DTiF
Digital Technologies in focus
Initiative of and funded by the Australian Government Department of Education, Skills and Employment
acara AUSTRALIAN CURRICULUM, ASSESSMENT AND REPORTING AUTHORITY

Years 9–10

Investigating environmental data with microcontrollers

Activity guide

Assessment focus: Australian Curriculum: Digital Technologies (digital systems and data)

Contents

Activity 0 – A broad overview of the hardware and software

Arduino in schools

Traditionally, Arduino activities begin with making an LED blink, but the measurement of environmental variables is comparatively easy and provides a lot more inherent interest to students.

The structure of an Arduino sketch, and why it's called a sketch, is available at: https://processing.org/overview/

The Arduino environment is now augmented by newer microcontrollers such as the ESP family

Expressif datasheet:

https://www.espressif.com/sites/default/files/documentation/esp32-wroom-32_datasheet_en.pdf

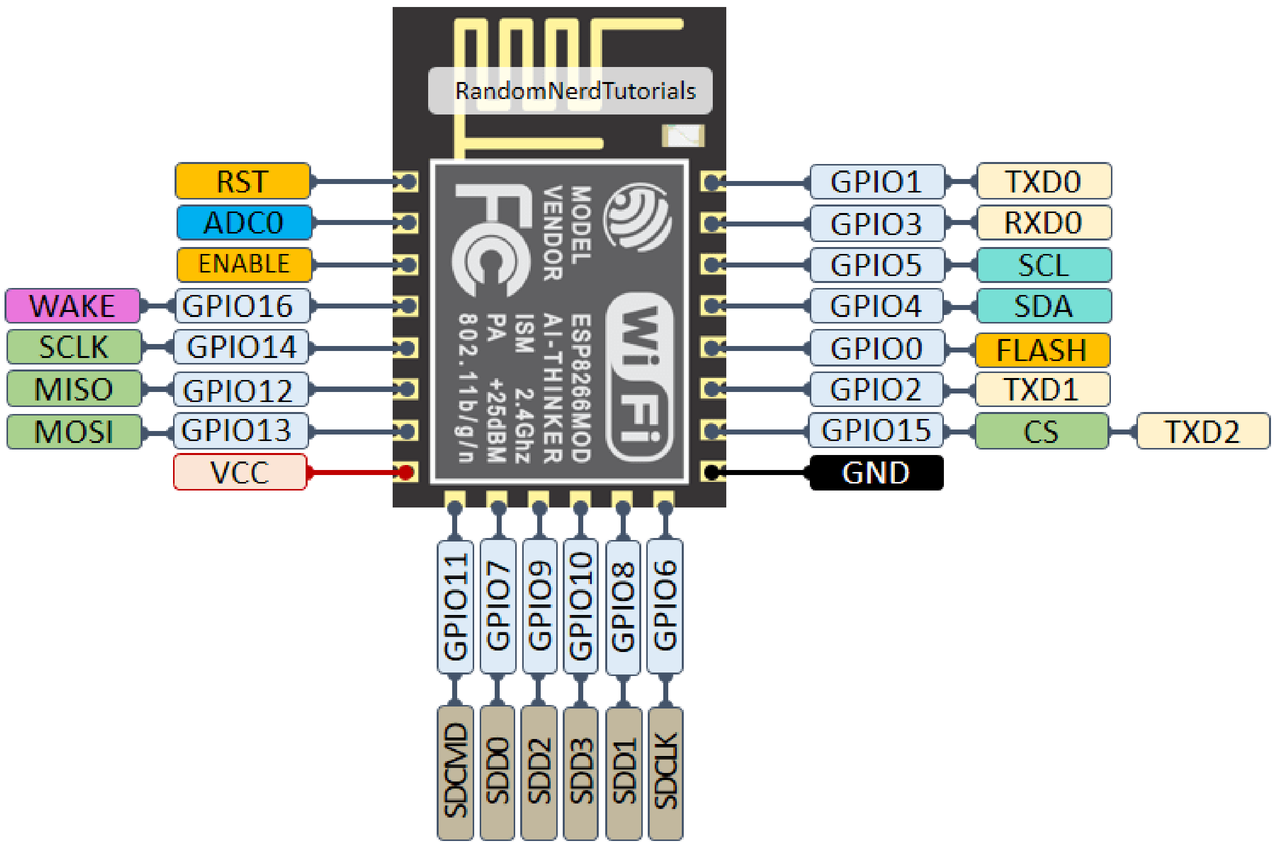
A description of the boards:

https://arduino-esp8266.readthedocs.io/en/latest/esp8266wifi/readme.html

Specification sheet for the Jaycar offering:

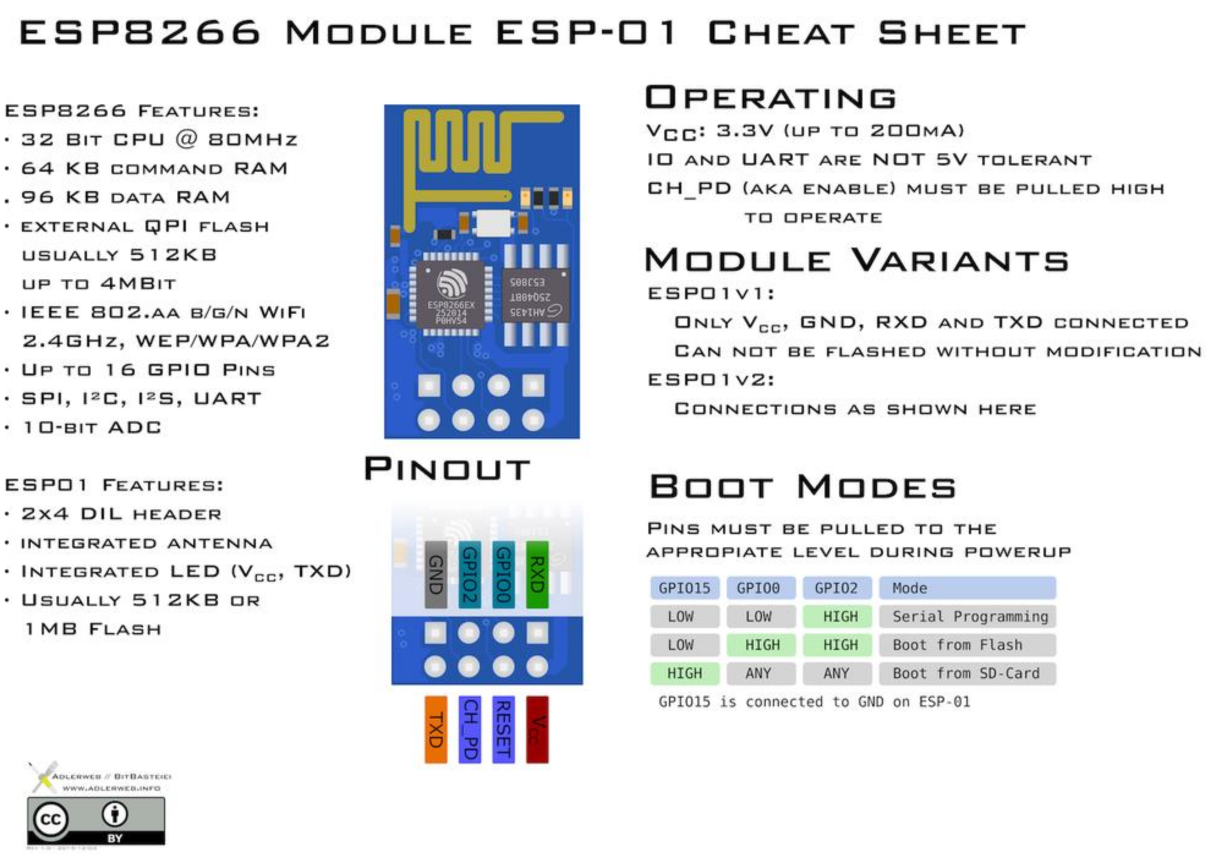
https://www.jaycar.com.au/medias/sys_master/images/images/9486646607902/XC3802-manualMain.pdf

Pinouts for ESP8266 family:



Pinouts for the ESP8266
Retrieved from: https://randomnerdtutorials.com/esp8266-pinout-reference-gpios/

Pinouts for the ESP-01S:



ESP8266 Module ESP-01 Cheat Sheet

ESP8266 Features:
- 32 Bit CPU @ 80MHz
- 64 KB command RAM
- 96 KB data RAM
- external QPI flash usually 512KB up to 4MBit
- IEEE 802.AA b/g/n WiFi 2.4GHz, WEP/WPA/WPA2
- Up to 16 GPIO Pins
- SPI, I²C, I²S, UART
- 10-bit ADC

ESP01 Features:
- 2x4 DIL header
- integrated antenna
- Integrated LED (V_{CC}, TXD)
- Usually 512KB or 1MB Flash

Operating

V_{CC}: 3.3V (up to 200mA)

IO and UART are NOT 5V tolerant

CH_PD (aka enable) must be pulled high to operate

Module Variants

ESP01v1:

Only V_{CC}, GND, RXD and TXD connected

Can not be flashed without modification

ESP01v2:

Connections as shown here

Boot Modes

Pins must be pulled to the appropriate level during powerup

GPIO15	GPIO0	GPIO2	Mode
LOW	LOW	HIGH	Serial Programming
LOW	HIGH	HIGH	Boot from Flash
HIGH	ANY	ANY	Boot from SD-Card

GPIO15 is connected to GND on ESP-01

Pinouts for the ESP-01S (a cut down version of the bigger ESP8266)
Retrieved from: https://www.instructables.com/Definitive-Guide-to-Setting-Up-Your-New-ESP01-Modu/

You may find reference to an ‘AT command set’ to program any of the ESP family. This is a hangover from early communications. ‘AT’ means ‘ATtention’, used to prepare a modem to receive instructions, and can be ignored as it is not used in this series of activities.

You can purchase the ESP8266 from Jaycar Electronics: https://www.jaycar.com.au/wifi-mini-esp8266-main-board/p/XC3802 and there are several other electronics suppliers (including eBay). It’s wise to contact the supplier and outline what you are planning to do with any items to make sure they will behave as expected. eBay especially offers very little information to help in determining the behaviour of components.

Add the ESP family to your boards in Arduino as:
http://arduino.esp8266.com/stable/package_esp8266com_index.json

The main focus of the ESP8266 is the wi-fi compatibility in a small form factor. It also has an 80 MHz microcontroller and 4 MB of on-board flash memory.

It can also be purchased in the cut down version (ESP-01S) or a next level version incorporating a more powerful, dual-core CPU, more input/output pins and Bluetooth (ESP32).

Serial drivers

Drivers are software that tell the computer how to communicate with peripherals – in this case how to do so serially over a universal serial bus (USB) connection.

Linux and Mac computers will have the drivers installed already, as part of the operating system.

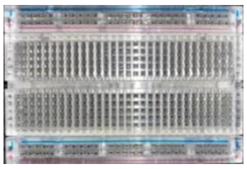
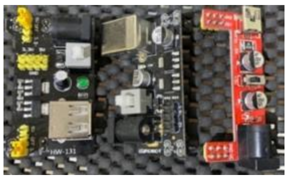
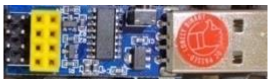
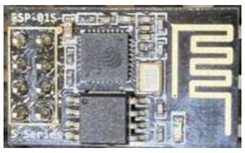
With Windows, you may need to manually install a driver, which will need administrator privileges and may involve the IT administrator of your system. See instructions: https://windowsreport.com/arduino-problem-windows-10

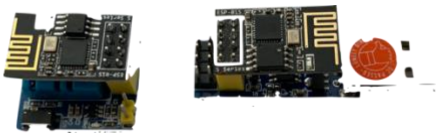
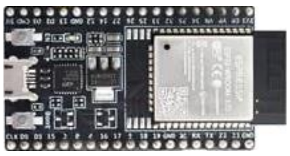
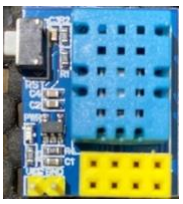
If your board has a USB port, it should now work right out of the box. For the ESP-01S and other boards without a USB connector, you'll need to purchase a FTDI USB serial converter.

Equipment: likely costs

All prices are in Australian dollars (A$).

Deals can be found, but it's a really good idea to buy one item first to test it before using with students.

Item	Approximate cost in A$
 Breadboard	$4–5
Dupont wires	$5–10 for a bunch Buy a lot, especially black and red for colour coding positive and negative (ground)
 Breadboard power supplies	$5–9 There are 3 pictured here, supplying power from 9 V batteries or wall warts, or USB A, B and micro. Most times you won't need these as the USB socket on the microcontrollers will supply power.
 USB to serial programmer	ESP programmer IOTMCU USB to serial converter, $2–7 Note: The IO is eye-oh, not the numerals 1 and zero. Needed for Activity 9
 ESP-01S	ESP-01S, $3–7 Needed for Activity 9

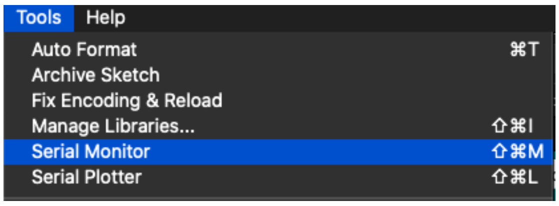
 Combo DHT11 and ESP-01 shown with ESP on FTDI programmer	I bought a kit containing 3 x ESP01, together with 3 x DHT11 with appropriate pins so that the 2 fitted together easily without a breadboard, and one FTDI USB programmer for less than $30 online. Ten dollars per student will give a personalised experience that can be programmed at school and then taken home.
ESP8266	$4–25 (your mileage may vary here). I paid $25 at a local electrical supply house for this board, but you can purchase online for much, much less. Just make sure that there is adequate documentation, or buy one, test and make a decision based on its operability. Look for Wemos, mini or D1.
 ESP32	$10–15
Light dependent resistor (LDR)	$1
 DHT11 temperature and humidity sensor	$2 – buy bundled with ESP-01S and a serial programmer/converter
Metriful environment sensors board	$70 from metriful.com

2.5 µm particle sensor	$70 from metriful.com

Activity 1 – Simply serial: how can I see what the output is for a given input?

What's happening here

Serial means one after another. The board can send data, one byte at a time, to the computer and show the output on a Serial Monitor, a software tool that shows (or monitors) what's happening on the serial port, not a separate monitor screen:



Find the Serial Monitor under Tools in the Arduino integrated development environment (IDE)

The Serial Monitor will show the output from the serial lines in a separate window.

Decomposition

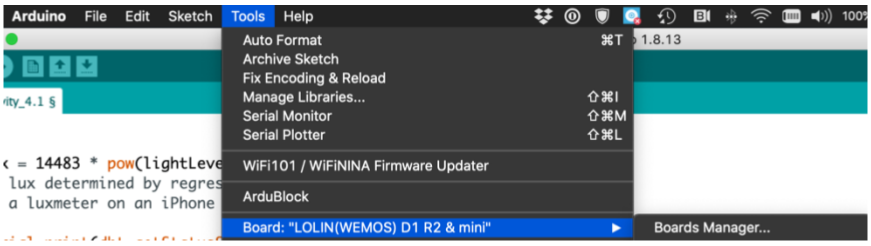
For this activity, we can decompose the problem to:

```
Configure the Arduino IDE to use an ESP8266 microcontroller

Connect the Arduino IDE to the microcontroller over USB

Check connectivity

Serially print data to check that the system is working correctly
```

Installing the ESP8266 board

Open Boards Manager from Tools > Board menu and install *esp8266* platform

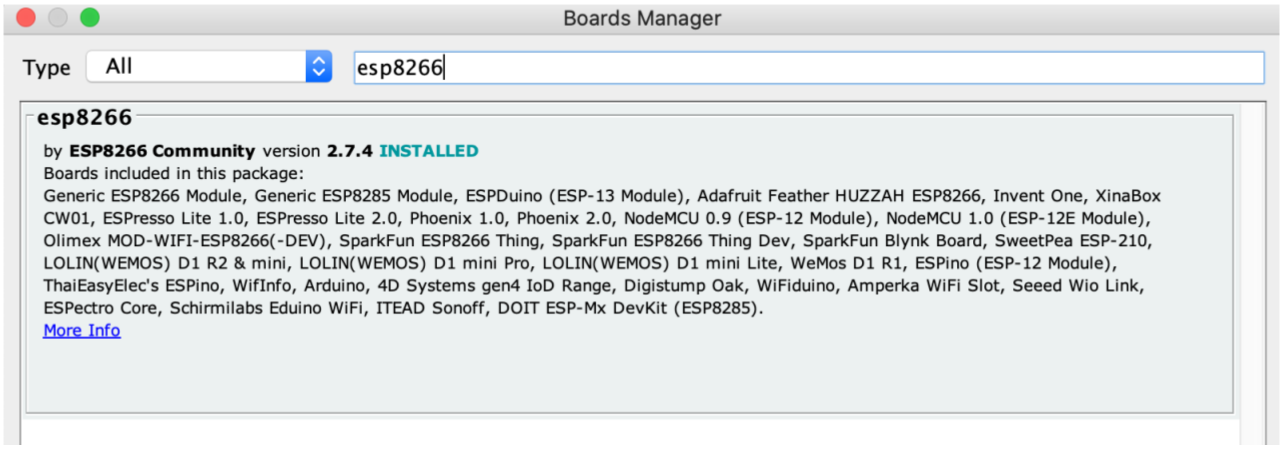
Don't forget to select your ESP8266 board from Tools > Board menu after installation.



```
< = 14483 * pow(lightLeve
 lux determined by regres
 a luxmeter on an iPhone
```

Arduino boards management

In the search box, search for and then add esp8266.



esp8266

by **ESP8266 Community** version **2.7.4** **INSTALLED**
Boards included in this package:
Generic ESP8266 Module, Generic ESP8285 Module, ESPDuino (ESP-13 Module), Adafruit Feather HUZZAH ESP8266, Invent One, XinaBox CW01, ESPresso Lite 1.0, ESPresso Lite 2.0, Phoenix 1.0, Phoenix 2.0, NodeMCU 0.9 (ESP-12 Module), NodeMCU 1.0 (ESP-12E Module), Olimex MOD-WIFI-ESP8266(-DEV), SparkFun ESP8266 Thing, SparkFun ESP8266 Thing Dev, SparkFun Blynk Board, SweetPea ESP-210, LOLIN(WEMOS) D1 R2 & mini, LOLIN(WEMOS) D1 mini Pro, LOLIN(WEMOS) D1 mini Lite, WeMos D1 R1, ESPino (ESP-12 Module), ThaiEasyElec's ESPino, WifInfo, Arduino, 4D Systems gen4 IoD Range, Digistump Oak, WiFiduino, Amperka WiFi Slot, Seeed Wio Link, ESPectro Core, Schirmilabs Eduino WiFi, ITEAD Sonoff, DOIT ESP-Mx DevKit (ESP8285).
More Info

The Arduino boards manager

The Arduino IDE will now output to a serial port on your computer. This will usually be via the universal serial bus (USB). (Make sure your USB lead is not a cheap power-only lead but one that will handle data as well).

Let’s get Serial

The Arduino language, an object-oriented programming language, has 3 handy functions: Serial.begin, Serial.print and Serial.println, all 3 being serial port *objects.*

We’ll use Serial.begin and Serial.println here. Serial.print will print the output on one line (like typing words without pressing ‘enter’ after each one on a keyboard).

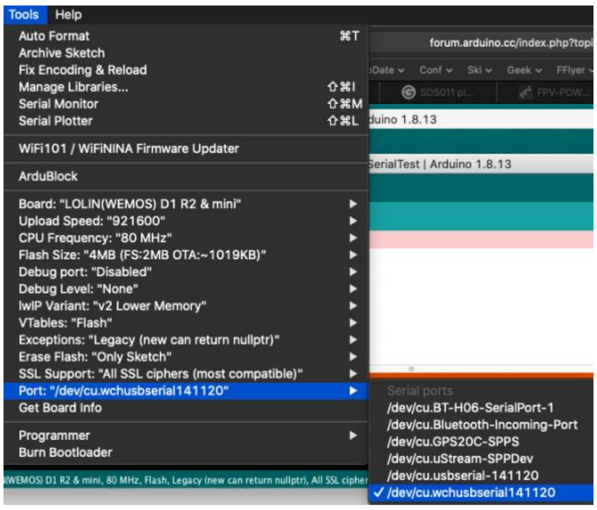
A sample program that tests your connections is below:

```
//comments have two slashes in front: the compiler knows to ignore
// anything after on the same line
void setup () // do this when the board is powered up
{
  Serial.begin(9600); //set the serial speed to 9600
}
void loop()
{
while (true) // basically forever
  {
  Serial.println("test"); //print the word "test" on a separate line
  delay(1000); //wait for a one second (1000 milliseconds)
  }
}
```

Serial killers

If you see gibberish when you look at your Serial Monitor, check that the speed you have specified matches the speed shown in the monitor.

Check also that the port is correctly chosen:



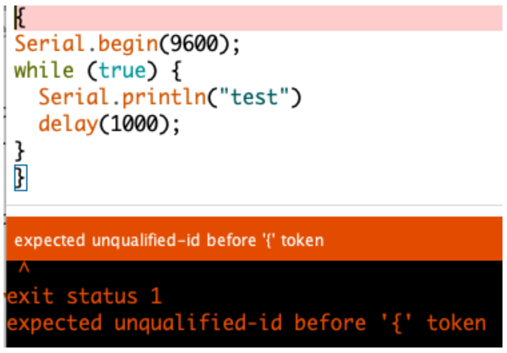
```
Tools  Help
Auto Format                                          ⌘T
Archive Sketch
Fix Encoding & Reload
Manage Libraries...                                 ⇧⌘I
Serial Monitor                                      ⇧⌘M
Serial Plotter                                      ⇧⌘L
WiFi101 / WiFiNINA Firmware Updater
ArduBlock
Board: "LOLIN(WEMOS) D1 R2 & mini"
Upload Speed: "921600"
CPU Frequency: "80 MHz"
Flash Size: "4MB (FS:2MB OTA:~1019KB)"
Debug port: "Disabled"
Debug Level: "None"
lwIP Variant: "v2 Lower Memory"
VTables: "Flash"
Exceptions: "Legacy (new can return nullptr)"
Erase Flash: "Only Sketch"
SSL Support: "All SSL ciphers (most compatible)"
Port: "/dev/cu.wchusbserial141120"
Get Board Info
Programmer
Burn Bootloader

Serial ports
/dev/cu.BT-H06-SerialPort-1
/dev/cu.Bluetooth-Incoming-Port
/dev/cu.GPS20C-SPPS
/dev/cu.uStream-SPPDev
/dev/cu.usbserial-141120
✓ /dev/cu.wchusbserial141120
```

Setting the port for communications with the microntroller

Code killers

The semicolon (;) after statements is essential. It’s one of the ways that the language tells whether you are starting a new instruction or continuing an existing one.

Sometimes, the Arduino environment gets confused and you need to look back a step to find the actual error. Note the missing semicolon below is actually missing from the line beforehand:



```
{
Serial.begin(9600);
while (true) {
  Serial.println("test")
  delay(1000);
}
}

expected unqualified-id before '{' token
^
exit status 1
expected unqualified-id before '{' token
```

A typical error message from Arduino

The error doesn’t say ‘you’re missing a semicolon’ – but this, and capitalisation, are the first things you should look for if you get an error. (For more information see Appendix 7 – Arduino tips.)

Why use the Serial Monitor?

- It's useful as a debugging tool. If a program doesn't work, you can comment lines out that you consider suspect and check the value of variables over serial.
- It's also useful as a debugging tool because you can output data to the Serial Monitor to see if what you thought was happening with data was actually happening.
- You can copy the serial data to your computer, and paste it into a spreadsheet for analysis.

Debugging example

If you get a serial output that looks like this:

```
16:14:49.476 -> TIMEOUT     nan     nan

16:14:52.500 -> TIMEOUT     nan     nan
```

then whatever you thought you were measuring is not actually being measured, or is in the wrong format, hence the monitor shows 'nan' which is 'not a number' in Arduino-speak. Here, you are also getting a timeout, so the microcontroller itself is not responding. You need to check that the right board is selected and that the right serial connection is being used.

Activity 2 – Use a light dependent resistor to measure light levels

Why?

Lighting determines our efficiency in carrying out a task, whether that's reading, writing, relaxing, doing close work like technical drawing, escaping a building in an emergency, or assembling a product.

This website describes the Australian standards for light levels in different workplace settings: https://www.ohsa.com.au/services/lighting-survey

What is an LDR?

Light falling on some substances changes their electrical conductivity. A light dependent resistor (LDR) is a sensor that changes its resistance when light is shone on its surface. How much it changes depends on the brightness of the light.

We'll use this sort of sensor here to see how bright our classroom is and (in Activity 3) determine if the light level is adequate for the task.

Decomposition

For this activity, we can decompose the problem to:

```
Connect the ESP microcontroller to a light dependent resistor using a breadboard

Use the Arduino IDE to connect to the ESP microcontroller using a serial USB
cable

Serially print the sensor value so we can double check what's happening
```

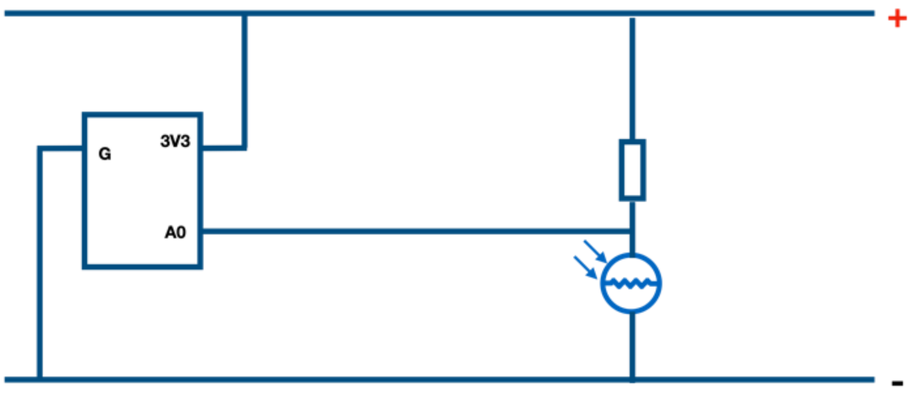
System set-up

If you're not familiar with the use of a breadboard, review Appendix 5: Background information on breadboards.

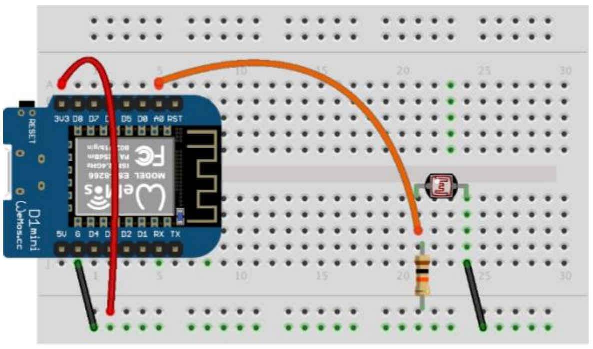
You will need:

- a breadboard
- an LDR
- a 10 kΩ resistor
- some Dupont connector wires.

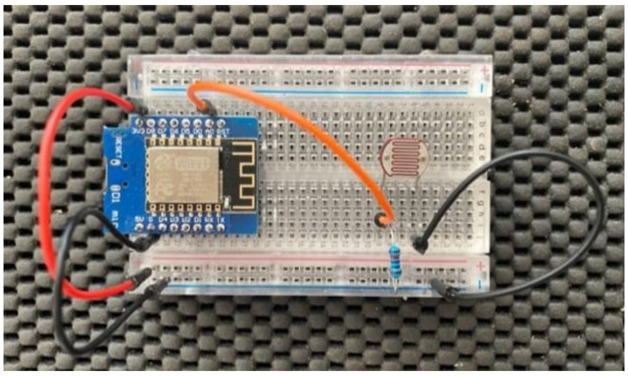
Note that the LDR is connected to the A0 pin on the esp8266. This is the analog pin as the LDR is presenting a voltage that is analog in nature.



A circuit diagram is useful to trace connections



This schematic diagram is useful to check wiring



This photo is useful to check your layout

How does this circuit work?

A resistor presented with 3 volts will pretty much always show 3 V. If presented with a 5 V supply, it will show 5 V. So, we can't just have the LDR on its own; we need to compare its value to a known, fixed resistor. This is called a resistive divider. We can then measure a change in voltage across the LDR as its resistance changes and use this as a measure of light level.

The sketch

```
/* A simple sketch for an ESP8266 to read the level of light falling on a Light
Dependent Resistor

   and report it via serial communications

   December 2020 CCBY 4 Levins

*/


int lightLevel = 0;


void setup() // The void keyword indicates that what follows is a function that
doesn't return any values.

// it would normally contain anything we want to happen only when the system is
first powered up


{

  Serial.begin(9600); //initialise the serial output

}


void loop() // this is the main working part of the program

{

  lightLevel = analogRead(A0); // get the analog value from the LDR circuit


  Serial.print("Light level: "); // print without a new line

  Serial.println(lightLevel); // print the value then start a new line

  delay(1000); // wait for a second before reporting a new light level

}
```

What am I seeing in the Serial Monitor?

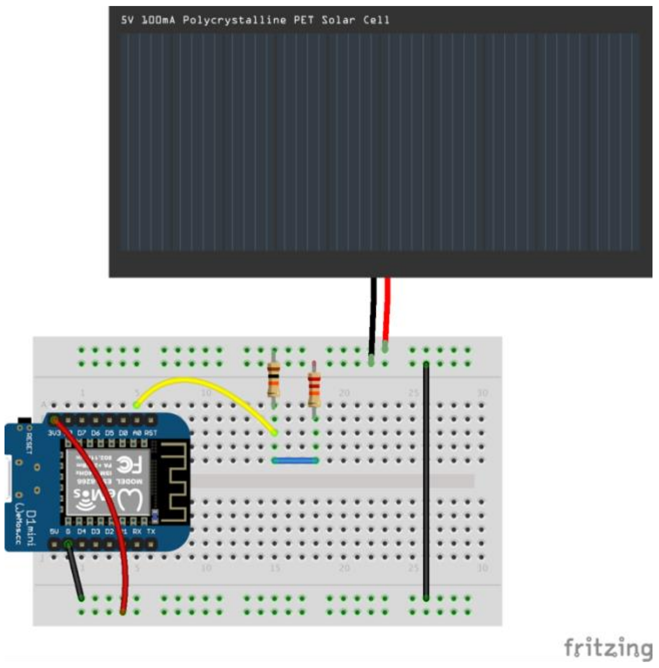
The number returned is just a number – it has no units and therefore is fairly useless to check if we have enough light to work. Notice also that the number gets bigger as the light gets dimmer, and smaller if the light gets brighter.

We need to convert this number to lux, because that's what is used to reference adequate light levels for safe and effective work areas. We'll look at this in the next activity.

Extension

Research shows that light that is not constant in intensity (a flickering light) can be detrimental to learning. See: https://theconversation.com/fluorescent-lighting-in-school-could-be-harming-your-childs-health-and-ability-to-read-124330

So we could use a small hobby solar panel to measure light levels to see if the light in the classroom is constant. Use a circuit like this:



Sample layout for measuring the voltage output of a hobby solar cell
(maximum output voltage 3 V)

Set the time for sampling (delay) to 100. Is the light level consistent? Is it different under fluorescent and LED lamps?

Activity 3 – Calibrating the LDR output using regression

What is regression?

Regression is the process of taking data and trying to fit a mathematical relationship between it. It's a basic form of artificial intelligence and it used to be hard, but modern spreadsheets make it easy.

In the Australian Curriculum: Mathematics, content description ACMSP251 asks that students 'Use scatter plots to investigate and comment on relationships between two numerical variables' and suggests 'using authentic data to construct scatter plots, make comparisons and draw conclusions'.

We'll use this here to build a scatter plot, then determine the relationship between LDR output and lux. We'll measure lux from a known source (a phone app that gives its output as lux) and compare it with the number that the LDR gives under the same conditions.

Regression will give us a formula that we can plug into our code so that the readout is calibrated to lux.

This activity is a great opportunity for formative assessment:

Modify the sketch from the previous activity to convert lightLevel to lux.

Decomposition

For this activity, we can decompose the problem to:

```
Collect concurrent readings from the light dependent resistor and from a light
meter which will give readings in lux

Find a relationship between the LDR reading and the indicated lux level

Use the relationship to convert light level to lux within the sketch

Output the light readings as lux

(optionally) use an IF structure to report light levels as text
```

Collecting data

If we can find an app that converts a phone to a light meter giving readings in lux, we can generate a scatter plot of lux v the LDR reading on the serial port, then use a spreadsheet to see what relationship exists.

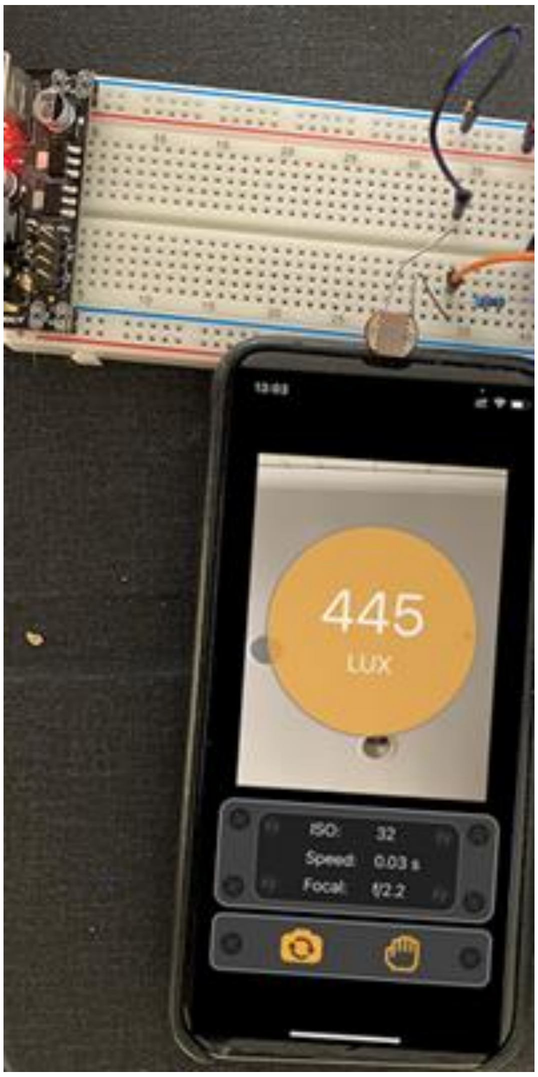
See https://www.photoworkout.com/best-light-meter-apps/ for (free) iOS and Android recommendations.

Modify the program from Activity 2 so that it sends the LDR value to the Serial Monitor once every 5 seconds:

- place a light source so that it evenly illuminates the LDR and the phone's camera
- move the light source away, in increments, to get a series of light readings from the low hundreds to just over 1,000. (One thousand is the recommended lux level for fine, detailed work, and 500 is a minimum for reading.)
- enter the lux readings and the LDR readings into 2 separate columns in a spreadsheet

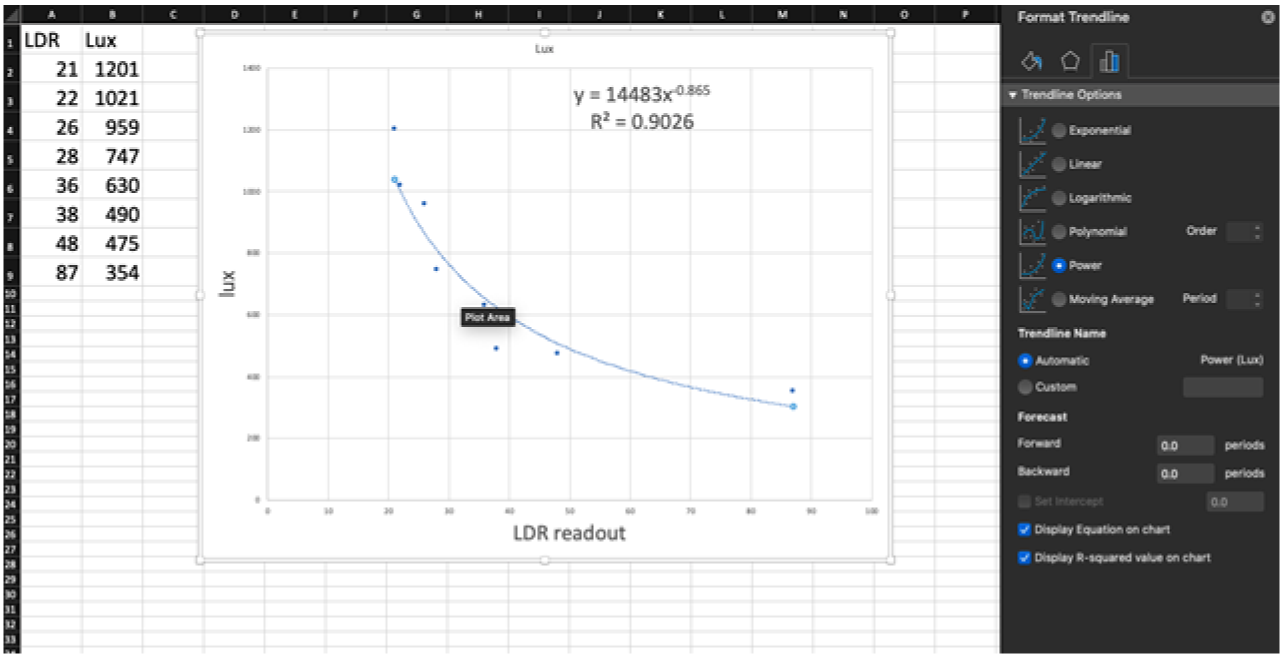
- click on any data point to select
    - In Excel, Select the chart, choose Chart Design > Add Chart Element > Trendline> more options
    - In Google Sheets, double-click the chart, double-click a chart, click Customize > Series, Click Trendline
    - In Apple Numbers, click the graph, then in the Format sidebar, click the Series tab. Click the disclosure arrow next to Trendlines, then click the pop-up menu and choose a type of **trendline**
- try a visual fit: a power relationship looks best with my data
- research shows that our classrooms should be between 500 and 1,000 lux for good reading conditions so we can ignore any entries below 100 and above 1,000, so delete all other entries except the target amount (between 100 and 1,000)
- note that the chart adapts
- display equation on chart (this may not be dynamic – if you change the chart you may need to choose to display the equation again). Note down or copy the equation.

System set-up



Lining up the LDR and the camera on the phone

What your spreadsheet should look like



LDR	Lux
21	1201
22	1021
26	959
28	747
36	630
38	490
48	475
87	354

$y = 14483x^{-0.865}$
$R^2 = 0.9026$

Format Trendline

Trendline Options
- Exponential
- Linear
- Logarithmic
- Polynomial — Order
- [✓ Power]
- Moving Average — Period

Trendline Name
- [✓ Automatic] Power (Lux)
- Custom

Forecast
- Forward 0.0 periods
- Backward 0.0 periods
- Set Intercept 0.0
- [✓ Display Equation on chart]
- [✓ Display R-squared value on chart]

Spreadsheet and scatter chart in Microsoft Excel

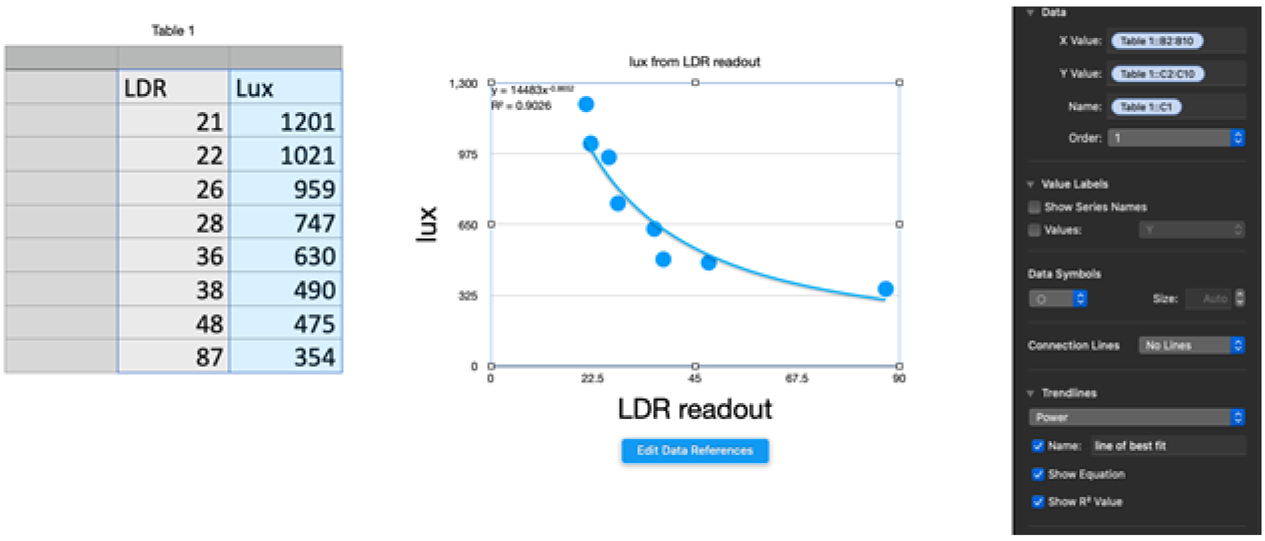


Table 1

LDR	Lux
21	1201
22	1021
26	959
28	747
36	630
38	490
48	475
87	354

$y = 14483x^{-0.865}$
$R^2 = 0.9026$

Data
- X Value: Table 1::B2:B10
- Y Value: Table 1::C2:C10
- Name: Table 1::C1
- Order: 1

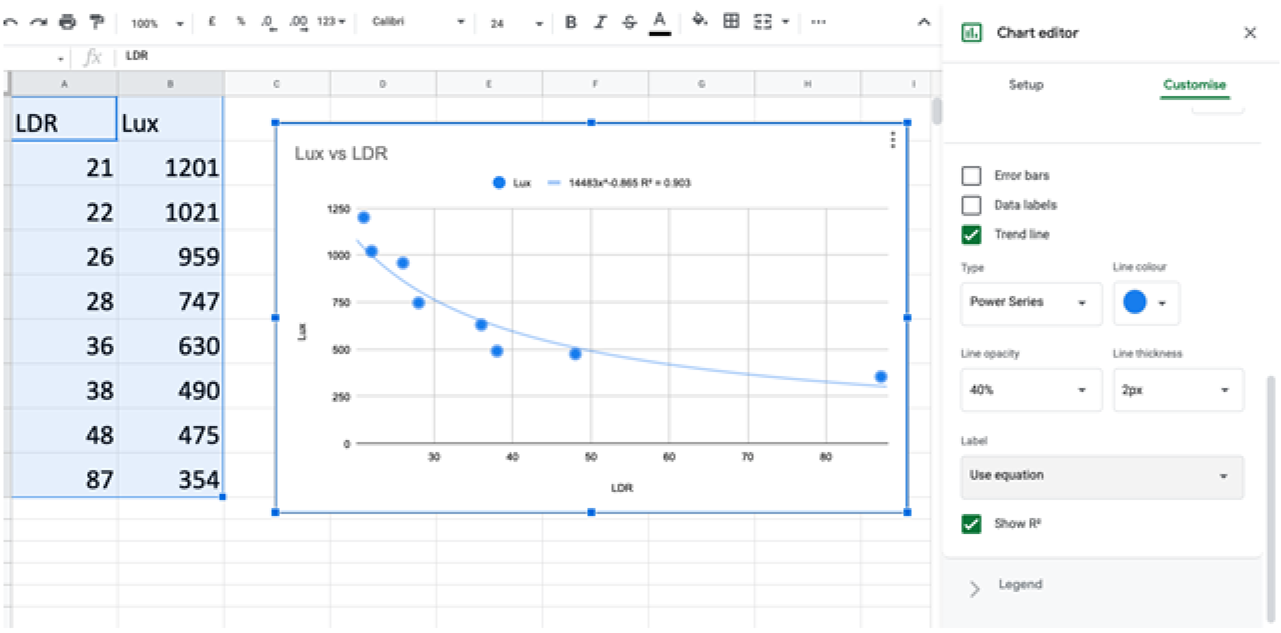
Value Labels
- Show Series Names
- Values: Y

Data Symbols — Size: Auto

Connection Lines: No Lines

Trendlines: Power
- [✓ Name:] line of best fit
- [✓ Show Equation]
- [✓ Show R² Value]

Spreadsheet and scatter chart in Apple Numbers



LDR	Lux
21	1201
22	1021
26	959
28	747
36	630
38	490
48	475
87	354

Chart editor

Setup | Customise

- [ ] Error bars
- [ ] Data labels
- [✓] Trend line

Type: Power Series

Line colour: ●

Line opacity: 40%

Line thickness: 2px

Label: Use equation

- [✓] Show R²

Legend

Spreadsheet and scatter chart in Google Sheets

The sketch

Note the equation that each gives is y = 14483x$^{-.865}$ or lux = 14483lightLevel$^{-.865}$ for my situation – yours may differ.

Students need to create a new variable called ‘lux’ and a formula to translate lightLevel into lux.

Note the commented out section in the sketch below:

```
/* A simple sketch for an ESP8266 to read the level of light falling on a Light
Dependent Resistor, convert to lux and report it via serial

December 2020 CCBY 4 Levins*/


int lightLevel = 0;

int lux = 0;


void setup() // The void keyword indicates that what follows is a function that
doesn’t return any values.

// it would normally contain anything we want to happen only when the system is
first powered up


{

  Serial.begin(9600); //initialise the serial output

}


void loop() // this is the main working part of the program

{

  lightLevel = analogRead(A0); // get the analog value from the LDR circuit
// lux = 14483 multiplied by (lightLevel raised to the power of -.865)
// search the web to find the appropriate formula elements for the formula above

  Serial.print(“Light level: “); // print without a new line
  Serial.println(lightLevel); // print the value then start a new line

  Serial.print(“lux: “); // print without a new line
  Serial.println(lux); // print the value then start a new line

  delay(1000); // wait for a second before reporting a new light level
}
```

Activity 4 – DHT11 measures temperature and humidity

What is a DHT11?

A DHT11 is a low-cost single package of sensors measuring temperature and humidity.

It uses a special communications protocol to send 2 messages on one wire. Read more: https://www.electronicwings.com/sensors-modules/dht11

It has 3 usable pins: positive, data and ground, but sometimes comes packaged as a 4-pin device with one pin not connected to anything. Make sure you check the documentation that the vendor provides or that the pins are clearly marked.

The DHT family also includes a more expensive, higher precision device referred to as a DHT22.

The use of Libraries to give extra functionality to a program

Imagine if you had to write down the instructions for making a drink every time you went to a shop.

You can just say, 'I'll have a cappuccino please' or 'a vanilla milkshake please' and your order will be made. This is an example of abstraction, where the fine detail is left out and only the one instruction given.

Computers do this too.

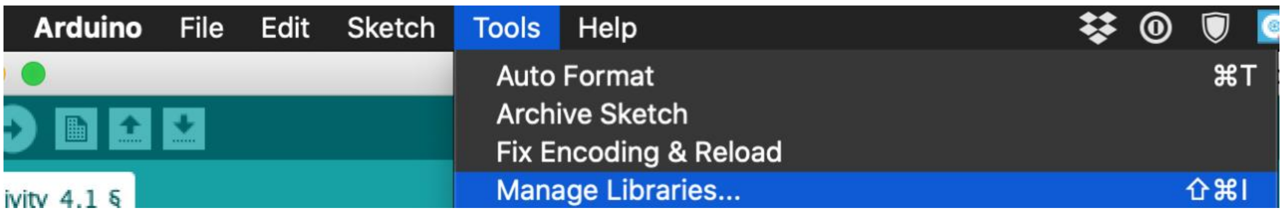
Communicating with a device such as a DHT11 can be very complicated, but it's a very common device, so why not write it once and share the instructions? Programming languages refer to this packaged set of instructions as a library. A library will contain all the objects needed to communicate with a device.

Libraries are shared on a repository (such as GitHub).

The library used here is specifically designed for DHT sensor packages and can be found on GitHub at: https://github.com/beegee-tokyo/DHTesp

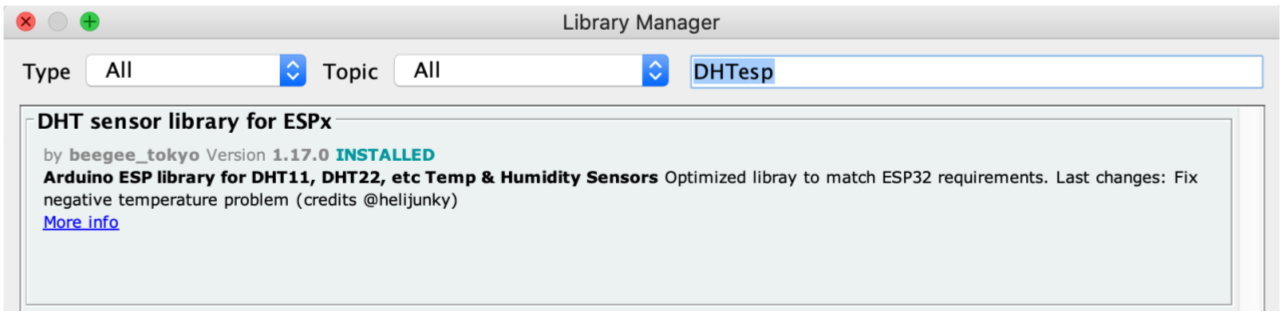
Read the README.md file (or just scroll down) to see comments from the author and the objects that are included in the library. (The library file on GitHub is discussed in more detail in Activity 5.)

To load the DHTesp library into your Arduino environment, go to Tools, Manage Libraries.



Tools…Manage Libraries in Arduino

Search for and install DHTesp.h.


Library Manager

Type: All | Topic: All | DHTesp

DHT sensor library for ESPx

by **beegee_tokyo** Version **1.17.0** **INSTALLED**
Arduino ESP library for DHT11, DHT22, etc Temp & Humidity Sensors Optimized libray to match ESP32 requirements. Last changes: Fix negative temperature problem (credits @helijunky)
More info

The DHT library will show 'INSTALLED' when done

You're all set. Now when you need to use the functionality provided by this library, you can type this at the top of your sketch:

```
#include "DHTesp.h"
```

and your sketch will know how to find the functionality it needs.

Decomposition

For this activity, we can decompose the problem to:

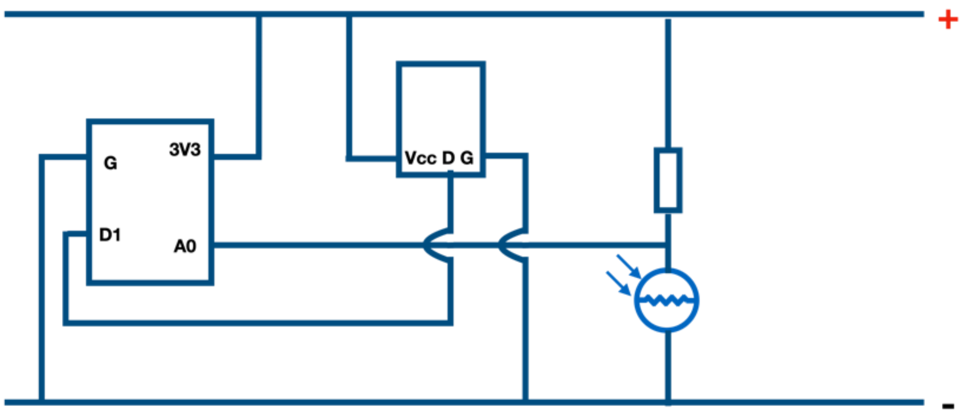
```
Add the DHT11 to the breadboard and then to the ESP microcontroller

Use the Arduino IDE to connect to the ESP microcontroller using a serial USB
cable

Serially print the sensor values so we can double check what's happening
```

System set-up

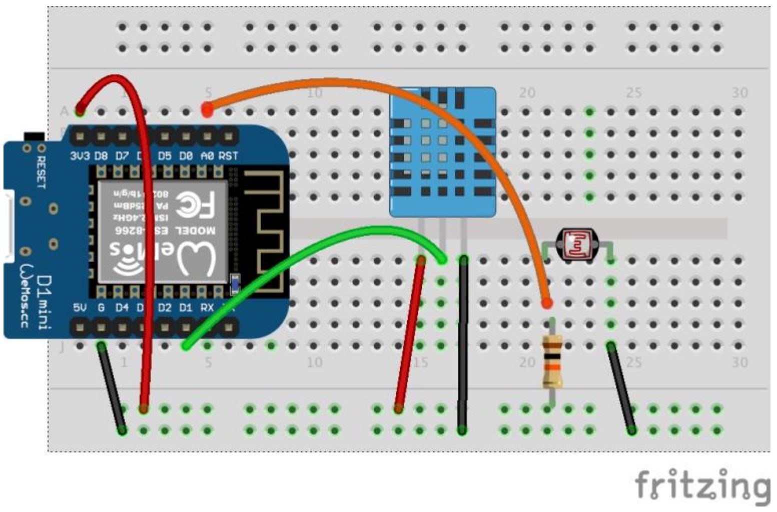
If you're not familiar with the use of a breadboard, consult Appendix 5: Background information on breadboards.

In addition to the resources for Activity 2, you will need:

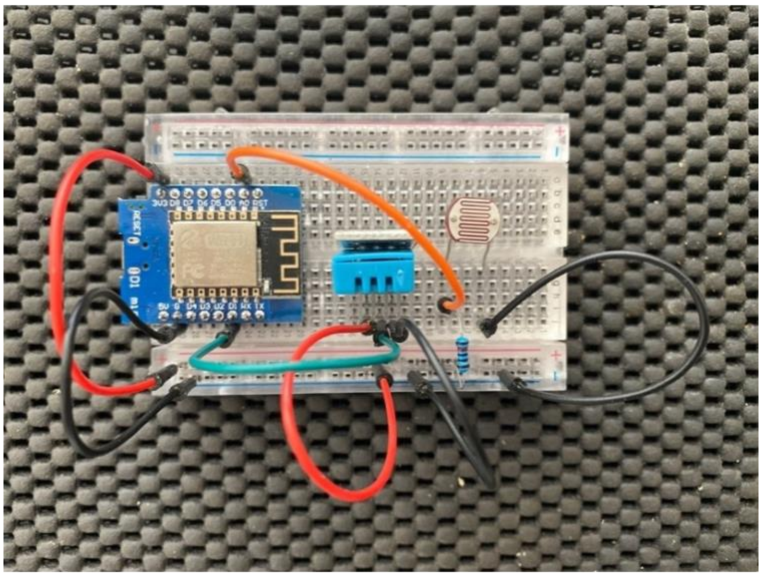
- a DHT11 temperature and humidity sensor
- more Dupont wires.



The circuit diagram is useful to trace connections



This schematic diagram is useful to check wiring


This photo is useful to check your layout

The sketch

```
// We have the library in our Arduino environment, but we need to specify that it
needs to be referred to in this particular sketch

#include “DHTesp.h”


int lightLevel = 0;


DHTesp dht; // Create an instance of the class DHTesp and call it dht for
simplicity


void setup()

{
  Serial.begin(9600);
  Serial.println(); // print an empty line so that the next one gets printed
cleanly
  String thisBoard = ARDUINO_BOARD; // get the board description
  Serial.println(thisBoard); // print the description of the board used

  // the backslash t combination below is ASCII for a TAB character,
  // so all the numbers produced are structured so that
  // they will line up nicely when pasted into a spreadsheet
  Serial.println("Status\tLight\tHumidity (%)\tTemperature (C)\t(F)\tHeatIndex
(C)\t(F)");

  // Connect DHT sensor to General Purpose Input Output (GPIO) pin 5,
  // which is on pin D1 just to annoy you: refer to the graphics in Activity 0
  dht.setup(5, DHTesp::DHT11);
}

void loop()
{
  lightLevel = analogRead(A0); // get the analog value from the LDR circuit

  // devices can only send data at a certain rate,
  // so there's not much use asking for data if they can't send it.
  // Let's ask the device how often we can ask for data
  delay(dht.getMinimumSamplingPeriod());

  // make a variable to hold the humidity value
  // which will be a floating point number
  float humidity = dht.getHumidity();

  // make a variable to hold the temperature value
  // which will be a floating point number
  float temperature = dht.getTemperature();

  Serial.print(dht.getStatusString());

  Serial.print("\t");
  Serial.print(lightLevel);
  Serial.print("\t");
  Serial.print(humidity, 1);
  Serial.print("\t");
  Serial.print(temperature, 1);
  Serial.print("\t");
  Serial.print(dht.toFahrenheit(temperature), 1);
  Serial.print("\t");
  Serial.print(dht.computeHeatIndex(temperature, humidity, true), 1);
  Serial.print("\t");
  Serial.println(dht.computeHeatIndex(dht.toFahrenheit(temperature), humidity, 
true), 1);
  delay(2000);
}
```

We can enter the formula for lux in the Arduino code

```
int lightLevel = 0;

DHTesp dht;

void setup()
{
  Serial.begin(9600);
  Serial.println(); // print an empty line so that the next one gets printed 
cleanly
  String thisBoard = ARDUINO_BOARD; // get the board description
  Serial.println(thisBoard); // print the description of the board used

  // the backslash t combination below is ASCII for a TAB character,
  // so all the numbers produced are structured to
  //  line up nicely when pasted into a spreadsheet
  Serial.println("Status\tLight\tHumidity (%)\tTemperature (C)\t(F)\tHeatIndex 
(C)\t(F)");

  // Connect DHT sensor to General Purpose Input Output (GPIO) pin 5,
  // which is on pin D1 just to annoy you: refer to the graphics in Activity 0
  dht.setup(5, DHTesp::DHT11);
}

void loop()
{
  lightLevel = analogRead(A0); // get the analog value from the LDR circuit

  // devices can only send data at a certain rate,
  // so there's not much use asking for data if they can't send it.
  // Let's ask the device how often we can ask for data
  delay(dht.getMinimumSamplingPeriod());

  // make a variable to hold the humidity value
  // which will be a floating point number
  float humidity = dht.getHumidity();

  // make a variable to hold the temperature value
  // which will be a floating point number
  float temperature = dht.getTemperature();

  Serial.print(dht.getStatusString());


  Serial.print("\t");
  Serial.print(lightLevel);
  Serial.print("\t");
  Serial.print(humidity, 1);
  Serial.print("\t");
  Serial.print(temperature, 1);
  Serial.print("\t");
  Serial.print(dht.toFahrenheit(temperature), 1);
  Serial.print("\t");
  Serial.print(dht.computeHeatIndex(temperature, humidity, true), 1);
  Serial.print("\t");
  Serial.println(dht.computeHeatIndex(dht.toFahrenheit(temperature), humidity,
true), 1);
  delay(2000);
}
```

Activity 5 – Outputs that are more meaningful

Why?

Temperature and humidity are useful measures of an environment, but combinations of them can produce differing levels of comfort.

AFor example, a temperature of 35 °C might be okay unless the humidity is high, in which case it’s not.

We need to know if the combination of humidity and temperature is comfortable or too dry, or where it is on a range from just comfortable to a severe level of discomfort.

Decomposition

For this activity, we can decompose the problem to:

```
Find out what functions the library offers

Supply the required data in the correct format to the function of our choice
```

System set-up

What does the DHTesp library allow us to do?

Developers often use a repository called GitHub to store, share and explain their work.

As mentioned earlier, the DHTesp library is stored and documents at: https://github.com/beegee-tokyo/DHTesp

Opening this site in a web browser will reveal a lot of things that may not make sense at first so let’s ignore them for the moment and scroll down to read the README.md file starting at **Functions.**

Here is a list of functions that the library offers, including that data needs to be supplied to the function if required.

Functions such as ***`float getHumidity()`*** have nothing in the parentheses therefore require no data – they simply report the humidity as a floating point number.

But functions such as byte computePerception(float temperature, float percentHumidity, bool isFahrenheit=false); need a bit more interpretation.

The function:

- will return a byte value: 8 bits
- requires the temperature and percent humidity as floating point numbers, and expects the temperature to be in Celsius (isFahrenheit is false).

The single bytes returned will be:

```
0 -> Dry
1 -> Very comfortable
2 -> Comfortable
3 -> Ok
4 -> Uncomfortable
5 -> Quite uncomfortable
6 -> Very uncomfortable
7 -> Severe discomfort
```

To make this more meaningful, we'll need to change the numerals into text.

Let's modify the sketch used in Activity 4 to use:

```
computePerception(temperature, percentHumidity, isFahrenheit=false)
```

and replace the temperature and Heat index in Fahrenheit used in the previous sketch.

So, we'll have to work out a way of looking up the number and retrieving the text that follows it.

We'll have to the structure the data into an array so that it suits our purpose:

```
{"Dry", "Very comfortable", "Comfortable", "Ok", "Uncomfortable", "Quite
uncomfortable", "Very uncomfortable", "Severe discomfort"};
```

But you could substitute your own words here; for example:

```
{"Dry as", "Yeah, like this", "Comfy", "Not bad", "Not Happy", "Nah. Don't like
this", " How hot is this?", "So hot! You've gotta be kidding!"};
```

Have a look at the sketch and note that the array is numbered from zero, so the 0th entry is 'Dry', the 2nd entry is 'Comfortable' and so forth.

It's common for all sort of things computing to start at zero, just like the activities in this document!

The sketch

```
#include "DHTesp.h"


// declare an array called pertext, that will be a constant made of characters

// separated (or delineated) by commas.

// The words describe the perception of the conditions


// We can then use an integer variable (pernum)

// to extract the term that best describes the conditions

const char* pertext[] = {"Dry", "Very comfortable", "Comfortable", "Ok",
"Uncomfortable", "Quite uncomfortable", "Very uncomfortable", "Severe
discomfort"};

int pernum = 0;


int lightLevel = 0;


// declare and initiate variables to hold the temperature and humidity values

// which will be a floating point number

float temperature = 0;

float humidity = 0;


DHTesp dht;


void setup()
```

```
{
  Serial.begin(9600);
  Serial.println(); // print an empty line so that the next one gets printed
cleanly

  String thisBoard = ARDUINO_BOARD; // get the board description
  Serial.println(thisBoard); // print the description of the board used

  // the backslash t combination below is ASCII for a TAB character,
  // so all the numbers produced are structured so that
  // they will line up nicely when pasted into a spreadsheet
  Serial.println("Status\tLight\tHumidity (%)\tTemperature (C)\tHeatIndex
(C)\tComfort");

  // Connect DHT sensor to General Purpose Input Output (GPIO) pin 5,
  // which is on pin D1 just to annoy you:
  dht.setup(5, DHTesp::DHT11);
}

void loop()
{
  lightLevel = analogRead(A0); // get the analog value from the LDR circuit



  // devices can only send data at a certain rate,
  // so there's not much use asking for data if they can't send it.
  // Let's ask the device how often we can ask for data
  delay(dht.getMinimumSamplingPeriod());

  // set variables to hold temperature and humidity values
  humidity = dht.getHumidity();
  temperature = dht.getTemperature();

  pernum = (dht.computePerception(temperature, humidity, false), 1);

  Serial.print(dht.getStatusString());

  Serial.print("\t");
  Serial.print(lightLevel);
  Serial.print("\t");
  Serial.print(humidity, 1);
```

```
    Serial.print("\t\t");
    Serial.print(temperature, 1);
    Serial.print("\t\t");
    Serial.print(dht.computeHeatIndex(temperature, humidity, false), 1);
    Serial.print("\t\t");
    Serial.println(pertext[pernum]);
    delay(2000);
  }
```

What am I seeing in the Serial Monitor?

Check that the output is consistent with what you expect.

Activity 6 – Wireless techniques for collecting data

This activity is designed as a background exercise, directing research that will involve learning how to connect to a cloud-based service and the security issues that may arise from such a process.

There is no algorithm or coding per se.

Why?

As we've seen in previous sketches, the use of \t in serial output means that data can be structured into columns and copy/pasted into a spreadsheet. (You can turn off Autoscroll to make selection of data easier and turn off and timestamp depending on your needs.)

However, this method (viewing the data on a computer screen) relies on the microcontroller being tethered to a computer using a serial cable and sometimes that is not feasible.

Because the ESP family of microcontrollers have wi-fi, we can send data out to a remote device or location wirelessly.

If you structure your data according to their requirements, you can send the data to several websites which can optionally chart the data and present these charts for public view via their browser.

The Internet of Things (IoT) is reliant on the wireless transmission of datasets, via wi-fi; bluetooth; cellular data on 3G, 4G and 5G.; Other low-power protocols such as LoraWAN, SigFox and NB-IoT are also available. You can find comparisons of each at: https://www.iotforall.com/iot-connectivity-comparison-lora-sigfox-rpma-lpwan-technologies

In this activity, we'll use wi-fi to send data to the online data aggregator: Thingspeak.com. Students may also be interested in exploring other cloud-based services such as TagoIO. There are many more, but we'll work with ThingSpeak here. It offers a free account that suits the following activities.

System set-up – thingspeak.com

From the ThingSpeak website:

> *'ThingSpeak is an IoT analytics platform service that allows you to aggregate, visualize, and analyze live data streams in the cloud. You can send data to ThingSpeak from your devices, create instant visualization of live data, and send alerts.'*

Users can create a free account which offers 3 channels, limiting update intervals to 15 seconds, which is fine for student use.

A guide to what is possible with this platform can be found here: https://au.mathworks.com/solutions.html?s_tid=gn_sol

The ESP family will talk to most wi-fi access points so that they can reach ThingSpeak.com, but your school may have issues with random devices accessing its wi-fi.

The teacher or student could use their own phone as a hotspot as there is very little data involved in these activities, but it may be prudent to purchase a portable hotspot.

In Australia there are 2 popular options:

- Optus Link Zone (Alcatel) pre-paid $69, 4G, 14 user limit: https://www.harveynorman.com.au/optus-link-zone-pre-paid-mobile-broadband.html
- Telstra 4GX Hotspot, 20 users, $27 per month for 12 months including Telstra Air access and no excess data charges (speed capped at 1.5 Mbps, which is more than adequate for these projects): https://www.telstra.com.au/internet/mobile-broadband/telstra-4gx-hotspot#orderSummary

Prices correct at time of writing

If you want greater control over access to the portable hotspot (for example, to prevent students taking advantage of an unfiltered internet access device), then you can use that hotspot's control page (see your hotspot manual) to restrict access to the media access control (MAC) addresses of the ESP devices.

MAC addresses are 48-bit addresses, usually expressed in hexadecimal (to save space!) that uniquely identify each internet-capable device.

ESP MAC addresses are found by including the following in your sketch:

```
Serial.print("MAC: ");
Serial.println(WiFi.macAddress());
```

MAC addresses can be changed or spoofed, and this is an interesting avenue for discussion, but any change will be reversed whenever the microcontroller is restarted.

Decomposition

For this activity, we can decompose the problem to:

```
Make a connection to a WiFi network

Using the network, authenticate and connect to a cloud platform

Build a structure to accept data to be displayed

Structure data in a manner that the platform requires

Upload data to the platform

Test
```

Security issues of data in the wild

IoT projects are perfect for investigating security issues as a lot of devices don't use secure communication (such as https) or are unable to be updated once installed in the wild. Devices can be easily purchased by any individual who can copy/paste software from the web and install insecure devices that may be recruited into botnets or act as attack surfaces for bad actors.

Some questions for research may include:

- Is it possible for a nation's electrical grid to be brought down by a botnet consisting of several thousand programmable light bulbs that have had their operating system compromised?
- Are microcontrollers in each light bulb manufactured to such tight cost constraints that they do not have sufficient memory to be updated for a security hole patch?

Starting points for research:

https://www.cnet.com/news/iot-attacks-hacker-kaspersky-are-getting-worse-and-no-one-is-listening/?ftag=CMG-01-10aaa1b

https://www.zdnet.com/article/this-old-security-vulnerability-left-millions-of-internet-of-things-devices-vulnerable-to-attacks/

https://owasp.org/www-pdf-archive/OWASP-IoT-Top-10-2018-final.pdf

Activity 7 – Lux and temp/humidity to ThingSpeak

Decomposition

For this activity, we can decompose the problem to:

```
Create an account to use on the ThingSpeak website

Get all the data from the connected LDR and DHT11 sensors (we’ve already done
that in Activities 4, 5 and 6)

Set up the ESP microcontroller to “talk” WiFi. Note, the microcontroller can act
both as a WiFi Access Point or as a WiFi client or station. Turn off the WiFi
Access Point by calling WiFi.mode(WIFI_STA)

Define the WiFi access credentials

Check connectivity

Serially print the sensor values so we can double check what’s happening

Structure the data to suit the Application Programming Interface provided by
ThingSpeak

Send the data

Check the website
```

System set-up

Note that we don’t need a library for this activity as ThingSpeak offers an application programming interface (API): a means of POSTing data to the website by structuring the data and sending it to the ‘api.thingSpeak.com’ website.

For more information about API see: https://en.wikipedia.org/wiki/API

The sketch

```
#include <ESP8266WiFi.h>

#include "DHTesp.h"


// declare an integer variable to hold the voltage value

// on the analog pin and initialise it to zero

int lightLevel = 0;

int lux = 0;

String api_key = "XXXX";        //  Enter your Write API key for ThingSpeak,
surrounded by inverted commas

const char *ssid =  "XXXX";     // replace with your WiFi name (ssid), surrounded
by inverted commas

const char *pass =  "XXXX";     // put in your WiFi Password, surrounded by
inverted commas

const char *server = "api.thingspeak.com";


DHTesp dht;

WiFiClient client;  //creates a client instance for connection defined in
client.connect()
```

```
void setup()
{


  WiFi.mode(WIFI_STA); //make the chip WiFi client (STAtion) only


  Serial.begin(9600);
  Serial.println(); // print an empty line so that the next one gets printed
cleanly
  String thisBoard = ARDUINO_BOARD; // get the board description
  Serial.println(thisBoard); // print the description of the board used


  // the backslash t combination below is ASCII for a TAB character,
  // so all the numbers produced are structured so that
  // they will line up nicely when pasted into a spreadsheet
  Serial.println("Status\tLight\tHumidity (%)\tTemperature (C)\t(F)\tHeatIndex
(C)\t(F)");


  // Connect DHT sensor to General Purpose Input Output (GPIO) pin 5,
  // which is on pin D1 just to annoy you:
  dht.setup(5, DHTesp::DHT11);


  Serial.println("Connecting to ");     // write the connection parameters for
WiFi out on the serial line
  // we can then see progress using the Serial Monitor
  Serial.println(ssid);                 // show the WiFi access point name


  WiFi.begin(ssid, pass);               // handover connection credentials


  while (WiFi.status() != WL_CONNECTED) // ! is a negator, so this will run while
the connection
    // hasn't been finalised and print "." on the Serial Monitor
    // every half second


  {
    delay(500);
    Serial.print("."); //lotsa dots until connection has been established
  }
  Serial.println("");
  Serial.println("WiFi connected"); //yay
}
```

```
void loop()
{
  lightLevel = analogRead(A0);  // get the analog value from the LDR circuit

  // make a variable to hold the humidity value
  // which will be a floating point number
  float humidity = dht.getHumidity();

  // make a variable to hold the temperature value
  // which will be a floating point number
  float temperature = dht.getTemperature();

  if (isnan(humidity) || isnan(temperature)) // "isnan" means Not A Number: if
the values are not a number
    // then something's wrong
  {
    Serial.println("Failed to read from DHT sensor!");
    return;
  }

  lux = 14483 * pow(lightLevel, -.865);
  // lux determined by regression analysis of actual measurements from
  // a luxmeter on an iPhone 11 and the raw data from the LDR circuit

  Serial.print(dht.getStatusString());

  Serial.print("\t");
  Serial.print(lux);
  Serial.print("\t");
  Serial.print(humidity, 1);
  Serial.print("\t");
  Serial.print(temperature, 1);
  Serial.print("\t");
  Serial.print(dht.computeHeatIndex(temperature, humidity, true), 1);
  Serial.print("\t");
  Serial.println(dht.computeHeatIndex(dht.toFahrenheit(temperature), humidity,
true), 1);

  if (client.connect(server, 80))  // if connect to the server on port 80 is true
(are there security issues here?)
  { // send data
```

```
    String data_to_send = api_key; // concatenate all the values we need into a
string

    // with the format that ThingSpeak requires

    data_to_send += "&field1=";

    data_to_send += humidity;

    data_to_send += "&field2=";

    data_to_send += temperature;

    data_to_send += "&field3=";

    data_to_send += lux;

    data_to_send += "\r\n\r\n"; // "\r" is ASCII for carriage return, likewise
"\n" is new line.

    // This is a hangover from when printers were just electric typewriters


    client.print("POST /update HTTP/1.1\n"); // all the client print stuff goes
via WiFi

    client.print("Host: api.thingspeak.com\n");

    client.print("Connection: close\n");

    client.print("X-THINGSPEAKAPIKEY: " + api_key + "\n");

    client.print("Content-Type: application/x-www-form-urlencoded\n");

    client.print("Content-Length: ");

    client.print(data_to_send.length());

    client.print("\n\n");

    client.print(data_to_send);

    delay(1000); //time for things to settle?


    Serial.println(" Sent to Thingspeak.");


  }

  client.stop(); // turn off WiFi unless needed to save energy


  Serial.println("Waiting...");


  /* ThingSpeak needs minimum 15 sec delay between updates, but it's set here to
5 minutes,

    which matches how the ThingSpeak charts are set up */

  delay(300000);


}
```

Activity 8 – Exploring other useful sensors and cloud services

CO_2 sensors

These come in 2 different styles: actual CO_2 measurement and implied CO_2 measurement.

The implied sensors measure volatile organic compounds (VOCs). These are carbon-based (organic) chemical compounds that evaporate at room temperature (volatile), and are present in normal human breath in a set proportion to the expelled CO_2. An algorithm assumes that VOCs detected are due to people's breath and then uses a correlation between VOC and exhaled CO_2. This is not always valid, especially after sports when students may apply a deodorant, but close enough for many measurements. They can cost upwards of A$10.

True CO_2 sensors are much more expensive and generally assume a level of 400 parts per million. More precise measurements can be made by calibration of these sensors, but that is outside the scope of this exercise.

UV sensors

These sensors determine the level of ultraviolet light.

Common sensors are the UVM-30A and the GUVA-S12SD. The former is advised here because it operates at 3.3 V – the same as the ESP family of microcontrollers. The latter will need a dual voltage supply.

The obvious application for these sensors would be providing a warning for an excess UV from the sun; perhaps alerting by placing a value on the school's webpage using an iFrame. See: https://www.w3schools.com/tags/tag_iframe.ASP

Also, experimentation with UV lamps as growth promotants would be a fertile ground for exploration.

Cloud services – Thingspeak and TagoIO

Both of these services provide free (limited) access for students and experimenters, via differing methods, including an application programming interface (API).

Comparison of the 2 and differing methods of access make for a good assessment exercise, as does an exploration of security issues that may arise from data collection, analysis and visualisation, together with transmission of data from one element of the system to another.

For ThingSpeak, see: https://thingspeak.com

For TagoIO, see: https://tago.io

Bear in mind that there are many other alternatives. See: https://www.sitelike.org/similar/tago.io/

Activity 9 – A possible summative assessment

Note: This activity is a good match with the summative assessment proposed in Appendix 2.

Why?

The ESP/DHT11 combo is a cheap way of gathering temperature and humidity data, but if we have to connect the module to a computer to power it, we lose a lot of that cheapness!

So, let's use battery power. Two AA or AAA batteries in series will give us 3.2 V with fresh batteries, which is more than enough to power the module (2.6–3.6 V), but how long will the battery power last?

What will affect battery power?

To answer this question, let's look at the components of the system that require power.

At a minimum, we have a sensor, a processor and wi-fi all needing power.

But do we need them turned on all the time? Is there any way we can turn the wi-fi off until it's needed?

Decomposition

For this activity, we can decompose the problem to:

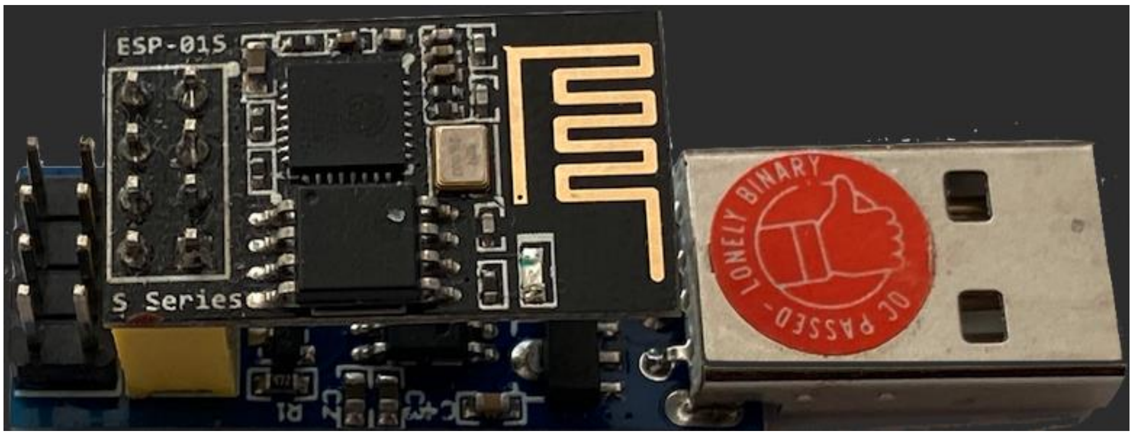
```
Identify the power requirements of each component of the system

Identify the need for each system availability

Find ways of turning off different components until they are needed

Modify the code to turn components on only when needed
```

Chips can be in 3 states: programmed, programmable and unknown (newly purchased)

Programming the board allows its memory to be written to so that you can get it to do your bidding.

This often involves pushing buttons or wiring pins together. See: https://diyprojects.io/esp01-which-programmer-choose-modification-switch-flash-mode/

Thankfully, using the FTDI USB programmer outlined in the introduction, we don't need to do this.



Using the ESP-01 and FTDI combination gives you access to the board for programming

Powering up

Power requirements: 2.6–3.6 V so 2 x AA or AAA batteries will supply this fine, but for how long?

You can turn off the wi-fi Access Point by calling WiFi.mode(WIFI_STA).

Look at Keywords in keywords.txt of the DHTesp library in your Arduino folder.

We find 2 useful functions here:

forceSleepBegin	KEYWORD2	sets power usage to 20 mA (milliamps)
forceSleepWake	KEYWORD2	sets power usage to 500 mA

Battery energy is measure in amp hours, meaning how many amps can be supplied for how long.

The following calculations are best case. In reality, as batteries wear down, they will be able to supply less voltage depending on the type of battery used. Alkaline batteries will drop their voltage too low for the ESP after about 1,000 mAh (Duracell Coppertop figures).

Lithium batteries would be your friend here as they can sustain voltage above 1.5 V before the energy drops below 2,500 mAh. See:
https://electronics.stackexchange.com/questions/134143/when-will-the-aa-battery-voltage-drop

AA alkaline batteries will supply, say, 1,000 mA for one hour or 500mA for 2 hours.

If we collect data at 15-minute intervals, sleeping in between, we would have 96 wake-ups in a day, each for, say 10 seconds, meaning 960 seconds = roughly 15 minutes drawing 500 mA = 125 mAh, and the balance of energy for sleeping.

Sleep energy will be 24 hours − 0.25 hours awake = 23.75 hours

23.75 hours at 20 mA = 475 mAh

Total = 475 (asleep) plus 125 (awake) = 500 mAh per day, which equals 2 days from our AAA batteries.

If we use lithium batteries instead which have 2,500 mAh, we could go for 5 days.

The sketch

Students should modify the sketch used in Activity 6 to accommodate power management (a suggested sketch is below).

What am I seeing in the Serial Monitor?

Students should test by altering code to see results in the Serial Monitor and add battery saving functions called after the statements in the code that collect and send data.

Once all is well, we can leave the serial code in (it won't go anywhere) or delete it and re-enable the known working wi-fi code from Activity 6.

Extension

Note: These links assume a good level of knowledge in electronics and a good degree of expertise in soldering.

Sleep deeper

The ESP-01 can sleep deeper than the software itself can do. Enabling Deep Sleep will drop current to about 3 mA!

This is not simple; you can't do this in software alone as it requires some fine soldering and there are some further software implications, but would make a great extension. See:

https://www.instructables.com/Enable-DeepSleep-on-an-ESP8266-01

Dive deeper into the ESP-01 pins and LEDs

https://www.instructables.com/How-to-use-the-ESP8266-01-pins/

The sketch – including serial output

```
#include <ESP8266WiFi.h>

#include "DHTesp.h"


//  Enter your Write API key for ThingSpeak, surrounded by inverted commas

String api_key = "XXXXXXXXXX";


// replace with your WiFi name (ssid), surrounded by inverted commas

const char *ssid =  "XXXX";


// put in your WiFi Password, surrounded by inverted commas

const char *pass =  "XXXX";

const char *server = "api.thingspeak.com";


// create an instance of the DHT object

DHTesp dht;


// create an instance of the WiFiClient object

WiFiClient client;


void setup()
```

```
{

  // To use GPIO2, we need to set GPIO0 to digital zero, or LOW
  pinMode(0, OUTPUT);
  digitalWrite(0, LOW);

  // make the chip WiFi client (STAtion) only
  WiFi.mode(WIFI_STA);

  Serial.begin(9600);

  // print an empty line so that the next one gets printed cleanly
  Serial.println();

  // get the board description
  String thisBoard = ARDUINO_BOARD;

  // print the description of the board used
  Serial.println(thisBoard);

  // the backslash t combination below is ASCII for a TAB character,
  // so all the numbers produced are structured so that
  // they will line up nicely when pasted into a spreadsheet
  Serial.println("Status\tHumidity (%)\tTemperature (C)");

  // Note that the pin number changes to pin 3 for this microcontroller
  dht.setup(2, DHTesp::DHT11);

  // write the connection parameters for WiFi out on the serial line
  // we can then see progress using the Serial Monitor
  Serial.println("Connecting to ");

  // show the WiFi access point name
  Serial.println(ssid);

  // handover connection credentials
  WiFi.begin(ssid, pass);

  // ! is a negator, so this will run while the connection
  // hasn't been finalised and print "." on the Serial Monitor
  // every half second
  while (WiFi.status() != WL_CONNECTED)

  {
    delay(500);

    //lotsa dots until connection has been established
    Serial.print(".");
  }
  Serial.println("");
  Serial.println("WiFi connected"); //yay
}

void loop()
{

  // make a variable to hold the humidity value
  // which will be a floating point number
  float humidity = dht.getHumidity();

  // make a variable to hold the temperature value
  // which will be a floating point number
  float temperature = dht.getTemperature();

  // check communication with DHT module
  // "isnan" means Not A Number: if the values are not a number
  // then something's wrong
  if (isnan(humidity) || isnan(temperature))
  {
    Serial.println("Failed to read from DHT sensor!");
    delay(1000);
    return;
  }

  Serial.print(dht.getStatusString());

  Serial.print("\t");
  Serial.print(humidity, 1);
  Serial.print("\t");
  Serial.print(temperature, 1);
  Serial.println("\t");

  // if connect to the server on port 80 is true (are there security issues here?
  // Should we be using another port for https?)
  if (client.connect(server, 80))

    // send data
  {


    // concatenate all the values we need into a string
    // with the format that ThingSpeak requires
    String data_to_send = api_key;

    data_to_send += "&field1=";
    data_to_send += humidity;
    data_to_send += "&field2=";
    data_to_send += temperature;
    // "\r" is ASCII for carriage return, likewise "\n" is new line.
    // This is a hangover from when printers were just electric typewriters
    data_to_send += "\r\n\r\n";

    // all the client print stuff goes via WiFi
    client.print("POST /update HTTP/1.1\n");
    client.print("Host: api.thingspeak.com\n");
    client.print("Connection: close\n");
    client.print("X-THINGSPEAKAPIKEY: " + api_key + "\n");
    client.print("Content-Type: application/x-www-form-urlencoded\n");
    client.print("Content-Length: ");
    client.print(data_to_send.length());
    client.print("\n\n");
    client.print(data_to_send);
    Serial.println(" Sent to Thingspeak.");

  }
  // turn off WiFi unless needed to save energy
  // A great opportunity to research low power mode
  // rather than simply turning WiFi off
  client.stop();

  Serial.println("Waiting...");
```

```
  // ThingSpeak needs minimum 15 sec delay between updates,
  // but it's set here to 5 minutes,
  // match this to how the ThingSpeak charts are set up
  delay(300000);

}
```

Activity 10 – Using a system on a chip

Why?

Individual sensors are ideal for starting out or for bespoke solutions, but sometimes a system on a chip (SoC) which carries a plethora of sensors for a specific function is better suited.

Such a SoC is provided by Metriful. See: https://www.metriful.com

What is this?

The Metriful SoC is compatible with a number of boards, including the ESP family, and provides measurement of:

- VOC (from which air quality and carbon dioxide levels can be deduced)
- temperature
- humidity
- air pressure
- sound levels (including weighting based on sound frequencies)
- light levels and spectra (from which lux can be deduced).

Decomposition

For this activity, we can decompose the problem to:

```
Explore the options for the Metriful board, on github.com/metriful/sensor Develop
an algorithm for outputting collected data to Serial only

Choose an IoT cloud platform such as ThingSpeak or TagoIO

Create an account on the platform of choice

IF you elect to go one step further

      Create an account on IFTTT.com and set up to send an email

to you when an environmental variable(s) are out of range

ELSE

      Use the cloud platform to create an iFrame that can be included

In your school's webpage

Present your completed platform page as evidence of your work

Document the procedure for use by a fellow student
```

System set-up

The SoC will happily talk to most boards with both Arduino and Python supported (Python on Raspberry Pi only at time of writing).

One warning: the Metriful examples in its library uses these lines of code:

```
while (!Serial)
 {
   ; // Wait for serial to connect
 }
```

The '!' or pling character means negate what follows. So, this line means 'if serial communications are not established'.

This is not supported by all ESP boards. They may ignore it, but I have found that some (particularly the Jaycar ones) don't – they just stall here, even if the serial connection is up and running.

The solution is to delete, or better still, comment out with a reason so that anyone else using your sketches is also alerted to this problem. See: https://arduino.stackexchange.com/questions/65017/arduino-ide-while-serial

Extension

Create an account on IFTTT.com and set up to send an email to you whenever the measurement for the environmental variable(s) are out of range, or if the power supply falls below 3 V.

Use your preferred cloud platform to create an iFrame that can be included in your school's webpage.

The sketch

```
/*
   IoT_cloud_logging.ino

   Example IoT data logging code for the Metriful MS430.

   This example is designed for the following WiFi enabled hosts:
     Arduino Nano 33 IoT
     Arduino MKR WiFi 1010
     ESP8266 boards (e.g. Wemos D1, NodeMCU)
     ESP32 boards (e.g. DOIT DevKit v1)

   Environmental data values are measured and logged to an internet
   cloud account every 100 seconds, using a WiFi network.

   Copyright 2020 Metriful Ltd.
   Licensed under the MIT Licence:
   Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal in
the Software without restriction, including without limitation the rights to use,
copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the
Software, and to permit persons to whom the Software is furnished to do so,
subject to the following conditions:
https://github.com/metriful/sensor/blob/master/LICENSE.txt

   For code examples, datasheet and user guide, visit
   https://github.com/metriful/sensor
   Adapted and Modified Levins, 2021
```

```
*/

#include <Metriful_sensor.h>
#include <ESP8266WiFi.h>
#include "ThingSpeak.h" // always include ThingSpeak header file after other header files and custom macros

//////////////////////////////////////////////////////////
// USER-EDITABLE SETTINGS

// How often to read and log data (every 100 or 300 seconds)
// Note: due to data rate limits on free cloud services, this should
// be set to 100 or 300 seconds, not 3 seconds.
uint8_t cycle_period = CYCLE_PERIOD_100_S; /* uint8_t defines the variable cycle_period as a fixed width of 8 bits.
  change this to char to make it easier for readers?
  Looks like CYCLE_PERIOD is part of a header file? It's only used once here
  I think I can make a case for uint8_t with UTF8 text.
  Indeed, char seems to imply a character, whereas in the context of a UTF8 string,
  it may be just one byte of a multibyte character.
  Using uint8_t could make it clear that one shouldn't expect a character at every position.
  In other words that each element of the string/array is an arbitrary integer
  that one shouldn't make any semantic assumptions about. */

// The details of the WiFi network:
const char *ssid = "XXXX"; // network SSID (name)ch
const char *password = "XXXX"; // network password

// IoT cloud settings
// This example uses the free IoT cloud hosting services provided
// by Tago.io or Thingspeak.com
// Other free cloud providers are available.
// An account must have been set up with the relevant cloud provider
// and a WiFi internet connection must exist. See the accompanying
// readme and User Guide for more information.

// The chosen account's key/token must be put into the relevant define below.
#define TAGO_DEVICE_TOKEN_STRING "PASTE YOUR TOKEN HERE WITHIN QUOTES"
#define THINGSPEAK_API_KEY_STRING "XXXXXXXXXXXX"
unsigned long myChannelNumber = XXXXXXXX;
```

```
// Choose which provider to use
bool useTagoCloud = false;
// To use the ThingSpeak cloud, set: useTagoCloud=false

// END OF USER-EDITABLE SETTINGS
/////////////////////////////////////////////////////////

/*#if !defined(HAS_WIFI)
  #error ("This example program has been created for specific WiFi enabled hosts
only.")
  #endif*/

WiFiClient client;

// Buffers for assembling http POST requests
char postBuffer[450] = {0};
char fieldBuffer[70] = {0};

// Structs for data
AirData_t airData = {0}; // an array for data?
AirQualityData_t airQualityData = {0};
LightData_t lightData = {0};
ParticleData_t particleData = {0};
SoundData_t soundData = {0};

void setup() {
  // Initialise the host's pins, set up the serial port and reset:

  //  Serial.begin(115200);

  SensorHardwareSetup(I2C_ADDRESS);

  //  connectToWiFi(SSID, password);
  WiFi.begin(ssid, password);



  // Apply chosen settings to the MS430
  uint8_t particleSensor = PARTICLE_SENSOR;
  TransmitI2C(I2C_ADDRESS, PARTICLE_SENSOR_SELECT_REG, &particleSensor, 1);
  TransmitI2C(I2C_ADDRESS, CYCLE_TIME_PERIOD_REG, &cycle_period, 1);

  // Enter cycle mode
  ready_assertion_event = false;
  TransmitI2C(I2C_ADDRESS, CYCLE_MODE_CMD, 0, 0);
}

void loop() {

  // Wait for the next new data release, indicated by a falling edge on READY
  while (!ready_assertion_event) {
    yield();
  }
  ready_assertion_event = false;

  /* Read data from the MS430 into the data structs.
    For each category of data (air, sound, etc.) a pointer to the data struct is
    passed to the ReceiveI2C() function. The received byte sequence fills the
    struct in the correct order so that each field within the struct receives
    the value of an environmental quantity (temperature, sound level, etc.)
  */

  // Air data
  // Choose output temperature unit (C or F) in Metriful_sensor.h
  ReceiveI2C(I2C_ADDRESS, AIR_DATA_READ, (uint8_t *) &airData, AIR_DATA_BYTES);

  /* Air quality data
    The initial self-calibration of the air quality data may take several
    minutes to complete. During this time the accuracy parameter is zero
    and the data values are not valid.
  */
  ReceiveI2C(I2C_ADDRESS, AIR_QUALITY_DATA_READ, (uint8_t *) &airQualityData,
AIR_QUALITY_DATA_BYTES);

  // Light data
  ReceiveI2C(I2C_ADDRESS, LIGHT_DATA_READ, (uint8_t *) &lightData,
LIGHT_DATA_BYTES);

  // Sound data
  ReceiveI2C(I2C_ADDRESS, SOUND_DATA_READ, (uint8_t *) &soundData,
SOUND_DATA_BYTES);


  /* Particle data
    This requires the connection of a particulate sensor (invalid
    values will be obtained if this sensor is not present).
    Specify your sensor model (PPD42 or SDS011) in Metriful_sensor.h
    Also note that, due to the low pass filtering used, the
    particle data become valid after an initial initialisation
    period of approximately one minute.
  */
  if (PARTICLE_SENSOR != PARTICLE_SENSOR_OFF) {
    ReceiveI2C(I2C_ADDRESS, PARTICLE_DATA_READ, (uint8_t *) &particleData,
PARTICLE_DATA_BYTES);
  }


  // Check that WiFi is still connected
  uint8_t wifiStatus = WiFi.status();
  /*  if (wifiStatus != WL_CONNECTED)


    {
      // There is a problem with the WiFi connection: attempt to reconnect.
      Serial.print("Wifi status: ");
      Serial.println(interpret_WiFi_status(wifiStatus));
      connectToWiFi(SSID, password);*/


  //WiFi.begin(ssid, password); repeated from earlier


  while (WiFi.status() != WL_CONNECTED) //! is a negator, so this will run while
the connection
    // hasn't been finalised and print "." on the Serial Monitor
    // every half second


  {
    delay(500);
    Serial.print(".");
  }
  Serial.println("");
  Serial.println("WiFi connected");
  ready_assertion_event = false;
  // }
  // Send data to the cloud
  if (useTagoCloud) {
    http_POST_data_Tago_cloud();
  }
  else {
    http_POST_data_Thingspeak_cloud();
  }
}


/* For both example cloud providers, the following quantities will be sent:
  1 Temperature (C or F)
  2 Pressure/Pa
  3 Humidity/%
  4 Air quality index
  5 bVOC/ppm
  6 SPL/dBA
  7 Illuminance/lux
  8 Particle concentration

   Additionally, for TagoIO, the following is sent:
  9  Air Quality Assessment summary (Good, Bad, etc.)
  10 Peak sound amplitude / mPa
*/

// Assemble the data into the required format, then send it to the
// Tago.io cloud as an HTTP POST request.
void http_POST_data_Tago_cloud(void) {
  client.stop();
  if (client.connect("api.tago.io", 80)) {
    client.println("POST /data HTTP/1.1");
    client.println("Host: api.tago.io");
    client.println("Content-Type: application/json");
    client.println("Device-Token: " TAGO_DEVICE_TOKEN_STRING);

    uint8_t T_intPart = 0;
    uint8_t T_fractionalPart = 0;
    bool isPositive = true;
    getTemperature(&airData, &T_intPart, &T_fractionalPart, &isPositive);
    sprintf(postBuffer, "[{\"variable\":\"temperature\",\"value\":%s%u.%u}",

            isPositive ? "" : "-", T_intPart, T_fractionalPart);


    sprintf(fieldBuffer, ",{\"variable\":\"pressure\",\"value\":%" PRIu32 "}",
airData.P_Pa);

    strcat(postBuffer, fieldBuffer);


    sprintf(fieldBuffer, ",{\"variable\":\"humidity\",\"value\":%u.%u}",

            airData.H_pc_int, airData.H_pc_fr_1dp);

    strcat(postBuffer, fieldBuffer);


    sprintf(fieldBuffer, ",{\"variable\":\"aqi\",\"value\":%u.%u}",

            airQualityData.AQI_int, airQualityData.AQI_fr_1dp);

    strcat(postBuffer, fieldBuffer);


    sprintf(fieldBuffer, ",{\"variable\":\"aqi_string\",\"value\":\"%s\"}",

            interpret_AQI_value(airQualityData.AQI_int));

    strcat(postBuffer, fieldBuffer);


    sprintf(fieldBuffer, ",{\"variable\":\"bvoc\",\"value\":%u.%02u}",

            airQualityData.bVOC_int, airQualityData.bVOC_fr_2dp);

    strcat(postBuffer, fieldBuffer);


    sprintf(fieldBuffer, ",{\"variable\":\"spl\",\"value\":%u.%u}",

            soundData.SPL_dBA_int, soundData.SPL_dBA_fr_1dp);

    strcat(postBuffer, fieldBuffer);


    sprintf(fieldBuffer, ",{\"variable\":\"peak_amp\",\"value\":%u.%02u}",

            soundData.peak_amp_mPa_int, soundData.peak_amp_mPa_fr_2dp);

    strcat(postBuffer, fieldBuffer);


    sprintf(fieldBuffer, ",{\"variable\":\"particulates\",\"value\":%u.%02u}",

            particleData.concentration_int, particleData.concentration_fr_2dp);

    strcat(postBuffer, fieldBuffer);


    sprintf(fieldBuffer, ",{\"variable\":\"illuminance\",\"value\":%u.%02u}]",

            lightData.illum_lux_int, lightData.illum_lux_fr_2dp);

    strcat(postBuffer, fieldBuffer);


    size_t len = strlen(postBuffer);

    sprintf(fieldBuffer, "Content-Length: %u", len);
    client.println(fieldBuffer);
    client.println();
    client.print(postBuffer);
  }
  else {
    Serial.println("Client connection failed.");
  }
}

// Assemble the data into the required format, then send it to the
// ThinSspeak.com cloud as an HTTP POST request.
void http_POST_data_Thingspeak_cloud(void) {
  client.stop();
  if (client.connect("api.thingspeak.com", 80)) {
    client.println("POST /update HTTP/1.1");
    client.println("Host: api.thingspeak.com");
    client.println("Content-Type: application/x-www-form-urlencoded");

    strcpy(postBuffer, "api_key=" THINGSPEAK_API_KEY_STRING);

    uint8_t T_intPart = 0;
    uint8_t T_fractionalPart = 0;
    bool isPositive = true;
    getTemperature(&airData, &T_intPart, &T_fractionalPart, &isPositive);
    sprintf(fieldBuffer, "&field1=%s%u.%u", isPositive ? "" : "-", T_intPart,
T_fractionalPart);
    strcat(postBuffer, fieldBuffer);

    sprintf(fieldBuffer, "&field2=%" PRIu32, airData.P_Pa);
    strcat(postBuffer, fieldBuffer);

    sprintf(fieldBuffer, "&field3=%u.%u", airData.H_pc_int, airData.H_pc_fr_1dp);
    strcat(postBuffer, fieldBuffer);

    sprintf(fieldBuffer, "&field4=%u.%u", airQualityData.AQI_int,
airQualityData.AQI_fr_1dp);
    strcat(postBuffer, fieldBuffer);

    sprintf(fieldBuffer, "&field5=%u.%02u", airQualityData.bVOC_int,
airQualityData.bVOC_fr_2dp);
    strcat(postBuffer, fieldBuffer);
    sprintf(fieldBuffer, "&field6=%u.%u", soundData.SPL_dBA_int,
soundData.SPL_dBA_fr_1dp);
    strcat(postBuffer, fieldBuffer);

    sprintf(fieldBuffer, "&field7=%u.%02u", lightData.illum_lux_int,
lightData.illum_lux_fr_2dp);
    strcat(postBuffer, fieldBuffer);

    sprintf(fieldBuffer, "&field8=%u.%02u", particleData.concentration_int,
            particleData.concentration_fr_2dp);
    strcat(postBuffer, fieldBuffer);

    size_t len = strlen(postBuffer);
    sprintf(fieldBuffer, "Content-Length: %u", len);
    client.println(fieldBuffer);
    client.println();
    client.print(postBuffer);
    Serial.println(fieldBuffer);
    Serial.println();
    Serial.print(postBuffer);
  }
  else {
    Serial.println("Client connection failed.");
  }
}
```

Glossary

analogue or analog
Of or relating to a signal voltage which can take any value in a range. For example, a light dependent resistor may present a voltage between 0 and 1 V to an analog inout on a microcontroller. The microcontroller will divide this range up into a binary number between 0 and 255 (8-bit or 2^8 analog) or 0–1023 (10-bit or 2^{10} analog).

general-purpose programming languages
Programming languages in common use designed to solve a wide range of problems. Examples include C#, C++, Java, JavaScript, Python, Ruby and Visual Basic. In this document, we concentrate on the Arduino variant of C++.

libraries
Pieces of code contained in a file. These pieces carry out specific tasks that need to be written once and re-used each time the environment of the library is used. For example, the library 'ESP8266Wi-fi.h' contains objects that allow interaction with the wi-fi capabilities of all members of the ESP family.

lux
A unit of the illumination or amount of light falling on a surface. For example, we can use a light meter or lux meter to measure the light reflected from a person's face to set the exposure and shutter speed of a camera. This is done automatically in most cameras. Lux is calculated from light levels, but also the colour make-up of the incident light.

objects
A programming structure that contains (encapsulates) properties (data) and methods (functions). An object is a variable.

pseudocode
A way of showing algorithms without use of any specific programming language. This makes the algorithm easy to understand for everyone, whatever programming language they might use. Pseudocode is usually written in English text. Purists would insist that pseudocode be written with no recognisable computer language words, but some common operation words, for example, *if, then* and *else,* are commonly found in pseudocode.

variable

A variable is the name given to a reserved area of computer memory to hold data that can change. They need to be declared in program code so that the computer knows to reserve sufficient memory. For example, a variable may be declared to hold the reading from a sensor as an integer (counting numbers: 0,1,2,3,4,5 etc.) or as a floating-point number (fractional numbers such as 1.56789), or it may be intended to hold a piece of text. In each case, the variable will need to be declared as a type that matches the data it is to hold. Variables may also need to be initialised to a set value before use.

wi-fi
A communications standard which is short for wireless fidelity. This networking system can have clients or access points to which clients connect. The access point can relay the client data to, or retrieve data from, a network for them. Some devices can act as both clients and access points.

See also the glossary for the Australian Curriculum: Technologies:
www.australiancurriculum.edu.au/f-10-curriculum/technologies/glossary/