spin. Before you do, return to the *PythonAnywhere* **Web** tab, then click on the big green button to restart your webapp:

Go on, you know you want to: click on this button!

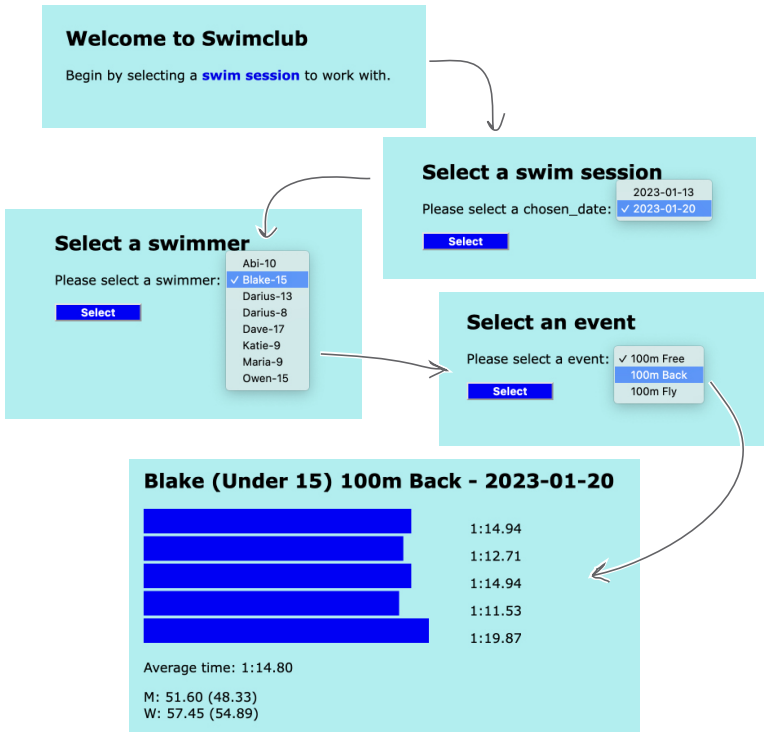




Test Drive

With your webapp reloaded, click on your webapp's blue link to take it for a spin.

It should come as no surprise that your webapp running on PythonAnywhere with MySQL matches the behavior of it running on your local computer with MariaDB.



That's great news. But... perhaps you haven't been so lucky?

If something went wrong, you may have been presented with a terse error message, something that looks like one of these:

This message has to be the bane of every web developer's existence, eh?

Internal Server Error

This message tells you a little more, but isn't very specific, is it?

Something went wrong :-(

Something went wrong while trying to load this website; please try again later.

If it is your site, you should check your logs to determine what the problem is.

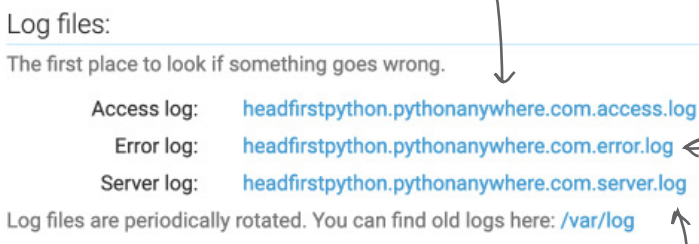
Now don't panic if you are seeing either of these messages. Flip the page to learn what to do if something like this happens to you.

Is something wrong with PythonAnywhere?

Nine times out of ten, when your webapp refuses to run on PythonAnywhere, it's something that you've done (or forgotten to do).

Your first port of call should always be to return to the **Web** tab on PythonAnywhere and scroll down until you see the section describing your webapp's log files:

The "Access log" contains information on successful interactions with your server.



The "Error log" contains information on requests to your webapp that have resulted in errors (producing those plain-as-Jane error messages from the previous page). Looking at this file should be your first port of call when something breaks.

The "Server log" file contains information on PythonAnywhere's backend technology and, although it may be less relevant to the running of your webapp's code, does often contain messages that might just make you chuckle. If you know the issue can't possibly be with your code, check this log file for issues with PythonAnywhere.

```
During handling of the above exception, another exception occurred:
**NO MATCH**
Traceback (most recent call last):
  File "/usr/local/lib/python3.10/site-packages/flask/app.py", line 2077, in wsgi_app
    response = self.full_dispatch_request()
  File "/usr/local/lib/python3.10/site-packages/flask/app.py", line 1525, in full_dispatch_request
    rv = self.handle_user_exception(e)
  File "/usr/local/lib/python3.10/site-packages/flask/app.py", line 1523, in full_dispatch_request
    rv = self.dispatch_request()
  File "/usr/local/lib/python3.10/site-packages/flask/app.py", line 1509, in dispatch_request
    return self.ensure_sync(self.view_functions[rule.endpoint])(**req.view_args)
  File "/home/headfirstpython/webapp/app.py", line 23, in display_swim_sessions
    data = data_utils.get_swim_sessions()
  File "/home/headfirstpython/webapp/data_utils.py", line 17, in get_swim_sessions
    with DBcm.UseDatabase(db_details) as db:
  File "/home/headfirstpython/.local/lib/python3.10/site-packages/DBcm.py", line 96, in __enter__
    self.conn = mysql.connector.connect(**self.configuration)
  File "/usr/local/lib/python3.10/site-packages/mysql/connector/__init__.py", line 272, in connect
    return CMySQLConnection(*args, **kwargs)
  File "/usr/local/lib/python3.10/site-packages/mysql/connector/connection_cext.py", line 94, in __init__
    self.connect(**kwargs)
  File "/usr/local/lib/python3.10/site-packages/mysql/connector/abstracts.py", line 1052, in connect
    self._open_connection()
  File "/usr/local/lib/python3.10/site-packages/mysql/connector/connection_cext.py", line 251, in _open_connection
    raise errors.get_mysql_exception(msg=exc.msg, errno=exc.errno,
mysql.connector.errors.DatabaseError: 2003 (HY000): Can't connect to MySQL server on 'localhost:3306' (111)
```

Here's an example entry in the "Error.log" file. Note the very last line, which is a pretty big clue as to what's broken.