Embedded Systems

MQTT

1. Serialise your sensor data into a byte-encoded JSON message
    a. Convert it to Python types `int`, `float`, `string` or `Boolean`, grouped into `list` or `dict` if necessary.
    b. Package it into a single Python `dict` with suitable keys to label each field
    c. Convert it to a JSON message with function `dumps()` from python module `json`

2. Send and receive MQTT messages using the Mosquitto test broker and the HiveMQ web client
    a. Load the web client

       https://www.hivemq.com/blog/full-featured-mqtt-client-browser/

    b. Connect to the broker at test.mosquitto.org using port 8080
    c. Subscribe to the topic IC.embedded/GROUP_NAME/#
    d. Publish a test message to the topic IC.embedded/GROUP_NAME and check it shows in the Messages feed

3. Connect to the broker from your Raspberry Pi
    a. Install the paho module for python

```
raspberrypi:~$ pip3 install paho-mqtt
```

    b. Connect to the test broker using unencrypted port 1883 and publish a message

```
>>> import paho.mqtt.client as mqtt
>>> client = mqtt.Client()
>>> client.connect("test.mosquitto.org",port=1883)
```

        i. `connect()` returns 0 if the connection was successful
        ii. use `mqtt.error_string(RETURN_CODE)` to decode other error numbers

```
>>> client.publish("IC.embedded/GROUP_NAME/test","hello")
```

        iii. `publish()` returns a message info object. Use the rc attribute of the object to find the result of the publish operation

```
>>> mqtt.error_string(MSG_INFO.rc) #MSG_INFO is result of publish()
```

    c. Check you have received the message on the web client

4. Set up encryption, using the instructions at https://test.mosquitto.org/ssl/
    a. Get the broker's certificate. You can download this straight to the Rapsberry Pi with

```
Raspberrypi:~$ wget https://test.mosquitto.org/ssl/mosquitto.org.crt
```

    b. Generate a private key
    c. Get a signed certificate from the broker
        iv. Generate a certificate signing request

```
Raspberrypi:~$ openssl req -out client.csr -key client.key -
new
```

v. Paste it into the form on the broker's website and copy the certificate to the Raspberry Pi

d. Make a secure connection to the broker. Repeat the stages of part 3 but add the security information before connecting. Use broker port 8884 this time.

```
>>> client.tls_set(ca_certs="mosquitto.org.crt",
certfile="client.crt",keyfile="client.key")
```

5. Set up an MQTT client on your laptop

a. (Option 1) Install mosquitto to publish and subscribe to MQTT messages

vi. https://mosquitto.org/download/

vii. Installation complicated in Windows!

b. (Option 2) Install Paho for desktop Python

6. Subscribe to messages on the Raspberry Pi or your laptop

a. Connect to the broker using the Paho library as before

b. Define a callback function that will be executed when a message is received

```
>>> def on_message(client, userdata, message) :
>>>    print("Received message:{} on topic
{}".format(message.payload, message.topic))
```

See https://www.eclipse.org/paho/clients/python/docs/#callbacks

c. Set the callback attribute of the Client instance to point to your callback function

```
>>> client.on_message = on_message
```

Read about Python syntax for writing functions, loops etc. if you are not already familiar with it. There are no brackets to group lines of code but instead the spacing at the start of the line is critical.

Every assignment acts like a C++ reference in Python. So when you write

```
>>> client.on_message = on_message
```

calling `client.on_message()` calls the function `on_message()`.

Even if you write

```
>>> a = 4
>>> b = a
>>> a = 5
```

You will find that is b is equal to 5. If you want to make a copy instead of a reference, this must be done explicitly with `copy.copy()`

d. Subscribe to your group's topic

```
>>> client.subscribe("IC.embedded/GROUP_NAME/#")
```

e. Run the polling function to process any incoming messages. The callback function will be called for every message that's received, but only when `loop()` is called

```
>>> client.loop()
```

The `Client.loop()` function is necessary because incoming messages are asynchronous: they can happen at any time. `Client.loop()` allows you to choose when to process new messages (and other events if you have defined other callbacks).

You can run the processing loop in the background using `Client.loop_start()` and `Client.loop_stop()`. This will run your callback as soon as the message arrives. However, since this could happen during any other part of your code, you should be careful with updating any variables that are in use.