# Storing weather data in a database

Sure, a display of the current conditions on screen is useful. But if you want to use data on past weather conditions, you need a way to collect it which stores the data in a form which is easier to access and process.

You are going to set up a sqlite3 database, create a schema and add functionality into your existing python that will log each reading it takes into this database.

First of all, you need to install **sqlite3**. Open LXTerminal and type:

sudo apt-get update install sqlite3

Press enter and confirm if prompted.

When this is done, navigate to your working directory (where your python files are) and type:

sqlite3 weather.db

Press enter. Sqlite3 will open in the command line with a new, empty database in the current directory, which will be saved into weather.db.

First you are going to create a new table to store the data from the very first sensor you hooked up – the anemometer. To do this we are going to write a line in a language called **Structured Query Language:**

CREATE TABLE anemometer (timestamp DATETIME, windspeed NUMERIC);

Don’t forget the semicolon, as all SQL statements must end with one. Press enter.

Enter .tables (notice the dot) and ‘anemometer’ should be displayed (the .tables command displays a list of tables in the database.)

Now we have a database ready with a table that can hold records with two fields: a timestamp for the moment at which the reading was taken, and a numeric field that will hold the wind speed read during that most recent time interval.

Open up your anemometer.py file in IDLE.

Add two import to the top of the file where you imported time, like so:

import time, sqlite3

**sqlite3** is imported so that you can connect to the database you just created.

Add the following code before the main part of your existing code:

con = sqlite3.connect(“weather.db”)

cur = con.cursor()  
print “Connected to database.”

This sets up a connection to the database and creates a cursor to allow us to execute SQL statements and receive results. Next, define this function:

def log(windspeed):  
 cur.execute(“INSERT INTO windspeed VALUES(DATETIME(‘now’), {})”  
 .format(windspeed))

This function will insert a given wind speed into the database along with the appropriate timestamp. Now in your main code, whenever you print a reading, call it, for example:

windspeed = pulses \* 0.1492  
print “windspeed: {0:0.2f}mph”.format(windspeed)  
log(windspeed)

Finally, at the end of your code, put:

con.close()

To close the connection to the database before the script terminates. However, as the script is, this will never be called. Can you think of a way to make sure it will be?

Try creating a table in the same database for each sensor and updating the python as appropriate.

Further Reading / Reference:

<http://www.tutorialspoint.com/sqlite/sqlite_quick_guide.htm>  
<http://www.w3schools.com/sql/default.asp>  
<http://www.tutorialspoint.com/sql/sql-rdbms-concepts.htm>  
<http://learnpythonthehardway.org/book/ex11.html>  
<https://en.wikibooks.org/wiki/Python_Programming/Threading>

# Setting up a local web page to display data

Now that you have all this data stashed away nicely, you can take it back out again and present it nicely. A simple and convenient way to view the data is via a web page that has dynamic content created from records in your database.

To create an environment where we can generate this content, you are going to use something called **flaskr.** This will allow you to ‘host’ a ‘website’ on your local machine- i.e. your Raspberry Pi.

Enter the following into LXTerminal:

sudo pip install Flask

In your working directory, create a folder called **flaskr**. Inside this, create two new folders called **static** and **templates.**

Inside the flaskr folder that you just created, create a new file called **weatherpage.py** and open it in IDLE.

The following code is all the code you need in order to produce content for just the anemometer data that you made available earlier.

import sqlite3  
  
from flask import Flask, request, session, g, redirect, url\_for, abort, \  
 render\_template  
  
DATABASE = ‘/home/pi/weather.db’  
DEBUG = True  
SECRET\_KEY = ‘development key’  
USERNAME = ‘admin’  
PASSWORD = ‘default’

# A Flask object made with this class’s name as an argument  
app = Flask(\_\_name\_\_)   
app.config.from\_object(\_\_name\_\_) # using the constants we defined above  
  
def connect\_db():  
 return sqlite3.connect(app.config[‘DATABASE’])

@app.before\_request(): #What to do when any page is requested  
 g.db = connect\_db()

@app.teardown\_request #What to do when the page has been generated  
def teardown\_request(exception):  
 db = getattr(g, ‘db’, None)  
 if db is not None:  
 db.close()  
  
@app.route(‘/’) #Generate content for the default page here (localhost/)  
def index():

#Current average windspeed, assuming that a value is collected  
 #several times in the specified 5 minutes.  
 cur = g.db.execute(“SELECT avg(windspeed) where timestamp >  
 datetime(‘now’, ‘-5 minutes’)”)

currentwindspeed = cur.fetchone()[0] #First field of first row  
 if currentwindspeed == None:  
 currentwindspeed = 0.0 #Make sure it’s safe to output

return render\_template(‘index.html’, windspeed=currentwindspeed)  
  
if \_\_name\_\_ == ‘\_\_main\_\_’: #if py execution was started from this script  
 app.run(host=’127.0.0.1’) #Use 0.0.0.0 for access over a LAN

Pretty simple, right? Now all we need to do is create an HTML layout for the data to be inserted into.

In your templates folder, create a new file called layout.html. This will contain the majority of the markup, that which does not change depending on the content.

For now, just put something basic like this:

<!doctype HTML>  
<head>  
 <title>Weather Data</title>  
 <link rel=stylesheet type=text/css href=”{{url\_for(‘static’,  
 filename=’style.css’)}}”>  
</head>  
<body>{% block windspeedblock %}  
</body>  
</html>

“{% block windspeed %}” is an example of a **Jinja** tag. The block tag allows you to mark where a block with the same name should be placed. The contents of the block are defined in a file like the one you are about to create.

Create a new file in the same folder (templates) and call it index.html. This is the file referred to in the return statement of the index() function in the flaskr python.

In this file, put:

{% extends “layout.html” %}  
{% block windspeedblock %}  
 <p>The current wind speed is {{windspeed}}mph.</p>  
{% endblock %}

You have now told Jinja to add the content in windspeedblock into the windspeedblock in layout.html. {{windspeed}} denotes printing the variable *windspeed,* which was defined with the name *currentwindspeed* in weather.py and passed as a return value as *windspeed.*

All you need to do now is to **run weather.py.** If it starts successfully, you can now open up a browser and go to **localhost:5000/** -where you should find a page with only a wind speed displayed on it, if your windspeed table is populated with recent data.

**Tip**  
  
If you want to add styles to change the appearance of your page, you will need to use a CSS stylesheet. Code for linking to one is already provided in the layout.html you made. The filename used there is style.css – use that or change it to your own.

**Further Reading / References:**

<http://silshack.github.io/fall2013/post/2013/11/06/Jonathan-Flask-Guide.html>  
<http://jinja.pocoo.org/docs/dev/templates/>  
<http://www.simplehtmlguide.com/cheatsheet.php>  
<http://www.tutorialspoint.com/css/css_quick_guide.htm>