Table of Contents

1. Installing SQLite

Objective: to collect data from a sensor and store them in a database.

SQLite is a relational database management system contained in a C programming library. In contrast to many other database management systems, SQLite is not a client-server database engine. Rather, it is embedded into the end program.

> SQLite is serverless, lightweight, opensource and supports most SQL code. SQLite stores data in a single file which can be stored anywhere.

SQLite is a popular *public domain* choice as embedded database software for local/client storage in application software such as web browsers. It is arguably the most widely deployed database engine, as it is used today by several widespread browsers, operating systems, and embedded systems (such as mobile phones), among others. SQLite has bindings to many programming languages like Python.

We will not enter into too many details here, but the full SQLite documentation can be found at this link: https://www.sqlite.org/docs.html

1.1. Installation:

Follow the below steps to create a database.

1. Install SQLite to Raspberry Pi using the command:

```
sudo apt-get install sqlite3
```

2. Create a directory to develop the project:

```
mkdir Sensors_Database
```

3. 3. Move to this directory:

```
cd Sensors_Database
```

4. Give a name and create a database like databaseName.db (in my case "sensorsData.db"):

```
sqlite3 sensorsData.db
```

A "shell" will appear, where you can enter with SQLite commands. We will return to it later.

```
sqlite>
```

Commands starts with a ".", like ".help", ".quit", etc.

5. Quit the shell to return to the Terminal:

```
sqlite> .quit
```

The above Terminal print screen shows what was explained.
The "sqlite>" above is only to ilustrated how the SQLite shell will appear. You do not need to type it. It will appear automatically.

2. Create and Populating a Table

Objective: In order to log DHT sensor measured data on the database, we must create *a table* (a database can contain several tables). Our table will be named "**DHT_data**" and will have three columns, where we will log our collected data:

- Date and Hour (column name: *timestamp*),
- Temperature (column name: *temp*), and
- Humidity (column name: *hum*).

2.1. Creating a table:

To create a table, you can do it:

- Directly on the SQLite shell, or
- Using a Python program.

2.1.a Using the SQLite Shell:

Open the database that was created in the last step:

```
sqlite3 sensorsData.db
```

And entering with SQL statements:

```
sqlite> BEGIN;
sqlite> CREATE TABLE DHT_data (timestamp DATETIME,  temp NUMERIC, hum NUMERIC);
sqlite> COMMIT;
```

NOTE: All SQL statements must end with ";". Usually, those statements are written using capital letters. It is not mandatory, but a good practice.

2.1.b Using Python:

```
import sqlite3 as lite
import sys
con = lite.connect('sensorsData.db')
with con:
   cur = con.cursor()
   cur.execute("DROP TABLE IF EXISTS DHT_data")
   cur.execute("CREATE TABLE DHT_data(timestamp DATETIME, temp NUMERIC, hum
NUMERIC)")
```

Open the above code from Moodle: **createTableDHT.py**
run it on your Terminal:

```
python3 createTableDHT.py
```

Wherever the method used, the table should be created. You can verify it on SQLite Shell using the **".table" command**. Open the database shell:

```
sqlite3> sensorsData.db
```

In the shell, once you use the ***.table* command**, the created tables names will appear (in our case will be only one: "DHT_table". Quit the shell after, using the *.quit* command.

```
sqlite> .table
DHT_data
sqlite> .quit
```

2.2. Inserting data on a table:

Let's input on our database three records of data, where each set will have three fields each: (timestamp, temp, and hum).
The *timestamp* field will be real and taken from the system, using the built-in function '**now**' and *temp* and *hum* are dummy data respectively.
Note that the time is in "UTC".
Just as in the case of creation of a table, you can insert data manually via SQLite shell or via Python.

2.2.a Using the SQLite Shell:

At the shell, you would do it, data by data using SQL statements like this (For our example, you will do it 3 times):

```
sqlite> INSERT INTO DHT_data VALUES(datetime('now'), 20.5, 30);
```

2.2.b Using Python:

And in Python, you would do the same but at once:

```
import sqlite3 as lite
import sys
con = lite.connect('sensorData.db')
with con:
    cur=con.cursor()
    cur.execute("INSERT INTO DHT_data VALUES(datetime('now'), 20.5, 30)")
    cur.execute("INSERT INTO DHT_data VALUES(datetime('now'), 28.5, 40)")
    cur.execute("INSERT INTO DHT_data VALUES(datetime('now'), 30.3, 50)")
```

Open the above code from Moodle: **insertTableDHT_1.py**

Run it on Pi Terminal:

```
python3 insertTableDHT_1.py
```

To verify that the above code worked, you can check the data in the table via shell, with the SQL statement:

```
sqlite> SELECT * FROM DHT_DATA;
```

3. Inserting and Verifying Data with Python

Objective: Inserting and retrieval using python. (i.e. retrieval implies printing the data on terminal)
Create a new python script called **insertDataTableDHT.py** as follows:

```
import sqlite3
import sys
conn =sqlite3.connect('sensorData.db')
curs=conn.cursor()

#function to insert data on a table
def add_data(temp, hum):
    curs.execute("INSERT INTO DHT_data values(datetime('now'), (?), (?))", (temp, hum))
    conn.commit()
# call the function to insert data
add_data(24.5, 30)
add_data(34.6, 34)

# print database content
print ("\n Entire database contents: \n")
for row in curs.execute("SELECT * FROM DHT_data"):
    print(row)

#close the database after use
conn.close()
```

Run the script on your Terminal:

```
python3 insertDataTableDHT.py
```

4. The DHT11 Temperature and Humidity Sensor

We have created a table in our database, where we will save all data that a sensor will read. We have also entered with some dummy data there. Now it is time to use real data to be saved in our table, air temperature and relative humidity. We will use the DHT11 sensor.

Overview

The low-cost DHT temperature & humidity sensors are very basic and slow but are great for hobbyists who want to do some basic data logging. The DHT sensors are made of two parts, a capacitive humidity sensor, and a thermistor. There is also a very basic chip inside that does some analog to digital conversion and spits out a digital signal with the temperature and humidity. The digital signal is fairly easy to be read using any microcontroller.

DHT11 vs DHT22

We have two versions of the DHT sensor, they look a bit similar and have the same pinout, but have different characteristics. The following are the specs.

DHT11 (usually blue)	**DHT22 (usually white)**
Ultra low cost	Low cost
3 to 5V power and I/O	3 to 5V power and I/O
2.5mA max current use during conversion (while requesting data)	2.5mA max current use during conversion (while requesting data)
Good for 20-80% humidity readings with 5% accuracy	Good for 0-100% humidity readings with 2-5% accuracy
Good for 0-50°C temperature readings ±2°C accuracy	Good for -40 to 125°C temperature readings ±0.5°C accuracy
No more than 1 Hz sampling rate (once every second)	No more than 0.5 Hz sampling rate (once every 2 seconds)
3 pins with 0.1" spacing	4 pins with 0.1" spacing

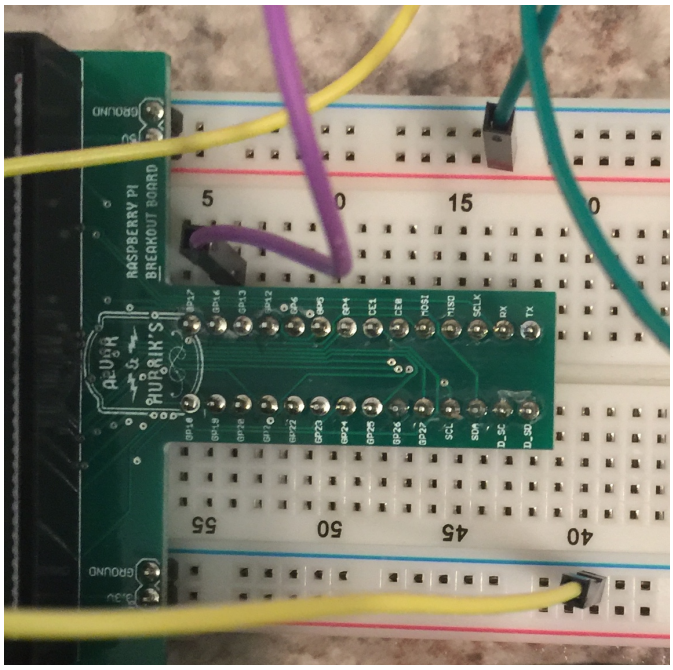
NOTE: Both use a single digital pin and are 'sluggish' in that you can't query them more than once every second (DHT11) or two (DHT22).

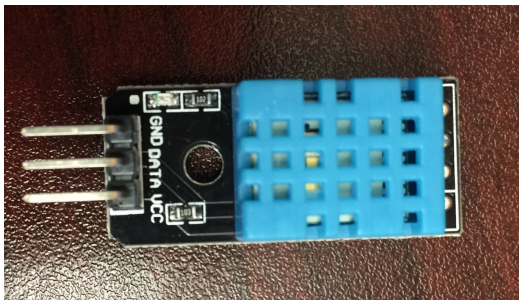
Both sensors will work fine to get Indoor information to be stored in our database.

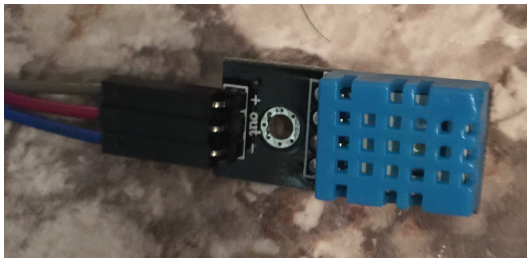
IMPORTANT:

The DHT11 has 3 pins (refer figures below):

- VCC or '+' - we connect to external 5V or to 3.3V from RPi on the breadboard;
- Data or out – we connect this to GP16 on the breadboard;
- Ground or '-' – we connect this to the ground on the breadboard.







Once the sensor is connected, we must also install its library on our RPi. We will do this in the next step.

5. Installing DHT Library

On your Raspberry, starting on /home, go to /Documents

```
cd Documents
```

Create a directory to install the library and move to there:

```
mkdir DHT11_Sensor
cd DHT11_Sensor
```

On your browser, go to the GITHub library: https://github.com/szazo/DHT11_Python

Download the library by clicking the download zip link to the right and extract the archive on your Raspberry Pi recently created folder. You will find the dht11.py – a class that can be imported into your python script to work with your sensor.

Open a test program (**DHT11_test.py**) from moodle:

```
import RPi.GPIO as GPIO
import dht11

#initialize GPIO
GPIO.setwarnings(False)
GPIO.setmode(GPIO.BCM)
GPIO.cleanup

# Readvalues on pin
instance = dht11.DHT11(pin=16)
result=instance.read()
humidity = round(result.humidity)
temperature = round(result.temperature,1)

if humidity is not None and temperature is not None:
    print('Temp={}*C Humidity={}%'.format(temperature, humidity))
else:
    print('Failed to get reading. Keep trying!')
```

Execute the program with the command:

```
python3 DHT11_test.py
```

6. Capturing Real Data

Now that we have both, the sensor and our database all installed and configured, it's time to read and save real data.

For that, we will use the code **DHT11_app.py**:

```
import RPi.GPIO as GPIO
import dht11
import time
import sqlite3
import sys

# initialize GPIO
GPIO.setwarnings(False)
GPIO.setmode(GPIO.BCM)
GPIO.cleanup

# set defaults
dbname = 'sensorData.db'
sampleFreq = 10 # time in seconds
pin=16

# fectch sensor readings
def getDHTdata():
    instance = dht11.DHT11(pin)
    result=instance.read()
    hum= result.humidity
    temp = result.temperature

    if hum is not None and temp is not None:
        hum = round(hum)
        temp = round(temp, 1)
        logData(temp, hum)

# log sensor data into database
def logData(temp, hum):
    conn =sqlite3.connect(dbname)
    curs=conn.cursor()
    curs.execute("INSERT INTO DHT_data values(datetime('now'), (?), (?))", (temp, hum))
    conn.commit()
    conn.close()

# display contents of data
def displayData():
    conn =sqlite3.connect(dbname)
    curs=conn.cursor()
    print ("\n Entire database contents: \n")
    for row in curs.execute("SELECT * FROM DHT_data"):
        print(row)
    conn.close()

# main function
def main():
    for i in range(0,3):
        getDHTdata()
        time.sleep(sampleFreq)
    displayData()

# Execute Program
main()
```

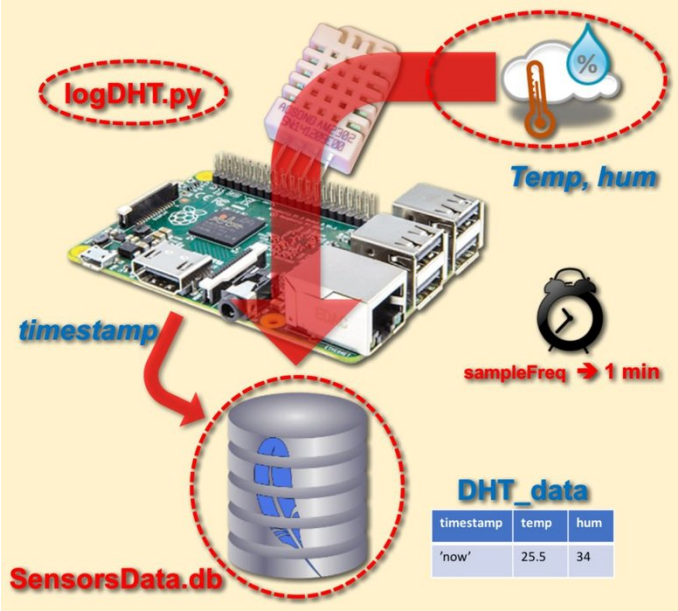
Run it on your Terminal:

```
python3 DHT11_app.py
```

The function *getDHTdata()* captures three samples of DHT sensor, test them for errors and if OK, save the data on database using the function *logData (temp, hum)*. The final part of code calls the function *displayData()* that prints the entire content of our table on Terminal.

Observe that the last three lines (records) are the real data captured with this program and the three previous rows were the ones manually entered before.

7. **Submit the following task:** ***Capturing Data Automatically***



At this point, your group must be able to implement a mechanism to read and insert data on our database automatically.

Task: Using the scripts provided as building blocks, your task is to device a system that log's sensor readings automatically for an indefinite period of time.

NOTE: The system is expected to gather sensor readings and store the readings until you kill the program with [Ctr+z].

Submit your code as a zipped file over Moodle and verify with the instructor at the end of class. One submission per group.