UKESF Arduino Guide

The following guide will introduce you to the [Arduino](https://www.arduino.cc/en/Guide/Introduction) microcontroller platform and the [Grove Beginner Kit for Arduino](https://www.seeedstudio.com/Grove-Beginner-Kit-for-Arduino-p-2895.html) and teach you basic programming skills. After completing this guide, you should have the confidence and inspiration to undertake electronics projects. Let’s dive in!

This guide attempts to cover the basics of Arduino. Where relevant, links with further information are provided for the curious readers.

Contents

[Introduction – What is Arduino 4](#_Toc66984969)

[Arduino Uno 4](#_Toc66984970)

[Grove Beginner Kit for Arduino 5](#_Toc66984971)

[Prerequisites 6](#_Toc66984972)

[The Arduino IDE 6](#_Toc66984973)

[The Seeeduino Lotus Board 6](#_Toc66984974)

[Digital Outputs 7](#_Toc66984975)

[Tutorial – Blinking an LED 7](#_Toc66984976)

[Exercise – Adding a Buzzer 9](#_Toc66984977)

[Key Points 9](#_Toc66984978)

[Digital Inputs 9](#_Toc66984979)

[Tutorial – Reading Switches 10](#_Toc66984980)

[Exercise – Inverting the Switch State 11](#_Toc66984981)

[Key Points 11](#_Toc66984982)

[Conditional Logic 11](#_Toc66984983)

[Tutorial – LED Button Trigger 11](#_Toc66984984)

[Exercise – LED Button Trigger with Alarm 13](#_Toc66984985)

[Key Points 13](#_Toc66984986)

[Analogue Outputs 13](#_Toc66984987)

[Tutorial – Making an LED Dim 15](#_Toc66984988)

[Exercise – Soft Blinking LED 16](#_Toc66984989)

[Key Points 17](#_Toc66984990)

[Analog Inputs 17](#_Toc66984991)

[Tutorial – LED with Controllable Intensity. 17](#_Toc66984992)

[Exercise – Smart Lamp 19](#_Toc66984993)

[Exercise – Sound-Reactive Lamp **Error! Bookmark not defined.**](#_Toc66984994)

[Key Points 19](#_Toc66984995)

[Serial Communication 19](#_Toc66984996)

[Tutorial – Sending a Message to the Computer 19](#_Toc66984997)

[Tutorial – Reading and Plotting Acceleration Data 20](#_Toc66984998)

[Exercise – Reading and Plotting Meteorological Data 20](#_Toc66984999)

[Exercise – Displaying Text on the OLED Screen 20](#_Toc66985000)

[Key Points 20](#_Toc66985001)

[Final Project – Weather Station 20](#_Toc66985002)

[Next Steps 20](#_Toc66985003)

[Temporary Section – Please Ignore This 21](#_Toc66985004)

# Introduction – What is Arduino

The Arduino platform is a microcontroller development system that is aimed at people who want to use programmable-electronic hardware without needing to delve deeply into how a microcontroller operates. This platform is useful as an introduction to programmable systems for young engineers who are yet to learn more about what happens inside microcontrollers.

## Arduino Uno

Figure 1 shows the hardware for an Arduino Uno. The microcontroller (a [Microchip ATmega328P](https://www.microchip.com/wwwproducts/en/ATmega328P)) is the large integrated circuit (IC) in the lower-left section of the board. The Arduino includes all the software and hardware needed to be powered and programmed through USB, and all its inputs and outputs are brought out on connectors (analogue and digital pins) running on along the side edges.

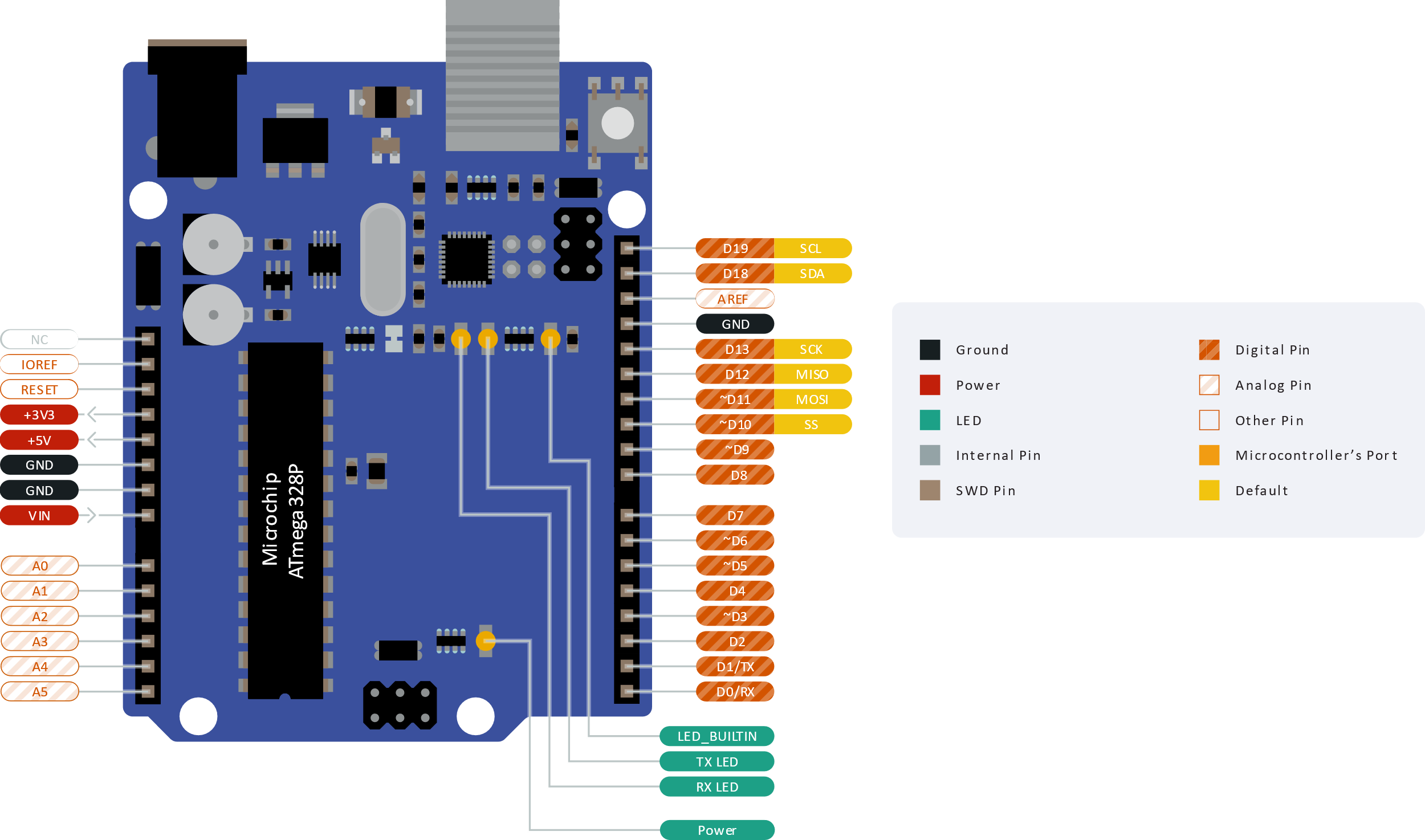


Figure . An Arduino UNO and its pins. The pin diagram has been simplified for the purposes of this guide. The full version is available at [bit.ly/3jg6PM2](https://bit.ly/3jg6PM2).

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, and mobile telephony to allow projects to be constructed quickly without knowing much electronics.

There are five pin types that are worth remembering and will be used throughout this guide (refer to Figure 1):

* **Digital inputs** connect to digital output peripherals such as buttons and switches. These are the pins labelled as *D0-D19*.
* **Digital outputs** connect to digital input peripherals such as LEDs, bar graph displays and RGB LEDs. These are the pins labelled as *D0-D19*.
* **Analogue inputs** connect to analogue output peripherals such as potentiometers, light intensity sensors and microphones. These are the pins labelled as *A0-A5*.
* **Analogue outputs** connect to analogue input peripherals such as dimmable LEDs and buzzers. These are the pins labelled as *~D3*, *~D5*, *~D6*, *~D10*, *~D11*. The “analogue” signals from these outputs might not be what you expect, but more on that later.
* **Serial communication** allows the connection to more complex peripherals such as computer, Bluetooth communication peripherals, accelerometers, displays and more! The two serial communication protocols that will be used later in this guide are [*I2C*](https://en.wikipedia.org/wiki/I%C2%B2C) and [*UART*](https://en.wikipedia.org/wiki/Universal_asynchronous_receiver-transmitter). Their connection pins are *SDA, SCL* and *RX*, *TX* respectively.

Inside the microcontroller on the Arduino Uno, there are several complex peripherals that the Arduino platform hides from the programmer by employing high-level functions to control them based on relatively simple instructions. The disadvantage of such system is that the user has much less control of the detailed operation of the peripherals. This is only a disadvantage, however, if the programmer wants to have such level of control. This disadvantage is also an advantage to people new to microcontrollers, as it allows the user to make things work relatively quickly. Arduino does support low-level access to the microcontroller’s peripherals for those that want or need it.

|  |  |
| --- | --- |
|  | Watch “An Introduction to the Arduino” (4:25 minutes): [bit.ly/3vc46ZK](https://bit.ly/3vc46ZK) |
|  | Watch “*You can learn Arduino in 15 minutes.*” (16:33 minutes) to get some background on Arduino boards: [bit.ly/30urp2H](https://bit.ly/30urp2H) |

## Grove Beginner Kit for Arduino

This guide is written around the Grove beginner kit for Arduino by Seeed Studio. The Grove system is a modification to the original Arduino Uno which brings various pins out on four-pin connectors and various peripherals that you connect as needed, as shown in Figure 2. The four-pin connectors have a similar naming convention to the Arduino Uno pins discussed in the previous section.

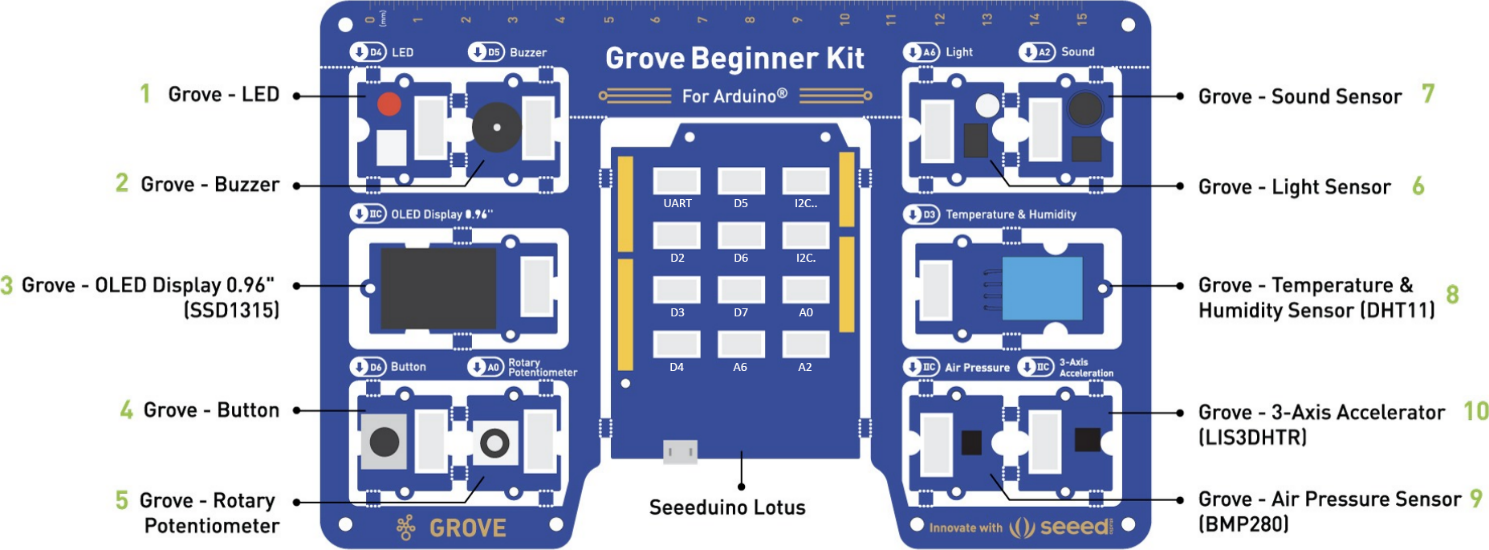


Figure . The Grove beginner kit for Arduino comprising of the Seeeduino Lotus and 10 peripherals. (Adapted from [bit.ly/2OSLrkZ](https://bit.ly/2OSLrkZ))

Despite the existence of the four-pin connectors, all peripherals come pre-connected to the Seeeduino via PCB tracks, so no cables are needed to connect! Once the peripherals are broken out of the big PCB, the provided Grove cables can be used to connect to the Seeeduino. It is recommended to leave the peripherals attached for this practical!

# Prerequisites

Before diving into the fun stuff, the Arduino Software (IDE) and the Seeeduino Lotus board need to be installed – this will make the programming of the device possible.

## The Arduino IDE

The Arduino IDE is available for Windows, Linux, and Mac OS X and can be downloaded from [arduino.cc/en/software](https://www.arduino.cc/en/software). Launching Arduino should show a window like the one in Figure 3 (without the annotations).

Graphical user interface, text, application

Description automatically generated

Figure . The Arduino IDE.

The IDE settings can be altered via **File > Preferences**. Here you can change the font, turn line numbers on, etc.

## The Seeeduino Lotus Board

To get started with the Seeeduino Lotus board, the CP2102 USB Driver for your OS needs to be downloaded and installed from [bit.ly/3rT2eD8](https://bit.ly/3rT2eD8).

After the driver is installed, connect the Seeeduino Lotus board to your computer using the provided USB cable and open the Arduino IDE.

In the Arduino IDE click on **Tools > Board > Arduino Uno** to select the correct development board model (refer to Figure 4). Then click on **Tools > Port > COMN**, where **N** is the port that your operating system has assigned to the Arduino (in this case it is **COM7**). Note that this number may change each time you connect the board!

|  |  |
| --- | --- |
| A picture containing graphical user interface  Description automatically generated | Graphical user interface, text, application  Description automatically generated |

Figure . Selecting the correct Board and Port in the Arduino IDE.

# Digital Outputs

Digitals signals are ones that have a *discrete amplitude* – in other words, the amplitude is one of a limited set of values. In Arduino, digital signals have two states: HIGH / 5 V / 1, and LOW / 0 V / 0. These could represent the state of an LED: setting a pin HIGH will light an LED connected to it up; setting it to LOW will have the opposite effect.

## Tutorial – Blinking an LED

Blinking an LED is the ["Hello, World!" program](https://en.wikipedia.org/wiki/%22Hello,_World!%22_program) for Arduino. It is typically the first program to write when learning a new microcontroller, as it’s a simple program but will introduce you to the development environment and test your connection to the hardware, as well as the hardware itself. Once you have done this, you should be comfortable with the basic Arduino system.

The first step is to connect the LED peripheral to the Arduino as shown in Figure 5. Note that if you haven’t broken out the peripherals and Arduino from the kit, this step is not necessary since they are connected via the PCB traces! This is applicable to all projects in this practical!

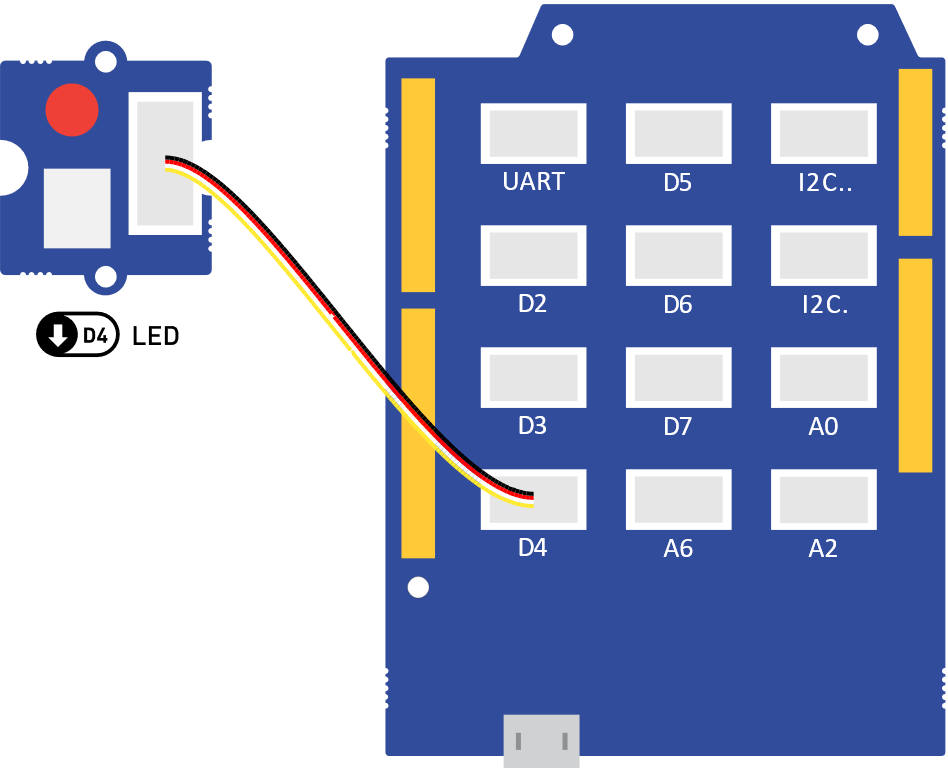


Figure . The LED peripheral connected to digital output D4.

Before we examine the typical Arduino program, type Code 1 below into the Arduino IDE, build it, and upload it. If there are errors in your code it will not upload, so will not run. Don’t ignore errors! Try to work out the cause(s) instead. A common cause is missing semi-colon(s) (;) from the end of statements. If the code ran successfully, you should see the LED blink with a period of half a second!

const int LED\_PIN = 4;  // Define the LED pin; It makes the code more readable.

void setup() {

  pinMode(LED\_PIN, OUTPUT);  // Initialize LED\_PIN as an output.

}

void loop() {

  digitalWrite(LED\_PIN, HIGH);  // Turn the LED on (HIGH is the voltage level).

  delay(250);                   // Wait for 250 milliseconds.

  digitalWrite(LED\_PIN, LOW);   // Turn the LED off by making the voltage LOW.

  delay(250);                   // Wait for 250 milliseconds.

}

Code . Code that will blink the built-in LED with a period of 500 milliseconds. The same code is available in **tutorials/01-LED-Blink/**.

Whilst is may seem a little time consuming to get you to type this in yourself it is important that you get used to writing code and finding the typos. The microcontroller can’t interpret what you type unless it is 100% correct so it requires an attention to detail.

Now let’s understand what the code above does, and how a typical Arduino program looks:

* **The first section**, at the top, is where you put the const int and #include items. These should be at the top, as the Arduino software reads the program top-to-bottom. The const int line creates a *constant integer* variable, called LED\_PIN and sets its value to 4, which refers to pin D4 of the Arduino. The #include statement is used to *include* other pieces of code, called libraries, in your program – more on that later!
* **The second section** is the first function you need and is called setup() (don’t worry about the void part). This function is called once at the start of the program and is used for code that initialises things before your main program executes. In this example, the LED\_PIN is made an output with the function call pinMode(LED\_PIN, OUTPUT). pinMode() takes two arguments: the first is the pin you are controlling, and the second is what you want the pin to be. You can only have one setup() function in your program.
* **The third section** is the main body of the program, called loop(). It is called loop() as it does exactly that – the code is executed in the order it appears, and when the program reaches the closing brace (}), it returns to the top of loop() and starts again. In this program, there are two functions that are called twice. The first one is called digitalWrite(pinNumber,   
  status). This function takes two arguments: the first is the pin you are driving (e.g. LED\_PIN), and the second is the logic level, which can be HIGH or LOW. The other function is delay(ms) which stops the code execution for the specified time (250 ms in the case above). You can only have one loop() function in your program.

You should have also notice the *human-readable* parts, starting with //. These are called comments and are used to describe what the code is doing. You can write anything inside these comments, since the Arduino program ignores everything on a line that follows //.

As an exercise, try adjusting the arguments in the delay() so that the LED is off for 750 ms and on for 250 ms.

## Exercise – Adding a Buzzer

Now, connect the buzzer from your kit as shown in Figure 6. Then, modify the program above, so that the buzzer is on, when the LED is off, and the buzzer is off, when the LED is on. If you get stuck, here are some hints:

1. Look at everything you’ve had to do for the LED.
2. In your loop(), you only need to add two more statements.
3. Put some tape over the buzzer! 😊 It’s loud!
4. If you’re still stuck, all solutions to the exercises are available in the **solutions** folder. The solution for this exercise is in **solutions/01-LED-Buzzer-Blink/**.

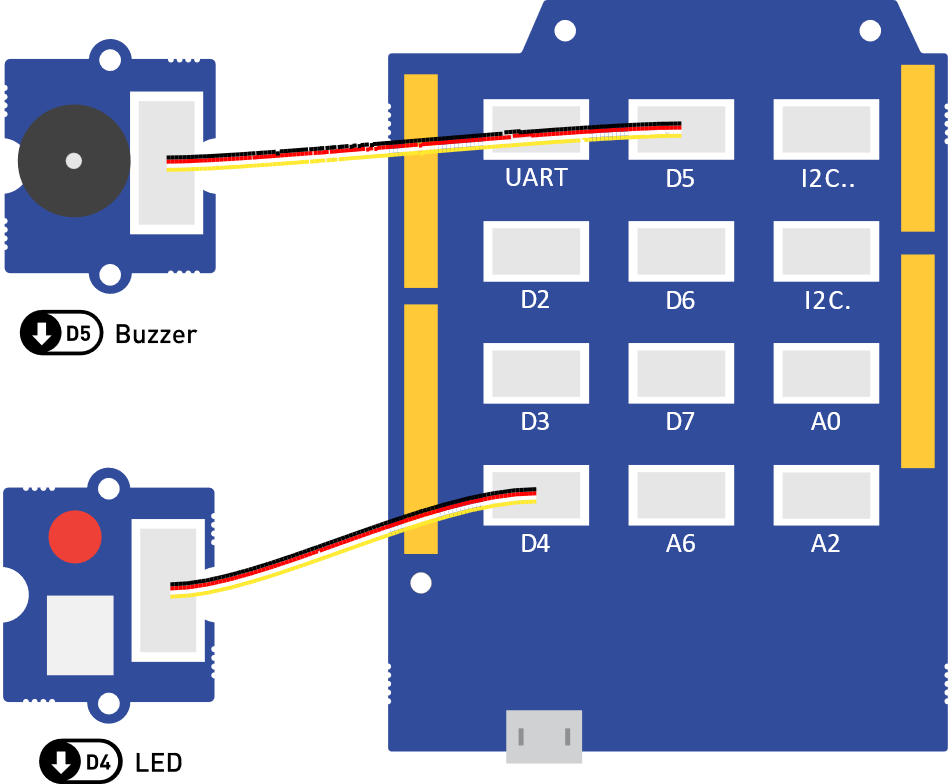


Figure . The LED peripheral connected to digital output D4 and the Buzzer connected to digital output D5.

## Key Points

Digital outputs provide on/off functionality to elements, which are connected to the digital output pins (*D0* to *D19*) of the Arduino.

const int is used at the top of an Arduino program to define global variables, such as pin definitions.

Each Arduino program has two compulsory functions - setup(), which is executed once at the beginning of the program, and loop(), which is executed repeatedly.

The pinMode() function configures a given pin to behave as an input or an output; the value of a digital input pin can be written using digitalWrite(). The delay() function pauses the program execution for a given amount of time.

The functions above, and all other core Arduino functions, are described in detail in [bit.ly/39QD7du](https://bit.ly/39QD7du).

# Digital Inputs

Like digital outputs, digital inputs in Arduino also have two states: HIGH and LOW. For example, these can be the states of a button. In this part of the practical, you will learn how switch the LED on and off using a switch.

First, we need to take a deeper look at *variables*. In the previous section we used a const int variable to store the value of an Arduino pin. The type of that variable was int, which means that it can store integers. The const keyword made that variable a constant, which means that it is not allowed to change its value while the program is running. Arduino has another type called bool, which comes from [Boolean algebra](https://en.wikipedia.org/wiki/Boolean_algebra#:~:text=In%20mathematics%20and%20mathematical%20logic,denoted%201%20and%200%2C%20respectively.), and stores a single two-state bit which can either be true (HIGH) or false (LOW). We will use this variable type below to store the state of a switch.

## Tutorial – Reading Switches

To start with the practical part of this section, connect the LED and the button as in Figure 7.

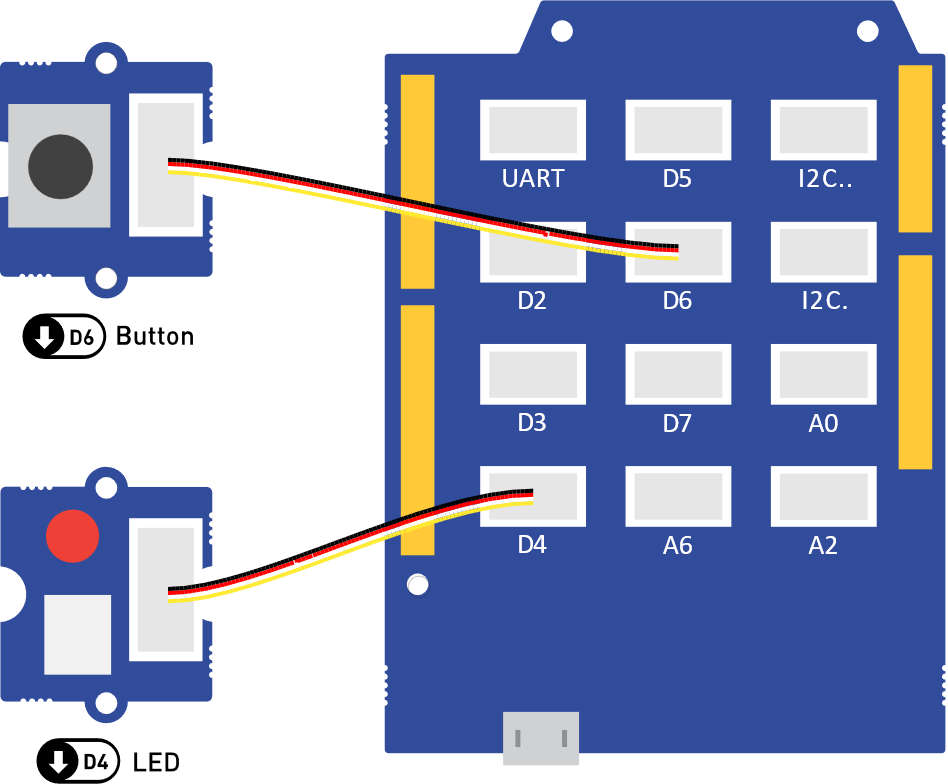


Figure . The LED peripheral connected to digital output D4, and the Button connected to digital input D6.­

Then, before running the program in Code 2, let’s understand it first. On the second line, the switch pin is defined the same way as the LED pin is. In setup(), there is a new function call to pinMode(), which sets the switch pin to be an INPUT pin. On line 4, a bool variable, called switchState, is declared that will store the state of the switch. Note that the const keyword is not used this time, since we want this value to be able to change during the program execution. The last new thing in the code below is the call to the digitalRead() function, which returns the state of the switch pin and stores it in the switchState variable. This variable is then used in the digitalWrite() to set the state of the LED.

const int ledPin = 4;

const int switchPin = 6;  // Define the switch pin.

bool switchState;  // A bool variable to store the state of the switch.

void setup() {

  pinMode(ledPin, OUTPUT);    // Initialize ledPin as an output.

  pinMode(switchPin, INPUT);  // Initialize switchPin as an input.

}

void loop() {

  switchState = digitalRead(switchPin);

  digitalWrite(ledPin, switchState);

}

Code . Code that will control the LED based on the switch state. The same code is available in **tutorials/02\_ReadingSwitches/**.

After uploading the code above to the Arduino, you should observe that the LED lights up when you press the button.

## Exercise – Inverting the Switch State

What if we wanted the LED to be on when the button is not pressed, and have it off when it is pressed? Putting an exclamation mark (!) in front of a Boolean will invert its value. For example, if the value, stored in switchState is true, then !switchState  would equal false. Modify the pin state inside the digitalWrite() function of Code 2, so that the state that is written to the LED is the opposite to the state of the button.

## Key Points

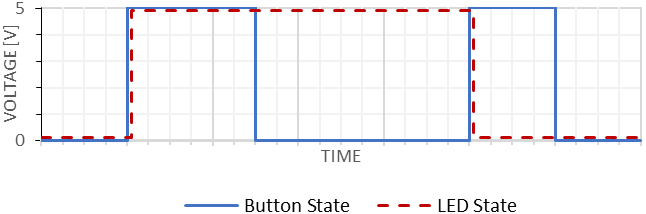
Boolean (bool) variables are used to store two-state values which can either be true (HIGH) or false (LOW).

When using digital inputs, the INPUT mode is used inside the pinMode() function.

Putting an exclamation mark (!) in front of a Boolean will invert its value.

# Conditional Logic

Although the examples above were useful for learning about digital inputs, they were so simple that the same functionality could be achieved without a microcontroller. For example, the functionality of Code 2 could be replicated by connecting a battery, an LED, and a switch in series. *Conditional logic* allows us to run certain parts of a program when certain conditions are met. For example, suppose that we wanted the LED to turn on when the button is pressed but the LED stays on even after the button is released. To turn the LED off, we would need to press the button again. This is visualised in Figure 8. The switch of a digital signal from low to high is called the [rising edge](https://en.wikipedia.org/wiki/Signal_edge) of the signal – the high to low transition is the falling edge.



Button is pressed

LED is on

Button is pressed

Figure . The digital signals of a button and an LED which is turned on/off on the [rising edge](https://en.wikipedia.org/wiki/Signal_edge) of the button.

## Tutorial – LED Button Trigger

To achieve the desired functionality above, we would need to write a program that monitors the previous and the current state of the button. Whenever the previous state was low, and the current state is high, a rising edge has occurred, and the button must have been just pressed down.

To better understand the code below, the whole functionality of the program can be formulated in an **if … then …** statement: **if** the previous button state was LOW **and** the current button state is HIGH, **then** toggle the LED state. This same functionality is written as Arduino code in the snippet below:

if (previousButtonState == LOW && currentButtonState == HIGH) {

  ledState = !ledState;            // Toggle the LED state.

  digitalWrite(ledPin, ledState);  // Update the LED to the new state.

}

You should have noticed that the Arduino syntax for the **if … then …** conditions is:

if (this condition is met) {

  then do this

}

You should have also noticed that the symbol for equality in Arduino is ==, instead of = which is the symbol for assignment. Lastly, you would have noticed the && symbol is the *logical* ***and*** *operator*.

After putting the logic above in a complete program, we get Code 3. For this tutorial keep the same connection as in Figure 7. Upload the code to your Arduino and test if the functionality is as expected.

const int ledPin = 4;

const int buttonPin = 6;  // Define the switch pin.

bool ledState;             // A bool to store the LED state.

bool previousButtonState;  // A bool to store the previous button state.

bool currentButtonState;   // A bool to store the current button state.

void setup() {

  pinMode(ledPin, OUTPUT);    // Initialize ledPin as an output.

  pinMode(buttonPin, INPUT);  // Initialize switchPin as an input.

}

void loop() {

  currentButtonState = digitalRead(buttonPin);  // Read current button state.

  // If a rising edge of the button is detected.

  if (previousButtonState == LOW && currentButtonState == HIGH) {

    ledState = !ledState;            // Toggle the LED state.

    digitalWrite(ledPin, ledState);  // Update the LED to the new state.

  }

  previousButtonState = currentButtonState;  // Current state becomes previous.

}

Code . The same code is available in **tutorials/03\_LedButtonTrigger/**.

## Exercise – LED Button Trigger with Alarm

Suppose that we now wanted a system that would beep twice every time the LED state changes. Assuming that we have setup an output pin called buzzerPin, we could add the following snippet in the conditional statement of Code 3 to produce a double beep sound:

digitalWrite(buzzerPin, HIGH);

delay(100);

digitalWrite(buzzerPin, LOW);

delay(100);

digitalWrite(buzzerPin, HIGH);

delay(100);

digitalWrite(buzzerPin, LOW);

delay(100);

That was a long snippet of code for a relatively simple functionality. Now suppose that for some reason we wanted to make the buzzer beep one hundred times. We could of course write 400 lines of code as described above, but there is a much neater way! The code snippet below allows you to generate any number of beeps, simply by storing the desired number in the numberOfBeeps variable. Your task now is to get familiar with the [for](https://www.arduino.cc/reference/en/language/structure/control-structure/for/)control structure from [bit.ly/39QD7du](https://bit.ly/39QD7du), understand the snippet below and add the functionality described in this subsection to Code 3. As always, if you get stuck, a solution with the completed code is provided in **solutions/03\_AlarmedLedButtonTrigger/**.

for (int i = 0; i < numberOfBeeps; i++) {

  digitalWrite(buzzerPin, HIGH);

  delay(100);

  digitalWrite(buzzerPin, LOW);

  delay(100);

}

## Key Points

Conditions in Arduino are written using the following syntax:

if (this condition is met) {

  then do this

}

The symbol for equality in Arduino is ==, whereas the symbol for assignment is =. The && symbol is the *logical* ***and*** *operator*.

The [for](https://www.arduino.cc/reference/en/language/structure/control-structure/for/)control structure is used to repeat a block of statements enclosed in curly braces.

The [signal edge](https://en.wikipedia.org/wiki/Signal_edge) in electronics is the term that describes the transition of a digital signal.

# Analogue Outputs

It is unusual for microcontrollers, especially small ones, to include some form of analogue output. However, Arduino includes a function called analogWrite(pin, val). How does it do it? It uses something called [*pulse width modulation*](https://en.wikipedia.org/wiki/Pulse-width_modulation) (PWM), where pin is the pin that you want to control and val is the *width* of the PWM ranging from 0 to 255.

But what is PWM exactly? Let’s have a look at Figure 9, which shows how the waveform from an “analogue output” of the Arduino would look. As the name PWM suggests, the Arduino generates a series of pulses. The signal, however, is digital since there are only two states: 0 V and 5 V. In fact, if you try to imagine an LED driven by this signal it would be blinking. So why is it called analogue? Your eyes can only respond to relatively slowly changing light. Anything flashing faster than about 50Hz will appear as a constant level. Therefore, if the signal in Figure 9 blinks fast enough, you would notice a dimmer LED, rather than a blinking LED.

Now let’s dissect the *width modulation* part in PWM. It essentially means that we can control (or modulate) the width of the pulses. The *duty cycle* describes the proportion of “on” time of the pulse within one period of the waveform. A duty cycle of 50% means that an LED driven by PWM will be “on” 50% of the time. Figure 9 shows that as the duty cycle is increased, the LED appears brighter and vice versa. Referring to the analogWrite(pin, val) function, the duty cycle can be inferred using .

|  |  |  |
| --- | --- | --- |
| Duty cycle: 0% |  | Icon  Description automatically generated |
| Duty cycle: 25% |  |  |
| Duty cycle: 50% |  |  |
| Duty cycle: 75% |  |  |
| Duty cycle: 100% |  |  |

Figure . A visualisation of the perceived brightness from an LED driven using PWM for 5 different duty cycles.

## Tutorial – Making an LED Dim

Now let’s try this in practice. Firstly, connect the LED as shown in Figure 10. Then, in the Arduino IDE type Code 4. You should be comfortable with reading and understanding the code by now – it is like the digital code from before, but it uses the analogWrite() function in the loop().

Build and upload the code to the Arduino. What do you observe? You should see an LED that shines at half of its full brightness. Experiment with different values for the WIDTH variable, such as 0, 64, 192, 255. What happens to the brightness of the LED as the width is changed?

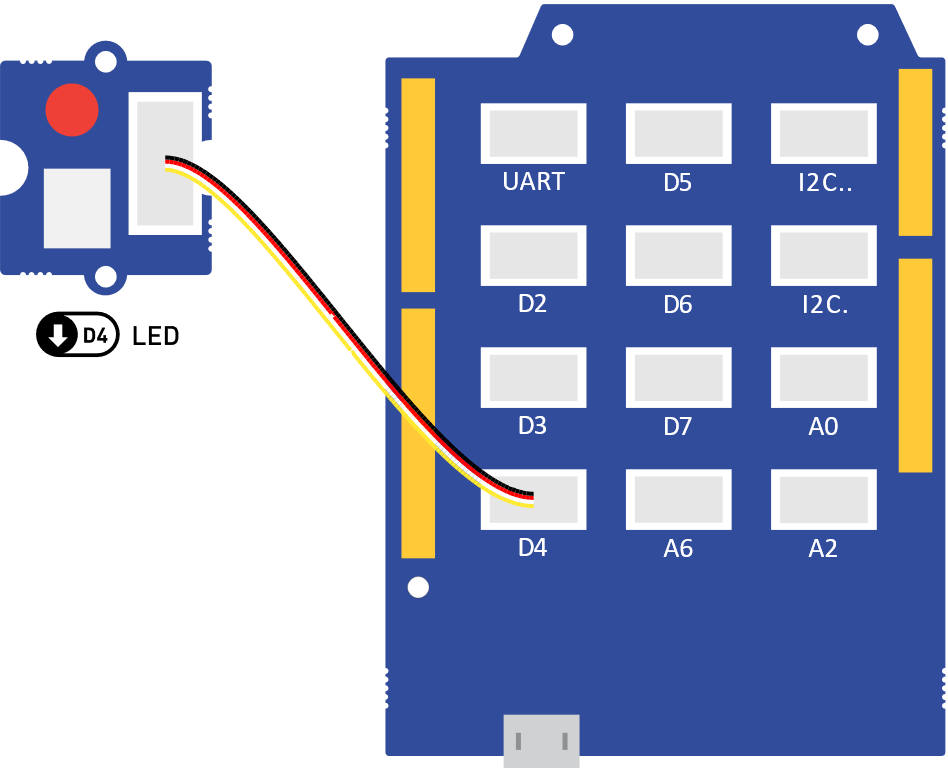


Figure . The LED peripheral connected to analogue output D4.

const int LED\_PIN = 4;  // Define the LED pin;

const int WIDTH = 128;  // Set the duty cycle to (100 \* 128 / 255) = 50%.

void setup() {

  pinMode(LED\_PIN, OUTPUT);  // Initialize LED\_PIN as an output.

}

void loop() {

  analogWrite(LED\_PIN, WIDTH);  // Drive the LED using PWM.

}

Code . Code that will make an LED dim, by driving it using PWM with duty cycle of 50%. The same code is available in **tutorials/** **04-LED-Dim/**.

## Exercise – Soft Blinking LED

In the digital output section, we started by making an LED blink. In this section we learned how to set an arbitrary brightness level to the LED, however its brightness would stay constant. In this exercise you will learn how to make the LED blink “softly” in a sinusoidal fashion. The complete program is provided for you in Code 5.

const int LED\_PIN = 4;        // Define the LED pin;

const float FREQUENCY = 0.5;  // Frequency [Hz] for soft-blinking.

void setup() {

  pinMode(LED\_PIN, OUTPUT);  // Initialize LED\_PIN as an output.

}

void loop() {

  // Set the width to 128 \* (1 + sin(2 \* pi \* f \* t)).

  int width = 128 \* (1 + sin(2 \* 3.14 \* FREQUENCY \* millis() / 1000));

  analogWrite(LED\_PIN, width);  // Drive the LED using PWM.

}

Code . Code that will blink make an LED dim, by driving it using PWM with duty cycle of 50%. The same code is available in **solutions/** **04-Soft-Blinking-LED/**.

Your task is to understand this code, then upload it to the Arduino and observe what happens. The new concepts to look at are [float](https://www.arduino.cc/reference/en/language/variables/data-types/float/) variables, the [sin()](https://www.arduino.cc/reference/en/language/functions/trigonometry/sin/) and [millis()](https://www.arduino.cc/reference/en/language/functions/time/millis/) functions. As a reminder, all core things to know about Arduino code are available at [bit.ly/39QD7du](https://bit.ly/39QD7du). Once you are comfortable with understanding the code, try different FREQUENCY values, such as 0.25, 1, 2. What do you observe?

## Key Points

Analogue outputs provide a way of generating “analogue” signals using PWM. The relevant Arduino pins are labelled with ~, such as *~D3*, *~D5*, *~D6*, *~D10*, *~D11.*

The analogue value of a signal is controlled by the duty cycle of the PWM signal. This can be calculated using .

The analogWrite() is used to control the duty cycle and accepts values between 0 and 255.

The float variables allow you to store non-integer values. The sin() function allows you to calculate the sine of a given input and the millis() function returns the number of milliseconds passed since the Arduino board began running the current program.

# Analog Inputs

Arduino programs work in the digital domain. This means that they cannot work with analogue signals directly. Instead, analogue signals are first passed through an [analogue-to-digital converter](https://en.wikipedia.org/wiki/Analog-to-digital_converter) (ADC). This is circuit that reads an analogue voltage and assigns a number to it that can be processed digitally. In the case of the ADC on the Arduino, it splits the input voltage into 1024 possible steps, with 0 V assigned the value 0 and 5 V assigned to 1023. It is linear, so we can predict the number the ADC will return for any input (see Figure 11). This number is always an integer and is equal to where Vin is the input voltage and Vmax is the maximum input allowed and is the [math floor function](https://en.wikipedia.org/wiki/Floor_and_ceiling_functions) (essentially it rounds down to the nearest integer). For the Arduino, is . An input of should give us .

Figure 11. Mapping between analogue input voltages and ADC readings on the Arduino.

Reading an analogue input value in Arduino is done using the analogRead(pinNumber) function. As mentioned above, this function returns an integer in the range , which can be stored in a variable of type int.

## Tutorial – LED with Controllable Intensity

In the previous section, we created a programmatically dimmable LED. However, it was impossible for someone to change its brightness without reprogramming the device. Let’s change this and create a program which allows us to change the LED’s brightness using a potentiometer.

We already know how to write an analogue value (from the previous section), and just learned how to read one (using analogRead()). One problem that you might have noticed, however, is that the values for analogue outputs range between 0 and 255, whereas those for analogue inputs range between 0 and 1023. We could, of course, convert between these two ranges by multiplying or dividing by 4. However, Arduino has a convenient function called map(), which can do the mapping between these ranges for us. You can read about the map() function at [bit.ly/3tbziqq](https://bit.ly/3tbziqq), or try and infer how it works from the code below – it is quite simple.

To complete this tutorial, connect the rotary potentiometer and the LED as shown in Figure 12. Then, in the Arduino IDE type Code 6 and upload it to the Arduino. After the code is uploaded, try rotating the potentiometer’s shaft. You should observe that this changes the LED’s brightness.

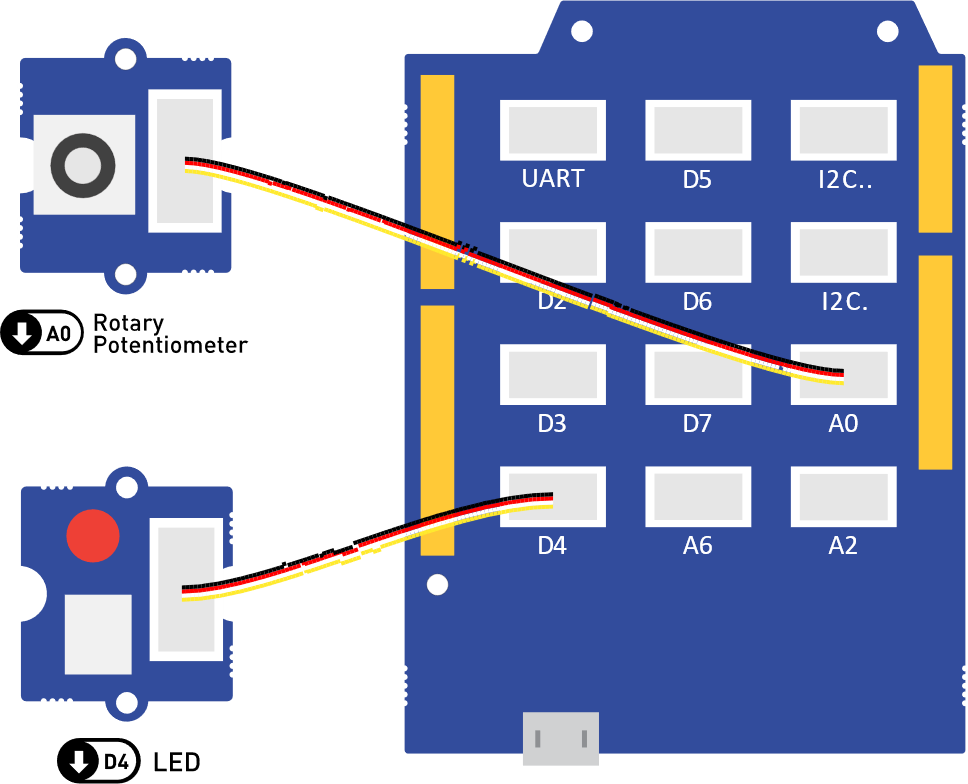


Figure . The LED peripheral connected to digital output D4, and the rotary potentiometer connected to analogue input A0.­

const int ledPin = 4;            // Define the LED pin.

const int potentiometerPin = 6;  // Define the potentiometer pin.

void setup() {

  pinMode(ledPin, OUTPUT);           // Initialize ledPin as an output.

  pinMode(potentiometerPin, INPUT);  // Initialize potentiometerPin as an input.

}

void loop() {

  // Read the potentiometer value in the range [0, 1023].

  int potentiometerValue = analogRead(potentiometerPin);

  // Convert the value from the range [0, 1023] to the range [0, 255].

  int ledValue = map(potentiometerValue, 0, 1023, 255, 0);

  // Set the converted value to the LED.

  analogWrite(ledPin, ledValue);

}

Code . Code that will make an LED dim based on the input from a rotary potentiometer. The same code is available in **tutorials/05-LED-Potentiometer/**.

Now try and change the code such that the map() function maps from to (instead of . This can be achieved by swapping the last two arguments (…, 255, 0), to be (…, 0, 255) and uploading the modified program to the Arduino. What change in behaviour do you observe?

## Exercise – Replacing the Potentiometer

Conveniently, all compatible analogue inputs act the same. This means that you can replace the potentiometer with the light sensor or the sound sensor and have an LED that is reactive to light or sound without changing the logic of your program. Try this for yourself!

## Key Points

Analogue input signals need to be converted to digital via an ADC, before using them in Arduino. The relevant Arduino pins are labeled as (e.g. ).

The analogRead(pinNumber) function is used to read analogue inputs. It returns an integer in the range .

The map() function can be used to map between two ranges, such as the range of analogue inputs and analogue outputs .

# Serial Communication

So far, we interfaced the Arduino to both digital and analogue inputs and outputs and this has probably sparked your creativity and exposed you to some of the many things Arduino can do. But how do we interact with more complex devices, such as a digital accelerometer, OLED displays and even your own computer? The answer is [serial communication](https://en.wikipedia.org/wiki/Serial_communication). Essentially, serial communication works by converting information to a stream of bits, which are sent over a wire. Don’t worry if this sounds complicated, you don’t need to understand the details to get it working.

## Tutorial – Communicating with a Computer

The [Serial](https://www.arduino.cc/reference/en/language/functions/communication/serial/) family of functions in Arduino allow us to use the [UART](https://en.wikipedia.org/wiki/Universal_asynchronous_receiver-transmitter) communication protocol which can be used to communicate to a computer. To use these functions, we need to first call

void setup() {

  Serial.begin(9600);               // Begin the Serial communication.

  Serial.println("Hello, World!");  // Send a message to the computer.

}

void loop() {}

Code 7.

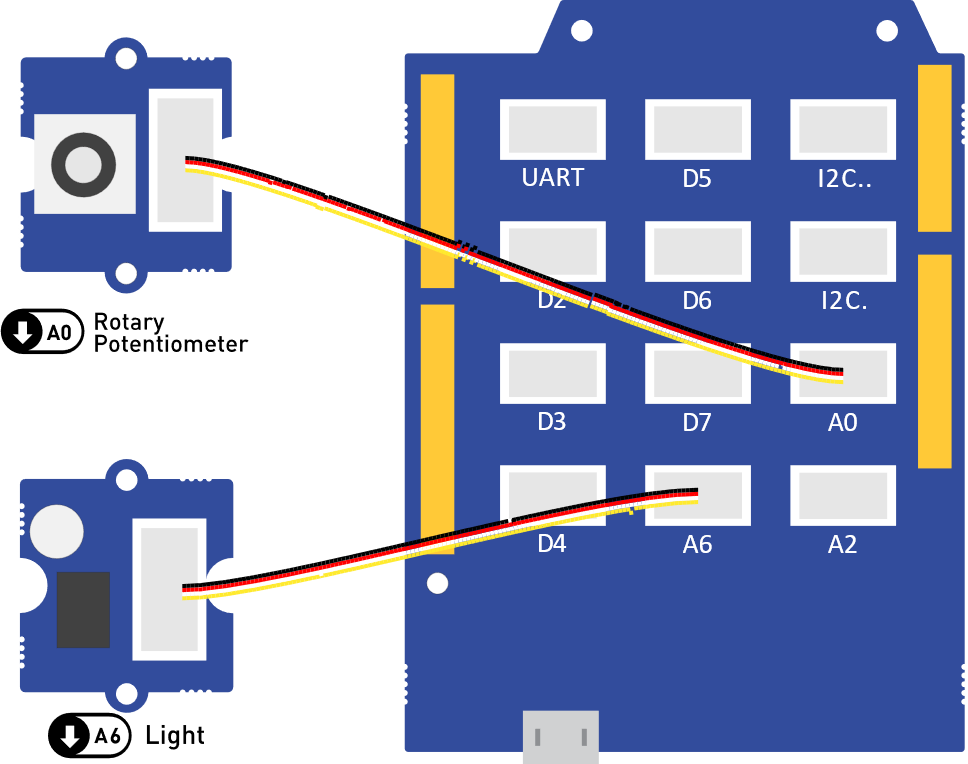


Figure 13.

Code 8.

## Tutorial – Reading and Plotting Acceleration Data

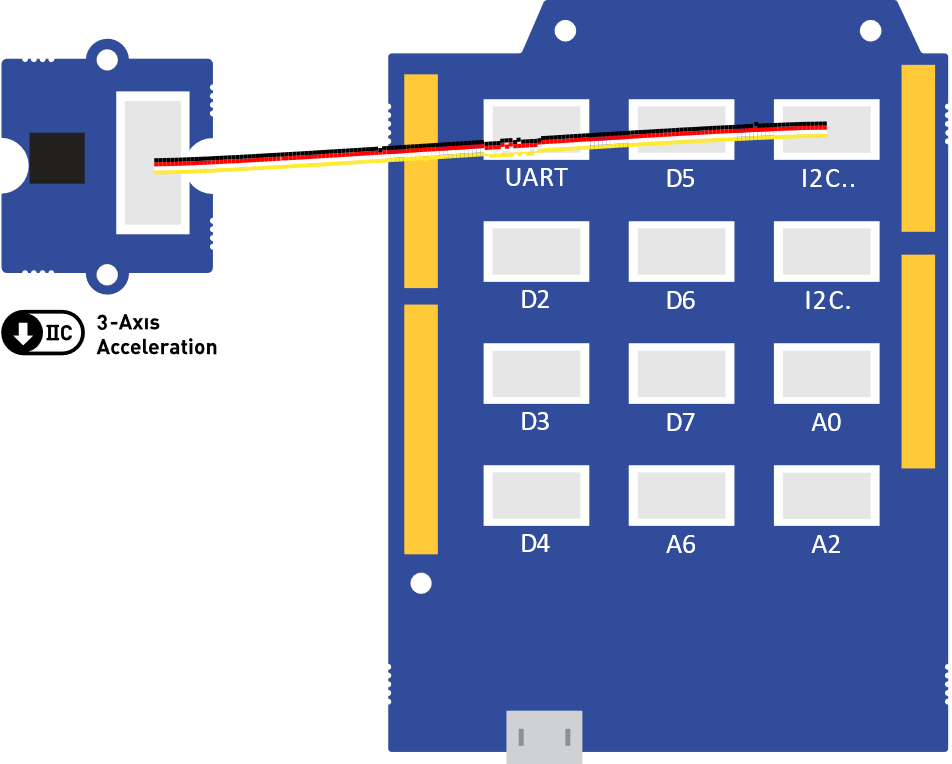


Figure 13.

Code 8.

## Key Points

# Final Project – Weather Station

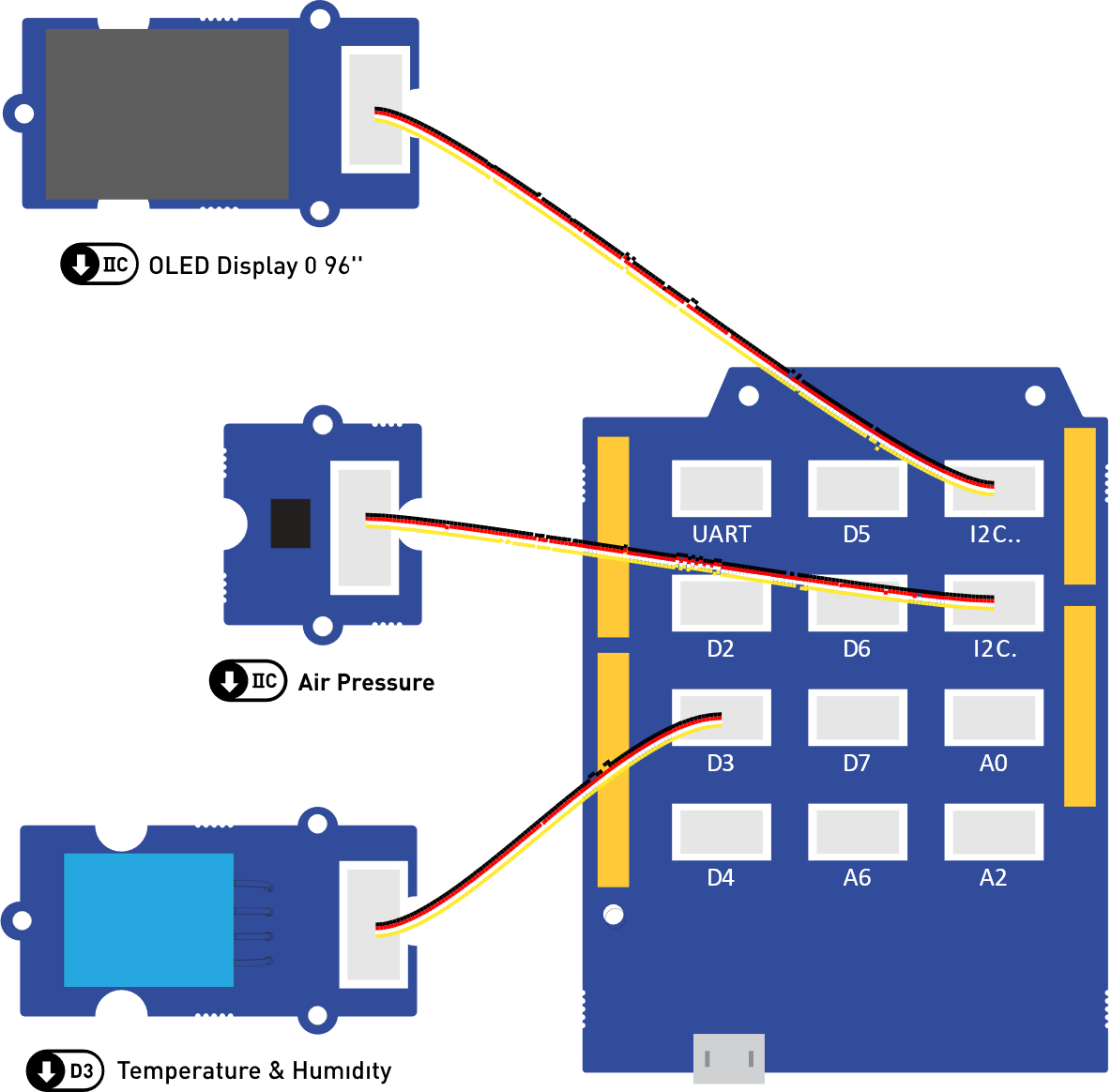


Figure 13.

# Next Steps

# Temporary Section – Please Ignore This

Convert px to cm, using 157 PPI: <https://pixelsconverter.com/pixels-to-centimeters>

Aston university document: <https://docs.google.com/document/d/1Zls-NdR7Ut9zBGM7ctqnLhijWYMZHfcr/edit>

Shape radius is 2.