Embedded Systems
ESP8266, micropython and I²C

1. Collect a kit
2. Research the sensor
    a. What does it do?
    b. What is the power supply voltage? Some of the modules have power supplies in addition to the sensor chip so look for the documentation for the module.
    c. What is the control flow for the sensor?
        i. Does it need enabling or configuring before use?
        ii. Does it measure automatically or on demand?
        iii. How do you configure it (if applicable)?
        iv. How do you request a measurement (if applicable)?
        v. How do you read back the result?
        vi. What conversion is needed to convert the result into something meaningful?
3. Set up Python development flow with ESP8266
    a. Install Python on your laptop
        i. Latest version is 3.6.4. Many people use version 2.7.14 for backward compatibility. Only do this if you must use legacy code.
    b. Connect your ESP8266 to a USB port and find the port name
        i. In Windows:
            1. Open a command prompt and type

```
wmic path Win32_SerialPort > dump.txt
```

            2. Open dump.txt in a text editor with word wrap turned off. Look for an entry like 'Silicon Labs CP210x USB to UART Bridge (COMX)', where COMX is the serial port for the ESP8266
        ii. In MAC or Linux:
            1. Open a command prompt and type

```
ls /dev/{tty,cu}.*
```

            2. Try the command with and without the cable connected and look for a line which only appears when it is connected. That is the name of your serial port.
        iii. If the system does not recognise the USB device, install the serial port driver from here: https://www.silabs.com/products/development-tools/software/usb-to-uart-bridge-vcp-drivers
    c. Establish communication with the ESP8266
        i. On Windows, install a terminal client like PuTTY
            1. Check the box 'Serial'
            2. Enter the serial port name under 'Serial Line'
            3. Set the speed to 115200
        ii. On MAC or Linux, you can use the built in Screen from the command line (with /dev/ttyS0 as the serial port):

```
screen /dev/ttyS0 115200
```

    d. Get a response from the module

    i. Reset the ESP8266 using the push button on the module. You should see a boot message (some of it garbled) and a Python prompt
    ii. Type `print("Hello")` and check that the command is successfully interpreted
4. Establish communication with your sensor
    a. Wire up the sensor to the ESP8266 module. I recommend using pins 4 and 5 for the I^2C port since these are labelled SCL and SDA. However, other pins can be used since the port is implemented entirely with a software library. Avoid pins 0,2,15 and 16 since these have LEDs or other circuitry connected to them.
    b. Connect the power supply appropriately and other connections that may be needed by your sensor
    c. Set up the I2C port by typing the commands given in the lecture
    d. Use I2C read and write functions as necessary to drive your sensor
    e. Once you have got the sensor working, put the commands into a text file to begin constructing a communication library.
        i. Copy the file to the ESP8266 using the ampy tool (see lecture)
        ii. Run it by typing `import filename`
            1. Leave out the .py extension from the filename
        iii. Statements in the top level of the file are executed straight away
        iv. Function definitions in the file are imported and can be executed by typing `filename.function()`
5. Process the sensor output to make a meaningful measurement
    a. I2C read and write functions receive and transmit data in `bytearray` data types, which are a list (vector) of bytes.
    b. Access individual bytes from the array by subscripting
        i. E.g. `data[0]`
    c. Convert multiple bytes to an integer using `int.from_bytes()` specifying whether the data is big or little endian
        i. E.g. `rawtemp = int.from_bytes(data,'big')`
    d. Convert to a floating point type subsequently if necessary
        i. E.g. `temp = float(rawtemp)/100.0`
    e. Unlike C or C++, variables in Python are not initialised with their data type. Instead, they adopt the type of whatever is assigned to them.
        i. E.g. `tempconst = 100` creates or reassigns a variable tempconst of type `int`
        ii. E.g. `tempconst = 100.0` creates or reassigns a variable tempconst of type `float`