Intelligent Machines Workbook #1

Micro-controller

Table of Contents

MIT Licence

Brief summary:

Permissions (what you can do)

Commercial use
Modification
Distribution
Private use

Limitations (what is not covered)

Liability
Warranty

Intelligent Machines

I will delve into this in more detail as we go, we will start off just getting the LED to blink and then introduce other components and sensors. This is not a thorough exploration of robotics alone but rather a merging of robotics and AI. It will be pretty simple at this stage but challenging none the less.

We will be putting a small neural network inside the micro-controller and also control a servo with the movement of a finger. So I will recommend that you work through the Algorithmic Art and Algorithmic Intelligence workbooks before you start this series of workbooks.

The aim is to make simple machines a little bit more intelligent than just sensors or motors or displays. Using AI can connect this tiny devices and suddenly make them very powerful rather than glorified adding machines, and before you get worried, they are not going to be sentient any time soon, not even the largest data centre models are remotely sentient.

How to use this curriculum

This is all about robotics, using a micro-controller. Some call it physical computing (coding) and that is what it is, you are stepping outside of the monitor and interacting with physical entities and environments, sensors, motors and displays. It is fun and fascinating at the same time.

This is not about building a humanoid robot that can help you with loading the dishwasher. We are talking about low cost components, sensors (button), simple motors (servo), displays (LEDs) etc.

It is assumed that you have already covered Algorithmic Art and the Artificial Intelligence workbooks. This, however is not entirely essential but very much highly recommend otherwise I would have to repeat much of both of these workbooks.

If you haven't coded before go back and work through the Algorithmic Art and AI workbooks they will make it much easier for these workbooks on robotics. If you are pretty competent at coding and have a good understanding of AI or Machine Learning then you could dive straight in and just go for it.

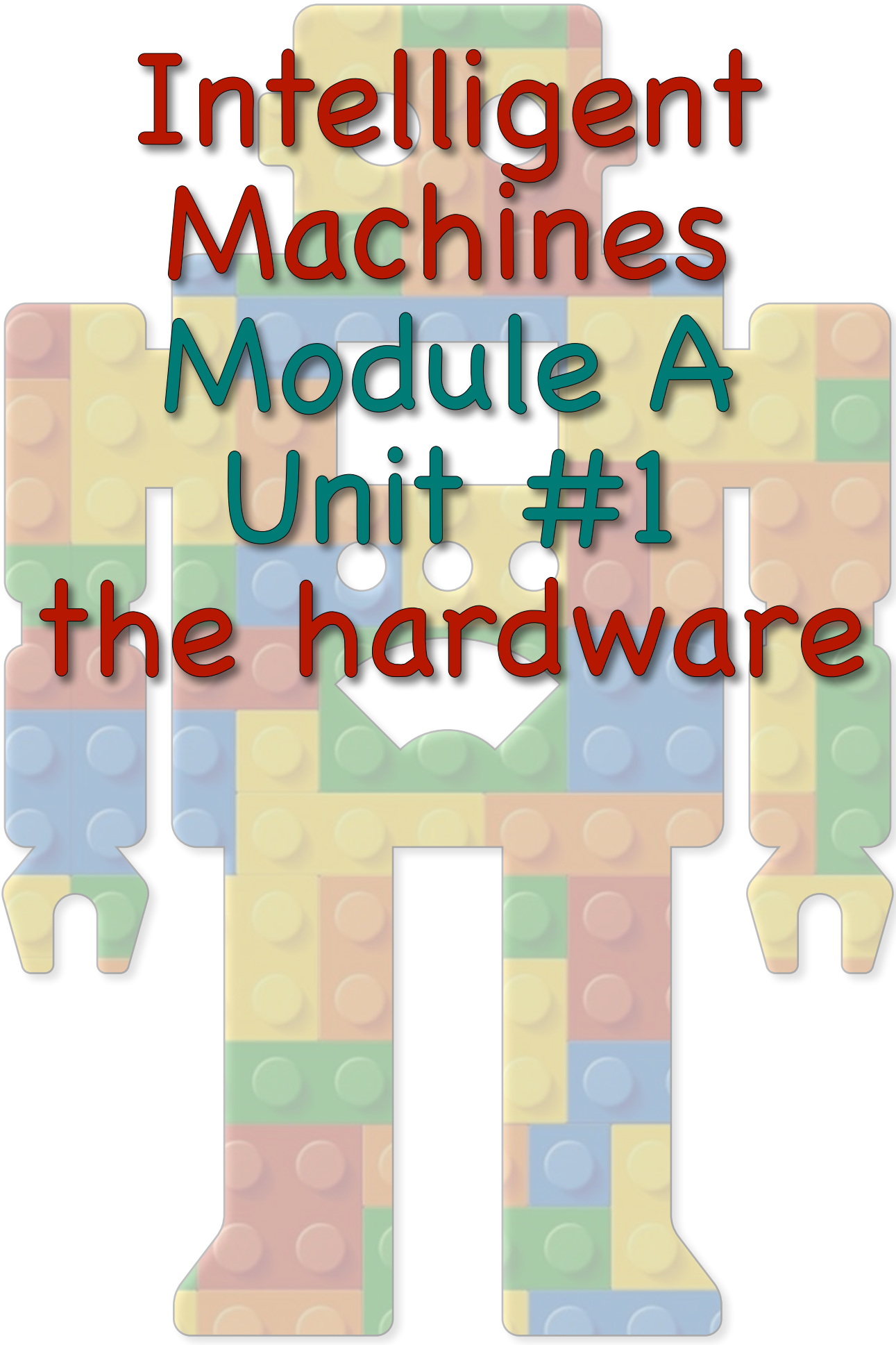
Currently, there are three workbooks, each one building on the other. For the p5.js and ml5.js bits you can do that in the browser, with the Arduino you will need to download the IDE software to run your programs (sketches). In the future I plan to have some more which are based on the ESP32 boards and cover, bluetooth and WiFi.

Unless you have two monitors, or a computer/laptop and another device to display the workbooks then I suggest downloading them and printing them off. They are pdf's so you could display them on a device. Work through each unit of each module of each workbook in turn. Take your time. Check connections in your wiring, check you have wired the circuits correctly.

Workbook #1 Quick Content Summary

This first workbook is to introduce you to the very fundamentals of coding robotics with the minimum number of components (and thus cost). It will also cover the basics of the code used with the Robotics which is C/C++. This is strikingly similar to p5.js. if you have already followed the other workbooks the the coding will be very familiar, although it is not exactly the same.

Read through the first two sections quickly but carefully before buying the components and before downloading the software. A really good place to buy the components is the Pi Hut which I believe can be accessed in the USA as well as in the UK and I presume the continent as well.



Intelligent Machines
Module A
Unit #1
the hardware

 Module A Unit #1: The Hardware

This first unit will describe all the components that will need for workbook #1 and workbook #2. For workbook #3 we will introduce an additional component, the servo. I have endeavoured to keep the amount of hardware needed to absolute minimum, as this is more about the coding than the robotics at this stage.

The main piece of equipment you will need is the micro-controller itself. For the first three workbooks we will using the Arduino Nano 33 BLE micro-controller board.

Important Notice:

The board we will be using in this series of modules and units is the Arduino Nano 33 BLE. It is the newer version 2. You can still get hold of version 1, but they are not readily available anymore. The difference between the two is the IMU. This is where it measures acceleration, rotation, and the magnetic field. The version 1 of the board uses the LSM9DS1 module, but version 2 uses the BMI270 and BMM150 modules. They do the same job but the latter are better apparently.

Make sure that you get the board that has the pins already soldered, you can buy them without pre-soldered pins, so take care when buying.

Devices
These are often microcontrollers that are embedded in a smart device. A common example may be something like a smart speaker. An example of a microcontroller is the Arduino Nano 33 BLE. Some of these devices can connect to Wi-Fi and transmit data from sensors.

Sensors
You can add a very wide variety of sensors. Whether motion, sound, light, etc.

Connectivity
This often forms a network of devices that are connected together through Wi-Fi or Bluetooth. They can communicate together and form a mesh network, or communicate independently to a central collection.

Data Processing
This is where the data collected by the devices is processed and analysed. This can be done on the devices themselves, in the cloud, or on a combination of both. This may also be linked to a machine learning algorithm which can act on the data immediately.

User Interface
A device can be connected directly through cables or Bluetooth. This makes the device very versatile, and the Arduino Nano 33 BLE is brilliant for this.

Processing
This is where Arduino Nano 33 BLE boards start to earn their keep. They have enough processing power which allows you to run, albeit relatively small, neural networks on them. Although they are nothing compared to LLM or big data models, they can still have a place. They have an extremely low energy requirement.

The ESP32-S3 boards do have considerably more memory than the Arduino boards, but for this exercise, I have found the Arduino Nano boards to be a bit more robust in using the Arduino Code Editor (IDE).

More than a humanoid robot

Definition

If you think of the word robotics, what does it make you think of? A killer humanoid machine, a robotic arm in a factory, even a robotic dog called Spot? Invariably, it is something that moves, completes tasks, and may or may not be humanoid, although a number can interact with you as if they were human.

More than a robot

The term robotics is more than just another word for robots; it encompasses not just motors but also sensors and displays. Another term that we could use is Physical Computing. Everything that is more complicated than a torch (flashlight) has a microchip in it and has therefore some code embedded in it.

The heart of a robot

At the heart of robotics is the microcontroller. It is the hardware that the code runs on. It is usually what looks like a very small black tile on a printed circuit board. There are, as you might expect, many different types. Each having varying functionality but in essence, do almost the same thing. They control very small electrical impulses through tiny switches or gates.

Purpose

We won't get too deep here because we are more focused on how to use these chips. For that, we have a wide variety of microcontrollers to choose from. We can wire up our motors, sensors, and displays to make them serve a particular purpose; your car, microwave, washing machine, TV all use them. They monitor what they are doing, feeding back information, allowing you some control to programme them, choose a set programme, or switch channels, for example.

Hard code

The machines (TV, cooker, etc.) need human input through pressing buttons, turning dials, or even speaking to them. These simple components are connected to the microcontroller, and that is where the coding comes in and responds to the press of a button and the turning of a dial.

Code that learns

What if instead of being a simple machine, we give it a brain and train it to perform tasks that might be more challenging to hard code? Machines that have learned or can learn, develop, and improve. That is what we are doing with intelligent machines.

Which micro-controller?

Too much choice

I alluded to the fact that there is a bewildering variety of microcontrollers. They all have their merits; some cost a few pounds/dollars/euros, but others seem to be considerably more expensive, although to be honest, nothing is ridiculously expensive.

TARDIS-like

The problem is not the choice but what we are going to do with the microcontroller; we are going to put a neural network inside the chip. Although it is going to be a very small neural network model, it will still demand a lot of memory. Most microcontrollers, by their very nature, have very small memory chips.

Introducing the Nano family

The microcontroller is usually the size of your index finger (I know people's index fingers vary in size, but you get the general idea), so you can't expect to run Windows on it. The good news is that there are some microcontrollers that do have enough memory, and one I particularly recommend is the Arduino Nano 33 BLE.

Other boards

If you demand much more memory, then I suggest an ESP32-S3 (Arduino do a Nano version of the ESP32-S3), but also XIAO do a tiny board the size of your thumbnail. Another board you could use is the Arduino Nano 33 BLE Sense v2, which has a large array of sensors but is twice the price. Something like an Arduino Uno just doesn't have the memory space.

What's on the board?

What does it have?

The Arduino Nano 33 BLE has a trio of sensors: an accelerometer, a gyroscope, and a magnetometer. It also has a built-in LED and RGB LED, which we can use. It also has a lot of pins to which we can connect different components, such as other sensors, displays, and motors.

Bluetooth

It has Bluetooth (but no Wi-Fi), which we can make use of by connecting several of these devices together in a mesh network.

So what?

Wearable tech and smart appliances make use of the latest developments in microchip technology. Some appliances have elements of AI in them, for instance, wake words for smart speakers.

Small is beautiful

Although AI (or machine learning) is predominantly the domain of larger machines that can handle a large model (trained Deep Neural Network), it is possible to install a small truncated Deep Neural Network on a small microcontroller if it has enough memory space.

Elegant solutions

This gives the opportunity to develop a range of applications and gadgets. But first, we need to have some understanding of how they work, and the Arduino Nano 33 BLE is a good board to start with. Not only does it have enough capacity, it also has lots of pins we can connect components to.

It is a starting point

In the end, you are unlikely to use it for anything commercial. There are other chips and form factors you would likely use being manufactured on a circuit board, but that is for another day.

Bigger is not necessarily better

Most talk about AI focusses on Large Language Models, which are costly in terms of GPUs (data centres), power (cooling and running). We are going in the opposite direction; these use very little power and have to be efficient. They will lack the breadth of an LLM, but that doesn't mean they are redundant. I predict we will see more development in this area, especially if the interest in LLMs wanes or the AI bubble bursts.

You can still be a player

Although only big tech companies can handle LLMs, they cost billions just to keep running. Embedded AI is accessible to anyone as long as you don't want to take on ChatGPT. Saying that these microcontrollers can be linked to an LLM or SLM (Small Language Model) and harness their power if you so wish, we may come onto that later.

How to get started

Ecosystem

The first thing is to understand the ecosystem. We are using an off-the-shelf board. We are going to use the Arduino IDE to programme the microcontroller. I will assume for this that you have completed the section on `p5.js/ml5.js` of this AI tutorial. We will be using that knowledge later on.

First things first

First, we need to connect our Arduino and learn how to programme it, looking at all its features. If you are familiar with programming a microcontroller, such as the Arduino Uno, then this is familiar territory. You could skip some of module A, but there may be things that are new to you, and in any case, it won't hurt to practice. Although the next couple of units will go into more detail about the hardware and software you will be using, I will give you a brief summary here.

Hardware

Mostly, you will need an Arduino Nano 33 BLE with soldered pins, which is important, a half-size breadboard, and a micro USB cable. As you get near to the AI bit, you will be introduced to buttons and how you can connect them to the Nano.

Software

To programme the board, we need to use an IDE of some sort. It is what we use to upload the code. We do have a choice. We could use the cloud version of the IDE, which is good for a number of reasons. However, I won't be using it, simply because the free version only allows you 25 uploads per day. Instead, we will use the downloadable version of the IDE (version 2.x) and use that. You have unlimited uploads! The downside is that you have to manually update boards and libraries as well as the IDE version (if there are any updates). You are informed when there are updates.

Computers

You can use any Mac, Windows PC, laptop, or Raspberry Pi or Linux operating system.

Chromebook woes

If you are using a Chromebook, you could use the Arduino Cloud, but you will have a problem (at the time of writing). The Arduino Nano 33 BLE is not compatible with the Arduino Cloud on the Chromebook. Hopefully, it will be in the future.

Hardware you will need

Which board

The main bit of kit you will need is a board. There are quite a few to choose from that are either made by Arduino or other companies. Using Arduino Nano 33 BLE boards is a very popular option as they are cheap (≈£25/€27) and easily available.

Tight budget

If you are on a very tight budget, you could also go for the XIAO ESP32-S3, which is the size of your thumbnail.

Minimal

I have endeavoured to keep the amount of electronics to a minimum in this tutorial and focus more on the code and the machine learning in particular. However, the whole point of microelectronics is that you use components and create some wonderful piece of technology or gadget.

Components

To start to develop a good understanding of how you can use the Arduino Nano 33 BLE board, I will introduce you to a small number of components. This will give you a sense of how to integrate them into a project.

Hardware bits and pieces

So here is a list of hardware (components) you will need. They are easily sourced on eBay, Amazon, or specialist stockists like PiHut or Pimoroni.

- Arduino Nano 33 BLE (with headers)
- Micro USB cable (to connect it to your computer)
- Breadboard (need a half-size)
- Buttons (push, momentary, tactile)
- Jumper leads (male-to-male sort to connect the components on the breadboard)

Arduino Nano 33 BLE

Soldered pins

Important, get the Arduino Nano 33 BLE with pins already soldered on unless you are a dab hand at soldering. Unlike the Uno, which operates on 5 volts, the Arduino Nano boards work on 3.3 volts. Some components are 5 volts only, some are dual-use, and some are designed for 3.3 volts only. Always check the specification.

Automation

With the built-in Bluetooth module, the Arduino Nano 33 BLE can communicate with other devices and sensors. This makes it suitable for a wide range of applications, such as home automation, remote monitoring, data logging, and more.

The IDE

The board's compatibility with the Arduino Integrated Development Environment (IDE) also makes it easy for beginners and hobbyists to get started with programming and electronics. The IDE provides a simple and intuitive interface for writing and uploading code to the board, as well as a large community of users and resources for learning and troubleshooting.

Digital and Analog

The Arduino Nano 33 BLE has a series of pins running along each edge of the board. Roughly one side is for analog components, and the other side is for digital components. Where the prefix A is for analog and the prefix D is for digital, followed by a pin number. There are other pins which have various functions; the ones we will be making use of will be the GND or ground pins.

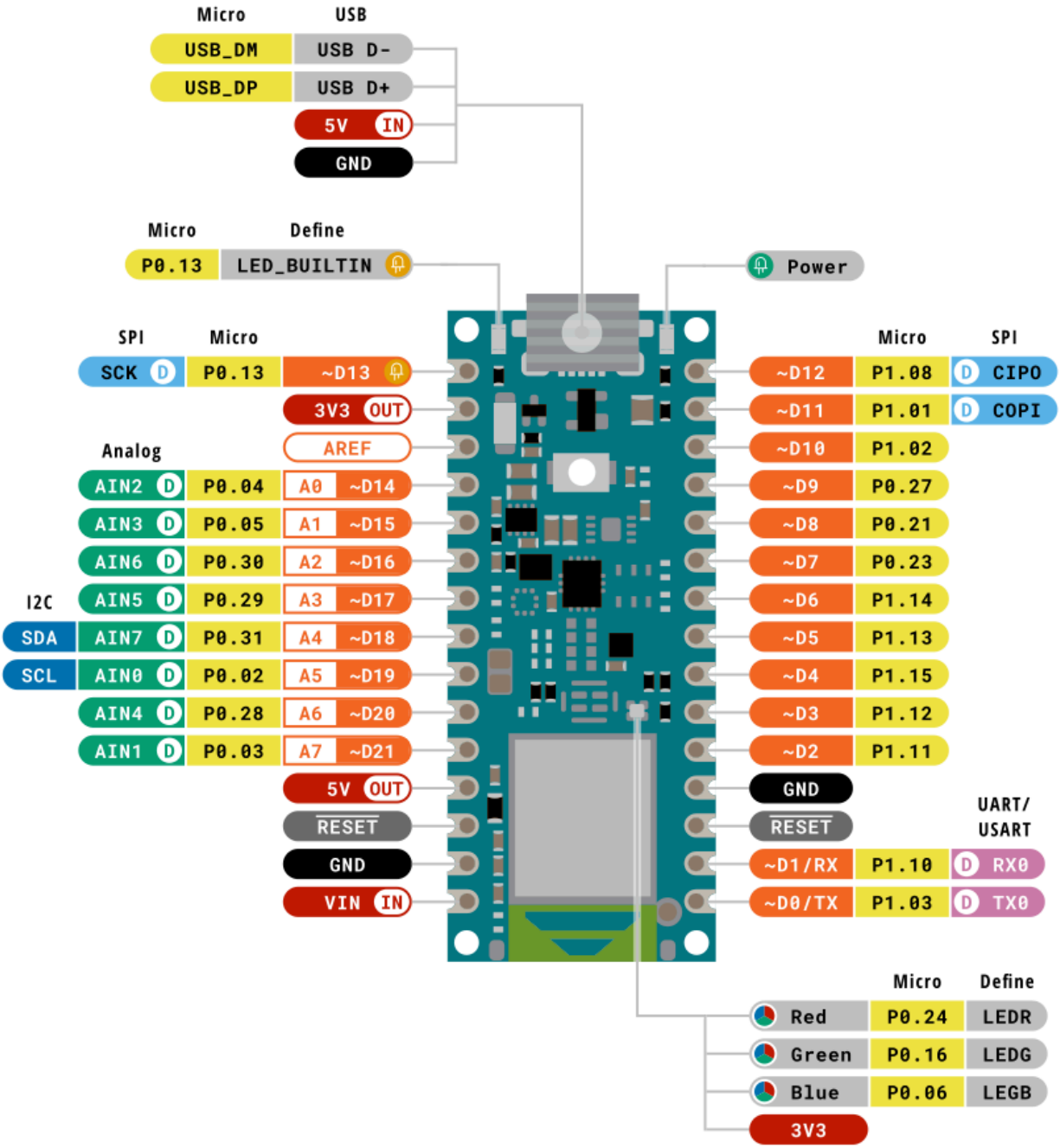
IMU

We have access to the accelerometer, gyroscope, and magnetometer. This is something we will make use of later. If you have a board (such as the ESP32), then you will have to buy a separate module. This is not a great problem but is something you will have to navigate. Hence why I have gone for the Nano 33 BLE.

Pin diagram

To work out which pin is what, you will need to use the pin out diagram. The pins are numbered on the front (very tiny). The analogue pins can also be used as digital pins if needed, see fig.1.

Figure 1: pin out



Get to know your board

Here are some images of the board.

Figure 2: oblique view

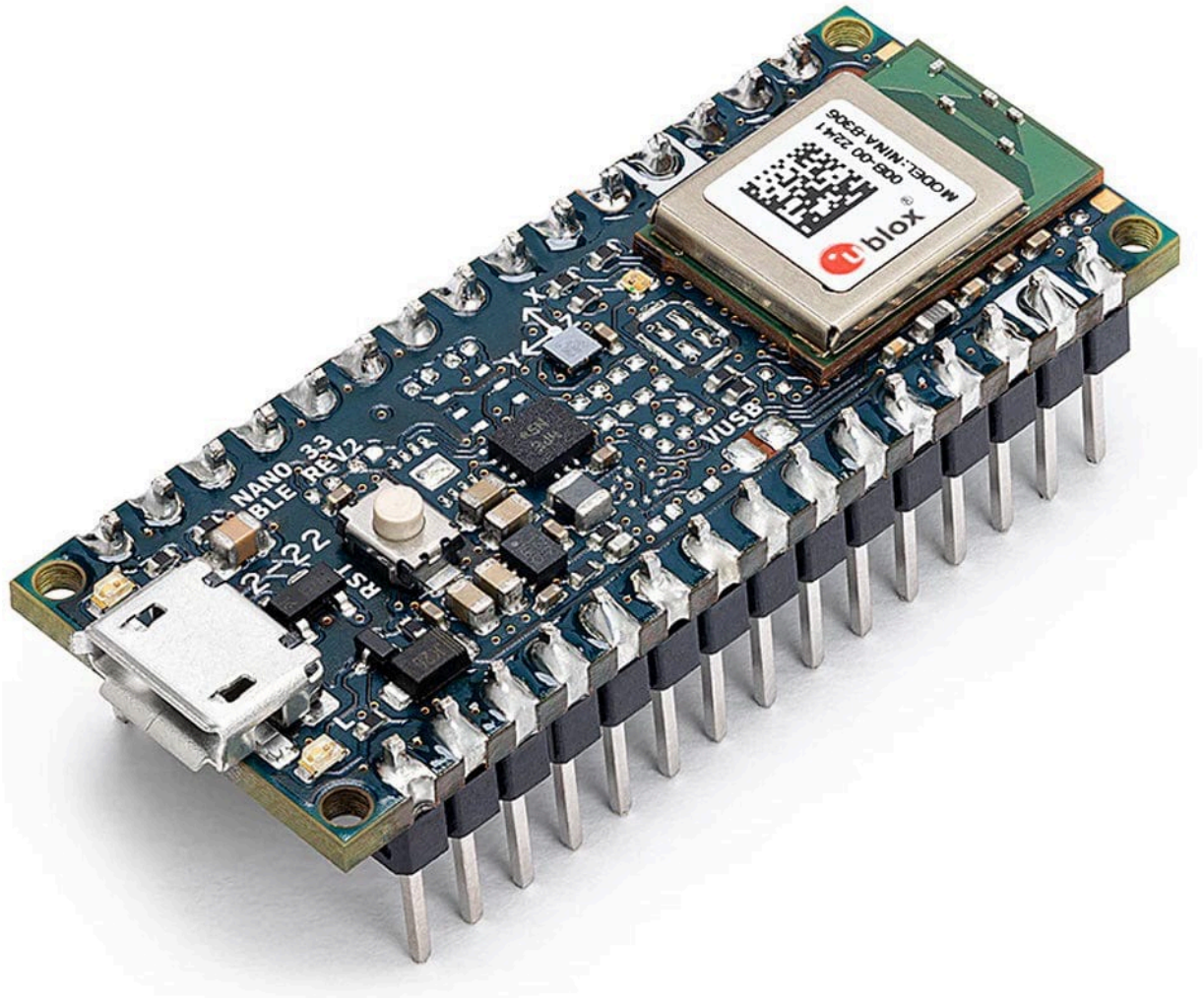


Figure 3: top view

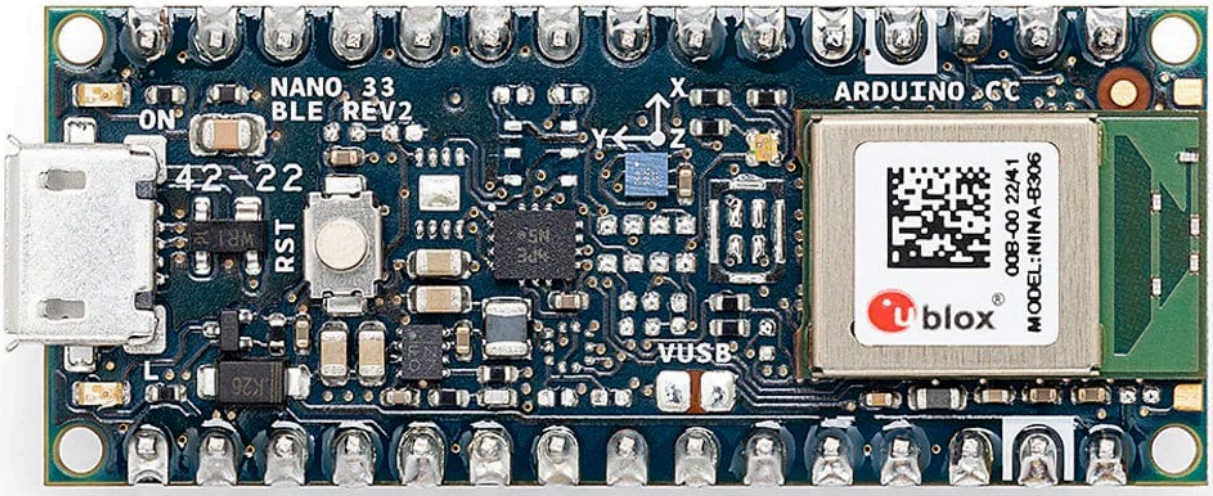
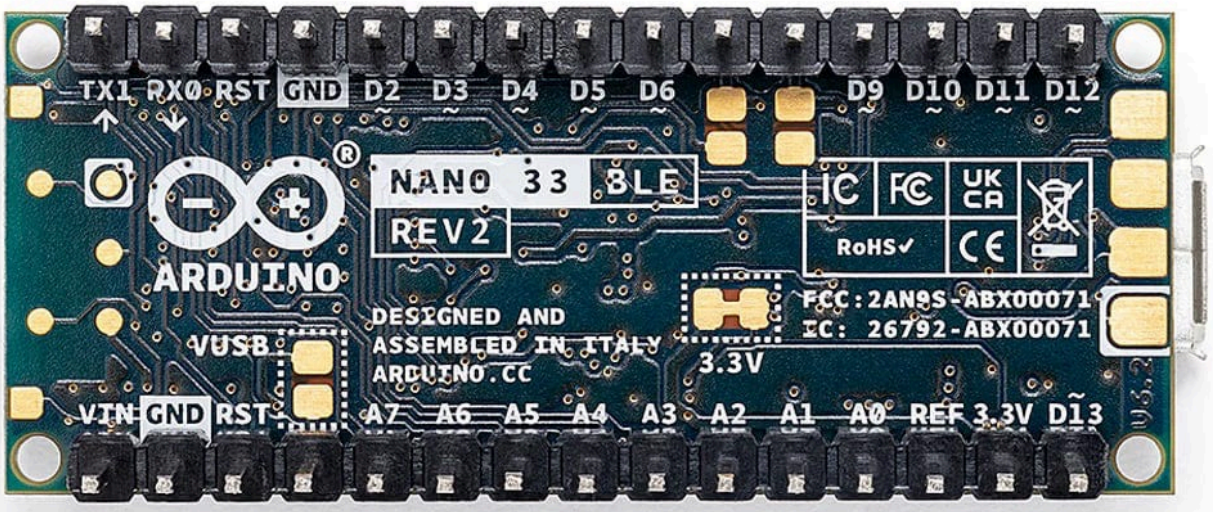
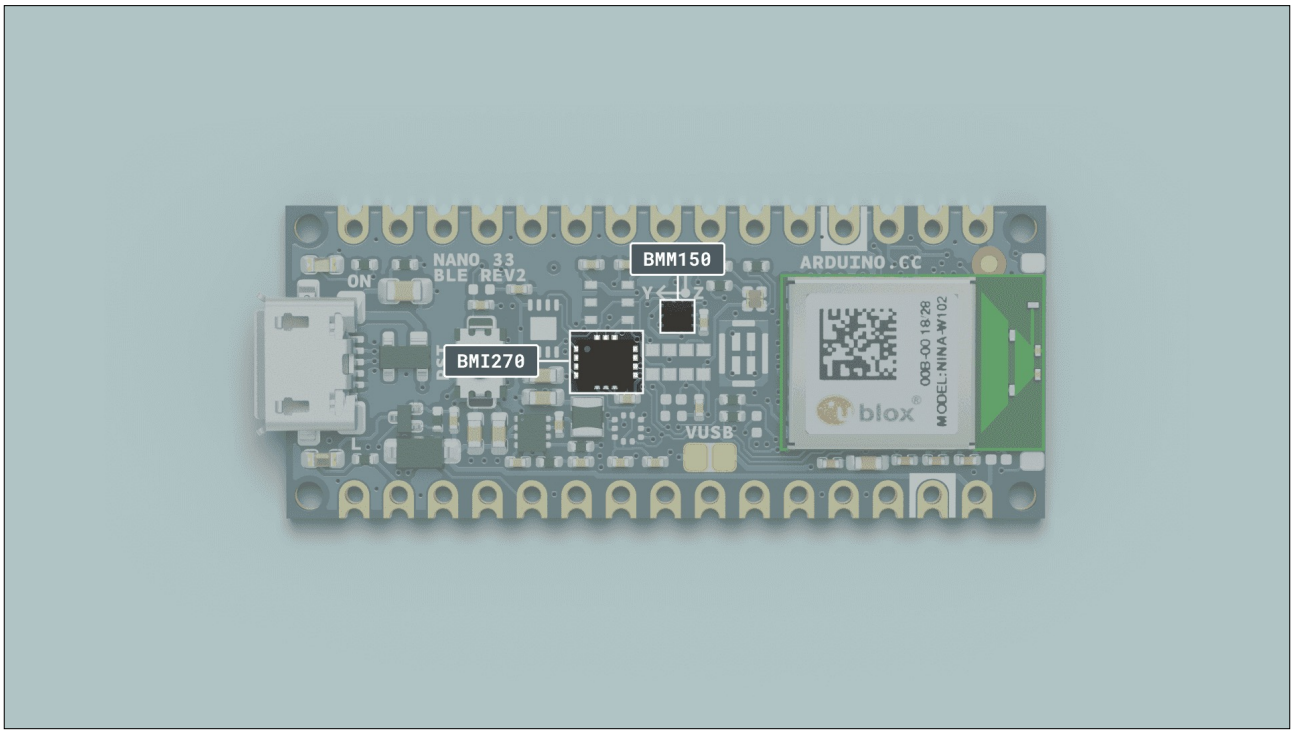


Figure 4: underneath



Below is an illustration of the IMU on the board. There are two sensors: the BMI270, which is the accelerometer and gyroscope. The other sensor, BMM150, which is a magnetometer that measures the magnetic field. More on those later.

Figure 5a



USB cable

To connect your Arduino Nano 33 BLE to your computer and to power it, you will need a micro USB cable (shown below). Make sure you get one that can do both of the following:

1. To power the Arduino Nano 33 BLE, from a 5-volt source such as a normal phone charger or battery pack.
2. It sends data to and from the computer and the Arduino Nano 33 BLE, such as uploading your code.

Figure 6: micro USB

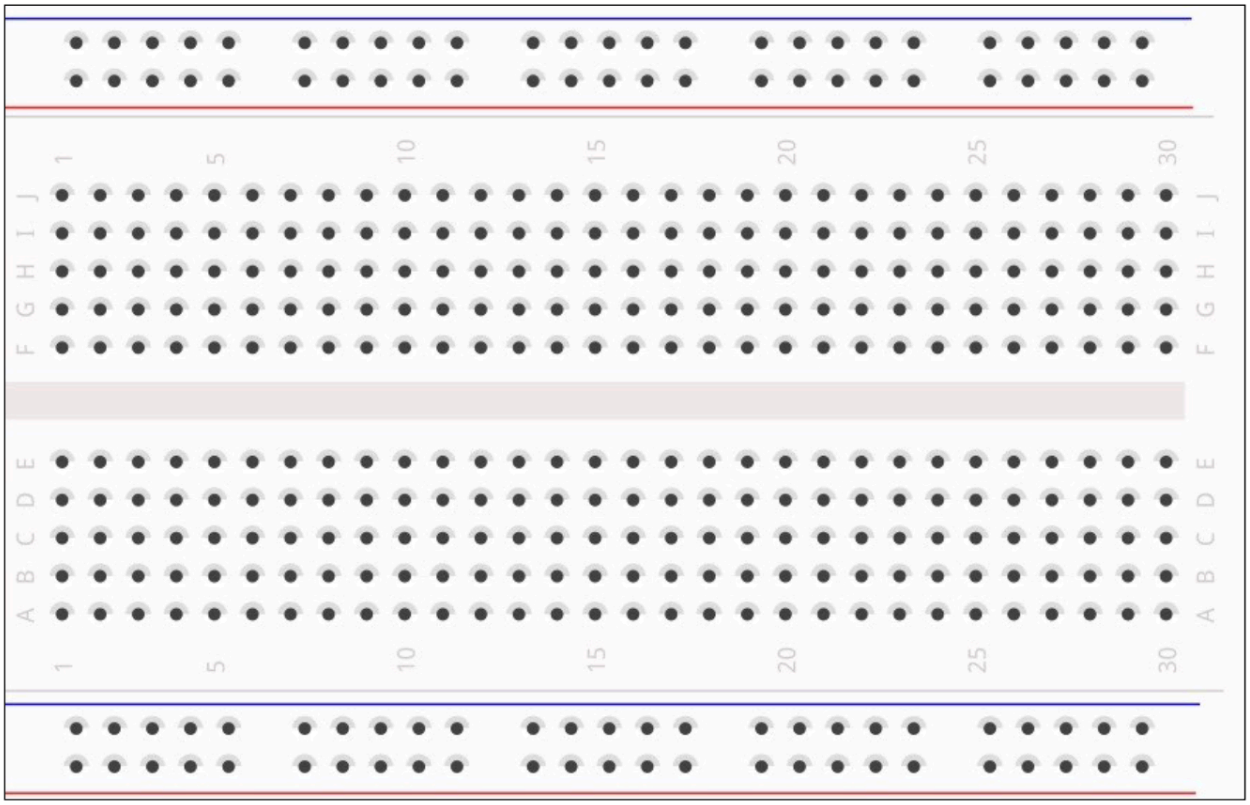
Breadboard

The breadboard is simply a great way to connect components without having to solder anything. There are strips of metal underneath that connect the holes together.

There are many sizes and varieties of breadboards. This one is a half-size breadboard; it has 400 pin points. They can be larger (full size) and smaller (mini). The half-size is perfect for our purposes and most projects. Whichever one you get, they all work on the same principle.

I place mine on a coaster, not essential but useful if you were to add components that don't fit onto a breadboard.

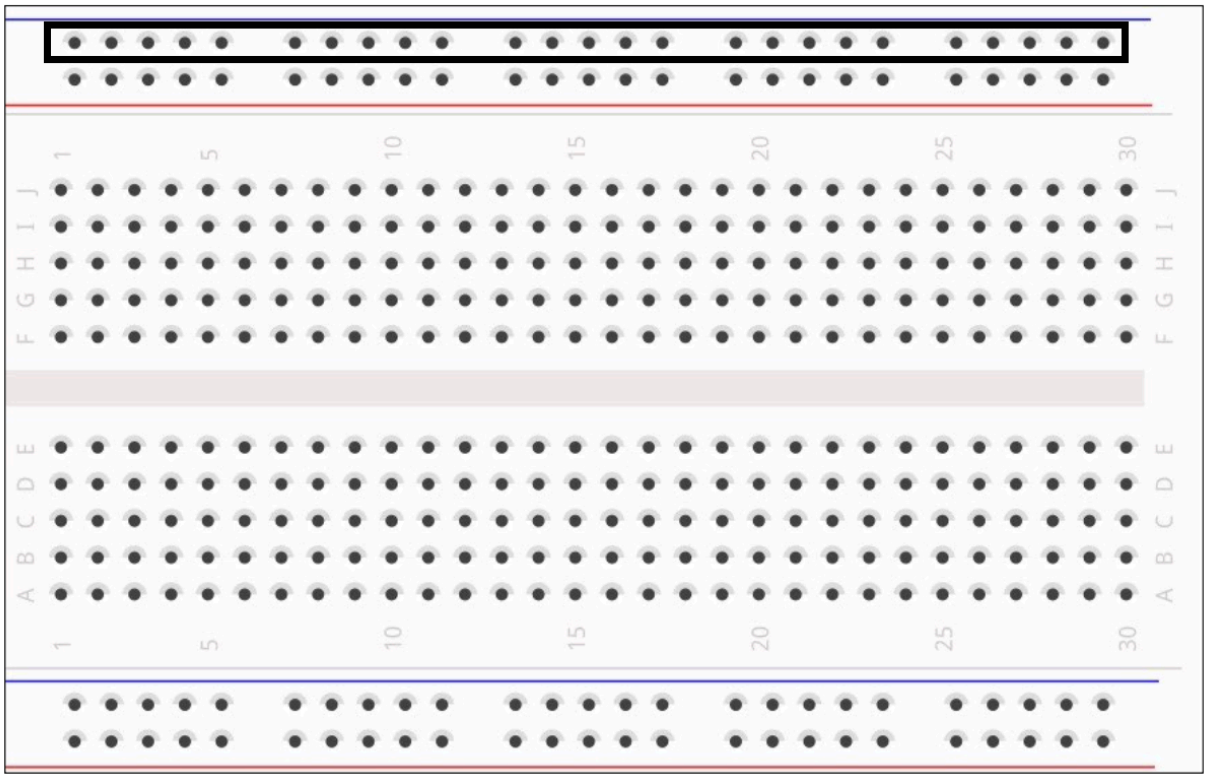
Figure 7: Half-Sized Breadboard



Using the Breadboard

The ground (GND) rail is connected along the whole length of the breadboard. This means that if you have a number of components all wanting to connect to the ground (GND), you can plug them all into the same one. This is the one near the blue line.

Figure 8: connecting along the length of the negative/ground rail

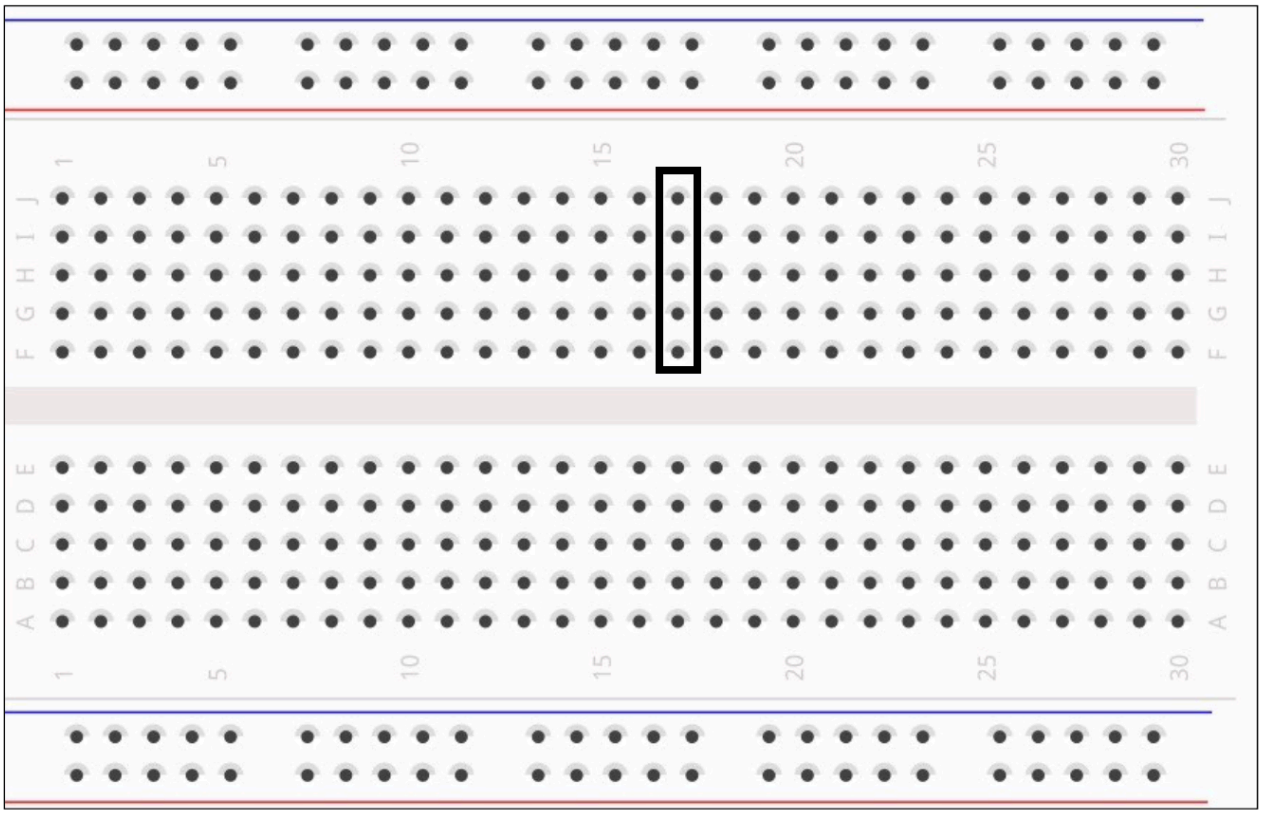


The positive (VCC) rail is connected across the entire length of the breadboard, as is the ground (GND). If you have lots of components that need a 5-volt supply (VCC), such as a servo, then you can use this to supply all of them. Convention is to follow the red line.

Figure 9: connecting along the length of the positive rail

The holes are connected across in rows of five, so that anything in that row is connected electrically. Each row of five holes is separate from all the other sets of five holes.

Figure 10: five connected pin holes



Adding the Arduino to the Breadboard

One of the reasons for getting the Arduino Nano 33 BLE with headers (pins) already soldered on is so that we can add it to the breadboard. Take care when doing this so that you do not bend the pins. Notice that it does not sit centrally, so it has to be offset to the left or right; it doesn't really matter, but I suggest doing what I have done because we use the digital pins side more than the analogue side.

Also notice that when it is connected (and powered), a little greenish (power) LED tells you that it is on.

Button

The button is a tactile, momentary type button, so it only completes the circuit while it is pressed. They come in a variety of sizes (and colours), and mainly they fit on the breadboard, although button modules don't.

One of the issues is that they will need a resistor in series so that the current isn't too high for the pin and would damage the board (this is also the case with LEDs). You can simply place a resistor in series or use a button module that has a resistor already built in. A third option is to use an internal resistor already within the board itself called a pull-up resistor.

For simplicity, we will use a version of fig.11 and use the pull-up feature of the Arduino Nano 33 BLE.

Figure 11: button

Jumper wires

These are wires widely used in electronics to connect components on a breadboard. They have pins sticking out or holes at the ends of the wires. Get ones that are as long as possible, or better still, get a mixture of lengths. There are three main types:

- Male-to-Male connectors (see Fig.12)
- Male-to-Female connectors
- Female-to-Female connectors

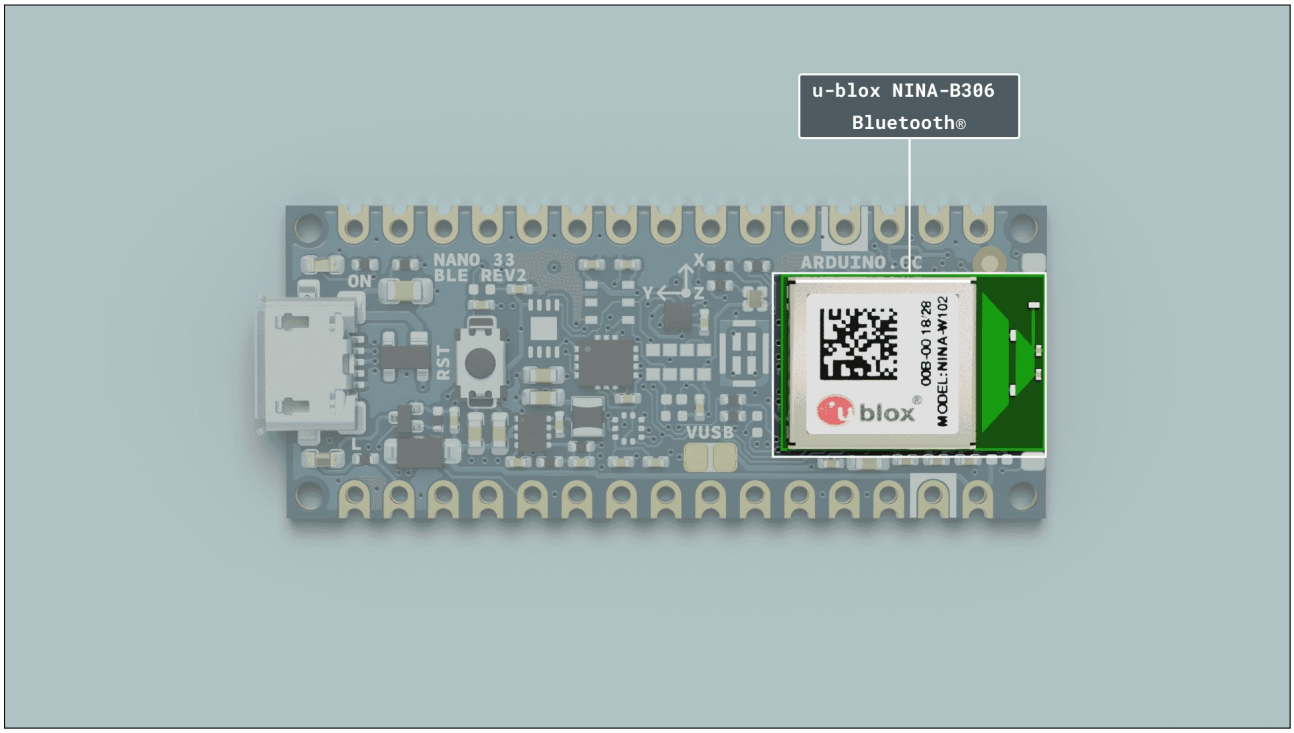
What you will need are Male-to-Male, and they usually come in packs of ten, which is plenty. You might also want to get Female-to-Male and Female-to-Female while you are at it.

Figure 12: male-to-male jumper leads

Bluetooth

The Arduino Nano 33 BLE also has Bluetooth, which means it can communicate with other Bluetooth devices, sending data to each other. There are two types of Bluetooth:

Figure 13: bluetooth module



Bluetooth Classic

Which can send lots of data but is energy-hungry, so only useful if powered directly.

Bluetooth Low Energy

Which is, as the name suggests, low energy because it is reserved for small amounts of data and therefore is more suitable for battery-powered devices.

ESP-NOW (*ESP boards only*)

This is another protocol that allows you to connect several ESP devices together. It has very low latency.

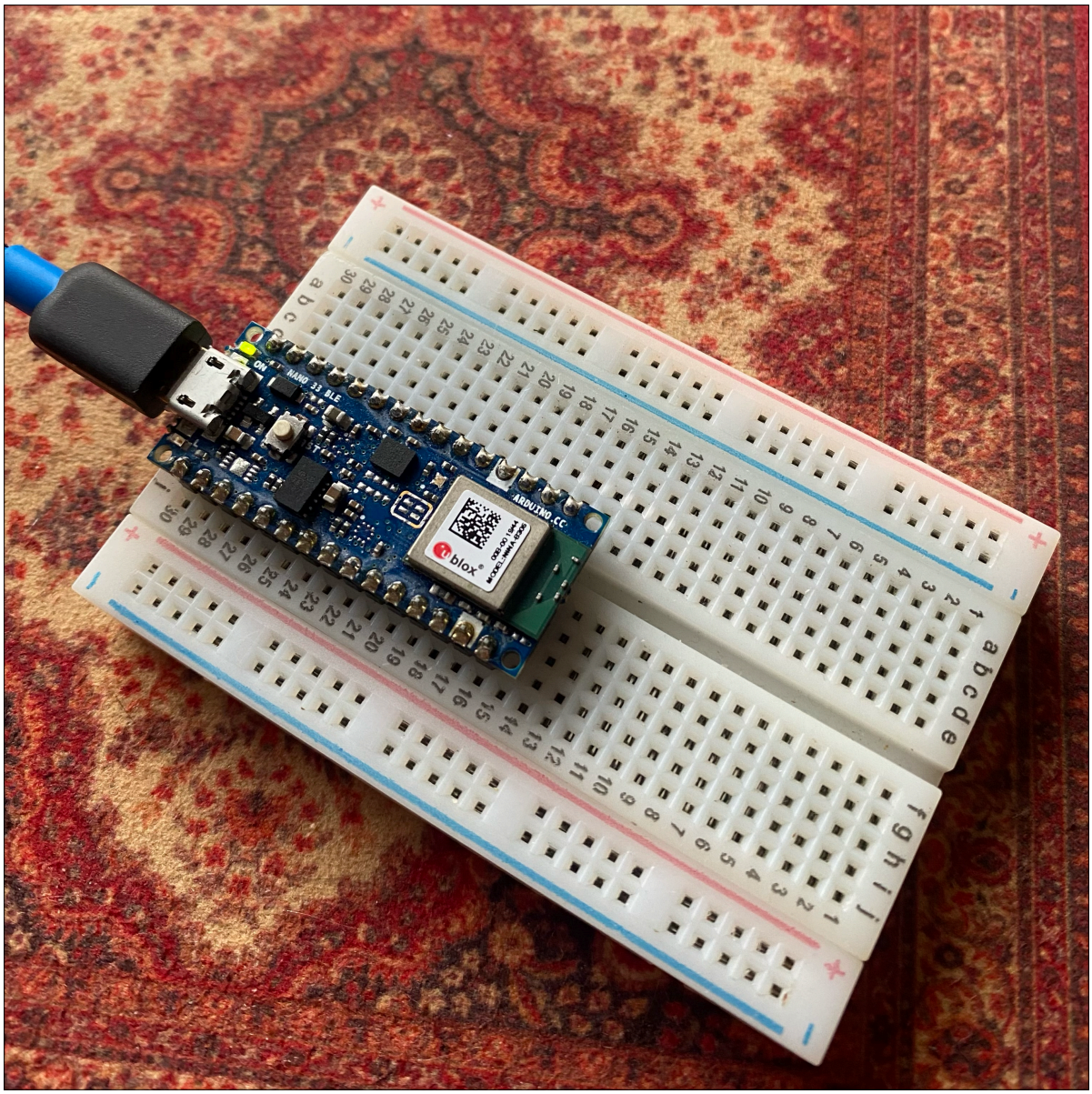
More than one

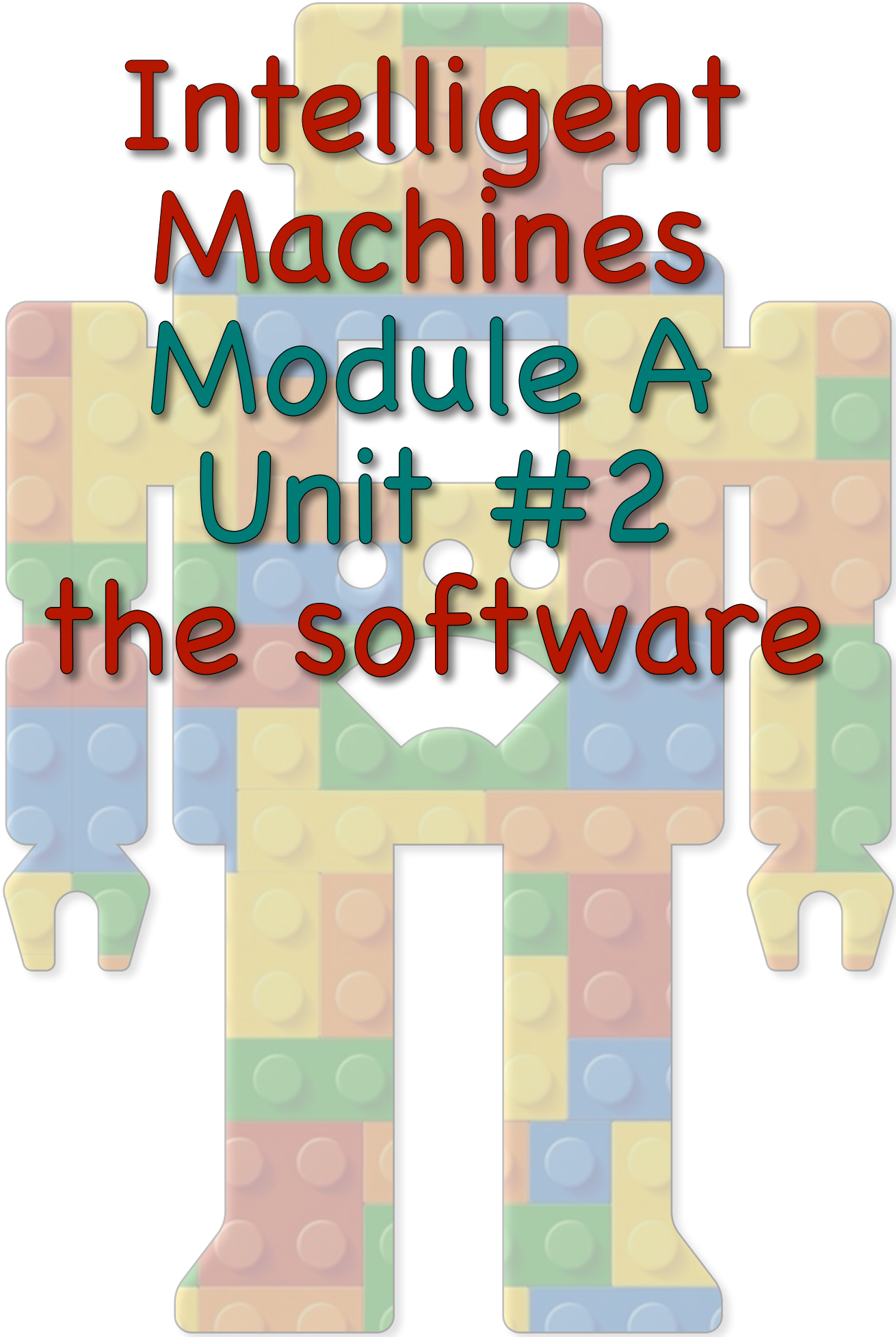
The board in Bluetooth mode is either a peripheral device or a central device, depending on whether it is sending data or receiving it. For obvious reasons, we will need at least two devices for that demonstration. They don't have to be exactly the same device.

Connecting to your computer

Your first step is to put the Arduino Nano 33 BLE on the breadboard as shown below. Use the USB cable to connect the microcontroller to your computer. The power LED should light up to tell you there is power going to the microcontroller and that you are now ready to upload your sketch. Before you can do that, we need software, and that is where the next unit explains all.

Figure 14: connecting your Arduino Nano 33 BLE





Intelligent Machines
Module A
Unit #2
the software

 Module A Unit #2: the software needed

To put code onto your device, in this case the Arduino Nano 33 BLE, you need a bit of software to help you. This is called an IDE, which stands for:

"Integrated Development Environment, a software application that combines common tools for writing and testing software code into a single graphical user interface (GUI). It streamlines the development process by providing a central platform for tasks like editing source code, debugging, and building applications, making developers more productive".

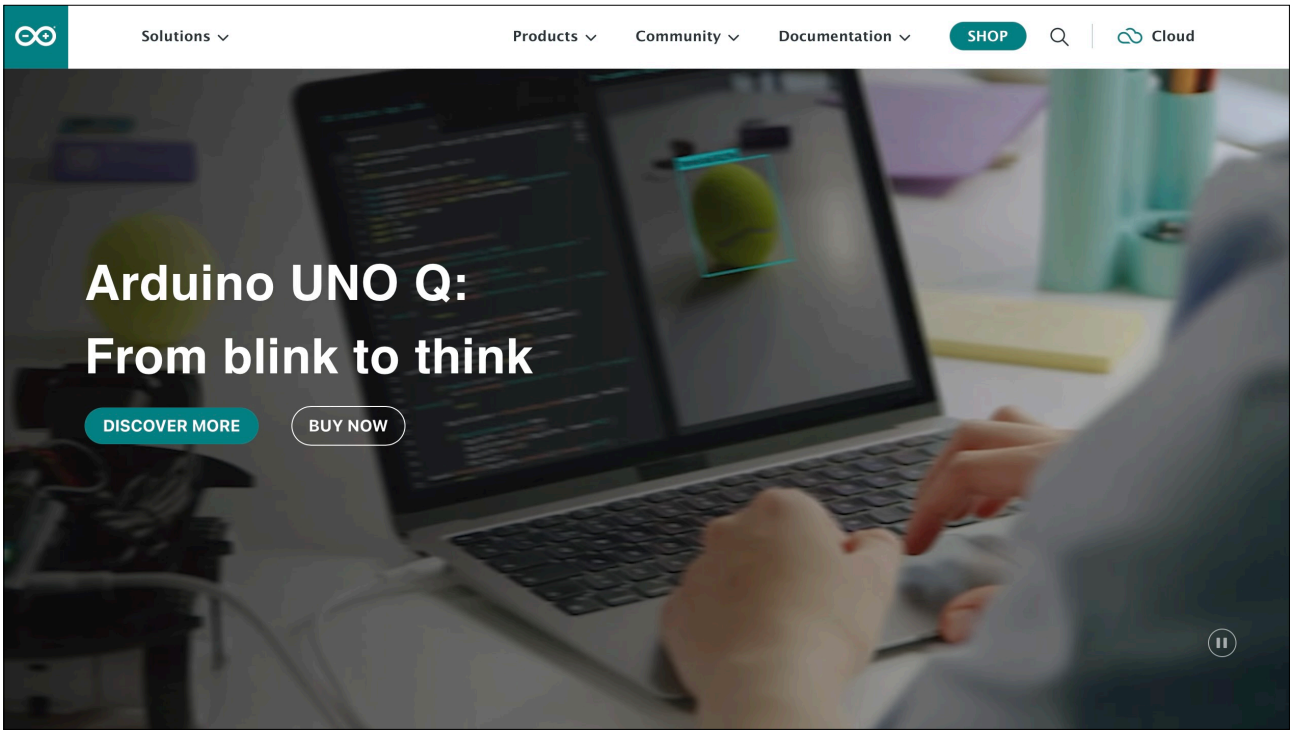
In other words, this is where you type in your code and send it to your device. An IDE can use any user-friendly language. It interprets what you write and converts it into meaningful code for the device. The coding language we are going to be using for the Arduino IDE is C/C++. It is very similar in format to p5.js.

The home page

You will need to download the software to upload the computer code to the Nano. Go to the Arduino website at:

https://www.arduino.cc/

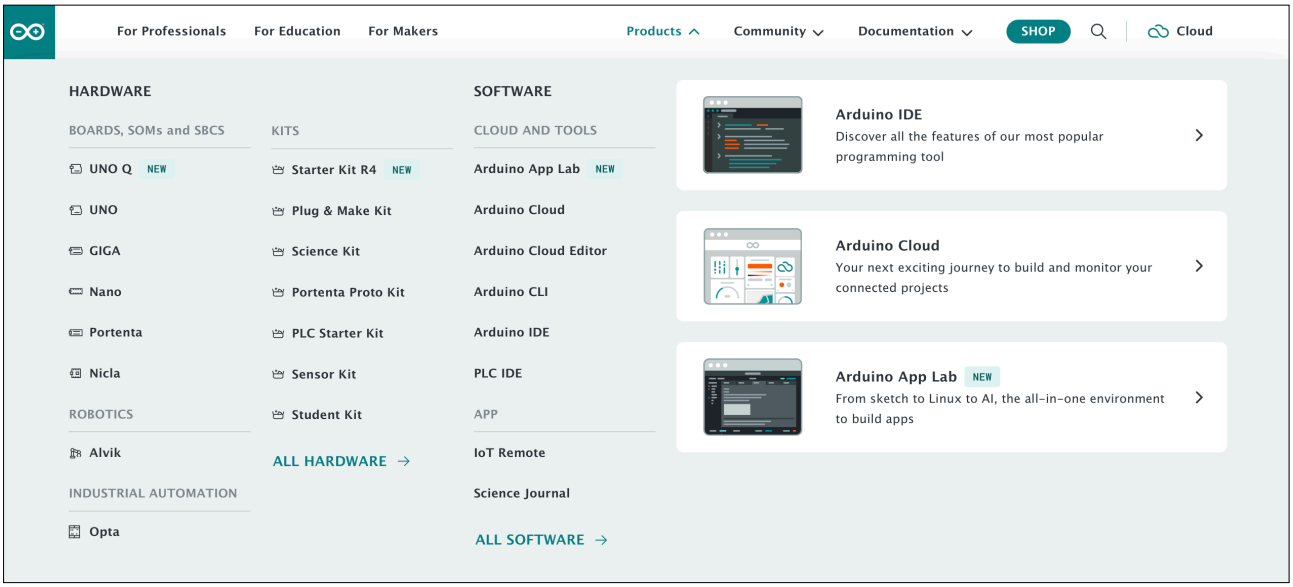
Figure 1: Arduino home page



The products

And click on the Products tab and then click on the Arduino IDE.

Figure 2: Arduino IDE



Download Arduino IDE

Then download the appropriate one for your operating system. You will see that there is a version 2.3.6 (as of writing).

Figure 3: download IDE

The icon

Once you have downloaded the software (you may be given the option to donate, but you don't have to), open it up. You should have a symbol like this. Click on it to open it up.

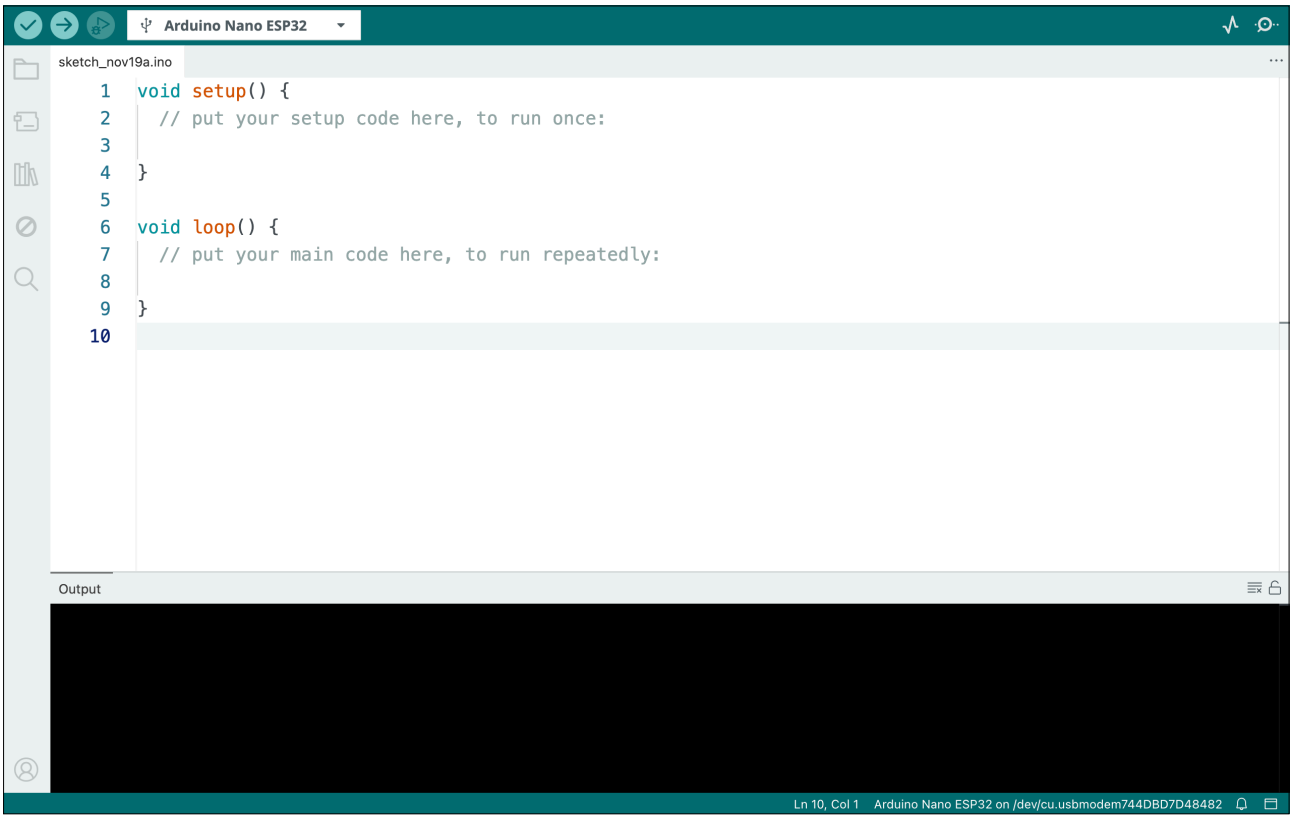
Figure 4: Arduino IDE icon

Getting Started

Once you have downloaded the software and opened it up, you should get a page like this. My preference is to delete all that and learn by typing it in again. Just like p5.js, all the code is called a sketch.

You will notice that instead of `function setup()` and `function draw()`, we have `void setup()` and `void loop()` instead. It is tantamount to the same thing.

Figure 5: default sketch



Notes

Just in case you are new here:

1. All the code that goes between the curly braces in the `void setup()` function happens just once.
2. All the code that goes between the curly braces in the `void loop()` function is a continuous loop.

```
void setup()
{
  ........
}

void loop()
{
  ........
}
```

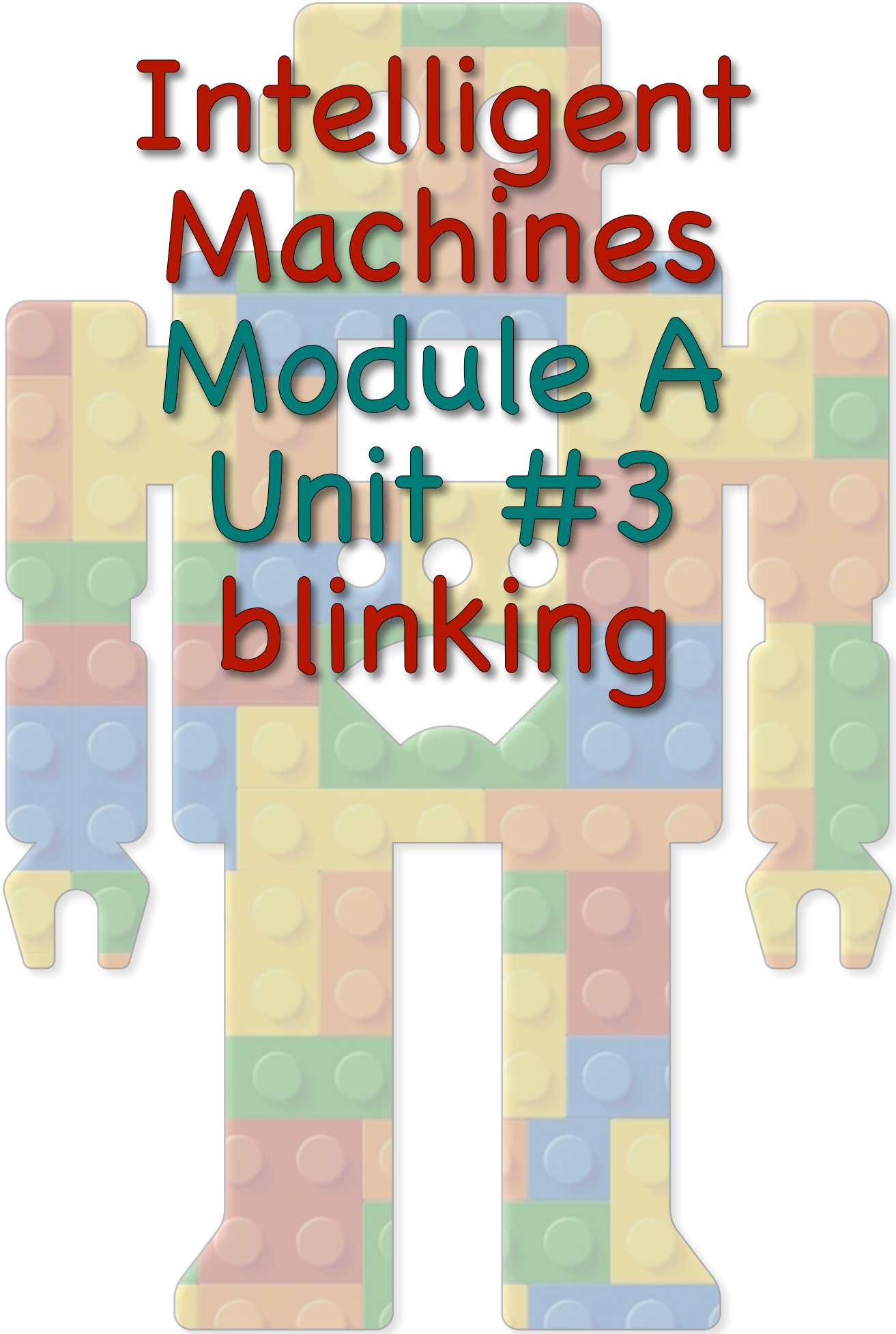
The buttons

When you want to upload the code, you press the arrow button (middle).

Figure 6: the main buttons

You may get an error message because it will first compile to check the code is OK, you can manually compile (left button tick) before uploading. The error message you get may or may not be very helpful. You can copy it and then see if there are any answers on the internet. Usually, there is a small typo somewhere, and the error message will give a hint.

Note: You may need to press the restart/reset button (grey) on the Arduino after you have uploaded. This is evident if the built-in LED is `glowing`.



Intelligent Machines Module A Unit #3 blinking

Module A Unit #3: blinking

Built-in LED

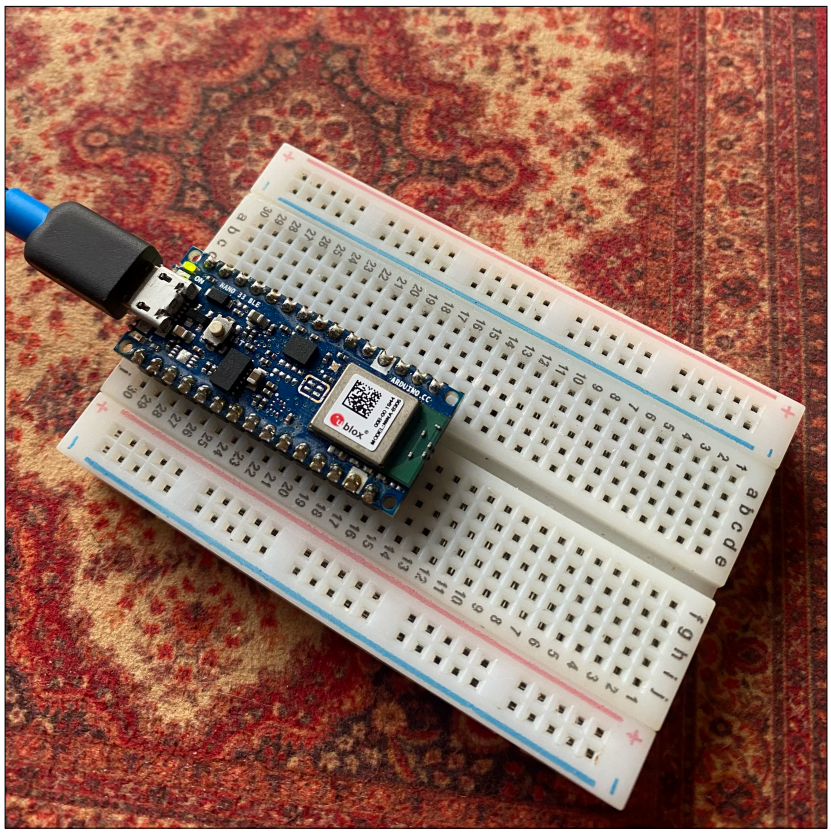
This is the Hello World! of coding a microcontroller, to blink an LED. The beauty of this particular board is that it has two other LEDs (apart from the power-on LED). One is situated on the other side of the USB connector. This LED is bright yellow when it is on.

RGB LED

The other LED is next to the RESET button and is an RGB LED, which means we can give it whatever colour we like, red, green, or blue, and a few colours in between.

Connect your Arduino Nano 33 BLE to your computer's USB port like so:

Figure 1: power LED should be on



The first sketch is a simple sketch where we are going to turn on the built-in LED. The built-in LED is also connected to digital pin 13 called D13, although we just use 13 without the D (it seems to know what you mean). We can also call it BUILTIN_LED, but for now, we will just use pin 13 as a reference.

Step 1
Delete the default code.

Step 2
Type in the code.

Step 3
Make sure your Arduino is connected.

Step 4
Check the board and port (under Tools).

Step 5
Press the upload button.

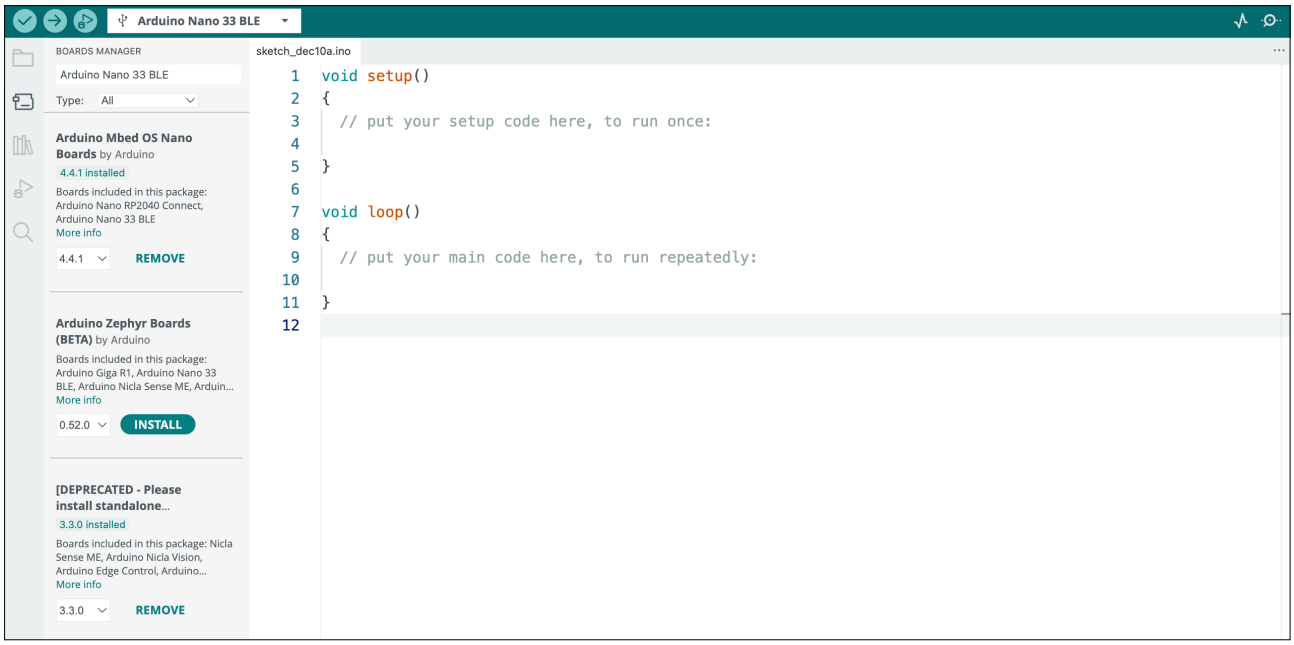
Step 6
Once everything is set up and working, you only need to do steps 2 and 5.

Installing the board software

Just to make sure we have the right board software installed for the Arduino Nano 33 BLE, we go to the list of icons on the left-hand side. Click on the second one down (outline of an Arduino Uno) and you should get the tab BOARDS MANAGER. Type Arduino Nano 33 BLE into the search box.

You will get three; the top one is the one you want. Click INSTALL and wait for it to be downloaded. You will notice that I have already downloaded it. It is important to keep up to date with all software. You usually get a reminder if there is an update.

Figure 7: downloading the board software



Sketch A3.1 LED on

As we have already stated the built-in LED is attached to pin 13. The following sketch will switch the LED on. Firstly, type the code below into the IDE as is and press the upload button (middle right pointing arrow). Make sure that you put the colons (;) and curly brackets { } where they should be.

! You may need to press the RESET button (grey) on the device after you have uploaded. Be aware that everything is case sensitive.

Arduino sketch

```
void setup()
{
  pinMode(13, OUTPUT);
}

void loop()
{
  digitalWrite(13, HIGH);
}
```

Notes

The first thing that happens is you get a little box appearing, which tells you it is compiling the code. This is when it might throw up any errors you have (if any). Once compiled, it will then upload, and you will be told when the upload is complete. At the same time, you will get a lot of information being printed to the console at the bottom.

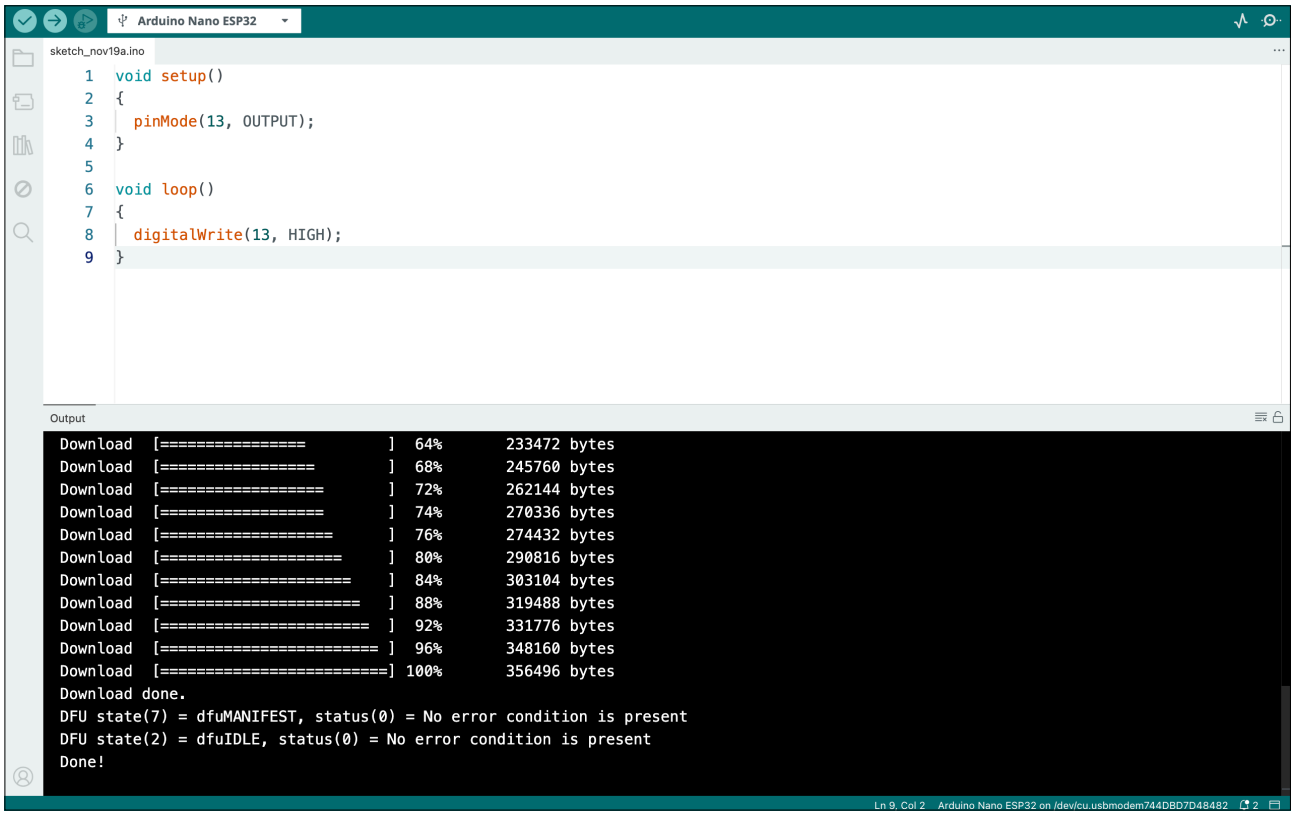
Challenge

Replace the 13 with LED_BUILTIN.

Code Explanation

`void setup()`	This happens once
`pinMode(13, OUTPUT);`	The pin on D13 is set to output as opposed to receiving an input from a device
`void loop()`	This is a continuous loop
`digitalWrite(13, HIGH);`	Then we tell pin 13 to be high which means 'on'

Figure 2: after you have uploaded the code successfully



Sketch A3.2 LED off

Now we have switched the LED on we want to switch the LED off. The red highlighted line is the new bit of code, either to be added or changed (saves typing everything out again). Remember that you may need to press the RESET button (grey) on the Arduino after you have uploaded.

Arduino sketch

```
void setup()
{
  pinMode(13, OUTPUT);
}

void loop()
{
  digitalWrite(13, LOW);
}
```

Notes

The LED should be switched off. Also notice that even for a minor change to the code, the compiling and uploading seem to take an eternity. This is one of the drawbacks of the Arduino IDE, but one we have to live with when using certain boards.

Code Explanation

`digitalWrite(13, LOW);`	The output to pin 13 (D13) is set to LOW which means off

Sketch A3.3 blinking good

For the next step we will switch the LED on for 1 second, which is 1000 milliseconds. The delay() function is measured in milliseconds. Then, switch it off for 1 second and back on again.

! If it doesn't start blinking, then press the reset button.

Arduino sketch

```
void setup()
{
  pinMode(13, OUTPUT);
}

void loop()
{
  digitalWrite(13, HIGH);
  delay(1000);
  digitalWrite(13, LOW);
  delay(1000);
}
```

Notes

You will notice that it blinks on and off continuously. This is because it is in a loop and returns to the first line of the code inside the void loop() function, repeating indefinitely (similar to function draw() in p5.js).

Challenge

Change the number of milliseconds from 1000 to 500.

Code Explanation

delay(1000);	The delay function stops everything for the number of milliseconds

Sketch A3.4 variables

Introducing variables. There are many sorts, but the first one we will use is an integer. The word int is short for integer, which is a type of data. An integer is a whole number, e.g. 1, 2, 34, 87, etc. The word delayPeriod is a made-up variable name. We are going to give it an initial value of 500.

Arduino sketch

```
int delayPeriod = 500;


void setup()
{
  pinMode(13, OUTPUT);
}

void loop()
{
  digitalWrite(13, HIGH);
  delay(1000);
  digitalWrite(13, LOW);
  delay(1000);
}
```

Notes

This doesn't do or change anything yet. We will make use of this variable soon.

Code Explanation

`int delayPeriod = 250;`	We have declared at the very beginning that delayPeriod is a variable, that it is an integer (int), and that it has a value of 500.

Sketch A3.5 using a variable

Instead of typing in the number every time we use `delay()`, we can create and use a variable `delayPeriod`. This is useful if we want to change the `delay()` later on; we now only have to change it once.

Arduino sketch

```
int delayPeriod = 500;


void setup()
{
  pinMode(13, OUTPUT);
}


void loop()
{
  digitalWrite(13, HIGH);
  delay(delayPeriod);
  digitalWrite(13, LOW);
  delay(delayPeriod);
}
```

Notes

Notice that it is blinking much faster now; each blink is half a second. Also notice that the way we named the variable means something. This is good practice rather than giving it a letter. It helps you later if you wonder what the variable is for, and others can come along later and look at your code. Also, if you combine two (or more) words, the first letter of the first word is lowercase, but the first letter of the other words is usually uppercase. This is one of the conventions commonly used.

Code Explanation

`delay(delayPeriod);`	The variable replaces the fixed value

Sketch A3.6 count

Now, what if we wanted to stop after, say, 10 blinks? First, we need to count them. We introduce another variable called count (a made-up variable name), we set it (initialise it) to 1, and in the loop(), we add 0 each time it blinks.

Arduino sketch

```
int delayPeriod = 500;
int count = 0;

void setup()
{
  pinMode(13, OUTPUT);
}

void loop()
{
  digitalWrite(13, HIGH);
  delay(delayPeriod);
  digitalWrite(13, LOW);
  delay(delayPeriod);
  count = count + 1;
}
```

Notes

This has no effect yet except to count.

Challenges

1. If you have done p5.js coding, you will know a shorthand version to add 1 to a variable.
2. Also, can you think of a way of stopping it after 10 blinks?

Code Explanation

`int count = 0;`	Creating a variable called count and initialised to zero
`count = count + 1;`	One is added on each iteration of the loop

Sketch A3.7 while loop

Now this is the tricky bit: how does it know when there have been 10 blinks? We introduce a `while()` loop straight into the `void loop()`. In other words, it will loop through while the `count` is less than 10. That is what the `<` means.

Arduino sketch

```
int delayPeriod = 250;
int count = 0;

void setup()
{
  pinMode(13, OUTPUT);
}

void loop()
{
  while (count <= 10)
  {
    digitalWrite(13, HIGH);
    delay(delayPeriod);
    digitalWrite(13, LOW);
    delay(delayPeriod);
    count = count + 1;
  }
}
```

Notes

To reorganise the code so it is formatted nicely, go to `Tools` and click `Auto Format`.
The `<` is called a `comparison operator`.

Notice that when you auto-format it changes the overall format to:

```
int delayPeriod = 500;
int count = 0;

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

void loop() {
  while (count <= 10) {
    digitalWrite(13, HIGH);
    delay(delayPeriod);
    digitalWrite(13, LOW);
    delay(delayPeriod);
    count = count + 1;
  }
}
```

Challenge

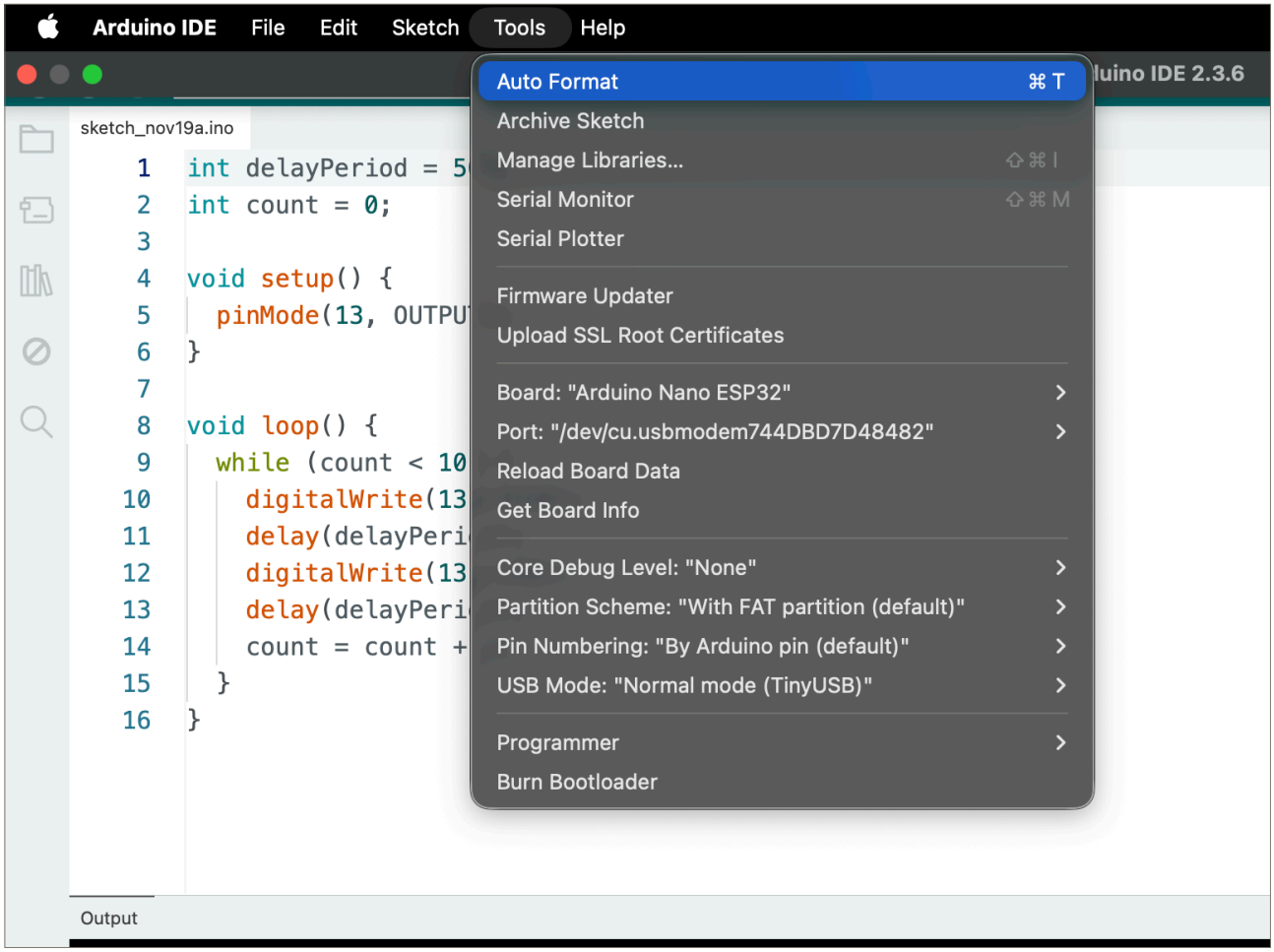
Change the number of times it blinks.

`Question`: Why does it blink `10` times when it stops before it reaches `10`?
Surely it should blink `9` times?
`Answer`: The counting starts from `0`, not `1`, so the first blink is when the count is `0`.

Code Explanation

`while (count < 10)`	The while() loop happens for as long as the condition is true i.e. while count is less than 10.

Figure 3: Auto Format



Formatting

The change in format is really what it should look like. If you prefer that format (and most do), then please code that way. I only do it my way to make the code a bit clearer to read; it is a trade-off.

Troubleshooting

If, like me, you can't upload the new code for some reason, a simple solution is to disconnect the board from the computer and connect it again. This seems to do the trick. The board can be a bit temperamental.

Another solution is to double-click on the reset button and then upload the code. If this fails, try uploading a simple blink sketch or check that you haven't made some error, such as getting the for loop variables wrong and entering into an infinite loop, as I did!!

Comparison Operators

There are more of them. Below is a list of the comparison operators.

!=	not equal to
<	less than
<=	less than or equal to
==	equal to
>	greater than
>=	greater than or equal to

Arithmetic Operators

These are the arithmetic operators; some may be familiar, and others not.

%	remainder
*	multiplication
+	addition
-	subtraction
/	division
=	assignment operator

Sketch A3.8 ten blinks

Another scenario is for there to be 10 blinks and then a break (of, say, one second) followed by another burst of 10 blinks, etc. For this, we will need an `if()` loop.

! Remove the `while()` loop and start with this...

Arduino sketch

```
int delayPeriod = 500;
int count = 0;

void setup()
{
  pinMode(13, OUTPUT);
}

void loop()
{
  digitalWrite(13, HIGH);
  delay(delayPeriod);
  digitalWrite(13, LOW);
  delay(delayPeriod);
  count = count + 1;
}
```

Sketch A3.9 adding the if()

Now to add in the `if()` loop.

Arduino sketch

```
int delayPeriod = 250;
int count = 1;

void setup()
{
  pinMode(13, OUTPUT);
}

void loop()
{
  digitalWrite(13, HIGH);
  delay(delayPeriod);
  digitalWrite(13, LOW);
  delay(delayPeriod);
  count = count + 1;
  if (count == 10)
  {
    count = 0;
    delay(1000);
  }
}
```

Notes

You should get 10 blinks, a 1 second pause, and then 10 more, and so on. We use two equal signs because it is a comparison. If you use just one equal sign = (by mistake), it is a mathematical assignment. If you get an error message at this point, that is probably why.

Code Explanation

`if (count == 10)`	An if() statement, if this is true (count is 10) then do that (whatever is in the loop)

Sketch A3.10 compound operator

As I alluded to earlier, introducing a compound operator. This one will appear quite often, `++`. We will replace `count = count + 1` with `count++`. It does exactly the same thing, adding `1` each time to the `count` variable.

Arduino sketch

```
int delayPeriod = 500;
int count = 0;

void setup()
{
  pinMode(13, OUTPUT);
}

void loop()
{
  digitalWrite(13, HIGH);
  delay(delayPeriod);
  digitalWrite(13, LOW);
  delay(delayPeriod);
  count++;
  if (count == 10)
  {
    count = 0;
    delay(1000);
  }
}
```

Notes

This is so very useful and very common, so it is worth introducing it now so that you can see how it works. Below I have included other ones that you will come across (less so).

Code Explanation

`count++;`	Shorthand for count plus 1

Compound Operators

These are very useful shortcuts.

++	Incrementally adds 1
--	Incrementally subtracts 1
+=	Adds a value e.g. x += 5 adds 5 to x each time
-=	Subtracts a value e.g. x -= 5 subtracts 5 from x each time
*=	Multiplies a value e.g. x *= 5 multiplies x by 5 each time
/=	Divides by a value e.g. x /= 5 divides x by 5 each time
%=	Remainder of a value e.g. x %= 5 will give you the remainder from divided by 5 each time

Sketch A3.11 introducing the for loop

Introducing a `for()` loop. You will see this a lot in coding, so now is as good a time to introduce it. It may look a bit bewildering at first, but it is perfectly logical, and we will spend a bit of time helping you to become familiar with this very important loop. First, let us go back to this starting point.

Arduino sketch

```
int delayPeriod = 500;


void setup()
{
  pinMode(13, OUTPUT);
}


void loop()
{
  digitalWrite(13, HIGH);
  delay(delayPeriod);
  digitalWrite(13, LOW);
  delay(delayPeriod);
}
```

Sketch A3.12 a loop of three parts

The `for(int i = 0; i < 10; i++)` loop has three parts separated by semi-colons (`;`).

1. The first part initialises a variable which we will call `i` and gives it an initial value of `0` but it can be any value: `int i = 0;`
2. Next, we put a limit on how big `i` is going to get in this example: `i < 10;`
3. Finally, we specify what increments we increase `i` by on each iteration (loop), we could specify increments of any value: `i++`

Arduino sketch

```
int delayPeriod = 500;


void setup()
{
  pinMode(13, OUTPUT);
}


void loop()
{
  for (int i = 0; i < 10; i++)
  {
    digitalWrite(13, HIGH);
    delay(delayPeriod);
    digitalWrite(13, LOW);
    delay(delayPeriod);
  }
  delay(1000);
}
```

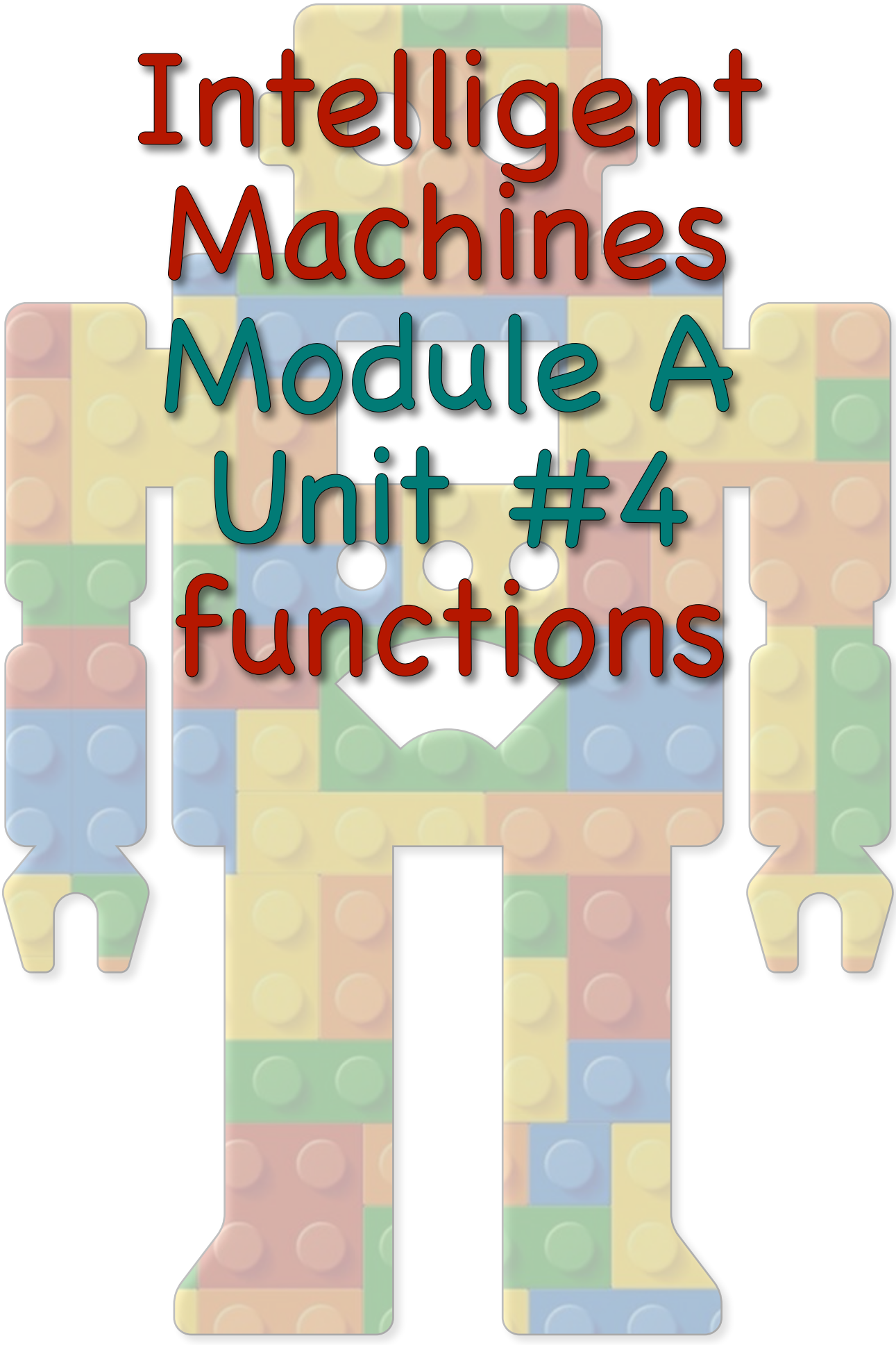
Notes

This produces exactly the same result as the `while()` loop and `if()` statement in a completely different way. You can indent inside the loop, but it is not essential. If you think about it, we initialise the count (called `i`) to `0`, we have a conditional statement `<` less than `10`, and finally we increment the count (`i`) by `1`. We do all of that in one line of code.

Code Explanation

`for (int i = 0; i < 10; i++)`	A for() loop, which acts as a counter for each loop, starts at zero, adds one each loop (iteration and continues until it reaches nine (not ten) which is ten loops altogether

Intelligent Machines Module A Unit #4 functions



 Module A Unit #4: functions

As you know, functions are a key part of the coding world. This is also true with robotics. This unit takes you through the basics and the syntax of functions in C/C++. It is very similar to p5.js, and so the learning curve is not too steep.

Definition

Segmenting code into functions allows a programmer to create modular pieces of code that perform a defined task and then return to the area of code from which the function was "called". The typical case for creating a function is when one needs to perform the same action multiple times in a programme.

Sketch A4.1 a blinking function

We have a function called `void setup()`, a function called `void loop()`, and now we are going to create one called `void blink()`. This is a made-up function where we are going to put our code for blinking.

Arduino sketch

```
int delayPeriod = 500;


void setup()
{
  pinMode(13, OUTPUT);
}


void loop()
{
  for (int i = 0; i < 10; i++)
  {
    digitalWrite(13, HIGH);
    delay(delayPeriod);
    digitalWrite(13, LOW);
    delay(delayPeriod);
  }
  delay(1000);
}


void blink()
{


}
```

Notes

We can put the function anywhere, but it does start at the beginning and work its way down.

Code Explanation

`void blink()`	We have created a new function called blink()

Sketch A4.2 filling the blink

Now we are going to call it from inside the `for()` loop. This is a cut-and-paste job, taking the code out and replacing it with the `blink()` function.

Arduino sketch

```
int delayPeriod = 500;

void setup()
{
  pinMode(13, OUTPUT);
}

void loop()
{
  for (int i = 0; i < 10; i++)
  {
    blink();
  }
  delay(1000);
}

void blink()
{
  digitalWrite(13, HIGH);
  delay(delayPeriod);
  digitalWrite(13, LOW);
  delay(delayPeriod);
}
```

Notes

This can be useful for keeping the code neat and tidy, especially where you want to use the same bit of code repeatedly. One of the targets that coders try to achieve is to write as few lines as possible to achieve the same result. It is considered bad form to repeat the same lines of code.

Code Explanation

`blink();`	This blink() function is called ten times in the for() loop

Sketch A4.3 is this an argument?

Here we will combine functions and arguments. Using the above sketch, we can rationalise it even more by using the two arguments in the function itself. We will start with just one and then add the other afterwards.

! Remove the line of code: `int delayPeriod = 500;`

Arduino sketch

```
void setup()
{
  pinMode(13, OUTPUT);
}

void loop()
{
  for (int i = 0; i < 10; i++)
  {
    blink(500);
  }
  delay(1000);
}

void blink(int delayPeriod)
{
  digitalWrite(13, HIGH);
  delay(delayPeriod);
  digitalWrite(13, LOW);
  delay(delayPeriod);
}
```

Code Explanation

`void blink(int delayPeriod)`	This is a variable inside a function, you can call it an argument. It will hold that value when it is called from somewhere else, in this case 250

Sketch A4.4 learning to count

We can go one step further by adding in the `count` value of `10`. There is a bit of cutting and pasting, so work through the sketch to see how it is doing what it does.

Arduino sketch

```
void setup()
{
  pinMode(13, OUTPUT);
}

void loop()
{
  blink(500, 10);
  delay(1000);
}

void blink(int delayPeriod, int count)
{
  for (int i = 0; i < count; i++)
  {
    digitalWrite(13, HIGH);
    delay(delayPeriod);
    digitalWrite(13, LOW);
    delay(delayPeriod);
  }
}
```

Notes

You have initialised them inside the function brackets. You draw their values from the `void loop()` function. We create an integer variable called `count`.

Challenge

Change the values in `blink(500, 10);`

Code Explanation

`int delayPeriod`	Both are variables that are arguments. They each take on the values of 500 and 10 respectively
`int count`	

Sketch A4.5 a blinking stop

❗ This is our new starting sketch.
In this next sketch, we will use the += operator to slow the blink down.

Arduino sketch

```
int delayPeriod = 1000;


void setup()
{
  pinMode(13, OUTPUT);
}


void loop()
{
  digitalWrite(13, HIGH);
  delay(delayPeriod);
  digitalWrite(13, LOW);
  delay(delayPeriod);
}
```

Sketch A4.6 increasing blink

Changing the delayPeriod to something much shorter, 20 milliseconds, and increasing it by 10 milliseconds on each blink.

Arduino sketch

```
int delayPeriod = 20;


void setup()
{
  pinMode(13, OUTPUT);
}

void loop()
{
  digitalWrite(13, HIGH);
  delay(delayPeriod);
  digitalWrite(13, LOW);
  delay(delayPeriod);
  delayPeriod += 10;
}
```

Code Explanation

delayPeriod += 10;	We add 10 to the variable delayPeriod on every iteration

Sketch A4.7 stop blinking

If we wanted to stop the blinking at 1000 milliseconds, we would need to first introduce another variable. We will call this increment and make it much bigger, i.e. 50.

Arduino sketch

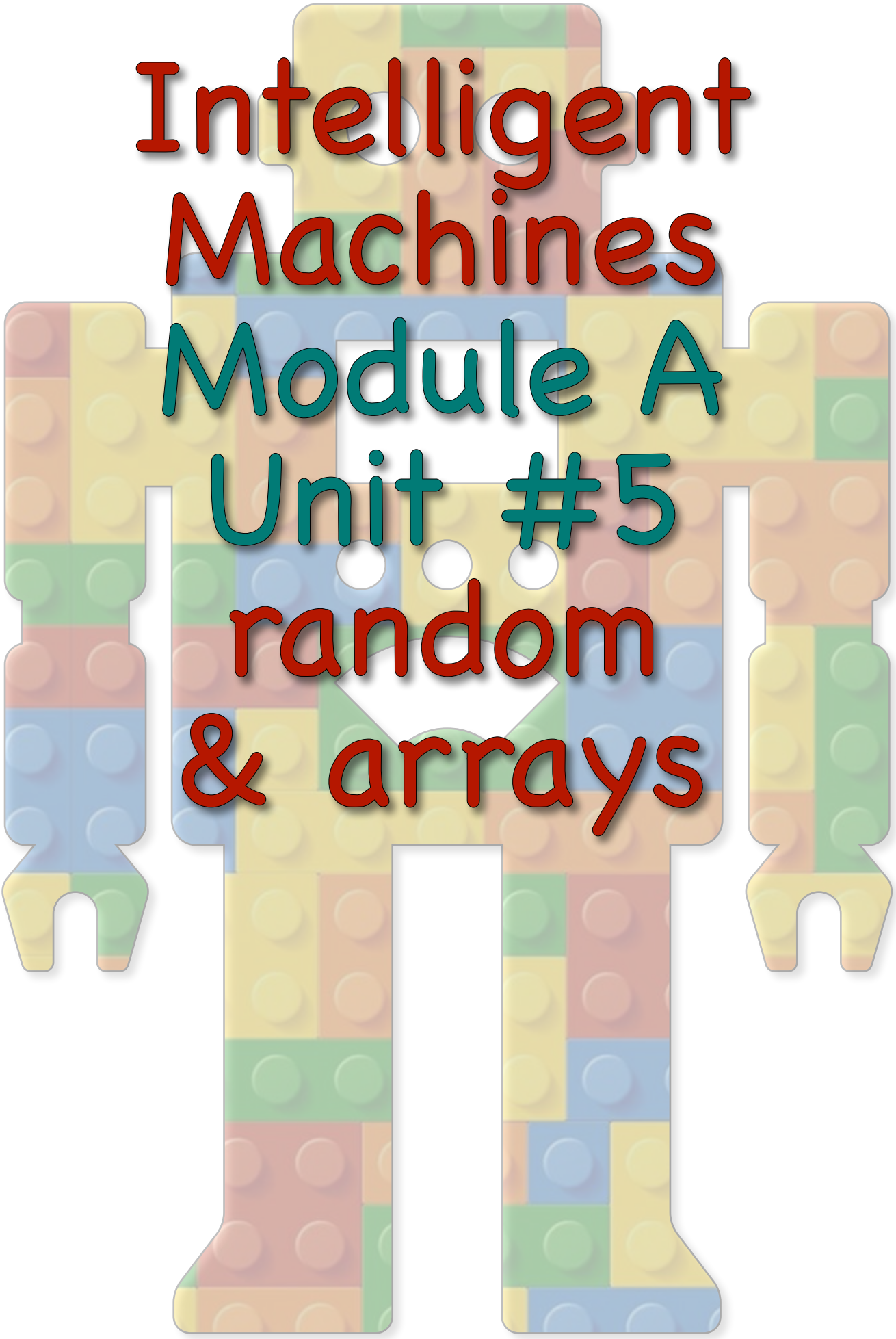
```
int delayPeriod = 20;
int increment = 50;

void setup()
{
  pinMode(13, OUTPUT);
}

void loop()
{
  digitalWrite(13, HIGH);
  delay(delayPeriod);
  digitalWrite(13, LOW);
  delay(delayPeriod);
  delayPeriod += increment;
}
```

Sketch A4.8 and stop

Stopping at 1 second (1000 milliseconds).

Arduino sketch

```
int delayPeriod = 20;
int increment = 50;

void setup()
{
  pinMode(13, OUTPUT);
}

void loop()
{
  digitalWrite(13, HIGH);
  delay(delayPeriod);
  digitalWrite(13, LOW);
  delay(delayPeriod);
  delayPeriod += increment;
  if (delayPeriod > 1000)
  {
    increment = 0;
  }
}
```



Intelligent Machines Module A Unit #5 random & arrays

Module A Unit #5: random and arrays

Arrays

Arrays are a very powerful tool for collecting, storing, and manipulating data. This is a simple introduction to its use and purpose.

Random

The `random()` function gives you a random number. Although random for most purposes, it does have a pattern to it. There are ways to improve the randomness using `randomSeed()`. It can take a random value for one of the pins (noise), so that each time the sketch is run, it chooses a different `randomSeed()`.

Sketch A5.1 a fast blink

We will start with a basic sketch that you are familiar with. This will blink reasonably fast.

Arduino sketch

```
void setup()
{
  pinMode(13, OUTPUT);
}

void loop()
{
  blink(250);
}

void blink(int delayPeriod)
{
  digitalWrite(13, HIGH);
  delay(delayPeriod);
  digitalWrite(13, LOW);
  delay(delayPeriod);
}
```

Notes

Rapid blinking!

Sketch A5.2 hard code

We will hard-code the delayPeriod for when the LED is off (LOW). Now it is on for 250 milliseconds and off for 100 milliseconds.

Arduino sketch

```
void setup()
{
  pinMode(13, OUTPUT);
}

void loop()
{
  blink(250);
}

void blink(int delayPeriod)
{
  digitalWrite(13, HIGH);
  delay(delayPeriod);
  digitalWrite(13, LOW);
  delay(100);
}
```

Challenge

Try a variety of combinations.

Sketch A5.3 adding an array

We are going to add an array for how long the LED is on for. The name of the array is `durations[]`. The data is stored in curly brackets `{}`, separated by commas. We are going to have the LED on for `eight` different periods of time (`elements` in the array).

Arduino sketch

```
int durations[] = {100, 100, 2000, 2000, 100, 100, 2000, 2000};

void setup()
{
  pinMode(13, OUTPUT);
}

void loop()
{
  blink(250);
}

void blink(int delayPeriod)
{
  digitalWrite(13, HIGH);
  delay(delayPeriod);
  digitalWrite(13, LOW);
  delay(100);
}
```

Notes

Nothing is going to change yet.

Challenge

You can create your own array of elements.

Code Explanation

`durations[]`	An empty array called durations. The [] brackets denote an array, this is then filled with the elements in the {} braces.

Sketch A5.4 calling the elements

Now we need to call each element of the array in turn. The counting in arrays is different to the way we count. The counting starts with 0, then 1, 2, 3, 4, 5... then two rather than start with 1, 2, 3, 4, 5... so the first element in the array is position or index 0 (not 1).

Now we need to loop through each one in turn and we can do this with a `for()` loop. Each `i` is an `index`. So `index[0]` is `100`, `index[1]` is `100`, `index[2]` is `2000`, `index[3]` is `2000`, `index[4]` is `100` and so on.

This translates to `durations[0]` is `100`, `durations[1]` is `100`, `durations[2]` is `2000` and so on, looping through one at a time till it gets to the last one and starts all over again.

Arduino sketch

```
int durations[] = {100, 100, 2000, 2000, 100, 100, 2000, 2000};


void setup()
{
  pinMode(13, OUTPUT);
}


void loop()
{
  for (int i = 0; i < 8; i++)
  {
    blink(durations[i]);
  }
}


void blink(int delayPeriod)
{
  digitalWrite(13, HIGH);
  delay(delayPeriod);
  digitalWrite(13, LOW);
  delay(250);
}
```

Notes

Each value is passed onto the `delayPeriod` for the `blink()` function. What you should see is two short blinks followed by two longer blinks, and then two short blinks and then two long blinks.

Challenge

If you aren't sure exactly how big the array is but need it for the `for()` loop, you cannot use `durations.length` as you could with p5.js; instead, use the `sizeof()` function (I am not going to go into why it is this way!!). Try it:

```
for (int i = 0; i < sizeof(durations)/sizeof(durations[0]); i++)
{
  .....
}
```

Sketch A5.5 a very basic sketch

! Starting with a very basic new blink sketch.

Arduino sketch

```
void setup()
{
  pinMode(13, OUTPUT);
}

void loop()
{
  digitalWrite(13, HIGH);
  delay(1000);
  digitalWrite(13, LOW);
  delay(1000);
}
```

Notes

If you are not convinced that the code has produced the changes you have entered, then press the RESET button to start the programme from the beginning.

Sketch A5.6 a random variable

Introduce a new variable called `delayRandom` and give it an initial value of `1000`. It is always a good idea to give a variable an initial value.

Arduino sketch

```
int delayRandom = 1000;


void setup()
{
  pinMode(13, OUTPUT);
}

void loop()
{
  digitalWrite(13, HIGH);
  delay(delayRandom);
  digitalWrite(13, LOW);
  delay(delayRandom);
}
```

Notes

This should provide no change whatsoever over the previous sketch.

Sketch A5.7 a random delay

Now we change the 1000 milliseconds to a random number between 0 and 1000. It will change every time it loops through.

Arduino sketch

```
int delayRandom = 1000;

void setup()
{
  pinMode(13, OUTPUT);
}

void loop()
{
  delayRandom = random(1000);
  digitalWrite(13, HIGH);
  delay(delayRandom);
  digitalWrite(13, LOW);
  delay(delayRandom);
}
```

Notes

It will blink at random rates, as if in Morse code.

Challenge

Create another random variable for the period the LED is off (LOW). It will create a very random effect.

Code Explanation

`random(1000);`	Gives you a random number between 0 and 1000

Sketch A5.8 random limits

We can create upper and lower limits, so now it has a random number between 500 and 2000 milliseconds.

Arduino sketch

```
int delayRandom = 1000;

void setup()
{
  pinMode(13, OUTPUT);
}

void loop()
{
  delayRandom = random(500, 2000);
  digitalWrite(13, HIGH);
  delay(delayRandom);
  digitalWrite(13, LOW);
  delay(delayRandom);
}
```

Notes

Does just what it says on the tin.

Code Explanation

`random(500, 2000);`	Returns a random number between 500 and 2000

Sketch A5.9 basic delay

! Start with a new basic sketch with the delay set to 1000.

Arduino sketch

```
void setup()
{
  pinMode(13, OUTPUT);
}

void loop()
{
  digitalWrite(13, HIGH);
  delay(1000);
  digitalWrite(13, LOW);
  delay(1000);
}
```

Notes

Feels like we have been here before.

Sketch A5.10 ten element array

We add an array of 10 elements. This is a blank or empty array, effectively 10 zeros.

Arduino sketch

```
int durations[10];


void setup()
{
  pinMode(13, OUTPUT);
}

void loop()
{
  digitalWrite(13, HIGH);
  delay(1000);
  digitalWrite(13, LOW);
  delay(1000);
}
```

Notes

It is considered good form to give the array an initial size.

Code Explanation

`int durations[10];`	This array is effectively empty but it has been given the dimensions of 10 elements.

Sketch A5.11 filling the array

In `setup()`, we fill each element with a random number between `0` and `1000`. We do this by looping through ten times, each time selecting a random number.

Arduino sketch

```
int durations[10];

void setup()
{
  pinMode(13, OUTPUT);
  for (int i = 0; i < 10; i++)
  {
    durations[i] = random(1000);
  }
}

void loop()
{
  digitalWrite(13, HIGH);
  delay(1000);
  digitalWrite(13, LOW);
  delay(1000);
}
```

Notes

This puts `10` random numbers into the array (just once) in `setup()`.

Code Explanation

`durations[i] = random(1000);`	As it cycles through the loop, [i] starts at zero and increments by 1 each loop. So for each position in the array it puts a random number from 0 to 1000 into that array. The [i] value is the index reference for the array, counting starts at zero not 1.

Sketch A5.12 loop in a function

Now we need to read from each element in a similar loop inside the `void loop()` function.

Arduino sketch

```
int durations[10];

void setup()
{
  pinMode(13, OUTPUT);
  for (int i = 0; i < 10; i++)
  {
    durations[i] = random(1000);
  }
}

void loop()
{
  for (int i = 0; i < 10; i++)
  {
    digitalWrite(13, HIGH);
    delay(durations[i]);
    digitalWrite(13, LOW);
    delay(durations[i]);
  }
}
```

Notes

You should see the same pattern of blinks every 10 seconds.

Challenge

Create another array for when the LED is `off`.

Code Explanation

`delay(durations[i]);`	This is the reverse of the previous operation. It goes through the array one element at a time starting at index 0 through to index 9, it reads its value and that is the length of the delay.

Sketch A5.13 an array of floats

A new sketch illustrates how to create an array of random `floats` to two decimal places by default.

Arduino sketch

```
float myArray[10];
bool repeat = true;

void setup()
{
  Serial.begin(9600);
  for (int i = 0; i < 10; i++)
  {
    myArray[i] = float(random(100))/100;
  }
}

void loop()
{
  delay(3000);
  if (repeat == true)
  {
    for (int i = 0; i < 10; i++)
    {
      Serial.print(i);
      Serial.print(" ");
      Serial.println(myArray[i]);
    }
  }
  repeat = false;
}
```

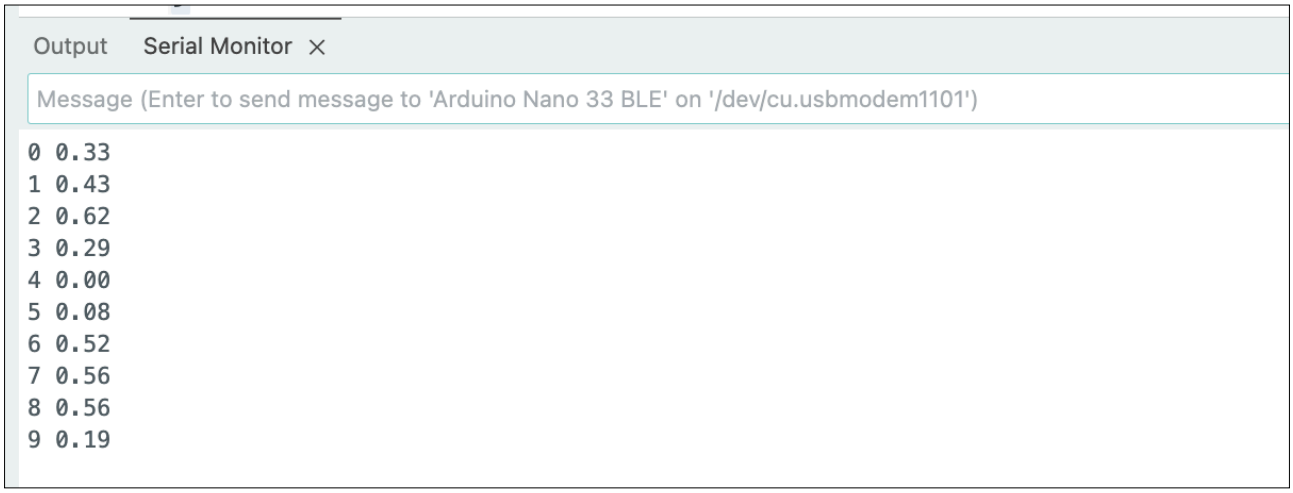
Notes

The delay of 3 seconds is necessary for the serial monitor to catch up, otherwise it misses the early part of the array.

Code Explanation

`float myArray[10];`	Create an array of 10 elements
`bool repeat = true;`	We need a mechanism to only send data once to the serial monitor, initialise boolean to true
`myArray[i] = float(random(100))/100;`	This creates a random number between 0-100 and then divides that number to give us a number to two decimal places
`delay(3000);`	Gives time for the serial monitor to catch up
`if (repeat == true)`	Only goes through the conditional loop if true
`Serial.print(i);`	Prints the index value starting at zero
`Serial.print(" ");`	Leave a space
`Serial.println(myArray[i]);`	Print the value in the array at that index, then carriage return (new line)
`repeat = false;`	Negates the condition

Figure A5.13



Sketch A5.14 to five decimal places

We simply increase the random number generated and divide it by that amount.

Arduino sketch

```
float myArray[10];
bool repeat = true;

void setup()
{
  Serial.begin(9600);
  for (int i = 0; i < 10; i++)
  {
    myArray[i] = float(random(100000))/100000;
  }
}

void loop()
{
  delay(3000);
  if (repeat == true)
  {
    for (int i = 0; i < 10; i++)
    {
      Serial.print(i);
      Serial.print(" ");
      Serial.println(myArray[i], 5);
    }
  }
  repeat = false;
}
```

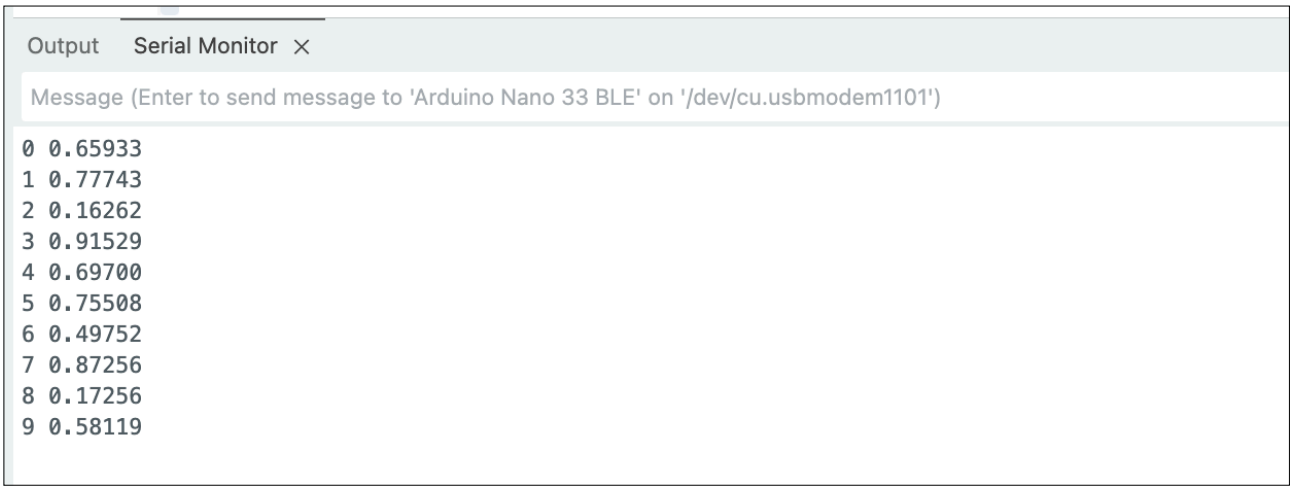
Notes

We do need to tell it we want 5 decimal places. If you keep resetting the board, you will notice that you get the same random numbers each time. There is a way round this.

Code Explanation

`myArray[i] = float(random(100000))/100000;`	Random to 100,000 and divide by 100,000
`Serial.println(myArray[i], 5);`	To five decimal places

Figure A5.14



Sketch A5.15 randomising the random

Because it will pick the same random values each time, we can change the random values by calling a particular `randomSeed()` such as `randomSeed(3)` for instance. This will give you the same values each time but will be different for other `randomSeeds(x)`. One way to randomise the `randomSeed()` is to gate the seed value for one of the pins which is fluctuating slightly all the time (noise in the system), a bit like static on a TV (if you can remember).

Arduino sketch

```
float myArray[10];
bool repeat = true;

void setup()
{
  Serial.begin(9600);
  randomSeed(analogRead(4));
  for (int i = 0; i < 10; i++)
  {
    myArray[i] = float(random(100000))/100000;
  }
}

void loop()
{
  delay(3000);
  if (repeat == true)
  {
    for (int i = 0; i < 10; i++)
    {
      Serial.print(i);
      Serial.print(" ");
      Serial.println(myArray[i], 5);
    }
  }
  repeat = false;
}
```

Notes

Every time you reset the board, you will get a different set of random numbers.

Code Explanation

`randomSeed(analogRead(4));`	Chose a pin that isn't being used for anything

Figure A5.15a

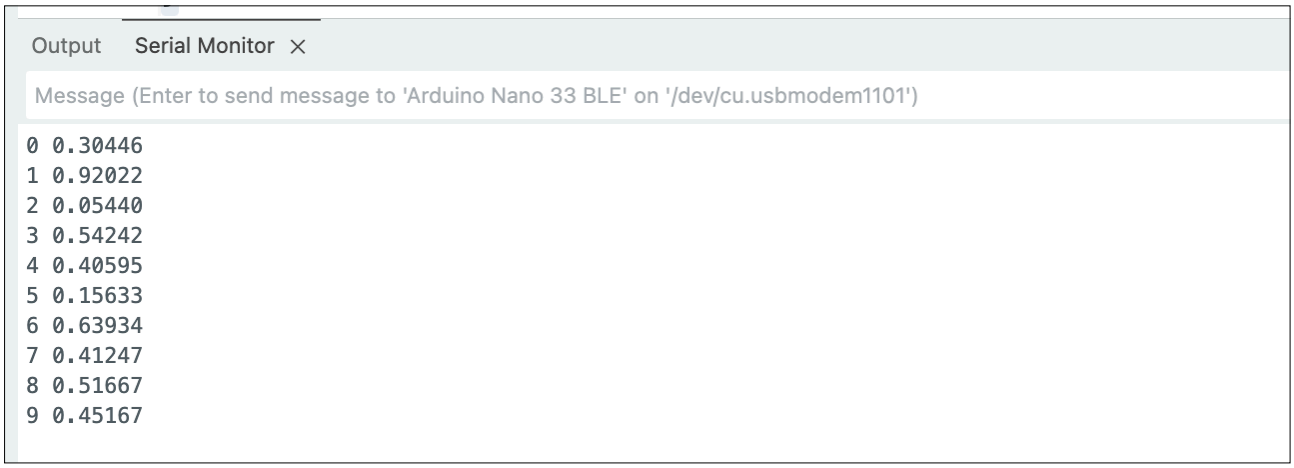
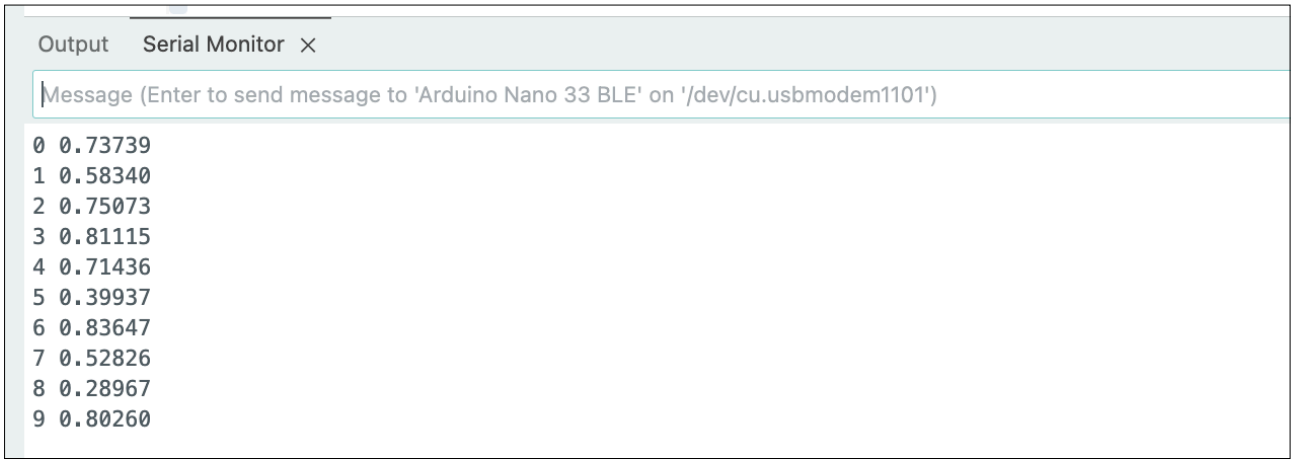
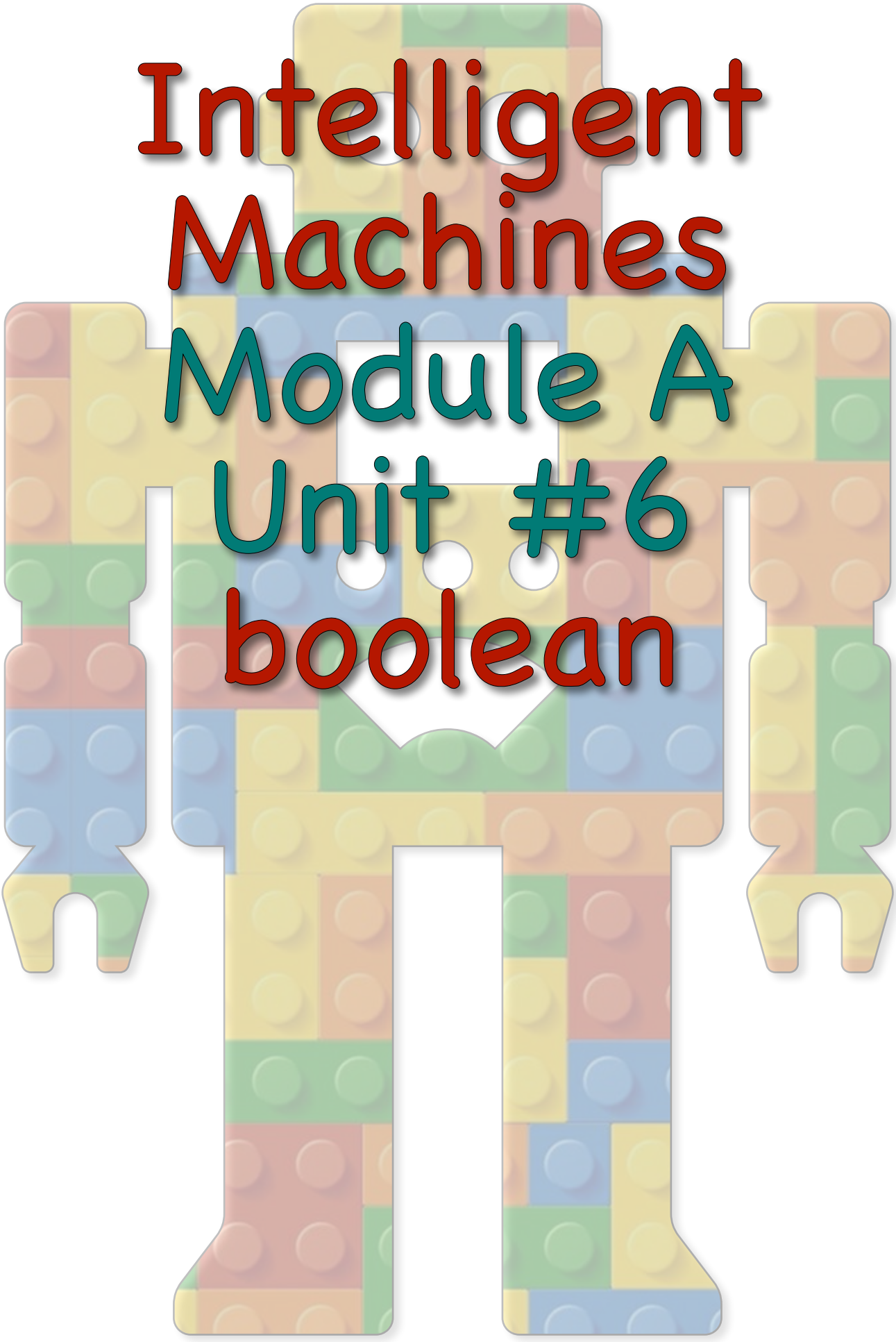


Figure A5.15b





Intelligent Machines Module A Unit #6 boolean

Module A Unit #6: boolean

Boolean is a form of logic which can be described as gates; the most common are AND gates and OR gates. There are more of them, and they form the basics of computing. They are also used widely in higher-level programming. Two symbols are used: && for AND and || for OR (called pipes). Trying to explain it makes it seem quite confusing, and yet it is literally the simplest logic. The best way to get a grasp is simply to use it, and you will see how simple yet powerful it is.

The if() statement can be similar to the while() loop: if something is true or a condition is met, then do something; for instance:

```
if(x < 100 && y > 50)
```

...means if x is less than 100 and y is greater than 50, then do something...

Sketch A6.1 analogWrite()

Going back to built-in LEDs, we can fade the LED using the analogWrite() function. The pins all support PWR (I won't go into the details here just yet), but it means we can assign a value to the pin, so if we give pin 13 a value of 255, that means fully on (100%). If we assign 0 (0%), then it is totally off. Any numbers in between those two values will give a brightness between the two. The following sketch will demonstrate this.

First, full brightness has a value of 255. Notice we put nothing in void setup(). The // means this comments out anything we write afterwards; the computer ignores it.

Arduino sketch

```
void setup()
{
  // nothing here
}

void loop()
{
  analogWrite(13, 255);
}
```

Notes

Nice bright LED.

Code Explanation

analogWrite(13, 255);	This writes a value (255 in this instance) to pin 13. Although it is a digital pin it is treating it as an analog input.

Sketch A6.2 minimum value

Now give it the minimum value of 0. This should be completely off.

Arduino sketch

```
void setup()
{
  // nothing here
}

void loop()
{
  analogWrite(13, 0);
}
```

Notes

Nicely done!

Code Explanation

`analogWrite(13, 0);`	This writes a value (0 in this instance) to pin 13. Although it is a digital pin it is treating it as an analog input.

Sketch A6.3 halfway house

This is somewhere in between, approximately 125. It is a bit hard to see, so we will do an incremental fade-in in the following sketches.

Arduino sketch

```
void setup()
{
  // nothing here
}


void loop()
{
  analogWrite(13, 125);
}
```

Notes

Nice but dim.

Sketch A6.4 short delay

The delay is small, and it is the amount of time (in milliseconds) that it shines at that intensity of brightness.

Arduino sketch

```
void setup()
{
  // nothing here
}


void loop()
{
  analogWrite(13, 125);
  delay(5);
}
```

Notes

Nothing to see here.

Sketch A6.5 increments

Instead of a fixed value, we want to increment it by a value of 1 every 50 milliseconds.

Arduino sketch

```
void setup()
{
  // nothing here
}


void loop()
{
  for (int i = 0; i < 256; i++)
  {
    analogWrite(13, i);
    delay(5);
  }
}
```

Notes

You should see it pulsing, return to 0, then increase to 255.

Code Explanation

`analogWrite(13, i);`	This increments to brightness from 0 through to 255 incrementing the value of i by 1 each iteration of the loop

Sketch A6.6 negative fade

Now to fade back down again. Notice the `i--` at the end of the decreasing fade. What you should have now is the LED getting rapidly brighter and then quickly fading and then brighter repeatedly.

Arduino sketch

```
void setup()
{
  // nothing here
}


void loop()
{
  for (int i = 0; i < 256; i++)
  {
    analogWrite(13, i);
    delay(5);
  }
  for (int i = 255; i > 0; i--)
  {
    analogWrite(13, i);
    delay(5);
  }
}
```

Notes

A nicely pulsing LED, up and down the scale.

Sketch A6.7 boolean operator

❗ New sketch.

Another way to do this is to use a Boolean operator with an `if()` statement. So start with a basic sketch, putting the LED on maximum brightness (255).

Arduino sketch

```
void setup()
{
  // nothing here
}


void loop()
{
  analogWrite(13, 255);
}
```

Notes

Start with it on.

Sketch A6.8 brightness

Now adding a variable for brightness.

Arduino sketch

```
int brightness = 255;


void setup()
{
  // nothing here
}


void loop()
{
  analogWrite(13, brightness);
}
```

Notes

Still the same.

Boolean Operators

This is all to do with logic gates: AND, OR, NOT.

!	logical NOT gate, i.e. a condition where something is not true can be written != (not equal to)
&&	logical AND, i.e. a condition where two things have to be true
||	Logical OR, i.e. a condition where either one of two conditions is true

Sketch A6.9 fadeValue

We want to fade it up and down. So the initial `brightness` we will change to `0` so that when it reaches `255`, then the brightness diminishes and when it is zero, it gets brighter. We are using a boolean operator that is called pipes `||` which means that one `OR` the other has to be true.

The `fadeValue` is an integer variable that is how much it will fade by in steps of `5` in this case.

Arduino sketch

```
int brightness = 0;
int fadeValue = 5;


void setup()
{
  // nothing here
}


void loop()
{
  analogWrite(13, brightness);
  brightness = brightness + fadeValue;
  if (brightness == 0 || brightness == 255)
  {
    fadeValue = -fadeValue;
  }
  delay(50);
}
```

Notes

Pulsing up and down. It toggles the value of `fadeValue` between `-5` and `5`, depending on whether it is starting from `0` or `255`. Just follow the logic.

Code Explanation

`if (brightness == 0 || brightness == 255)`	If brightness is equal to 0 or 255 then change fadeValue from positive to negative or vice versa

Sketch A6.10 without delay

Sometimes it is critical that we don't use the function `delay()` simply because the whole programme stops while `delay()` is operating, and you might want other things to happen while there is a delay. There is a way of doing the blink sketch without using the `delay()` function.

! Starting with a very basic sketch.

Arduino sketch

```
void setup()
{
  pinMode(13, OUTPUT);
}


void loop()
{
  digitalWrite(13, HIGH);
}
```

Notes

Starting again.

Sketch A6.11 change the state

Instead of using HIGH and LOW for on and off, we will give these a variable name so that we can change the state of each. We will call this variable ledState and set it to LOW initially.

Arduino sketch

```
int ledState = LOW;


void setup()
{
  pinMode(13, OUTPUT);
}


void loop()
{
  digitalWrite(13, ledState);
}
```

Notes

We now have control.

Challenge

You can check if it is working by changing the ledState to HIGH and uploading it again.

 Sketch A6.12 a constant interval

We want the LED to blink on/off for one-second intervals, so we have a variable called interval, and we will make it a constant (ie fixed) value of 1000. const is short for constant and means it can never be changed accidentally.

Arduino sketch

```
int ledState = LOW;
const int interval = 1000;


void setup()
{
  pinMode(13, OUTPUT);
}


void loop()
{
  digitalWrite(13, ledState);
}
```

Notes

Nothing to change yet.

Code Explanation

`const int interval = 1000;`	Data types: const short for constant and cannot be changed, int short for integer, float has a decimal point

Data types

You will come across different data types, which are different ways of expressing data or information. They are very important features of coding a microcontroller.

`const`	Short for constant eg pin number
`int`	Short for integer eg 3
`float`	Has a decimal point eg 3.76
`long`	Can use much bigger numbers
`unsigned`	Can only be a positive number
`char`	A type of string (text)

Sketch A6.13 unsigned long

The Arduino starts counting milliseconds as soon as the sketch starts running with the `millis()` function. We can use that to check how much time has elapsed since a particular moment in time. So we need to get the number of milliseconds that have elapsed since a particular point in time.

This number is going to get very big very quickly, and we don't want it to become negative for any reason. So the data type we use is called `long`, and we can make sure it is always positive by defining it as `unsigned`.

Our variable will be called `currentMillis`.

Arduino sketch

```
int ledState = LOW;
const int interval = 1000;


void setup()
{
  pinMode(13, OUTPUT);
}


void loop()
{
  unsigned long currentMillis = millis();
  digitalWrite(13, ledState);
}
```

Notes

A lot to take in for one line of code.

Sketch A6.14 the previous number

We also need another similar variable to hold the previous number of milliseconds. We will call this `previousMillis()`. As the sketch runs, we will subtract the `currentMillis()` from the interval and compare it to the `interval` with an `if()` statement.

Arduino sketch

```
int ledState = LOW;
const int interval = 1000;
unsigned long previousMillis = 0;


void setup()
{
  pinMode(13, OUTPUT);
}


void loop()
{
  unsigned long currentMillis = millis();
  digitalWrite(13, ledState);
}
```

Notes

This is a reference point: `previousMillis`.

Sketch A6.15 checking the difference

We need an `if()` statement to check the difference and reset the `previousMillis()`.

Arduino sketch

```
int ledState = LOW;
const int interval = 1000;
unsigned long previousMillis = 0;


void setup()
{
  pinMode(13, OUTPUT);
}


void loop()
{
  unsigned long currentMillis = millis();
  if (currentMillis - previousMillis >= interval)
  {
    previousMillis = currentMillis;
  }
  digitalWrite(13, ledState);
}
```

Notes

It compares the `interval` to the difference between the current and previous `millis()` values.

Sketch A6.16 toggle the state

And now to change the state depending on whether it is LOW to become HIGH or HIGH to become LOW. It toggles between them. That is why we need to keep track of the state of the LED; if it is off, then we want it on; if on, then off.

Arduino sketch

```
int ledState = LOW;
const int interval = 1000;
unsigned long previousMillis = 0;

void setup()
{
  pinMode(13, OUTPUT);
}

void loop()
{
  unsigned long currentMillis = millis();
  if (currentMillis - previousMillis >= interval)
  {
    previousMillis = currentMillis;
    if (ledState == LOW)
    {
      ledState = HIGH;
    }
    else
    {
      ledState = LOW;
    }
  }
  digitalWrite(13, ledState);
}
```

Notes

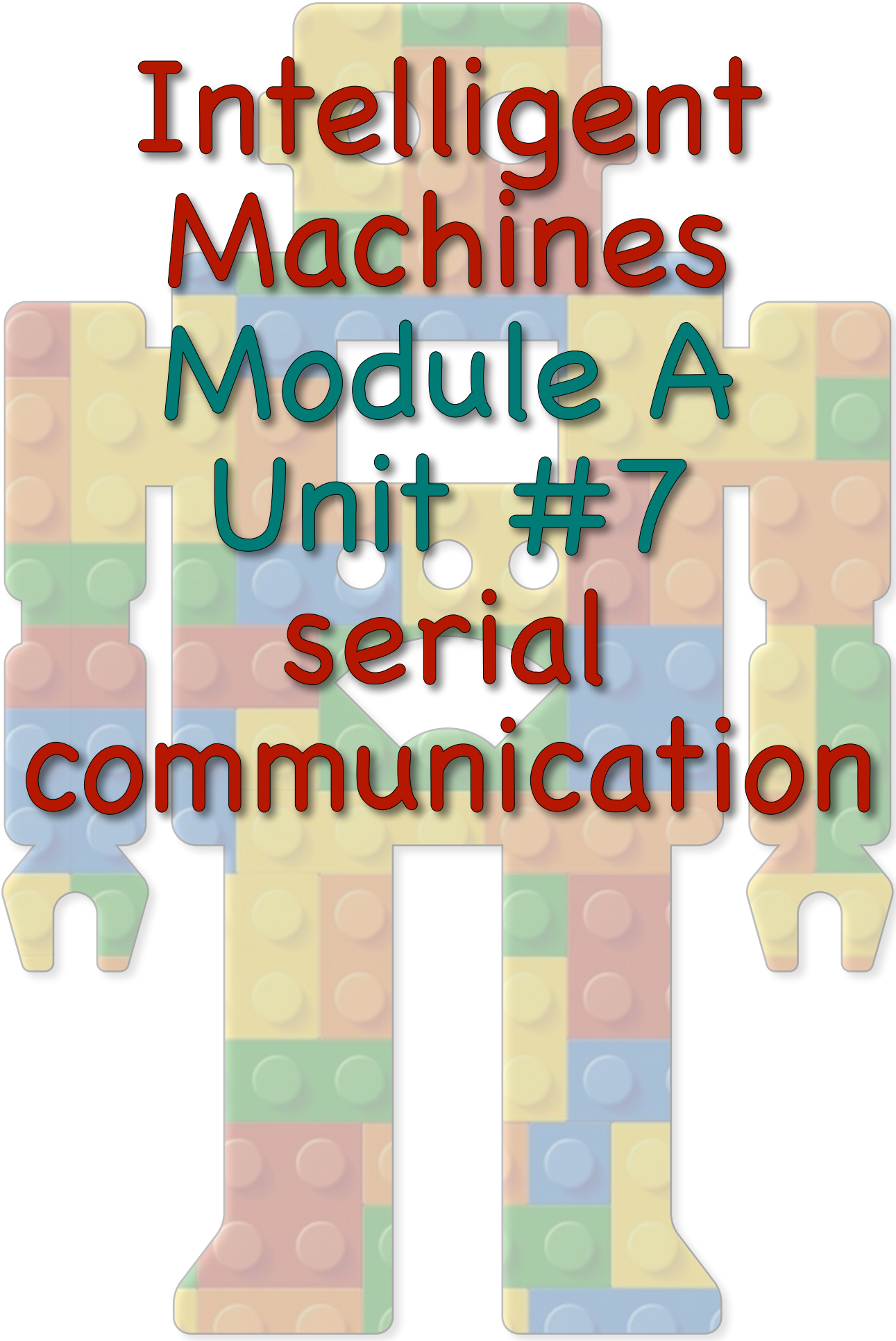
It should blink for one second. Basically, it checks the state every `1000` milliseconds (interval length), counts until that is true.

Challenge

Change the `interval`.

Code Explanation

`if ... else`	Another way of using an if statement where there is more than one condition.

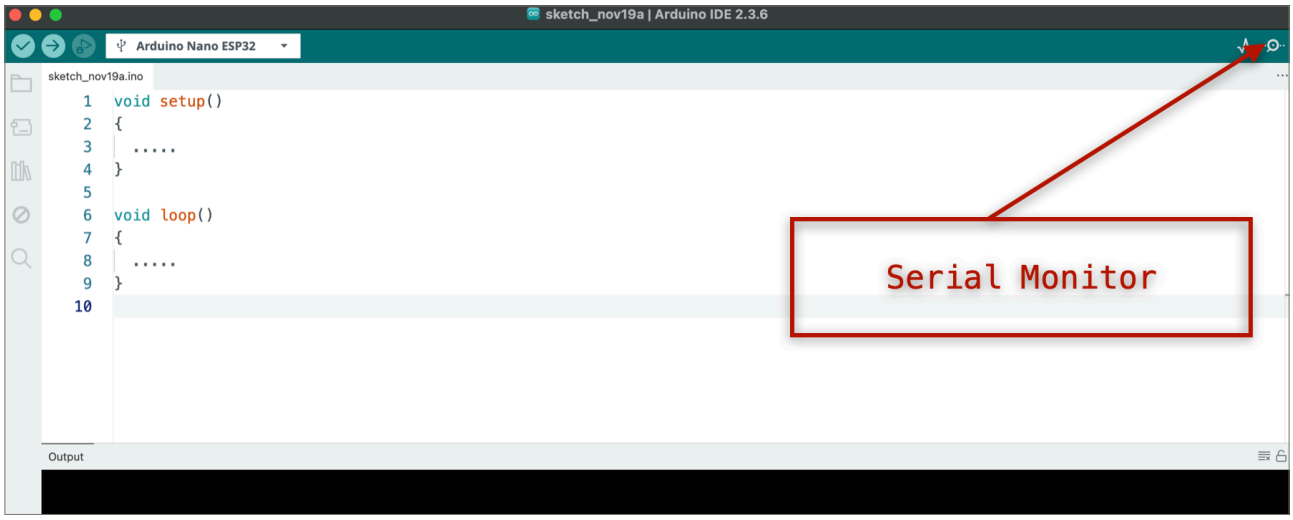


Intelligent Machines Module A Unit #7 serial communication

Module A Unit #7: serial communication

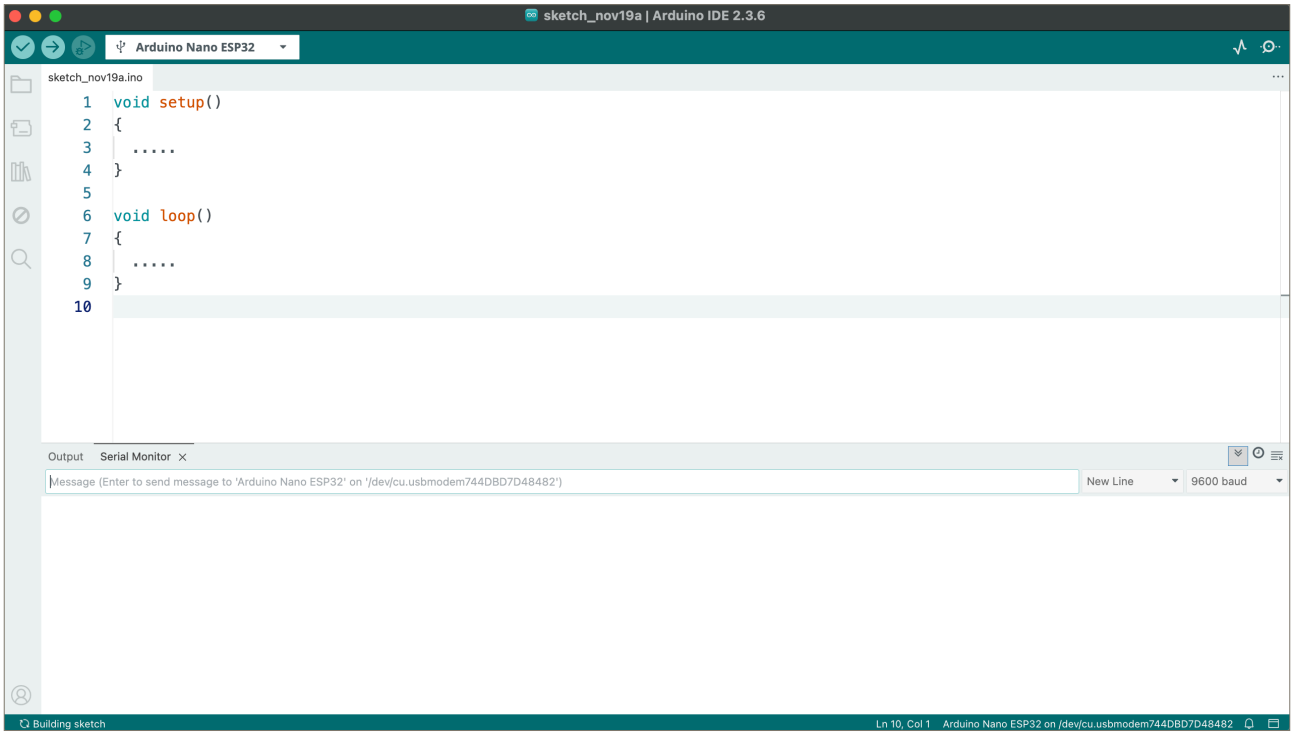
The serial monitor is accessed by clicking on the button shown below. There is a bar at the top, which is where you can type in commands or data to send to the Arduino. The box below is where the data being sent from the Arduino to your computer appears.

Figure 1: serial monitor



You should get a window pop-up like this...

Figure 2: serial monitor revealed



Sketch A7.1 Hello World!

We can write to the serial monitor. In this first simple example, we will just send the message Hello World!. We need to set up communication in the void setup() function.

The baud rate signifies the data rate in bits per second. The default baud rate in Arduino is 9600 bps (bits per second). So, upload the sketch, then click on the icon (the one that looks like a magnifying glass) near the top right-hand corner.

Arduino sketch

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

void loop()
{
  Serial.print("Hello World!");
}
```

Notes

The problem is that it scrolls it across the window. We can get it to scroll downwards in the next sketch. You might need to cancel the serial monitor (click on the x) and re-select it.

Challenge

Try some other word or phrase.

Code Explanation

`Serial.begin(9600);`	Sets up the serial communication, the 9600 is the rate it communicates
`Serial.print();`	Prints a number or text to the monitor

Figure A7.1


Output
Serial Monitor
Message (Enter to send message to 'Arduino Nano ESP32' on '/dev/cu.usbmodem744DBD7D48482')
Hello World!Hello World!Hello World!Hello World!Hello World!Hello World!Hello World

Sketch A7.2 new line

Now scrolling downwards.

Arduino sketch

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

void loop()
{
  Serial.println("Hello World!");
}
```

Notes

Remember to use double quotes `"...."` for text.

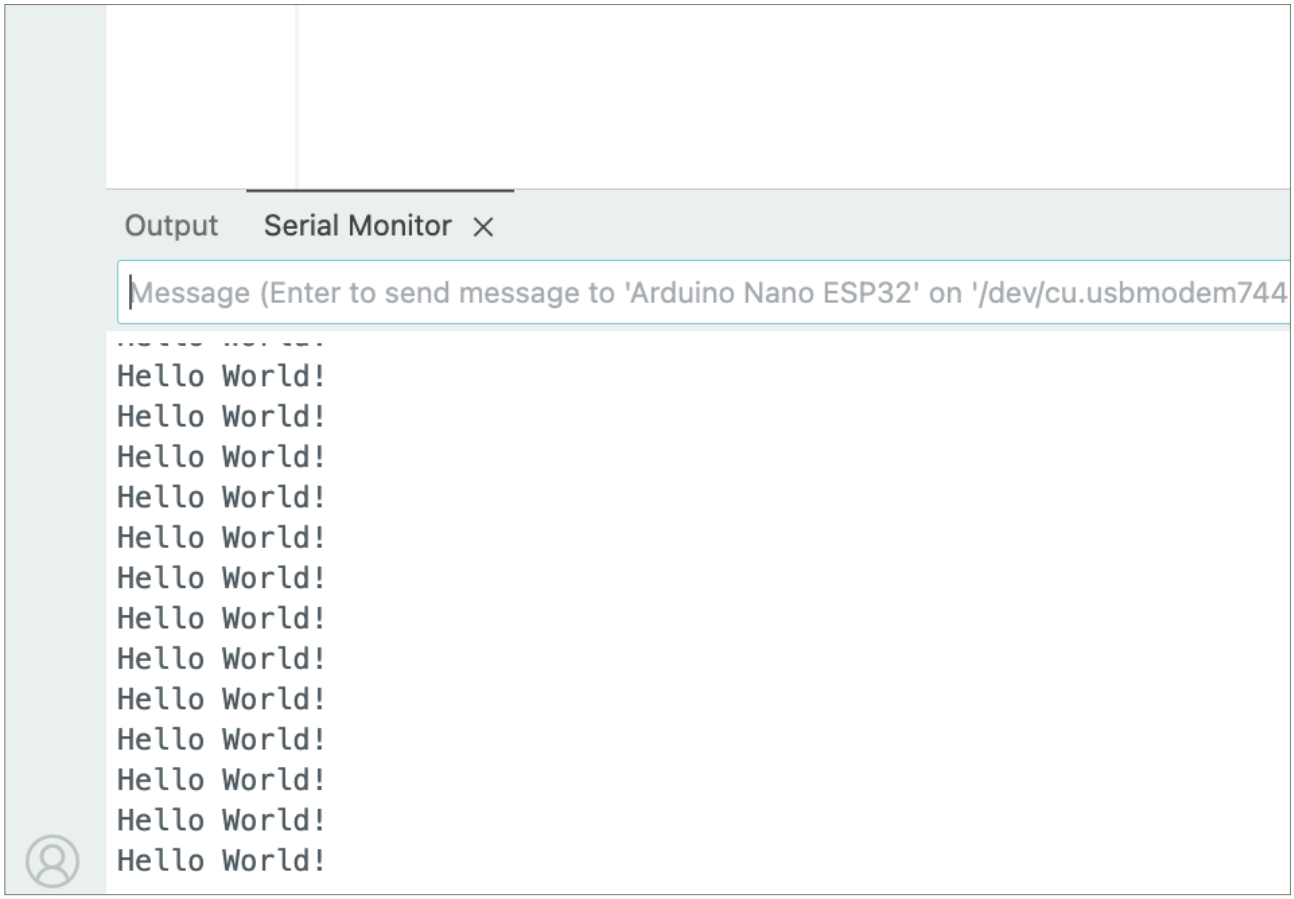
Challenge

See what happens if you remove the speech marks.

Code Explanation

`Serial.println();`	Prints on a new line if it is repeated on each iteration

Figure A7.2


Output
Serial Monitor
Message (Enter to send message to 'Arduino Nano ESP32' on '/dev/cu.usbmodem744
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!

Sketch A7.3 LED on/off

❗ A newish sketch

Now, to use the basic blink sketch to tell us when the LED is on and then off.

Arduino sketch

```
void setup()
{
  Serial.begin(9600);
  pinMode(13, OUTPUT);
}

void loop()
{
  digitalWrite(13, HIGH);
  Serial.println("LED on");
  delay(1000);
  digitalWrite(13, LOW);
  Serial.println("LED off");
  delay(1000);
}
```

Figure A7.3


14 delay(1000);
15 }
Output
Serial Monitor
Message (Enter to send message to 'Arduino Nano ESP32' on '/dev/cu.usbmodem744DBD7D48482')
LED off
LED on
LED off
LED on
LED off
LED on
LED off
LED on
LED off
LED on
LED off
LED on
LED off

Sketch A7.4 return Hello World

❗ A new sketch

We are going to type `Hello World!` into the `message box`, send it to the Arduino (by pressing return). The Arduino will receive it and send it back in the `serial monitor`.

Arduino sketch

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

void loop()
{
  if (Serial.available() > 0)
  {
    char input = Serial.read();
    Serial.print(input);
  }
}
```

Notes

This may seem like a pointless exercise, but I want to introduce the concept of sending information (text or otherwise) via the serial connection to affect change.

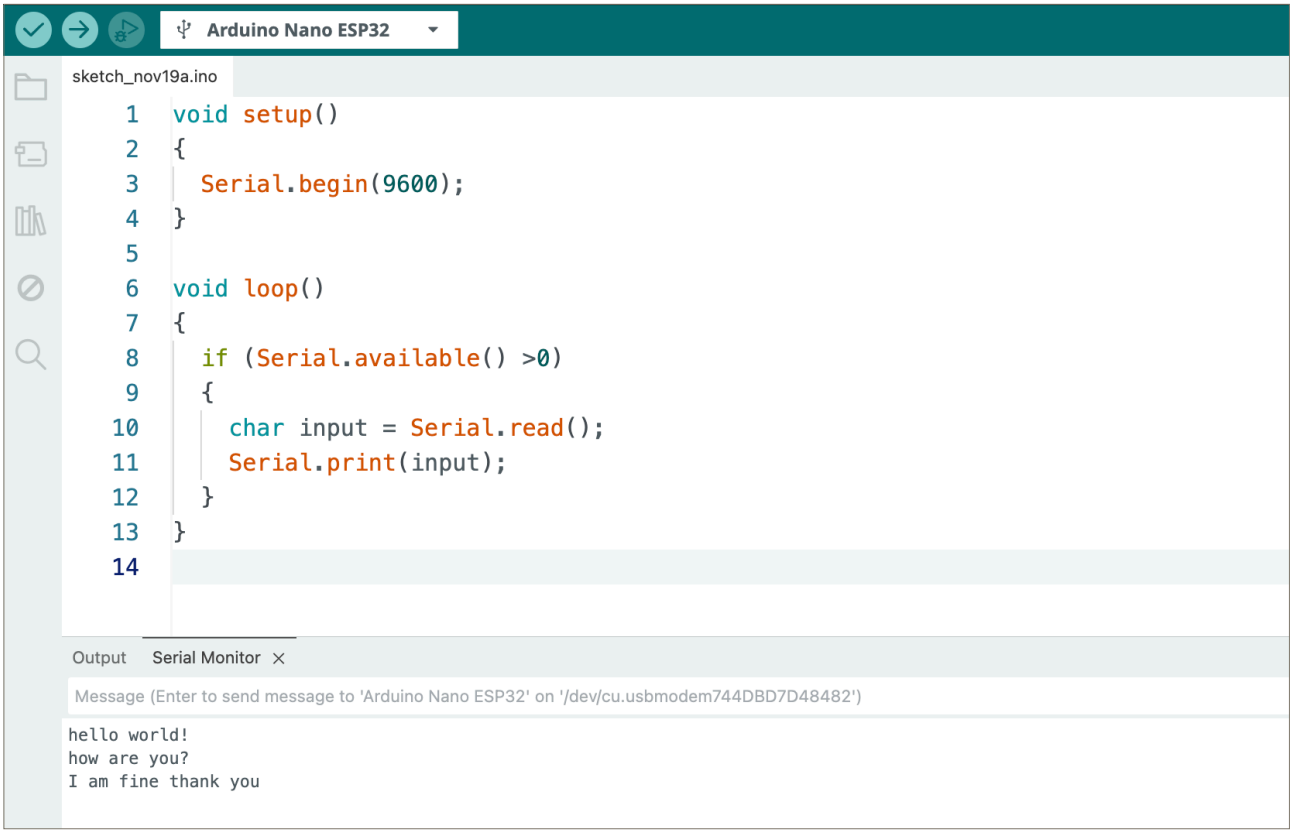
Challenge

Type in your own message.

Code Explanation

`Serial.available()`	Checks to see if there is anything
`char input = Serial.read();`	Takes each character one at a time as it reads incoming data

Figure A7.4


Arduino Nano ESP32
sketch_nov19a.ino
1 void setup()
2 {
3 Serial.begin(9600);
4 }
5
6 void loop()
7 {
8 if (Serial.available() >0)
9 {
10 char input = Serial.read();
11 Serial.print(input);
12 }
13 }
14
Output
Serial Monitor
Message (Enter to send message to 'Arduino Nano ESP32' on '/dev/cu.usbmodem744DBD7D48482')
hello world!
how are you?
I am fine thank you

 Sketch A7.5 controlling an LED

We are going to use this feature to turn the LED on and off. So that when we type H, it turns the LED on, and L to turn the LED off. As before, we need to check if there is any data coming in.

Arduino sketch

```
void setup()
{
  Serial.begin(9600);
  pinMode(13, OUTPUT);
}

void loop()
{
  if (Serial.available() > 0)
  {
    char input = Serial.read();
    if (input == 'H')
    {
      digitalWrite(13, HIGH);
      Serial.println("LED on");
    }
    if (input == 'L')
    {
      digitalWrite(13, LOW);
      Serial.println("LED off");
    }
  }
}
```

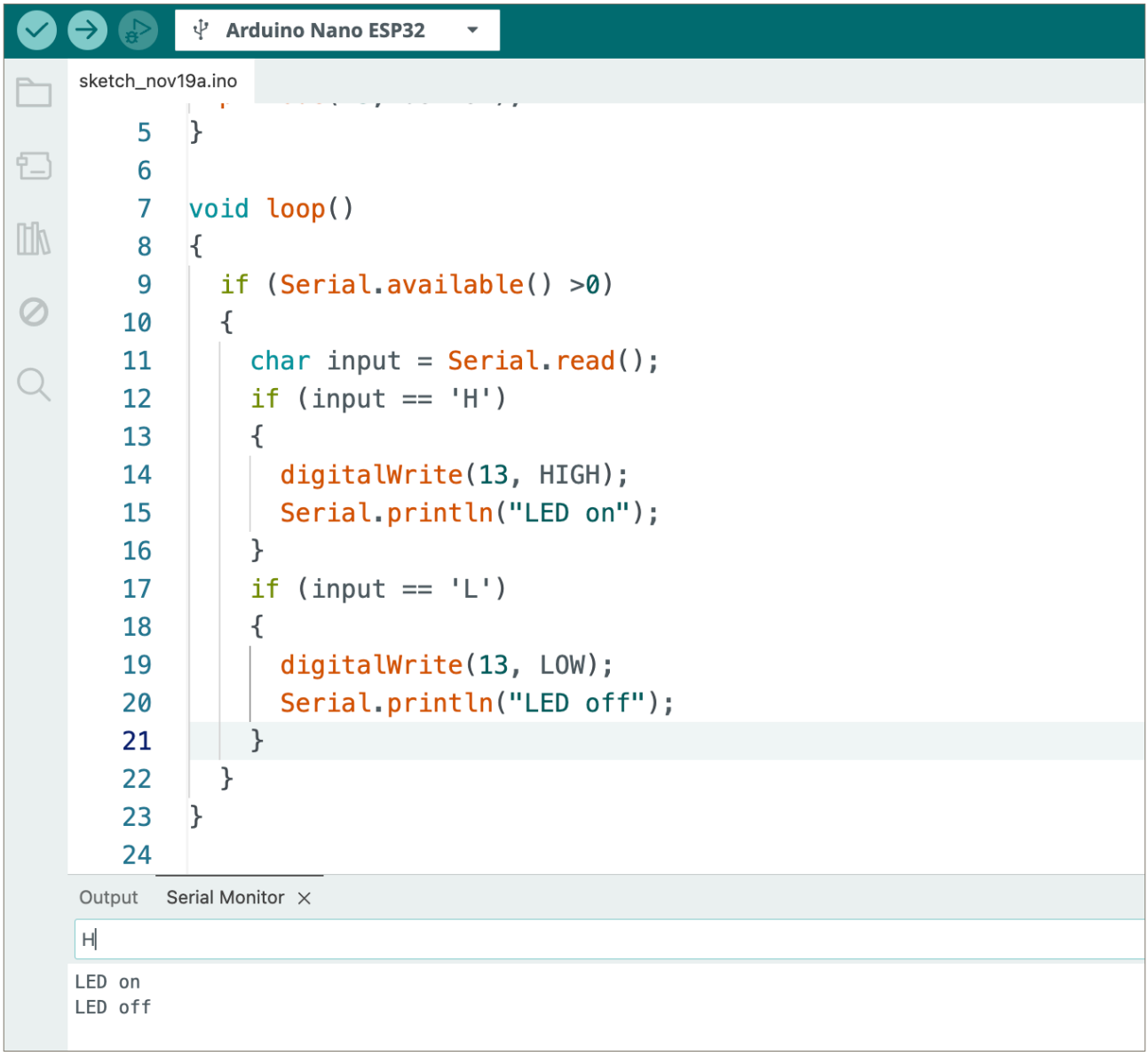
Notes

The LED should respond to H and L being typed in (press return to send).

Challenge

What happens with lowercase h and l? How could you mitigate this?

Figure A7.5


Arduino Nano ESP32
sketch_nov19a.ino
}
void loop()
{
if (Serial.available() >0)
{
char input = Serial.read();
if (input == 'H')
{
digitalWrite(13, HIGH);
Serial.println("LED on");
}
if (input == 'L')
{
digitalWrite(13, LOW);
Serial.println("LED off");
}
}
}
Output
Serial Monitor
H
LED on
LED off

Sketch A7.6 levels of brightness

Instead of switching the LED on and off, we can give the brightness a value. To input the value of the brightness, we use the parseInt() function. To test that it works, send in 255, (press return), then send 1, (press return), and then send 255 again followed by 100. You can therefore switch it between these values to control the brightness.

Arduino sketch

```
void setup()
{
  Serial.begin(9600);
  pinMode(13, OUTPUT);
}

void loop()
{
  if (Serial.available() > 0)
  {
    int input = Serial.parseInt();
    if (input != 0)
    {
      analogWrite(13, input);
    }
  }
}
```

Notes

This, again, is a simple illustration. The reason for having the line of code: if(input != 0) . . . is because the Serial Monitor sends a 0 value on a continuous loop, and this solves that problem; otherwise, it would light up only briefly.

Challenge

Can you think of something else you could control?

Code Explanation

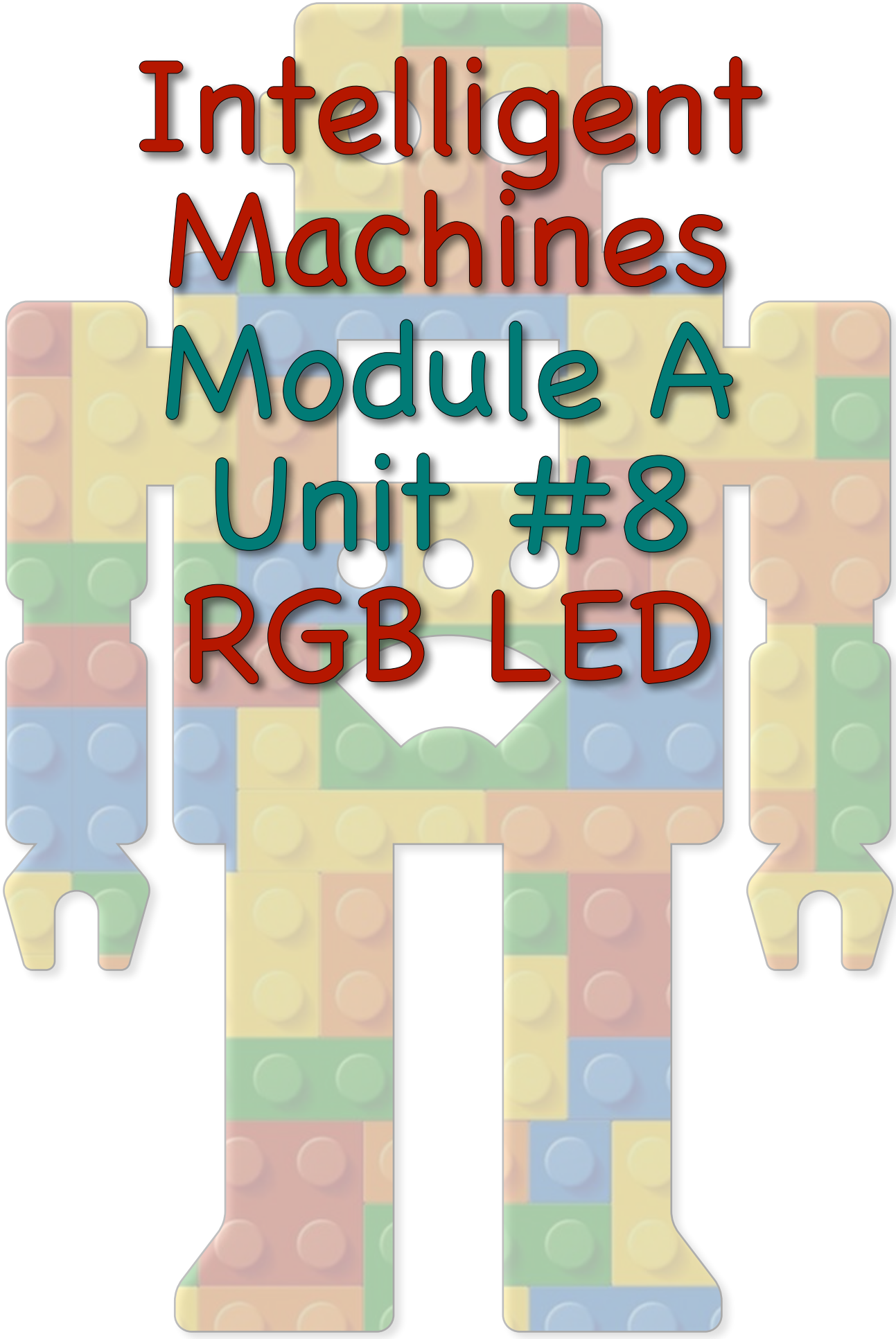
`int input = Serial.parseInt();`	The input is an integer (int) not a letter (char) hence the parseInt() function

Intelligent Machines

Module A

Unit #8

RGB LED



 Module A Unit #8: RGB LED

The board does have another built-in LED; it is an RGB LED, that means it has a red (LEDR) component, green (LEDG) component, and a blue (LEDB) component. So you can have a red, blue, or green LED, and you can combine them to create colours like yellow, cyan, magenta, or white. They are not PWM pins, so we cannot mix them together to create lots of colours as you can with p5.js.

Sketch A8.1 built-in RGB LED

To start with, we will have red on for one second, green on for one second, blue on for one second, and all off for one second. This is slightly counterintuitive: for a particular colour LED to be on, it has to be set to LOW; conversely, to switch a colour LED off, it has to be set to HIGH. I know, confusing, isn't it?

Arduino sketch

```
void setup()
{
  pinMode(LEDR, OUTPUT);
  pinMode(LEDG, OUTPUT);
  pinMode(LEDB, OUTPUT);
}

void loop()
{
  // RED
  digitalWrite(LEDR, LOW);
  digitalWrite(LEDG, HIGH);
  digitalWrite(LEDB, HIGH);
  delay(1000);

  // GREEN
  digitalWrite(LEDR, HIGH);
  digitalWrite(LEDG, LOW);
  digitalWrite(LEDB, HIGH);
  delay(1000);

  // BLUE
  digitalWrite(LEDR, HIGH);
  digitalWrite(LEDG, HIGH);
  digitalWrite(LEDB, LOW);
  delay(1000);

  // RGB OFF
  digitalWrite(LEDR, HIGH);
  digitalWrite(LEDG, HIGH);
  digitalWrite(LEDB, HIGH);
```

```
  delay(1000);
}
```

Notes

They should cycle through those RGB colours and then stop.

Code Explanation

`pinMode(LEDR, OUTPUT);`	LEDR is the red component of the LED, initialise it as an output
`digitalWrite(LEDR, LOW);`	Writes to the LEDR that it is to switch it on
`digitalWrite(LEDR, HIGH);`	Writes to the LEDR that it is to switch it off
`delay(1000);`	Waits for one second

Sketch A8.2 more colours

Adding in all the other colours for good measure.

Arduino sketch

```
void setup()
{
  pinMode(LEDR, OUTPUT);
  pinMode(LEDG, OUTPUT);
  pinMode(LEDB, OUTPUT);
}

void loop()
{
  // RED
  digitalWrite(LEDR, LOW);
  digitalWrite(LEDG, HIGH);
  digitalWrite(LEDB, HIGH);
  delay(1000);

  // GREEN
  digitalWrite(LEDR, HIGH);
  digitalWrite(LEDG, LOW);
  digitalWrite(LEDB, HIGH);
  delay(1000);

  // BLUE
  digitalWrite(LEDR, HIGH);
  digitalWrite(LEDG, HIGH);
  digitalWrite(LEDB, LOW);
  delay(1000);

  // YELLOW
  digitalWrite(LEDR, LOW);
  digitalWrite(LEDG, LOW);
  digitalWrite(LEDB, HIGH);
  delay(1000);

```

```
  // MAGENTA
  digitalWrite(LEDR, LOW);
  digitalWrite(LEDG, HIGH);
  digitalWrite(LEDB, LOW);
  delay(1000);

  // CYAN
  digitalWrite(LEDR, HIGH);
  digitalWrite(LEDG, LOW);
  digitalWrite(LEDB, LOW);
  delay(1000);

// WHITE
  digitalWrite(LEDR, LOW);
  digitalWrite(LEDG, LOW);
  digitalWrite(LEDB, LOW);
  delay(1000);

  // RGB OFF
  digitalWrite(LEDR, HIGH);
  digitalWrite(LEDG, HIGH);
  digitalWrite(LEDB, HIGH);
  delay(1000);
}
```

Notes

Just for fun. If you could alter the value of each one, then you could create more colours, but they are either HIGH or LOW, unfortunately.

Sketch A8.3 looping round the array

❗ New sketch.

Instead of hard coding everything, let's see if we can achieve the same thing but with just fewer lines of code. To achieve this, we will need to use `for()` loops and `arrays`. The array will contain the names of the LEDs. We are treating them as integers, even though they are clearly not numbers. We start with all the LED colours `off` (`HIGH`), then switch one LED colour `on` (`LOW`), one at a time.

Arduino sketch

```
int led[] = {LEDR, LEDG, LEDB};


void setup()
{
  for (int i = 0; i < 3; i++)
  {
    pinMode(led[i], OUTPUT);
  }
}


void loop()
{
  for (int i = 0; i < 3; i++)
  {
    digitalWrite(LEDR, HIGH);
    digitalWrite(LEDG, HIGH);
    digitalWrite(LEDB, HIGH);
    digitalWrite(led[i], LOW);
    delay(1000);
  }
}
```

Notes

Cycles through the colours.

Challenges

1. How would you remove them all for one second?
2. What would happen if you didn't have them all off (HIGH) in the `for()` loop?

Code Explanation

`int led[] = {LEDR, LEDG, LEDB};`	Create an array called led. Put three elements into that array, the names of the LEDs.

`for (int i = 0; i < 3; i++)`
`{`
`  pinMode(led[i], OUTPUT);`
`}`
	A for loop starts at 0 (i = 0), adds 1 each time (i++) and stops before it gets to 3 (i < 3).

`pinMode(led[i], OUTPUT);`	led[i] is the index reference in the array. index[0] is LEDR, index[1] is LEDG and index[2] is LEDB

Sketch A8.4 random selection

Another solution: Now it will select an LED at random.

! Remove the `for()` loop and the curly brackets.

Arduino sketch

```
int led[] = {LEDR, LEDG, LEDB};

void setup()
{
  for (int i = 0; i < 3; i++)
  {
    pinMode(i, OUTPUT);
  }
}

void loop()
{
  int i = random(0, 3);
  digitalWrite(LEDR, HIGH);
  digitalWrite(LEDG, HIGH);
  digitalWrite(LEDB, HIGH);
  digitalWrite(led[i], LOW);
  delay(1000);
}
```

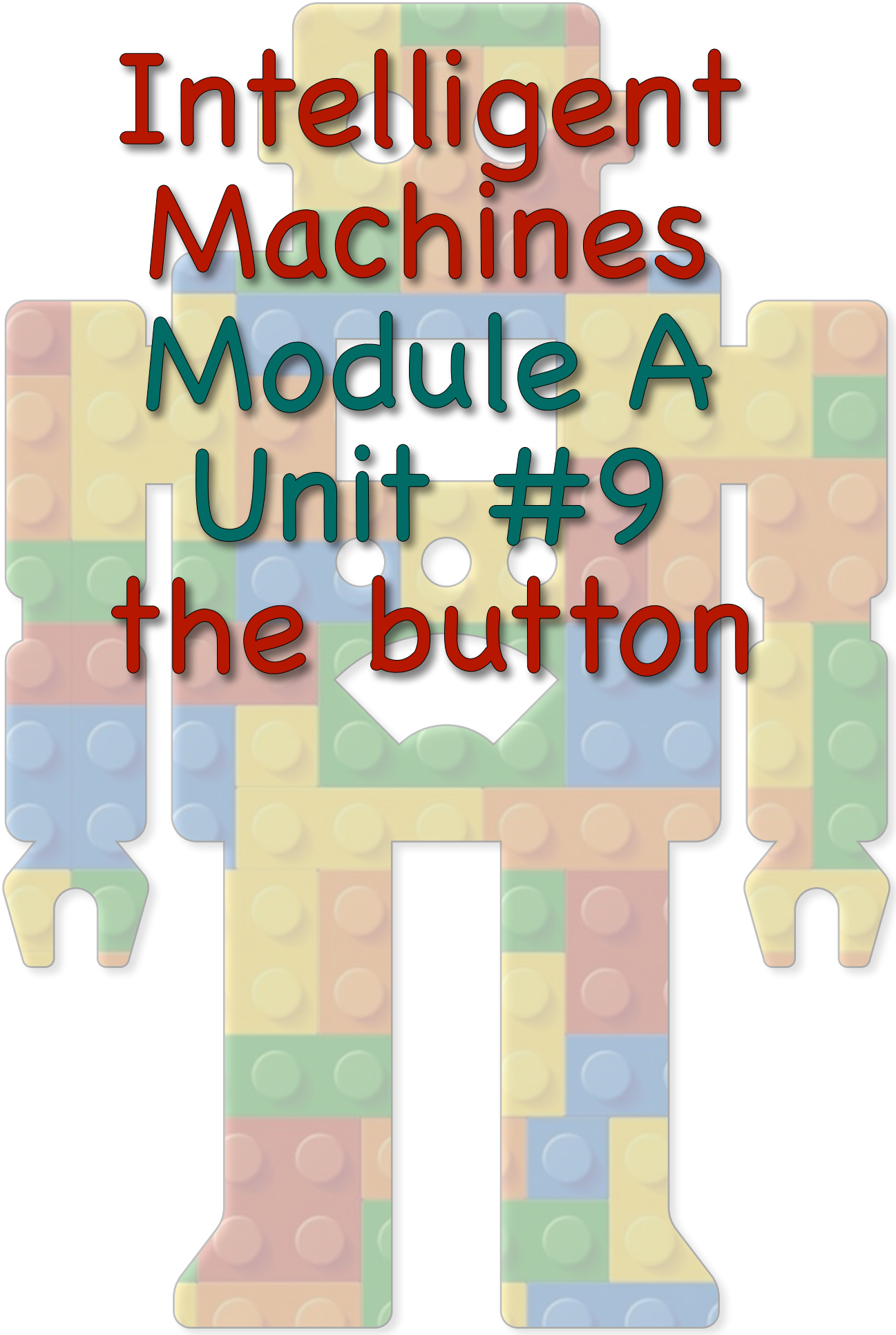
Notes

It will select the same one more than once. Notice that it is random between 0 and 3, that is, it is inclusive of 0 but up to 3 but not including 3, as we only have three LEDs.

Challenge

For a pleasing effect, make the delay much smaller, e.g. 100.

Intelligent Machines
Module A
Unit #9
the button



Module A Unit #9: the button

The button is a tactile or momentary push button. It makes contact with a metal plate when you push down and completes the circuit. When you stop pushing, it releases and is no longer in contact and hence breaks the circuit. See Fig. 2.

As you can see, it has four pins protruding from the body of the button and a black button on top. This kind of button fits nicely on the breadboard. The pins are connected as shown in Fig. 2. When you press the button, you connect the pins from left to right (as seen in the diagram below). The pins top to bottom are already connected internally.

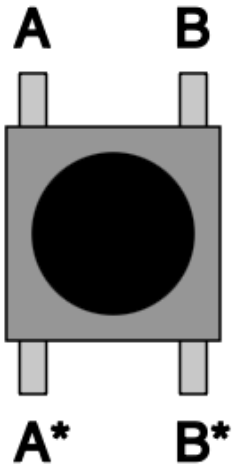
To use this button, see Fig.1 (rather than a button module), we need a resistor. The beauty of the Arduino is that it has a built-in resistor we can use with the button (but not with an LED!). It is called a pull-up resistor, more on that later.

Figure 1: button

How the button is connected

As you look down at the button, it has four legs (pins). The pins A and A* are connected permanently, as are B and B*. When you press the button, A and B are connected, and A* and B* are also connected.

Figure 2: looking down



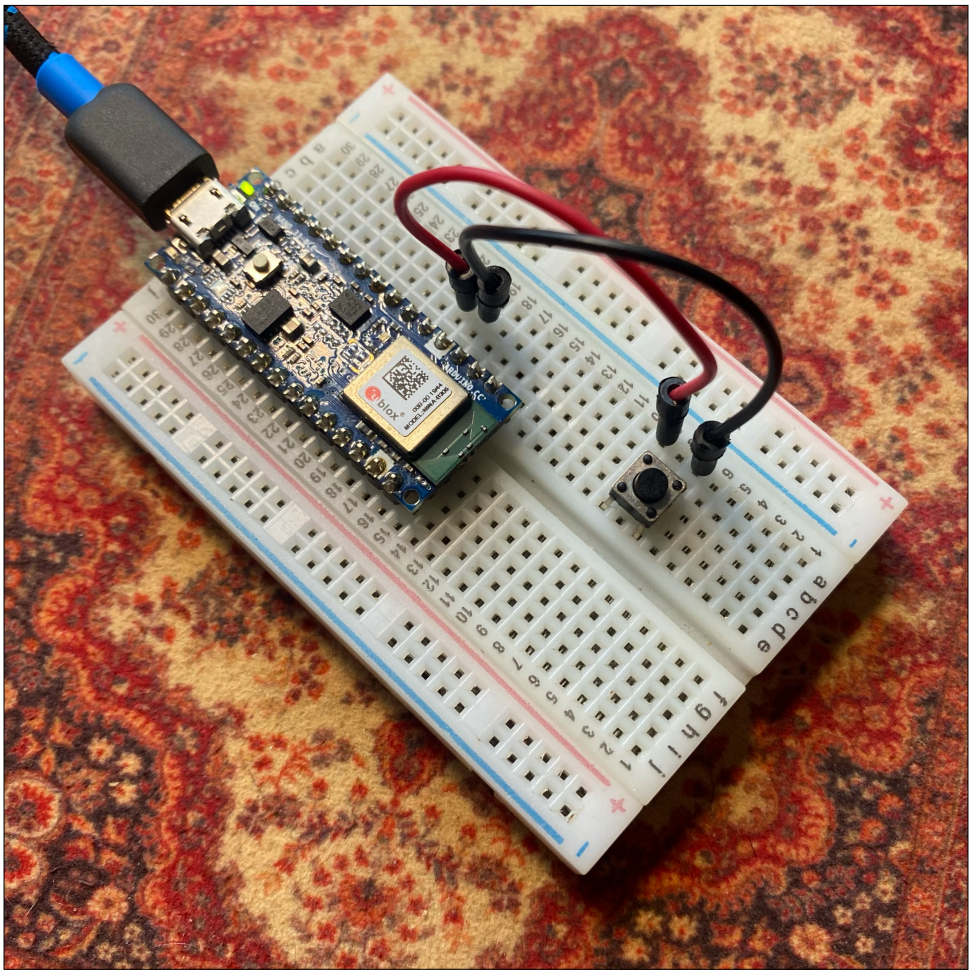
What you will need

The full list is here. Look at the image below and the wiring diagram (fig.5).

1 x Arduino Nano 33 BLE
1 x breadboard
1 x button
2 x male-to-male jumper leads

You could add a resistor in series with the button, but the Nano has a built-in resistor that you can pull up. This saves you the bother. The downside is that the logic (`HIGH` = off and `LOW` = on) is reversed.

Figure 3: component setup



Circuit Diagram for the button

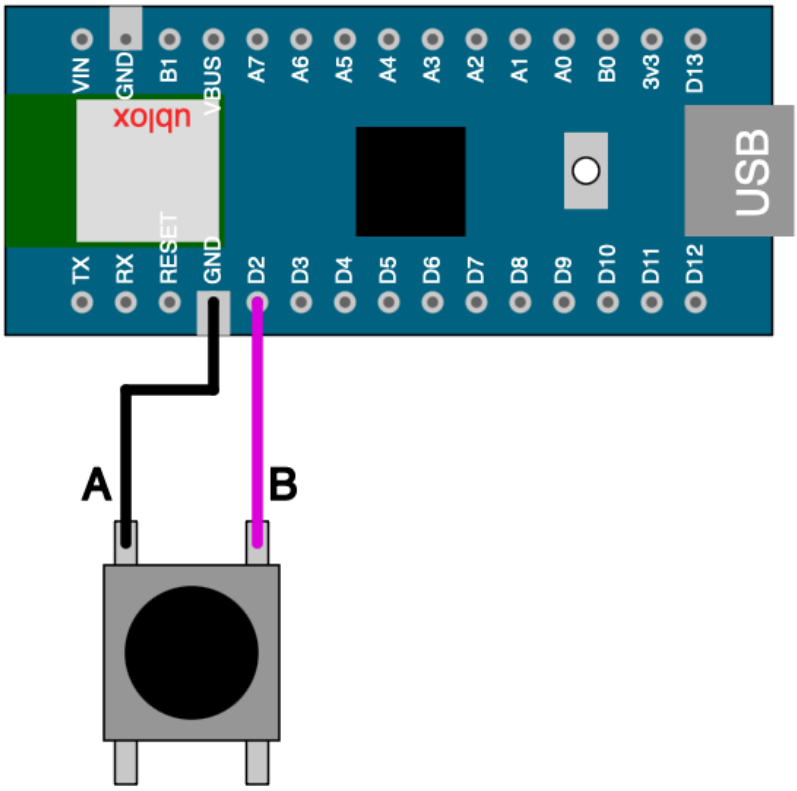
You will need two wires to connect the button to the device.

Button Pins	Arduino Pins
A	GND
B	2 (digital pin D2)

We are going to add the button to the breadboard. You can put it anywhere away from the board itself.

See Fig.4 below. The wiring diagram shows the connections.

Figure 4: wiring diagram



Sketch A9.1 LED button

Connect the button as shown, then after writing the code and uploading it to the Arduino, press the button. The LED should come on when pressed and off when released. When using the pull-up resistor, the default (not pressed) state is HIGH. This is similar to the RGB LED, still a little bit counterintuitive; if you use an external resistor, the opposite (more logical) is true.

Arduino sketch

```
int ledPin = 13;
int buttonPin = 2;

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

void loop()
{
  int button = digitalRead(buttonPin);
  if (button != LOW)
  {
    digitalWrite(ledPin, LOW);
  }
  else
  {
    digitalWrite(ledPin, HIGH);
  }
}
```

Notes

As an alternative, I have given the button and the LED variable names. This allows you to make changes once rather than sifting through the whole code and changing it multiple times.

Code Explanation

`pinMode(buttonPin, INPUT_PULLUP);`	Using the pull-up (internal) resistor
`if (button != LOW)`	The != means not. If the button is not LOW (pressed)...

Sketch A9.2 LED toggle

In this sketch, we are doing more than just switching it on and off with the button, but toggling it so that on one press the LED is on and on the next press of the button the LED switches off. This is more difficult than the previous sketch. Hold the button for a second each time.

! This works very badly because of a condition known as bounce (we will look at that in the next sketch).

Arduino sketch

```
int ledPin = 13;
int buttonPin = 2;
bool ledState = LOW;
bool lastButtonState = HIGH;
bool currentButtonState = HIGH;

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

void loop()
{
  lastButtonState = currentButtonState;
  currentButtonState = digitalRead(buttonPin);
  if(lastButtonState == LOW && currentButtonState == HIGH)
  {
    ledState = !ledState;
    digitalWrite(ledPin, ledState);
  }
}
```

Notes

I will be honest with you, this requires some logical thought to work out what is happening. In simple terms, when you press the button, the state of current and previous end up being LOW, but when the button is released, the current then becomes HIGH, thus changing the ledState from either HIGH to LOW or LOW to HIGH.

Just work through the sketch to follow the logic; the logic isn't flawed, but the button is. The problem is that the contacts, as you press the button, jump or bounce and give false readings. The next sketch addresses this problem by taking into account the bounce.

Code Explanation

`bool ledState = LOW;`	This boolean variable defines the state of the LED, initially it is set to on.
`bool lastButtonState = HIGH;`	We introduce a boolean previous state to compare to the current.
`bool currentButtonState = HIGH;`	This boolean state is also the current and is used to compare to the the previous.
`lastButtonState = currentButtonState;`	This updates the state of the previous state.
`currentButtonState = digitalRead(buttonPin);`	The latest state is updated by reading if the button has been pressed.
`if(lastButtonState == LOW && currentButtonState == HIGH)`	When the button is pressed the state is LOW but when the the button is released it is HIGH.
`ledState = !ledState;`	When the above condition is true: the lastButtonState is LOW and the currentButtonState is HIGH the LED is turned on if off and on if off.

Sketch A9.3 debounce

I recommend starting a new sketch; too many changes to highlight. To tackle the bounce problem, we need to introduce some sort of delay to counter that. We will call that debounceDelay.

Arduino sketch

```
int ledPin = 13;
int buttonPin = 2;
int ledState = LOW;
int buttonState = LOW;
int lastButtonState = LOW;
int currentButtonState = LOW;

long lastDebounceTime = 0;
long debounceDelay = 50;

void setup()
{
  pinMode(ledPin, OUTPUT);
  pinMode(buttonPin, INPUT_PULLUP);
  digitalWrite(ledPin, ledState);
}

void loop()
{
  currentButtonState = digitalRead(buttonPin);
  if (currentButtonState != lastButtonState)
  {
    lastDebounceTime = millis();
  }
  if ((millis() - lastDebounceTime) > debounceDelay)
  {
    if (currentButtonState != buttonState)
    {
      buttonState = currentButtonState;
      if (buttonState == HIGH)
      {
        ledState = !ledState;
```

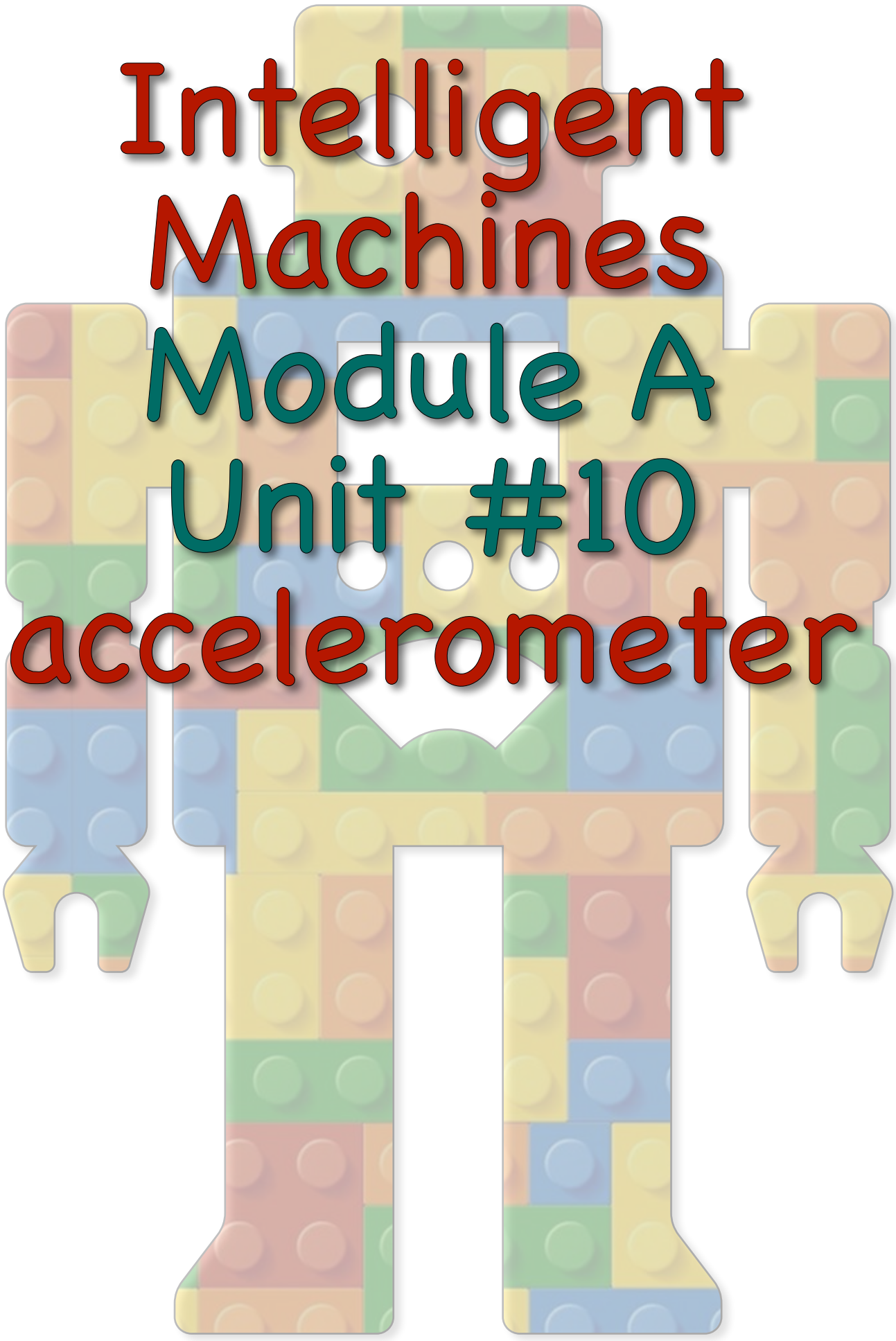
```
      }
    }
  }
  digitalWrite(ledPin, ledState);
  lastButtonState = currentButtonState;
}
```

Notes

This is more about Boolean logic than the code. But all that code is just to toggle an LED on or off. This is what coding is all about: problem solving. This is a workaround for a hardware issue.

Code Explanation

`long lastDebounceTime = 0;`	These two lines of code are the key to this working, they have to be long because they use the millis() function and the number can get big very quickly.
`long debounceDelay = 50;`	



Intelligent Machines Module A Unit #10 accelerometer

Module A Unit #10: the accelerometer

Utilising the built-in BMI270 and BMM150 sensor module (IMU)

IMU stands for: inertial measurement unit. It is an electronic device that measures and reports a body's specific force, angular rate, and the orientation of the body, using a combination of accelerometers, gyroscopes, and oftentimes magnetometers.

The IMU system on the Nano 33 BLE Rev2 is a combination of two modules, the 6-axis BMI270, and the 3-axis BMM150, that together add up to a combined 9-axis IMU system that can measure acceleration, as well as rotation and magnetic fields all in 3D space.

The Arduino BMI270_BMM150 library allows us to use the Nano 33 BLE Rev2 IMU system without having to go into complicated programming. The library takes care of the sensor initialisation and sets its values as follows:

accelerometer
the range is set at [-4, +4]g +/-0.122 mg and the output data rate is fixed at 104 Hz.

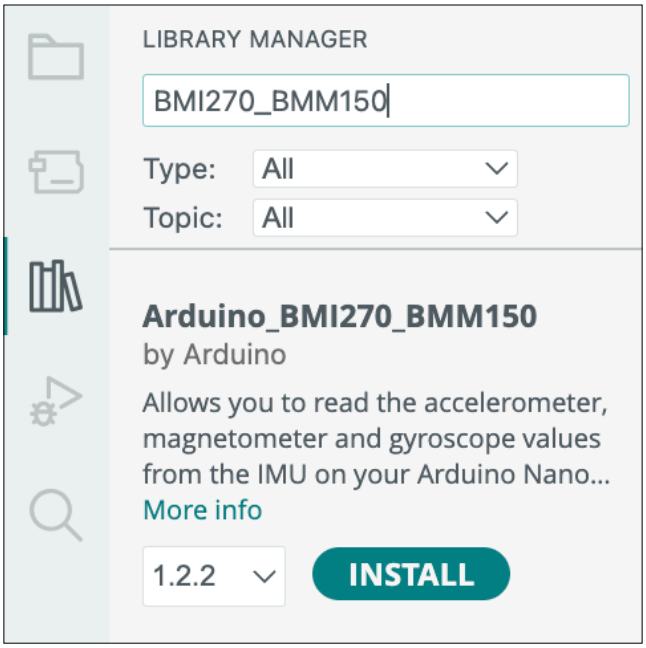
gyroscope
the range is set at [-2000, +2000] dps +/-70 mdps and the output data rate is fixed at 104 Hz.

magnetometer
the range is set at [-400, +400] uT +/-0.014 uT and the output data rate is fixed at 20 Hz.

The BMI270_BMM150 Library

To use the library you will need to install the library by first clicking on the button on the left-hand side that looks like a pile of books (hint: `library`), and then type in the name of the sensor, find it, and click on INSTALL as shown in Fig. 1. You may need to do this for all the libraries needed.

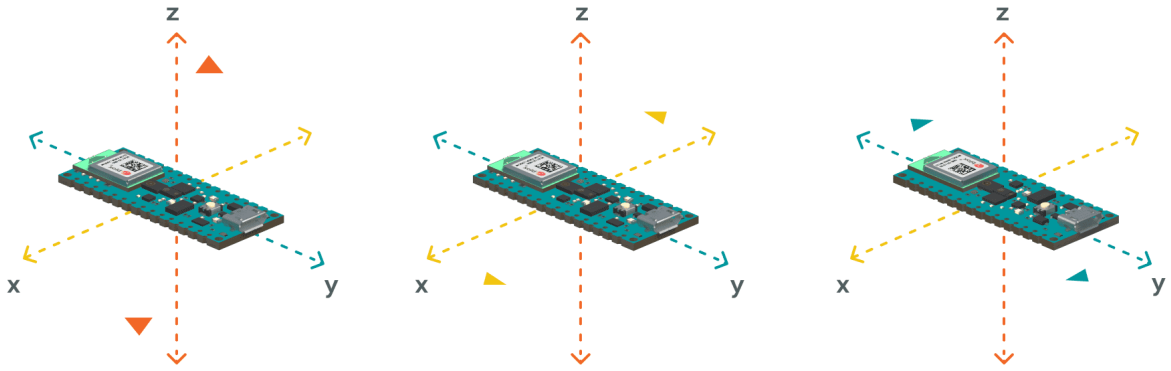
Figure 1: library manager



The Accelerometer

In this unit we will be looking at the accelerometer. An accelerometer is an electromechanical device used to measure acceleration forces. Such forces may be static, like the continuous force of gravity, or, as is the case with many mobile devices, dynamic to sense movement or vibrations.

Figure 2: the accelerometer



In this example, we will use the accelerometer as a level that will provide information about the position of the board. With this application, we will be able to read what the relative position of the board is as well as the degrees, by tilting the board up, down, left, or right.

Sketch A10.1 accelerometer

This will simply print out the raw data for the x, y, and z axes; just move the microcontroller around. Don't forget to press the RESET button.

Arduino sketch

```
#include "Arduino_BMI270_BMM150.h"

void setup()
{
  Serial.begin(9600);
  while (!Serial);
  if (!IMU.begin())
  {
    Serial.println("Failed to initialize IMU!");
    while (1);
  }
  Serial.print("Accelerometer sample rate = ");
  Serial.print(IMU.accelerationSampleRate());
  Serial.println(" Hz");
  Serial.println();
  Serial.println("Acceleration in G's");
  Serial.println("X\tY\tZ");
}

void loop()
{
  float x, y, z;
  if (IMU.accelerationAvailable())
  {
    IMU.readAcceleration(x, y, z);
    Serial.print(x);
    Serial.print('\t');
    Serial.print(y);
    Serial.print('\t');
    Serial.println(z);
  }
}
```

Notes

We can see the results if you select the `serial monitor`.

Challenge

Try the serial plotter if you can find it.

Code Explanation

`#include "Arduino_BMI270_BMM150.h"`	This is the library which has to have been already installed
`while (!Serial);`	Just waits for the serial port to open
`if (!IMU.begin())`	Checks to see if it hasn't started
`while (1);`	This creates a continuous loop until the IMU has kicked in
`if (IMU.accelerationAvailable())`	Checks to see if the data is available
`IMU.readAcceleration(x, y, z);`	Reads the data for x, y and z axis
`Serial.print('\t');`	Inserts a tab space

Figure 3: The serial monitor is the one on the right of the two icons

Figure A10.1: the Serial Monitor

Sketch A10.2 accelerometer tilting text

This will give a text indication of the movement tilting up/down/left/right and the degree of tilt.

Arduino sketch

```
#include "Arduino_BMI270_BMM150.h"

float x;
float y;
float z;
int degreesX = 0;
int degreesY = 0;

void setup()
{
  Serial.begin(9600);
  while (!Serial);
  Serial.println("Started");
  if (!IMU.begin())
  {
    Serial.println("Failed to initialize IMU!");
    while (1);
  }

  Serial.print("Accelerometer sample rate = ");
  Serial.print(IMU.accelerationSampleRate());
  Serial.println("Hz");
}

void loop()
{
  if (IMU.accelerationAvailable())
  {
    IMU.readAcceleration(x, y, z);
  }

  if (x > 0.1)
  {
```

```
    x = 100 * x;
    degreesX = map(x, 0, 97, 0, 90);
    Serial.print("Tilting up ");
    Serial.print(degreesX);
    Serial.println("  degrees");
  }

  if (x < -0.1)
  {
    x = 100 * x;
    degreesX = map(x, 0, -100, 0, 90);
    Serial.print("Tilting down ");
    Serial.print(degreesX);
    Serial.println("  degrees");
  }

  if (y > 0.1)
  {
    y = 100 * y;
    degreesY = map(y, 0, 97, 0, 90);
    Serial.print("Tilting left ");
    Serial.print(degreesY);
    Serial.println("  degrees");
  }

  if (y < -0.1)
  {
    y = 100 * y;
    degreesY = map(y, 0, -100, 0, 90);
    Serial.print("Tilting right ");
    Serial.print(degreesY);
    Serial.println("  degrees");
  }
  delay(1000);
}
```

Notes

The above is a nice, simple example. The next sketch is an example taken from the documentation.

Code Explanation

`while (1);`	This is a continuous loop the library hasn't started
`Serial.print(IMU.accelerationSampleRate());`	The sample rate is measured in Hz
`x = 100 * x;`	Multiply the x value by 100
`degreesX = map(x, 0, 97, 0, 90);`	Maps the value of x from between 0-97 to 0-90 when tilted to the left

Figure A10.2

```
Tilting up 90  degrees
Tilting up 88  degrees
Tilting up 92  degrees
Tilting up 46  degrees
Tilting right 17  degrees
Tilting down 20  degrees
Tilting down 60  degrees
Tilting left 31  degrees
Tilting down 59  degrees
Tilting left 31  degrees
Tilting down 61  degrees
Tilting left 31  degrees
Tilting down 62  degrees
Tilting left 30  degrees
Tilting down 61  degrees
Tilting left 31  degrees
```

Sketch A10.3 a more complex version

This is from another example provided; it does the same thing, roughly, although it looks a little more complex compared to the previous sketch.

Arduino sketch

```
#include "Arduino_BMI270_BMM150.h"

#define MINIMUM_TILT 5
#define WAIT_TIME 500

float x, y, z;
int angleX = 0;
int angleY = 0;
unsigned long previousMillis = 0;

void setup()
{
  Serial.begin(9600);
  while (!Serial);
  if (!IMU.begin())
  {
    Serial.println("Failed to initialise IMU!");
    while (1);
  }

  Serial.print("Accelerometer sample rate = ");
  Serial.print(IMU.accelerationSampleRate());
  Serial.println("Hz");
}

void loop()
{
  if (IMU.accelerationAvailable() && millis() - previousMillis >= WAIT_TIME)
  {
    previousMillis = millis();
    IMU.readAcceleration(x, y, z);
    angleX = atan2(x, sqrt(y * y + z * z)) * 180 / PI;
    angleY = atan2(y, sqrt(x * x + z * z)) * 180 / PI;
```

```
    if (angleX > MINIMUM_TILT)
    {
      Serial.print("Tilting up ");
      Serial.print(angleX);
      Serial.println(" degrees");
    }
    else if (angleX < -MINIMUM_TILT)
    {
      Serial.print("Tilting down ");
      Serial.print(-angleX);
      Serial.println(" degrees");
    }
    if (angleY > MINIMUM_TILT)
    {
      Serial.print("Tilting right ");
      Serial.print(angleY);
      Serial.println(" degrees");
    }
    else if (angleY < -MINIMUM_TILT)
    {
      Serial.print("Tilting left ");
      Serial.print(-angleY);
      Serial.println(" degrees");
    }
  }
}
```

Notes

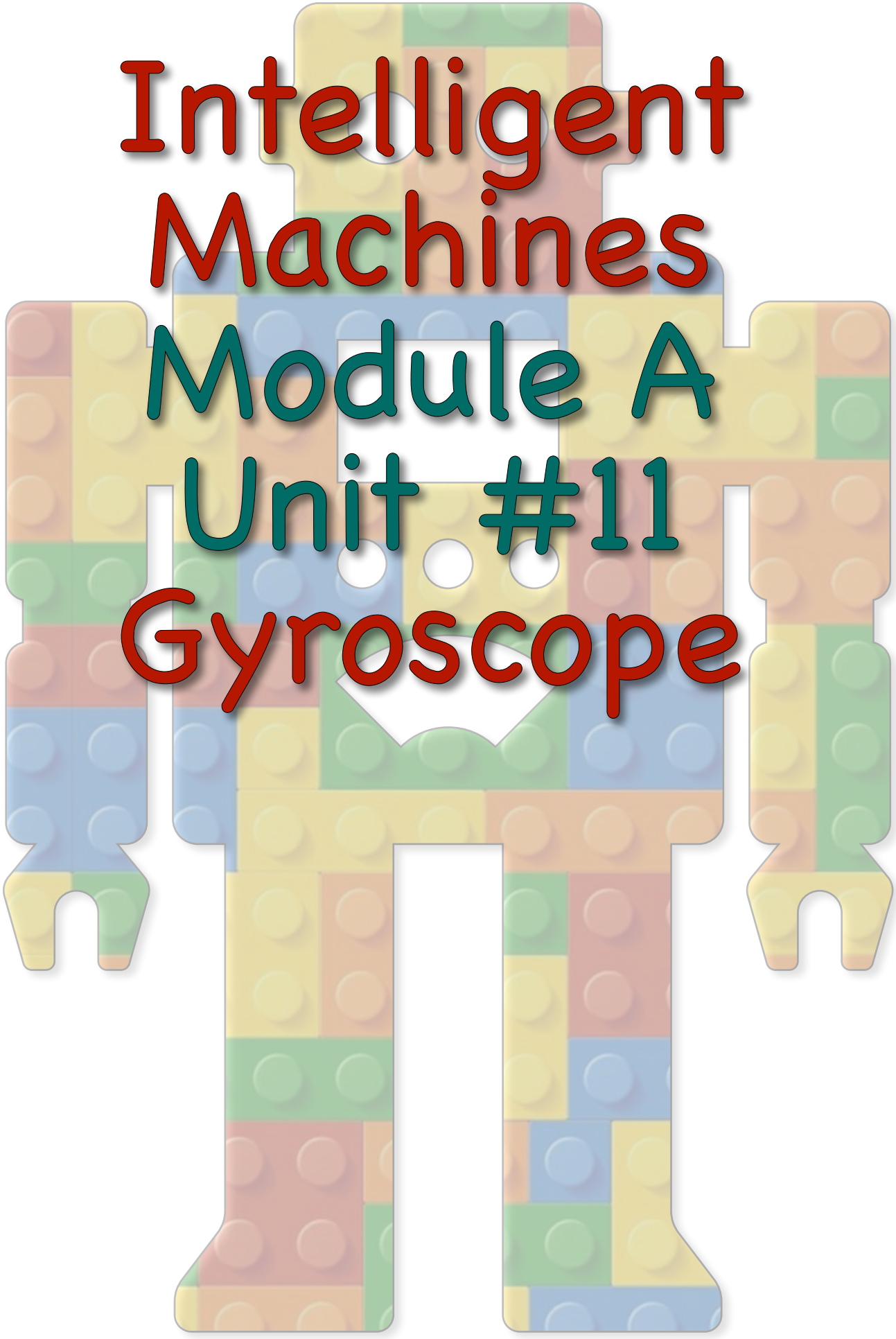
Just plough through the code. Not the place to go into detail here.

Challenge

Recommend researching atan2 and how the angle is calculated.

Code Explanation

`#define MINIMUM_TILT 5`	Threshold for tilt detection in degrees
`#define WAIT_TIME 500`	How often to run the code (in milliseconds)
`angleX = atan2(x, sqrt(y * y + z * z)) * 180 / PI;`	Calculates the tilt angles in degrees



Intelligent Machines Module A Unit #11 Gyroscope

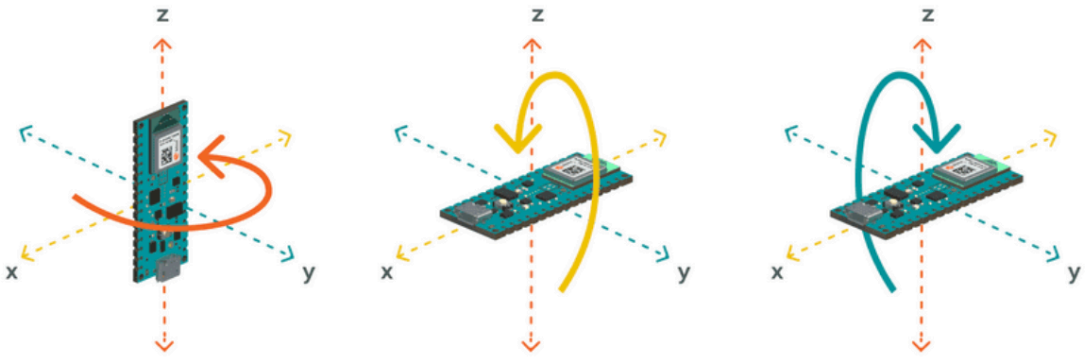
Module A Unit #11: the gyroscope

This unit will focus on the 3-axis gyroscope sensor of the IMU system on the Arduino Nano 33 BLE Rev2, to measure the direction of force on the board to emulate an object's crash. This will be achieved by utilising the values of the gyroscope's axes and later printing the return values through the Arduino IDE serial monitor.

Gyroscope sensors are also called Angular Rate Sensors or Angular Velocity Sensors. Measured in degrees per second, angular velocity is the change in the rotational angle of the object per unit of time.

In this example, we will use the gyroscope as an indicator for the direction of the force that is applied to the board. This will be achieved by swiftly rotating the board for an instant in four directions: forward, backward, to the left, and to the right. The results will be visible through the Serial Monitor.

Figure 1: the gyroscope



Sketch A11.1 gyroscope values

This prints the raw data for the gyroscope. Then we can see the data in the serial monitor. After that, we can see it in the serial plotter (see below).

Arduino sketch

```
#include <Arduino_BMI270_BMM150.h>

float x;
float y;
float z;

void setup()
{
  Serial.begin(9600);
  while (!Serial);
  if (!IMU.begin())
  {
    Serial.println("Failed to initialize IMU!");
    while (1);
  }
  Serial.print("Gyroscope sample rate = ");
  Serial.print(IMU.gyroscopeSampleRate());
  Serial.println(" Hz");
  Serial.println();
  Serial.println("Gyroscope in degrees/second");
  Serial.println("X\tY\tZ");
}

void loop()
{
  if (IMU.gyroscopeAvailable())
  {
    IMU.readGyroscope(x, y, z);
    Serial.print(x);
    Serial.print('\t');
    Serial.print(y);
    Serial.print('\t');
    Serial.println(z);
```

```
    delay(100);
  }
}
```

Notes

Very similar to the accelerometer sketch. It uses the same module. The serial plotter can be accessed by clicking on the icon next to the serial monitor.

Challenge

Have the RGB LED change colour depending on the angle of rotation.

Figure 2: The serial plotter on the left and the serial monitor on the right

Figure A11.1a: the serial monitor

Figure A11.1b: the serial plotter

Sketch A11.2 gyroscope collision

The collision is the direction of the force as a statement when it exceeds a certain value (threshold). Just rotate it about the x-axis in either direction or the y-axis in either direction.

Arduino sketch

```
#include <Arduino_BMI270_BMM150.h>

float x;
float y;
float z;
int plusThreshold = 30;
int minusThreshold = -30;

void setup()
{
  Serial.begin(9600);
  while (!Serial);
  Serial.println("Started");
  if (!IMU.begin())
  {
    Serial.println("Failed to initialize IMU!");
    while (1);
  }
  Serial.print("Gyroscope sample rate = ");
  Serial.print(IMU.gyroscopeSampleRate());
  Serial.println(" Hz");
  Serial.println();
  Serial.println("Gyroscope in degrees/second");
}

void loop()
{
  if (IMU.gyroscopeAvailable())
  {
    IMU.readGyroscope(x, y, z);
  }
```

```
  if(y > plusThreshold)
  {
    Serial.println("Collision front");
    delay(500);
  }

  if(y < minusThreshold)
  {
    Serial.println("Collision back");
    delay(500);
  }

  if(x < minusThreshold)
  {
    Serial.println("Collision right");
    delay(500);
  }

  if(x > plusThreshold)
  {
    Serial.println("Collision left");
    delay(500);
  }
}
```

Notes

Rather than data, we get a message.

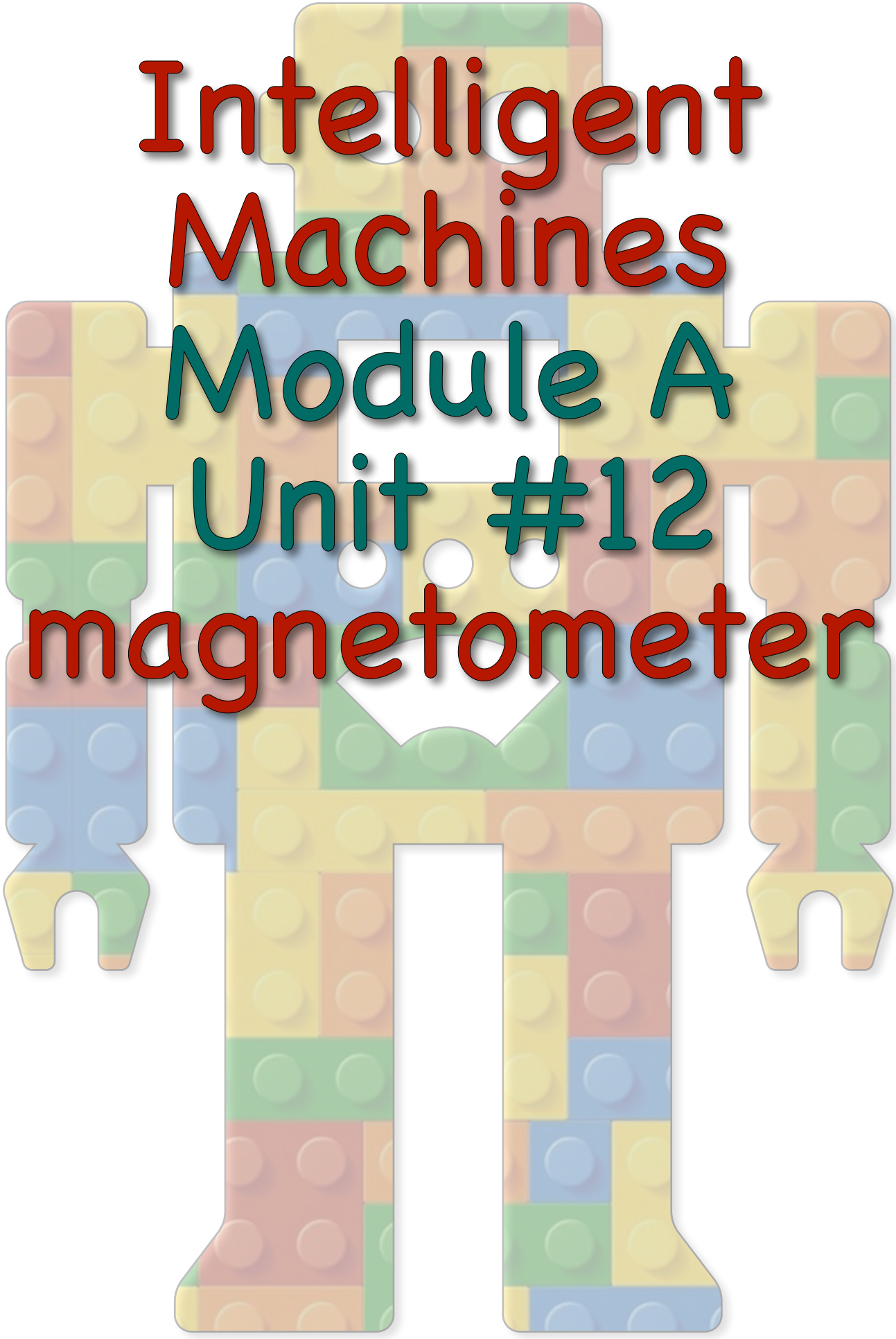
Challenge

Use the RGB LED as well.

Figure A11.2: Rotating in the x and y axis

```
Collision front
Collision front
Collision front
Collision back
Collision front
Collision back
Collision front
Collision back
Collision front
Collision left
Collision right
Collision left
Collision right
Collision left
Collision right
Collision left
Collision left
```

Intelligent Machines Module A Unit #12 magnetometer



Module A Unit #12: the Magnetometer

A magnetometer is a device that measures magnetism, that is, the direction, strength, or relative change of a magnetic field at a particular location. This works with the Earth's magnetic field as well as a strong electrical magnetic field.

It is possible to use it as a compass, but there is quite a bit of calibrating to do, and we will give it a miss in this unit. Although I would be interested to see if you use AI to figure it out if you have the raw data and the accurate angle of the Arduino. Maybe.

Figure 1: moving the Arduino over an electric cable

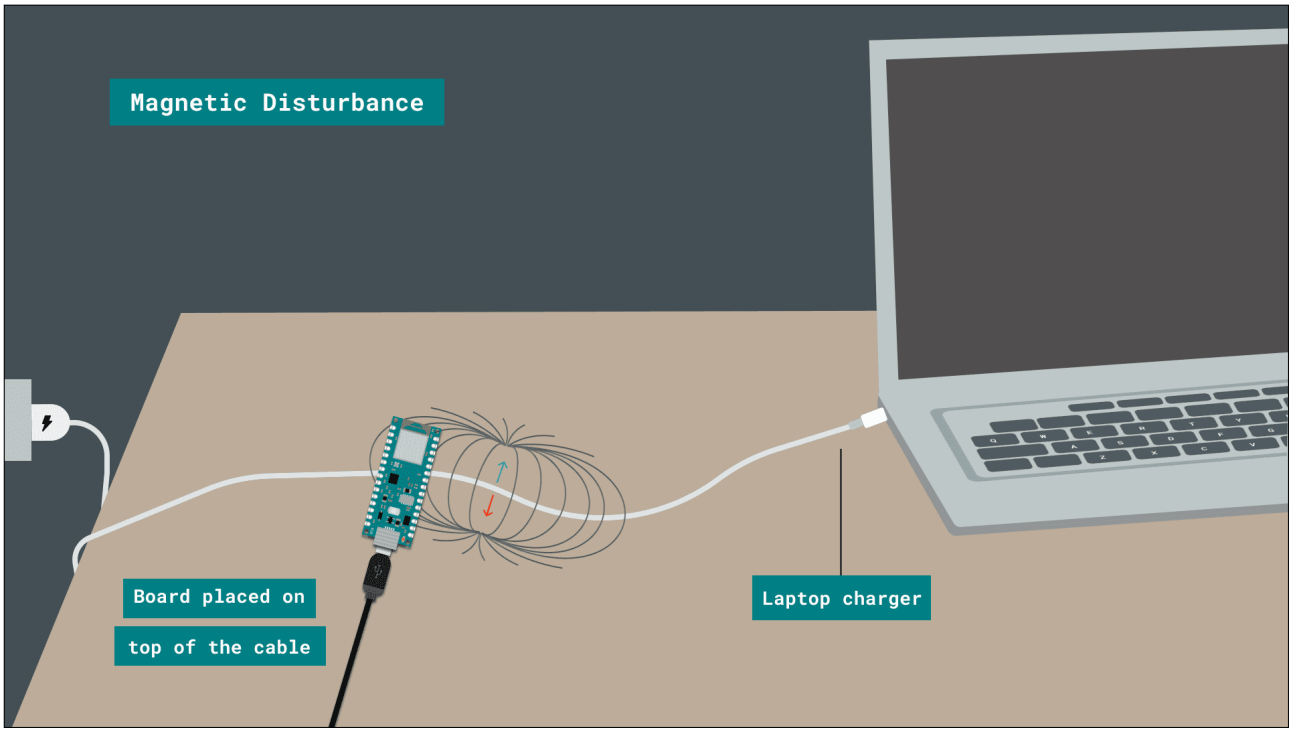
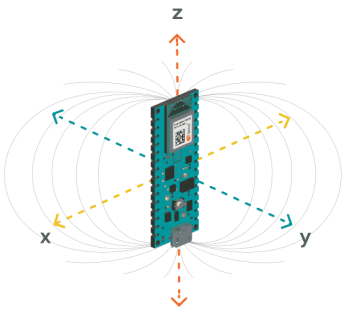


Figure 2: The x, y and z axes



Sketch A12.1 magnetometer values

Gives you the raw data.

Arduino sketch

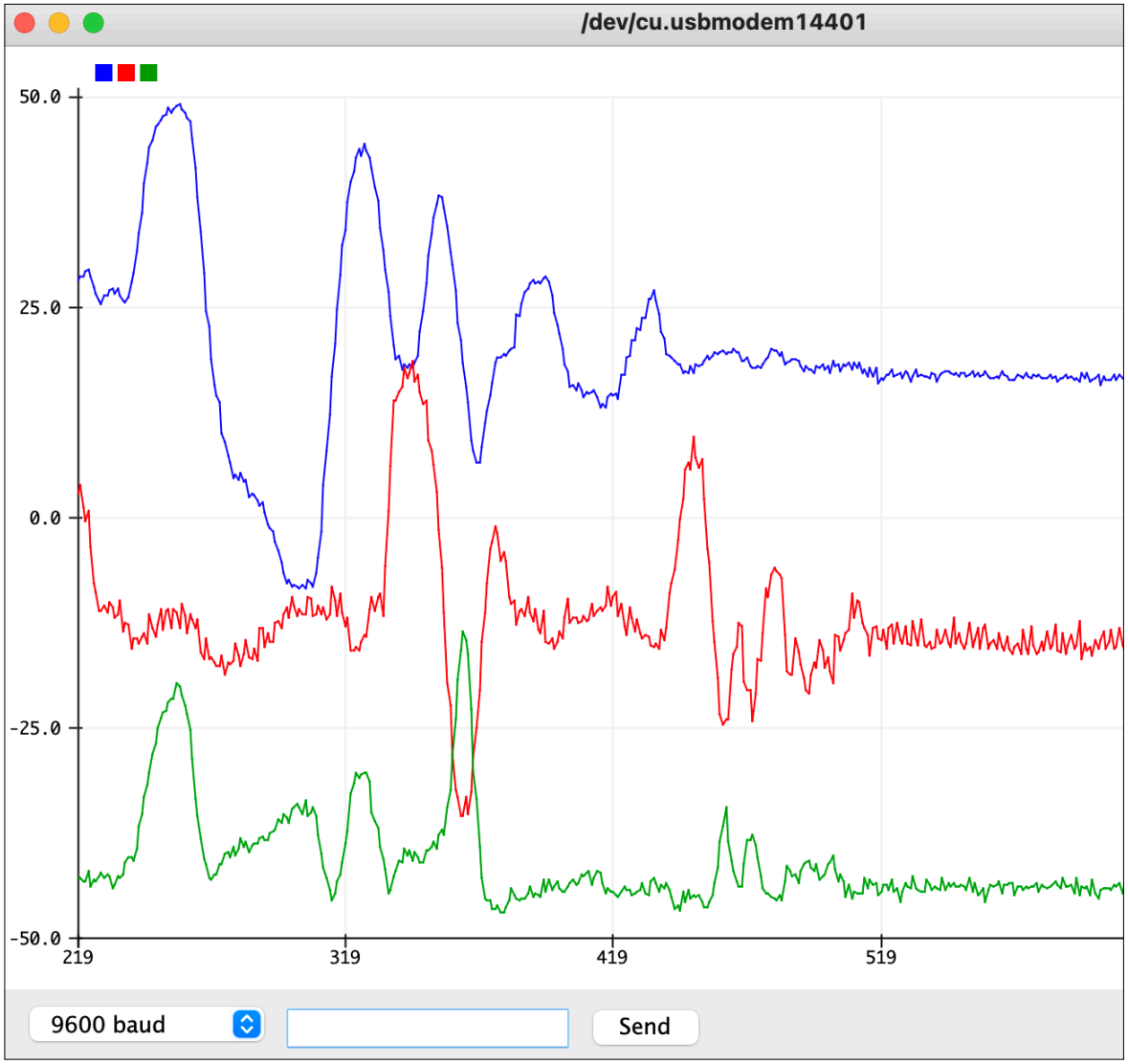
```
#include "Arduino_BMI270_BMM150.h"

float x;
float y;
float z;

void setup()
{
  Serial.begin(9600);
  while (!Serial);
  Serial.println("Started");
  if (!IMU.begin())
  {
    Serial.println("Failed to initialize IMU!");
    while (1);
  }
  Serial.print("Magnetic field sample rate = ");
  Serial.print(IMU.magneticFieldSampleRate());
  Serial.println(" Hz");
  Serial.println();
  Serial.println("Magnetic Field in uT");
  Serial.println("X\tY\tZ");
}

void loop()
{
  if (IMU.magneticFieldAvailable())
  {
    IMU.readMagneticField(x, y, z);
    Serial.print(x);
    Serial.print('\t');
    Serial.print(y);
    Serial.print('\t');
```

```
    Serial.println(z);
  }
}
```

Notes

Very similar to the `accelerometer` and the `gyroscope`. Also, we can see the data in the serial monitor and plotter.

Figure A12.1



Sketch A12.2 magnetometer detection

Using the built-in LED, this is not as effective as I would like, but it is still a demonstration. Notice that it is a slimmed-down version. I used a fridge magnet to get a better effect.

Arduino sketch

```
#include "Arduino_BMI270_BMM150.h"

float x;
float y;
float z;
float ledValue;

void setup()
{
  IMU.begin();
}

void loop()
{
  IMU.readMagneticField(x, y, z);
  ledValue = map(x, 0, 100, 0, 256);
  analogWrite(LED_BUILTIN, ledValue);
  delay(500);
}
```

Notes

You get something from the LED, but it's a bit underwhelming.

Challenges

1. See if you can improve it.
2. Can you get it to work as a compass?