

Interacting with coding across the curriculum

A handbook for teachers

Foreword

In his 2012 collection of essays, Marc Prensky included as chapter 23, *The True 21st Century Literacy is Programming*, a piece originally published in Edutopia as “*Programming is the new Literacy.*” In introducing this essay, Prensky wrote:

Programming like problem-solving, is a skill that should be infused into all our school subjects from math and science to social studies and English. (p192)

The Australian Government’s “Learning Potential” website (learningpotential.gov.au) says that “When children learn to code, it helps them to develop essential skills such as problem solving, logic and critical thinking.”

The Australian Digital Technologies curriculum states that its aim is to develop the knowledge, understanding and skills to ensure that, individually and collaboratively, students:

- design, create, manage and evaluate sustainable and innovative digital solutions to meet and redefine current and future needs
- use computational thinking and the key concepts of abstraction; data collection, representation and interpretation; specification, algorithms and implementation to create digital solutions
- confidently use digital systems to efficiently and effectively automate the transformation of data into information and to creatively communicate ideas in a range of settings
- apply protocols and legal practices that support safe, ethical and respectful communications and collaboration with known and unknown audiences
- apply systems thinking to monitor, analyse, predict and shape the interactions within and between information systems and the impact of these systems on individuals, societies, economies and environments.

The Arduino UNO is a programmable board that is an ideal tool for taking students on a coding journey that can draw on or enhance the learning in multiple curriculum areas. There is a range of sensors compatible with the Arduino (and numerous other boards) that allow real world application without the need for high levels of coding skill.

The examples provided in this booklet are by no means exhaustive, but have been selected because they can lead to explorations and conversations in Science, Mathematics and the Social Sciences.

Coding is commonly regarded as falling under the umbrella of STEM and governments and policy makers have, in recent years, declared STEM to be a priority. STEM is generally accepted to encapsulate the combining of the four disciplines (Science, Technology, Engineering and Mathematics), but what is still unclear is whether the best approach to teaching STEM sees the four disciplines fully integrated, partially integrated or not integrated at all. There remain many who question the effectiveness of an integrated curriculum and posit that teaching an interdisciplinary or integrated curriculum can reduce the level of understanding of the core disciplines of STEM. The material in this handbook is not intended to argue the case for a particular STEM pedagogy, but rather is intended simply to provide pathways for student interaction with coding within the core subjects.

If we do not yet have universal clarity re how best to teach STEM, we might well ask why all students should be required to engage in coding and if they must, to what extent? I think it is essential for one to understand the basics of car’s engine, but I do not believe that all drivers need to be motor mechanics. In much the same way, given the technological revolution in which we live, an understanding of coding is essential, and a focus on both computational and critical thinking is highly desirable, but I do not think that all students (or teachers) need to be high level coders.

The exercises in this handbook aim to provide an introduction to coding and to demonstrate that when incorporated with a core body of knowledge, coding can help our students turn their focus to the creative use of the technology to tackle authentic problems and we as teachers can explore and gain greater understanding of “teaching for transfer” and how this is impacted by the age and cognitive development of the students.

Rob Sieben
Director, The Hartley institute
rsieben@hartley.edu.au

Table of Contents

Installing the Arduino Programming software	6
mBlock for junior and middle primary students	6
The UNO and accessories.....	6
Connecting the UNO to your computer	7
Some basic commands and syntax	7
The Setup function	8
The Loop function.....	8
Some basic coding rules	8
Digital versus Analogue	9
Printing to the screen	9
Getting started with the ‘blink’ example	10
Making an external LED blink	12
Creating the traffic light sequence	15
Introducing an ARRAY (An extension topic)	18
Using an array to make multiple LEDs light up in a sequence	20
Morse Code (An extension topic)	21
Introducing sensors and libraries	25
Using a sensor to drive decisions.....	25
Night light	25
Reading data from an analog sensor	26
Temperature and humidity.....	29
Soil moisture.....	32
Rewriting with an Array (An extension Activity)	35
Collecting data in a file.....	37
Creating proximity alarm – the speed of sound	39
The Internet of Things (IoT).....	45
Thinextra’s Xkit	46
Installing the Xkit on the UNO	46
Installing the Xkit libraries	47
Accessing the data via the Sigfox website (backend).....	48
Accessing the data via an automated system	49
Understanding the Xkit payload	51
Converting the payload into useable data for the classroom	52
Kobo Collect	54

Hexadecimal.....	57
Solutions to Challenges.....	61
mBlock – A drag and drop coding platform for younger students.....	65
Operators.....	76
Assignment operators	76
Comparative operators.....	76
Logical operators	76
IF statements	76
IF..ELSE statements.....	77
FOR statements	77
WHILE statements	77

Installing the Arduino Programming software

The software to control/program your Arduino (or Arduino compatible) board is free to download and run. It runs on Windows, Mac OS X, and Linux.

If you have a good and reliable internet connection, you might choose to run the online version of the software known as the Arduino Web Editor. Go to <https://auth.arduino.cc> to run the online editor.

If you prefer to install the software on a device so that it can be used without being online, download and install the appropriate version from <https://www.arduino.cc/en/Main/Software>

This software can be used with any Arduino compatible board.

mBlock for junior and middle primary students

For these students, rather than expecting students to type the code, a free coding program, mBlock, provides a 'drag and drop' solution. mBlock is based on Scratch, the open-source coding platform with which many schools are already familiar. Instructions re using mBlock are provided at the end of this handbook.



Figure 1 - The UNO

The UNO and accessories

The Arduino name can only be used on genuine Arduino such as Freeduino, Boarduino, DuinoTech etc. This handbook will simply refer to the 'UNO'. As the designs are open source, any clone board is 100% compatible with the Arduino and therefore any software, hardware, shields, etc. will also be 100% compatible.

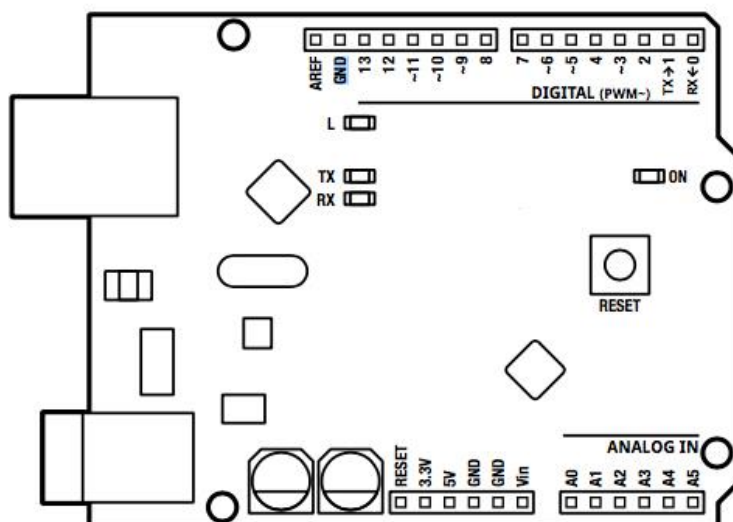


Figure 2 - Typical layout of an UNO

The UNO has three main banks of pins - digital, analog and power. The power bank pins are used to deliver 3.3 volts or 5 volts to devices, and the analog and digital pins are used to communicate with those devices.

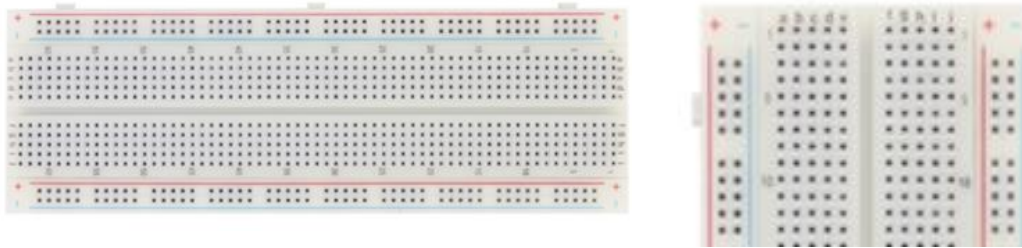


Figure 3 - A typical breadboard

A breadboard is the construction base for building elementary circuits. The breadboard is labelled at the end of each column and numbered at every fifth row. The outer columns labelled '+' and '-', and all the sockets in a column are connected. The internal columns are divided into two discrete blocks labelled 'a' to 'e' and 'f' to 'j' respectively, and the rows are numbered. All five sockets in the same *row* are connected (eg a5 is connected to e5, and f5 is connected to j5).

Connecting the UNO to your computer

Connect your UNO to your computer with an AB USB cable, sometimes called a USB printer cable. The UNO will automatically draw power from the computer using the USB cable. The power LED (labelled PWR or ON) should light up.

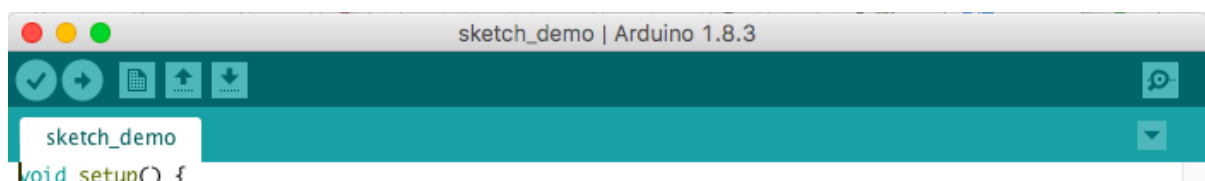
Some basic commands and syntax



ACARA element(s)

Managing and operating ICT

If you open the Arduino software (Arduino IDE), on the menu bar across the top of the Arduino window, you will see the following:



On the left-hand side of the toolbar, you can see the following icons:





is used to check your code (sketch) for errors.



is used to upload your sketch onto the Arduino.



is used to create a new tab for a new sketch.



is used to open an existing sketch.



is used to save the current sketch.

Every Arduino sketch consists of two essential parts or blocks of code. The first is the setup function and the second is the loop function.

The Setup function

The setup function contains the instructions that will run once when the UNO is powered up, and effectively prepares the Arduino environment to run the loop function.

The setup block of code begins with the statement “**void** setup()”. The void keyword indicates that the setup function is expected to return no information.

The various pins on the UNO may be used for input or output and it is within the setup() function that we define which is required. This happens in the setup section of the code/sketch.

The setup() function must be included even if there are no instructions to run.

The Loop function

The loop function contains the instructions that control the UNO and the devices connected to it. The loop function, as the name suggests, loops repeatedly as long as the UNO is powered and begins with the statement “**void** loop()”.

The loop function contains the instructions that tell the input and output pins what to do and under what circumstances to do it.

Some basic coding rules

Single lines are treated as comments if they begin with //.

Multiple lines are treated as a block comment if they are preceded by /* and followed by */.

An instruction concludes with a semi-colon.

Some of the basic commands that you will use in a loop function are digitalWrite, digitalRead, analogWrite, analogRead, Serial.print, Serial.println and delay. The following is a list of these commands and an explanation of the use in each case.


```
pinMode(pinX,OUTPUT); //is the command that states that PinX is to be used as an
                        // output.
digitalWrite(pinX, HIGH); // is the command used to turn on PinX.
digitalWrite(pinX, LOW);  // is the command used to turn off PinX.
digitalRead(pinX);        //reads the digital input being received by pinX
analogRead(pinX);         //reads the analog input being received by pinX
analogWrite(pinX,127);    // is the command used to turn on pinX and the 127 tells the UNO
                        // to use 50% of the available power. The number used to define the
                        // power must be between 0 and 255. 127 is half-way along the
                        // scale,
                        // hence the 50% power.
delay(1000);              //pauses the program for 1 second (1000 milliseconds).
Serial.print();           //prints the data from an input to the serial port of the computer.
Serial.println();         //prints the data from an input to the serial port of the computer,
                        //but on a new line.
```

Digital versus Analogue

Digital data consists only of 1s and 0s, whilst analogue data can be considered to be variable, with available values between 0 and 255.

When a pin is used, the data it sends or receives will be either digital or analogue and hence we have commands that tell the Arduino what sort of data to expect.

Hint: You will need to use the American spelling of ‘analog’ when writing code for the UNO

Printing to the screen

	ACARA element(s) Managing and operating ICT Communicating with ICT
--	---

Open a new sketch window. (File, New in the Arduino IDE window or use the appropriate icon on the toolbar)



Type the following.

```
void setup() {
  Serial.begin(9600);
}

void loop() {
  Serial.print("One,");
  delay(2000);
  Serial.print("Two,");
  delay(2000);
  Serial.println("Three,");
  delay(2000);
  Serial.println("Four");
  delay(2000);
  Serial.println();
}
```

Make sure your UNO is connected to your computer and use the “Upload” button to load the sketch onto your UNO.

To read the data as it is printed to the Serial port, click on the magnifying glass that appears on the right-hand side of the tool bar of the Arduino IDE software. This will open a window in which the values being written will appear.



The **Serial.print** instruction does not move to a new line after it has printed, whilst a **Serial.println** instruction moves the cursor to the next line after it has printed. In the example above, then, One, Two and Three will all appear on the same line, but after printing “Three,” the code instructs the cursor to move to the next line in preparation for the next command, where the word “Four” is printed. Again the cursor goes to a new line and this time, prints an empty line before returning to the beginning of the loop.

Getting started with the ‘blink’ example

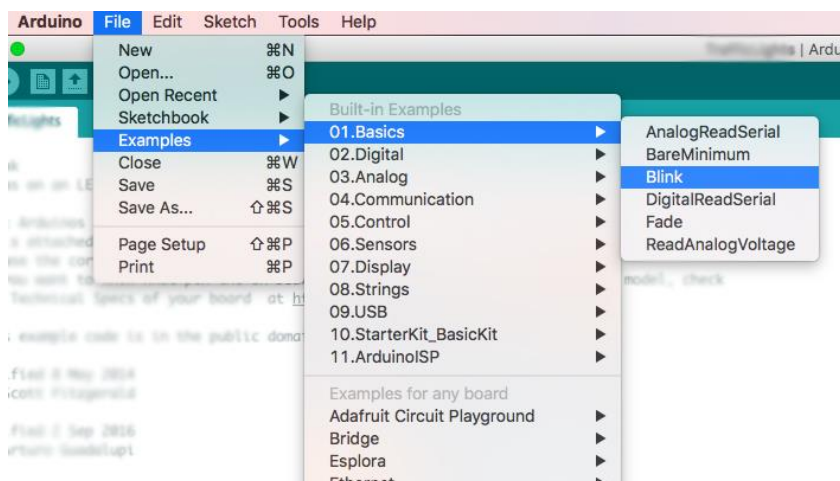


ACARA element(s)

Managing and operating ICT
Creating with ICT
Investigating with ICT

The Arduino IDE software refers to blocks of code as sketches. The software comes with several example sketches. ‘Blink’ is one such example.

Open the Arduino IDE software and click on ‘File’ and select ‘Examples’ from the drop-down menu and then select ‘Basics’.



You should now see a new window that contains a single tab viz:



If you maximise this window you will see that the following code appears.

```
// the setup function runs once when you press reset or power the board
void setup() {
  // initialize digital pin LED_BUILTIN as an output.
  pinMode(LED_BUILTIN, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {
  digitalWrite(LED_BUILTIN, HIGH); // turn the LED on by making the voltage
  HIGH                                //(HIGH is the voltage level)
  delay(1000);                        // wait for a second
  digitalWrite(LED_BUILTIN, LOW); // turn the LED off by making the voltage
  LOW
  delay(1000);                        // wait for a second
}
```

This code will make the built-in LED on the UNO blink on and off every second. You do not need to connect any external wires or LEDs to the UNO.

Make sure that your UNO is connected to your computer via the USB cable provided with the UNO.

Upload the Blink sketch to your UNO and you should see the built-in LED begin to blink every second.

Study the code and you will see that the author has provided a commentary with the commands to tell us what each line of code does.

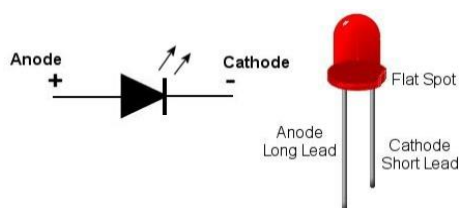
You can see that the setup “initialises the digital pin LED_BUILTIN as an output.

On the UNO, the LED_BUILTIN pin is connected to digital pin 13.

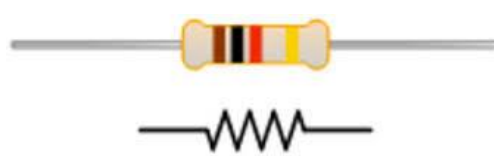
We can prove this, but before we do, we need to understand how an electric circuit and a typical LED works.

Current flows through a circuit in much the same way as water flows through a pipe. If there is no obstruction in the pipe the water will flow quickly. If we constrict or obstruct the pipe the water meets resistance and slows down. In an electrical circuit, we slow the current by putting resistance into the circuit in the form of appliances or resistors. The symbol for resistance is ‘ Ω ’ and it is measured in ohms.

LEDs (Light emitting diodes) only allow current to run in one direction. If you place them the wrong way around, they will not light up. The longer leg connects to the +ve side (anode) and the shorter to the –ve side (cathode). If your LED doesn’t light up try flipping it around. If the current flows too quickly an LED will burn out. An LED provides very little resistance, so to slow the current down we add resistance by putting a resistor into the circuit with the LED.



A typical LED beside the symbol used to represent an LED in a circuit diagram.



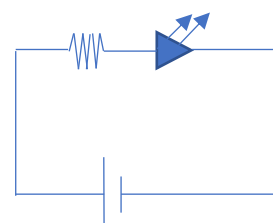
A typical resistor and the symbol used to represent a resistor in a circuit diagram.

Making an external LED blink

<p>Hardware requirements:</p> <p>UNO</p> <p>Breadboard</p> <p>1 x LED (any colour)</p> <p>1 x 220Ω resistor</p> <p>2 x wire connectors</p>		<p>ACARA element(s)</p> <p>Investigating with ICT</p> <p>Creating with ICT</p> <p>Managing and operating ICT</p> <p>Using measurement</p> <p>Analysing, synthesising and evaluating reasoning and procedures</p> <p>Reflecting on thinking and processes</p>
--	--	---

Take the LED and insert the longer leg into j20 on a breadboard and the shorter leg into J22 on the breadboard. Take the 220-ohm resistor and plug it into h20 and h15 on the breadboard. Connect f15 on the breadboard to d13 on the UNO and connect f22 to the adjacent GND pin.

If we illustrate this as an electrical circuit, then it looks as illustrated here. (The positive terminal is represented by the longer of the two lines.)



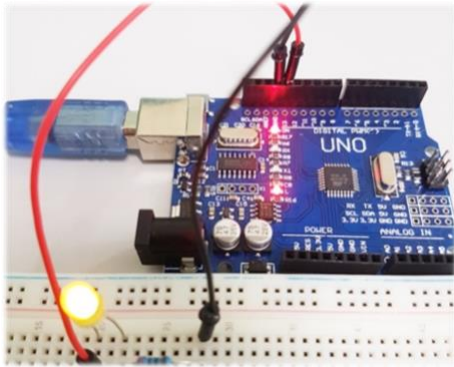


Figure 4 - Using the UNO to control an external LED

You will see that both the built-in LED and the LED plugged in to the UNO blink simultaneously.

Let's now remove reference to the built-in LED within the code and replace it with reference to digital pin 13.

We can do this by removing the reference to LED_BUILTIN and replacing it with the number 13.

```
pinMode(13, OUTPUT);
void loop() {
  digitalWrite(13, HIGH); // turn the LED on by making the voltage HIGH
  delay(1000);           // wait for a second
  digitalWrite(13, LOW);  // turn the LED off by making the voltage LOW
  delay(1000);           // wait for a second
}
```

Upload this to the UNO and note that the built-in LED and the LED in digital pin 13 still blink simultaneously.

This proves that digital pin 13 and the built-in LED are on the same connection.

Take the connection from digital pin 13 (d13) and connect it to digital pin 8 (d8) on the UNO.

Challenge 1:

What change or changes would you need to make to the code to make digital pin 8 blink rather than digital pin 13?

You can see that if the code we were using was longer and more complicated, having to search through multiple lines to replace one pin number with another would be tedious and the chance of missing one would be high, meaning that the code would fail.

We can get around this by introducing what we call a **variable**. Just as in **mathematics**, a variable is a name that we use to store a number.

The first thing we need to do is tell the UNO that we want to use a variable.

We do this by inserting a command *before* the setup() function. We can use whatever names we wish for our variables, but I suggest you choose a name that reflects its purpose eg:

```
int ledPin1 = 13; // tells the UNO that the name ledPin1 will be used to represent pin 13
or perhaps if using no more than one LED of any colour, something like:
int RedLED = 13; // tells the UNO that the Red LED will be using pin 13.
```

Next, we need to use the setup function to tell the UNO that ledPin1 is to be used as an output and within the loop() function we need to edit the instructions to make sure that the UNO reads and writes to ledPin1.

The code should now read:

```
int ledPin1 = 13;

void setup() {
  pinMode(ledPin1, OUTPUT);
}

void loop() {
  digitalWrite(ledPin1, HIGH); // turn the LED on by making the voltage HIGH
  delay(1000);                // wait for a second
  digitalWrite(ledPin1, LOW);  // turn the LED off by making the voltage LOW
  delay(1000);                // wait for a second
}
```





Upload this to the UNO and note that the built-in LED and the LED in pin 13 again blink simultaneously.

Save your modified code/sketch to a new name, using File, Save As rather than simply overwriting the original Blink sketch by using the save button.

Challenge 2

What change do you need to make to this code in order to make the LED connected to digital pin 8 blink rather than that connected to digital pin 13?

Creating the traffic light sequence

<p>Hardware requirements:</p> <p>UNO</p> <p>Breadboard</p> <p>1 x red LED</p> <p>1 x green LED</p> <p>1 x yellow LED</p> <p>3 x 220Ω resistor</p> <p>10 x wire connectors</p>	   	<p>ACARA element(s)</p> <p>Creating with ICT</p> <p>Managing and operating ICT</p> <p>Comprehending texts through listening, reading and viewing element</p> <p>Analysing, synthesising and evaluating reasoning and procedures</p> <p>Word knowledge element</p> <p>Reflecting on thinking and processes</p> <p>Inquiring – identifying, exploring and organising information and ideas element</p> <p>Self-management element</p>
---	---	--

In the last exercise, you modified the blink example sketch to introduce a variable (ledPin1) which you used to control a digital pin. Make sure that you have this modified sketch open.

Using the breadboard: -

Place a red LED into j1 and j3 with the longer leg in j1.

Place a yellow LED into j11 and j13 with the longer leg in j11.

Place a green LED into j21 and j23 with the longer leg in j21.

Place a 220-ohm resistor between h3 and h8.

Place a 220-ohm resistor between h13 and h18.

Place a 220-ohm resistor between h23 and h28.

Connect Digital pin 8 on the UNO to f1, digital pin 9 to f11 and digital pin 10 to f21.

Connect GND on the UNO to the -ve row, pin 1

Connect f8, f18 and f28 to the -ve row.

You should be able to trace a closed circuit from each digital pin across to the positive /longer leg of an LED, to the shorter leg of the LED and then through the resistor to the negative row of the breadboard and from there, back to the GND on the UNO.

If you have connected everything correctly, you might see the red LED blinking. If it is not blinking, look at the code to determine why it is not blinking. Does your code begin by defining the variable ledPin1 as 8? If not, edit it to be as follows:

```
int ledPin1 = 8;
```

The red LED connected to pin 8 will now be blinking. Neither of the other LEDs will be blinking.

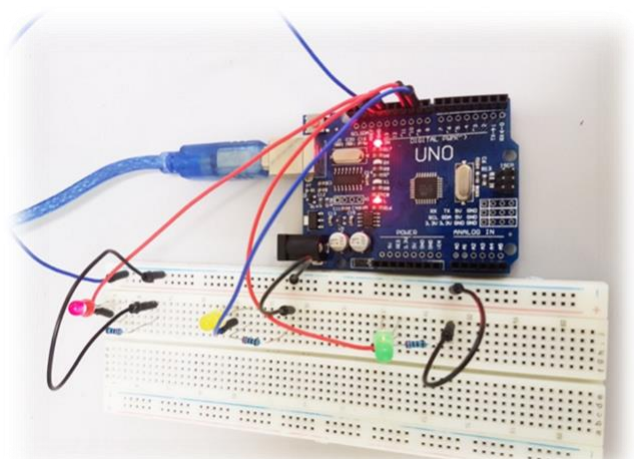


Figure 5 - Using the UNO to control multiple LEDs

Can you see how to modify the code to have the yellow LED connected to pin 9 also blink at the same time as the red LED blinks?

To do this, you must firstly tell the UNO that you intend to use digital pin 9 Then you must setup pin 9 as an output and finally you must insert the commands to turn digital pin 9 on and off.

The finished code will look like the following:

```
int ledPin1 = 8;
int ledPin2 = 9;

void setup() {
  pinMode(ledPin1, OUTPUT);
  pinMode(ledPin2, OUTPUT);
}

void loop() {
  digitalWrite(ledPin1, HIGH); // turn the ledPin1 on by making the voltage HIGH
  digitalWrite(ledPin2, HIGH); // turn the ledPin2 on by making the voltage HIGH
  delay(1000);                // wait for a second
  digitalWrite(ledPin1, LOW);  // turn the ledPin1 off by making the voltage LOW
  digitalWrite(ledPin2, LOW);  // turn the ledPin2 off by making the voltage LOW
  delay(1000);                // wait for a second
}
```

Challenge 3

Modify the code to have the LEDs in digital pins 8 and 9 alternate, rather than blink together.

Let's now add the third LED and modify the code to make the three LEDs come on one at a time, each for 5 seconds.

The code to do this will need to define three ledPins, declare each to be an output and then instruct the UNO to turn one on and the other two off in three different configurations.

The finished code will probably look like the following:

```
int ledPin1 = 8;
int ledPin2 = 9;
int ledPin3 = 10;

void setup() {
  pinMode(ledPin1, OUTPUT);
  pinMode(ledPin2, OUTPUT);
  pinMode(ledPin3, OUTPUT);
}
```



```

void loop() {
  digitalWrite(ledPin1, HIGH); // turn the ledPin1 on
  digitalWrite(ledPin2, LOW); // turn the ledPin2 off
  digitalWrite(ledPin3, LOW); // turn the ledPin3 on
  delay(5000);                // wait for 5 seconds
  digitalWrite(ledPin1, LOW); // turn the ledPin1 off
  digitalWrite(ledPin2, HIGH); // turn the ledPin2 on
  digitalWrite(ledPin3, LOW); // turn the ledPin3 off
  delay(5000);                // wait for 5 seconds
  digitalWrite(ledPin1, LOW); // turn the ledPin1 off
  digitalWrite(ledPin2, LOW); // turn the ledPin2 off
  digitalWrite(ledPin3, HIGH); // turn the ledPin3 on
  delay(5000);                // wait for 5 seconds
}

```

It is quite likely that students will generate an alternative solution, and that this solution will also work.

For example:

```

int ledPin1 = 8;
int ledPin2 = 9;
int ledPin3 = 10;

void setup() {
  pinMode(ledPin1, OUTPUT);
  pinMode(ledPin2, OUTPUT);
  pinMode(ledPin3, OUTPUT);
}

void loop() {
  digitalWrite(ledPin1, HIGH); // turn the ledPin1 on
  delay(5000);                // wait for 5 seconds
  digitalWrite(ledPin1, LOW); // turn the ledPin1 off
  digitalWrite(ledPin2, HIGH); // turn the ledPin2 on
  delay(5000);                // wait for 5 seconds
  digitalWrite(ledPin2, LOW); // turn the ledPin2 off
  digitalWrite(ledPin3, HIGH); // turn the ledPin3 on
  delay(5000);                // wait for 5 seconds
  digitalWrite(ledPin3, LOW); // turn the ledPin3 off
}

```

The fact that there are different solutions to the one problem should be used to illustrate that there is not necessarily only one way to solve a problem, and that the differences are really more a measure of the efficiency of the solution. What constitutes efficiency is itself a topic for conversation. Efficiency can be measured by the number of lines of code, the amount of processing needed, the power used by the device in processing the commands etc. One also needs to consider whether all possibilities have been accommodated. For example, in the above alternative solution, the second assumes that the LEDs are all in the OFF state when the code is initiated. That could, of course be addressed quite simply by inserting lines at the start setting all LEDs to OFF.





Challenge 4

What change would be needed to have the yellow LED come on again, one second *before* the green LED turns off and the red LED comes on?

What is the actual sequence that traffic lights follow? Can you modify your code to produce a simulation of a set of traffic lights?

There are several ways of assessing success with this. For primary aged children I would accept any solution that had the yellow light on for less time than the red and green lights, provided that order in which the sequence occurs is correct. For older students I would also be looking to see if the total time that the yellow and green lights are on is equal to the length of time that the red light is on. I would draw a simple intersection on the board and discuss what would happen if the sum of the yellow and green is greater than the time the red light is on.

Introducing an ARRAY (An extension topic)

Hardware requirements: UNO Breadboard 1 x red LED 1 x green LED 1 x yellow LED 3 x 220Ω resistor 10 x wire connectors	   	ACARA element(s) Creating with ICT Managing and operating ICT Comprehending texts through listening, reading and viewing element Analysing, synthesising and evaluating reasoning and procedures Reflecting on thinking and processes Inquiring – identifying, exploring and organising information and ideas element Self-management element
--	--	---

Sometimes we can avoid having to write multiple lines of code by using an array. An array is a collection of pieces of data that are all of the same type. The first element of an array is element number 0. The second element is element number 1 and so on. This means that an array of size 10, will have elements 0 to 9. We call the number of the element the INDEX.

An array is denoted by the use of square brackets – Array_name []

The elements of an array can be listed in curly brackets if they are numbers or between double quotes if they are alpha-numeric characters: -

`int` Array_name [4] = {2,4, 6, 8}; or `char` Array_name [4] = "ABCD";

In the traffic light example, we defined three LEDs and the digital pins to which they were connected. This took several lines of code. If we used an array, we could have replaced the following:

```
int ledPin1 = 8;
int ledPin2 = 9;
int ledPin3 = 10;
```

by the following:

```
int ledPin[3] = {8,9,10};
```

or if we use a variable to allow us to vary the number of LEDs,

```
const int numLEDs = 3; // sets the number of LEDs to a constant value of 3
int ledPin[numLEDs] = {8,9,10}; // defines the digital pins to be used as pins 8,9, and 10.
```

Whilst the second of these is neat, you would need to explain the use of “const” to set the number of LEDs as a constant integer because the use of the variable within another array demands that. In most classes, I would probably only use the first array to define the three pins (`int ledPin[3] = {8,9,10};`) and not over-complicate the issue.

Now the setup as follows:

```
void setup() {
  pinMode(ledPin1, OUTPUT);
  pinMode(ledPin2, OUTPUT);
  pinMode(ledPin3, OUTPUT);
}
```





can be changed to:

```
void setup() {
  for(int i = 0; i < 3; i++) {
    pinMode(ledPin[i], OUTPUT); // Defines the ledPin0 to be an OUTPUT
                                // and repeats for ledPINs 1 and 2.
  }
}
```

Challenge 5

What *other* change to the original code will now be needed? Hint: - what is the number/index of the first element of an array?

Using an array to make multiple LEDs light up in a sequence

Hardware requirements: UNO Breadboard 2 x red LED 2 x green LED 2 x yellow LED 6 x 220Ω resistor 14 x wire connectors	   	ACARA element(s) Creating with ICT Managing and operating ICT Comprehending texts through listening, reading and viewing element Analysing, synthesising and evaluating reasoning and procedures Reflecting on thinking and processes Inquiring – identifying, exploring and organising information and ideas element Self-management element
--	--	---

The aim in this activity is to program the Arduino to use an array to turn on six LEDs one after the other. In this example, the pins will light up in a sequence and then light up again in the reverse of the initial sequence. The apparent “randomness” of the pattern is controlled by the order in which the pins are defined in that array.

```

int timer = 100;          // Sets “timer” as a variable to control the timing of the
                          // sequence. The higher the number, the slower the timing.
int ledPins[] = { 2, 7, 4, 6, 5, 3 }; // an array of pin numbers to which LEDs are
                                      // attached

int pinCount = 6;        // the number of pins (i.e. the length of the array)





void setup()
{
  for (int i = 0; i < pinCount; i++)
  {
    pinMode(ledPins[i], OUTPUT);
  }
}

void loop() {
  // loop from the lowest pin to the highest:
  for (int i = 0; i < pinCount; i++)
  {
    digitalWrite(ledPins[i], HIGH);
    delay(timer);
    digitalWrite(ledPins[i], LOW);
  }

  // loop from the highest pin to the lowest:
  for (int i = pinCount - 1; i >= 0; i--) {
    digitalWrite(ledPins[i], HIGH);
    delay(timer);
    digitalWrite(ledPins[i], LOW);
  }
}

```

Morse Code (An extension topic)

<p>Hardware requirements:</p> <p>UNO</p> <p>Breadboard</p> <p>1 x yellow LED</p> <p>1 x 220Ω resistor</p> <p>2 x wire connectors</p>	   	<p>ACARA element(s)</p> <p>Creating with ICT</p> <p>Managing and operating ICT</p> <p>Comprehending texts through listening, reading and viewing element</p> <p>Analysing, synthesising and evaluating reasoning and procedures</p> <p>Reflecting on thinking and processes</p> <p>Inquiring – identifying, exploring and organising information and ideas element</p> <p>Self-management element</p>
--	---	--

Morse code was used in the days before voice communication to transmit messages via a series of ‘dots’ and ‘dashes’. Your goal is to create an SOS light flashing sequence using the built in LED on the UNO.

*Create a New Sketch, but before you start writing code, use File / Save As this project as **BlinkingSOS***

About Morse Code

Adapted from <http://www.nu-ware.com>

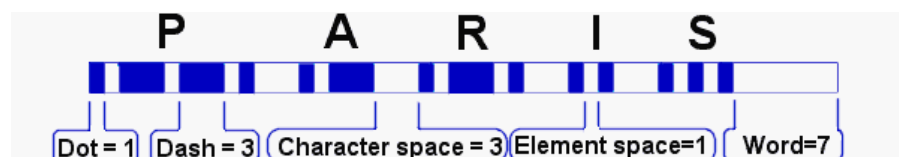


Figure 6 - Visual representation of Morse Code.

The basic element of Morse code is the dot and all other elements can be defined in terms of multiples of the dot length. The word Paris above shows the duration of each element.

Here are the ratios for the other code elements:

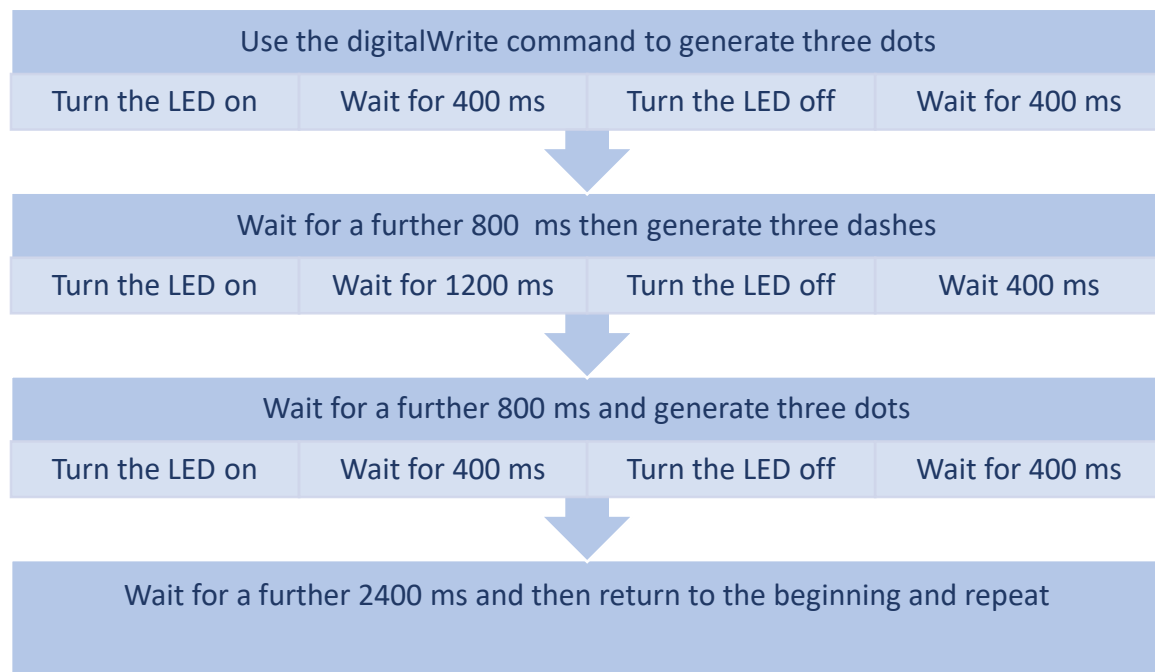
Dash length	Dot length x 3
Element Space (Pause between elements)	Dot length
Space (Pause between characters)	Dot length x 3
Pause between words	Dot length x 7

An SOS sequence looks like this: ● ● ● — — — ● ● ●

Use the following tables as a reference to help determine the actual code required.

	Morse Code Element	Duration (ms)	What needs to happen – Pseudocode
Letter S (Three dots)	Dot	400	LED on, delay 400, LED off
	Element Space (same as dot length)	400	Delay 400
Space	Space (Dot length x3)	1200 (400x3)	Delay 1200
Letter O (Three dashes)	Dash (Dot length x3)	1200	LED on, delay 1200, LED off
End of word	Pause between words (Dot length x7)	2800 (400x7)	

Sometimes it helps to write the actions that you need to code in words in order to get a clear understanding of the process.



The code to do this will resemble the following.

```
void setup()
```

```
    // initialize digital pin LED_BUILTIN as an output.
```

```
    pinMode(LED_BUILTIN, OUTPUT);
```

```
}
```

```
void loop() {
```

```
    digitalWrite(LED_BUILTIN, HIGH); // turn the LED on for 400 ms (dot)
```

```
    delay(400);
```

```
    digitalWrite(LED_BUILTIN, LOW); // turn the LED off for 400 ms
```

```
    delay(400);
```

```
    digitalWrite(LED_BUILTIN, HIGH); // turn the LED on for 400 ms (dot)
```

```
    delay(400)
```

```
    digitalWrite(LED_BUILTIN, LOW); // turn the LED off for 400 ms
```

```
    delay(400);
```

```
    digitalWrite(LED_BUILTIN, HIGH); // turn the LED on for 400 ms (dot)
```

```
    delay(400);
```

```
    digitalWrite(LED_BUILTIN, LOW); // turn the LED off for 1200 ms (pause between letters)
```

```
    delay(1200);
```

```
    digitalWrite(LED_BUILTIN, HIGH); // turn the LED on for 1200 ms (dash)
```

```
    delay(1200);
```

```
    digitalWrite(LED_BUILTIN, LOW); // turn the LED off for 400 ms
```

```
    delay(400);
```

```
    digitalWrite(LED_BUILTIN, HIGH); // turn the LED on for 1200 ms (dash)
```

```
    delay(1200);
```

```
    digitalWrite(LED_BUILTIN, LOW); // turn the LED off for 400 ms
```

```
    delay(400);
```

```
    digitalWrite(LED_BUILTIN, HIGH); // turn the LED on for 1200 ms (dash)
```

```
    delay(1200);
```

```
    digitalWrite(LED_BUILTIN, LOW); // turn the LED off for 1200 ms (pause between letters)
```

```
    delay(1200);
```

```
    digitalWrite(LED_BUILTIN, HIGH); // turn the LED on for 400 ms (dot)
```

```
    delay(400);
```

```
    digitalWrite(LED_BUILTIN, LOW); // turn the LED off for 400 ms
```

```
    delay(400);
```

```
    digitalWrite(LED_BUILTIN, HIGH); // turn the LED on for 400 ms (dot)
```

```
    delay(400);
```

```
    digitalWrite(LED_BUILTIN, LOW); // turn the LED off for 400 ms
```

```
    delay(400);
```

```
    digitalWrite(LED_BUILTIN, HIGH); // turn the LED on for 400 ms (dot)
```

```
    delay(400);
```

```
    digitalWrite(LED_BUILTIN, LOW); // turn the LED off for 2800 ms (pause between words)
```

```
    delay(2800);
```

```
}
```

Challenge 6

Can you re-write this code to use a number of “FOR” statements to eliminate the number of repeated lines?

Challenge 7

Can you create a Morse Code sequence using a LED to transmit your initials and date of birth (ddmm) at the end of the SOS sequence completed previously.

eg. If your name is John Smith, birthday 16th April, your whole sequence would be: SOS JS 1604

Use the following legend to assist you.

International Morse Code

1. The length of a dot is one unit.
2. A dash is three units.
3. The space between parts of the same letter is one unit.
4. The space between letters is three units.
5. The space between words is seven units.

A • —
B — • • •
C — • — •
D — • •
E •
F • • — •
G — — •
H • • • •
I • •
J • — — —

K — • —
L — — • •
M — —
N — •
O — — —
P • — — •
Q — — • —
R • — •
S • • •
T —

U • • —
V • • • —
W • — —
X — • • —
Y — • — —
Z — — • •

1 • — — — —
2 • • — — —
3 • • • — —
4 • • • • —
5 • • • • •
6 — • • • •
7 — — • • •
8 — — — • •
9 — — — — •
0 — — — — —

You might now try and substitute an alarm/buzzer in place of the LED and see if you can transmit the same Morse Code using sound rather than flashing lights.

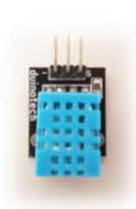
See how well your students can listen to the code and write it down as they hear it.

Introducing sensors and libraries

The real power of microcontrollers like the UNO is in their capacity to use sensors. There is a large range of sensors available for the UNO and most cost only a few dollars. In this handbook, you will use sensors that measure light, temperature, humidity, moisture and barometric pressure.

Most developers, when producing a complex sensor, also provide the code necessary to operate that sensor so that the end-user does not need to re-invent the wheel. Where this is the case, the developer provides their code(s) in a library and all the end-user needs to do, is insert the library into their sketch and then use the commands provided with the library to call the code that operates the sensor. This makes it possible for teachers and students to use sophisticated sensors with only a basic knowledge of coding.

It is the data that we can collect using these sensors that provides the opportunities to integrate with subjects across the curriculum. Analysis of data falls into the Mathematics and Science domains, whilst decisions based on the data provides opportunities for critical thinking in areas like the Social Sciences. In the following chapters we explore how some of the readily available sensors can be used in this way.



Temperature and Humidity sensor



Moisture sensor

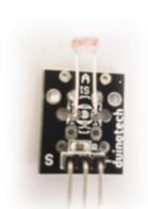







Photo sensor
(Light Dependent Resistor)

Using a sensor to drive decisions

Night light

Hardware requirements: UNO Breadboard 1 x LED (any colour) 1 x photo sensor (LDR) 1 x 220Ω resistor 1 x 470Ω resistor 7 x wire connectors	    	ACARA element(s) Creating with ICT Managing and operating ICT Comprehending texts through listening, reading and viewing Analysing, synthesising and evaluating reasoning and procedures Reflecting on thinking and processes Inquiring – identifying, exploring and organising information and ideas Self-management Using measurement
--	---	--

In this example, we use a photo sensor to control an LED which will turn on when the light level drops below a designated level.

To do this, we need two circuits, one managing the photo sensor and a second controlling the LED.

I have provided two examples here, as the photo sensor might be a stand-alone sensor, or it might be on a small board with a built-in resistor.

Using individual components -

Place a photo sensor (light dependent resistor) into j1 and j5 on the bread board

Connect f1 to the +ve column.

Place the 470-ohm resistor across h5 and h10 and connect h10 to the -ve column.

Connect f5 to A0 on the UNO.

Put the LED into j20 and j22 with the longer leg into j20.

Place the 220-ohm resistor across h20 to h25.

Connect f22 to the -ve column and connect f25 to D8 on the UNO.

Connect the 5V on the UNO to the +ve Column and the -ve Column to GND on the UNO.

This completes both required circuits.

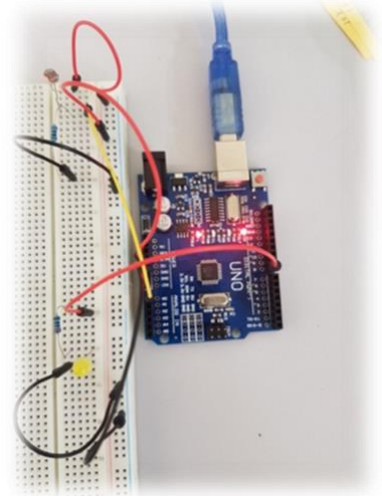


Figure 7 - photo sensor and separate resistor

Using a photo sensor with built-in resistor –

Connect the positive leg of the sensor to the 5V pin on the UNO.

Connect the negative leg of the sensor to a GND pin on the UNO.

Connect the Signal leg (S) to A0 on the UNO.

Put the LED into j20 and j22 with the longer leg into j20.

Place the 220-ohm resistor across h20 to h25.

Connect f22 to GND on the UNO and connect f25 to D8 on the UNO.

This completes both required circuits.

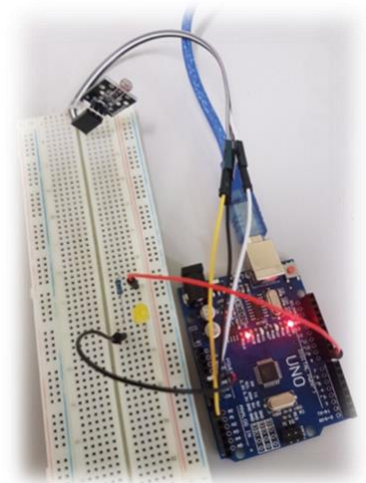


Figure 8 - photo sensor with built-in resistor

The first thing we need to do is write the code to manage the sensor.

We do this by telling the UNO that a sensor will be used and into which pin it will be connected. Because the photo sensor is an analogue device, we will connect it to analog pin 0 (A0). The code to do this is :

```
int sensorPin = 0
```

Reading data from an analog sensor

We will read the data coming in to the pin A0 but in order to read the data we need to instruct the UNO to print the data to the screen. We do this by using the serial port of the computer.

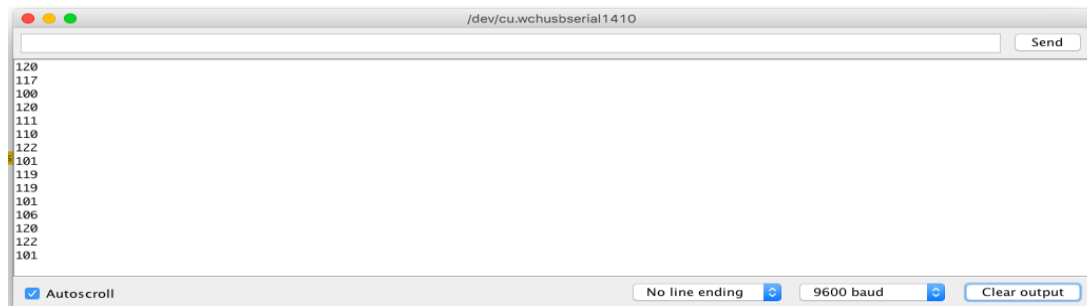
The code required is:

```
int sensorPin = 0;

void setup()
{
  Serial.begin(9600); // initialize serial communication at 9600 bits per second
}

void loop()
{
  // read the input on analog pin 0:
  // and print out the value you read
  Serial.println(analogRead(sensorPin));
  delay(2000);
}
```

Remember that to read the data as it is printed to the Serial port, you click on the magnifying glass that appears in the tool bar of the Arduino IDE software. This will open a window in which the values being written will appear.



Put your finger over the photo sensor and note the change in the numbers that appear.

As you cover the sensor, the numbers on the screen will either decrease or increase, depending on whether or not the sensor has a built-in resistor or not and how it is connected to that resistor. You will need to cover and uncover the sensor and note the range of numbers and how they reflect the conditions.

In the example used, the sensor produced numbers that decreased as the light level decreased, and whilst it was possible to get the reading to zero, it was dark enough to want to turn on a light when the reading dropped to under 300.

We can use these changes in the sensor's output to control our LED. We do this by telling the UNO that we are using an LED connected to digital pin 8 and that this digital pin will be an output. We can then instruct the UNO to turn the LED on or off based on the values that the photo sensor is providing.

The code to do this might be as follows:

```
int ledPin = 8;

void setup()
{
  pinMode(ledPin,OUTPUT);
}

void loop()
{
  int val = analogRead(sensorPin);
  if (val < 300)
    digitalWrite(ledPin,HIGH);
  else
    digitalWrite(ledPin,LOW);
  delay(2000);    // waits for 2 seconds before reading the photo sensor again
}
```

Combine the code to control the LED with the code used to control the sensor to produce:

```
int sensorPin = 0;
int ledPin = 8;

void setup()
{
  // initialize serial communication at 9600 bits per second:
  Serial.begin(9600);
  pinMode(ledPin,OUTPUT);
}

void loop()
{
  // read the input on analog pin 0 and print out the value you read:
  Serial.println(analogRead(sensorPin));
  int val = analogRead(sensorPin);
  if (val < 300)
    digitalWrite(ledPin,HIGH);
  else
    digitalWrite(ledPin,LOW);
  delay(2000);    // waits for 2 seconds before reading the photo sensor again
}
```






Upload this code to the UNO and test whether or not the LED comes on when you cover the photo sensor and whether the LED goes out when the sensor is exposed to light. You have created a basic night light.

Just as we have used a photo sensor to turn a light on when the natural light source drops below a set level, we could have used a motion sensor, and then could turn the light on when movement is detected.

Challenge 8

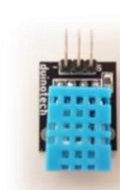
If we wanted the LED to remain on for five minutes after sensing movement, what delay would we need to use between readings of the sensor? Where would you put the delay command to keep the light on, and why?

Temperature and humidity

<p>Hardware requirements:</p> <p>UNO</p> <p>Breadboard</p> <p>1 x temperature and humidity sensor (DHT11)</p> <p>1 x 3 wire extension cable</p> <p>6 x wire connectors</p>	    	<p>ACARA element(s)</p> <p>Creating with ICT</p> <p>Managing and operating ICT</p> <p>Comprehending texts through listening, reading and viewing</p> <p>Analysing, synthesising and evaluating reasoning and procedures</p> <p>Inquiring – identifying, exploring and organising information and ideas</p> <p>Self-management</p> <p>Using measurement</p> <p>Using fractions, decimals, percentages, ratios and rates</p>
--	---	---

In this example we will use a DHT11 sensor which measures both temperature and humidity.

The developers provide a library to control this sensor, so we need to download that library, install it into the Arduino IDE software and then include it in our code.

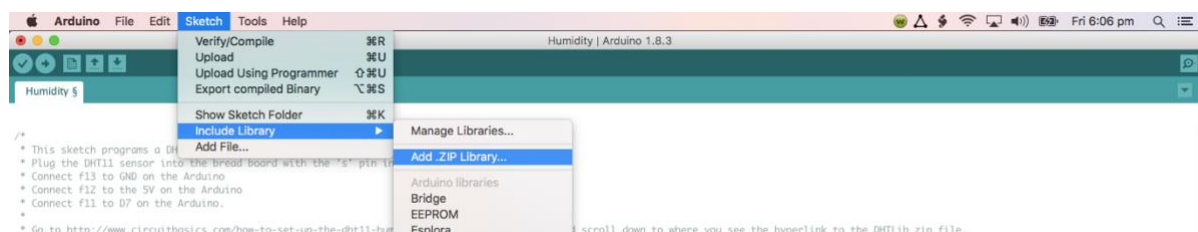


Go to <http://www.circuitbasics.com/how-to-set-up-the-dht11-humidity-sensor-on-an-arduino/> and scroll down to where you see the hyperlink to the DHTLib zip file.

Download and save this zip file to your desktop.

In the Arduino IDE program, select Sketch, Include Library, Add .ZIP library and browse to where you save the DHTLib.zip file. Select the zip file and this will add the DHT Library to the available libraries for your UNO.

The DHT11 has four pins, but is usually mounted onto a small board that uses three pins. One is labelled ‘-’ and is therefore, the negative pin. The middle pin is the positive pin and the remaining pin is labelled ‘s’ and this is the ‘signal’ or data pin.



Use the three-wire extension cable and three connection wires to plug the DHT11 sensor into the bread board with the 's' pin in i11, the +ve pin in i12 and the -ve pin in i13.

Connect f13 to GND on the UNO
 Connect f12 to the 5V on the UNO
 Connect f11 to D7 on the UNO.

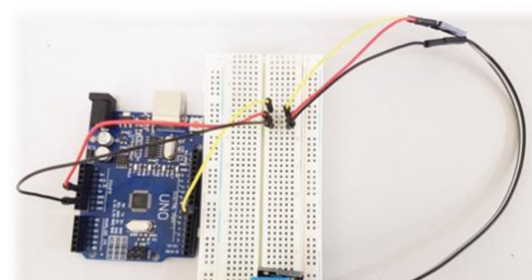


Figure 9 – The DHT11 records temperature and humidity

In this example, the code begins by calling two libraries, one to allow us to measure time and one that controls the sensor.

To do this, in the Arduino IDE program, select Sketch, Include Library and then choose the Time library from the list of available libraries. This will insert two lines into your sketch: -

```
#include <Time.h>
#include <TimeLib.h>
```

Repeat and this time select the library DHTLib which will insert the following command into your sketch: -

```
#include <dht.h>
```

The next thing we need to do is call or run the DHT library of commands, using the instruction: -

```
dht DHT;
```

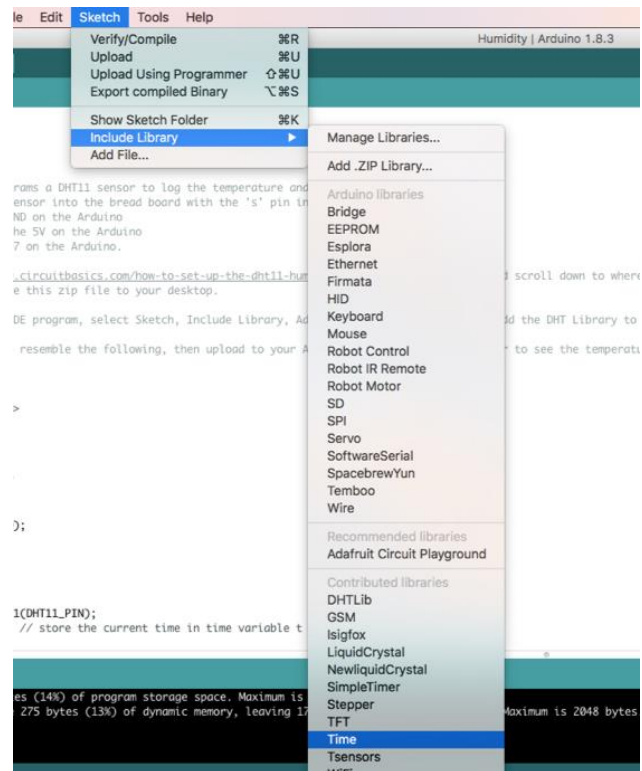
The next thing is to define the pin that will be used as the input to receive the data from the sensor. In this example I have used digital pin 7 on the Arduino: -

```
#define DHT11_PIN 7
```

The Setup() function contains only the one command needed to open the communication between the UNO and the serial port on the computer: -

```
void setup()
{
  Serial.begin(9600);
}
```

The loop() function reads the data coming in to digital pin 7 and prints it to the serial port so that we can display it on the screen. So that we do not have just a string of numbers that we cannot interpret, we also include instructions that print words “Temperature”, “Humidity” and “Time” to identify the data.



The complete sketch is as follows: -

```
#include <Time.h>
#include <TimeLib.h>
#include <dht.h>

dht DHT;

#define DHT11_PIN 7

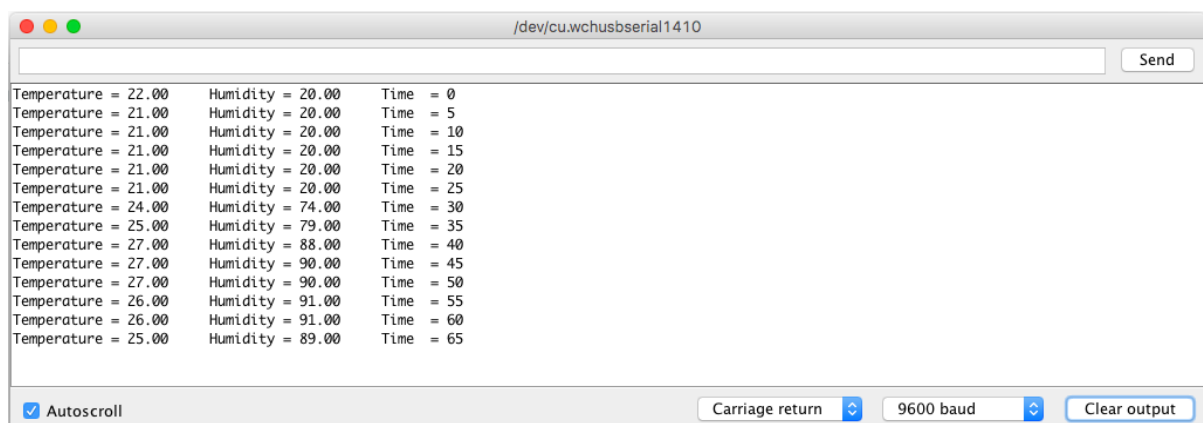
void setup()
{
  Serial.begin(9600);
}

void loop()
{
  int chk = DHT.read11(DHT11_PIN);
  int time_t = now(); // store the current time in time variable t
  Serial.print("Temperature = "); //prints the word "Temperature"
  Serial.print(DHT.temperature); // prints the actual temperature
  Serial.print(" Humidity = "); // prints the word "Humidity"
  Serial.print(DHT.humidity); // prints the actual humidity
  Serial.print(" Time = "); // prints the word "Time"
  Serial.print(time_t); // prints the time that the sensor has been powered
  Serial.println(""); // forces a new line
  delay(5000);
}
```

Click on the magnifying glass that appears in the tool bar of the Arduino IDE software to open the window in which the values being written will appear.








The results of each reading of the sensor will appear as one line of data.



Using this sensor, we can explore a number of factors that impact the temperature and/or humidity. For example, placing the sensor on the floor and then raising it to the ceiling will produce a variation in temperature. Breathing into the sensor will change the humidity. Discuss why this might be the case.

Soil moisture

<p>Hardware requirements:</p> <p>UNO</p> <p>Breadboard</p> <p>1 x red LED</p> <p>1 x green LED</p> <p>1 x yellow LED</p> <p>3 x 220Ω resistor</p> <p>1 x Soil Moisture sensor</p> <p>1 x 3 wire extension cable</p> <p>10 x wire connectors</p>	    	<p>ACARA element(s)</p> <p>Creating with ICT</p> <p>Managing and operating ICT</p> <p>Comprehending texts through listening, reading and viewing</p> <p>Analysing, synthesising and evaluating reasoning and procedures</p> <p>Reflecting on thinking and processes</p> <p>Inquiring – identifying, exploring and organising information and ideas</p> <p>Self-management</p> <p>Using measurement</p> <p>Using fractions, decimals, percentages, ratios and rates</p>
---	---	---

This sketch uses three LEDs to indicate the moisture level in soil.

Place a red LED into j1 and j3 with the longer leg in j1.

Place a yellow LED into j11 and j13 with the longer leg in j11.

Place a green LED into j21 and j23 with the longer leg in j21.

Place a 220-ohm resistor between h3 and h8.

Place a 220-ohm resistor between h13 and h18.

Place a 220-ohm resistor between h23 and h28.

Connect Digital pin 8 on the UNO to f1, digital pin 9 to f11 and digital pin 10 to f21.

Connect h8, h18 and h28 to the -ve column on the UNO and then connect the -ve column to GND on the UNO.

Connect the three pins of the soil moisture sensor to the UNO. The +ve pin to 5V, the -ve pin to GND and the signal pin to A0 on the UNO.

This completes the circuit.

Plug the sensor into a container with some soil. A small bucket or ice-cream container is ideal.

The moisture sensor is an analog device, so we need to define the analog pin that we will use to receive the data from the sensor, and the three digital pins that will be used to control the LEDs.

The code to do this is: -

```
int sensorPin = A0;
int ledPin1 = 8;
int ledPin2 = 9;
int ledPin3 = 10;
```

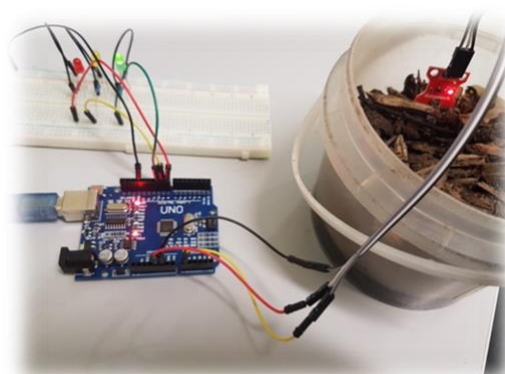


Figure 10 - using soil moisture to control LEDs

Next, the setup() function is used to define whether the pins are to be used as inputs or outputs.

```
void setup() {  
  pinMode(ledPin1, OUTPUT);  
  pinMode(ledPin2, OUTPUT);  
  pinMode(ledPin3, OUTPUT);  
  pinMode(sensorPin, INPUT);  
  Serial.begin(9600);  
}
```

The next step is to read the sensor in different moisture conditions.

```
void loop() {  
  int val = analogRead(sensorPin);  
  Serial.println(val);  
  delay(2000);  
}
```

The complete sketch should now read: -

```
int sensorPin = A0;  
int ledPin1 = 8;  
int ledPin2 = 9;  
int ledPin3 = 10;  
  
void setup()  
  pinMode(ledPin1, OUTPUT);  
  pinMode(ledPin2, OUTPUT);  
  pinMode(ledPin3, OUTPUT);  
  pinMode(sensorPin, INPUT);  
  Serial.begin(9600);  
}  
  
void loop() {  
  int val = analogRead(sensorPin);  
  Serial.println(val);  
  delay(2000);  
}
```

Click on the magnifying glass that appears in the tool bar of the Arduino IDE software to open the window in which the values being written will appear.



Note the value when the soil is dry. Slowly add some water near the sensor and take note of the change in the value as the soil changes from dry to damp to wet.

In the example provided, when the soil was dry, the reading was below 25. When damp, the reading was between 25 and 250, and when wet the reading was over 250.

By plugging these values into the code, it is possible to have different LEDs come on to indicate the degree of moisture in the soil.

The Loop() function will become: -

```
void loop() {  
  int val = analogRead(sensorPin);  
  Serial.println(val);  
  delay(2000);  
  
  // Configure LEDs to indicate moisture levels.  
  
  if (val < 25)  
  {  
    digitalWrite(ledPin1,HIGH);  
    digitalWrite(ledPin2,LOW);  
    digitalWrite(ledPin3,LOW);  
  }  
  else  
  if (val > 25 && val < 250)  
  {  
    digitalWrite(ledPin1,LOW);  
    digitalWrite(ledPin2,HIGH);  
    digitalWrite(ledPin3,LOW);  
  }  
  else  
  // if (val > 250)  
  {  
    digitalWrite(ledPin1,LOW);  
    digitalWrite(ledPin2,LOW);  
    digitalWrite(ledPin3,HIGH);  
  }  
}
```

When the sensor detects moisture levels the UNO will light up the appropriate LED: -

Red = dry, Amber = damp and Green = wet.

Challenge 9

Why is the statement 'if (val > 250)' commented out of the code?

Again, it is worth noting that there are multiple ways of coding this activity.

We could achieve the same result without using If...else, but rather just using three if statements.

Do we need to include three digitalWrite instructions in each block of the code?






Consider the following variation for the Loop() function.

```
void loop() {
  int val = analogRead(sensorPin);
  Serial.println(val);

  digitalWrite(ledPin1,LOW); // set all three LEDs to the OFF position
  digitalWrite(ledPin2,LOW);
  digitalWrite(ledPin3,LOW);
  // Configure LEDs to indicate moisture levels.

  if (val < 25)
  {
    digitalWrite(ledPin1,HIGH);
  }
  else
  if (val > 25 && val < 250)
  {
    digitalWrite(ledPin2,HIGH);
  }
  else
  {
    digitalWrite(ledPin3,HIGH);
  }
  delay(2000);
}
```

Rewriting with an Array (An extension Activity)

<p>Hardware requirements:</p> <p>UNO</p> <p>Breadboard</p> <p>1 x red LED</p> <p>1 x green LED</p> <p>1 x yellow LED</p> <p>3 x 220Ω resistor</p> <p>1 x Soil Moisture sensor</p> <p>1 x 3 wire extension cable</p> <p>10 x wire connectors</p>	    	<ul style="list-style-type: none"> • ACARA element(s) <p>Managing and operating ICT</p> <p>Comprehending texts through listening, reading and viewing</p> <p>Analysing, synthesising and evaluating reasoning and procedures</p> <p>Reflecting on thinking and processes</p> <p>Inquiring – identifying, exploring and organising information and ideas</p> <p>Sustainability - participating critically and acting creatively in determining more sustainable ways of living.</p>
---	---	---

Can you see how you might be able to rewrite the entire sketch using an array to control the digital pins and/or LEDs?

This block of code could be replaced with:

```
int sensorPin = A0;
int ledPin[3] = {8,9,10};

void setup() {

  for(int i = 0; i < 3; i++) {
    pinMode(ledPin[i], OUTPUT); // defines each of the digital pins to be an OUTPUT
    pinMode(sensorPin, INPUT);
    Serial.begin(9600);
  }
}






void loop() {
  int val = analogRead(sensorPin);
  Serial.println(val);

  for(int i = 0; i < 3; i++) {
    digitalWrite(ledPin[i], LOW); // set all three LEDs to the OFF position
  }
  // Configure LEDs to indicate moisture levels.
  if (val < 25)
  {
    digitalWrite(ledPin[0],HIGH);
  }
  else
  if (val > 25 && val < 250) // note the use of && for 'and'
  {
    digitalWrite(ledPin[1],HIGH);
  }
  else
  {
    digitalWrite(ledPin[2],HIGH);
  }
  delay(2000);
}
```

If we can program LEDs to turn on and off based on the sensor readings, then we could clearly use this data to make other decisions, like whether or not to water gardens or irrigate fields based on the moisture level in the soil rather than the more traditional ‘day of the week’ model, thereby establishing a link between coding and environmental sustainability.

Senior students might investigate coding the UNO to use a bridge to switch a solenoid on and off based on soil moisture levels.

Collecting data in a file

Hardware requirements: UNO Breadboard 1 x temperature and humidity sensor (DHT11) 1 x 3 wire extension cable 6 x wire connectors	    	<ul style="list-style-type: none"> ACARA element(s) Managing and operating ICT Analysing, synthesising and evaluating reasoning and procedures Inquiring – identifying, exploring and organising information and ideas Interpreting statistical information Sustainability - participating critically and acting creatively in determining more sustainable ways of living.
---	---	---

Sometimes we want to collect the data for analyses rather than simply control a device like an LED. One simple way to do this is to collect the data as it is written to the serial port.

Open a web browser and visit the following web site.

<https://learn.sparkfun.com/tutorials/terminal-basics/coolterm-windows-mac-linux>

Here you will find a link to download a software program called ‘CoolTerm’.

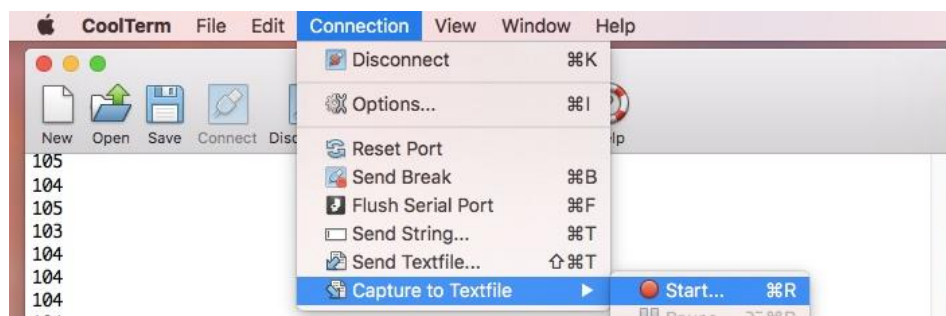
CoolTerm has versions for any operating system. Install and open CoolTerm.

Click on the Options icon and if the Serial Port displayed is not correct, select the correct option and click OK. Click the File, Save as Default option to save the configuration for next time.

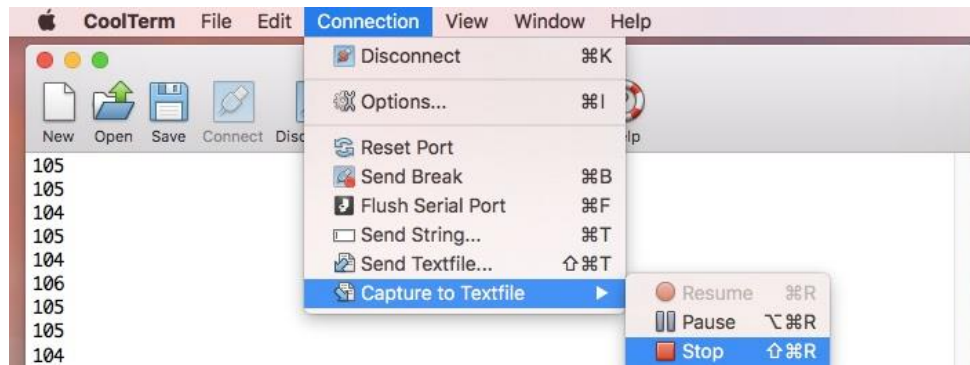
Make sure that the UNO is connected to the computer and upload a sketch that writes to the serial port. The temperature and humidity sketch is a good example.

Click in CoolTerm and open a connection by clicking Connect on the toolbar. You should now see the data that is being printed to the serial port appear in the CoolTerm window.

Click on the Connection option and choose Capture to text file and Start.



The data being written to the serial port will now be captured by CoolTerm until you stop it. Do this by repeating the “Capture to Textfile” process and choosing Stop rather than Start.








You can now use a text editing program or a spreadsheet program like Microsoft Excel or Numbers to open the text file that has been saved on your computer's desktop.

This is a good opportunity to link into the Mathematics curriculum through the analysis of the data collected by the sensor(s). Using 'real data' we can explore mathematical concepts like maximum, minimum, average, median, mode, standard deviation etc., depending on the age and year level of the students. The same data can be used to develop the skills needed to generate and interpret graphical data.

Similarly, the data collected can be used to initiate conversations around topics in Science. For example, as suggested earlier, when using a temperature sensor to measure the temperature in different locations, the conversation might lead to an exploration of the difference in temperature at floor level with that nearer to the ceiling in a classroom. This in turn should lead to an understanding that hot air rises (and cool air falls) and potentially why this might be the case.

Creating proximity alarm – the speed of sound

Hardware requirements: UNO Breadboard 1 x HC-SR04 ultrasonic distance sensor 1 x Buzzer 6 x wire connectors	    	ACARA element(s) Creating with ICT Managing and operating ICT Comprehending texts through listening, reading and viewing Analysing, synthesising and evaluating reasoning and procedures Reflecting on thinking and processes Inquiring – identifying, exploring and organising information and ideas Using measurement Using fractions, decimals, percentages, ratios and rates
--	---	---

This activity can be treated as simply creating a proximity alarm, along the lines of employed in modern automobiles to alert drivers when sensors in the front or rear bumper bars approach obstructions.

The real power of the activity, however, is the capacity to use the sensor to explore how the speed of sound varies according to atmospheric/climatic conditions.

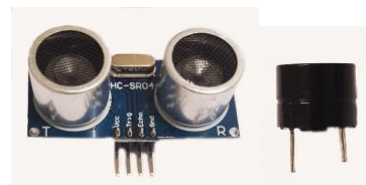


Figure 11 - Ultrasonic distance sensor and active buzzer

On the front of the ultrasonic range finder are two metal cylinders. These are transducers that generate or receive electrical signals. In the ultrasonic range finder, there is a transmitting transducer and receiving transducer. The transmitting transducer converts an electrical signal into the ultrasonic pulse, and the receiving transducer converts the reflected ultrasonic pulse back into an electrical signal.

The ultrasonic pulse transmitted by the sensor rebounds when it strikes a solid barrier and is then received by the sensor. Using the simple relationship between speed and time, we can then calculate the distance travelled by the ultrasonic pulse.

$$\text{Distance} = \text{speed} \times \text{time}$$

The ultrasonic pulse is a sound pulse and therefore travels at the speed of sound. For simple applications, we can treat the speed of sound as a constant, say 340 m/s, but the reality is that the speed of sound in air varies according to the temperature and humidity.

The formula for the speed of sound in air with temperature and humidity accounted for is:

$$\text{Speed of Sound} = 331.4 + 0.606T + 0.0124H$$

Where T is the temperature in degrees C and H is the Humidity expressed as a percentage.

For example, at 26 °C and 30% humidity, sound travels at a speed of:

$$331.4 + (0.606 \times 26) + (0.0124 \times 30) = 347.528 \text{ m/s}$$

and if the temperature were to increase to 30 °C, the speed of sound would increase to 349.952 m/s.

We can see that the temperature has almost fifty times more effect than humidity on the speed of sound (because 0.606 is almost 50 times 0.0124).

In the first of the following two examples, no consideration is made for variations in the speed of sound.

Place the ultrasonic sensor onto the breadboard such that the two transducers face away from the board and the four pins of the sensor are in j15, j16, j17 and j18.

This will equate to:

- the ground pin (GND) of the sensor being in j15
- the Echo pin of the sensor being in j16
- the Trigger pin of the sensor being in j17
- the voltage pin of the sensor (Vcc) being in j18

Connect f15 to GND on the UNO, f16 to d13, f17 to d10 and f18 to the 5V.

Plug an active buzzer onto the breadboard, such that the positive pin is in a1 and the negative pin is in a4. Connect e1 to d5 on the UNO and e4 to GND.

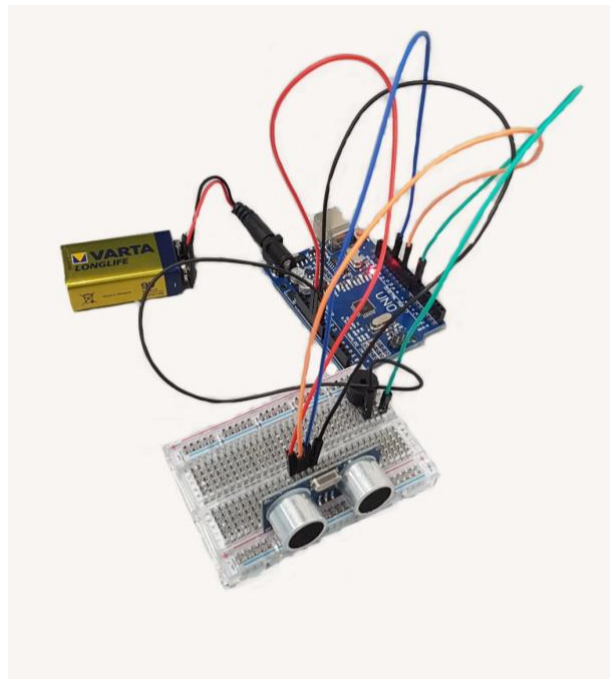


Figure 12 - An ultrasonic sensor and proximity buzzer

The following code will use the ultrasonic sensor to record distance and when that distance is less than 30 cm, will trigger an alarm that will beep 5 times per second.

Because no allowance is made for variation of the speed of sound, the constant 344 m/s is used to calculate the distance that the ultrasonic pulse travels.

Because the pulse travels to the point at which it is reflected and then returns, the total distance travelled is double the actual distance to the object, and thus the calculations must take this into consideration by dividing the total distance by two.

Create a new sketch and save to the name “Ultrasonic1” before entering the following code.

```
int trigPin = 10;
int echoPin = 13;
int alarmPin = 5;

void setup() {
  Serial.begin (9600);
  pinMode(trigPin, OUTPUT);
  pinMode(echoPin, INPUT);
}
void loop() {
  float duration, distance;
  digitalWrite(trigPin, LOW);
  delayMicroseconds(2); // wait for 2 microseconds

  digitalWrite(trigPin, HIGH); //sends a 10 second sequence of ultrasonic pulses
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);

  duration = pulseIn(echoPin, HIGH); //the time taken for the pulse to return to the sensor
  distance = (duration / 2) * 0.0344; // uses 344 m/s as the speed of sound, and dividing by 2
                                     // to allow for the fact that the distance travelled is double
                                     // the actual distance to the object.

  if (distance >= 400 || distance <= 2){
    Serial.print("Distance = ");
    Serial.println("Out of range");
  }
  else
  {
    if (distance < 30)
    {
      digitalWrite(alarmPin,HIGH);
      delay(200);
      digitalWrite(alarmPin,LOW);
      delay(200);
      Serial.print("Distance = ");
      Serial.print(distance);
      Serial.println(" cm");
    }
    else
    {
      Serial.print("Distance = ");
      Serial.print(distance);
      Serial.println(" cm");
    }
  }
  delay(500);
}
}
```

The ultrasonic sensor is accurate to within around 3 mm over a distance of 4 metres (400 cm), but we also do not know exactly from where in the transducer the measurement initiates. It is a good idea to “calibrate” the sensor if more precision is required.

To do this, run the sketch, and place a solid object at a known distance from the front lip of the sensor. When you open the serial port to see the distance being recorded, you can see any error and then edit the line in the code where the distance is calculated.

eg If the distance is 5 mm greater than the actual known distance, then edit the line of code to read: $\text{distance} = (\text{duration} / 2) * 0.0344 - .5$

Upload and re-run the code to see whether the distance is now more accurate.

If we are to use the temperature and humidity to calculate the speed of sound more accurately, we need to be able to determine the temperature and humidity. You will recall that in an earlier exercise we used the DHT11 sensor to record the temperature and humidity and so we can merge that code with our current code.

Before doing this, however, save the current code to a new name, eg “Ultrasonic2”

Locate the line of code that determines the distance:

```
distance = (duration / 2) * 0.0344;
```

and replace it with :

```
speed = 331.4 + (0.606 * DHT.temperature) + (0.0124 * DHT.humidity);  
distance = (duration / 2) * (speed / 10000);
```

If we wanted to “prove” that the speed of sound varies with temperature (and humidity), we can place the sensor in a situation where the distance will remain fixed and record the time taken for the rebounding ultrasonic pulse to be received by the sensor when the temperature and/or humidity are changed. A shoe box or similar is a good solution.

Because the distance is fixed, we do not need to record it. We simply need to see if there is a change in the time taken for the ultrasonic pulse to return to the sensor to know whether or not the speed of the pulse has varied.

The following code will generate the required data:

```
#include <dht.h>
#define DHT11_PIN 7

int trigPin = 10;
int echoPin = 8;
dht DHT; // calls the library function that records the temperature and humidity

void setup() {
  Serial.begin(9600); // opens the serial port on the computer
  pinMode(trigPin, OUTPUT); // sets the trigger pin on the ultrasonic sensor as an output
  pinMode(echoPin, INPUT); // sets the echo pin on the sensor as an input
}

void loop() {

  int chk = DHT.read11(DHT11_PIN); // reads the temperature and humidity
  float duration, distance; // defines the variables for duration and distance
  digitalWrite(trigPin, LOW); // sets the ultrasonic sensor trigger to OFF
  delayMicroseconds(2); // waits two microseconds

  digitalWrite(trigPin, HIGH); // turns on the trigger pin to send an ultrasonic pulse
  delayMicroseconds(10); // sends the pulse for 10 microseconds
  digitalWrite(trigPin, LOW); // turns off the pulse

  duration = pulseIn(echoPin, HIGH); // records how long it takes for the echo pulse to return to
  the sensor
  Serial.print("Total Time = ");
  Serial.print(duration);
  Serial.print(" and Temperature = ");
  Serial.print(DHT.temperature);
  Serial.print("c");
  Serial.print(" and Humidity = ");
  Serial.print(DHT.humidity);
  Serial.println("%");

  delay(2000);
}
```

Power the UNO and sensors and note the time, temperature (and humidity) being recorded. You can now move the box into an area where the temperature is markedly different. It might mean just moving out of an air-conditioned room, or if you have access, place the box inside a refrigerator. Watch the recorded temperature change over the next few minutes as the effect of the temperature variation takes effect.

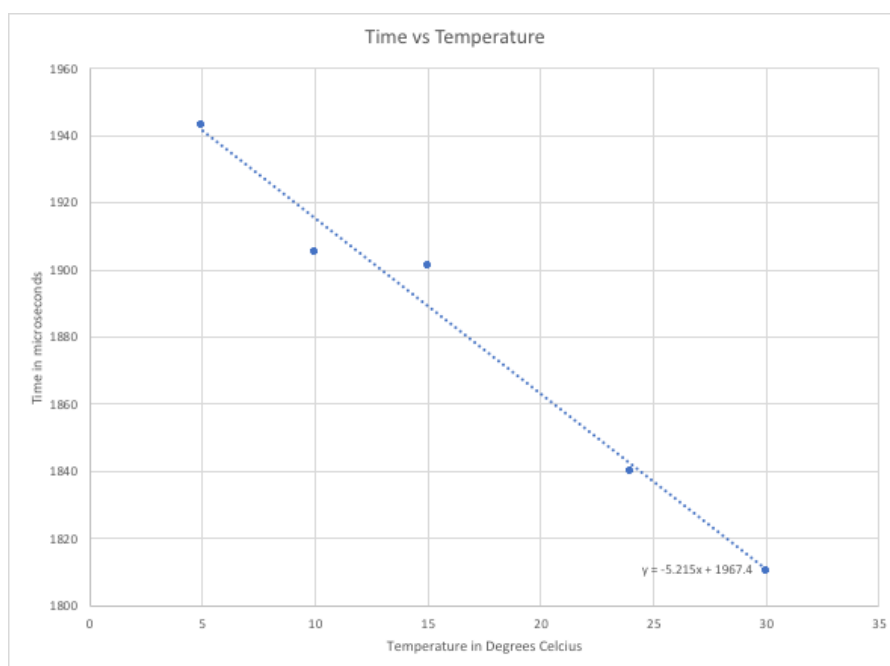
If you place the box in a variety of locations where the temperature differs, for example, in a cold room or freezer, outside in the fresh air (but not in direct sunlight), in an air-conditioned room, you can record any variation in the time taken for the ultrasonic pulse to travel back and forth. Because the Temperature and humidity sensor (DHT11) takes time to adjust to changes in temperature and/or humidity, you need to allow sufficient time for the readings to stabilise in each location. This might take several minutes.

We know that the distance is fixed by the length of the box, and so we can conclude that if there is any variation in the time taken for the pulse to rebound to the sensor, then the speed of sound has changed.

The time is being recorded in milliseconds and to demonstrate any relationship between the time taken and the temperature, it is a good idea to graph the time against the temperature.






For example, using the above method, the following five data points were collected:

Time in microseconds	Temperature in degrees C
1810	30
1840	24
1901	15
1905	10
1943	5



We can see from the trendline that as the temperature decreases, the time taken for the pulse to travel a fixed distance increases, which in turn tells us that the speed of sound decreases.

The Internet of Things (IoT)

Hardware requirements: UNO Thinxtra Xkit	    	ACARA element(s) Creating with ICT Managing and operating ICT Comprehending texts through listening, reading and viewing Analysing, synthesising and evaluating reasoning and procedures Reflecting on thinking and processes Inquiring – identifying, exploring and organising information and ideas Interpreting statistical information Social awareness Social management Using measurement Using fractions, decimals, percentages, ratios and rates
--	---	--

The biggest disadvantage of the above examples and trying to use sensors in real life situations, is that these examples are cumbersome and the number of wires and the breadboard make them difficult to move around or to expose to the weather.

In an ideal world we would be able to deploy these sensors without all the wires, and the good news is that we now have the capacity to do this because of new technologies using low power wireless networks that allow simple sensors to connect to the internet. The network of these devices is commonly referred to as the Internet of Things.

Wikipedia defines the Internet of Things as being the network of physical devices, vehicles, and other items embedded with electronics, software, sensors, actuators, and network connectivity which enable these objects to collect and exchange data.

Experts estimate that the IoT will consist of about 30 billion objects by 2020.

Even as we begin to engage with the IoT technologies, there are those who, recognising that data is only useful when it is analysed and acted upon and that because this process demands the involvement of people and other systems, have renamed the “Internet of Things” to the “Internet of Everything”

The final stage in our journey is to use the UNO to take advantage of these technologies.



Thinextra's Xkit

Thinextra is a technology company building Internet of Things (IoT) solutions and one device that is ideal for education is their Xkit, which uses an UNO to send temperature, barometric pressure, light and accelerometer data across a low power wide area network (LPWAN). One such low power network, Sigfox, is currently rolling out across Australia and already across New Zealand. Sigfox provides IoT connectivity in 45 countries over an area of more than 3.8 million km². By adding a Thinextra IoT shield to the UNO, we can gather data without the need for wired sensors, and because we then transmit that data across the Sigfox network, we can easily collect data from remote locations. Collecting data from multiple remote locations also helps the students develop an understanding of how researchers compile data sets.

Installing the Xkit on the UNO

Visit <https://www.thinextra.com/xkit/> and download the Xkit Installation Guide.

Xkit supports various sources of power supplies such as 9V, 5V and 3.3V by properly setting the jumpers on Xkit.

It is critically important to note that incorrectly configuring the jumpers can damage the Xkit, rendering it unable to communicate.

In the first instance, it is recommended that you configure the Xkit to receive its power from the Arduino (UNO), as shown in the adjacent diagrams.

Once you have checked the jumper configuration is correct and plugged the Xkit into the UNO, screw the wireless antenna onto the Xkit.

Next, you need to register the Xkit with the Sigfox network.

Go to <https://backend.sigfox.com/activate>

Choose Thinextra logo (Do not choose the Arduino logo!!)

Choose your country and again select the Thinextra logo.

Enter device information.

(Sigfox ID and PAC can be found on the package or on the barcode sticker included in the package)

Fill in account details.

Click Subscribe button.

If you have multiple devices to register and they are all owned by the school, they can all be registered to the same account, but if the Xkits are to be owned by multiple people (for example if each student is to own his or her own Xkit), then each Xkit should be registered to an account setup by the owner of the device.

Power From Arduino

i. Put jumpers on J17 (pins 1-2), J15 (pins 2-3), P4, P14 (pins 1-2 and 3-4) and P9 (pins 1-2 and 3-4).



ii. Plug the Xkit to Arduino.
iii. Plug the Arduino board to PC with USB cable supplied.



Connect the Xkit to your computer using the usual connection between the UNO and the USB port on the computer. By default, the Xkit should run a pre-installed demo sketch, transmitting a message containing the temperature value, the output voltage from the photovoltaic sensor, the pressure value and the acceleration every 10 minutes.

In order to be able to program the Xkit, or modify the demo sketch, we need to install the Xkit libraries that are provided with the Xkit.

Installing the Xkit libraries

Download the Xkit Arduino firmware repository as a zip file from <https://github.com/Thinextra/Xkit-Sample>

Unarchive/Unzip the zip file. You will require the libraries directory (including the sub-folders Isigfox, SimpleTimer and Tsensors), so take note of where you place the unarchived/unzipped folders.

Open the Arduino software and use the menu options to include a library from zip file, you can browse to each of the folders and include them one at a time. Do not be concerned that the menu option says add from zip file because once you proceed, it says browse to select zip file or folder. You need to do this for the Isigfox, SimpleTimer and Tsensors folders that reside inside the libraries folder of the unzipped XKit-Sample.

Locate the folder/location of your Arduino sketches and copy the DemoApp folder to that location: -

Restart the Arduino IDE software.

Open the DemoApp sketch.

Scroll down through the code until you come to the following block of code: -

```
// Init timer to send a Sigfox message every 10 minutes
unsigned long sendInterval = 600000;
timer.setInterval(sendInterval, timeIR);
```

Edit the code by replacing the 600000 to 10000 so that for the time being, a message will be sent every 10 seconds rather than every ten minutes.

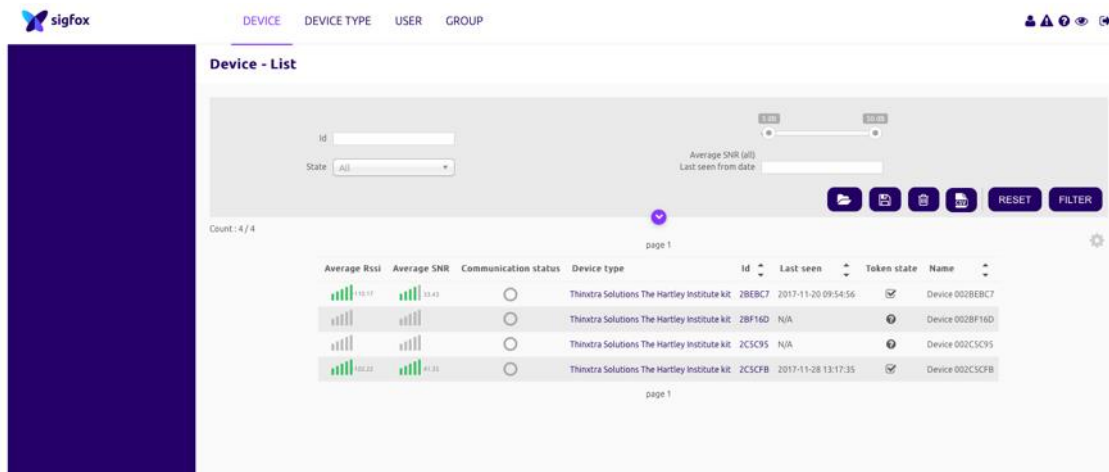
In order to upload a sketch from the computer to the Xkit, you MUST remove the jumpers on pin 9.

Make sure that the jumpers are removed from pin 9 and then upload the DemoApp. *Once the upload is completed, replace the jumpers on pin 9.*

You should shortly see a blue light on the Xkit blink every ten seconds as the Xkit transmits data via the Sigfox low power wide area network (LPWAN).

Accessing the data via the Sigfox website (backend)

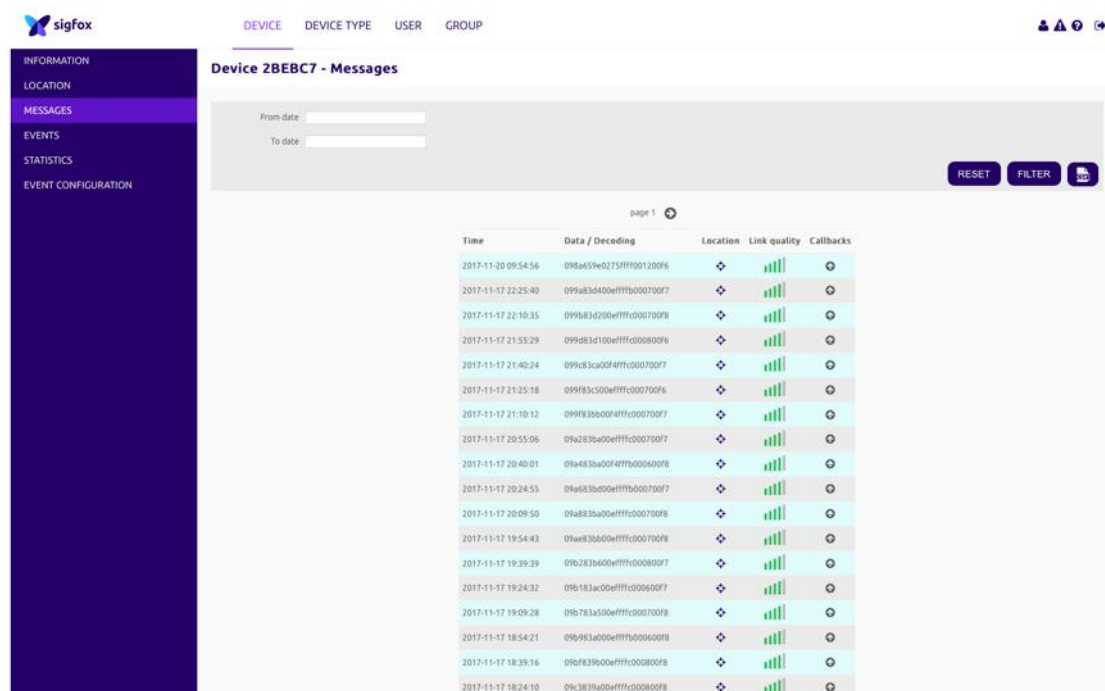
If you log into the Sigfox backend (www.backend.sigfox.com) and click on Device, you will see a list of registered devices.



The screenshot shows the Sigfox backend interface. The top navigation bar includes 'DEVICE', 'DEVICE TYPE', 'USER', and 'GROUP'. The left sidebar is dark blue. The main content area is titled 'Device - List'. It features a search bar with 'Id' and 'State' filters, and a 'Count: 4 / 4' indicator. Below the search bar is a table of devices. The first device, '2BEB7', is highlighted. The table columns are: Average RSSI, Average SNR, Communication status, Device type, Id, Last seen, Token state, and Name.

Average RSSI	Average SNR	Communication status	Device type	Id	Last seen	Token state	Name
10.17	13.43	○	Thinstra Solutions The Hartley Institute kit	2BEB7	2017-11-20 09:54:56	✓	Device 002BEB7
10.17	13.43	○	Thinstra Solutions The Hartley Institute kit	2BF16D	N/A	?	Device 002BF16D
10.17	13.43	○	Thinstra Solutions The Hartley Institute kit	2CSC95	N/A	?	Device 002CSC95
10.22	13.21	○	Thinstra Solutions The Hartley Institute kit	2CSCFB	2017-11-28 13:17:35	✓	Device 002CSCFB

Click on the Device ID of your Xkit and a menu will appear on the left-hand side of the window. Choose Messages from the menu and you will see the data that is being transmitted along with the timestamp showing when it arrived.



The screenshot shows the Sigfox backend interface for a specific device. The top navigation bar includes 'DEVICE', 'DEVICE TYPE', 'USER', and 'GROUP'. The left sidebar is dark blue and has a menu with 'INFORMATION', 'LOCATION', 'MESSAGES', 'EVENTS', 'STATISTICS', and 'EVENT CONFIGURATION'. The 'MESSAGES' option is selected. The main content area is titled 'Device 2BEB7 - Messages'. It features a search bar with 'From date' and 'To date' filters, and a 'Count: 4 / 4' indicator. Below the search bar is a table of messages. The first message is highlighted. The table columns are: Time, Data / Decoding, Location, Link quality, and Callbacks.

Time	Data / Decoding	Location	Link quality	Callbacks
2017-11-20 09:54:56	098a59e0275ff001200f6	◆	100%	○
2017-11-17 22:25:40	099a83d400efffb000700f7	◆	100%	○
2017-11-17 22:10:35	099a83d400efffb000700f8	◆	100%	○
2017-11-17 21:55:29	099a83d400efffb000700f6	◆	100%	○
2017-11-17 21:40:24	099a83d400efffb000700f7	◆	100%	○
2017-11-17 21:25:18	099a83d400efffb000700f6	◆	100%	○
2017-11-17 21:10:12	099a83d400efffb000700f7	◆	100%	○
2017-11-17 20:55:06	09a283b000efffb000700f7	◆	100%	○
2017-11-17 20:40:01	09a283b000efffb000700f8	◆	100%	○
2017-11-17 20:24:55	09a283b000efffb000700f7	◆	100%	○
2017-11-17 20:09:50	09a283b000efffb000700f8	◆	100%	○
2017-11-17 19:54:43	09a283b000efffb000700f8	◆	100%	○
2017-11-17 19:39:39	09a283b000efffb000700f7	◆	100%	○
2017-11-17 19:24:32	09a283b000efffb000700f7	◆	100%	○
2017-11-17 19:09:28	09a283b000efffb000700f8	◆	100%	○
2017-11-17 18:54:21	09a283b000efffb000700f8	◆	100%	○
2017-11-17 18:39:16	09a283b000efffb000700f8	◆	100%	○
2017-11-17 18:24:10	09a283b000efffb000700f8	◆	100%	○

Using the 'CSV' button to the upper right corner of the page, you can download the data to a text file, which can be loaded into any of the myriad spreadsheet programs for analysis of the data. I will explore the analysis later.

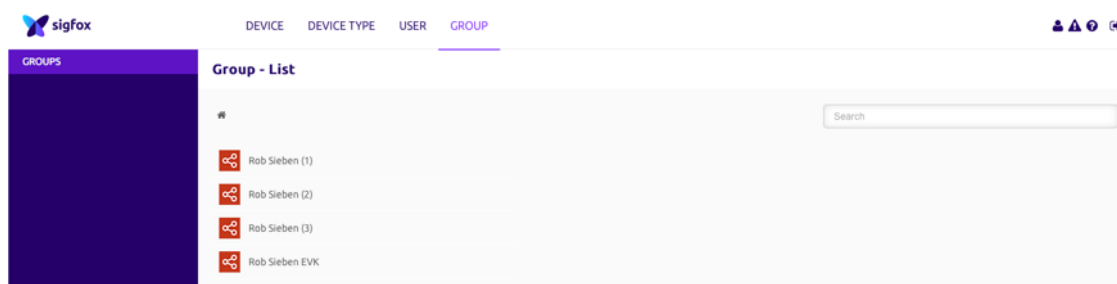
Accessing the data via an automated system

Whilst the data can be downloaded as a CSV file, it might be preferable to avoid having students needing to log into the Sigfox backend, and thus you might like to automate retrieving the data to a website/file of your choice.

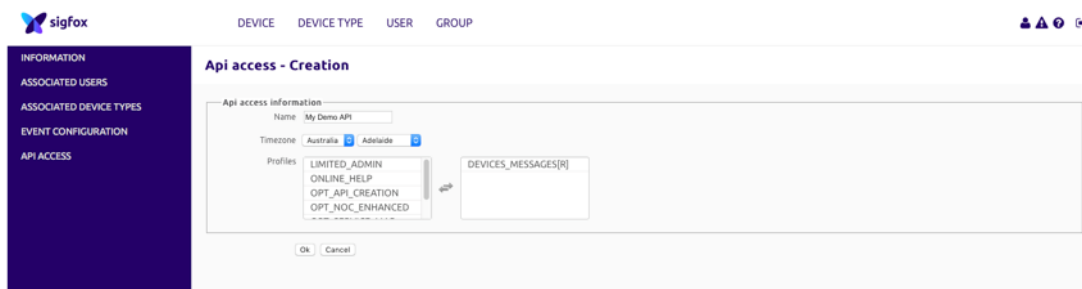
Sigfox provides a mechanism through their backend to setup processes called “Callbacks” to facilitate this, but it might be easier for the average person to use a Google Sheet with a short script that accesses the data and downloads it into pre-defined columns of the sheet.

Whilst more convenient in the long run, there is some setup work required to facilitate this.

When an Xkit is registered to the Sigfox network, it is automatically added to a ‘Group’. If you log in to the Sigfox backend and click on ‘Group’, you will see one group for every device that has been registered.



If you select one of the groups, and then select API Access from the menu on the left, you need to create a new API. Assign a name to your API and select the data messages from the profiles options.



Select OK to save this API and you will see displayed on the screen, the login name and password for your API. These are the details that will be required by any script that you create to download the data automatically from the Sigfox backend.

If you also click on the URL for this page, you will see an address that resembles the following:

<https://backend.sigfox.com/api-user/59eff4739e93a14edbd4d58/list>

The long alpha-numeric string is the {device_id} that will be required in the script line that accesses the data:

GET <https://backend.sigfox.com/api/devices/{device-id}/messages>

Depending on individual teacher’s coding capability, an external party might be required to assist in generating the required script, but once done, teachers can easily provide the core

script to his/her students and guide the students through the minor changes to personalise the script for each student and his/her Xkit.

The finished script will access the Sigfox backend and pass login and password credentials before downloading the data from the Xkit. It can then insert that data into designated columns of a spreadsheet of database, from where it can be manipulated, analysed and displayed.

In a Google Sheet, the result will resemble the following:

	A	B	C	D	E	F
	Device	Time	Data	Snr	LinkQuality	SeqNumber
1	Device					
2	2C5CFB	1511749927	0a3c839b0320fff8000800f5	32.11	GOOD	697
3	2C5CFB	1511749913	0a3b839c0480fff7000800f5	34.11	GOOD	696
4	2C5CFB	1511749904	0a3b839b0419fff8000800f6	33.6	GOOD	695
5	2C5CFB	1511748805	09f483a70432fff2000300f4	38.53	GOOD	694
6	2C5CFB	1511747906	098a5d690414fff2000300f4	39.36	GOOD	693
7	2C5CFB	1511389962	0a6883070102fff2000300f5	37.27	GOOD	692
8	2C5CFB	1511389064	0a6482fe0102fff2000400f6	36.36	GOOD	691
9	2C5CFB	1511388163	0a6382fc0102fff3000300f4	38.75	GOOD	690
10	2C5CFB	1511387264	0a6082f10102fff2000400f4	38.74	GOOD	689
11	2C5CFB	1511386364	0a5f82f00102fff2000400f6	39.56	GOOD	688
12	2C5CFB	1511385465	0a6182f30102fff2000400f4	38.46	GOOD	687
13	2C5CFB	1511384563	0a6382ff0102fff3000400f5	39.01	GOOD	686
14	2C5CFB	1511383664	0a6482fc0102fff2000400f5	40.51	GOOD	685
15	2C5CFB	1511382765	0a6482f20102fff2000300f5	38.99	GOOD	684
16	2C5CFB	1511381865	0a6582ea0102fff3000300f4	39.14	GOOD	683
17	2C5CFB	1511380965	0a6982de0102fff2000400f4	39.22	GOOD	682
18	2C5CFB	1511380067	0a6982cf0102fff2000400f4	38.75	GOOD	681
19	2C5CFB	1511379165	0a6b82be0102fff2000300f5	39.01	GOOD	680
20	2C5CFB	1511378267	0a6e82c10102fff3000300f5	38.16	GOOD	679
21	2C5CFB	1511377365	0a6f82bf0102fff3000400f5	38.11	GOOD	678
22	2C5CFB	1511376466	0a7082bf0102fff2000400f4	39.36	GOOD	677
23	2C5CFB	1511375565	0a7082c10102fff3000300f5	38.93	GOOD	676
24	2C5CFB	1511374666	0a7082c10102fff2000300f4	39.46	GOOD	675
25	2C5CFB	1511373766	0a7182d90102fff1000400f4	38.32	GOOD	674

This sheet has data feeding in from two Xkits (Serial numbers 2C5CFB and 2BEB7). The raw data feeds into two separate sheets and that data is then converted into a useable format in two other sheets, one of which resembles the following:

Raw data	Signal strength	Date time	Temperature	Pressure	Photo	X_Acc	Y_Acc	Z_Acc
0a3c839b0320ff8000800f5	32.11	27/11/17 13:2:7	26.2	101073	0.8	262.112	0.032	0.98
0a3b839c0480ff7000800f5	34.11	27/11/17 13:1:53	26.19	101076	1.152	262.108	0.032	0.98
0a3b839b0419ff8000800f6	33.6	27/11/17 13:1:44	26.19	101073	1.049	262.112	0.032	0.984
09f483a70432ff2000300f4	38.53	27/11/17 12:43:25	25.48	101109	1.074	262.088	0.012	0.976
098a5d690414ff2000300f4	39.36	27/11/17 12:28:26	24.42	71739	1.044	262.088	0.012	0.976
0a6883070102ff2000300f5	37.27	23/11/17 9:2:42	26.64	100629	0.258	262.088	0.012	0.98
0a6482fe0102ff2000400f6	36.36	23/11/17 8:47:44	26.6	100602	0.258	262.088	0.016	0.984
0a6382fc0102ff3000300f4	38.75	23/11/17 8:32:43	26.59	100596	0.258	262.092	0.012	0.976
0a6082f10102ff2000400f4	38.74	23/11/17 8:17:44	26.56	100563	0.258	262.088	0.016	0.976
0a5f82f00102ff2000400f6	39.56	23/11/17 8:2:44	26.55	100560	0.258	262.088	0.016	0.984
0a6182f30102ff2000400f4	38.46	23/11/17 7:47:45	26.57	100569	0.258	262.088	0.016	0.976
0a6382ff0102ff3000400f5	39.01	23/11/17 7:32:43	26.59	100605	0.258	262.092	0.016	0.98
0a6482fc0102ff2000400f5	40.51	23/11/17 7:17:44	26.6	100596	0.258	262.088	0.016	0.98
0a6482f20102ff2000300f5	38.99	23/11/17 7:2:45	26.6	100566	0.258	262.088	0.012	0.98
0a6582ea0102ff3000300f4	39.14	23/11/17 6:47:45	26.61	100542	0.258	262.092	0.012	0.976
0a6982de0102ff2000400f4	39.22	23/11/17 6:32:45	26.65	100506	0.258	262.088	0.016	0.976
0a6982cf0102ff2000400f4	38.75	23/11/17 6:17:47	26.65	100461	0.258	262.088	0.016	0.976
0a6b82be0102ff2000300f5	39.01	23/11/17 6:2:45	26.67	100410	0.258	262.088	0.012	0.98
0a6e82c10102ff3000300f5	38.16	23/11/17 5:47:47	26.7	100419	0.258	262.092	0.012	0.98
0a6f82bf0102ff3000400f5	38.11	23/11/17 5:32:45	26.71	100413	0.258	262.092	0.016	0.98
0a7082bf0102ff2000400f4	39.36	23/11/17 5:17:46	26.72	100413	0.258	262.088	0.016	0.976
0a7082c10102ff3000300f5	38.93	23/11/17 5:2:45	26.72	100419	0.258	262.092	0.012	0.98
0a7082c10102ff2000300f4	39.46	23/11/17 4:47:46	26.72	100419	0.258	262.088	0.012	0.976

Understanding the Xkit payload

Because the LPWAN transmits only very small packets of data, the Xkit code converts the data being collected to hexadecimal. Hexadecimal strings can contain a relatively large amount of data whilst still being very short and so can be transmitted across the Sigfox LPWAN.

In the case of the Xkit, the temperature data is a value recorded to two (2) decimal places and in order to avoid transmitting the decimal point, we multiply the temperature by 100 before converting it to a hexadecimal value.

Similar calculations are applied to the barometric pressure and the light reading.

Below is an example of what a typical string of data from the Xkit looks like.

“c808b485f504feff0700f800”

This string can be broken into blocks of 2 characters.

“c8”, “08”, “b4”, “85”, “f5”, “04”, “fe”, “ff”, “07”, “00”, “f8”, “00”

If you examine the DemoApp code, you will see the following block of code:

```
buf_str[0] = tempt.bytes[0];  
buf_str[1] = tempt.bytes[1];  
buf_str[2] = pressure.bytes[0];  
buf_str[3] = pressure.bytes[1];  
buf_str[4] = photo.bytes[0];  
buf_str[5] = photo.bytes[1];  
buf_str[6] = x_g.bytes[0];  
buf_str[7] = x_g.bytes[1];  
buf_str[8] = y_g.bytes[0];  
buf_str[9] = y_g.bytes[1];  
buf_str[10] = z_g.bytes[0];  
buf_str[11] = z_g.bytes[1];  
  
Send_Pload(buf_str, payloadSize);
```

Converting the payload into useable data for the classroom

This block tells us that the first two blocks of the data give us the temperature, the third and fourth give us the barometric pressure, the fifth and sixth blocks give us the light reading. The remaining blocks give us accelerometer data (the angle that the device is sitting at), but we do not need that data for this exercise.

If we look at our data blocks again, “c808b485f504feff0700f800”, the first two blocks are “c8” and “08”. We would need to interrogate the code further to establish that the two temperature blocks are actually transmitted in reverse order, using a format called ‘little-endian’ in which you store the least significant byte in the smallest address. This means that the actual temperature string is not “c808” but rather is “08c8”! Remember too, that that this number is the actual temperature multiplied by 100 to avoid having to send a decimal point.

The decimal equivalent of this hexadecimal number is 2248, so when we divide by 100, the actual temperature is 22.48 degrees Centigrade.

In the same way, the barometric pressure is the third and fourth blocks in reverse order, and this time the data was divided by 3 before being transmitted in hexadecimal. This means that the pressure was “85b4” which converts to a decimal value of 102684 pascals.

The light/photo reading is delivered by the fifth and sixth blocks, again in reverse order and this time multiplied by 1000 to remove the decimal. So the block “04f5” converts to a decimal reading of 1269, which when we divide by 1000 to revert to the correct value gives us 1.269 units.

If we were interested in position of the device, we could also convert the accelerometer data from hexadecimal to decimal values.

We can do all this in a spreadsheet using functions to calculate the values for us. Indeed, that is precisely what was done in the earlier Google Sheet example. Remember that it is just as likely that the data has been downloaded manually from the Sigfox backend and in that case, we need to open the CSV file and manually embed the necessary formulae (functions) to convert the hexadecimal data back to useable decimal data.

The following is an example of what this looks like.

Raw data	Signal strength	Date time	Temperature	Pressure	Photo	X Acc	Y Acc	Z Acc
c808b485f504feff0700f800	Good	11/7/17 17:02	22.48	102684	1.269	262.136	0.028	0.992
d208b485ff04feff0700f700	Good	11/7/17 16:52	22.58	102684	1.279	262.136	0.028	0.988
c208af859804feff0700f600	Good	11/7/17 16:42	22.42	102669	1.176	262.136	0.028	0.984
b208ad851904fcff0700f700	Good	11/7/17 16:32	22.26	102663	1.049	262.128	0.028	0.988
af08ac85cb03feff0600f700	Excellent	11/7/17 16:22	22.23	102660	0.971	262.136	0.024	0.988
b908a985f201fdff0800f700	Excellent	11/7/17 16:12	22.33	102651	0.498	262.132	0.032	0.988
d308a785f201feff0900f700	Good	11/7/17 16:02	22.59	102645	0.498	262.136	0.036	0.988
ec08a8850a02feff0800f7	Good	11/7/17 15:52	22.84	102648	0.522	262.136	0.032	0.988
1d09a5851c05feff0a00f700	Good	11/7/17 15:42	23.33	102639	1.308	262.136	0.04	0.988
e808a6858703f9ff0000f600	Good	11/7/17 15:32	22.8	102642	0.903	262.116	0	0.984
e008a985cb03f9ff0100f800	Excellent	11/7/17 15:22	22.72	102651	0.971	262.116	0.004	0.992
b508a685e803f7ff0000f600	Good	11/7/17 15:12	22.29	102642	1	262.108	0	0.984
b008a8857b04f8ff0100f800	Good	11/7/17 15:02	22.24	102648	1.147	262.112	0.004	0.992
9c08a685a803f8ff0000f800	Excellent	11/7/17 14:52	22.04	102663	1	262.112	0	0.992

My personal preference is to modify the DemoApp code so that the data transmitted is not using the little-endian format. It is easier for students to manipulate the data when the blocks are transmitted in the order required. To achieve this, we would modify the DemoApp code so that the block building the payload reads thus:








```

buf_str[0] = tempt.bytes[1];
buf_str[1] = tempt.bytes[0];
buf_str[2] = pressure.bytes[1];
buf_str[3] = pressure.bytes[0];
buf_str[4] = photo.bytes[1];
buf_str[5] = photo.bytes[0];
buf_str[6] = x_g.bytes[1];
buf_str[7] = x_g.bytes[0];
buf_str[8] = y_g.bytes[1];
buf_str[9] = y_g.bytes[0];
buf_str[10] = z_g.bytes[1];
buf_str[11] = z_g.bytes[0];

Send_Pload(buf_str, payloadSize);

```

Kobo Collect

Hardware:		ACARA element(s)
Computer and mobile phone		Creating with ICT
		Managing and operating ICT
		Analysing, synthesising and evaluating reasoning and procedures
		Reflecting on thinking and processes
		Inquiring – identifying, exploring and organising information and ideas
		Interpreting statistical information
		Using measurement

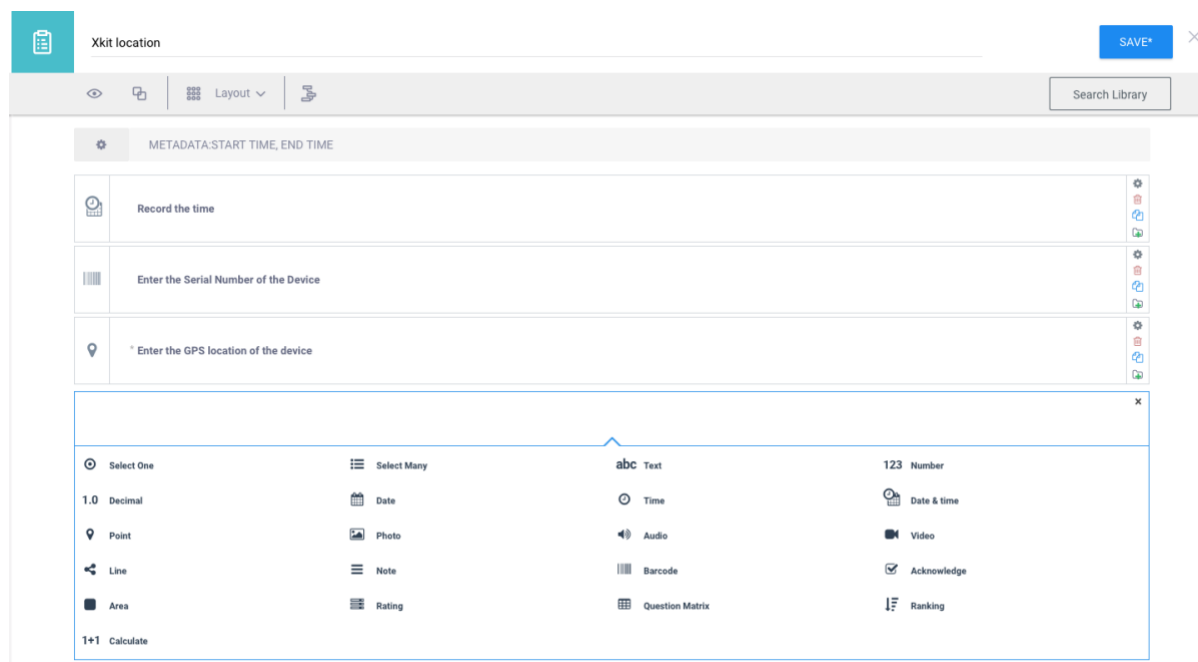
You will note that the data collected from the Xkit does not include the location of the device. This data, is valuable, particularly when collecting data from multiple devices simultaneously.

One way of getting around this dilemma is to employ an app such as Kobo Collect, which is freely available. The app runs on Android devices but the web version can also be accessed on iPhones and other devices.

Go to www.kobotoolbox.org and create an account for yourself and another for your class.

Log in as yourself and select NEW project.

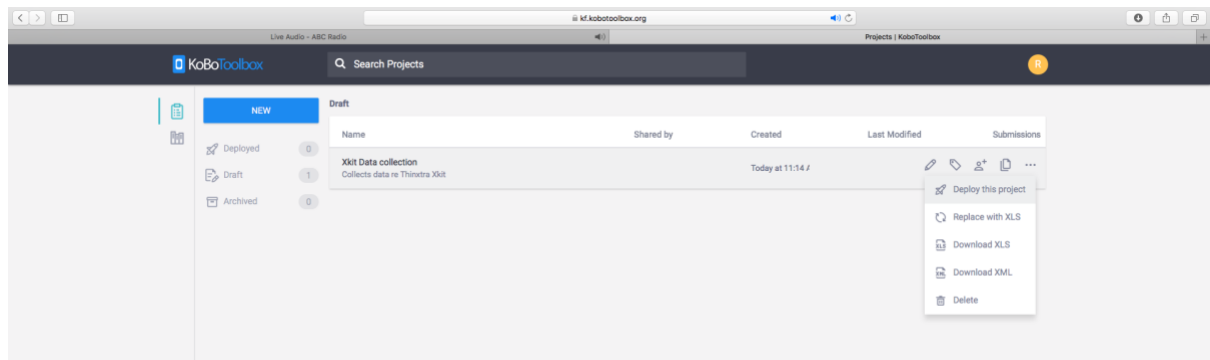
You can now create a new form that can be used to collect data. Each time you add a new question, you will be prompted for the type of data.



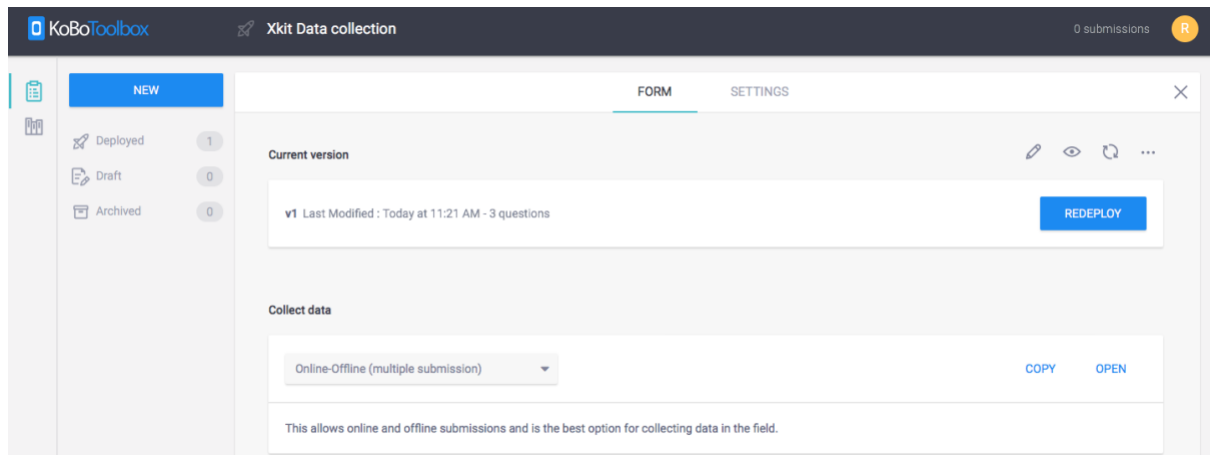
The screenshot shows the Kobo Collect web interface. At the top, there's a header with a 'Xkit location' field and a 'SAVE*' button. Below the header is a toolbar with icons for view, copy, layout, and print. The main area displays a form with three questions: 'Record the time', 'Enter the Serial Number of the Device', and '* Enter the GPS location of the device'. A modal window is open, showing a grid of question types: Select One, Select Many, Date, Photo, Note, Rating, Text, Time, Audio, Barcode, Question Matrix, Number, Date & time, Video, Acknowledge, and Ranking. The 'Date & time' and 'Barcode' options are highlighted.

In the illustrated example, I have selected three types of questions (date and time, barcode and location).

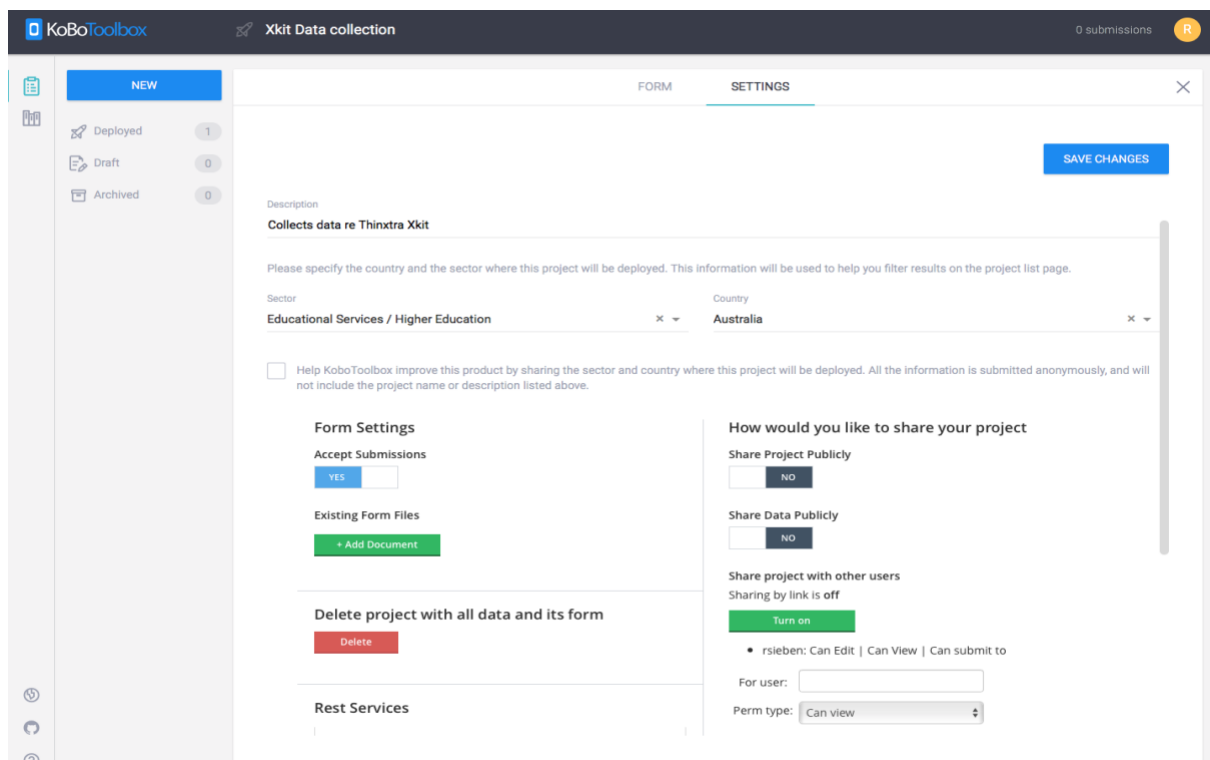
Once you are happy with your questions, save your form and click the back icon. You will now see your form in a list of all your forms. Place your cursor over the form and select the “More Actions” at the far right. Select to deploy the form.



Once deployed, you will be able to edit the form or view data collected by the form.



If you now click on settings, you will be able to view and edit all the permissions for this form.



Enter the username for your class and assign the permission “Can submit to” and save the settings.








Students in your class can now open the Kobo Collect app on their Android devices and get the form you have shared and can submit their data.

This is both a good exercise in the scientific process of data collection, but also lays a foundation for the understanding that scientific data sets do not come neatly bundled.

By matching the Kobo data with the Sigfox data, we can create a data set that now not only provides temperature, barometric pressure and light information, but also identifies the location of the device. We do this by joining together the data based on the serial number of the devices.

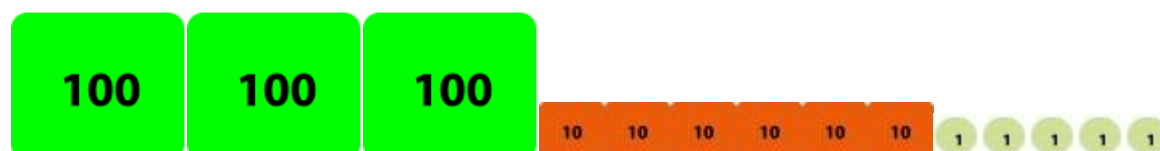
If we were to draw the location of the devices on a map, and then join together those devices that reported the same barometric pressure at the same time, we would create an isobar, the basis of the common weather map! In this way we will have linked a coding/STEM activity to both science and geography(HASS) within the curriculum.

Hexadecimal

Hardware requirements: UNO Thinextra Xkit	      	ACARA element(s) Comprehending texts through listening, reading and viewing Analysing, synthesising and evaluating reasoning and procedures Reflecting on thinking and processes Inquiring – identifying, exploring and organising information and ideas Estimating and calculating with whole numbers Recognise and use patterns and relationships Social awareness Social management Recognising culture and developing respect Exploring connections between representations of number and pattern and how they relate to aspects of counting and relationships of Aboriginal and Torres Strait Islander cultures.
---	---	--

We have been raised to count using the decimal or base 10 number system. If you count from zero to ten, you follow the sequence 0,1,2,3,4,5,6,7,8,9 and then, because we have no more digits, or no single digit to represent the next number, we use two digits, a 1 and a 0 to make 10. The zero tells us that there are no ‘units’ and the 1 tells us that there is one group of ‘ten’. If we consider the different columns used in writing decimal numbers, we have the units, tens, hundreds and thousands columns.

Pictorially, we can show a number like 365 as:-



Another way of listing this is as powers of ten. The units are the 10^0 , the tens are 10^1 s, the hundreds are 10^2 s and the thousands are 10^3 s. We could illustrate this in a table thus: -

Number	10^3	10^2	10^1	10^0
13			1	3
26			2	6
365		3	6	5
3189	3	1	8	9

We can apply similar methods to counting in other systems and this presents us with a good opportunity to consider the counting systems of the Aboriginal and Torres Strait Islander cultures.

The Australian Aboriginal and Torres Strait Islander cultures have long been said to have very few numbers. As the following table shows, this is clearly not the case.

Language and location	1 (One)	2 (Two)	3 (Three)	4 (Four)	5 (Five/hand)	Many
Angkamuthi (Cape York)	Ipima	Udhima	Wuchama	-	-	Makyan
Badjirri (Tinnenburra)	Garritja	Bulanha	Bulaha-garritja	Bulanha-bulanha	Marra	
Baradha (Nebo)	Warba	Bularu	Bularu-Warba	Bularu-bularu	Maal	Dalmarri
Barrungam (Dalby)	Wengge or Pia	Colooboola	Coloomboola-pia	Kangooai	Maa	Toogool-gurra
Bayali (Keppel Bay)	Webben	Booli	Koorel	-	Moolloon	-
Bidjara (Charleville)	Wangarra	Bularu	Bularu Wangarra	Bularu-bularu	Mara	Banya Banya
Bigambul (South-West Queensland)	Bardja	Boolul	Boolul-bardja	Goonimbilla	Marr	Mayburra

You might notice that in a number of these cultures, whilst there are numbers for one and two, the numbers for three and four are written as compound numbers, one plus two and two plus two. It is not uncommon for these cultures to consider five in relation to ‘hand’.

Some cultures have numbers for one, two and three, but they then treat four as the compound number, two plus two. Yet other cultures have numbers to ten or twenty without using compound numbers and without relating to ‘hand’

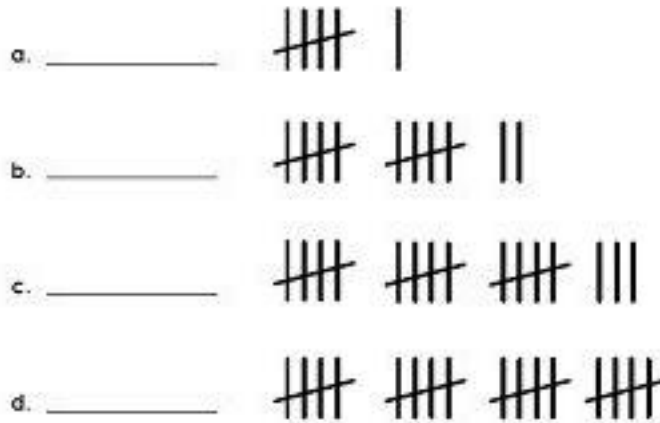
The Kaurna people, who are the traditional owners of the Adelaide Plains region they have distinct numbers for one to five, although we might posit that they regard five as the compound number, four plus one.

Kuma, purlaitye, marnkutye, yerrabula and yerrabula kuma

Students might investigate why these cultures do not “need” the numbers that we use in our cultures. Common perceptions that the absence of larger numbers is an indication that the people are primitive should be dispelled. Indeed, students should be made aware that the need for numbers can be directly attributed to the way that these cultures hunt and gather food on a needs basis and that the absence of large numbers in no way limits their ability to count large quantities very accurately.

John Harris’ 1987 *Australian Aboriginal and Islander mathematics* is a good source when looking to counter many of the common fallacies concerning Aboriginal number systems and enhance cultural respect.

The Aboriginal and Torres Strait Islander cultures count very accurately and quickly by grouping items into small groups and then just counting the groups. Many of us would be familiar with counting items in groups of five using the tally marks system. The method of seeing small numbers of items and not having to count them to know how many there are, is called subatizing. We could use the following as a subatizing exercise.



In effect, we might consider the counting systems of some of the Aboriginal and Torres Strait Islander cultures as a Base 5 system because of the use of “hand” as the representation of the number five and their technique of grouping into sets of five in order to count larger quantities.

Another number system we should consider is Base 2. Base 2 (Binary) is a good example to consider, because it is the basic language on which computers are built. In a binary system, there are only two digits, 0 and 1. A computer can be considered to be a complex network of wires, each either carrying current (on) or not carrying current (off). Numbers in Binary are displayed as powers of two.

Number	2^4 (2 x 2 x 2 x 2 = 16)	2^3 (2 x 2 x 2 = 8)	2^2 (2 x 2 = 4)	2^1 (2)	2^0 (1)
13		1	1	0	1
26	1	1	0	1	0

Base 3 is a number system based on powers of three. In base three there are only three digits, 0, 1 and 2.

Number	3^4	3^3	3^2	3^1	3^0
13			1	1	1
26			2	2	2

Base 8 (Octal) is a number system based on powers of eight and there are only eight digits, 0, 1, 2, 3, 4, 5, 6, and 7

Number	8^4	8^3	8^2	8^1	8^0
13				1	5
26				3	2

Can you see that as the ‘base’ of the number system increases, the length of the number will decrease? This is very useful when we want to transmit large numbers but can only transmit very short data strings. To do this, we use base 16 or hexadecimal, a number system based on powers of sixteen.

Hexadecimal allows us to make very large numbers into very short data strings and it is particularly useful because any number that is a power of 16 is also a power of two, and so can be easily converted back into binary, the language of computers.

Since we don't have single digits for 10, 11, 12, 13, 14, 15 and 16, we use the letters A, B, C, D, E, and F. That is, the 'digits' are 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, and F.

Number	16^4	16^3	16^2	16^1	16^0
13					D
26				2	0

Challenge 10

Can you complete the following table?

Decimal	Binary	Base 5	Octal	Hexadecimal
63				
127				
365				
3189				

Challenge 11

Open the sketch you wrote earlier to measure the moisture level in soil and which lit up one of three LEDs based on the moisture level.

Can you insert a block of code that will read the moisture value, convert it to hexadecimal and then print both the hexadecimal and decimal values to the serial port?

Solutions to Challenges

Challenge 1:

```
void setup() {  
  pinMode(8, OUTPUT);  
}  
  
void loop() {  
  digitalWrite(8, HIGH); // turn the LED on (HIGH is the voltage level)  
  delay(1000);           // wait for a second  
  digitalWrite(8, LOW);  // turn the LED off by making the voltage LOW  
  delay(1000);           // wait for a second  
}
```

Challenge 2

Edit the first line of the code to read:

```
int ledPin1 = 8
```

Challenge 3

```
int ledPin1 = 8;  
int ledPin2 = 9;  
  
void setup() {;  
  pinMode(ledPin1, OUTPUT);  
}  
  
void loop() {  
  digitalWrite(ledPin1, HIGH); // turn the ledPin1 on (HIGH is the voltage level)  
  digitalWrite(ledPin2, LOW);  // turn the ledPin2 on (HIGH is the voltage level)  
  delay(1000);                 // wait for a second  
  digitalWrite(ledPin1, LOW);  // turn the ledPin1 off by making the voltage LOW  
  digitalWrite(ledPin2, HIGH); // turn the ledPin2 off by making the voltage LOW  
  delay(1000);                 // wait for a second  
}
```

Challenge 4

```
int ledPin1 = 8;  
int ledPin2 = 9;  
int ledPin3 = 10;  
  
void setup() {;  
  pinMode(ledPin1, OUTPUT);  
  pinMode(ledPin2, OUTPUT);  
  pinMode(ledPin3, OUTPUT);  
}  
  
void loop() {  
  digitalWrite(ledPin1, HIGH); // turn the ledPin1 on  
  digitalWrite(ledPin2, LOW);  // turn the ledPin2 off  
  digitalWrite(ledPin3, LOW);  // turn the ledPin3 on  
  delay(5000);                 // wait for 5 seconds  
}
```

```

digitalWrite(ledPin1, LOW); // turn the ledPin1 off
digitalWrite(ledPin2, HIGH); // turn the ledPin2 on
digitalWrite(ledPin3, LOW); // turn the ledPin3 off
delay(5000);                // wait for 5 seconds
digitalWrite(ledPin1, LOW); // turn the ledPin1 off
digitalWrite(ledPin2, LOW); // turn the ledPin2 off
digitalWrite(ledPin3, HIGH); // turn the ledPin3 on
delay(4000);                // wait for 4 seconds
digitalWrite(ledPin1, LOW); // turn the ledPin1 off
digitalWrite(ledPin2, HIGH); // turn the ledPin2 off
digitalWrite(ledPin3, HIGH); // turn the ledPin3 on
delay(1000);                // wait for 1 seconds
}

```

Challenge 5

Because the Array indices begin at 0 not 1, the ledPins need to be changed from ledPin1, ledPin2 and ledPin3 to ledPin0, ledPin1 and ledPin2.

Challenge 6

```

void setup()
// initialize digital pin LED_BUILTIN as an output.
pinMode(LED_BUILTIN, OUTPUT);
}

void loop() {
  for (int i = 1; i < 4; i++)
  { digitalWrite(LED_BUILTIN, HIGH); // turn the LED on for 400 ms (dot)
    delay(400);
    digitalWrite(LED_BUILTIN, LOW); // turn the LED off for 400 ms
    delay(400);
  }
  digitalWrite(LED_BUILTIN, LOW); // turn the LED off for 1200 ms (pause between
letters)
  delay(800);
  for (int i = 1; i < 4; i++)
  { digitalWrite(LED_BUILTIN, HIGH); // turn the LED on for 1200 ms (dash)
    delay(1200);
    digitalWrite(LED_BUILTIN, LOW); // turn the LED off for 400 ms
    delay(400);
  }
  digitalWrite(LED_BUILTIN, LOW); // turn the LED off for 1200 ms (pause between
letters)
  delay(800);
  for (int i = 1; i < 4; i++)
  { digitalWrite(LED_BUILTIN, HIGH); // turn the LED on for 400 ms (dot)
    delay(400);
    digitalWrite(LED_BUILTIN, LOW); // turn the LED off for 400 ms
    delay(400);
  }
  digitalWrite(LED_BUILTIN, LOW); // turn the LED off for 2800 ms (pause between
words)
  delay(2400);
}

```

Challenge 8

A delay of 5 minutes or $5 \times 60 \times 1000$ milliseconds (300,000 milliseconds) would be required.

We cannot simply scan for movement every 5 minutes, because if motion occurred just after reading the sensor, the light would remain off, despite the movement. We actually want to delay after turning on the LED but then, after the 5 minutes, scan every 2 seconds again until motion is detected. To do this we would need 2 separate delays. The delay in reading the sensor remains as it is and we include an additional delay of 5 minutes after the command to turn the LED on.

Challenge 9

The two if statements prior to this one address the situations where the moisture reading is less than 20 or between 20 and 250. The only remaining possibility is that the reading is over 250 so we do not need to ask if this is the case, because we have already dealt with any other possibilities.

Challenge 10

Decimal	Binary	Base 5	Octal	Hexadecimal
63	111111	223	77	3F
127	1111111	1002	177	7F
365	101101101	2430	555	16D
3189		100224	6165	C75

Challenge 11

```
int sensorPin = A0;
int ledPin1 = 8
int ledPin2 = 9
int ledPin3 = 10;

void setup() {
  pinMode(ledPin1, OUTPUT);
  pinMode(ledPin2, OUTPUT);
  pinMode(ledPin3, OUTPUT);
  pinMode(sensorPin, INPUT);
  Serial.begin(9600);
}

void loop() {
  int val = analogRead(sensorPin);
  Serial.print(val);
  Serial.print(',');
```

```
// The following block converts the analog reading to Hexadecimal
// so that if we wanted to transmit it via the IoT, this value could be used.
// It is not required for the basic task of calibrating LEDs.
```

```
char hexval[4]="0000";
char Hexcharacters[16] = "0123456789ABCDEF";
int y = val;
for (int i = 0; i < 4; i++)
{
    int j = 3-i;
    int hexa = y/16;
    Serial.print(hexa);
    Serial.print(',');
    int hexb = y%16;
    hexval[j] = Hexcharacters[hexb];
    Serial.print("hexval[j] = ");
    Serial.println(Hexcharacters[hexb]);
    y = hexa;
}
Serial.print("Hex Conversion is ");
Serial.println(hexval);
delay(2000);
// Configure LEDs to indicate moisture levels.

if (val < 25)
{
    digitalWrite(ledPin1, HIGH);
    digitalWrite(ledPin2, LOW);
    digitalWrite(ledPin3, LOW);
}
else
if (val > 25 && val < 250)
{
    digitalWrite(ledPin1, LOW);
    digitalWrite(ledPin2, HIGH);
    digitalWrite(ledPin3, LOW);
}
else
{
    //if (val > 250)
    digitalWrite(ledPin1, LOW);
    digitalWrite(ledPin2, LOW);
    digitalWrite(ledPin3, HIGH);
}

}
```

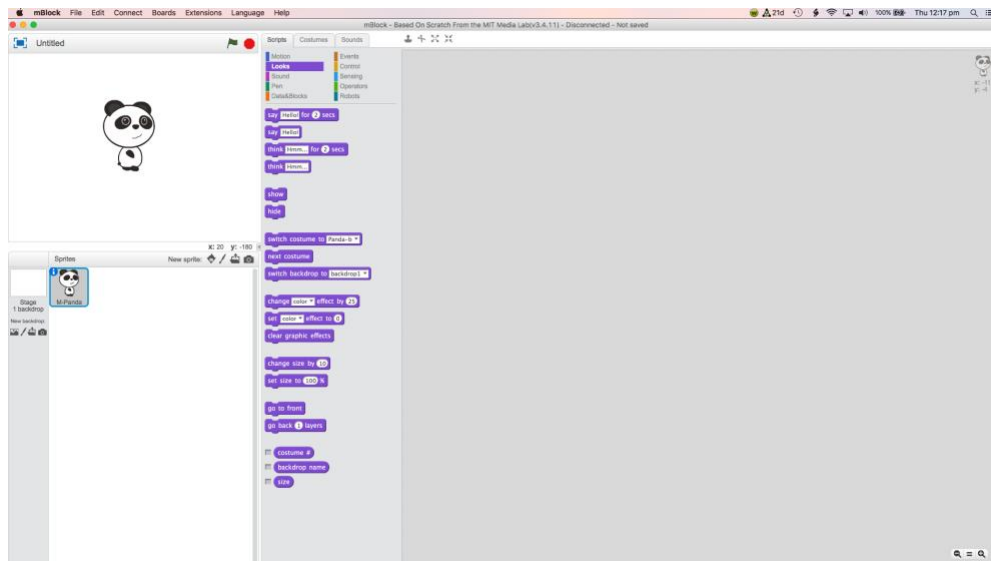

mBlock – A drag and drop coding platform for younger students

Coding is well within the capabilities of junior and middle primary students when you use an appropriate platform. Many schools use Scratch for simple coding activities. mBlock is based on Scratch, but has built-in capability to communicate with an Arduino.

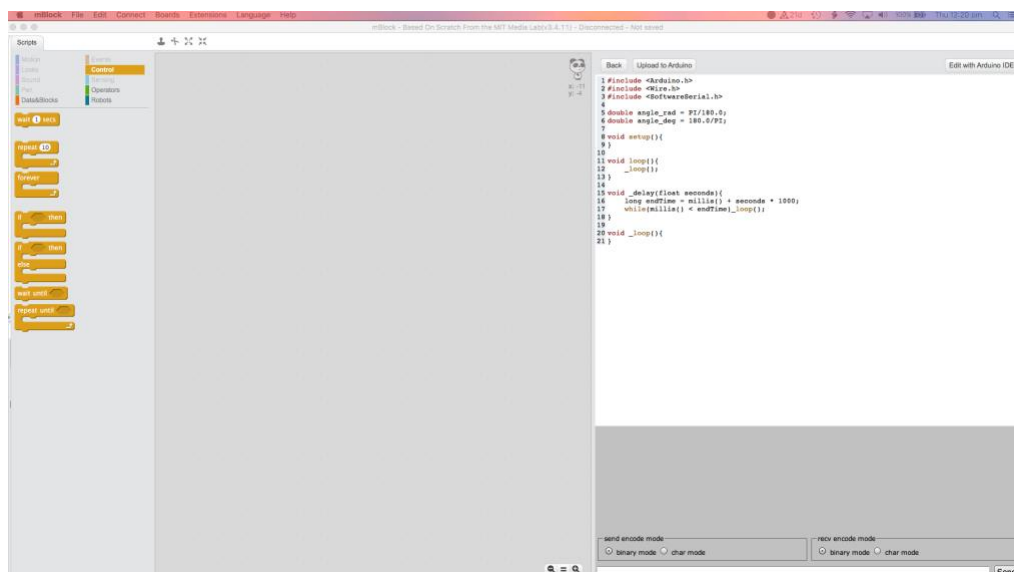
To download mBlock, open a web browser and go to <http://www.mblock.cc> . As with the Arduino IDE, it is possible to run a web version of mBlock or you can download a version according to your operating system.

Downloading the software means that you can run the software without needing to be connected to the Internet.

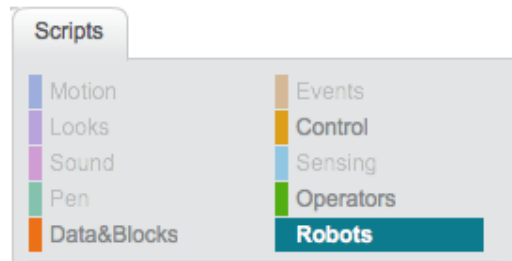
Once installed, run the mBlock program.



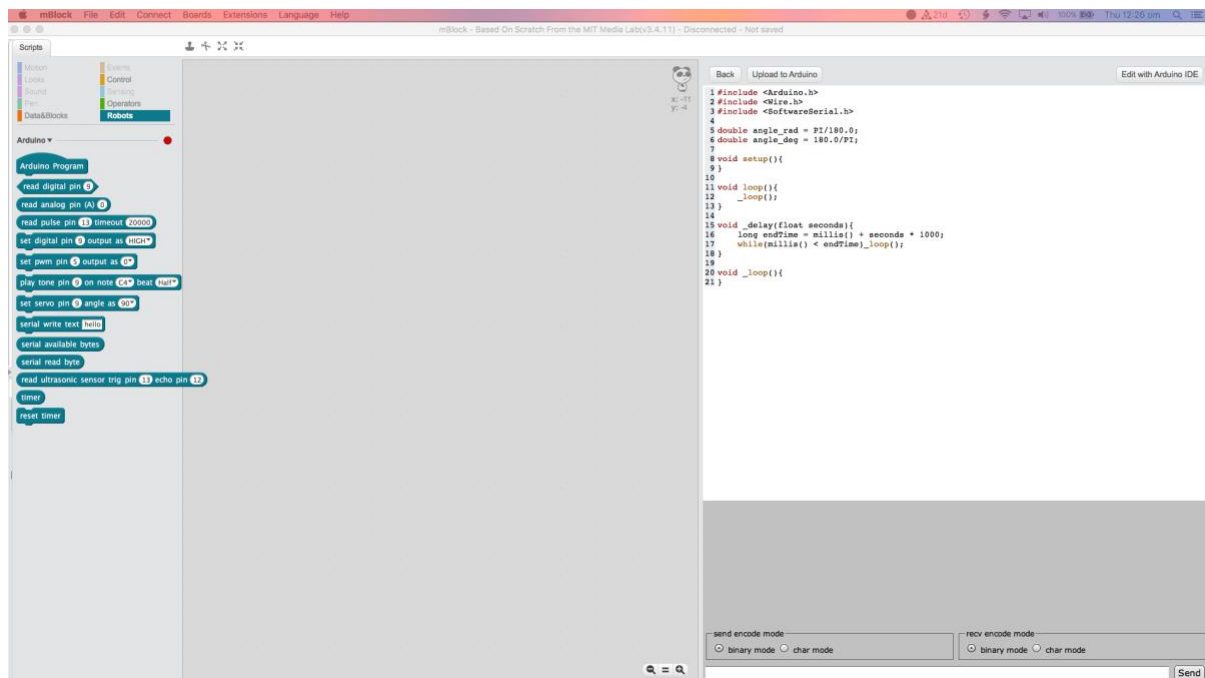
You will need to select the EDIT option on the menu bar and choose “Arduino Mode” and then ensure that under the EXTENSION option, choose the “Arduino”. Under BOARDS, select the Arduino UNO. The screen will change to resemble the following.



In the upper left corner of the screen are the options for the coding commands.



If you select Robots, you will see the basic commands that will be needed, regardless of whether you plan to program a robot or not.

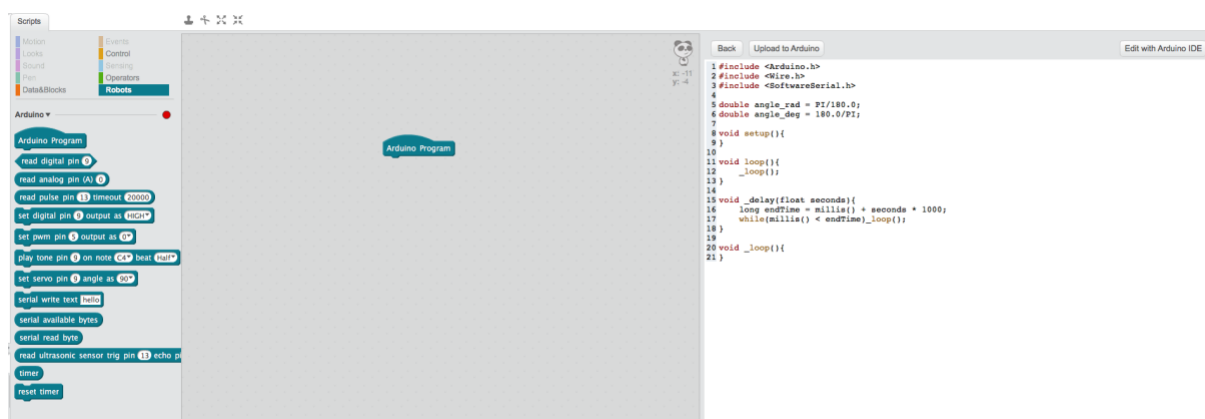


The screen is divided into three major sections. The left panel contains the instructions that we will use to build the block of code.

The far-right panel displays the actual code that will be uploaded to the Arduino and whilst young students do not need to write this code, it can help to be able to read the code and discuss it in class.

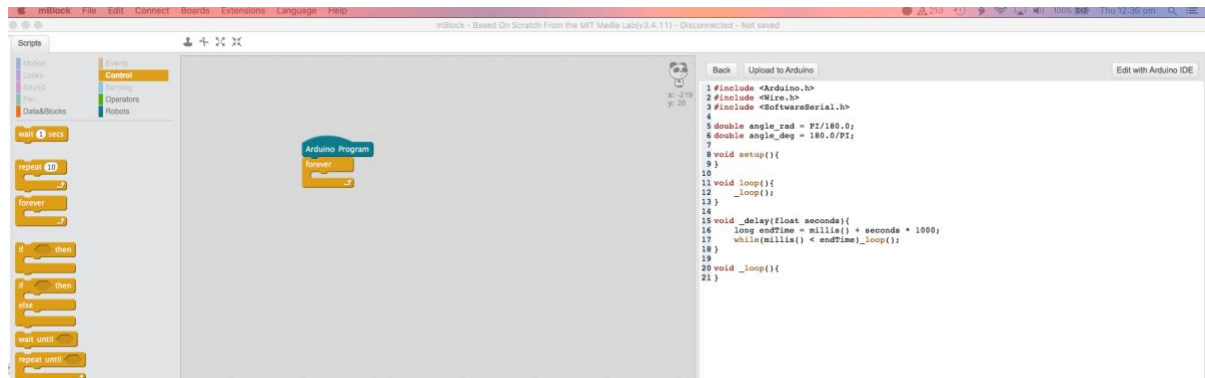
The centre panel is where we drop the instructions that we drag from the left-hand panel.

To begin any code for an Arduino, start by dragging the 'Arduino Program' block to the centre panel. This will tell the software that the code being built is to be sent to an Arduino and the software will then read the blocks and build the correct code in the right-hand panel.



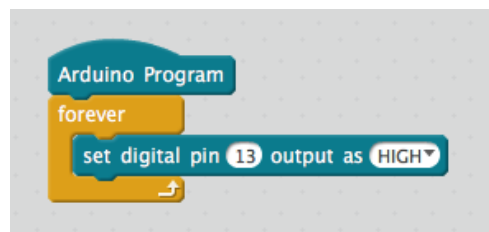
When building a block of code, we do not need to write the `setup()` block that we need to write in the Arduino IDE.

The `Loop()` block is the block of code than ‘runs forever’ when the Arduino is powered, so in the control section of the command options, you should find the “forever” command block and drag that into the centre panel and join it to the Arduino Program block.



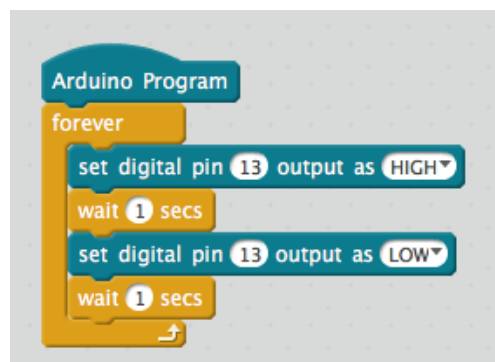
To code the Blink example that appears on page 10 of this handbook, locate the command to ‘set digital pin 9 output as HIGH’ and drag this into the centre panel and place it inside the ‘forever’ block.

We want to use pin 13, so click on the 9 and change it to 13.



When you place this block, you might notice that the code in the right-hand panel automatically updates to include a command in the `setup()` block and also includes a command in the `loop()` block.

In order to make the LED on the UNO blink, we need to leave the LED on for a second and then turn it off. We do this by adding a ‘wait 1 second’ block and then by adding a further block to set the digital pin to LOW. Another wait command and the code is finished and ready to be uploaded to the UNO.



To upload a block of code to the UNO, you must ensure that mBlock knows that the UNO is connected to the serial port. Plug the UNO into the USB port on your computer and then select **CONNECT** on the main menu. If mBlock has recognised the UNO and you have selected the serial port correctly, the top of the mBlock window will now display the words “Serial Port Connected” (It will also say “Not Saved” if the code has not been saved).

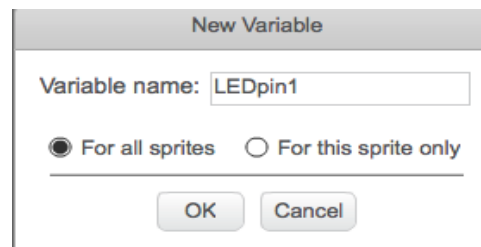
As long as the UNO is connected, you can click on ‘Upload to Arduino’ at the top at the right-hand panel and the code will upload. You should then see the built-in LED begin to blink every one second.

Should you want to refer to the actual code, the right-hand panel displays the full code and you should be able to identify the setup command that configures Pin 13 (the built-in LED) to be an output and the loop commands that turn the LED on and off.

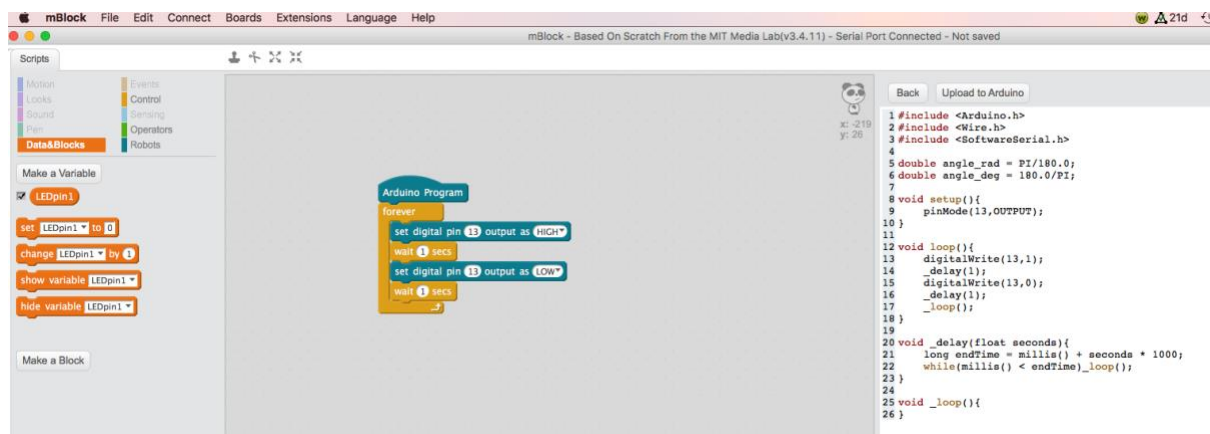
Now wiring up a breadboard as described on page 12 of this handbook will allow you to prove again that the external Pin 13 on the UNO is connected to the BUILT-IN LED.

With mBlock, we can also use variables as described on page 13 of this handbook.

In the ‘Data&Blocks’ section of the command options, click on “Make a Variable” and enter a name for your variable.

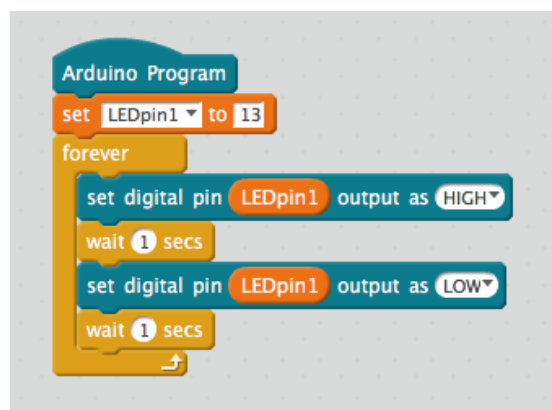


Click OK and the variable will now appear in the left-hand panel.



We can now set the pin number to be used as part of the setup and then anywhere that the variable name appears, that pin will be used. Note that the command to set the pin number does not need to be inside the ‘forever’ block, as it will happen before anything else begins.

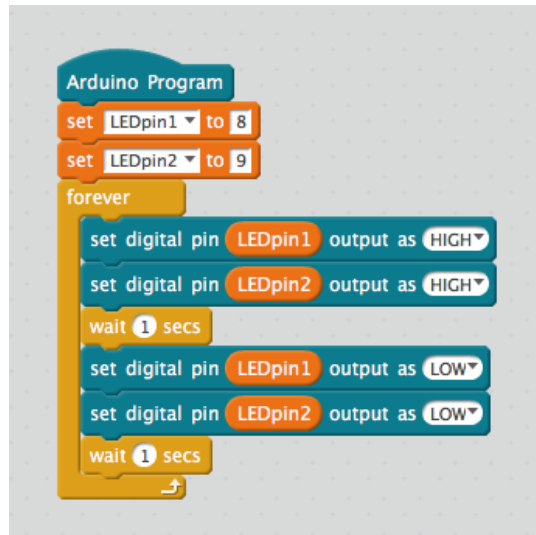
The resulting block will look like the following: -



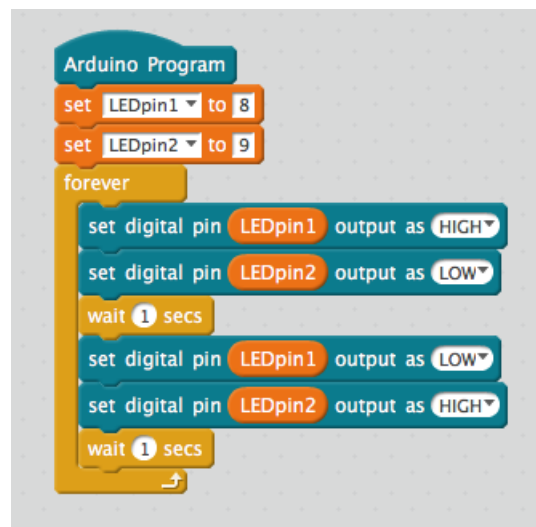
Follow the instructions on page 15 to configure three LEDs.

The first step with your code is to change the number of the pin assigned to the variable LEDpin1. Change it from 13 to 8 and upload to the UNO. You will need to make sure that the UNO is connected, as in mBlock it disconnects after uploading. Once you have uploaded you should see the LED connected to digital pin 8 begin to blink.

In order to complete the next stage of the exercise as per page 16, make a second variable and set it to be digital pin 9. Add the lines of code to turn on LEDpin2 and to turn it off again. The code should look like this.



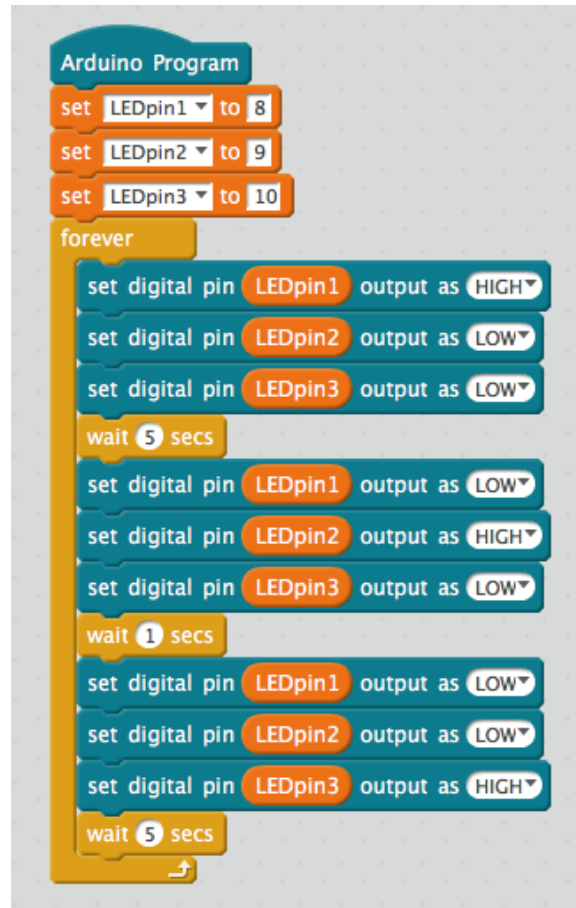
To make the LEDs blink alternately, the code needs to be changed to: -



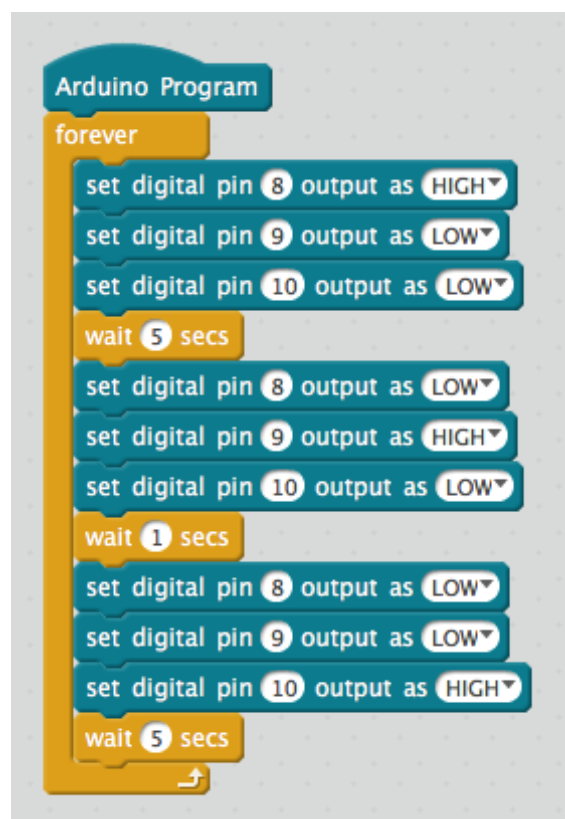
To complete the traffic light sequence you need a third variable, LEDpin3 set to use digital Pin 9, and now you will need an additional block inside the 'forever' block.

For some younger students, the concept of a variable over-complicates the exercise, and so the final code for the traffic light sequence is presented here both with and without the use of variables.

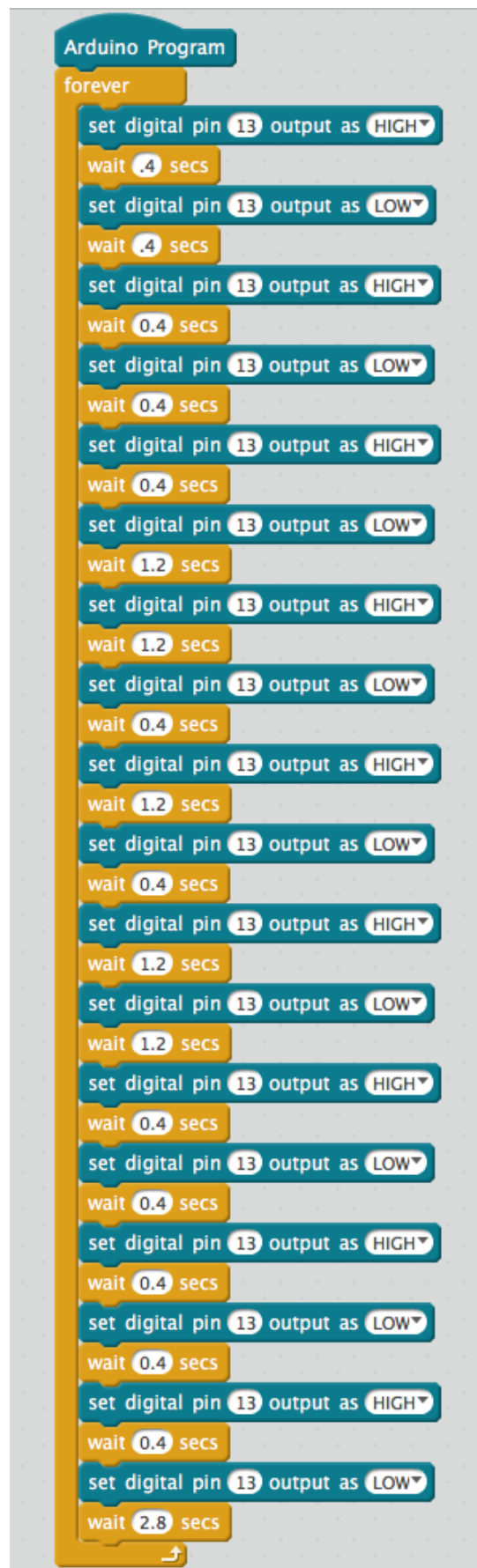
With Variables



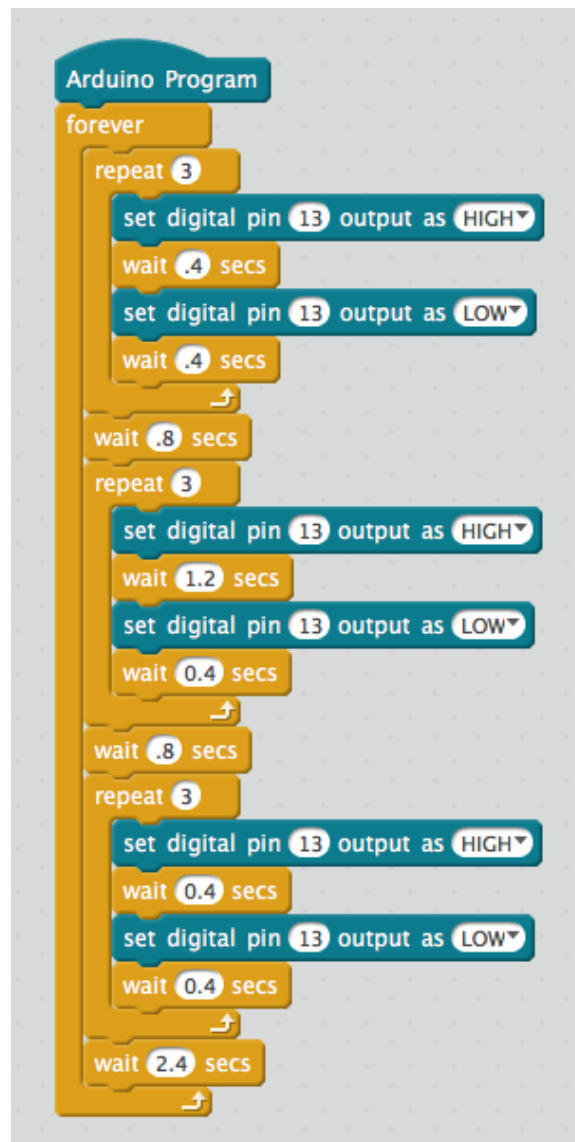
Without variables



The Morse Code extension code that appears on page 23 can also be generated within mBlock and the finished code is shown here.



If a particularly able student were to tackle the page 23 challenge to use FOR statements to re-write this code, in mBlock that would mean using a 'repeat' control. A solution to the challenge might resemble the following.



On page 25 of this handbook, we look at the capacity of the UNO to use sensors. With simple sensors, the mBlock is still a suitable interface, but with the more complex sensors, where the developer has provided a library that needs to be installed, mBlock is no longer a suitable platform. The first of the sensors that is considered is the photo (light) sensor.

Follow the directions on page 26 to connect the various components for this exercise.

At the top of page 27 is a block of code that simply reads the data being collected by the photo sensor and then prints it to the serial port of the computer.

In mBlock the block of code to do this is as follows.



The Arduino IDE has the capacity to display this to the screen, but mBlock does not, so we need to find an alternative way to read the data that the photo sensor is producing. To do this, we use an additional piece of software called a terminal emulator.

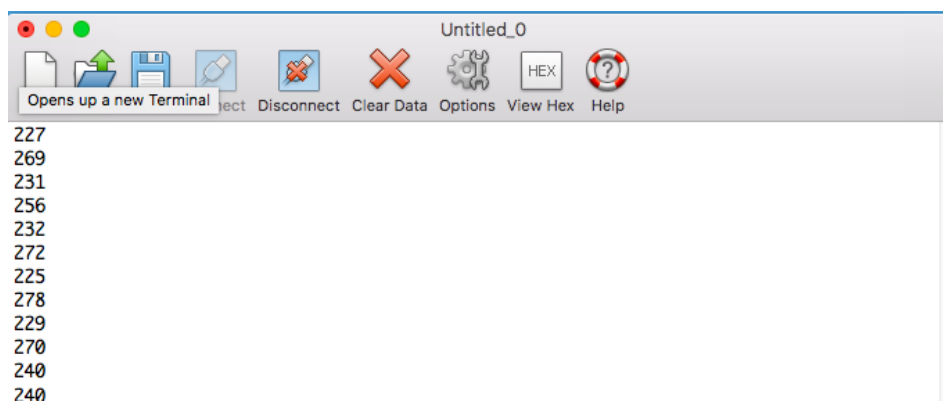
Open a web browser and visit the following web site.

<https://learn.sparkfun.com/tutorials/terminal-basics/coolterm-windows-mac-linux>

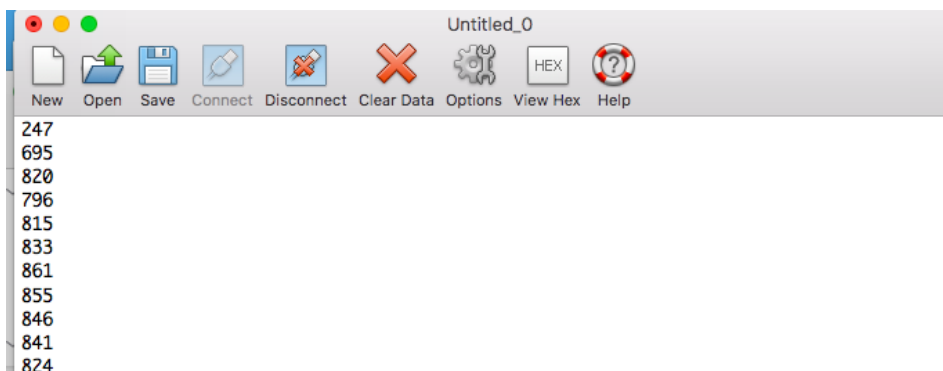
Here you will find a link to download a software program called 'CoolTerm'. CoolTerm has versions for any operating system. Install and open CoolTerm.

Click on the Options icon and if the Serial Port displayed is not correct, select the correct option and the correct baud rate and then click OK. Click the File, Save as Default option to save the configuration for next time.

Make sure that the UNO is connected in mBlock and upload this code to the UNO and then click in CoolTerm and open a connection by clicking Connect on the toolbar. You should now see the data that is being printed to the serial port appear in the CoolTerm window.



Cover the photo sensor and observe the change in the numbers displayed by CoolTerm.



Depending on the particular photo sensor, the values might increase or decrease as you cover and uncover the sensor. In the example illustrated here, the values increased as the sensor was covered. We use this information to add an IF statement to the code to turn an LED on or off according to the value coming from the photo sensor.

In the case illustrated, we would want the LED to turn on when the value is greater than 300.

In mBlock, you build an IF statement by using blocks from three different sections of the command blocks.

First you drag an IF...ELSE block into the centre panel from the 'Control' section.

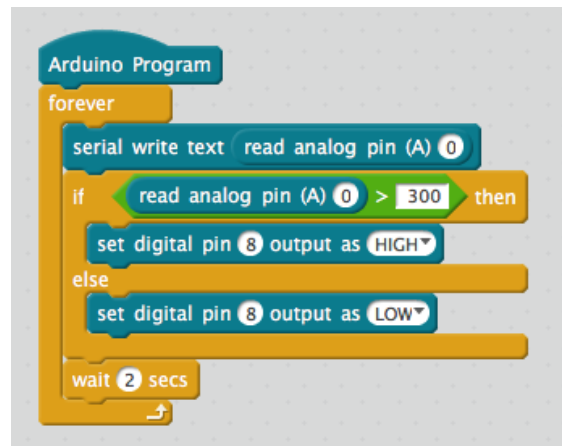
Then drag the appropriate comparison block from the 'Operators' section and drop it into the IF...ELSE block in the top space.

Next, you populate the operator with the command to read the Analog pin and the value against which you wish to make the comparison.

In the remaining two spots within the IF....ELSE block, drag the instructions to turn the LED on and off.

Don't forget to include a wait statement after the IF...ELSE block to allow a pause between readings of the photo sensor values.

Thus, the code block to achieve this has two parts, one to read the sensor and one to control the LED. If you have followed the instructions carefully, then your mBlock code will be as follows.



Upload this code to the UNO and as you cover the phot sensor the LED should now turn on and as you uncover the phot sensor the LED will go off again.

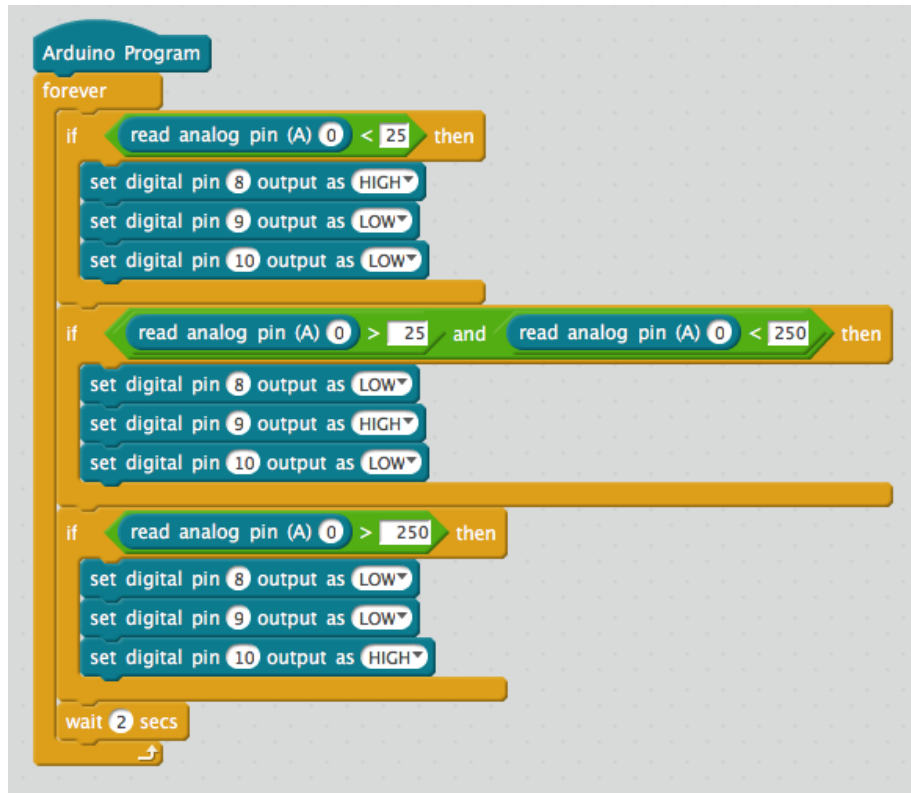
Using the same logic and process that has been used to calibrate the night light, we can calibrate an analog soil moisture sensor as described on page 33 of this handbook.

This allows us to create the necessary code in mBlock to control the three LEDs based on the level of moisture in the soil.

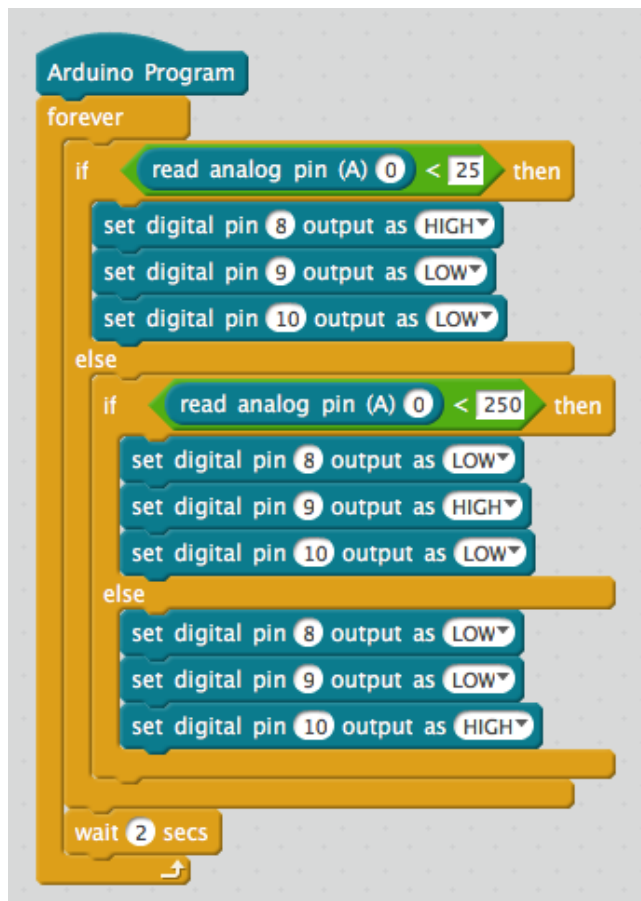
You could determine these values to suit your own definitions in class with the students, but for the sake of brevity, I am going to assume that a value of less than 25 indicates dry soil, a value between 25 and 250 indicates moist or damp soil and a value over 250 indicates that the soil is wet or very wet. Using these values, the mBlock code will be as follows.

There are several ways of achieving the result, so two possibilities are provided here, perhaps providing basis for a class conversation as to how and why they differ.

Solution 1



Solution 2



Operators

Operators in coding can be broken into three categories, those that compare values, those that assign (or set) values and those that join together statements as part of logical statements.

Assignment operators

- `+` Adds a value; example `y = x+3` adds 3 to x and assigns the answer to y
- `-` Subtracts a value; example `y = x-3` subtracts 3 from x and assigns the answer to y
- `*` Multiplies two values; example `y=x*3` multiplies x by 3 and assigns the answer to y
- `/` Divides one value by another; example `y=x/3` divides x by 3 and assigns the answer to y
- `%` Modulo; example `y=x%3` divides x by 3 and assigns the remainder value to y
- `++` increments the value of a variable by 1
- `--` decreases the value of a variable by 1

Comparative operators

- `==` compares two values to see if they are equal
- `!=` compares two values to see if they are not equal
- `>` compares two values to see if the first is greater than the second
- `<` compares two values to see if the first is less than the second
- `>=` compares two values to see if the first is greater than or equal to the second
- `<=` compares two values to see if the first is less than or equal to the second.

Logical operators

We can join statements together using the following operators.

- `&&` means the first statement and the second must both be true
- `||` means either the first or second can be true to satisfy the requirement

A third and more confusing operator is the `!` symbol.

This is used to indicate that what follows it is either zero or is a statement that must be false

IF statements

If statements commonly check for a specified condition, and if that condition is met, executes the statements that occur within the If statement.

Example:

```
If(val<100){  
digitalWrite(13,HIGH);           // turns on pin 13 if val is less than 100  
}
```

IF..ELSE statements

Like an If statement, except that this statement includes an instruction to perform if the If statement condition is not met.

Example:

```
If(val<100){
digitalWrite(13,HIGH);
digitalWrite(8,LOW);
}
else{
digitalWrite(13,LOW);
digitalWrite(8,HIGH);
}
```

// turns on pin 13 and turns off pin 8
// if val is less than 100
//but if val is greater than 100,
//turns on pin 8 and turns off pin 13

FOR statements

For statements are used to control when a statement or series of statements should be executed.

Example:

```
for(int i=1; i<4;i++)
{ digitalWrite(13,HIGH);
delay(1000);
}
```

//says start with i=1 and increment I by 1
//until I gets to 3, performing the
//following statements
//for each value of i

WHILE statements

While statements

Example:

```
Int i=0
while ( i<4)
{ digitalWrite(13,HIGH);
delay(1000);
i++;
}
```

//is the same as the above IF statement


```
while ( analogRead(0)<100)
{ digitalWrite(13,HIGH);
}
```

//says read analog pin 0 and if the value is
// less than 100, turn on Pin 13 until such
// time as the value exceeds 100 and only
// then will the WHILE loop exit and the
next
// line of code be executed.

These statements are by no means exhaustive but are sufficient for the vast majority of what the average primary or middle years student would need to create quite comprehensive and powerful code.

NOTES