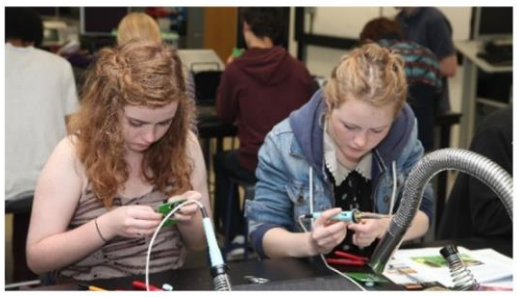
Spark their Imagination 2024 Student Guide

This guide will introduce you to the Arduino microcontroller platform, the Grove Beginner Kit for Arduino and basic programming skills. After completing this guide, you should have the confidence and inspiration to undertake your own Electronics projects. Let's start!

Contents





Introduction – What is Arduino?

The Arduino is a microcontroller development platform aimed at people who want to use programmable-electronic hardware without needing to delve deeply into how a microcontroller operates. Figure 1 shows an original Arduino Uno board viewed from the top. The microcontroller chip itself is the large black rectangle in the lower-left part of the board. It is a Microchip ATmega328P and is the brains of the Arduino. Chips like this are also called integrated circuits, or ICs, and can contain anywhere from a few thousand to billions of transistors. The rest of the Arduino board includes the hardware necessary to power and program the board through USB and interface to the inputs and outputs of the microcontroller (through the row of connectors on the side of the board).

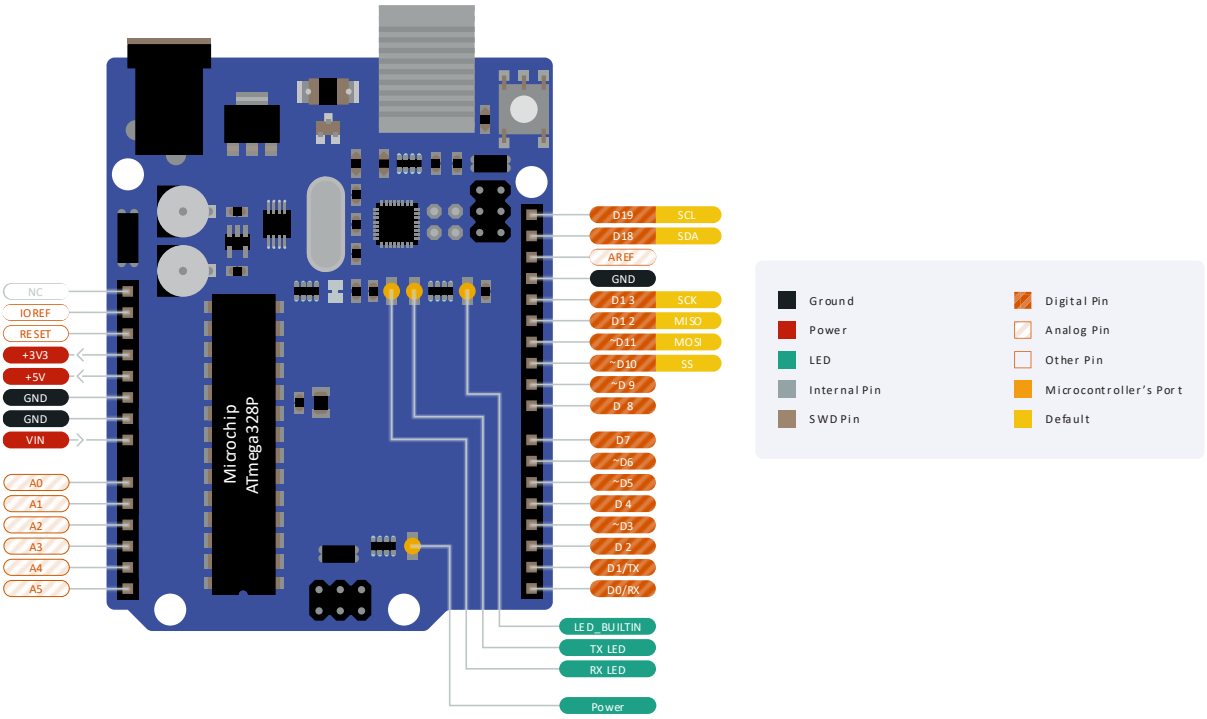


Figure 1. An Arduino UNO and its pins. The pin diagram has been simplified for the purposes of this guide. The full version is available at bit.ly/3jg6PM2.

Apart from allowing an easy interface with external components, these connectors are designed so that add-on printed circuit boards (PCBs) - often called shields - can be added. You can buy shields for things like motor control, GPS, mobile telephony and more to allow projects to be constructed quickly and with ease.

The Grove Beginner kit for Arduino

The board you will be using today is shown in Fig. 2 and looks slightly different than the original Arduino Uno. It integrates the main microcontroller board called *Seeduino Lotus* sitting in the middle, to several preconnected sensors and transducers on the side. The Seeduino Lotus is essentially a modified Arduino Uno with a slightly different hardware interface but works exactly like an Arduino board would. The nice thing about the kit is that it is self-contained, and you don't need to separate any of the sensor modules for them to work. You can connect to the sensors via the four-pin connectors on the board (the white sockets) or simply program them directly through the preconnected tracks on the printed circuit board or PCB.

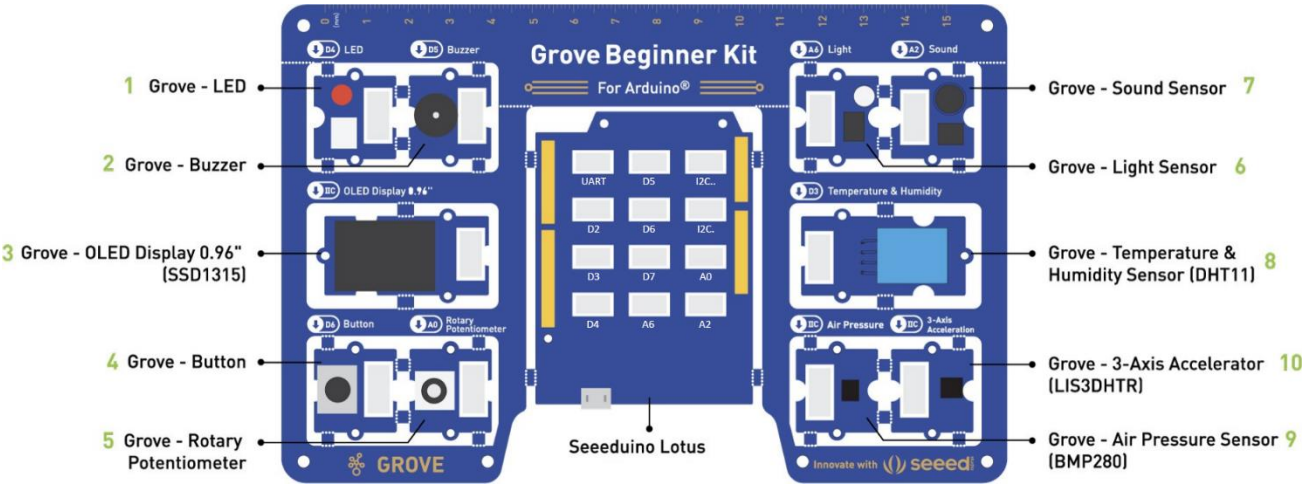


Figure 2. The Grove beginner kit for Arduino comprising of the Seeeduino Lotus and 10 peripherals. (Adapted from bit.ly/2OSLrkZ)

Figure 3 shows an example of the PCB tracks that connect a sensor to the main board, so no cables are required unless explicitly stated otherwise. Note however that once a sensor is broken out of the big PCB (which you can do thanks to the small perforations in the board), the provided Grove cables can be used to connect them back to the Seeeduino Lotus. **We recommend leaving the peripherals attached for this practical!**

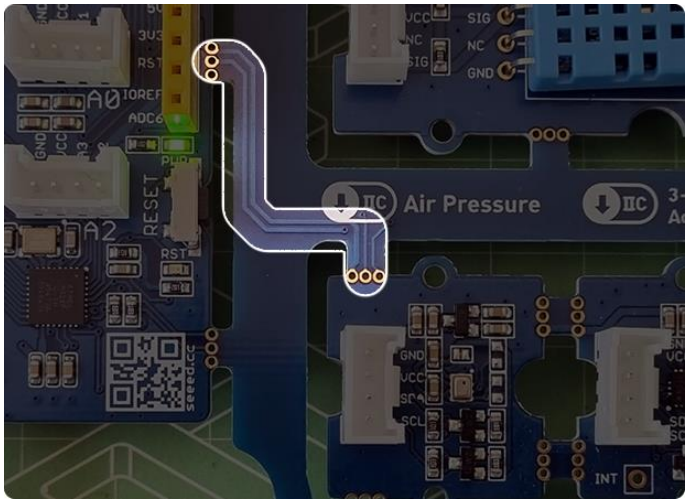


Figure 3. The PCB tracks, that go through the break-out points of the PCB and connect the peripherals to the Arduino. Thanks to these, no cables are needed to use the peripherals.

Interfacing with the Arduino

There are two main types of input/output pins (I/O pins) on the Arduino and Seeduino Lotus boards:

- **Digital I/O pins**:
  - Can have a value of either High (5V) or Low (0V).
  - Connect to digital peripherals such as buttons, switches, displays and more.
  - Are labelled 0 – 13 in the Arduino IDE and on the on the left connector of the Seeduino Lotus board.
- **Analogue I/O pins:**
  - Can work with continuous values between (and including) 5V and 0V.

- Connect to analogue peripherals such as potentiometers, light sensors, microphones and more.
- Are labelled *A0, A1, etc*. In the Arduino IDE and as "ANALOG IN" at the top of the right connector of the Seeduino Lotus board.

Another important way of interacting with the Arduino is **serial communication** which allows a connection to more complex peripherals such as computer, Bluetooth, accelerometers and more. We will investigate serial communication in Exercises 3 and 4.

The Arduino IDE and Sketch Template

Note: Please complete the steps to install the Arduino IDE and other components detailed in the "Software Installation" document before you start. If you are unsure whether Arduino is already installed, please ask for help.

Let's start by looking at the Arduino IDE and the empty "sketch" template where you will write your code.

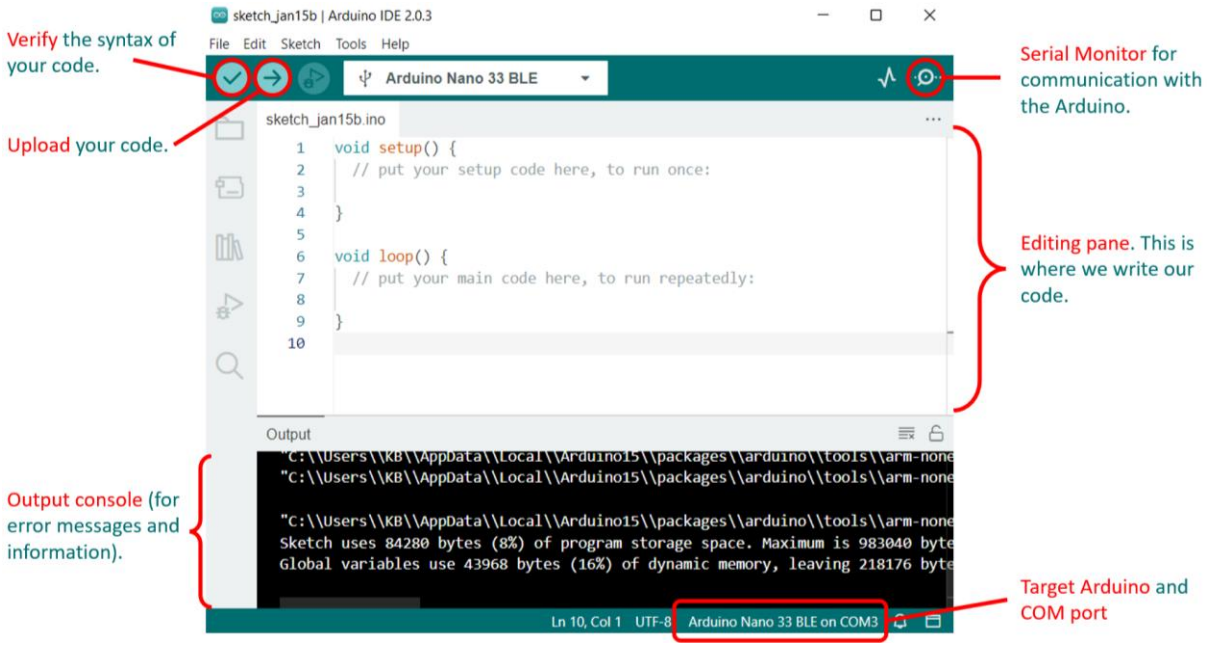


Figure 4. The Arduino IDE showing an empty sketch.

When you open the Arduino IDE for the first time, you should see the code for an empty program with only the `setup()` and `loop()` functions defined. This is where you will fill in your code and is referred to as ***sketch***. A sketch is simply a text file that describes your program in the Arduino language. More specifically:

- The `setup()` function describes what the Arduino needs to do once, like initialise a sensor, or start a serial communication protocol to communicate with a computer or peripheral.
- The `loop()` function describes the things you want the Arduino to do repeatedly, in a loop. Things like measure temperature, output a sound on the buzzer or display something on a screen. This is where most of your code will go and can be considered the main program of the Arduino.

Saving your Sketch

Once you start writing code, it is recommended that you save your sketch to a convenient location using the **File > Save As…** menu. The default location to save your sketches is usually "/Documents/Arduino/" and is fine for our purposes.

Debugging and Errors

You can use the *verify* button in the Arduino IDE (see Fig. 4) to check if you have written your code correctly and the *upload* button to upload the finished code to your board. If there are *errors* in your code, the IDE will warn you and stop you from uploading it. You will instead need to work out the cause(s) of the errors and "debug" your code. Some common causes of errors/bugs are: spelling errors, missing semi-colon(s) (;) at the end of code statements, undefined variables, and many more… debugging can take time and patience is key!

Note: errors will be printed at the bottom of your Arduino IDE sketch environment, usually in red or orange to highlight that there is an issue. However, the IDE does not always point you to the exact line of the code where the error exists, and you may have to look through your code several times to find the source of the error.

Exercise 1 – Blinking an LED

Blinking an LED is a good "Hello World" program for Arduino. It is a simple program that blinks a light on and off and will introduce you to the development environment. It is also a good test to check your connection to the hardware, as well as the hardware itself.

Step 1

In your empty sketch, *before* the `setup()` function, define the name and pin number of the LED you will use (in our case, the Grove kits' red LED is connected to pin D4, and is written on the top left of the Grove kit board). Therefore, we will define our LED pin to be digital pin number 4:

```
const int ledPin = 4;  //Define the name and pin number of the LED to blink
```

Step 2

In the `setup()` function (the code that runs only once) we need to declare whether the pin in question should be an Input or an Output. In our case, we are going to drive the LED from the Grove kit, which means our pin should be defined as an Output. We can do this with a function called `pinMode()` as shown below:

```
void setup() {
  pinMode(ledPin, OUTPUT);  // Initialize the ledPin as an Output.
}
```

Step 3

Next, in the `loop()` function we will define what we want our LED to do. Remember, this function will run through the code we write in it repeatably, in a loop, forever. Since we want to blink the LED we need to decide when it should turn on and off and for how long it should stay in each state. We can implement this using a function called `digitalWrite()` which either turns the LED ON by outputting a High voltage to our LED pin (i.e. 5V or a 1 state) or a Low voltage (i.e. 0V or 0 state). To keep the LED in a state for some period of time we use the `delay()` function. This function tells the program to wait for a certain amount of milliseconds before moving on to the next line of code. These two functions result in the following code:

```
void loop() {
  digitalWrite(ledPin, HIGH);  // Turn the LED on (High, 1, or 5V).
  delay(250);                  // Wait for 250 milliseconds.
  digitalWrite(ledPin, LOW);   // Turn the LED off (Low, 0, or 0V).
  delay(250);                  // Wait for 250 milliseconds.
}
```

Step 4

Once you have completed the above code click the verify button and sort through any errors you encounter. Ask for help if you need it. Once you are clear of errors, you're ready to upload your code. If you see your LED blink, congratulations! You've written your first working Arduino code.

Exercise 2 – Buzzing the Buzzer

In this exercise you will learn how to use the buzzer on the Grove kit. It sits next to the red LED and our goal is to play a tone on it that stays on for one second and then off for one second.

Step 1

Launch a new sketch (save your old one and open a new one from **File > New**). Then, before the `setup()` function, define the name and pin number the buzzer is connected to (D5 on the Grove kit) as well as the frequency of the tone we wish to play:

```
const int buzzerPin = 5;  //Define the name and pin number of the buzzer
const int toneFrequency = 200;  //Frequency of tone in Hz
```

Step 2

In the `setup()` function declare whether the buzzer is an input or an output. Since we want to play a tone on the buzzer, we need to define the pin as an output:

```
void setup() {
  pinMode(buzzerPin, OUTPUT);  // Initialize the buzzerPin as an Output.
}
```

Step 3

Next, in the `loop()` function let's define the behaviour we want. We want to play a tone of 200 Hz for 1s, then turn the buzzer off for 1s, and repeat. To play a tone on a buzzer the Arduino language provides two functions: `tone()` and `noTone()`. They both need the pin number that the buzzer is connected to as arguments, but the `tone()` function also needs the frequency of the tone to be played. Our code will thus look as follows:

```
void loop() {
  tone(buzzerPin, toneFrequency);  // Play a tone of 200 Hz on the buzzer
  delay(1000);                     // Wait for 1000 milliseconds or 1s.
  noTone(buzzerPin);               // Turn the buzzer off.
  delay(1000);                     // Wait for 1000 milliseconds or 1s.
}
```

Step 4

Once you have completed the above code click the verify button and sort through any errors you encounter. Call for help if you need it and finally upload your code.

Try changing the frequency of the tone, and the duration it is on and off to explore the different sounds you can make.

Tutorial - Serial Communication and Talking to your Computer

So far, we have used the digital and analogue I/O's to interface between our microcontroller and sensors on the Grove kit. But how do we interact with more complex devices such as your computer or more complex sensors like a digital accelerometer? The answer is *serial communication*. Essentially, serial communication works by converting information to a stream of bits, which are then sent between two devices over one or more wires.

The Serial family of functions in Arduino allow us to use the UART communication protocol to communicate with a computer. To use these Serial functions, we need to begin a serial communication protocol by using the function `Serial.begin(9600)`. The value `9600` is called the 'baud rate' of the connection and defines how fast the data is to be sent. When using an Arduino device you will encounter the `begin()` function often to initialise different peripherals and communication protocols.

After beginning the serial protocol, you can use functions from the Serial family, such as `Serial.print()` – which sends the text in the parentheses to the computer, and `Serial.println()` – which does the same, but also starts a new line. Let's test this functionality. Type the below code into a new sketch and upload it. Then, in the Arduino IDE, go to **Tools > Serial Monitor** (or click the looking glass icon in the top-right). This will open a new window showing the message we sent from our board (see Fig. 5).

Try sending different messages using both, `Serial.print()` and `Serial.println()`, to get a feeling of how they work.

```
void setup() {
  Serial.begin(9600);               // Begin the Serial communication.
  Serial.println("Hello, World!");  // Send a message to the computer.
  //You can add more print statements here
}

void loop() {} // For now, our loop() function does nothing.
```

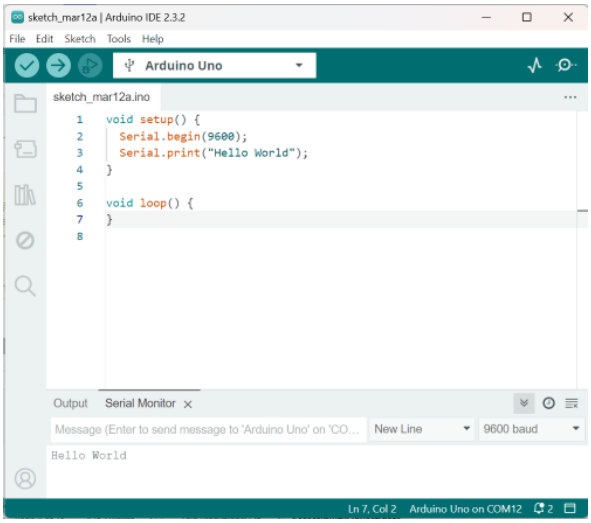


Figure 5. A preview of the Serial Monitor tool in Arduino IDE 2.3.2, showing the "Hello, World!" message that was sent over USB from the Arduino.

Exercise 3 – The Accelerometer

In this exercise you will learn to how to read data from a 3-axis accelerometer and plot it as a graph on the Arduino Serial Plotter. An accelerometer is a device that measures the rate of change of velocity of a body. For example, an accelerometer at rest on the surface of Earth will measure the acceleration due to Earth's gravity as 1 g (gravitational force equivalent) or approximately 9.81 m/s^2. Using the accelerometer on the Grove kit, we can measure acceleration in three axes. Let's see how to do that:

Step 1

First, before the `setup()` function, include the UKESF library (which will handle some of the accelerometer functions for us) and define an Accelerometer variable called "myAccelerometer":

```
#include <UkesfSixthFormers.h> // Include the UKESF library of functions

Accelerometer myAccelerometer;  // Create an instance of the accelerometer
```

Step 2

Next, begin (or initialise) Serial communication and the Accelerometer:

```
void setup() {
  Serial.begin(9600);     // Begin the Serial communication.
  myAccelerometer.begin();  // Begin the Accelerometer.
}
```

Step 3

Then in the `loop()` function, read the accelerometer values and print them to your serial monitor. Note that we need to use floating point variables or `float` because the acceleration is given as a decimal number.

```
void loop() {
  float x = myAccelerometer.readX();  // Read x-axis acceleration
  float y = myAccelerometer.readY();  // Read y-axis acceleration
  float z = myAccelerometer.readZ();  // Read z-axis acceleration
  Serial.print(x);
  Serial.print(" ");
  Serial.print(y);
  Serial.print(" ");
  Serial.println(z);
  delay(10); // Delay the program by 10ms for stability when looping
}
```

Step 4

Finally, there are two ways you can view the data you read from the accelerometer. The first is using the Serial monitor like you did before. The second is the *Serial Plotter*. You can access the plotter from **Tools > Serial Plotter.** This is a very useful tool that plots the data for us in graph form. Try tilting and turning your board to identify each of the three axes of the accelerometer.

Exercise 4 – Combining it all

In this exercise you will combine what you have learned in the previous three exercises into one program that will implement a tilt sensor with a warning light and sound. Specifically, the program should:

- Read data from the accelerometer.
- Calculate the pitch and roll (angles) of your board in degrees, based on that data.
- If the roll (or pitch) exceeds 20 degrees, light the LED and sound the buzzer.

This time, the code includes only the comments of what the program should include, and your task is to fill them in**. If you are unsure about any of the syntax or pin numbers, look back to exercises 1-3 for help.**

Step 1

To start off we again need to include the UKESF library to access the accelerometer and define the pin numbers and variables the program will use later. Fill in the missing lines of code before each comment below:

```
// Include the UKESF library of functions

// Define the name and pin number of the LED
// Define the name and pin number of the buzzer
// Frequency of tone in Hz
// Create an instance of the accelerometer
```

Step 2

Next, initialise and begin the LED and buzzer, Serial communication and accelerometer.

```
void setup() {
  // Initialize the ledPin as an Output.
  // Initialize the buzzerPin as an Output.
  // Begin the Serial communication.
  // Begin the Accelerometer.
}
```

Step 3

Finally, we implement the main program in the `loop()` function. In it, we want to do three things:

1) Read the accelerometer data
2) Calculate the roll and pitch
3) Light the LED and sound the buzzer IF the roll is > 20 degrees.

Let's look at each of these tasks in turn.

1) Reading the accelerometer data.

Look back to exercise 3 for the code used to read data from the accelerometer and printing it out to the Serial monitor. Although the program does not require us to print to Serial, it is a good idea to do so for debugging purposes. We can then use the Serial monitor after uploading our code to sanity check and make sure we are getting data from the accelerometer as we would expect.

```
void loop() {
  // Read x-axis acceleration
  // Read y-axis acceleration
  // Read z-axis acceleration

  // Print the value to Serial for debugging help
}
```

2) Calculating roll and pitch from accelerometer data.

Pitch, roll and yaw are rotational forces (illustrated in Figure 6) and deriving them as angles from accelerometer data is not a trivial task. However, to help us with the calculation search for “pitch and roll Arduino” or “pitch and roll accelerometer” online and you will find many excellent resources explaining how a 3-axis accelerometer can be used to derive pitch and roll. Two good explanations are given by:

- DF Robot tutorial with quick derivation: https://wiki.dfrobot.com/How_to_Use_a_Three-Axis_Accelerometer_for_Tilt_Sensing
- Application note from NXP with more rigorous derivation: https://www.nxp.com/files-static/sensors/doc/app_note/AN3461.pdf

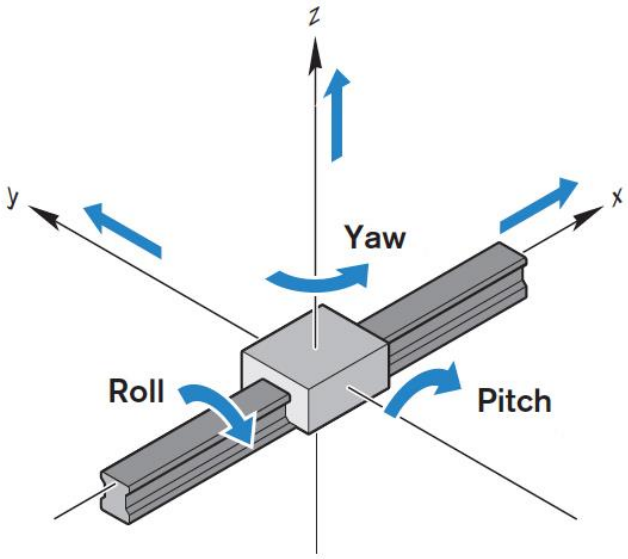


Figure 6. An illustration of pitch, roll and yaw of a linear system, in this case a rigid beam.

While the complete derivation of the roll and pitch angles are too long to include in this document, we suggest having a quick read through the provided links to get an understanding of what's going on. The final equations are given below and require the use of pi, which in Arduino is written as "M_PI".

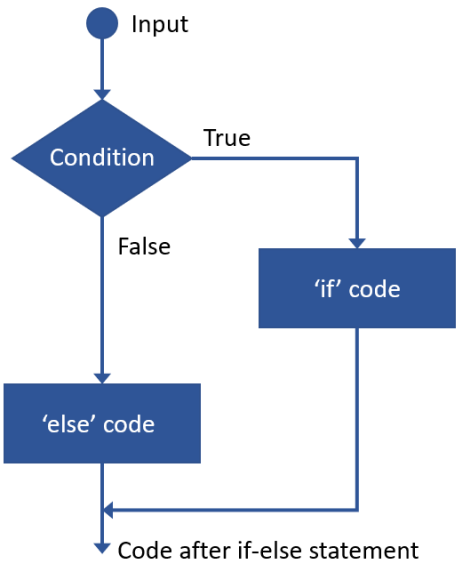
```
void loop() {
  // Read x-axis acceleration
  // Read y-axis acceleration
  // Read z-axis acceleration

  // Print the value to Serial for debugging help

  //Calculate Roll and Pitch based on accelerometer data
  //See provided links for derivation.
  float roll  = (atan2(-y, z)*180.0)/M_PI;
  float pitch = (atan2(x, sqrt(y*y + z*z))*180.0)/M_PI;
}
```

3) Light the LED and sound the buzzer if the roll is > 20 degrees.

Finally, we want to turn the LED on and sound the buzzer, but only if the roll or pitch angle (you can choose!) is > 20 degrees in either direction (we must therefore use the absolute value function `abs()`). To do this, we must include a conditional if-else statement in our code, which implements the logical sequence illustrated by the flow diagram in Figure 7. The if-else statement tests an input variable against a condition (i.e., is the angle > 20 degrees?) and takes the True or False path depending on the result. When a condition is True, only the code inside the curly brackets enclosing the 'if' statement is executed and when it is False, the 'else' statement is executed.




```
if(condition is True){
 // Execute this code (condition is True)
}
else{
 // Execute this code (condition is False)
}
```

Note! The `condition` in the if statement can be any logical test, for example:

To check for negative temperatures, you could write:

```
if(temp < 0)
```

To check for an angle of 45 degrees, you could write:

```
if(angle == 45)
```

Figure 7. Flow diagram of an if-else statement in Arduino and the corresponding code. Depending on if the condition is True or False different code gets executed and the rest is skipped.

Now that we understand if-else statements here is the complete code template for our program. Try to complete it based on your previous work and ask for help if you need it.

```
void loop() {
  // Read x-axis acceleration
  // Read y-axis acceleration
  // Read z-axis acceleration

  // Print the value to Serial for debugging help

  //Calculate Roll and Pitch based on accelerometer data
  //See provided links for derivation.
  float roll  = (atan2(-y, z)*180.0)/M_PI;
  float pitch = (atan2(x, sqrt(y*y + z*z))*180.0)/M_PI;

  //if-else statement to check if the roll is > 20 deg
  //Note the use of the absolute value abs()
  if(abs(roll) > 20){
    // Turn the LED on (High, 1, or 5V).
    // Play a tone of 200 Hz on the buzzer
  }
  else{
    // Turn the LED off (Low, 0, or 0V).
    // Turn the buzzer off.
    }

  delay(10); // Delay the program by 10ms for stability when looping
}
```

Open-Ended Design Challenge

Well done on making it through the introduction to the Grove kit and Arduino IDE! You now have enough experience to explore your own designs using the board.

This is where your personal creativity and curiosity come in to play and we will leave it up to you to decide what you want to do, but here are some tips to get you started.

A good way to find inspiration is to search for ideas online. There are thousands of previous Arduino projects on the web and often you can find something related to what you want to achieve. For example, if you would like to play a song using the buzzer, search for "Arduino play tune on buzzer" and you will find many examples. The tricky bit can be to discern which example is best suited to your needs, or how exactly someone else's code works. Often this is a process of trial and error. You could also combine bits and pieces of different code to make your own program. To get started with this process we have collected some links and ideas below. You could for example:

- Play tunes using the buzzer (e.g. happy birthday). See this Arduino tutorial.

- Display custom text on the OLED screen based on sensor input. For example:
    - Display text on the screen for different orientations of the board using the accelerometer, expanding on the previous tilt sensor exercise.

- Use the microphone to pick up sound and inspect the audio waveform using the Serial Plotter. Set a threshold to turn on the LED if the sound level is above the threshold.

- Use the push button to increment and display the number of times the button has been pressed on the OLED screen. Here you could search online for "*Arduino number of button presses display*" for tips on how to achieve this.

- Come up with your own project, create something new and make your ideas come to life. It is immensely satisfying and the best part of Electronics. Have fun!

Final Words and Further Resources

We hope you had fun getting to know the Grove Beginner kit in this workshop! Creativity and imagination lie at the heart of Electronics and there are many more resources to keep you going on your own. Searching online for project ideas and inspiration is the best way to find out how to do things and there is also a large community of forums and hubs where you can ask questions and get help if you need it. Here is a short list of sites worth checking out:

- The Arduino homepage.
- The Arduino project hub.
- Seeedstudio's guide for the Grove kit: Grove-Beginner-Kit-For-ArduinoPDF.
- Seeedstudio's article with a collection of tutorials: Arduino Project Roundup.
- UKESF videos for the Grove kit: Introduction and Level meter project.
- Arduino projects on Instructables: https://www.instructables.com/Arduino-Projects/
- Search for "*your project idea + Arduino*" and you'll find tons of resources!

Thank you for participating in this workshop and good luck with your future Electronics projects!

About the UK Electronics Skills Foundation

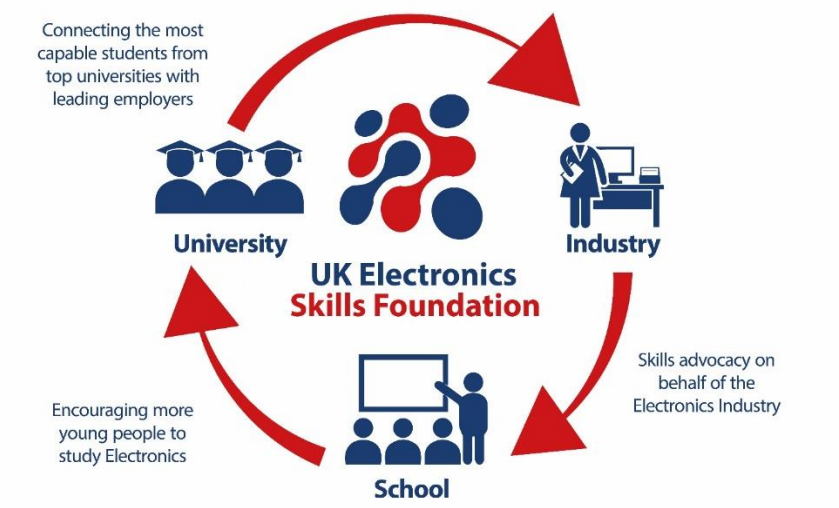
The purpose of the UKESF is to tackle the skills shortage in a coherent way. Our aim is to:

- With our partners, provide opportunities for them to develop their **interest** in Electronics and engineering, through to university study and/or apprenticeship.
- At university, ensure that undergraduates are encouraged to pursue careers in the Electronics sector, and they are supported in their professional **development** so when they graduate, they are equipped with work-ready skills and experience.
- After graduation from university, we will help create a community of Electronics engineers to secure the future pipeline. We will **build relationships** and act as the representative voice for the sector on skills.

We are an independent charitable foundation, established in 2010, at the nexus of an extensive network of partners and collaborators, including 27 universities and around 75 companies. On behalf of the electronics sector, we will build relationships, provide thought leadership and act as the representative voice on skills related matters.

Registered charity number: SC043940

www.ukesf.org



"Moving beyond talk about the skills shortage to take positive action is what the UKESF is all about."

Stew Edmondson, CEO, UKESF