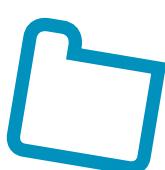
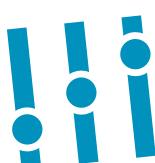
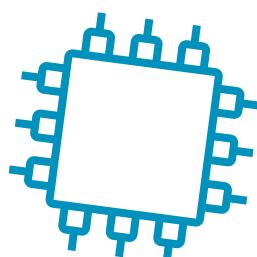


Arduino MKR

projects

for

schools



With updates, 2025

arm school program

Arm Education Media is an imprint of Arm Limited, 110 Fulbourn Road, Cambridge, CB1 9NJ, U.K.

Copyright © 2025 Arm Limited (or its affiliates). All rights reserved.

No part of this publication may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording or any other information storage and retrieval system, without permission in writing from the publisher, except under the following conditions:

Permissions

- You may download this book in PDF format from the [Arm.com](#) website for personal, non-commercial use only.
- You may reprint or republish portions of the text for non-commercial, educational or research purposes but only if there is an attribution to Arm Education.

This book and the individual contributions contained in it are protected under copyright by the Publisher (other than as may be noted herein).

Notices

Knowledge and best practice in this field are constantly changing. As new research and experience broaden our understanding, changes in research methods and professional practices may become necessary.

Readers must always rely on their own experience and knowledge in evaluating and using any information, methods, project work, or experiments described herein. In using such information or methods, they should be mindful of their safety and the safety of others, including parties for whom they have a professional responsibility.

To the fullest extent permitted by law, the publisher and the authors, contributors, and editors shall not have any responsibility or liability for any losses, liabilities, claims, damages, costs or expenses resulting from or suffered in connection with the use of the information and materials set out in this textbook.

Such information and materials are protected by intellectual property rights around the world and are copyright © Arm Limited (or its affiliates). All rights are reserved. Any source code, models or other materials set out in this reference book should only be used for non-commercial, educational purposes (and/or subject to the terms of any license that is specified or otherwise provided by Arm). In no event shall purchasing this textbook be construed as granting a license to use any other Arm technology or know-how.

The Arm corporate logo and words marked with ® or ™ are registered trademarks or trademarks of Arm Limited (or its affiliates) in the U.S. and/or elsewhere. All rights reserved. Other brands and names mentioned in this document may be the trademarks of their respective owners. For more information about Arm's trademarks, please visit <https://www.arm.com/company/policies/trademarks>.

Arm is committed to making the language we use inclusive, meaningful, and respectful. Our goal is to remove and replace non-inclusive language from our vocabulary to reflect our values and represent our global ecosystem.

Arm is working actively with our partners, standards bodies, and the wider ecosystem to adopt a consistent approach to the use of inclusive language and to eradicate and replace offensive terms. We recognise that this will take time. This book may contain references to non-inclusive language; it will be updated with newer terms as those terms are agreed and ratified with the wider community.

Contact us at education@arm.com with questions or comments about this course. You can also report non-inclusive and offensive terminology usage in Arm content at terms@arm.com.

ISBN: 978-1-911531-47-0

For information on all Arm Education Media publications, visit our website at
<https://www.arm.com/resources/education/books>

To report errors or send feedback, please email edumedia@arm.com



CONTENTS

Introduction 7

1 A Flashing LED 8

2 Flashing an External LED 12

3 Responding to a Button 15

4 LED Strobes on Demand 19

5 Reading and Writing with the Serial Monitor 24

6 Introduction to the Arduino 27

7 The IoT Cloud 31

8 IoT Customer Satisfaction Ratings 35

9 Air Quality Sensor 38

10 Storing IoT Data in a Spreadsheet 42

11	Pedometer	49
12	Baseball or Cricket Bat Metrics	55
13	Biodome (Part 1)	60
14	Biodome (Part 2)	64
15	Data Analysis and Machine Learning	68
16	Using Machine Learning to Identify a Hit or a Miss	71
17	Web Scraper	77
18	GPS Treasure Hunt	83
19	Lock Box	87
20	Final Project	92



INTRODUCTION

This book introduces learners to the exciting world of microcontrollers and the Internet of Things. Learners will use both simulators and physical devices to build systems and solve real-life problems. A variety of input and output devices will be used to receive instruction from the user, sense the environment, and display both status information and complex outputs.

The book takes a gradual approach to learning, with simulations and emulators used to build confidence, before learners begin constructing basic physical circuits. A variety of Internet services are applied to turn the devices into true Internet of Things devices, both to instruct the device to carry out a task and to report data from the device's sensors.

As well as stand-alone devices and those connected to the Internet of Things, learners will experience some other exciting modern technologies such as Machine Learning-based Artificial Intelligence using the popular Python language.

All the code examples and solutions for these projects are available on GitHub: <https://github.com/arm-university/Arduino-MKR>

Understanding the Arduino

Learners often see microcontrollers such as the Arduino as delicate, confusing and scary. While there are elements of delicacy to the Arduino, such as the pins being easy to bend, learners should not fear the Arduino. It helps to think about the Arduino—and any microcontroller—as simply being a small computer; all it does is take inputs, calculate the right thing to do, and produce some outputs. The fact that we use sensors instead of keyboards and mice, and LEDs and motors rather than a screen, should not detract from this.

The hardware

There are a multitude of Arduinos on the market, from the tiny “nano”, the ubiquitous “uno”, and many more. Although this book focuses on the MKR series, with in-built WiFi, the general layout of almost all Arduinos are the same. That is, with a number of ground (GND) and positive (3.3 V or 5 V) pins; and a range of digital (identified by a number) and analogue (identified by 'A' followed by a number) input/output pins, some of which support pulse width modulation (PWM, identified by a ~). With a bit of exploration any project in this book should be possible on almost any Arduino, although having built-in WiFi certainly makes things easier. See Chapter 6 for more information and diagrams on the Arduino.

1. A FLASHING LED

Setting the scene

One of the great things about microcontrollers, such as the Arduino, is that they are extremely small and only include the things that are really needed—these often don't include big parts such as a screen or keyboard, which have a huge environmental impact to build and run.

This project introduces you to the idea of programming Arduino microcontrollers, but in the safe environment of an emulator. Although this project is very simple, it will form the basis of a lot of future projects. You will use the skills covered here to check that everything is working, test connections, and even output useful information.

Success criteria

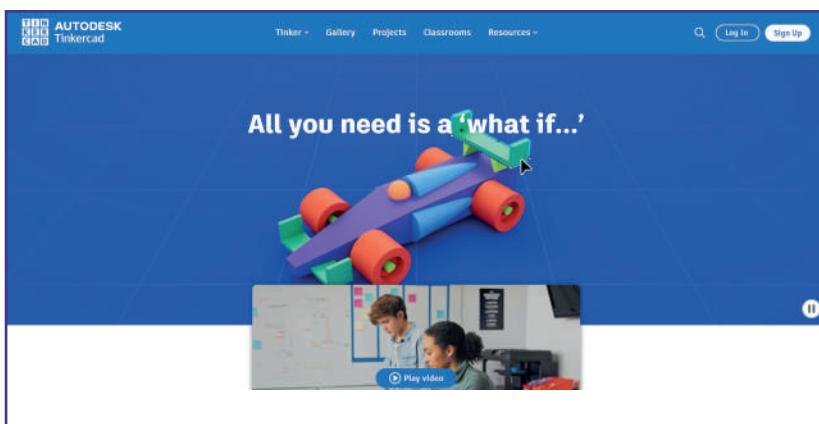
- Create a Tinkercad account and log in.
- Add an Arduino to your Tinkercad workspace.
- Create a simple program using Tinkercad and programming blocks.
- Run the program to make the LED blink.

1

Microcontrollers such as the Arduino are really just small computers. They can be programmed, just like a normal computer, and they can be connected to bits of hardware, just like a normal computer. However, because microcontrollers are so small, you have to do this programming and connecting somewhat differently. Before you start using real circuits and plugging things into them, it's a good idea to test things out first. To do this you are going to use an *emulator*, a computer program that pretends to be an Arduino.

2

For your emulator you will use a website called Tinkercad. Visit the website at <https://www.tinkercad.com/>, create an account, and sign in.



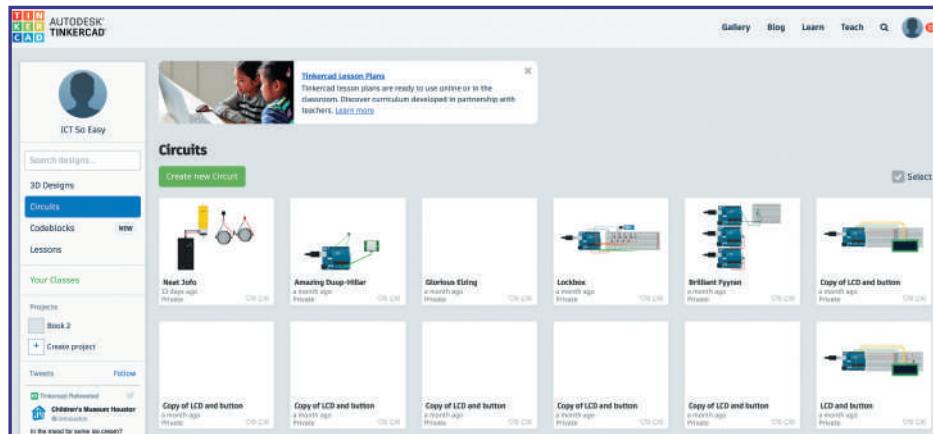
PRO TIP

Existing Google accounts are a great way to sign up and log in to Tinkercad quickly. However, some accounts with parental controls in place might block access to Tinkercad.

3

Throughout the following chapters you will be creating **circuits** in Tinkercad.

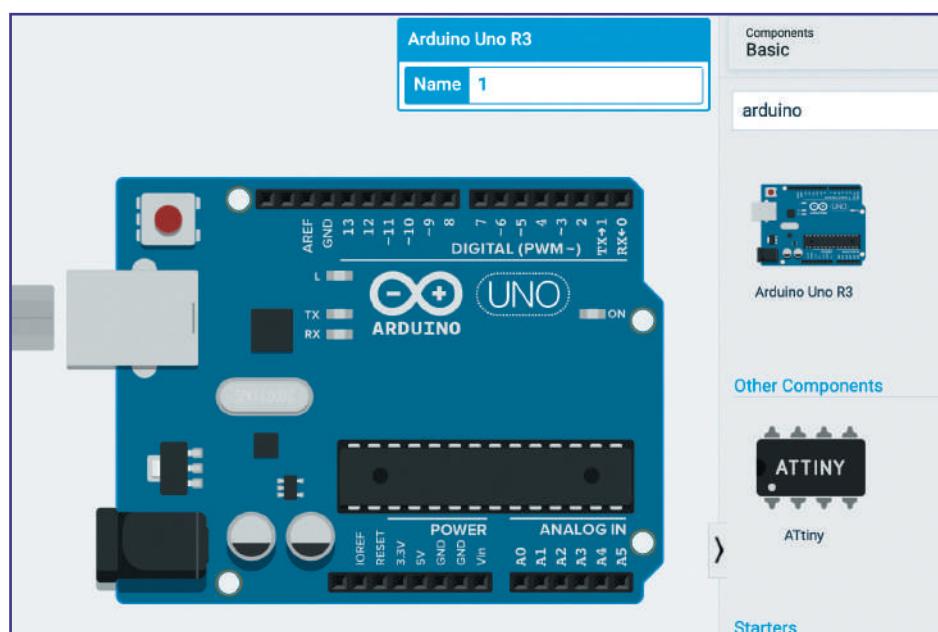
You can have as many circuits as you need. Just select **Circuits**, then click on **Create new Circuit** whenever you need to create a new circuit.



4

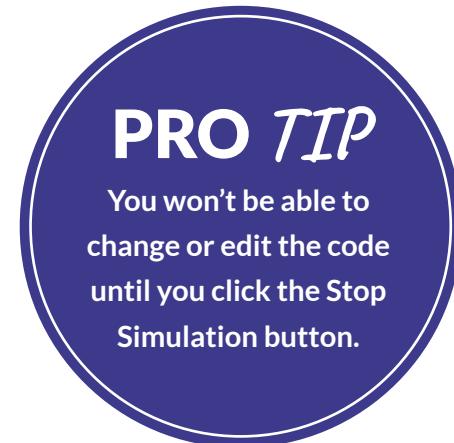
The main part of the Tinkercad screen is a workspace. You can drag **components** from the right-hand side on to the workspace and, when needed, connect them with wires. For this project you just need an Arduino on your workspace. You can either search for it or browse the list, then drag it on to your workspace.

If you have access to a real Arduino, you could compare it to the one on TinkerCad. It may look a bit different unless you have the same model. The model on TinkerCad is an Arduino Uno.



5

Microcontrollers only become useful when you tell them to do something. You do this by programming the microcontroller. The Arduino Uno, like most Arduinos, has a built in LED (light-emitting diode). An LED is like a light bulb but uses less power and is, therefore, better for the environment. Open the **Code** menu at the top right. You will see some blocks of code ready prepared. Click on the **Start Simulation** button to see the LED flash.



6

Change the time in the orange “wait” blocks to adjust the amount of time the LED flashes on and off. It’s worth looking at the terminology used here. The terms HIGH and LOW represent the voltage that the Arduino outputs: HIGH means the Arduino outputs a high voltage, which turns the LED on; LOW means the Arduino outputs a low voltage, which turns the LED off.



7

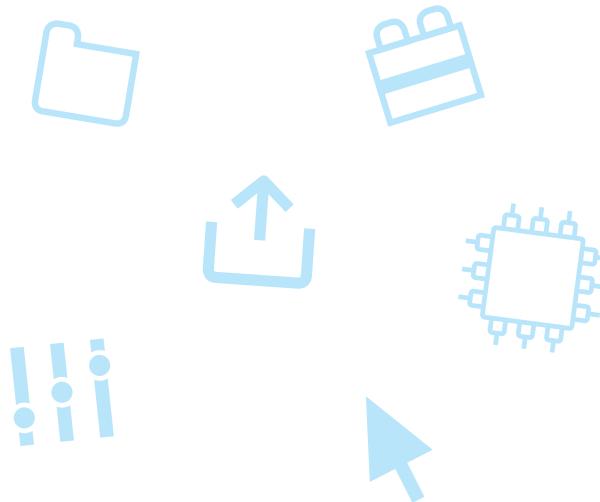
Change the **Blocks** menu to **Blocks + Text** to see the actual programming code for this program. The code is written using a combination of the programming languages C and C++, which you will use later in this book.

The screenshot shows the Tinkercad interface with the "Blocks + Text" menu selected. On the left, there is a palette with categories: Output (set built-in LED to HIGH), Input (set pin 0 to HIGH), Notation (set pin 3 to 0), Control (rotate servo on pin 0 to 0 deg), Math (play speaker on pin 0 with tone 60), Variables (turn off speaker on pin 0), and Text (print to serial monitor "hello world"). On the right, the code view shows the following C/C++ code:

```
1 void setup()
2 {
3     pinMode(13, OUTPUT);
4 }
5
6 void loop()
7 {
8     digitalWrite(13, HIGH);
9     delay(3000); // Wait for 3000 millisecond(s)
10    digitalWrite(13, LOW);
11    delay(3000); // Wait for 3000 millisecond(s)
12 }
```

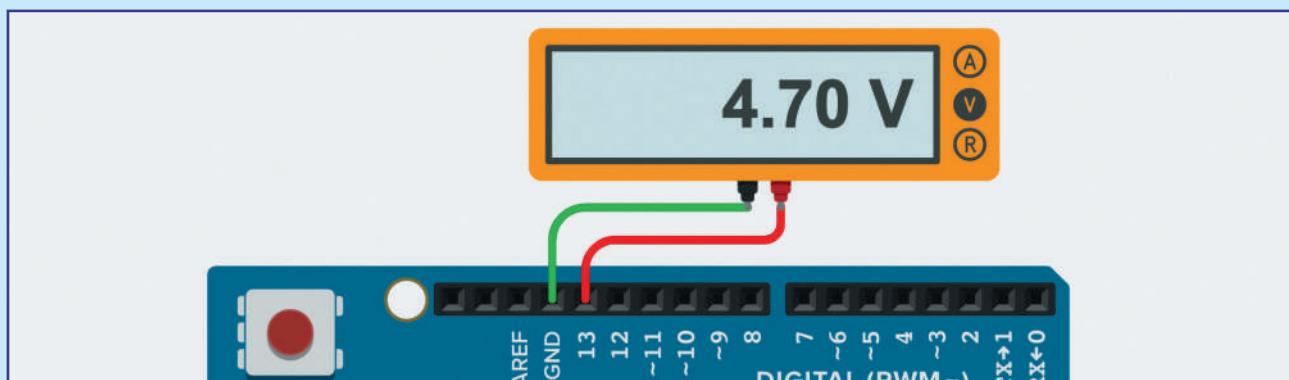
PRO TIP

Everything that can be done in blocks can be done in C and C++. The reverse isn’t necessarily true, however, so Tinkercad blocks you from changing back from C and C++ to blocks. Make sure you know the code before you change to just the text view, in case you cannot undo the process.



Testing

The code provided is simple and has been pre-written, so it should just work. However, it is always worth checking that your circuit is giving the expected readings. Find a multimeter and drag it on to your workspace. Draw a line to connect a wire from the black terminal on the multimeter to the GND hole on your Arduino. Draw another wire from the red terminal to the 13 hole. These holes are called *pins*—if you turn an LED over you will see legs or pins sticking out of the base. The built-in LED is connected to pin 13. When you start the simulation, the multimeter should read about 4.7 V when the pin is “HIGH” and 0 V when the pin is “LOW”.



Stretch tasks

- Swap the wires around on the multimeter and research why the voltage reading is now negative when the pin is “HIGH”.
- Reduce the delay times between the LED being on and the LED being off. Identify the fastest flashing rate that you can manage with the LED.
- Microcontrollers don’t (normally) have screens, so they often use LED flashes to indicate successes, failures and statuses. Change the flashing timings, so that the LED flashes “OK” in Morse code.

PRO TIP

Terminology is often used interchangeably. In this case the term “delay” (used in code) is the same as “wait” (used in blocks).

Final thoughts

LEDs are far more efficient than traditional light bulbs because they use a lot less power to produce a similar amount of light. Research the difference in the amount of electricity used by an LED light bulb and a traditional light bulb. How much of a saving would you make by swapping all the light bulbs in your home to LEDs?

2. FLASHING AN EXTERNAL LED

Setting the scene

In Chapter 1 you learned how to make the built-in LED on an Arduino flash. While this is useful for testing things, it is somewhat limited by:

- being attached to the Arduino
- only being a single LED

It is more common to want to build one or more external LEDs into your circuit and control the LEDs from your Arduino. The Arduino could form part of a larger system in which the LED could poke through a case containing your circuit, and display whether the device is working properly or not. Consumer electronics contain many precious metals, of which we have a limited supply, and being able to use a low-powered LED as an indicator, rather than installing something larger like a screen, is far better for the environment.

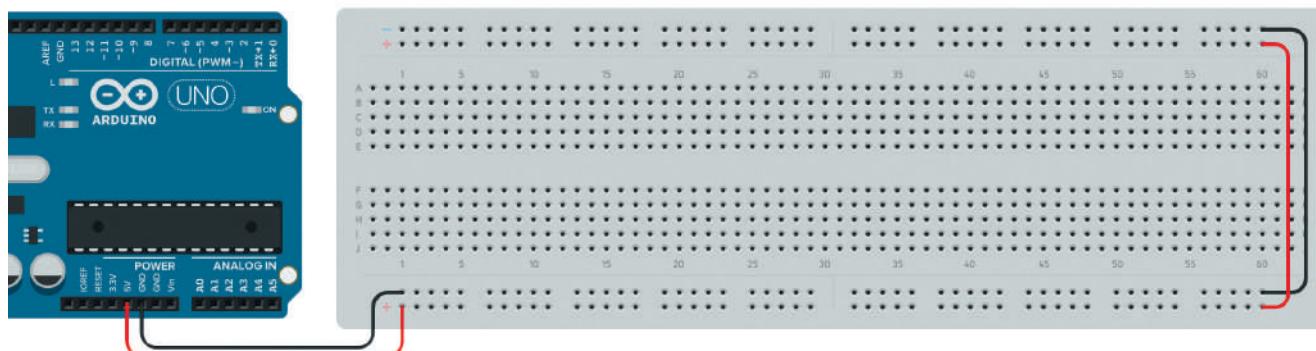
Success criteria

- Add a breadboard to the workspace.
- Add an LED to the circuit.
- Connect the LED to the Arduino with wires.
- Reduce the amount of voltage reaching the LED.
- Connect the LED to a digital pin.

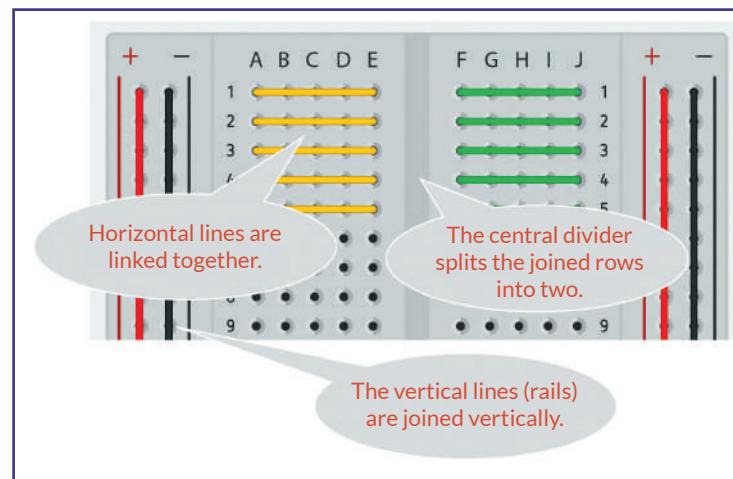
1

While it is perfectly possible to build a whole circuit by joining wires to the ends of components such as an LED, it is far more common to use a breadboard to build your circuit. Breadboards come in a variety of sizes, but most have these common features:

- There are positive (+) and negative (-) rails, which run along the top and bottom of the breadboard. This means you can easily attach components to a high voltage (the +) and to ground (the -) without running lots of wires back to the Arduino. The rails are often bridged top to bottom for even more ease of use.



- The columns are numbered, and the rows are lettered. All the pins in a column are normally connected together, although there is usually a gap in the middle. Look at a breadboard in Tinkercad. You will see that column 1 rows A to E are connected, then there is a gap, then column 1 rows F to J are connected.
- You can insert component legs into the holes to build the circuit.

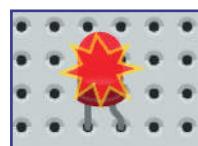


2

It is always worth building up a circuit in stages. That way you can test at each stage to ensure that everything is working. To test this circuit, set up a breadboard and drop an LED on to it. You will see the legs of the Arduino pop into two holes of consecutive columns. Because they are separate columns, they are not connected to each other.

LEDs are directional, so they will only light up if you wire them the right way around. The *cathode* (the left leg on Tinkercad and the shorter leg on a real LED) should be connected to ground, and the *anode* (the other one) should be connected via the + rail to the 5 V pin on the Arduino. When you start the simulation, the LED will light up. Having a 'correct' direction is known as polarity.

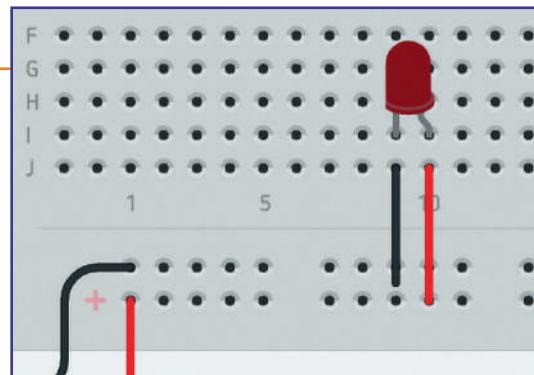
3



When you start your simulation, you will notice a red flash symbol over the LED. This indicates that there is

too much voltage getting to the resistor. In a real circuit, this could burn out the LED or even set it on fire.

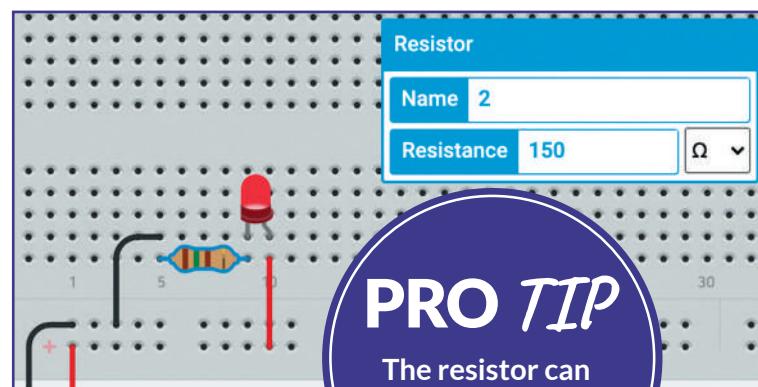
It is time to program the Arduino to turn the LED. Resistors are measured in ohms (Ω) and for this LED you need at least $150\ \Omega$. When you are building real circuits, you will need to calculate the correct resistor.



PRO TIP

If the LED doesn't light up, it may be wired the wrong way around. The wiring should be as follows:

- Arduino GND → negative rail → LED cathode
- LED anode → positive rail → LED 5 V.

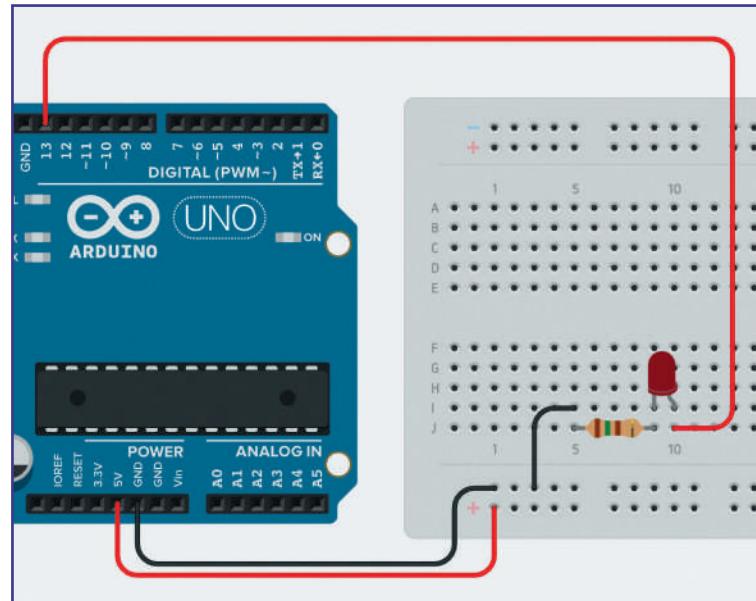


PRO TIP

The resistor can go either side of the LED.

4

Now that you know the LED circuit is working, it is time to have the Arduino turn the LED on and off, rather than giving it a constant power. To do this, remove the wires from the positive rail to the anode. Instead, connect the anode to pin 13 on the Arduino. Do you remember from Chapter 1 that the built-in LED is connected to pin 13? This means that the same code you used in Chapter 1 will work here, so you can go ahead and start the simulation. The LED should flash on and off.



Testing

You can test this circuit in the same way as the circuit in Chapter 1. Use a multimeter and connect the black terminal to the cathode and the red terminal to the anode. You should see intermediate values of approximately 2 V. If not, you can in turn move the multimeter wires (or probes) back through the circuit until you find the problem.

Stretch tasks

- Experiment with the value of the resistors and see what happens if you use larger or smaller values.
- Identify the highest value resistor you can use and still see the LED lit up, and the lowest which does not show an error (an explosion symbol over the LED). Note that this can be harder to identify in real-life circuits, so it is worth finding a value where it is obvious that the LED is on.
- Change the LED colour and investigate which ones need higher or lower resistors to produce full brightness.
- Research “LED resistor calculators”. You already know that the Arduino produces ~5 V and the ideal resistor is $\sim 150\ \Omega$, so what must the forward voltage of the LED be?

Final thoughts

LEDs come in a range of sizes, shapes, and colors, and are used in all sorts of devices around the home. It can be interesting to look around your home and see how many different LEDs you can find. Consider the impact on the environment of using traditional light bulbs instead of LEDs. Similarly, look at all of the devices in your home that use LEDs and consider the impact on the environment should the developers have chosen to use LCD displays or other, larger, devices to show the status of the device.

3. RESPONDING TO A BUTTON

Setting the scene

So far, you have used the Arduino as an output device—the LED is giving you information (albeit not a lot of information at this stage) from the Arduino. The next step along your journey is to use an input device to give the Arduino some information. You will do this by using a button. The Arduino can read the state of this button and light (or not light) the LED.

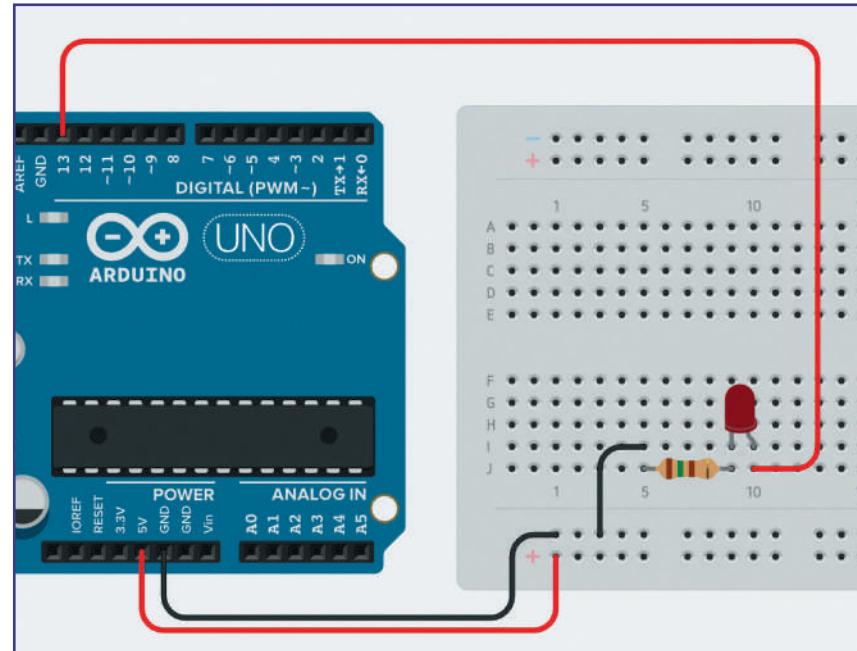
The great thing about buttons, compared to other ways of communicating with the device, is their simplicity. Unlike using a cell phone over the web or even a rotating dial, there is very little that can go wrong with a button. This means that the device will last longer, which is better for the environment.

Success criteria

- Add a button to the circuit from Chapter 2.
- Connect the button to pin 13.
- Read the input from the button.
- Turn the LED on when the button is pressed.
- Turn the LED off when the button is unpressed.

1

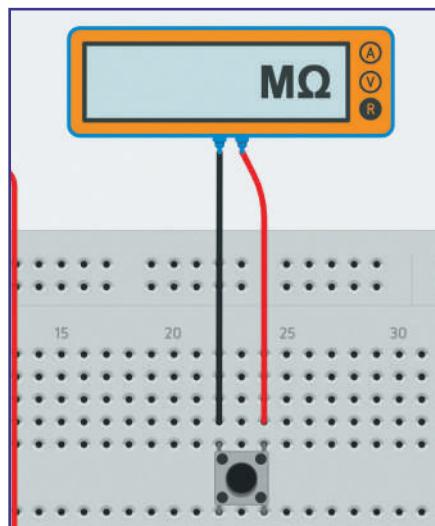
Start this project using the same circuit you finished with at the end of Chapter 2. The LED should be connected to ground and pin 13, via a $150\ \Omega$ resistor. If you try this on a real-life Arduino, you may not have exactly the right resistor; you may have a resistor in the region of $220\text{--}270\ \Omega$. Anything around that value will be fine.



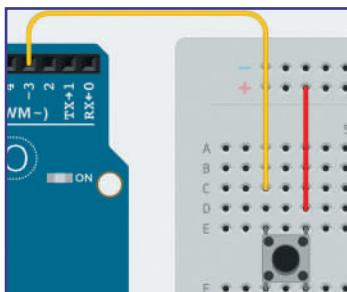
2

The button has legs or *terminals*—like the pins on the Arduino. If you hover your mouse over them, the terminal names will show up. Add a button to your breadboard. The button will fit nicely across the middle section of the breadboard. The “switching” happens between the numbered terminals—between 1a and 2a or between 1b and 2b.

Connect the multimeter to two of the terminals, then press the R button on the multimeter to measure the resistance. When the button is unpressed, the multimeter will read $M\Omega$, which means there is infinite resistance. When you press the button, the reading should change to $0\ \Omega$, which means there is no resistance.



3



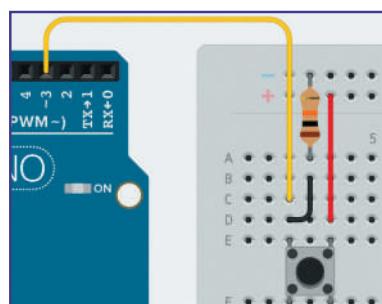
To connect the button, connect a 2 terminal to the positive rail, and connect a 1 terminal to a digital pin on the Arduino, such as D3. When the button is unpressed, the Arduino will not receive any voltage and will see a “LOW” signal. When the button is pressed, voltage gets through and the Arduino will see a “HIGH” signal (which you can see as a positive voltage if you connect a multimeter).

PRO TIP

Hover your mouse over the terminals of the button to see the labels showing what each terminal is called.

4

However, in real life connecting the button in this way can cause a problem. If you try to make this circuit, when the button is not pressed you will see the LED flickering slightly. This is due to the open circuit voltage floating—this means that the signal doesn’t know if it should be low or high, and any stray signal can cause it to be misread.



PRO TIP

The exact resistance doesn't matter here. If you are doing this project on a real circuit, a resistance of about $1\ k\Omega$ ($1,000\ \Omega$) will be fine.

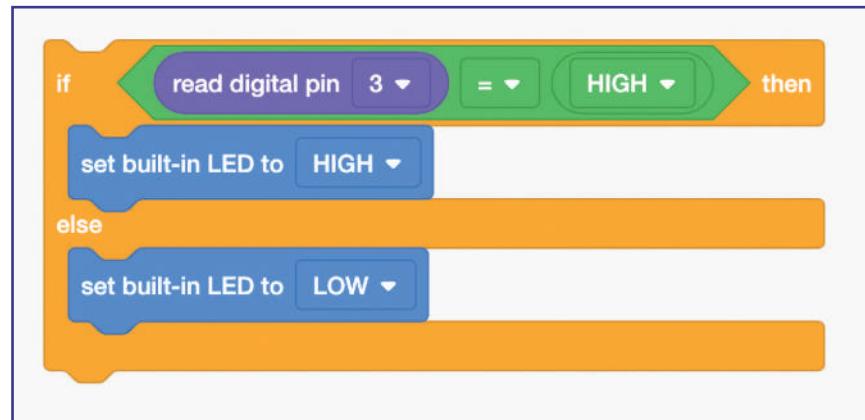
To prevent this problem, connect a high-resistance ($10\ k\Omega$ or $10,000\ \Omega$) resistor between the open side of the button (the side that goes to the Arduino) and ground. This resistor pulls the voltage down to ground when the button is not pressed, so there is a definite “LOW” signal. When the button is pressed, it is easier for the electricity to flow to the Arduino and so a “HIGH” signal is read. This type of resistor is called a *pulldown resistor*.

The fun bit, of course, is to make the Arduino read the button and act accordingly. Delete the existing code blocks and build up some code to look like those shown here. What is happening is the Arduino reads the signal at digital pin 3 and compares it to what it wants it to be, “HIGH”. If the signal and the desired reading are equal, then it sets the LED to be “HIGH”, else (the programming term for “otherwise”) it sets the LED to be “LOW”.

Because your LED is connected to the same pin as the built-in LED, you can use the “set built-in LED to...” block. The “set pin *n* to...” block will work just as well—just change the *n* to 13. Almost every programming language has an “if...then...else” (sometimes just an “if...else”) statement. These are called *conditional statements* and allow you to alter the flow of a program.

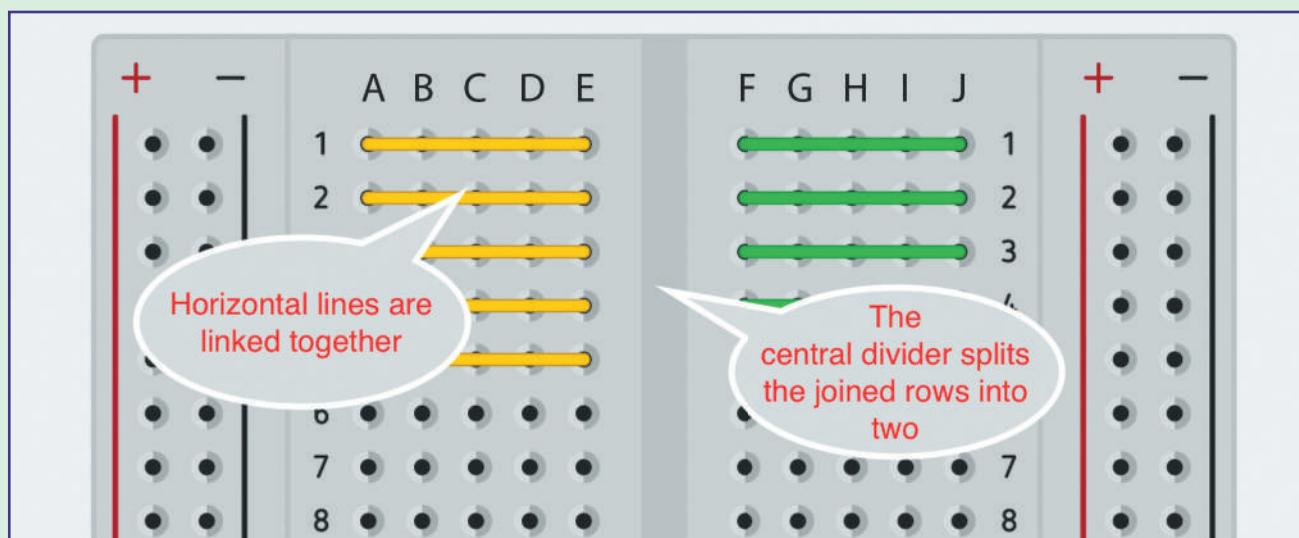
PRO TIP

Make sure you disconnect the multimeter from the button. The current that goes through the multimeter can make the Arduino “think” the button is being pressed!



Testing

You can (carefully) measure the circuit using the multimeter again. When testing, try pressing the button on and off and watch the LED carefully. It is always worth color-coding your wires to help with tracing them, it is normal for + (positive) wires to be red and - (ground) wires to be black. Signal wires can be a different color—yellow is quite commonly used.



Stretch tasks

- Pullup resistors work in a similar way. However, one terminal of the button is connected to ground and the signal terminal connects to the “read digital pin” and to a positive voltage via a high-resistance resistor. Build a pullup resistor and explore how they work.
- Research and consider situations where either a pullup or a pulldown resistor would be preferable. Being able to produce both is useful.
- Add a second button to turn a second LED on or off.
- Add a third LED, which lights up only when both buttons are pressed.

Final thoughts

Buttons and switches are two of the most common types of input device for microcontroller circuits. Spend some time walking around your home to find out how many switches and buttons there are. It is also interesting to see what other input devices and components you can find.

Part of solving an engineering problem is to use the right tool for the job; why do you think a TV remote has buttons rather than a touch screen? Often the reason is because buttons are cheap, robust and low power, all of which have a positive impact on the longevity of the device.



4. LED STROBES ON DEMAND

Setting the scene

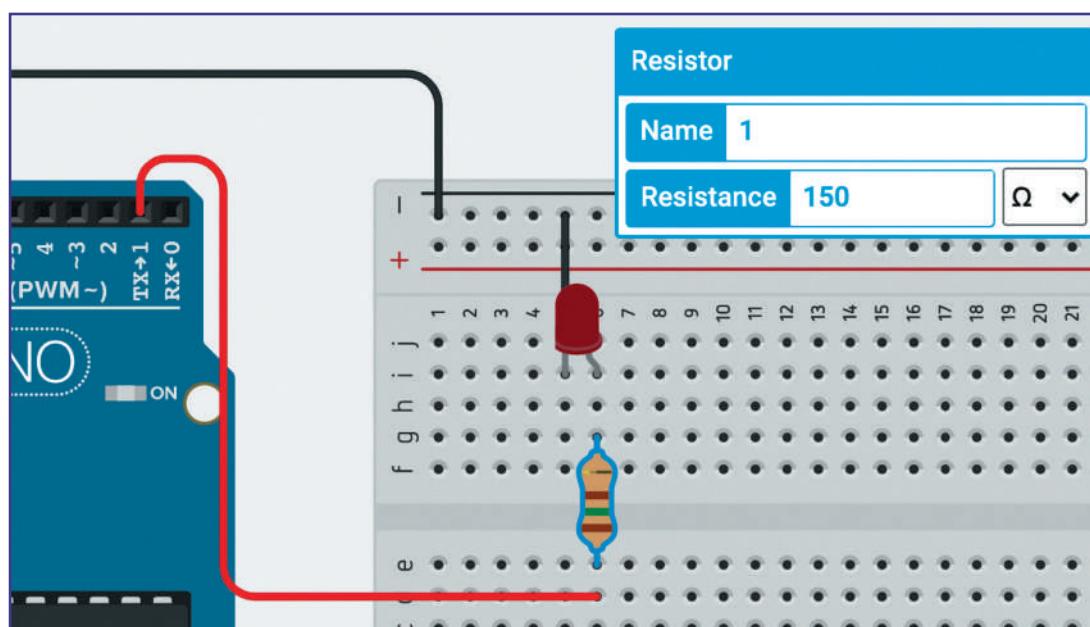
Learning to manipulate multiple outputs is a crucial skill. In this chapter you will be controlling no less than five different LEDs at once. You will notice that things start getting a bit confusing at this point—your code blocks get quite repetitive. Therefore, in this chapter, you will also take your first steps into the wonderful world of C and C++ programming and create some sub-routines. Problems that are too big to solve easily often end up getting solved poorly, which can lead to inefficiencies or even critical mistakes. Instead, we decompose the problem, or break it down into simpler steps, which can be solved and tested far easier.

Success criteria

- Connect five LEDs up to five digital pins.
- Test that all five LEDs can be turned on and off.
- Create sub-routines to strobe the LEDs left and right.
- Connect a second button to control which way the lights strobe.

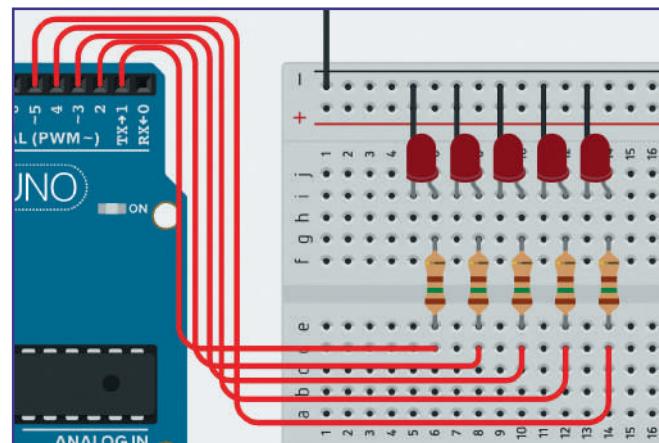
1

Connecting the LEDs is not really any more difficult than the connection you made in Chapter 2. Each LED has its cathode connected to ground via the negative rail, and its anode connected, via a $150\ \Omega$ resistor, to a digital pin. For simplicity, use digital pins 1 to 5.

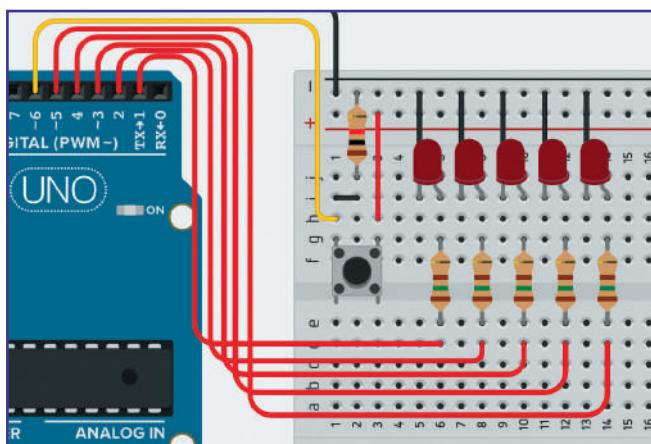


2

In Chapter 2, you learned how important it is to build and test circuits one stage at a time. The first test you need to do is check you have the LEDs wired up correctly. To do this, simply delete the existing code and replace it with five “set pin *n* to...” blocks, all set to HIGH and changing the pin numbers to 1 to 5. When you run the simulation, all five LEDs should light up.



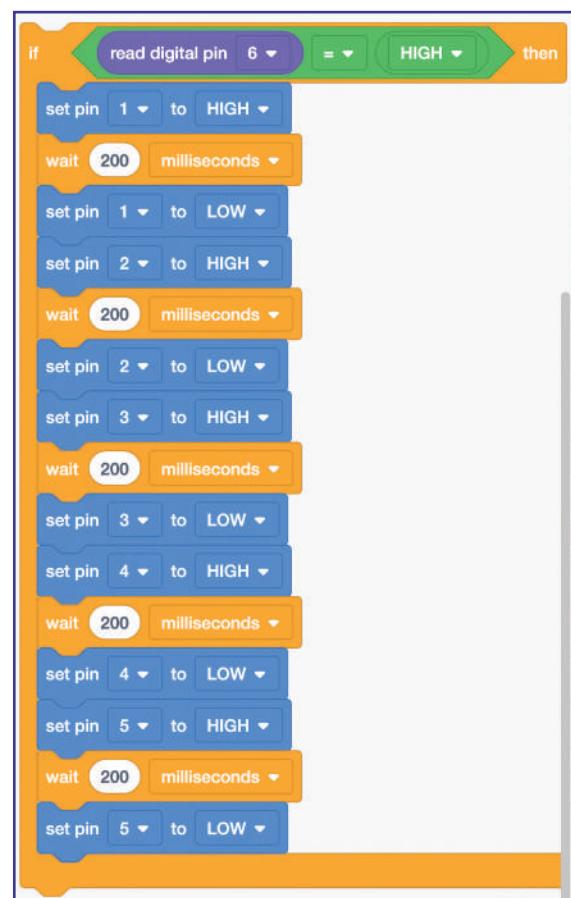
3



The aim of this project is to make the LEDs strobe when a button is pressed. To do this, add a pulldown button in the same way as you did in Chapter 3.

4

Now build your code. The code will be similar to the program in Chapter 3, with the difference here that after each LED is lit it needs to be turned off again. You also need to add some delays to create the strobe effect. A microprocessor is slow compared to a modern PC or laptop, but still works at a rate of about 20,000,000 instructions per second! The code shown here adds delays of 200 milliseconds (one-fifth of a second), so the whole “strobe” takes one second.



Now comes the tricky part. You need to add a second button and program the LEDs to strobe in different directions depending on which button is pressed. It is possible to do this using code blocks, but the blocks are going to become very long and unwieldy.

A more efficient method is to split your code into “chunks” and give each chunk of code a name such as **strobeLeft** and **strobeRight**. To do this, change to **Text** code and create two empty blocks of code for your strobing:

```
void strobeLeft() {  
}
```

and

```
void strobeRight() {  
}
```

PRO TIP

Get your code working in blocks first before you convert them to text. Then play around and explore. Don’t worry if things don’t work out; you can always “undo”.

```
void strobeRight()  
{  
    digitalWrite(1, HIGH);  
    delay(200); // Wait for 200 millisecond(s)  
    digitalWrite(1, LOW);  
    digitalWrite(2, HIGH);  
    delay(200); // Wait for 200 millisecond(s)  
    digitalWrite(2, LOW);  
    digitalWrite(3, HIGH);  
    delay(200); // Wait for 200 millisecond(s)  
    digitalWrite(3, LOW);  
    digitalWrite(4, HIGH);  
    delay(200); // Wait for 200 millisecond(s)  
    digitalWrite(4, LOW);  
    digitalWrite(5, HIGH);  
    delay(200); // Wait for 200 millisecond(s)  
    digitalWrite(5, LOW);  
}
```

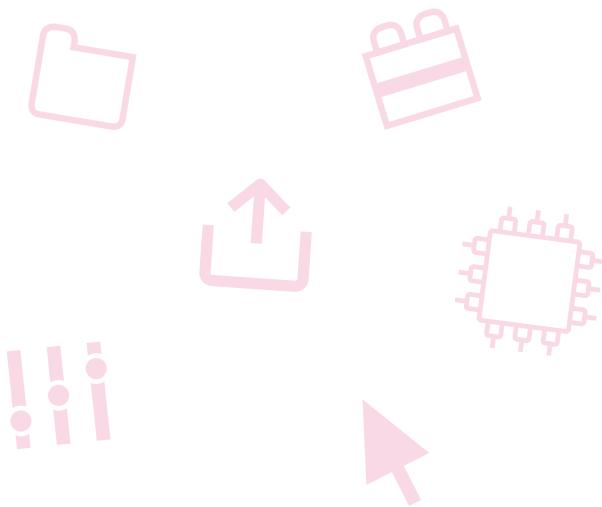
Note that the capitalization (and lack of spaces) in the names is very important!

“Void” just means the block of code (which is called a **function**) will not return an “answer”.

Now add the lines of code that strobe the LEDs. Cut and paste the **digitalWrite()** lines and the **delay()** lines from the **loop()** sub-routine into the **strobeRight()** sub-routine (between the **{braces}**).

PRO TIP

You will see that some of the **identifiers** (names of things in code) have a strange capitalization, such as **digitalWrite()**. As you can’t use spaces in names, it is common to start a name with lower case and then start each new word with a capital. This is called “camel case” and is one of the more commonly used syntax styles.



6

Copy and paste the same lines of code into **strobeLeft()** and adjust them as needed.

You could also make a function called **allOff()** to turn all the LEDs off.

```
void strobeLeft()
{
    digitalWrite(5, HIGH);
    delay(200); // Wait for 200 millisecond(s)
    digitalWrite(5, LOW);
    digitalWrite(4, HIGH);
    delay(200); // Wait for 200 millisecond(s)
    digitalWrite(4, LOW);
    digitalWrite(3, HIGH);
    delay(200); // Wait for 200 millisecond(s)
    digitalWrite(3, LOW);
    digitalWrite(2, HIGH);
    delay(200); // Wait for 200 millisecond(s)
    digitalWrite(2, LOW);
    digitalWrite(1, HIGH);
    delay(200); // Wait for 200 millisecond(s)
    digitalWrite(1, LOW);
}

void allOff()
{
    digitalWrite(1, LOW);
    digitalWrite(2, LOW);
    digitalWrite(3, LOW);
    digitalWrite(4, LOW);
    digitalWrite(5, LOW);
}
```

PRO TIP

Anything after the // is a comment—a note for the programmer, which is completely ignored by the computer.

7

Finally, you need to call the **strobeLeft()** and **strobeRight()** functions at the appropriate time. You can do this using an **if..else..if..else** statement and calling each function by name.

```
void loop()
{
    if (digitalRead(6) == HIGH) {
        strobeRight();
    } else {
        if (digitalRead(7) == HIGH) {
            strobeLeft();
        } else {
            allOff();
        }
    }
}
```

PRO TIP

If your code doesn't run, start with the line the error states the problem is on, and work upwards. Sometimes the error, such as a missing bracket, is just a few lines before!

Testing

Finding and removing errors from code is called *debugging*. This can be stressful, but only if you let it be! A good method is to write as few lines as possible before trying the code out. The fewer things you change, the fewer places an error can creep in. Therefore, start by creating an `empty()` function and test the code, then put some code inside the function and test it again... and again at the next stage.

Stretch tasks

- Add more buttons and more sub-routines to create a range of strobe patterns.
- Try to combine existing sub-routines rather than rewriting them from the beginning.
Create a `rightWave()` routine which strobos right three times.
- Research the Arduino `random()` function and the `if()` construct.
- Make a randomized routine play when the button is pressed.

Final thoughts

Decomposition and pattern recognition are two of the corner stones of computational thinking.

- When you split your code into sub-routines, you *decomposed* the problem, which means you converted it into simpler problems to solve. Computer scientists do this when they are designing a solution, rather than afterwards.
- Looking for patterns is another way to make code more efficient. For example, if you add a third button which strobos left then right, it would be more efficient to call the two existing sub-routines one after the other rather than to write a whole new section of code.

Sub-routines and sub-systems are a critical part of engineering design; whenever you look at something that has been made—a car, a computer, or a house—try to get into the habit of looking for the sub-systems that have been developed independently of the whole.

5. READING AND WRITING WITH THE SERIAL MONITOR

Setting the scene

This is the final chapter in which you will use an emulator. This chapter introduces you to two exciting new topics.

- Firstly, you will learn to input data from an analogue device. In the real world many devices are *analogue* (continuous) such as the temperature sensor in your refrigerator, the smoke detector in your smoke alarms.
- Secondly, you will learn to output data from the Arduino to the serial monitor, a text window on your computer.

Once you start working with real Arduinos, from the next chapter onwards, you will quickly learn how difficult it can be trying to figure out what is going on. Using the serial monitor allows you to output text back to your computer, which can help you work out what is happening.

Success criteria

- Connect + and – rails.
- Connect a potentiometer to the Arduino.
- Open the serial monitor.
- Read from the potentiometer.
- Write to the serial monitor.
- Read from the serial monitor.

1

The first thing you need to do is make sure you can output data on the serial monitor. You will be doing this entirely in C and C++ code, so make sure you open the code in **Text** format. You need to do two things to make the serial monitor work:

1. Tell the Arduino to begin communicating, and how fast. The speed is known as the *baud rate*. Set the speed to 9,600 in the **setup()** function. Note: the majority of the sub-routine you will see in this book will have void at the start. **void setup()** means it is a subroutine called “setup,” which does not receive any extra information (called “parameters”) inside the brackets, and does not give a result back (void means “nothing” comes back).
2. Tell the Arduino to output something to the serial monitor. Use the code **Serial.println("some text")** in the **loop()** function. **println()** is short for “print line”.

It is also a good idea to have a delay after outputting, otherwise the text will appear faster than you can read it! Open the **Serial Monitor** at the bottom of the screen before running the code.

The screenshot shows the Arduino IDE interface. The Text editor contains the following code:

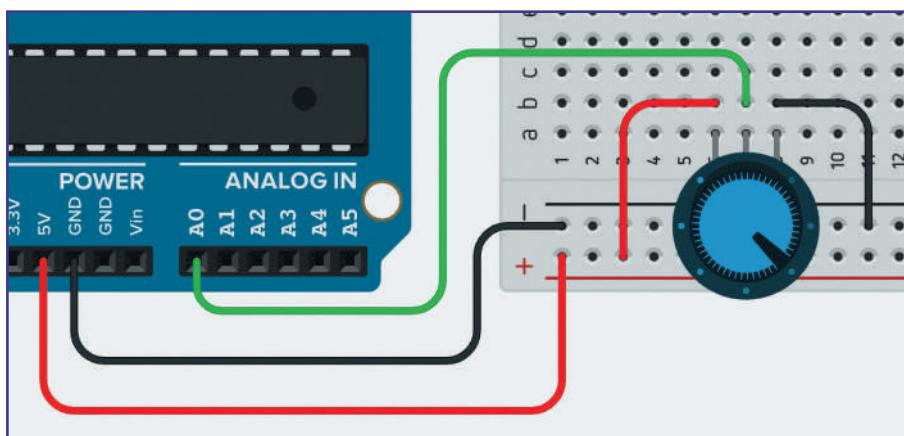
```
1 void setup()
2 {
3     Serial.begin(9600);
4 }
5
6
7 void loop()
8 {
9     Serial.println("hello");
10    delay(1000);
11 }
```

The Serial Monitor window below shows the text "hello" repeated seven times, indicating the output of the code.

2

So far, you have been reading *digital data*, which is data that can have only discrete values. Specifically, you have been reading buttons which are either pressed or not pressed. You are now going to read an analog device called a *potentiometer*. A potentiometer rotates so it can have an infinite number of positions. This is known as a *continuous input*. The Arduino cannot handle infinite positions—it has a range of 1,024 values and assigns the one that is closest to the potentiometer reading. Data which has a continuous range of values is analog, and so the potentiometer must be connected to an analogue pin: the opposite of this digital data which has discrete values (most often “on” or “off”).

Connect up your breadboard as in previous chapters and wire up a potentiometer with the left-hand pin connected to positive, the right-hand pin connected to negative, and the middle (signal) pin connected to one of the analog (A) pins on the Arduino.



PRO TIP

Make sure you connect the wires the right way round or you may get strange readings from the potentiometer.

3

Follow this process for reading an analog input:

1. Create a *variable* (a memory location to which you give a name) to store the value. This will be a whole number, which is called an *integer*.
2. Set the analog pin as an input in the `setup()` function.
3. In the main loop, use `analogRead()` to get the value of the input (from the potentiometer) and store it in the variable.
4. Act on the data read. In this case, use `Serial.println()` but, instead of outputting a string in quotation marks, output the contents of the variable.

```
1 int sensorValue = 0;
2 int incomingByte = 0;
3
4 void setup()
5 {
6   pinMode(A0, INPUT);
7   Serial.begin(9600);
8 }
9
10 void loop()
11 {
12   sensorValue = analogRead(A0);
13   Serial.println(sensorValue);
14   delay(100);
15 }
```

You can also use the serial monitor to read text from the keyboard and send it to the Arduino. To do this, follow these steps:

1. Declare an integer to hold the input data. The input data is read 1 byte (8 bits) at a time as a numerical code and so is stored as an integer.
2. Check if there is data available on the serial monitor.
3. If so, act on it. This action could be as simple as echoing the code back to the serial monitor, or it could be much more complex.

Testing

If you see numbers instead of text on the serial monitor, you may have used single quotation marks instead of double. Move the dial on the potentiometer to vary the reading. If you implement an echo in Step 4 but the text zips past too fast, try increasing your delay to 200 ms or more.

Stretch tasks

- Add a second potentiometer and echo the readings to the serial monitor.
- Swap a potentiometer for a temperature sensor and test your circuit. How would you identify the temperature with a real temperature sensor?
- Add a test so that: if you type “a” into the serial monitor, the reading of the first potentiometer is shown; if you type “b”, the reading of the second potentiometer is shown; if nothing is typed then nothing is shown.
- Amend the previous stretch task to toggle between the two potentiometers by having a button that is pressed to toggle between them.

Final thoughts

Many sensors you will come across in the Internet of Things (IoT) are analog, and most are read in a very similar manner to the potentiometer. This project used the potentiometer because it's easy to vary the reading for testing. Sensors are one of the key things that make the IoT so powerful. Being able to sense (read) the condition of something means you can report or act upon that reading. Consider which conditions you could sense around your home and how you could act upon those readings to improve your quality of life or save electricity.

```
void loop()
{
    sensorValue = analogRead(A0);
    Serial.println(sensorValue);
    if (Serial.available() > 0) {
        // read the incoming byte:
        incomingByte = Serial.read();

        // say what you got:
        Serial.print("I received: ");
        Serial.println(incomingByte);

    }
    delay(100);
}
```

PRO TIP

The output numbers are ASCII codes. You can easily find a list of ASCII codes online. Be aware that the codes for upper case and lower case letters (and numeric digits) differ. For example, if you want to check if the user enters “A”, you need

`if (incomingByte == 65)` but if you want to check if they enter “a”, you need
`if (incomingByte == 97)`

6. INTRODUCTION TO THE ARDUINO

Setting the scene

In this project you will start exploring the possibilities that an Arduino gives you to explore the world of the Internet of Things (IoT). You will do this by looking at what is included on the Arduino and how you can connect the Arduino to a computer. For the first time, you will be using a real piece of hardware. Many developers use emulators such as Tinkercad to develop their solutions, but it is essential to then test these thoroughly on real hardware, as sometimes things work differently in real life. While this project is in itself very simple, it will give you the skills needed to progress with the future projects.

Success criteria

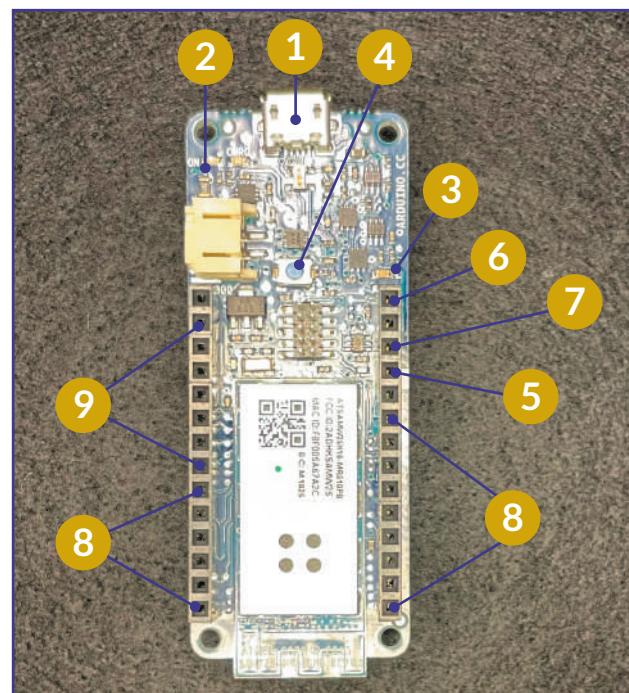
- Identify the key parts of an Arduino.
- Connect a shield to an Arduino.
- Connect an Arduino to a computer.
- Sign into Arduino Cloud.
- Load a sketch to an Arduino.

1

Arduinos vary by model, so it's a good idea to get to know the essential parts of your Arduino before you start plugging components into it. At best, if you get things wrong, your project won't work. At worst, you could damage components or your Arduino itself.

The essential parts you need to identify are:

- 1 **USB socket:** where you connect your Arduino to your computer
- 2 **power LED:** showing that the Arduino is plugged in and receiving power
- 3 **built-in LED:** useful for testing and indicating that things are happening
- 4 **reset button:** quicker than disconnecting and reconnecting your USB cable
- 5 **ground pins:** all circuits need to return to ground
- 6 **5 V and 7 3.3 V pins:** for providing power to your circuits
- 8 **digital and 9 analog pins:** for controlling components or reading from sensors

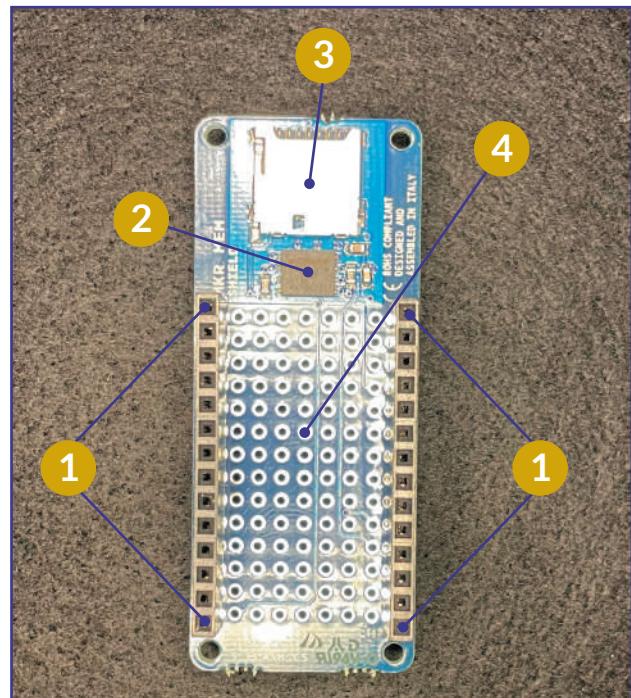


2

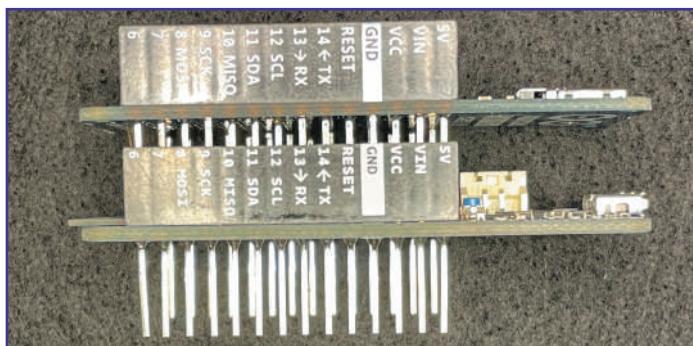
There is also a wide range of shields that you can use with your Arduino. Shields expand the functionality or connectivity of the Arduino. You can often build circuits on a shield before connecting it to the Arduino. Each type of shield has different parts.

The MemShield has these parts:

- ① pins that mirror those of the Arduino
- ② a memory chip
- ③ an SD card slot
- ④ a prototyping area where you can build circuits



3



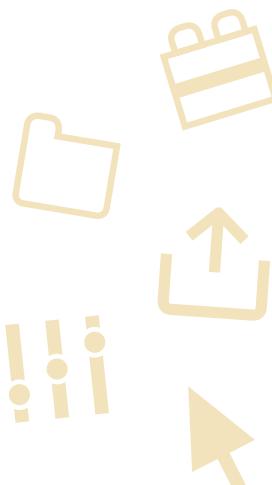
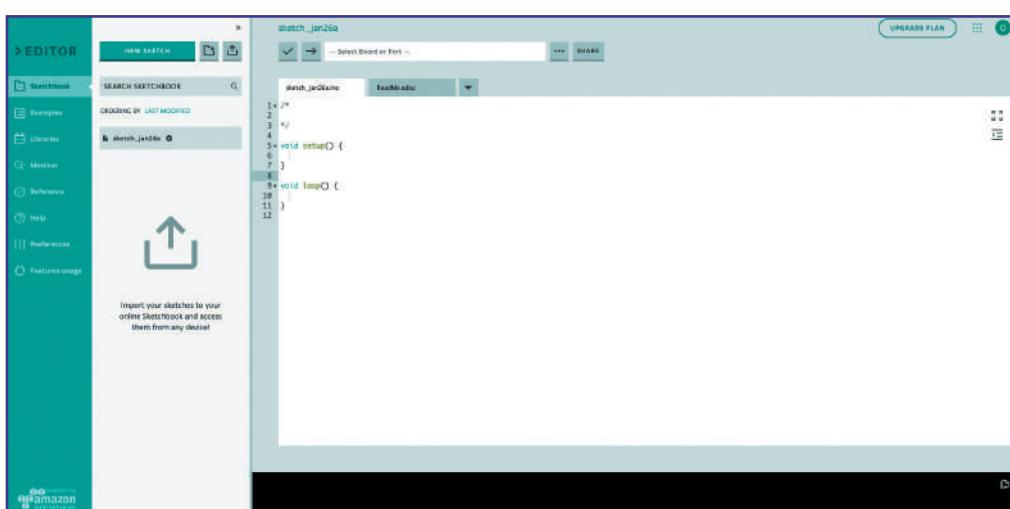
It is really important to plug the shield into the Arduino slowly and carefully. The pins are easily bent and can even snap off. It is just as important to unplug the shield carefully, as the shield has a tendency to twist as it comes out, which can bend the pins out of line.

4

When you are programming Arduinos, there are several applications that you can use.

For the projects in this book, you will be using the web-based editor available at

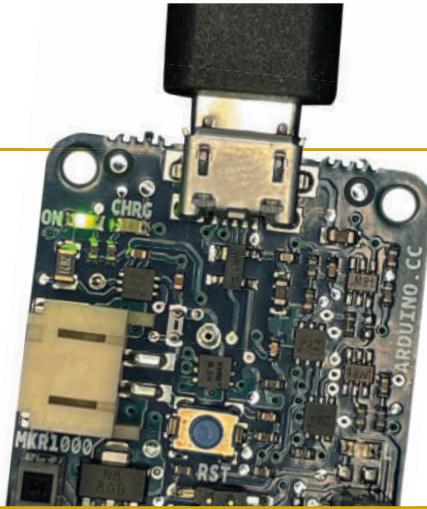
<https://create.arduino.cc/>. You will need to install the agent so that your web browser can connect to the Arduino.



5

First, check that the Arduino and USB cable are working.

When you plug one end of the USB cable into your computer and the other into the Arduino, the green power LED on the Arduino should light up. The built-in LED may or may not light up or flash depending on whether a test program was pre-loaded.



6

Next, ensure that the computer can connect to and program the Arduino. You should be able to select your Arduino from the drop-down menu at the top of the web editor. You can then choose **>EDITOR** from the left-hand side of the window, select **Examples** and then under **01.BASICS** select **Blink**. This will load the Blink code—called a sketch—into the editor. If the code looks familiar, that is because it is similar to the first program you emulated in Chapter 1!

SEARCH EXAMPLES

SHOWING EXAMPLES FOR MKR WIFI 1010

BUILT IN FROM LIBRARIES

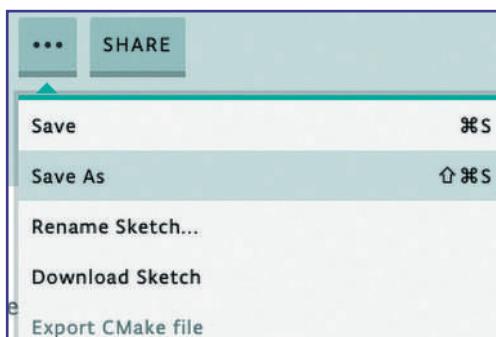
01.BASICS (6)

- AnalogReadSerial
- BareMinimum
- Blink**
- DigitalReadSerial
- Fade
- ReadAnalogVoltage

02.DIGITAL (9)

7

You now have two options. When you are typing in your own programs you can click on the **Verify** button (the check mark) to ensure your code will compile correctly. Once you are confident that your code will compile correctly, you can click on the **Upload** button (the arrow) to compile the code and transfer the program on to the Arduino. If this doesn't work, make sure you read the output carefully!



Success: Saved on your online Sketchbook and done uploading Blink

```
readWord(addr=0xe000ed00)=0x410cc601  
readWord(addr=0x41002018)=0x10010305  
writeWord(addr=0xe000ed0c,value=0x5fa0004)
```

PRO TIP

If you want to read (and experiment with) the code, you will need to click the ellipsis (three dots) and save a copy of the code in your own area.

Hopefully this project will work out fine. (It should if you haven't changed any code!) Once the compilation and transmission are complete, the build-in LED on the Arduino should start flashing.



PRO TIP

As long as you don't change anything, there is no need to verify the Blink code. However, when you write your own sketches, verifying the code will help you track down problems and save you some time.

Testing

As this is only an introduction, not a lot should go wrong. If your Arduino does not show up in the window or the sketch fails to upload, check your USB cable connection. Try changing the cable, the USB socket, the adaptor if you use one, and ultimately the computer. If none of these changes fixes the problem, you may have a faulty Arduino.

Stretch tasks

This chapter was simply an introduction to the Arduino: what an Arduino looks like and how to connect it.

- In Chapters 1 to 4 you used an emulator to create some simple but useful circuits. Re-create those circuits and code using your real Arduino, to get used to using physical devices.
- Spend some time looking at the different Arduinos available at <https://www.okdo.com/c/arduino-shop/arduino/>. Identify the different features of each model. Create a comparison table.
- Spend some time looking at the different shields available at <https://www.okdo.com/c/arduino-shop/arduino/>. Create a comparison table for them. Identify the projects that you could complete with the different types of shield and add them to your table. Try ideating some ideas—you may be able to use your ideas in future projects.
- Run your Arduino on a battery without being plugged in to your computer. You may need to research how to "unblock" your serial port monitor, as the Arduino won't be able to connect to the computer.

Final thoughts

Arduinos are powerful, yet tiny enough to fit almost anywhere. Most can run on batteries, and you have already seen some of the shields that can be used to extend their functionality. Think about some of the problems you could solve at home, at school, or in your community by linking an Arduino project to the Internet.

7. THE IOT CLOUD

Setting the scene

This chapter introduces the use of the Cloud. The Cloud forms an interface between us (using a web browser) and the hardware (the IoT) which enables us to control the hardware and react to sensor readings from anywhere. You will start by programming a simple button to turn on the LED.

The “Cloud” is one of the most exciting developments of recent years, with Internet-enabled heating systems, error-reporting fridges and even IoT coffee machines all helping everyday people to reduce their environmental impact; we are now able to control devices regardless of where we are and only when we need them, rather than relying on a “best fit” timer.

Success criteria

- Connect an external LED to the Arduino and ensure it works.
- Create a sketch that listens for a button press from the IoT Cloud and turns the LED on or off.
- Create a button in the IoT Cloud.
- Link the Cloud button to the sketch.

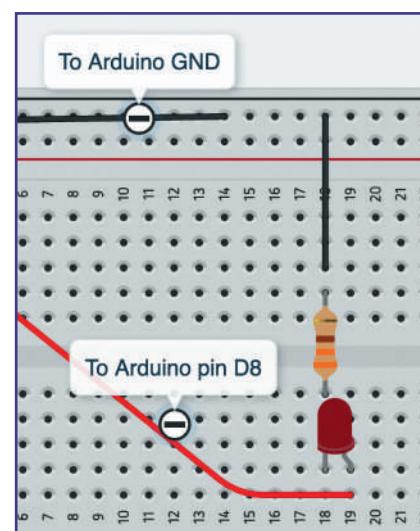
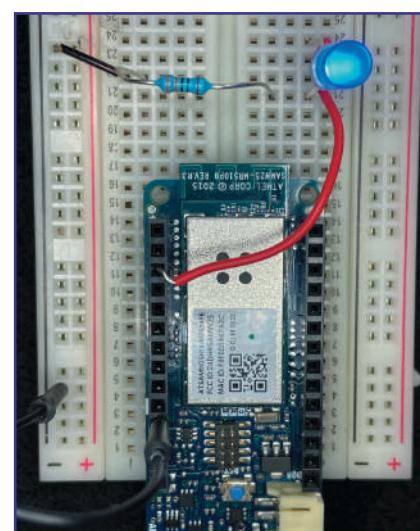
1

In this chapter you will build your first simple Cloud-based device. There are three main parts to this:

- a circuit, which will be controlled
- a piece of code on the Arduino, which will control the circuit and communicate with a web server
- a web-based dashboard, which will communicate via the web server with the Arduino code

2

The circuit you are going to create is a simple one. Connect an LED between the Arduino pin D8 and ground, via a resistor. The example shown here uses a $330\ \Omega$ resistor, but any resistor between $220\ \Omega$ and $470\ \Omega$ should be fine.



3

It is always a good idea to decompose problems and test each section separately. In this case, it is worth testing the circuit before you try to control it via the web.

The [TheIoTCloud_Program1](#) sketch on the Arm GitHub repository will test the LED.

```
/*
Test program to ensure external LED is working.
*/
int LED1 = 8;

// the setup function runs once when you press reset or power the board
void setup() {
    // initialize specified pin as an output.
    pinMode(LED1, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {
    digitalWrite(LED1, HIGH); // turn the LED on (HIGH is the voltage level)
    delay(1000); // wait for a second
    digitalWrite(LED1, LOW); // turn the LED off by making the voltage LOW
    delay(1000); // wait for a second
}
```

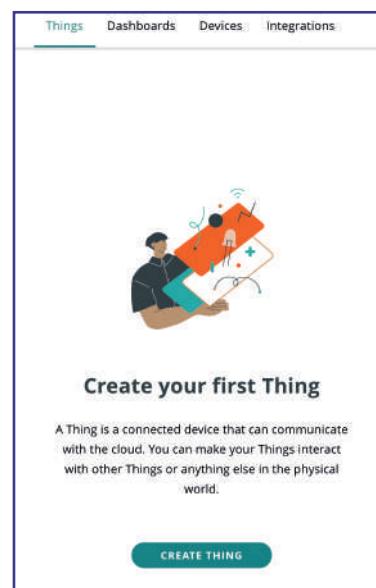
4

The first real difference to your work in Chapter 6 is that, instead of going to Arduino Web Editor, you will need to go to IoT Cloud. This allows you to create a *thing* (remember that IoT stands for Internet of Things).

5

To make Thing work you will need to do a bit of setup, which involves:

- Adding a device (your Arduino), so the Thing knows which device will control the circuit.
- Configuring network settings to enable the Arduino to connect to your WiFi network. The network settings will need to be the same SSID and password for wherever the device is going to be situated.
- Adding a variable, which enables Things to happen. The dashboard you are going to create will change that variable, and the Arduino code will monitor the variable and act accordingly.



Untitled

Setup Sketch Serial Monitor

Variables

Variables are what you can monitor or control to make your Thing function. For example a temperature or a smart lamp. Once created, you can use them in your Sketch.

ADD VARIABLE

Device

Select the device you want to use or configure a new one.

Select Device

Network

Enter your network credentials to connect your device.

Configure

PRO TIP

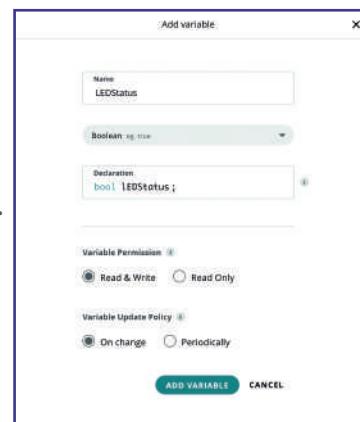
If your device isn't found, check all the connections and ensure the test program uploads.

PRO TIP

If your device is showing as disconnected, come back and check this step. Both the SSID and password are case sensitive and must be exact!

6

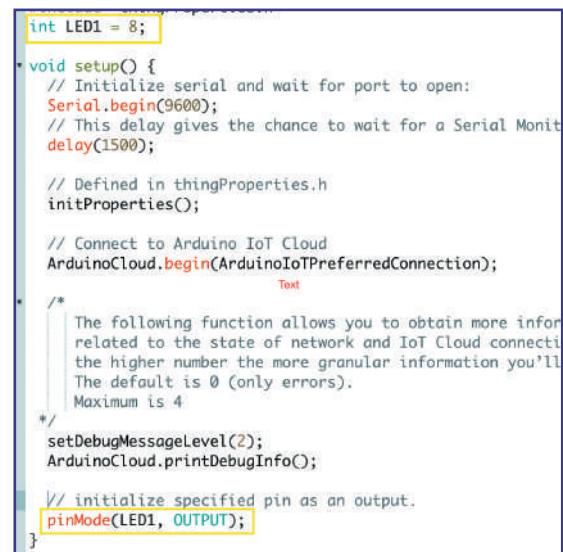
The circuit in this project will light (or not light) an LED. The LED being lit will be a True or False state, called a *Boolean*. Create your variable and give it a sensible name (it is called **LEDStatus** in this example), set it to a Boolean, enable it to be both **Read & Write** and set it to update **On change**. Note that web editor will change some declarations to use a lower-case letter at the start. When this happens it is important that you use the same case when referring to it.



7

Once you have completed the setup, enter the sketch section of the thing. Happily, a lot of the code has been automatically generated for you! Use the same settings as in the test program.

- Create an integer variable to store the LED pin.
- Set the pin mode to output.



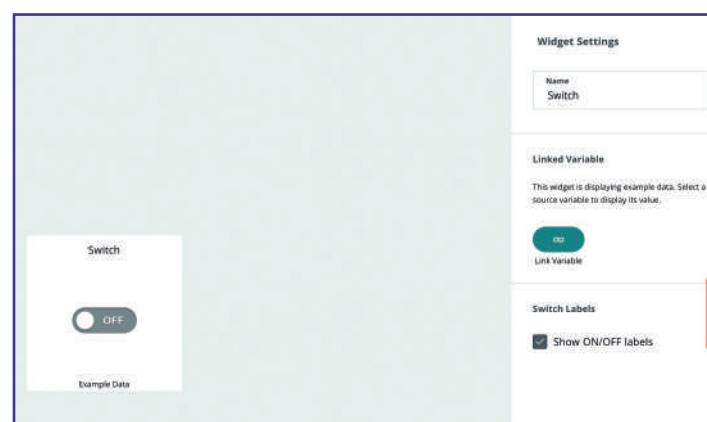
8

```
void onLEDStatusChange() {  
    if (LEDStatus == true) {  
        digitalWrite(LED1, HIGH);  
    } else {  
        digitalWrite(LED1, LOW);  
    }  
}
```

A function called **onLEDStatusChange()** will have been automatically created. This is called whenever the **LEDStatus** variable is changed by the web server. The LED can be turned on or off depending on the value of the variable. You can find the final code, [TheIoTCloud_Program2](#) on the Arm GitHub repository. A bit of extra code has been included for testing.

9

Once the code is all set up, you can develop the dashboard. While this sounds complicated, it is just a case of following a wizard to create a web page with your switch on **itClick** on the **Dashboard** link and add a widget—the LED is a Boolean, which can be either off or on, so it makes sense to choose a **switch** widget. Then link the switch to the variable **LEDStatus** that you created in Step 8.

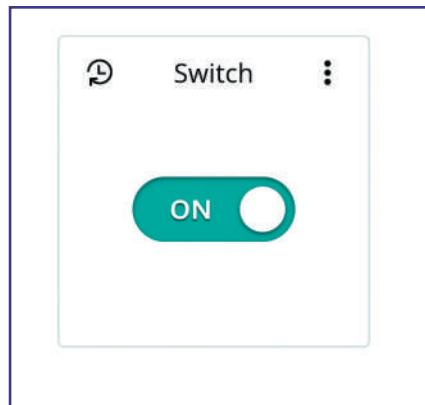


10

Finally, transfer the completed sketch to the Arduino. To do this, go back to **Things** and, just like in Arduino Cloud, click on the **Upload** arrow. If all is well, you will see the usual blinking transfer lights and confirmation text.

```
1 // Sketch generated by the Arduino IoT Cloud Thing "Untitled"  
2 https://create.arduino.cc/cloud/things/52776341-1635-48d8-9be1-ce2e9acb5795  
3  
4 Arduino IoT Cloud Variables description  
5  
6 The following variables are automatically generated and updated when changes are made to the  
7  
8 bool LEDStatus;  
9  
10 Variables which are marked as READ/WRITE in the Cloud Thing will also have functions  
11 which are called when their values are changed from the Dashboard.  
12
```

11



Of course, the fun part is to test the whole thing out. Once your sketch has uploaded, go back to the dashboard and click the switch. The LED on the Arduino should mirror the status of the switch.

Testing

Flick the web-based switch to test that the LED turns on and off. Ask someone to log into a different WiFi network (or use a smartphone on cellular data) to prove that the connection is going across the Internet, not via the serial cable.

Stretch tasks

- Add a second LED that turns off when the main LED is on, and vice versa.
- Read the **TheIoTCloud_Program2** code on the GitHub repository to find out how to output serial messages. Include a message to state which LED is turning on or off.
- Keep a count of how many times the LED is turned on and output this count to the serial monitor.

Final thoughts

Wasted electricity is a worldwide problem. In 2018 it was estimated that £4.4 billion was wasted in Britain alone through lights being left on. The example in this project could simulate the lighting in your home, with each LED being a lightbulb in a room. Consider which other electrical devices you could turn on and off remotely using a website accessed from your mobile phone. You could decompose all of the connectable systems in your home and plan a series of sub-systems. How else could you use this technology to improve your lifestyle, while protecting the environment? This project has been one-directional with the Cloud service 'pushing' to the Arduino. In the next chapter you will look at communication in the opposite direction.

8. IOT CUSTOMER SATISFACTION RATINGS

Setting the scene

One of the major limitations of Arduinos is that they only have a finite number of pins. If you want to have a lot of inputs or outputs, you can very quickly run out of pins. In this chapter you will learn how to reduce the number of pins used by inputs by daisy-chaining several buttons together. You are going to set up an IoT customer satisfaction rating system, with a button for each rating option. You might have seen these in various public buildings—often rating the cleanliness for example.

Success criteria

- Connect a single button using a pulldown resistor.
- Connect additional buttons using a daisy chain of resistors.
- Read which button has been pressed.
- Codify which button is being pressed.
- Connect to the IoT server.
- Store data on the IoT server.

1

This project uses the power of resistors to reduce voltage flow, and the ability of the Arduino's analog pins to read a flow as a value between 0 and 1,023. You will use a simple analog reader code (similar to that used in Chapter 5) to echo the reading to the serial monitor.

```
int reading = 0;

void setup()
{
    pinMode(A0, INPUT);
    Serial.begin(9600);
}

void loop()
{
    reading = analogRead(A0);
    Serial.println(reading);
    delay(100);
}
```

2

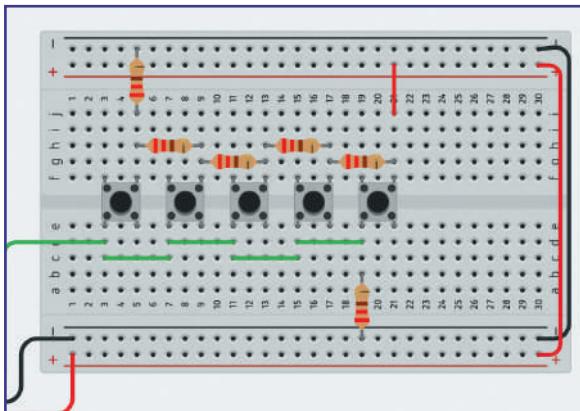
Put several buttons on the breadboard (daisy-chaining).

- Connect the left-hand legs together with short jumper wires.
- Connect the right-hand legs together with $220\ \Omega$ resistors.
- Connect the left-hand leg of the first resistor to the analog pin.
- Connect the right-hand leg of the first resistor with another $220\ \Omega$ resistor to ground.

PRO TIP

The exact value of the resistors isn't too important. $220\ \Omega$ resistors are commonly used with LEDs, so most makers often have lots of them!

- Connect the left-hand leg of the final button with yet another $220\ \Omega$ resistor to ground.
- Connect the 5 V wire to the right-hand leg of the final button.

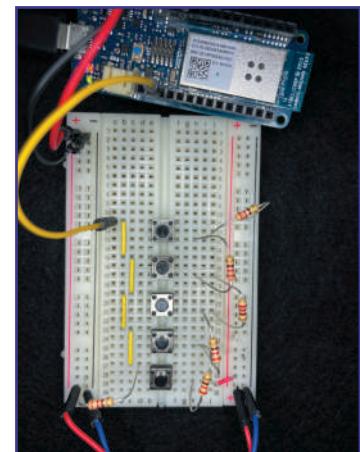


PRO TIP

Set the delay quite high so a single push is not recorded several times.

3

This type of circuit is called a *voltage divider* because the voltage of 5 V is split between the signal wire (the analog pin) and ground. When no buttons are pressed, the power flows through all the linked resistors to ground and nothing gets to the signal (which is pulled to ground by the final resistor you put in). When you press a button, the further along the row that button is, the more resistors the voltage has to flow through and so the more it is reduced. Write down the values of each resistor when you press each button.



4

Of course, ratings aren't much use if you can't see what people have rated you. To make the device Internet-enabled, start by creating a new thing on Arduino Cloud. You will communicate using a *string* variable. Every time you press a button, you add a new Rating on to the end of the string. You will also need a dashboard with a "value" widget linked to the variable.

PRO TIP

If your Arduino shows as "disconnected" in the dashboard, double check the WiFi settings and try refreshing everything.

Value
Good | Good | OK | Bad | Awful |
Awesome | Good | Good | OK |

The code in the loop needs to check which button has been pressed. To do this, use an ***if..else..if..else*** construct like the one you used in Chapter 4. However, due to variations in temperature and outside influences, you won't always get a nice steady number from the real buttons in the same way as when using the emulator. Therefore, it is important to test a range of values. Try testing about 30 either side of the values you wrote down in Step 4, just to be sure.

```
void loop() {
    ArduinoCloud.update();
    // Your code here
    reading = analogRead(A1);
    Serial.println(reading);
    if (reading > 150 && reading < 200) {
        customerRating = customerRating + "Awesome | ";
    } else if (reading > 230 && reading < 330) {
        customerRating = customerRating + "Good | ";
    } else if (reading > 400 && reading < 460) {
        customerRating = customerRating + "OK | ";
    } else if (reading > 660 && reading < 720) {
        customerRating = customerRating + "Bad | ";
    } else if (reading > 1000) {
        customerRating = customerRating + "Awful | ";
    }
    delay(200);
}
```

Testing

The breadboard is getting quite busy with this project, so if things aren't working as expected, make sure everything is in exactly the right place. It's very easy to plug a resistor into the wrong column and then nothing works! Test your device with short and long presses or several presses at once. Adjust the timings or delays to make sure everything works as well as possible. Keep an eye on the serial monitor to see what is happening behind the scenes.

Stretch tasks

- Can you edit the code to enable you to reset the ratings list?
- Explore how many buttons you can add. (There are rumors of up to 50!)
- Can you add a button that sends the choice to the dashboard after the choice has been selected?
- Can you add indicator LEDs to show which button has been pressed?
- Sometimes you might accidentally double press a button. Explore “debouncing” to prevent this.
- In the testing section you ensured that long presses of the button only count as a single item. Adjust this so that for presses of longer than one second, each second the button is held adds another item of the same type (so holding the “OK” button for three seconds will add three “OK” ratings).

Final thoughts

There are a number of ways to expand the number of inputs and outputs available to your Arduino. However, many involve additional circuits such as shift registers or multiplexers. Using a voltage divider is not only a useful skill but saves the financial and environmental cost of additional hardware.

9. AIR QUALITY SENSOR

Setting the scene

In previous chapters you have read data, both digital (on or off) and analog (continuous), from devices such as buttons and potentiometers. With enough knowledge, time, and perseverance, you can build complex devices using combinations of simple input devices and sensors. More commonly, however, you will find a sensor device that has already been created for your purpose, so you just need to apply your experience of reading values from a pin and acting on those values.

In this chapter, you will use an air quality sensor, known as an MQ-135, to read the air quality around you and report it to the serial monitor.

Success criteria

- Connect the Arduino to the MQ-135.
- Install the MQ-135 library.
- Initialize the sensor.
- Calibrate the sensor.
- Read and output the current air quality reading.

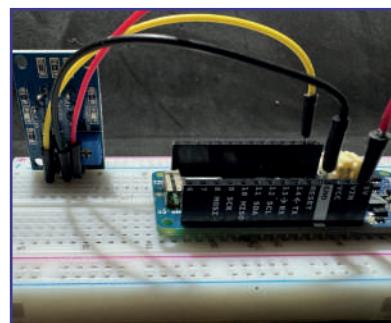
1

MQ-135s are inexpensive sensors made by several manufacturers. Always make sure to check the pins are in the expected places. Ours run (from left to right) A0 (analog output), D0 (digital output), ground, VCC. Always connect to the correct pins by their labels, not our order!



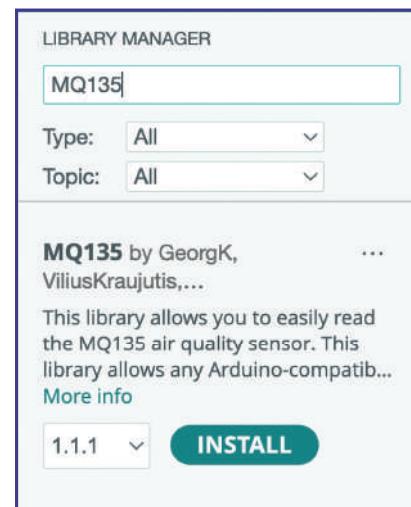
2

We've used a slightly different Arduino here—this is a MKR1010 WiFi. However, it has the same connections as most others. Simply connect the 5 V wire on the Arduino to VCC on the sensor (we used a red wire), ground to ground (we used a black wire) and A0 on the sensor to A0 on the Arduino (we used a yellow wire). You could use male–female wires between the Arduino ports and the sensor, or female–female wires between the Arduino pins and the sensor pins if you prefer.



3

Libraries are often available to help you use additional hardware or software without having to write the low-level code yourself. If you go to **Tools** and **Manage Libraries** you can search for **MQ135** and **install** the library for this sensor.



4

The MQ-135 needs to be calibrated, that is, taught what is normal. It is suggested that this is done outside for at least 24 hours—which is pretty risky (not many people want to leave their laptop plugged in outside for that long!). We'd recommend you find a quiet room where you can leave it with the window open. We have provided the code **MQ135_calibrate.ino** which you can run—write down the final value output in the serial monitor window before running the next part of the project.

A screenshot of the Arduino IDE. The top half shows the code for "MQ135_calibrate.ino":

```
1 #include "MQ135.h" // MQ-135 library
2 const int ANALOGPIN = 0; // Pin to read the MQ-135
3 MQ135 gasSensor = MQ135(ANALOGPIN); // Initialise the MQ-135
4 int count = 0; // Count the number of reads
5 float total = 0; // Total up the read values
6
7 void setup() {
8     // put your setup code here, to run once:
9     Serial.begin(9600); // Sets the serial port to 9600
10 }
11
12 void loop() {
13     float rzero = gasSensor.getRZero(); // Read the RZero value
14     total = total + rzero; // Total it up
15     count = count + 1; // Increment the count
16     float avg = total / count; // Calculate the average
17     Serial.println(avg); // Output the average
18     delay(1000); // Wait 1s
19 }
20
```

The bottom half shows the "Serial Monitor" window with the message "Message (Enter to send message to 'Arduino MKR WiFi 1010' on '/dev/cu.usbmodem1101')". It displays the following output:

```
199.21
199.21
199.21
199.21
199.21
199.21
```

5

Let's look through the monitoring code, **MQ135_monitor.ino**. First, we include the library, define which pin we will be reading the data from, and then define the **RZERO** value. This is the calibration value that we got in Step 4.

We then have to initialize the gas sensor by calling an *object* (MQ135) and giving it some starting data—the pin number and the **RZERO** value. This first call to create the object is called a *constructor*.

Everything else is pretty simple—after starting the serial monitor we can call a method (a piece of code belonging to the object) to get the ppm—or parts per million—that the sensor is reading.

```
MQ135_monitor.ino
1 #include "MQ135.h"                                // MQ-135 library
2 const int ANALOGPIN = 0;                          // Pin to read the MQ-135
3 const float RZERO = 200.24;                        // Define the calibration setting
4
5 MQ135 gasSensor = MQ135(ANALOGPIN, RZERO); // Initialise the MQ-135
6
7 void setup() {
8     Serial.begin(9600);                            // Sets the serial port to 9600
9 }
10
11 void loop() {
12     float ppm = gasSensor.getPPM();               // Read the parts per million
13     Serial.println(ppm);                          // Output the parts per million
14     delay(1000);                                // Wait 1s
15 }
```

PRO TIP

If the first message doesn't show, this is because the serial monitor isn't ready yet. Some sketches use `while (!Serial)` to wait for the serial monitor to initialize, but this will block the whole program if the device has no serial monitor (for example, if it is running off a battery and not plugged in to a computer), so a 10-second delay before outputting anything may be a better choice!

6

We now have what feels like a fairly pointless number being output every second. In fact, *ppm* stands for parts per million and indicates how much unhealthy gas (primarily carbon dioxide) there is in the air around the sensor.

400 ppm: The average level of CO₂ in outdoor air—this is what the sensor will be checked and calibrated against.

400–1,000 ppm: A typical level for occupied spaces with good air exchange.

1,000–2,000 ppm: A level that can cause complaints of poor air and drowsiness.

2,000–5,000 ppm: A level that can cause headaches, sleepiness, and stuffy air.

```
MQ135_monitor_with_messages.ino
1 #include "MQ135.h"                                // MQ-135 library
2 const int ANALOGPIN = 0;                          // Pin to read the MQ-135
3 const float RZERO = 201.96;                        // Define the calibration setting
4 const int AWESOME = 400;                          // This is better than clean outdoors!
5 const int GOOD = 1000;                            // The limit between good and bad
6 const int BAD = 2000;                            // The limit between bad and awful
7
8 MQ135 gasSensor = MQ135(ANALOGPIN, RZERO); // Initialise the MQ-135
9
10 void setup() {
11     Serial.begin(9600);                            // Sets the serial port to 9600
12 }
13
14 void loop() {
15     float ppm = gasSensor.getPPM();               // Read the parts per million
16     Serial.println(ppm);                          // Output the parts per million
17     if (ppm < AWESOME) {
18         Serial.println("Awesome air");
19     } else if (ppm < GOOD) {
20         Serial.println("Good air");
21     } else if (ppm > BAD) {
22         Serial.println("Bad air - open a window!");
23     } else {
24         Serial.println("Awful air - get out now!");
25     }
26     delay(1000);                                // Wait 1s
27 }
```

Serial Monitor X Output
Message (Enter to send message to 'Arduino MKR WiFi 1010' on '/dev/cu.usbmodem1101')
Good air
537.58

PRO TIP

If nothing is changing, your air quality must be super stable (and hopefully super good)! Try breathing on the sensor for a while to increase the carbon dioxide levels. Ours was good (really good—we have a breezy kitchen) but gentle breathing got it to bad, and by doing an impression of a steam train over the sensor we got it to awful.

Now we have this information, we can do a sequence of checks and output messages to the user. We have put this file in [MQ135_monitor_with_messages.ino](#) for you—see how we have run checks against the boundary values? By doing the checks in sequence, we only need to check one boundary at a time.

Testing

It's quite hard to test the values thoroughly, but it can be interesting to leave the sensor somewhere where the air quality gets quite bad—if you use a laptop, you could put it near the back of a car. Just be careful not to breathe any polluted air yourself: make sure you step away from the vehicle before the engine is started. This project really does show the limitations of tethered devices (devices which are connected to a computer). You will learn how to de-tether your device from the computer in the next chapter.

Stretch tasks

- Most devices are not linked to a computer that can display serial output—it would be more useful to use a visual indicator at this point. Add LEDs that show what is happening. You could use green for clean air, orange for low pollution, and red for high pollution.
- Try making the LEDs flash when pollution is very high and then run the device from a USB battery pack without being connected to a computer.
- Explore different locations around your home that may have differing air quality. Record the values in your bedroom, the kitchen, the lounge, and differing parts of your garden and compare the results.
- Push the readings to the Arduino Cloud IoT dashboard and use the built-in widgets to display as a graph.

Final thoughts

There are many sensors available. There is a company called [Grove](#) that creates sensors, along with shields to make connection easier. Unfortunately, their air quality sensor is one which does not work with newer Arduinos such as the MKR range! Why not explore the Grove website and identify some of the other sensors that are available? Which sensors could you combine to make a system that would benefit your local environment? Consider whether that system would need to be IoT-enabled or whether it could provide all the necessary information without linking to the Internet using lights or sounds. Air pollution in the U.K. alone is believed to contribute to 40,000 premature deaths a year, so if you discover any interesting results do consider sharing your findings to raise awareness.

10. STORING IOT DATA IN A SPREADSHEET

Setting the scene

In Chapter 9, you used an air quality sensor to collect air quality data and, if you completed the Stretch task, published it to the Arduino Cloud as a graph. Sometimes, however, you might just want to log data in a spreadsheet so you can do your own analysis.

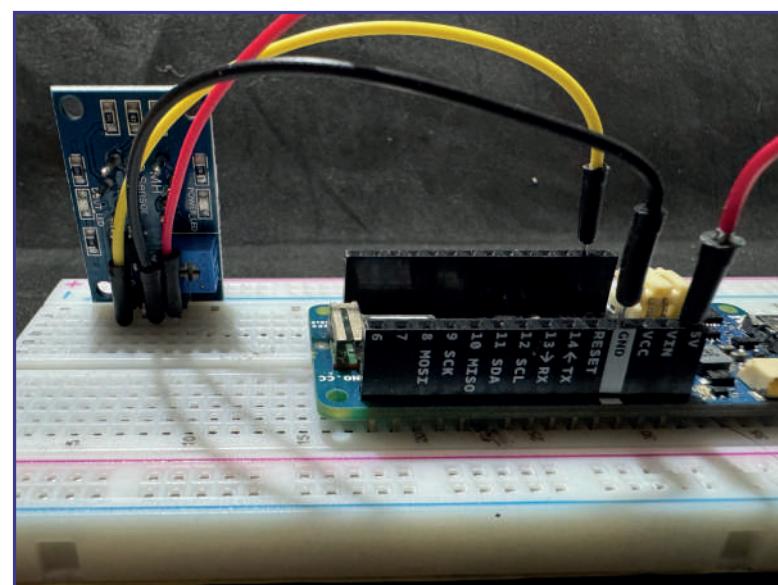
In this chapter you will learn a lot of new concepts: you will write code to handle the WiFi connection; you will use a “middleware” service; and you will store data in the Google Cloud Platform. By logging data into a spreadsheet, we are then able to carry out detailed analysis that will enable us to spot trends and, hopefully, to act upon them. If you identify poor air quality in your bedroom at certain points of the day (such as rush hour) you can close your window at these times to protect your own health.

Success criteria

- Create a Google Sheets spreadsheet to store data.
- Create a Google script to insert data to the spreadsheet.
- Create a middleware link between the Arduino and the Google script.
- Connect to WiFi directly from the Arduino.
- Write the data from the Arduino to the Google spreadsheet.
- Automatically identify concerning values on the spreadsheet.

1

This chapter will cut out the Arduino Cloud, so you need to connect your Arduino to the air quality sensor and collect data periodically to store yourself. This is very similar to what you did in the previous chapter (although we'll only be collecting data every 10 seconds or so) so make sure you have the same circuit connected and your working code to hand. Do note, there is quite a lot happening in this chapter so it may take a bit longer to complete.



2

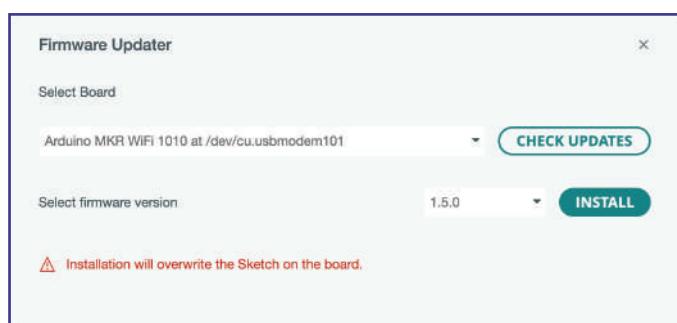
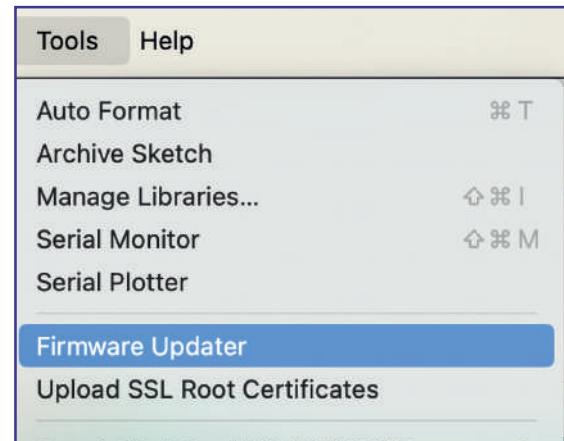
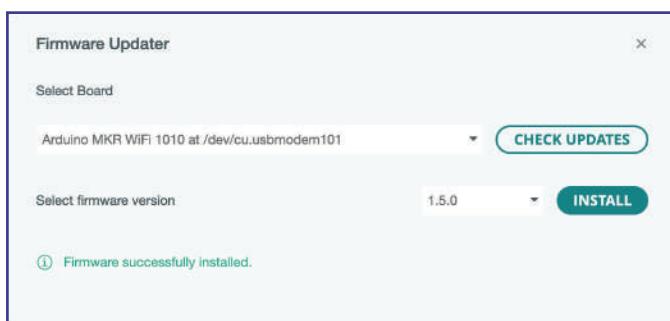
First let's ensure our Arduino can see the WiFi we need to connect to. If you are using the Arduino software, you will need to install the WiFiNINA library—if you're using the cloud it is already installed.



Then load the `wifi_check.ino` file and run it. We have provided the code for you and you don't need to worry about understanding it—we'll introduce the main concepts later. For now, look at the output.

```
Please upgrade the firmware  
MAC: 8C:AA:B5:8A:A8:2C  
Scanning available networks...  
** Scan Networks **  
number of available networks:9  
0) Reconnecting Signal: -58 dBm Encryption: WPA2  
1) WimbledonCommon Signal: -59 dBm Encryption: WPA2  
2) oculus Signal: -59 dBm Encryption: WPA2  
3) Reconnecting Signal: -63 dBm Encryption: WPA2  
4) WimbledonCommon Signal: -63 dBm Encryption: WPA2  
5) BT-3HCPX7 Signal: -72 dBm Encryption: WPA2  
6) EE WiFi Signal: -72 dBm Encryption: None  
7) Reconnecting Signal: -77 dBm Encryption: WPA2  
8) WimbledonCommon Signal: -77 dBm Encryption: WPA2
```

If you see **Please upgrade the firmware** you will need to update the firmware through **the Tools > Firmware Updater** menu, selecting the correct board (we're using the MKR1010 here) then installing.



Hopefully you should see your serial monitor listing all of the available networks.

3

The best way to complete this project is in reverse, building the system from the spreadsheet backward. First, create a Google sheet with the headings you want; in this case a timestamp and an air quality reading. You will need to copy the part of the sheet URL between /d/ and /edit and save it somewhere—this is called the **sheet reference**.

You should make any formatting changes to your spreadsheet at this time.

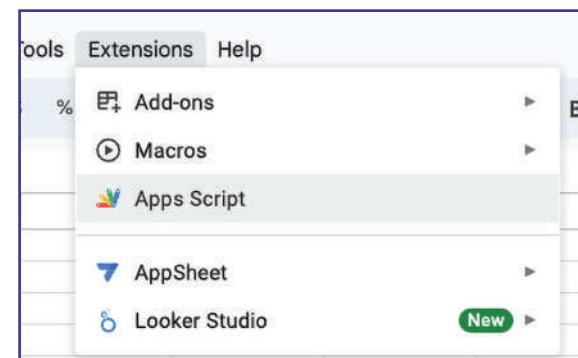
B4	A	B	C
1	Timestamp	Air Quality Reading	
2			
3			

PRO TIP

Rather than trying to copy down long alphanumeric strings by hand, create a notepad file where you can paste these strings for future reference.

4

You are going to create a script (**Extensions > Apps Script**) that is attached to the spreadsheet and will receive the data from the sensor and put it in the right place. A script is a piece of programming that acts as middleware that sits on the server to receive data from the Arduino and put it into the spreadsheet.



5

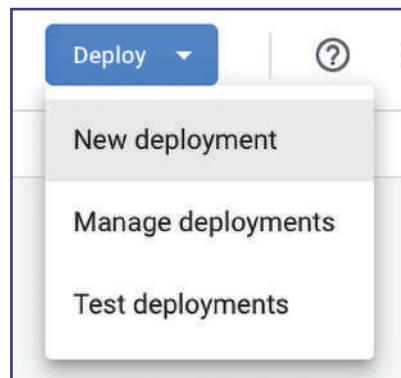
You can get the Google script code directly from the Arm GitHub repository and paste it into your project. This code has two functions:

- A helper function simply removes quotation marks that can cause problems.
- A **doGet()** function which goes through the data received from the sensor, adds a timestamp, and then adds the data into a new row on the spreadsheet. All you have to do is enter your sheet reference where indicated.

```
1 //-----  
2 // Based on code by Mogsdad@Stackoverflow,  
3 // jarkomityaz.appspot.com and Stephen Borsay  
4 //-----  
5  
6  
7 // Receive parameters on query string and  
8 // inserts in order.  
9  
10 // Add date to spreadsheet on next available row  
11 function doGet(e) {  
12   var result = 'OK'; // assume success  
13   if (e.parameter == undefined) { // Nothing defined  
14     result = 'No parameters provided';  
15   } else {  
16     var id = 'enter sheet reference here'; //Update this  
17     var sheet = SpreadsheetApp.openById(id).getActiveSheet();  
18     var newRow = sheet.getLastRow() + 1;  
19     var rowData = []; // Blank data  
20     rowData[0] = new Date(); // Add timestamp  
21     for (var param in e.parameter) { // Add other parameters  
22       var value = stripQuotes(e.parameter[param]);  
23       rowData[rowData.length] = value;  
24     }  
25     var newRange = sheet.getRange(newRow, 1, 1, rowData.length);  
26     newRange.setValues([rowData]); // Add the data to the spreadsheet  
27   }  
28   return ContentService.createTextOutput(result); // Output result  
29 }  
30  
31  
32 // Helper - remove quotes  
33 function stripQuotes(value) {  
34   return value.replace(/["']/g, "");  
35 }  
36
```

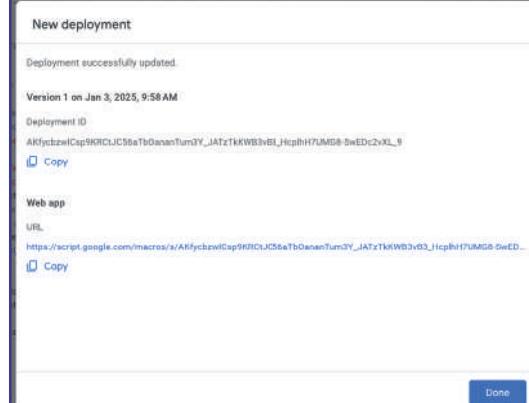
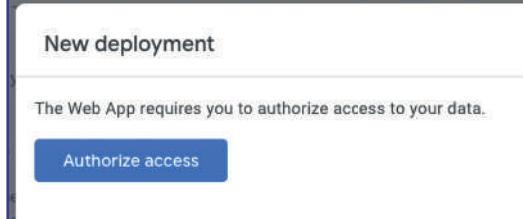
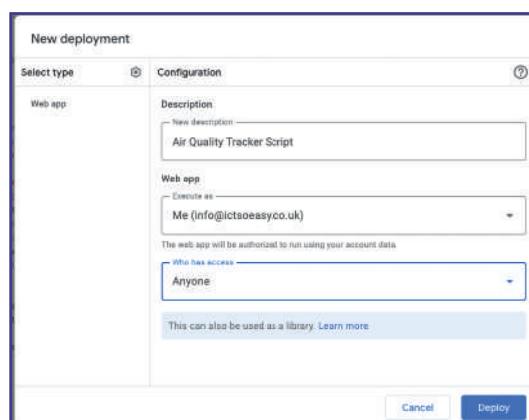
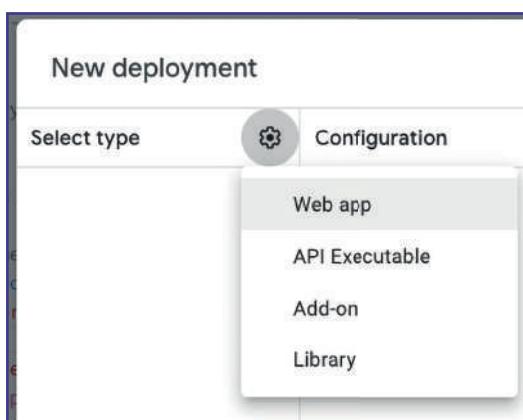
6

After saving the script, make sure you deploy the script as a new **Web App**, ensuring that **Anyone** can access it. You will need to work through the steps to **authorize** the script. At the end, make sure you copy the web app URL into your notepad file!



PRO TIP

Do note, if you need to edit your script at all (for example, if you forgot to put your sheet reference in)—you need to edit the deployment and tell it to deploy a new version.



7

Now comes the complex part: getting the Arduino to post the data to the middleware script. First, we need to add some new things to the top of our code.

- The SSID and password for your WiFi.
- The SSL port and website address; notice that the port is an integer but the website is an array of characters. This is very important.

```
#include "MQ135.h" // MQ-135 library
#include <iFinINA.h> // WiFi library
#include <SPI.h> // SPI library

const int ANALOGPIN = 0; // Pin to read the MQ-135
const float RZERO = 200.24; // Define the calibration setting
const char* MY_SSID = "SECRET"; // Your WiFi SSID
const char* MY_PWD = "SECRET"; // Your WiFi password

const int SSLPORT = 443; // The port for secure connections
const char WEBSITE[] = "script.google.com"; // Server address to connect to
const char* DEPLOYMENTID = "AKfycbzwICsp9KRCtJC56aTb0ananTum3Y_JATzTkKWB3vB3_HcpjhH7UMG8-SwEDc2vXL_9"; // Deployment ID from your script

MQ135 gasSensor = MQ135(ANALOGPIN, RZERO); // Initialise the MQ-135

WiFiSSLClient client; // Initialise the client

int status = WL_IDLE_STATUS; // The WiFi status
unsigned long lastConnectionTime = 0; // last time you connected to the server, in milliseconds
const unsigned long postingInterval = 10L * 1000L; // delay between updates, in milliseconds
```

- The deployment ID from your Google script.
- An object of **WiFiSSLClient** called **client**—this is what will actually send the data.
- **lastConnectionTime** and **postingIntervals**—this allows us to make sure we post every 10 seconds or so without blocking the loop.
- A status code for the WiFi, which starts off as idle.

```
void setup() {
    Serial.begin(9600); // Sets the serial port to 9600
    while (!Serial) {
        ; // Wait for serial. Remember - this blocks so you might prefer a delay!
    }

    if (WiFi.status() == WL_NO_MODULE) {
        Serial.println("Communication with WiFi module failed!");
        while (true);
    }

    while (status != WL_CONNECTED) {
        Serial.print("Attempting to connect to WPA SSID: ");
        Serial.println(MY_SSID);
        status = WiFi.begin(MY_SSID, MY_PWD);
        delay(10000);
    }
    Serial.println("Connected");
    printCurrentNet();
    printWiFiData();
}
```

You also need to include the code for connecting to the WiFi—don't worry we've included the whole **MQ135_monitor_ss.ino** a file in GitHub for you! This runs a few checks and then calls **WiFi.begin()** to connect, printing out the data about the network connection once it is connected.

8

The loop is pretty simple; we check how long it is since we last posted some data (if you post too regularly, Google may block you!). Then we calculate the PPM, build up a string for the URL (the text that goes in the address bar of your browser *after* the website name) which includes the data we are sending as a string. Then we pass that URL to our new function, **httpRequest()**.

9

Now it's just a case of writing that function. We close any existing connections, just in case, then connect to the server and send the request using the http protocol. We could include all of this in the loop but having it separate makes our code more readable.

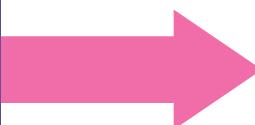
```
void loop() {
    if (millis() - lastConnectionTime > postingInterval) {
        float ppm = gasSensor.getPPM(); // Read the parts per million
        Serial.println(ppm); // Output the parts per million
        // Build the URL string
        String URL = (String) "/macros/s/" + DEPLOYMENTID + "/exec?reading=" + (String) ppm;
        httpRequest(URL);
    }
}
```

```
// this method makes a HTTP connection to the server:
void httpRequest(String URL) {
    // close any connection before sending a new request.
    // This will free the socket on the Nina module
    client.stop();
    // if there's a successful connection:
    if (client.connect(WEB SITE, SSLPORT)) {
        Serial.println("connecting...");
        // send the HTTP PUT request:
        client.println("GET "+URL+" HTTP/1.1");
        client.println("Host: "+(String) WEB SITE);
        client.println("User-Agent: ArduinoWiFi/1.1");
        client.println("Connection: close");
        client.println();
        Serial.println("Data sent.");
        // note the time that the connection was made:
        lastConnectionTime = millis();
    } else {
        // if you couldn't make a connection:
        Serial.println("connection failed");
    }
}
```

10

If all goes as planned, you should see your Arduino connect in the serial monitor, and the data will start appearing in your spreadsheet.

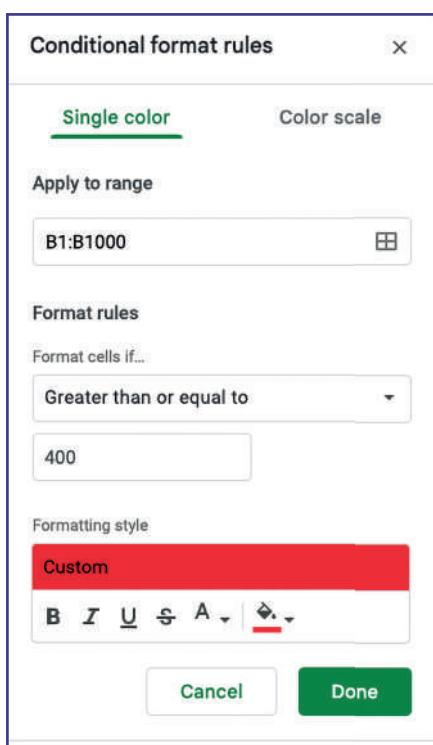
```
SSID: Reconnecting  
BSSID: F0:2F:74:EF:EB:08  
signal strength (RSSI):-57  
Encryption Type:4  
  
IP Address: 192.168.50.164  
192.168.50.164  
MAC address: 8C:AA:B5:8A:A8:2C  
384.53  
connecting...  
Data sent.  
369.04  
connecting...  
Data sent.  
353.99  
connecting...  
Data sent.  
369.04  
connecting...  
Data sent.
```



	A	B
1	Timestamp	Air Quality Reading
2	1/3/2025 11:29:13	384.53
3	1/3/2025 11:29:26	384.53
4	1/3/2025 11:29:33	369.06
5	1/3/2025 11:29:33	369.06
6	1/3/2025 11:29:39	369.04
7	1/3/2025 11:29:42	369.06
8	1/3/2025 11:30:09	384.53
9	1/3/2025 11:30:22	353.99
10	1/3/2025 11:30:36	384.53
11	1/3/2025 11:30:46	384.53
12	1/3/2025 11:30:48	369.04
13	1/3/2025 11:30:50	384.53
14	1/3/2025 11:31:00	369.04
15	1/3/2025 11:31:14	369.04
16	1/3/2025 11:31:28	369.04
17	1/3/2025 11:31:40	433.68
18	1/3/2025 11:31:54	369.04

11

Google sheets has a function called Conditional formatting ([Format > Conditional formatting](#)). By selecting column B then conditional formatting, you can set all values above a trigger value to highlight as red.



PRO TIP

If all does not go as planned, there is a `printWiFiStatus()` routine in our sketch that might help you track down WiFi related problems.

Testing

Don't be tempted to increase the delay speed between posts too much because Google might suspect you are a spammer and block you. If things don't work out, try working backward one step at a time from the spreadsheet.

If the air quality where you are is too good to vary much, try breathing on to the sensor while it is reading. The CO₂ in your breath will appear to be pollution.

Stretch tasks

- Remember we added more code to tell us what the different PPM readings mean as well as the raw reading. Log this data to a separate column in the spreadsheet. You might need to research how to build a query string.
- Add conditional formatting to highlight clean air as green, and air which is moving toward unclean but not yet polluted as yellow.
- Can you alter the Arduino code to only log changes in value?
- Identify changes which are minimal (caused by the sensor fluctuating) and adjust your code so that it does not record these; for example, if the changes are less than ±2 of the previous reading, you may choose to disregard them.

Final thoughts

This has been quite a complex project with a number of sub-systems (Google Sheets, Google scripts, and Arduino) all having to work together to achieve your goal. This is quite a common scenario. Remember, the systems used in-between are called *middleware*.

Think about:

- Where could you place your air quality sensor to capture useful data about air quality?
- What adjustments do you think you would need to make to the device to make it work in the real world?
- In the previous chapter we discussed the impact of pollution on health; large scale monitoring is important but requires consideration of power and connectivity. It may be worth contacting local groups who might collaborate to share data and build up a larger map of pollution levels.

11. PEDOMETER

Setting the scene

Step counters, also known as *pedometers*, are very widely used. Many people set themselves a target of 10,000 steps a day. What you might not know is that this number became the standard because it was included in the name of an early Japanese pedometer from 1964. The pedometer was called “Manpokei”, which means “10,000 steps meter”. Possibly more important than achieving a number of steps is walking at an appropriate rate, which is surprisingly hard to do.

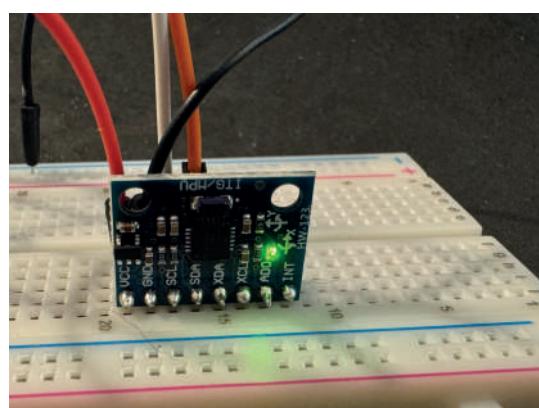
In this chapter, you will learn some new skills: how to translate accelerometer data into a number of steps; how to produce audible output using a piezo buzzer; how to use an RGB LED; and how to use an Arduino that is stand-alone and not connected to a computer. All of this will support you in building a device that will help you learn to walk at an appropriate rate and maintain that rate.

Success criteria

- Connect an accelerometer and piezo buzzer to the Arduino.
- Read data from the accelerometer data and display it on the serial monitor.
- Analyze the data to identify walking rates.
- Connect an RGB LED to the Arduino.
- Make the RGB LED change color and the piezo buzzer sound depending on walking rates.

1

In this chapter you will be connecting an external device, an accelerometer, to the Arduino. These come under various names and guises. We have used an MPU-6050 with a GY-521 3-axis accelerometer. It does have other sensors as well, but it's the GY-521 that you need to look for, with an I²C (or I2C) connection. Ours needed the headers soldering on—you can get them with headers attached already but it is a useful skill to learn! Plug your sensor into the breadboard and check the labels on the board. The VCC will need to be connected to 5 V on the Arduino (we used a red wire), GND to GND (black wire), SCL to SCL (pin 12 on the MKR1010—we used a white wire), and SDA to SDA (pin 11 on the MKR1010—an orange wire).



2

Once connected, you might need to install the relevant library. You should be used to this by now—we used the **GY521** library by Rob Tillaart.

We have produced a sample file,

accel_test.ino, which you can use to test the accelerometer is working. The key thing is to read the sensor (once) then extract the x, y and z acceleration. We have the accelerations output on a single line, **x**, **y**, and **z**.

1.10	-0.01	-0.00	
1.09	-0.01	-0.01	
0.31	-0.06	-0.07	
1.09	-0.01	-0.01	
1.10	-0.01	-0.02	
1.10	-0.01	-0.03	
1.10	-0.01	-0.02	

LIBRARY MANAGER

GY521

Type: All

Topic: All

GY521 by Rob Tillaart
<rob.tillaart@gmail.com>
0.6.1 installed

Arduino library for GY521 angle measurement
[More info](#)

0.6.1 REMOVE

Hopefully you will see numbers fairly similar to ours. Two of the numbers should be fairly close to 0 and one should be fairly close to 1. The non-negative number is reading the constant acceleration of gravity. You will see a little bit of fluctuation, partially due to the fact that we have not calibrated the accelerometer. Accuracy is not too important for this project but feel free to read about calibration yourself.

3

While there are a variety of complex mathematical methodologies, often involving machine learning, which you can implement over the data to detect steps, if you simulate stepping by holding the accelerometer flat and tapping your hand against the table, you may notice something in the data. As you tap the table, some of the values (in our case, all three) change quite substantially. You can capture that change to identify a step. Our second test program, **accel_test_2.ino**, shows this difference. We've also decreased the delay between measurements at the bottom of the code to 50 ms as the actual "step" happens quite quickly. By plotting the data on the serial **plotter** we can see the biggest change on a step is on the x—in fact, it drops from between 1 and 1.2 to a negative number, and back again.



4

It's often useful to base your program on existing working code—you could use our example code and testing `x` and `old_x` to detect a step.

```
if (oldx < 0 && x > 0) { //If it changed in x from negative to positive G
  Serial.println("X Step");
}
```

The test shown here checks if the old `x` value was below 0 and the new one is above (the bit on the graph where the line is starting to come back up again from the step).

Note that we are using *floats*, which are **real** numbers that have decimal parts, unlike the **integers** or whole numbers that we have been using so far.

5

To detect the *rate* of walking, the easiest thing to do is measure the time between the current step and the previous one. The Arduino includes a function called `millis()`, which tells you how many milliseconds have passed since it powered on.

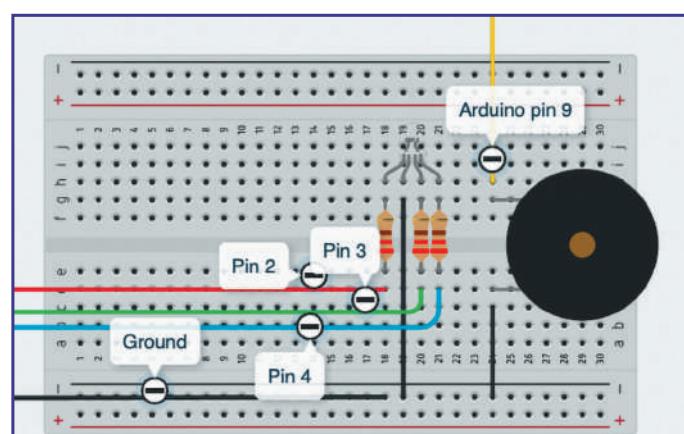
Take the time of the previous step away from the current time to find the elapsed time. You can then decide on “good” intervals, “bad” intervals, and “risky” intervals, and act accordingly. `millis()` is a built-in function which is included

(automatically) in the Arduino libraries.

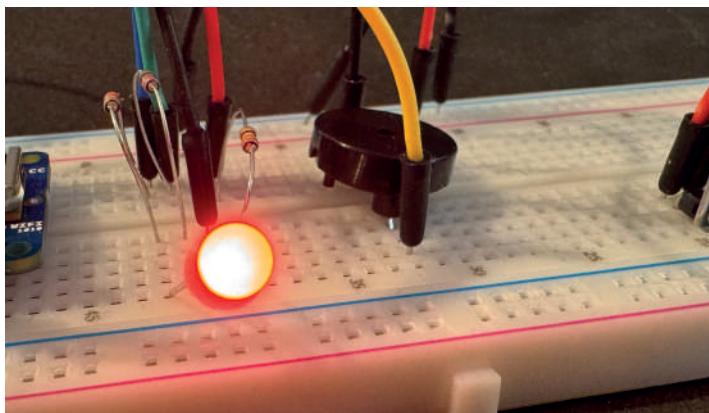
```
void step() { //A step has happened
  oldTime = newTime; //Save the old time
  newTime = millis(); //Get the new time
  elapsedTime = newTime - oldTime; //Figure out the difference
  Serial.print(elapsedTime);
  Serial.println();
  if (elapsedTime > badRate) { //Step was too long?
    // Do something bad
    Serial.println("Step - bad rate"); //Notify
  } else if (elapsedTime > warnRate) { //Almost too long!
    // Give a warning
    Serial.println("Step - warning rate"); //Notify
  }
  else { //All good
    // Do something good
    Serial.println("Step - good");
  }
  delay(100); //Wait so as to avoid double-steps
}
```

6

It is a fair assumption that you won't be carrying your computer around with you when you go out for a walk. The Arduino will have to work from a battery pack and communicate with you in some other way. You can use a piezo buzzer, to which we can send signals. (The higher the signal in hertz, the higher the sound.) You can also use an RGB LED, which is actually three LEDs (red, green, and blue) in one component. By mixing how much power each LED receives, you can produce almost any color!



- ▶ Connect the buzzer to an output pin, such as 9, and ground.
- ▶ Connect the RGB LED to three output color pins that support PWM (Pulse Width Modulation). These are normally labeled with a ~. In the example in the image, pins 2, 3, and 4 have been used.
- ▶ Run each of the pins through a $220\ \Omega$ resistor, and connect the final pin to ground (without a resistor).



PRO TIP

You can identify the RGB LED pins as follows: the longest pin is normally ground; the single pin next to ground is red; the shortest pin is blue; the remaining pin, between blue and ground, is green. You can test by making sure the long one is grounded and using a resistor to connect each leg in turn to 5 V.

7

Arduinos are well set up for controlling buzzers. The **tone()** function takes two parameters: the pin the button is connected to, and the signal (in hertz or Hz) to play. (A *parameter* is a piece of information that is passed into a routine for it to use)

tone(buzzer, 1000); will therefore play a 1,000 Hz signal. 1,000 Hz is quite a high tone, while 200 Hz is quite a low tone. After a delay, **noTone(buzzer)** will turn the buzzer off.

The function **descending()** plays 1,000 Hz, then 500 Hz, then 200 Hz for 200 ms each. This function is used as a signal to tell the user that the Arduino is working. We've also created **ascending()**.

The function **buzz()** plays 1,000 Hz for 200 ms when the user takes too long to take a step. We've also created **beeps()** for you.

```
void descending() { //A descending tone for signalling
  tone(buzzer, 1000); // Send 1KHz sound signal...
  delay(200);
  tone(buzzer, 500); // Send 500Hz sound signal...
  delay(200);
  tone(buzzer, 200); // Send 200Hz sound signal...
  delay(200);
  noTone(buzzer); // Stop sound...
}

void buzz() { //The warning 'buzz'
  tone(buzzer, 1000); // Send 1KHz sound signal...
  delay(200); // ...for 1 sec...
  noTone(buzzer); // Stop sound...
}
```

PRO TIP

If you are running your Arduino from a battery and the buzzer doesn't buzz, make sure you've disabled the **while(!Serial)** line. This line blocks the program until it can set up a serial connection, and if there's no wire, there's no serial connection!

8

It is a little harder to control RGB LEDs. To make a particular color, you need to know how much of each color (red, green, blue) is needed, using a value for each color between 0 and 255. You can find the RGB values by searching the Internet for “RGB [color name]”, e.g., “RGB pink”. Use an `analogWrite()` command to send the relevant value to each pin of the RGB LED. The function `RGB_color()`, shown here, takes the three values (red, green, and blue) and writes the values to the pins.

```
void RGB_color(int red_light_value, int green_light_value, int blue_light_value) {  
    //Write an RGB colour to the RGB LED  
    analogWrite(red, red_light_value);  
    analogWrite(green, green_light_value);  
    analogWrite(blue, blue_light_value);  
}
```

9

The `step()` function makes calls to the `RGB_color()` and `buzz()` functions as appropriate. You will probably need to experiment a bit to get the correct values for the time differences!

```
void step() { //A step has happened  
    oldTime = newTime; //Save the old time  
    newTime = millis(); //Get the new time  
    elapsedTime = newTime - oldTime; //Figure out the difference  
    Serial.print(elapsedTime);  
    Serial.println();  
    if (elapsedTime > badRate) { //Step was too long?  
        buzz(); //Buzzing  
        Serial.println("Step - bad rate"); //Notify  
        RGB_color(255,0,0); //Red light!  
    } else if (elapsedTime > warnRate) { //Almost too long!  
        Serial.println("Step - warning rate"); //Notify  
        RGB_color(255,255,0); //Yellow light  
    }  
    else { //All good  
        Serial.println("Step - good"); //Notify  
        RGB_color(0,255,0); //Green light  
    }  
    delay(100); //Wait so as to avoid double-steps  
}
```

Testing

This project may well need a lot of testing.

- First, decide on an orientation for your accelerometer and stick to it. What is **up** when the circuit is on your desk might not be **up** in your pocket!
- Then take a really good look at the data and decide which plane you will take for registering the step: x, y, or z. We've left all three in the **pedometer.ino** code for you.
- If double steps are being counted, increase the delay time.
- If not enough steps are being counted, try reducing the delay time.
- Finally, plug in a battery (such as a mobile phone auxiliary battery pack) and go for a walk. You will probably have to tweak, or "dial in", the elapsed times you use to trigger a warning as well as what counts as a "good" pace.

Stretch tasks

- Alter the buzzes to give different messages rather than just "good". You can get quite creative!
- Add buttons or potentiometers to adjust the elapsed times without having to reprogram.
- Research "pedometer code" on the Internet and see if you can allow the program to identify a step regardless of the accelerometer orientation.
- Carrying an Arduino around in your hand isn't comfortable. Build a way of attaching the pedometer and a battery pack to your arm or to your clothing.
- Why not count steps and have the pedometer play a merry tune when you hit that 10,000?

Final thoughts

IoT devices are very rarely plugged into computers and often have to go relatively long periods of time before reporting back to base. This project has shown you two ways to interact with a user when "offline".

Pedometers have been around for a long time (the 10,000 step pedometer was launched in 1964!), but with the advent of IoT it is not necessary for devices to be completely standalone. Think about how you could boost the power of your pedometer warning device by adding functionality and taking it online. What impact could this have on your own exercise?

12. BASEBALL OR CRICKET BAT METRICS

Setting the scene

More and more professional sportspeople are using Machine Learning and wearable technology during training to monitor and improve their performance. If a runner can identify small changes in their posture that reduce muscle strain, or a cyclist can identify a slightly more effective stance that will reduce wind resistance, then they increase their chances of winning. The knock-on effect is that developments in techniques and equipment like running shoes is passed down to ordinary people, so we can all run with fewer injuries and cycle with less resistance.

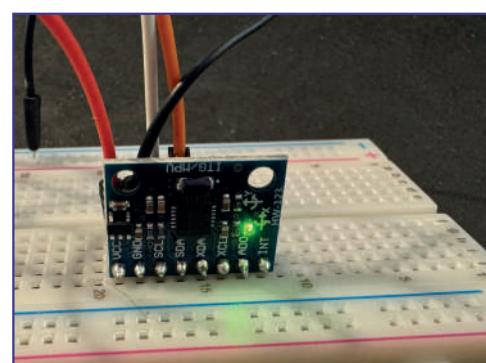
In this chapter you will learn how to attach an accelerometer to a cricket or baseball bat and measure wirelessly how g-forces change during a swing.

Success criteria

- Connect an accelerometer to the Arduino.
- Use the Arduino to measure g-forces.
- Connect a button to the Arduino to collect a sequence of g-forces.
- Write the g-force sequence to a Google Sheets spreadsheet ready for analysis.
- Connect the system to a baseball bat or cricket bat.
- Analyze the data to identify a “hit” and a “miss”.

1

As in previous chapters, the accelerometer is connected to the Arduino using an I²C (or I2C) connection.

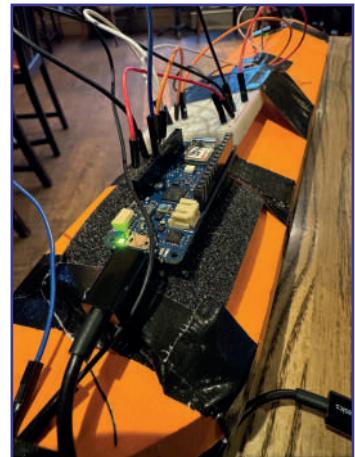


2

After including the library file and connecting the accelerometer, it has to be initialized. As this may take some time, it is worth putting it into a loop with a delay and outputting it to the serial monitor when it is completed. You won’t have access to the serial monitor in the final system, but this will help you get a feel for when the accelerometer is ready. You could always use an LED to indicate this if you wished.

3

When prototyping a system, prettiness is not normally too important. We used some foam and some strong tape to attach the Arduino to the base of a bat. We attached the sensor to the far end (all on the back), and as our wires weren't long enough, we used a breadboard halfway to extend them. We also put a push button near the handle to be able to trigger the reading and taped a battery underneath.



4

The Arduino has pull-up resistors built in (although not as strong as some use-cases might need). For our purposes this is perfect, as we can just connect the button between ground and pin 13 and initialize it with: **pinMode(buttonPin, INPUT_PULLUP);** We can then test our circuit by reading the button and, if it is "LOW", reading the sensor acceleration into some variables and outputting them.

```
void loop() {  
    int buttonPressed = digitalRead(buttonPin);  
    if (buttonPressed == LOW) {  
        Serial.println("Button");  
        sensor.read();  
        x = sensor.getAccelX();  
        y = sensor.getAccelY();  
        z = sensor.getAccelZ();  
        Serial.print(x);  
        Serial.print("\t");  
        Serial.print(y);  
        Serial.print("\t");  
        Serial.print(z);  
        Serial.println();  
        delay(50);  
    }  
}
```

5

The g-force measurement requires a little bit of maths. The x, y, and z acceleration can be read into three floats (**x**, **y**, and **z** seem like suitable names!). These give you measurements in three planes.

```
Serial.print(" ");  
double gforce = sqrt(x*x+y*y+z*z);  
Serial.println(gforce);
```

You can then use Pythagoras' theorem to calculate the overall force:

- square each of the forces
- add them together and then find the root of the sum.

6

After a bit of experimentation, 200 readings at 20 ms delays gives us enough time to record a good swing of the bat. Because we know how many readings we'll be taking, we've got two advantages. We can use a different type of loop (a **for** loop) to repeat exactly 200 times, and we can use an array which is a clever way for us to structure data. This array can contain, in this case, 200 related pieces of the same type, all in the same variable, and access them just by using an index number.

Set up an array of 200 doubles using the line **double readings[200];**. Then use a loop over to take 200 readings at 20 ms intervals and add them into the array. The index we use for the array entry is the same counter variable the loop uses.

7

The method for sending data to the Google Sheets spreadsheet for analysis is very similar to the methods you used earlier, in Chapter 10. Two key points are:

- Remember to update the deployment ID so you are sending data to the correct Google spreadsheet.
- Concatenate (join together) the 200 readings into a comma-separated list so that they can be sent as a single variable. You know that there are 200 readings, so you can use a **for** loop.

```
String URL = (String) "/macros/s/" + DEPLOYMENTID + "/exec?" + "data=";  
for (int counter = 0; counter < 200; counter++) {  
    URL += readings[counter] + ",";  
}
```

PRO TIP

Generally, if you know how many times you are going to do a loop, then you use a **for** loop; if you don't, you use either a **do...while** loop (if you need to repeat at least once) or a **while** loop (if you might not need to repeat at least once).

8

The script attached to the Google spreadsheet will have to be a little bit different for this application. It needs to take the single data string, split it on the commas to give the 200 readings again, and loop through that array to insert them into the spreadsheet. The full script code (**gforce.gs**) is available on the Arm GitHub repository. We've left a space in the first cell so that you can enter your name.

```
//Add data to spreadsheet on next available row  
function doGet(e) {  
  var result = 'Ok'; //assume success  
  if (e.parameter == undefined) {  
    result = 'No parameters provided';  
  } else {  
    var id = '1i0cMHgX7KGYACKxjJe5Pq_VEgzsjzUKpAcrrv3IYCrI';  
    var sheet = SpreadsheetApp.openById(id).getActiveSheet();  
    var newRow = sheet.getLastRow() + 1;  
    var rowData = [];  
    rowData[0] = ""; //Leave blank for name  
    for (var param in e.parameter) {  
      var value = stripQuotes(e.parameter[param]);  
    }  
    Logger.log("Here");  
    Logger.log(value);  
    var datapoints = [{}];  
    datapoints = value.split(",");  
    Logger.log(datapoints);  
    for (var dp in datapoints) {  
      rowData.push(datapoints[dp]);  
    }  
    var newRange = sheet.getRange(newRow, 1, 1, rowData.length);  
    newRange.setValues([rowData]); //Write the new row to the sheet  
  }  
  return ContentService.createTextOutput(result);  
}
```

9

At this point, once you have checked that everything is working, you can set up your bat. The type of bat doesn't really matter. The bat shown in the image in Step 3 is a soft cricket bat, which has flat surfaces for attaching the equipment. Because it is soft, it is less likely to do too much damage if you accidentally let go of it. It is absolutely essential to use non-conductive tape and that the tape is as strong as possible. The image shows thin strips of duct tape wrapped right all the way around the bat. The accelerometer needs to be as close to the tip of the bat as possible, as that is probably where the g-forces will change the most.

PRO TIP

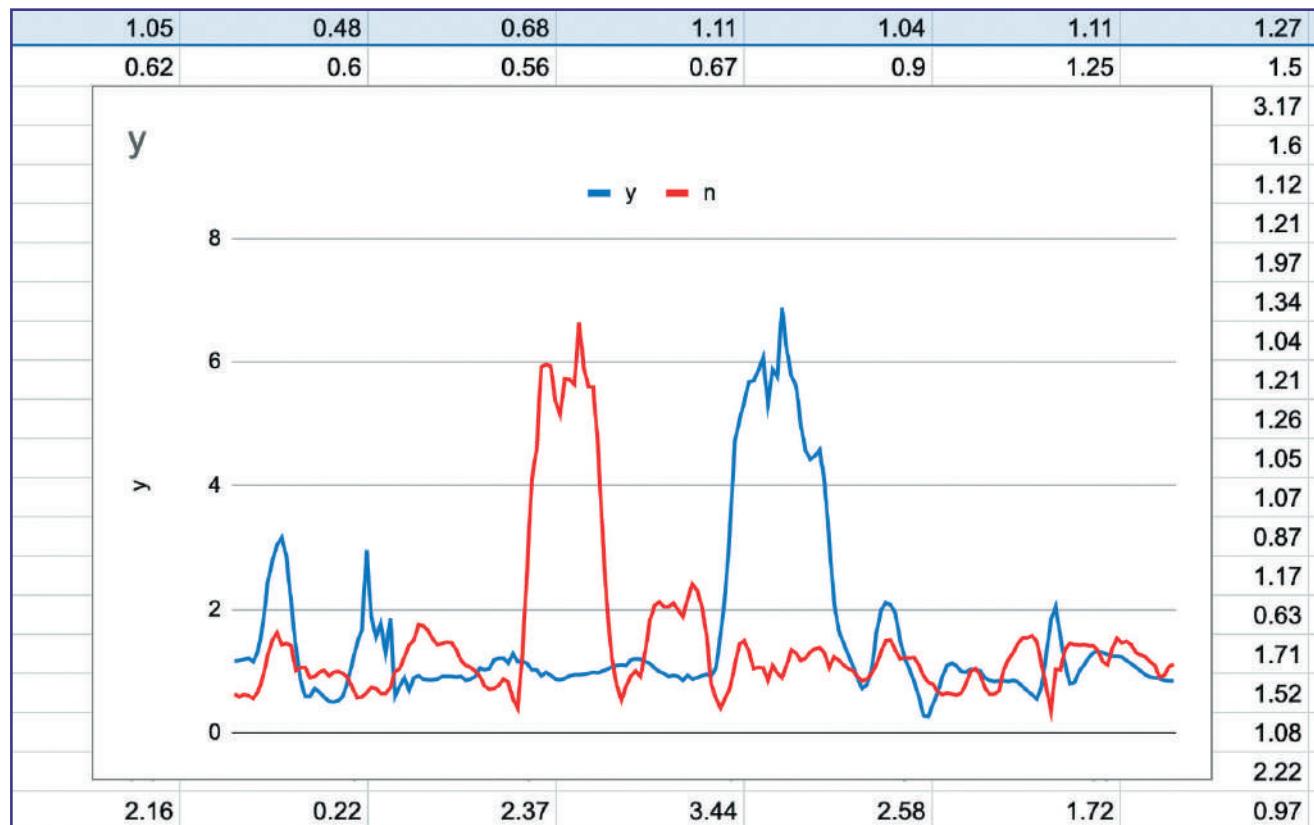
Once the battery is in use, you will no longer be able to see data on the serial monitor. It is a good idea to test everything with the system connected to the bat but still connected to the computer, before you finally plug in the battery.

PRO TIP

If everything works up until you unplug from the computer, make sure you don't have a blocking serial monitor. Begin the serial monitor then wait (for example, for 10,000 ms) rather than using a `while (1)` or `while (!Serial)` loop around it. This is one of the essential parts of de-tethering.

10

Once everything is connected securely, take some test swings and make sure the data is feeding through. Create some charts of the data to aid in your analysis. It is up to you to draw your own conclusions about what the data means!



Testing

This project will take quite a bit of time to test. It is best to work in a group of three: one to throw the ball, one to (hopefully) hit the ball, and one to record whether the result is a hit or a miss. It is worth deleting any glancing blows, as you may find you have more parity between your results without these distractors.

Stretch tasks

- This project collects a single, non-directional g-force. Adjust the system in order to record the forces in the x, y, and z planes as well as the non-directional force.
- Look at the results you have logged (either for non-directional or separate planes) and see if you can identify patterns. It may be easier if you plot some of the results on a chart.
- Can you, as a human, detect the difference between a “hit” and a “miss”?
- Change the number of readings being made. Identify the optimal number without wasting data.
- Change the frequency of readings being made. Identify how quickly you can read them before you run into problems.

Final thoughts

In 1912 the men’s indoor pole-vaulting world record was 4.02 m. In 2020, the world record was 6.18 m, which is over 50% higher. This same pattern can be seen throughout sport.

There is no one magic bean that allows sportspeople suddenly to become better, but better knowledge about what they are doing and how they are doing it has a large impact. Find out about the ethical arguments for and against the use of technology to improve sporting performance. What are your own opinions on this issue?

Anyone who spends time in the gym or has a heart monitor on their wrist knows that IoT devices abound in the sporting world. Every advance in the technology gives us more data and more opportunity to increase the general health of everyone. Consider whether you are willing to share your personal data from IoT devices, knowing that the data may be used for other purposes as well as monitoring health.

13. BIODOME (PART 1)

Setting the scene

A biodome is a sealed and self-contained environment designed to sustain life. In this chapter you will explore some of the functionality provided by the Arduino MKR ENV Shield. You will use this functionality to monitor and control your own biodome environment by turning on lights as needed and enabling an air circulation fan.

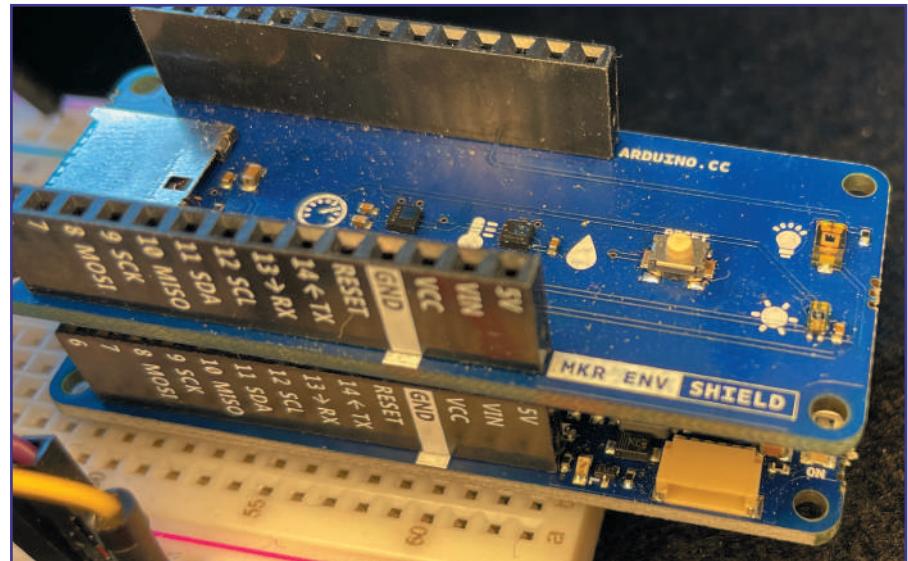
It is not suggested that you try to maintain any form of animal living accommodation with this system without very careful supervision. Plants are a lot hardier than the average hamster, gold fish or sibling.

Success criteria

- Connect the MKR ENV Shield to the Arduino and examine the sensor readings available.
- Connect and enable four bright white LEDs on a timer.
- Connect and enable a fan motor using a motor driver on a timer.
- Make the lights also turn on according to a sensor reading.
- Make the fan also turn on based on a sensor reading.

1

The Arduino MKR ENV Shield is a tremendously powerful piece of hardware. It provides information on temperature, humidity, pressure, illuminance, and a range of ultraviolet information. Connect the shield by plugging it directly into your Arduino headers (the sockets at the top of the Arduino). Be very careful not to bend the pins.



If you want to mount the shield somewhere at the top of your biodome, you might need to extend it using jumper wires. In this case, you will need to extend +5V, GND, 6, 7, 11, and 12 for most of the sensors, and A2 for the light sensor. If you want to use the SD card, you will need yet more jumper wires! You can then search for ENV in the Arduino examples file and look at the sensor readings on the serial monitor.

Connect four bright LEDs independently to resistors (220 Ω is a good value) and to a common ground. Then connect each LED to one of the unused digital pins (you can use pins 1, 2, 3, and 5, as 4 is used by the SD card). Set these as output pins and set them as “HIGH” for 12 hours of the day, and “LOW” for 12 hours of the day.

Without an Internet connection, an Arduino does not know the current time. You could get the current time over WiFi, but let’s save that for a later chapter (you will learn the skills in Chapter 17). For now, assume that you will turn on the Arduino at midday exactly. Include `<Time.h>` in your code to make time functions accessible. Then use `setTime()` at the start of the `setup()` routine to initialize the time to 12:00 (the date doesn’t matter).

You can then implement one or more conditions. For example, if you want the lights to be on all night, the conditions would be:

```
time_t t = now(); //get the time
if (hour(t) >= 19 or hour(t) < 7) {
    turnOnLights();
} else {
    turnOffLights();
}
```

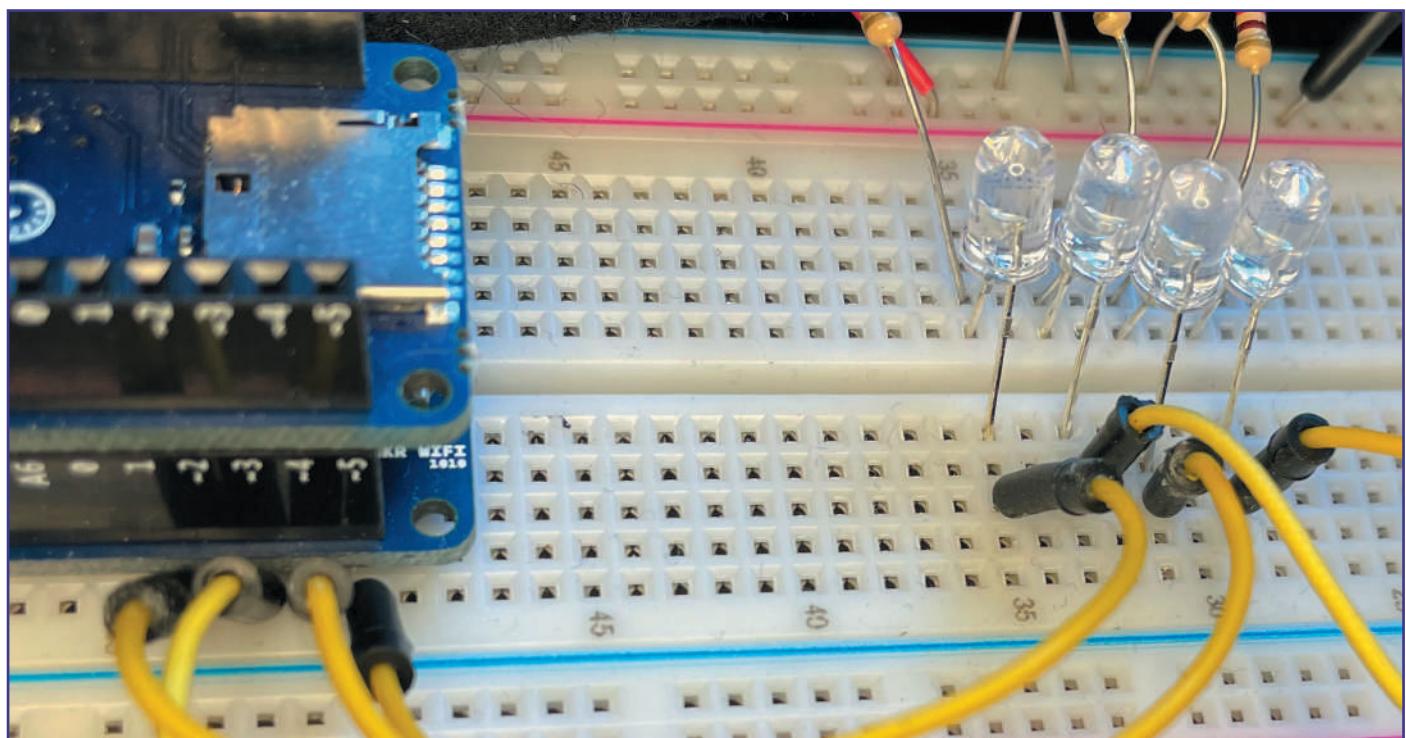
PRO TIP

Sometimes even LEDs demand more power than an Arduino can provide. If this is the case, you can use transistors in a way similar to the motor driver for the fan to use an external power supply.

This enables the lights between 19:00 and 06:59.

Note that these LEDs may not produce the light that plants need to grow.

They do, however, work as a good indicator that the plants need more light.

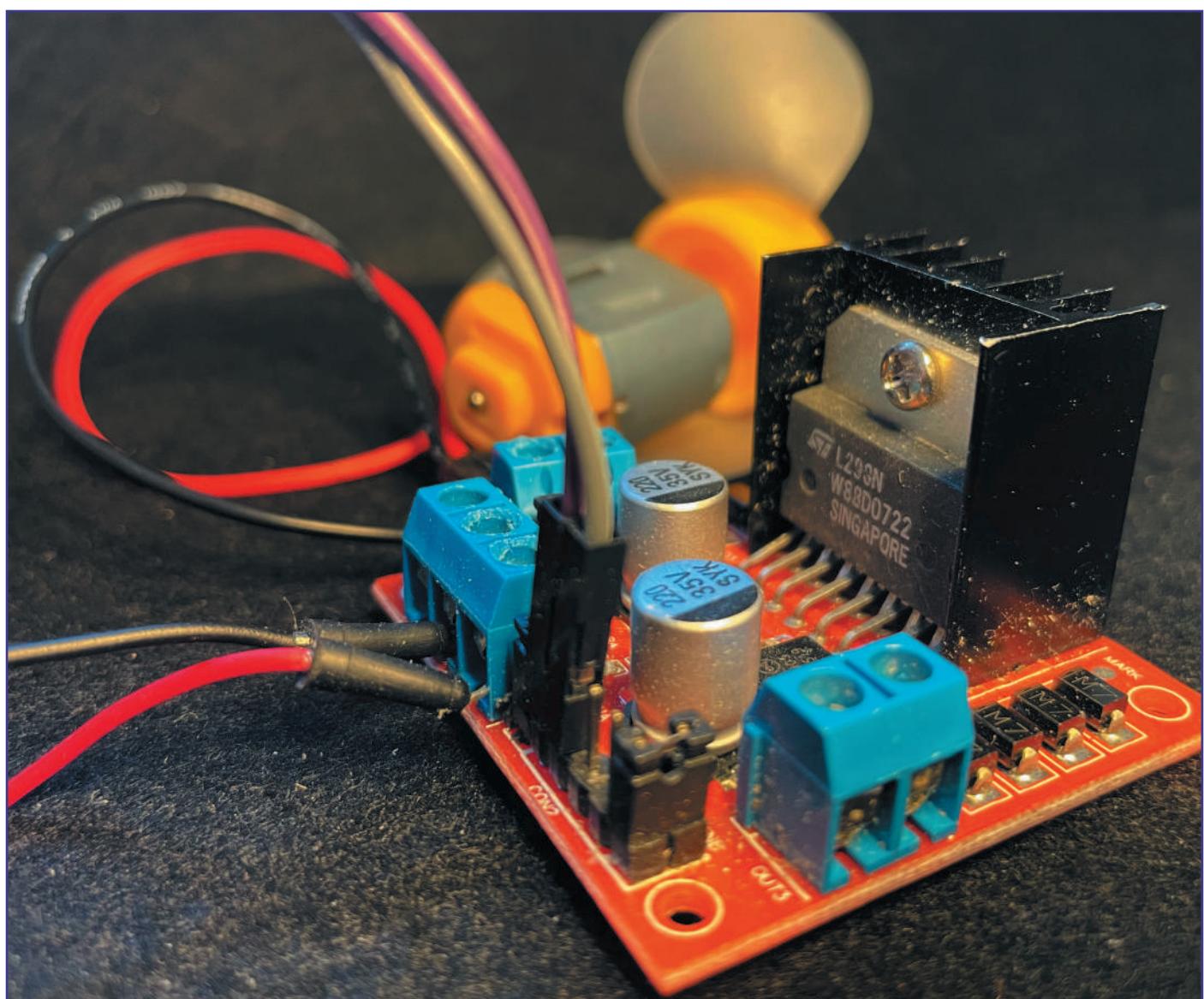


3

Fans are a bit trickier to set up than LEDs. Fans use motors, which draw a lot more power, particularly on start-up, than an Arduino can provide. An L298N device can enable an Arduino to turn a motor on or off, while allowing the power to be drawn from somewhere more suitable such as batteries or a mains adaptor.

The simple circuit shown in this example is connected as follows:

- The motor of the fan is connected to OUT1 and OUT2 (it doesn't matter which way round).
- The power supply is then connected to the power block on the front, with ground in the middle and +5 V from a battery pack on the right.
- An additional wire (not shown) needs to be run from the ground connection on the L298N to the ground on the Arduino.
- Finally, Arduino pins 8 and 9 are connected to the L298N pins IN1 and IN2. When both are high (or both low) the motor stops; when they are opposite the motor spins. Condition can be used, testing the current minute and spinning the fan for the first 15 minutes of every hour in a similar way to the code at the end of Step 2.



4

Decide what the minimum luminescence should be. Set this as a constant to be easily changeable. Then combine the condition that turns the lights on or off with an **or** operator that compares the current illuminance reading with the minimum needed.

```
//Turn the lights on or off  
time_t t = now();  
Serial.println(hour(t));  
Serial.println(minute(t));  
if (hour(t)>=22 || hour(t)<7 || illuminance < minIllum) {  
    turnOnTheLights();  
} else {  
    turnOffTheLights();  
}
```

5

Decide on a maximum temperature and set this as a constant. Then combine this into the condition that turns on the fan.

```
if (minute(t)<0 || temperature > maxTemp) {  
    turnOnFan();  
} else {  
    turnOffFan();  
}
```

Testing

You can test this project quite easily by varying the current time you set. To test the illuminance, simply put your hand over the MKR ENV Shield and see if the LEDs turn on. To test the fan, set the maximum temperature to just below the current temperature, then hold the fan close to the sensor. The fan will cool the sensor down, which should turn the fan off. The sensor will then heat up again and repeat the process.

Stretch tasks

- The code used in Steps 4 and 5 used constants. No numbers should be hard-coded into working code, so go through the code and replace all numbers with named constants.
- The MKR ENV Shield has an SD slot on it. Research how this works and log the times and readings to the SD card for later analysis.

Final thoughts

This system is an example of a *control system*. A control system reads sensors and changes something based on the readings, which in turn can have further impact on the sensors. Control systems can result in increased efficiency. For example, consider how much energy is wasted when growing crops that have too much or too little light, water, or heat. Spend some time researching control systems used for crop management.

14. BIODOME (PART 2)

Setting the scene

In the previous chapter you set up some of the basic elements of a biodome. In this chapter you will extend your biodome by adding some new components and logging the data to the Cloud.

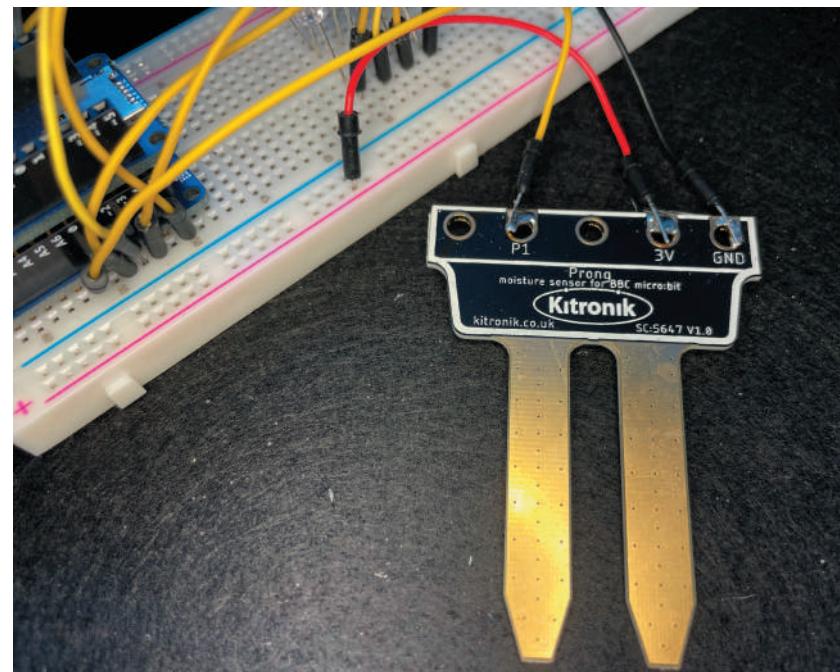
The reason for logging the data is pretty simple: you won't get the settings right first time. For example, you may get a notification to water the biodome, but find that it is already moist enough. Of course, your Arduino won't be plugged into a computer, so the only way for you to know what is happening is to look at logged data.

Success criteria

- Connect a moisture sensor and show the readings.
- Connect a servo to raise a "water me" flag.
- Log data to the Internet.
- Analyze the data to identify problems.

1

The first new component you will add is a soil moisture detector. The example shown in the image is built for a different microcontroller, a BBC micro:bit, and it does not have a direct interface with the Arduino. You may be able to use the holes and supplied bolts to connect wires to ground, 3.3 V (not 5 V!) and an analog input pin. Alternatively, you could use a soldering iron to attach some jumper wires (as in the example shown).



2

The moisture detector is very simple—it is read on an analog pin just like some of the other sensors you have used, and the value can then be output to the screen. Use a conditional statement to note that the soil is too dry if the reading is below a certain value (which you will have to find out by experimentation).

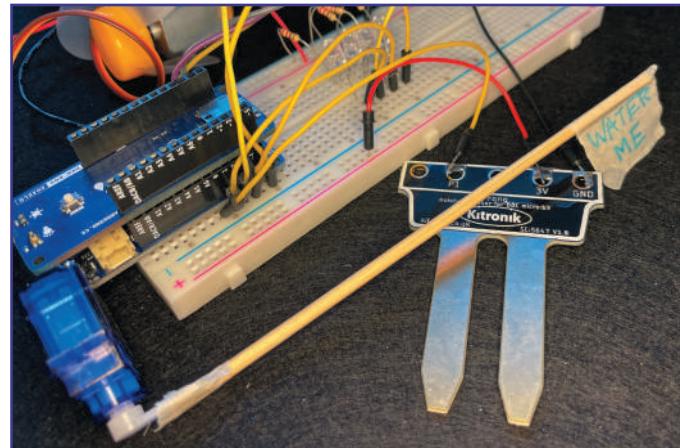
```
float moisture = analogRead(moistureSensor);
Serial.print("M: ");
Serial.println(moisture);
if (moisture < 200) {
  Serial.println("Too dry!");
} else {
  Serial.println("Moisture acceptable.");
}
```

PRO TIP

If you do not have any moist soil available, you can test the moisture detector by laying the prongs over a moist paper towel, or even by wetting a finger and joining the prongs.

3

An identifier to water the biodome would be really useful. You could use LEDs, but these don't show up very well during the day. You could also use buzzers, but these will become annoying if you don't have time to water the biodome straight away! Instead, you will use a servo (a small rotational motor) to—literally—raise a flag saying watering is needed. Use jumper wires to connect the red (5 V), black/brown (ground) and yellow/orange (signal) wires to 5 V, ground, and a digital pin on the Arduino.



4

Servos have their own libraries, **Servo.h**, which you need to include in the code.

Then create a servo object—in the example it is called **flag**:

Servo flag;

Attach the servo object to the correct pin at setup:

```
flag.attach(flagPin);
```

Set the angle for the servo in the conditional statement, using **flag.write(angle)**.

Try a 90° angle to make the flag stand upright and 0° to make it lie down, but you may need to test these to check they work as required.

```
float moisture = analogRead(moistureSensor);
Serial.print("M: ");
Serial.println(moisture);
if (moisture < 200) {
  flag.write(90);
} else {
  flag.write(0)
}
```

5

In Chapter 12, you logged data to a Google Sheets spreadsheet. You can use exactly the same techniques for this project. As a minimum, you should be logging temperature, humidity, illuminance, and moisture. It is also a good idea to include status fields for the fan, lights, and flag.

The only difference for this project is the Google Apps script. The parameters of the URL are not necessarily read in order, so if you use the script from earlier projects your data will appear jumbled. You need to update the script to check the variable name of each parameter (which needs to match the name in the Arduino code) and put them in the relevant position.

```
for (var param in e.parameter) {  
  var value = stripQuotes(e.parameter[param]);  
  switch (param) {  
    case 'temp':  
      colpos = 1; break;  
    case 'humid':  
      colpos = 2; break;  
    case 'illum':  
      colpos = 3; break;  
    case 'moist':  
      colpos = 4; break;  
    case 'lights':  
      colpos = 5; break;  
    case 'fan':  
      colpos = 6; break;  
    case 'flag':  
      colpos = 7 ; break;  
  }  
  rowData[colpos] = value; //Add parameter to rowData  
}
```

6

Once you have a reasonable amount of data in your spreadsheet, you can analyze the data and identify any problems.

From the data we initially captured in the example system, we identified that some of the flags are the wrong way around: 0 is normally used for False, and 1 for True. Also check that the flags for the output devices (lights, fan, and flag) correspond to the correct deviations in the data.

A	B	C	D	E	F	G	H
TimeStamp	Temperature	Humidity	Illuminance	Moisture	LightsOn	FanOn	FlagUp
3/18/2021	21.84	52.29	13.55	1	1	0	1
3/18/2021	21.72	52.22	13.55	101	1	0	1
3/18/2021	21.59	52.21	13.55	821	1	0	0
3/18/2021	21.48	52.49	14.19	12	1	0	1
3/18/2021	21.45	52.5	14.19	4	1	0	1
3/18/2021	21.45	52.25	14.19	10	1	0	1
3/18/2021	21.48	52.04	16.13	1	1	0	1
3/18/2021	21.41	52.86	15.48	2	1	0	1
3/18/2021	21.37	53.35	16.13	1	1	0	1
3/18/2021	21.41	52.59	15.48	1	1	0	1
3/18/2021	21.61	48.04	1.29	8	1	0	1
3/18/2021	27.56	49.43	16.77	2	1	1	1
3/18/2021	24.96	47.59	5.81	5	1	0	1
3/18/2021	23.27	48.07	5.16	2	1	0	1
3/18/2021	22.34	49.68	5.16	4	1	0	1
3/18/2021	21.78	50.49	5.16	3	1	0	1
3/18/2021	21.37	50.34	2.58	2	1	0	1

PRO TIP

Every time you update a Google Apps script, you need to redeploy and update the deployment ID in the Arduino code. If you forget this step, you can spend a long time trying to debug something that is never being called!

Testing

You will need to use a lot of trial and error with this project. Once you have made a biodome and connected your IoT device, you will need to watch both the data and the plants carefully to ensure that the settings you have chosen for temperature, moisture content, and light levels are resulting in appropriate growth. You may need to adjust the settings up or down accordingly.

Make sure you keep records of everything you change and all your findings along the way.

It is worth splitting your testing to cover the individual sub-systems—test the temperature with the fan, the moisture sensor with the flag, and so on. In larger engineering projects, systems are split into sub-systems for development, implementation, testing, and maintenance; this is normally reflected in the code, which is also sub-divided into modules.

Stretch tasks

- The Arduino automatically sets the time to midday on the 1st of January 2020 when it is powered on. Try replacing this default time with the current time obtained from the Internet. The website worldtimeapi.org will return the current date/time for wherever you are in the world. For example, accessing [https://timeapi.io/api/time/current/
zone?timeZone=Europe%2FLondon](https://timeapi.io/api/time/current?zone=Europe%2FLondon) will give plain text information that you can parse to set the time in the Arduino correctly.
- Once the Arduino is unplugged from the computer, there is no way of knowing if it is connected to the Internet. Unblock the serial monitor check and add an LED to indicate whether the Arduino managed to connect. You could use an RGB LED to provide more discrete information.
- The fan is turned either on or off. Adjust this so that the fan speed increases as the temperature goes further over the designated trigger level.
- Plan a system to add more water to the system should the soil moisture levels become too low.

Final thoughts

The biodome in this project is quite autonomous, although there you will still need to monitor it and add water if necessary. There have been several research projects into biodomes that can support human life—such biodomes will be essential to allow future habitation of other planetary objects, including the Moon and Mars. Look for some of the fascinating videos on YouTube about current human-habitable biodome projects, planned future projects, and past projects that have not necessarily turned out as planned.

Consider how much automation is optimal. With enough planning, you could automate entire farms, but would the benefits (such as cost savings and improved productivity) outweigh the drawbacks (such as reduced employment opportunities and the need to provide power for such a large system)?

15. DATA ANALYSIS AND MACHINE LEARNING

Setting the scene

There is a tendency to think of Artificial Intelligence (AI) as machines behaving like humans. Thankfully, this is still a long way in the future—one of the most common uses of AI in current computing is for Machine Learning (ML). Being able to categorize data using AI is used in many fields, including medicine, where recent developments have found that an AI system is more effective at spotting some cancers than human doctors.

In this chapter you will analyze some data using the popular programming language Python. You will analyze a set of data (training data) and identify links between the data. This model can then be used to make predictions. The model in this chapter uses a famous set of data about three species of iris flowers. Four criteria were measured for 150 flowers.

Success criteria

- Import the libraries needed for this project.
- Import and check data.
- Split the data into training data and test data.
- Train the classifier with a range of different settings.
- Identify the number of neighbors to use for classifier.
- Show the accuracy of the model on the test data.
- Test the classifying using sample data.

1

Machine Learning is not a simple process. Luckily, lots of libraries have already been created to perform many of the more complex processing for us. The examples shown in this chapter were created using the popular website replit.com, which automatically installs and updates the libraries. If you are using Python on your computer or on another site, you may have to install these libraries yourself.

```
import pandas as pd #a data analysis library
import seaborn as sns #used for data visualization
sns.set_palette('husl') #sets the colours for visualization
import matplotlib.pyplot as plt #enables the graphs
from sklearn import metrics #measures performance
from sklearn.neighbors import KNeighborsClassifier #classifier
from sklearn.linear_model import LogisticRegression #used for accuracy
from sklearn.model_selection import train_test_split #split's data
```

PRO TIP

Python is sensitive to both space and case, so make sure you are extra careful in your coding. The lines after a # symbol are comments and are just for your information.

2

Next, give Python some data to analyze. Download `iris.csv` from the Arm GitHub repository. This CSV (comma-separated value) file contains the data you will be analyzing. The data is read into a data frame using a system called *Pandas*. Pandas lets you view parts of the data and check on some common statistics.

```
data = pd.read_csv('iris.csv')
input('Show data [Press ENTER]')
print(data)
input('Show head [Press ENTER]')
print(data.head())
input('Show info [Press ENTER]')
print(data.info())
input('Show description [Press ENTER]')
print(data.describe())
```

3

You need to organize the data by:

- splitting it into the columns of data (X) and the data you are trying to find out (y)
- splitting the data further into *training data* (which will be used to build the neural network) and *test data* (which will be used to test that the neural network is effective)

Splitting the data is really important, as if you train the network using the whole data set, there will be no way of testing how accurate it will be (it will “over fit”).

```
X = data.drop(['Species'], axis=1)
y = data['Species']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4,
random_state=5)
```

4

The machine learning code will look at links between the data and the correct classification, organizing the data into nodes. The nodes which have data which is close to that of other nodes are called *neighbors*.

```
k_range = list(range(1,26))
scores = []
for k in k_range:
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, y_train)
    y_pred = knn.predict(X_test)
    scores.append(metrics.accuracy_score(y_test, y_pred))
```

Without going into too much detail, the number of neighbors used in the neural network can affect accuracy significantly. In the test model, for example, one neighbor gives just under 97% accuracy, two neighbors give less than 92%, and so on. The model is tried automatically on a range of values and compared to the test data.

More information on how a “k-nearest neighbors algorithm” works is available online; searching for KNN on Wikipedia is a great place to start.

5

The results of the tests are plotted on a graph to show the user the variation in visual format.

```
plt.plot(k_range, scores)
plt.xlabel('Value of K for KNN')
plt.ylabel('Accuracy Score')
plt.title('Accuracy Scores for Values of K of K-Nearest-Neighbors')
plt.show(block=False)
```

PRO TIP

Sometimes the graph can take some time to show up. If it does not display after a couple of minutes, try refreshing.

6

Following some more mathematics, the final accuracy of the model can be displayed. The test model returned an accuracy of over 98%, extremely good results for such a small dataset.

```
logreg = LogisticRegression()
logreg.fit(X_train, y_train)
y_pred = logreg.predict(X_test)
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

7

```
neighboursChoice = int(input('What value of n gives best accuracy? (We think 12 works well!): '))
knn = KNeighborsClassifier(n_neighbors=neighboursChoice)
knn.fit(X, y)

SepalLengthCm = float(input('Sepal length?'))
SepalWidthCm = float(input('Sepal width?'))
PetalLengthCm = float(input('Petal length?'))
PetalWidthCm = float(input('Petal width?'))
print("Prediction:",knn.predict([[SepalLengthCm, SepalWidthCm,
PetalLengthCm, PetalWidthCm]]))
```

Once you have built and tested the model, you can build a classifier using the number of neighbors that you identified from the graph. Then use the classifier to test new data against the model.

Testing

Although the model is self-testing, you can test it yourself by entering new data that the system has not seen before. You could delete a couple of lines from the model before running it and use that. Alternatively, you could search on the Internet for more measurements to try out with the model.

Stretch tasks

- Reduce the amount of data significantly and explore the impact of this on the accuracy of the system.
- Create a graph from the data in the above task to show the impact on accuracy.
- The 0.4 in `train_test_split` shows the amount of data (as a percentage) to be used for the test data. Try adjusting this upwards to explore the impact on the accuracy of the model.
- Further data sets are available from <https://www.kaggle.com/datasets>. Try to build a similar system to try and predict strokes.

Final thoughts

The classifier used in this system is called a *k*-nearest neighbor (KNN) classifier. Common uses of KNN classifiers include forecasting stock market trends, customer profiling, and identifying the chances of certain medical conditions given pre-existing knowledge about a patient. These are incredibly useful tools, but there are many potential problems. For example, facial recognition, another form of Machine Learning, has been shown to be biased against some skin tones.

Consider what problems in the world will become solvable as our understanding of what is possible with Machine Learning expands. Also consider what could go wrong if bias is introduced into the training data.

PRO TIP

Machine Learning requires a significant amount of processing power. More data leads to more accurate systems—data sets that require specialist hardware such as graphics processing units (GPUs) and multiple hours of computation are not uncommon.

16. USING MACHINE LEARNING TO IDENTIFY A HIT OR A MISS

Setting the scene

In this chapter, you will apply Machine Learning (ML) techniques to the ball-hitting data you collected in Chapter 12. Your aim in this chapter is to try to train a model to identify whether each swing of the bat was a “hit” or a “miss”.

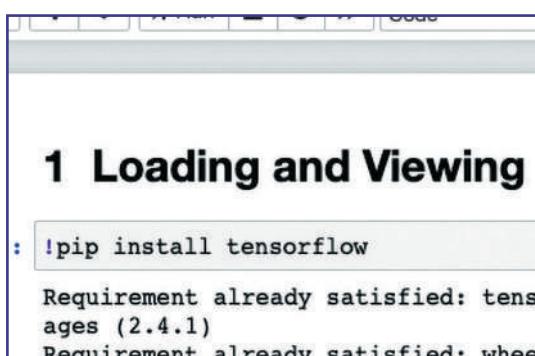
In Chapter 14, you ran a ML system on a remote server. ML algorithms using neural networks such as the one for this chapter require a bit more power, and it is recommended that you use Jupyter notebooks and, in this case, run this on your own computer. The easiest way to install Jupyter is via Anaconda (www.anaconda.com). This allows you to run pieces of code in individual cells, so you can monitor what is happening much more easily.

Success criteria

- Install TensorFlow.
- Import modules, set up some basic settings and create some useful sub-routines.
- Load data into the system and plot sample data.
- Split the data set into test and training data.
- Add some more useful sub-routines and build a model.
- Classify the test data and review the accuracy.

1

Once you have installed Jupyter and created a new notebook, you need to make sure you have the correct software libraries installed to run your code. Luckily, Jupyter gives you easy access to an installation method called *pip*—all you have to do is type the following “magic” command in the top cell and hit the run button, and Jupyter will install TensorFlow for you.



```
1 Loading and Viewing
: !pip install tensorflow
Requirement already satisfied: tensorflow (2.4.1)
Requirement already satisfied: wheel
```

!pip install tensorflow

TensorFlow has a whole host of professional-level tools for ML and will make a lot of your tasks much simpler!

PRO TIP

If any other libraries are missing, you can use the same command to install them.

2

You need to include a large number of Python modules, some of which were included in the previous chapter, and some of which are new to this chapter. As there is a lot of detail here, it is worth downloading and reviewing the full code ([bat hits](#)) from the Arm GitHub repository.

You also need to set up some basic settings:

- Use the `set_style` function (in the Python data visualization library Seaborn) to set up the plots with a white grid background.
- Create a list of the names of the classifications (0 will be a “hit” and 1 will be a “miss”).
- Create a list of the colors for the classifications when plotting on the grid—the example shown here uses dark orange for hits and steel blue for misses.

```
# General settings
sns.set_style('whitegrid') # Plots will have a white grid
# Variables that will help us work with the classes
class_names = ['hit', 'miss']
class_colors = ['darkorange', 'steelblue']
```

3

The code to instigate ML is long and complex. As with all programming projects, it is a good idea to split the code up into named sub-routines that carry out a specific task. For this part of the project, you need sub-routines to:

- load data from a csv file and convert it to the right format
- plot the data samples
- plot a single sample

```
def load_data(filename):
    hits_df = pd.read_csv(filename, header=None) # Use Pandas to load the data into a Pandas DataFrame
    print('Loaded from', filename)
    hits_data = hits_df.values # Convert from a Pandas DataFrame to a numpy array
    print('The shape of hits_data is', hits_data.shape)
    print('Number of samples of class 0 (hit)', (hits_data[:,0].astype(int) == 0).sum())
    print('Number of samples of class 1 (miss)', (hits_data[:,0].astype(int) == 1).sum())
    print('')
    return hits_data
```

Once again, you can download the full code from the GitHub repository.

4

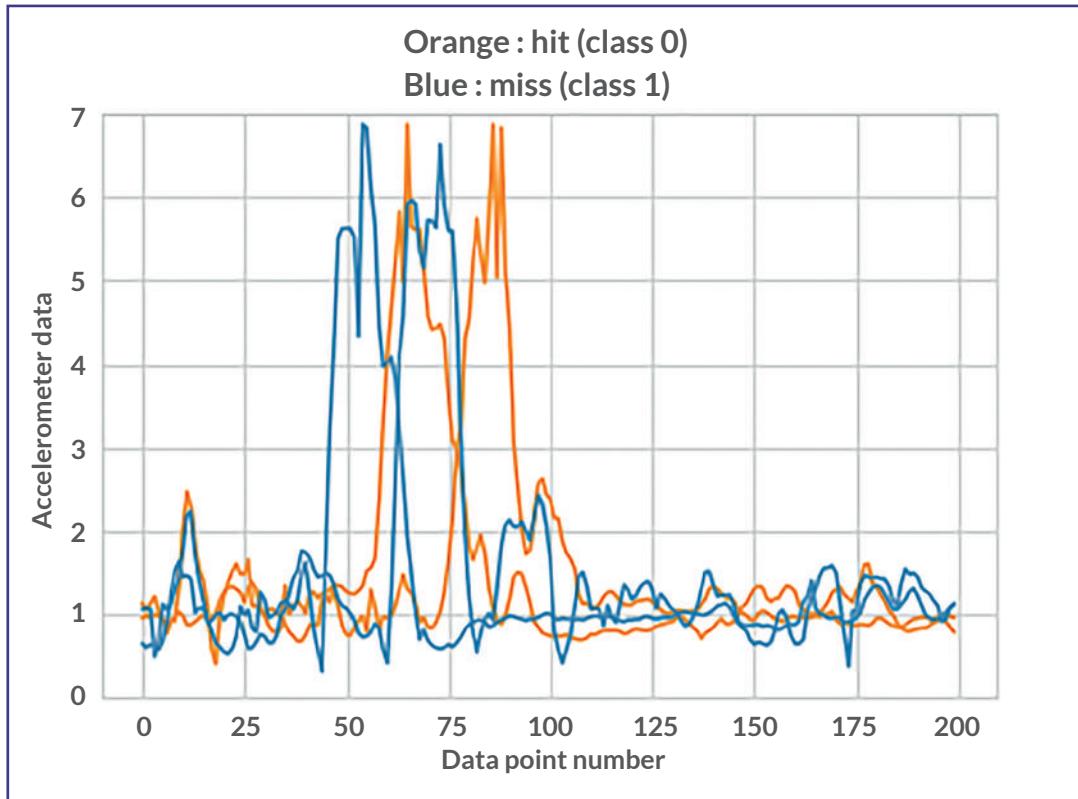
Now load the data into the system. Start by downloading your original data from your Google Sheets spreadsheet as a csv file. You need to make sure you put 0 (hit) or 1 (miss) in the first column and remove the headings. Save the csv file into the same folder as your Jupyter notebook so that you can access it. As there is a sub-routine for it, loading the data is as simple as calling `load_data()` with the filename as a parameter. You can see in the screenshot that there were 39 samples: this really isn’t enough for this kind of system—hundreds would be better!

```
filename = 'BatHits_TSD.csv'
hits_data = load_data(filename) # This is a numpy array

Loaded from BatHits_TSD.csv
The shape of hits_data is (39, 201)
Number of samples of class 0 (hit) 14
Number of samples of class 1 (miss) 25
```

5

You can show the *shape* of the data as a matrix using `data.shape`, which is useful for checking that the data looks right. To create a visual of the data, call `plot_data_samples()` to show what the data looks like on a graph. The example shown here used four random samples, 0, 1, 6, and 7 to get an idea of how the data compared. You can notice some variations—the misses have a rounded first peak, whereas the peaks for the hits are spikier.



6

With most ML, the idea is to split your data into:

- training data, which the model uses to identify the categories and teach itself how to differentiate between them
- test data, which the model doesn't get to see until it has been trained

As the example used only 39 samples, the `test_size` chosen was 10. The more samples you have to train the system the better, but you also need to leave enough test data to check the system properly!

```
test_size = 10 ### CHANGE PARAMETER HERE ###

data_train, data_test, labels_train, labels_test = train_test_split(
    data, labels, test_size=test_size, random_state=21, stratify=labels)

print('The shape of train_data is', data_train.shape)
print('The shape of test_data is', data_test.shape)
print('Training data:')
print('Number of samples of class 0', (labels_train == 0).sum())
print('Number of samples of class 1', (labels_train == 1).sum())
print('Test data:')
print('Number of samples of class 0', (labels_test == 0).sum())
print('Number of samples of class 1', (labels_test == 1).sum())
```

7

Now that you have uploaded your data into the system and checked that it looks right, the next stage is to build a model. You need new sub-routines to:

- plot a comparison between the test data and the training data
- plot the accuracy and the loss
- then build the model itself

These sub-routines are all available in the GitHub repository.

```
def build_model(print_summary=False):
    ''' Return a model with randomly initialised weights '''
    model = Sequential([
        Dense(8, input_dim=input_dim, activation='relu', name='Layer1'),
        Dense(4, activation='relu', name='Layer2'),
        Dense(1, activation='sigmoid', name='OutputLayer')
    ])

    optimizer = keras.optimizers.Adam()
    model.compile(loss='binary_crossentropy', optimizer=optimizer, metrics=['accuracy'])
    if print_summary:
        print(model.summary())
    return model
```

PRO TIP

If you struggle to get Anaconda and Jupyter to work, this code should still run in replit.com. However, the graphs may not show up!

8

To build the model, set the input dimensions (from the training data shape) and then call the **build_model** routine.

Then run through the *epochs*. (You can think of epochs as “training sessions” of the ML system.) For each epoch, the loss is calculated and the weightings of the model are adjusted to try to reduce the loss next time around, while (hopefully) also increasing the accuracy. (All of this happens behind the scenes with some very complicated mathematics!)

```
Epoch 1/20
4/4 [=====] - 0s 49ms/step - loss: 1.5131 - accuracy: 0.6358 - val_1
loss: 0.7120 - val_accuracy: 0.7000
Epoch 2/20
4/4 [=====] - 0s 11ms/step - loss: 0.7485 - accuracy: 0.5681 - val_1
loss: 0.6975 - val_accuracy: 0.5000
Epoch 3/20
4/4 [=====] - 0s 12ms/step - loss: 0.7362 - accuracy: 0.3697 - val_1
loss: 0.7474 - val_accuracy: 0.3000
Epoch 4/20
4/4 [=====] - 0s 11ms/step - loss: 0.7668 - accuracy: 0.3530 - val_1
loss: 0.7362 - val_accuracy: 0.4000
```

9

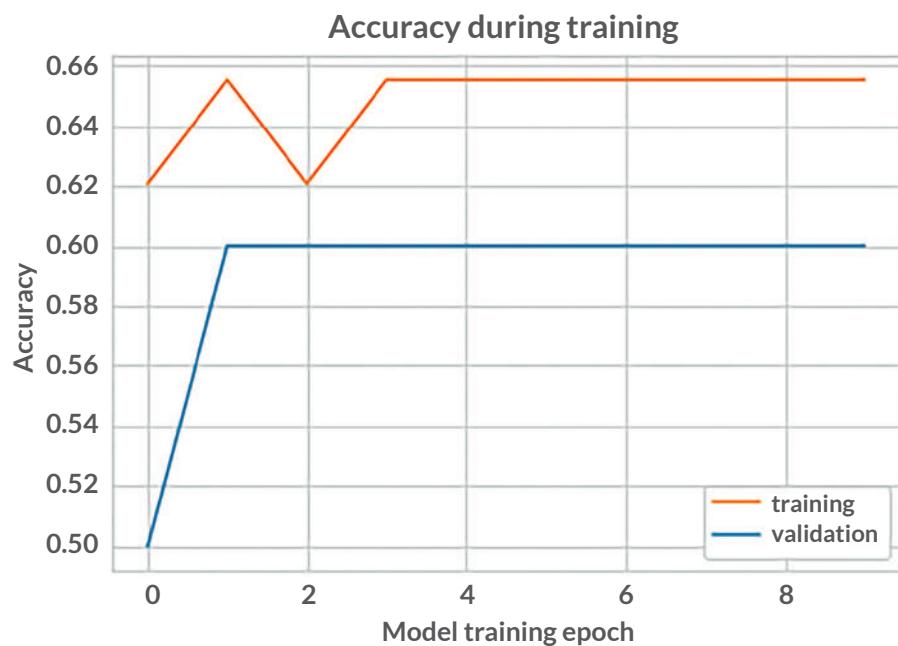
Now that you have a model that is trained, the next step is to run the model on the test data to classify the data. If you have enough training data (and good data to start with), you should see reasonable accuracy and relatively low loss. If these values are not what you would like, you will need to go get more data or adjust some of the parameters.

```
2/2 [=====] - 0s 2ms
Validation accuracy: 0.5
Validation loss: 0.7777040600776672
test_size: 10
batch_size: 8
epochs: 20
```

10

You can visualize the losses and accuracy by using the plot sub-routines you produced in Step 7. These can also help you adjust the training parameters to maximize accuracy.

```
# Plot the training log's accuracy data.
plot_accuracy(log)
```



PRO TIP

If you find your results are disappointing, there is a good chance you need to add a lot more data. Ideally for this project, try to get at least 200–300 samples of 200 data points.



11

Finally, start using your classifier, either with new data or with existing data that you present to the model as “new” data. In the screenshot you can see that sample number 3 was chosen, randomly—which the classifier identified correctly! Make sure you try a range of samples to get to know your model.

```
sample_num = 3 ### CHANGE PARAMETER HERE ###
data_sample = data_test[sample_num]
data_sample = np.array([data_sample,]) # Convert the data sample into the
probability = model.predict(data_sample)
print('Model: probability of belonging to class 1:', probability[0][0])
print('Predicted class:\t', (np.round(probability)[0][0].astype(int))) # \
print('True class:\t\t', labels_test[sample_num])

Model: probability of belonging to class 1: 0.50909096
Predicted class: 1
True class: 1
```

Testing

There are lots of things you can vary in this project, particularly the batch sizes (the amount of data analyzed at a time) and the number of epochs. Experiment with varying these and re-run the project several times to explore the impact they have. Then test the system with a new set of test data.

Stretch tasks

- The code used in this chapter is an adjusted sub-set of that in <https://github.com/jarusgnuj/ai-ml-wksh> from an Open University workshop in which data was collected about the floor a robot was walking on. Explore the sample code in the workshop notes and identify what increased the accuracy of the robot walking code.
- Identify other ML-based experiments you could do with the accelerometer and carry them out. What kind of movements give more clearly differentiated data?
- <https://www.kaggle.com/> has time-series data on the daily climate change. Use your skills from this chapter to build a model and predict the temperature on a specific date, given the atmospheric readings.

Final thoughts

ML is one of the most exciting up-and-coming areas of Computing and is just starting to make its way into degree and post-graduate education. With enough development, Machine Learning could become more accurate at classification and problem-solving than humans. Recent developments have resulted in software that is more accurate than human doctors at detecting breast cancer, for example.

Consider what other aspects of health and well-being could be improved if ML systems become more prevalent in early diagnosis procedures. However, it is also important to consider who is deemed at fault if (and when) things do go wrong.

17. WEB SCRAPER

Setting the scene

The Internet holds vast amounts of data. It is estimated that, in 2020, about 2.5 quintillion bytes of data were produced by humans every single day. A lot of this data is produced, processed, and stored in specialist systems. However, an awful lot is in far less structured and much of this data just passes us by.

This project introduces two new concepts: how to use the Arduino as an IoT device that pulls and displays data (rather than sensing and sending to the Cloud); and how to connect a display to allow the device to work when disconnected from a computer. Modern websites are hugely complex, with the useful data hidden away inside layer after layer of HTML code. This chapter involves a simple website that just shows a random joke every time you access it.

Success criteria

- Connect and test an LCD display screen.
- Connect a button to the Arduino to “trigger” data scraping.
- Use a web library to retrieve data.
- Parse useful elements from web data.
- Display those useful elements.

1

LCD screens are a great way of displaying simple text-based data. A variety of types and configurations are available. Traditionally they have required quite a few wires to set up. We recommend getting one with a I²C adapter connected to it, otherwise you have to make a lot of extra connections.



2

The I²C adapter requires four connections:

- Ground can be connected to the common ground from the Arduino.
- VCC can be connected to the 5 V connection on the Arduino.
- SDA can be connected to SDA on the Arduino (pin 11 on the MKR1010)
- SCL can be connected to SCL on the Arduino (pin 12 on the Arduino)

```
1 #include <Wire.h>
2 #include <LiquidCrystal_I2C.h>
3
4 LiquidCrystal_I2C lcd(0x27, 16, 2);
5 void setup()
6 {
7     lcd.init();                                // Initialize the lcd
8     lcd.backlight();                           // Turn the backlight on
9     lcd.setCursor(2,0);                        // Set cursor col 2 row 0
10    lcd.print("Hello, world!");                // Output a string
11    lcd.setCursor(1,1);                        // Set cursor col 1 row 1
12    lcd.print("Arm Education");               // Output a string
13 }
14
15 void loop()
16 {
17 }
```

To test the LCD, you can search for the [LiquidCrystal_I2C](#) library in your Arduino software, install it, and then run the code in

LCD_Hello:

Don't worry, we've put it in GitHub for you.



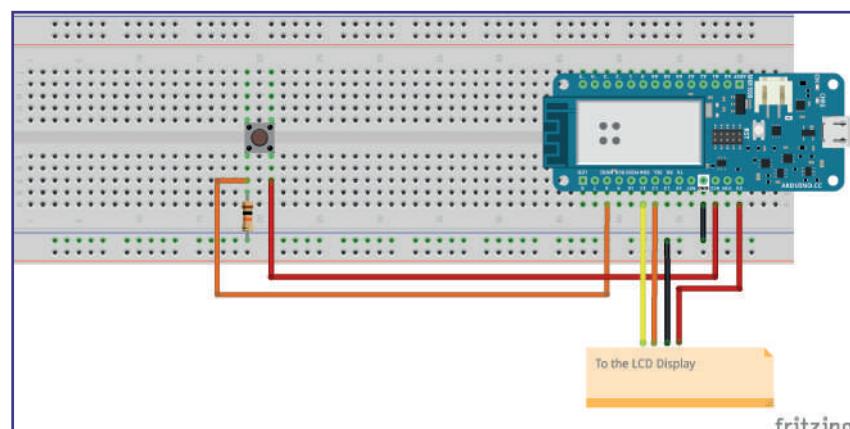
PRO TIP

If the screen doesn't show a message, there are two things to try; firstly—on the back of the screen is a little screw.

With the screen turned on, turn the screw completely anti-clockwise until squares appear, then back clockwise until the letters appear. If that doesn't work, try installing the [I2CScanner](#) library and loading the scanner from the examples folder to check the correct code is being used.

3

To trigger a “scrape”, you need to connect a pulldown button. If you are following the example code, connect the button to pin D8.



PRO TIP

You may find that the pulldown button causes fluctuations within the LCD display when it is pressed and can cause the serial monitor to disconnect or the screen to blank. If this happens, you can link to the VCC power on the Arduino instead of the 5 V for the positive side of the button.

4

We are used to telling the user what is happening on the serial monitor, but it is a good idea to use the LCD screen too. Use the skills you learned from storing IoT data in the Cloud to setup a WiFi connection. This time, however, use the LCD screen to print “Wait!” at the start, and then “Ready!” once a connection is made.



PRO TIP

Different Arduinos use different libraries for WiFi connectivity. The example code for this project ([Web_Scraper.ino](#)) includes the references for the MKR1010 using WiFiNINA—you might need to change these to references for other boards. The best way of finding the correct libraries is by joining the Arduino forums and discussing with others that share a board with you. Don't forget you need to use **WiFiSSLClient** for most connections (those that use https) and **WiFiClient** for those that don't (that use http). Ictsoeasy uses https.

5

There are three key stages involved in getting, manipulating, and displaying the data from the website.

The first is to tell the website to send you the data. We check for the button press and then call the **httpRequest()** function.

In the **httpRequest()** function, the line **client.connect(WEBSITE, SSLPORT)** connects to the server. By wrapping this up in an **if** statement, you can ensure that data is only requested if the connection worked.

```
void loop() {
    if (digitalRead(btn) == HIGH) { // Button is pressed
        Serial.println("\nStarting connection to server...");
        httpRequest();
        delay(1000);
    }
}
```

Then these lines are part of the http request and are sent to the client:

➤ **GET /path/to/website/**

file.php HTTP/1.1 will instruct the server to send a particular web page.

➤ **Host: thewebsite.com**

ensures the server knows which site you're trying to access (web servers can host multiple web sites).

➤ **User-Agent:**

ArduinoWiFi/1.1 tells the web server what kind of "browser" we are using—in this case our own!

➤ **Connection: close** followed by a blank line will tell the server that you've finished with it.

In this example, the jokes can be accessed at <https://ictsoeasy.co.uk/badjokes/bad-jokes-f.php> so the host is `ictsoeasy.co.uk` and the path is `/badjokes/bad-jokes-f.php` (plus the extra bit we added on the end).

6

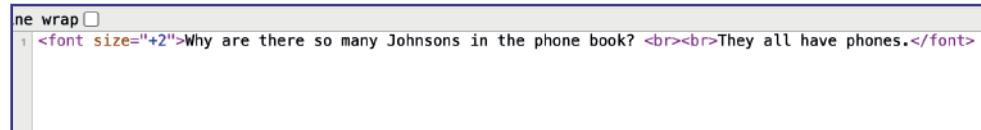
The second stage is to read the data from the server, and continue reading it for as long as it is available. This, after a short delay, uses `client.read()` and reads one character at a time, appending it on to the string `htmlContent`. We can then output this to the serial monitor for checking.

```
Serial.println("finished connection");
delay(1000); // Delay for the data to arrive
htmlContent = "";
// Echo the full html data for checking
while (client.available()) {
    char c = client.read();
    htmlContent += c;
}
Serial.println(htmlContent);
```

7

The final stage is to parse the HTML content for the bits you want.

Thankfully this web page is very simple: it gives a joke prompt and a punchline, separated by two `
` tags, all inside some `` tags.



```
ne wrap □
1 <font size="+2">Why are there so many Johnsons in the phone book? <br><br>They all have phones.</font>
```

```
Serial.println(htmlContent);
// Extract the prompt and punchline
getPromptAndPunchline(htmlContent);
// Display the joke
```

We can use `indexOf()` to search for key parts of the code, and `substring()` to split the code into strings for the prompt and the punchline. We've put all of this inside a function `getPromptAndPunchline()` which we pass `htmlContent` to.

```
void getPromptAndPunchline(String html) {  
    //Get the prompt and punchline  
    int start = htmlContent.indexOf(">") + 1;  
    prompt = htmlContent.substring(start);  
    int end = prompt.indexOf("<");  
    punchline = prompt.substring(end + 8);  
    prompt = prompt.substring(0, end);  
    end = punchline.indexOf("<");  
    punchline = punchline.substring(0, end);  
    //Display on serial for debugging  
    Serial.print("Prompt: ");  
    Serial.println(prompt);  
    Serial.print("Punchline: ");  
    Serial.println(punchline);  
}
```

8

Now to display the joke! The LCD module has a very handy function for autoscrolling text across the lines. However,

```
void scrollMessage(int row, String message, int delayTime, int totalColumns) {  
    for (int i=0; i < totalColumns; i++) {  
        message = " " + message;  
    }  
    message = message + " ";  
    for (int position = 0; position < message.length(); position++) {  
        lcd.setCursor(0, row);  
        lcd.print(message.substring(position, position + totalColumns));  
        delay(delayTime);  
    }  
}
```

the lines must be less than 40 characters long—and the jokes are often longer than that! So, we've provided you with a function to do that for you.

We've then wrapped the calls, plus a clear screen call, inside a function to complete our code.

```
getPromptAndPunchline(htmlContent);  
// Display the joke  
displayJoke();  
} else {  
    // if you couldn't make a connection:  
    Serial.println("connection failed");  
}
```

```
void displayJoke() {  
    lcd.clear();  
    scrollMessage(0, prompt, 250, 16);  
    scrollMessage(1, punchline, 250, 16);  
}
```

You can see our complete code on the Arm GitHub repository—it's called [web_scraping.ino](#).

PRO TIP

Updating the screen is quite fast—you can experiment with the delays and so on to be quite creative.

Testing

Connecting to WiFi can also cause lots of problems. Ensure that you output as much as possible to the serial monitor while you are developing your code. You can always comment the print statements out later if they are not needed. Don't forget to update the firmware: WiFiNINA and your Arduino firmware have to be in sync or it will not work. This can be quite difficult to figure out, especially if you have swapped Arduinos and forgotten about it.

When you are figuring out how to “chop down” HTML code, you need to work iteratively. Explore the source code and find one bit at a time, display it to the serial monitor, then do the next bit. You can view the source code in your web browser—but this can be quite a challenging task with more complex websites!

Stretch tasks

- Most LCD screens allow you to alter the brightness of the backlight. Use a potentiometer and vary the voltage between full brightness and no backlight.
- Strings are simple to use, but wasteful of memory. Try to amend the code to use char arrays wherever possible instead.
- Use the source code inspector (by pressing F12 if you use the Chrome browser) to explore the contents of an X stream (or similar). Extend your code to find and display the latest tweet for a given hashtag.
- Extend the previous task to include using a button to move on to displaying the next tweet in a stream.

Final thoughts

The joke site used in this chapter is one of the simplest available. Building a scraper for more complex websites is far more complex. However, data is incredibly powerful. For example, you could create a gardening project by scraping weather data and programming your Arduino to alter how much light, shade, or water it provides to the garden. What other data can you find online that you can scrape and use with your Arduino to support future projects?

18. GPS TREASURE HUNT

Setting the scene

Not so long ago the only way to know where you were was by using a mixture of map reading and dead reckoning (calculating your position), together with a reasonable amount of luck. Now, the latest GPS devices can pinpoint you to within 30 cm! GPS, and the associated sat-nav systems, help reduce wasted journeys and are invaluable for locating people in an emergency.

GPS is one example of the huge amounts of data that are all around us. All we need is a sensor to receive the data. In this chapter you will use the inexpensive Arduino MKR GPS Shield to identify your location. To make things a bit more fun, the final part of the chapter shows you how to use this device to play a treasure hunt game.

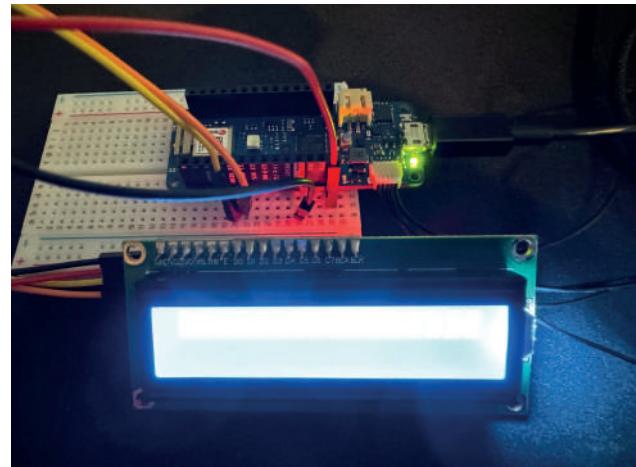
Success criteria

- Connect and test the LCD display screen.
- Connect and test the GPS shield module.
- Display the GPS details on the serial monitor.
- Display the GPS details on the LCD screen.

1

The circuit in this chapter should be quite familiar. In fact, if you still have it, you can re-use the circuit from Chapter 17.

Just remove the button and the wires that are attached to it.



2

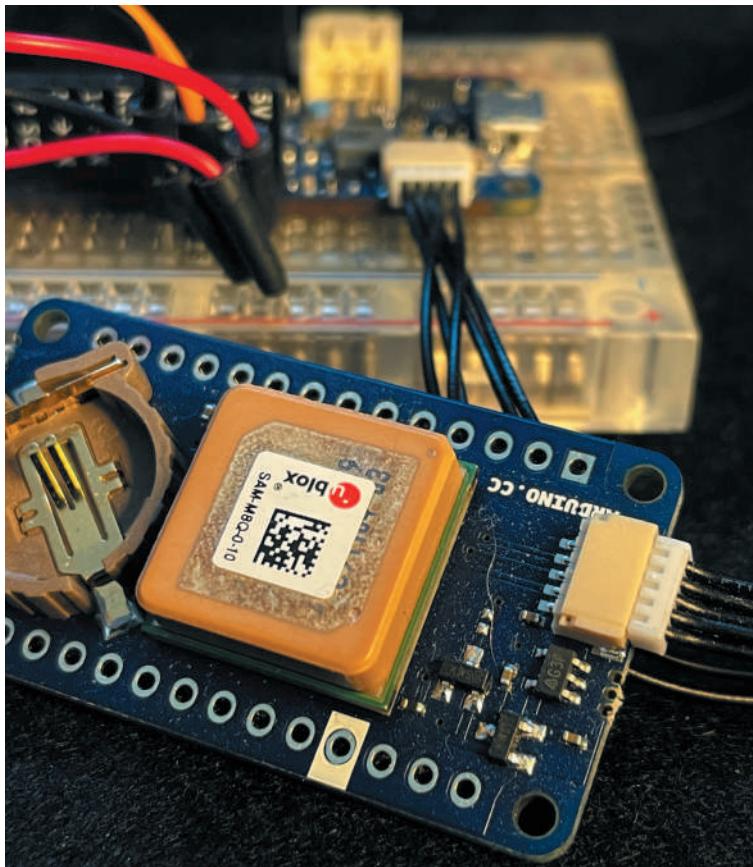
This device is designed to be used when you are out walking. Unless you also want to carry a computer with you, it is a good idea to use the LCD to mirror anything you put on to the serial monitor. You can test the LCD at the same time as (trying) to start the serial monitor in the setup routine.

```

//Initialise serial monitor
lcd.begin(16,2);
lcd.setCursor(0,0);
lcd.print("Trying serial");
// initialise serial communications and wait for port to open:
Serial.begin(9600);
delay(10000); //Wait 10s for serial monitor if connected
lcd.setCursor(0,0);
if (!Serial) {
  lcd.print("No serial.  ");
} else {
  lcd.print("Serial ready ");
}

```

3



You can plug the Arduino MKR GPS Shield directly into the MKR1010 5-pin socket. If you are using a different Arduino without the socket, you will have to either mount headers or figure out which pin in the plug needs connecting to which pin on the Arduino and use jumper wires. Using the shield means soldering the headers, and you will need to edit the code to say **GPS.begin(GPS_MODE_SHIELD)**. It is a bit more robust and we found works better. Connecting using the attached plug enables you to put the GPS Shield near a window for better reception.

4

The GPS module uses the **Arduino_MKRGPS.h** library. To initialize the GPS module, simply use **GPS.begin()** (or **GPS.begin(GPS_MODE_SHIELD)**) if needed) in the setup routine. As with the serial monitor, it is worth reporting whether this works (it returns 1 or True) or doesn't work (it returns 0 or False) on the LCD as well as on the serial monitor.

```

lcd.setCursor(0,1);
lcd.print("Trying GPS");
Serial.println("Trying GPS");
delay(3000);
lcd.setCursor(0,1);
if (GPS.begin()) {
  lcd.print("GPS ready.");
  Serial.println("GPS ready");
} else {
  lcd.print("GPS failed.");
  Serial.println("GPS failed");
}

```

5

The GPS module has functions for returning all the relevant information.

- As long as **GPS.available()** returns True, you can call **GPS.latitude()**, **GPS.longitude()**, and **GPS.altitude()** (all floats), and **GPS.satellites()**, an integer.
- If **GPS.available()** returns False, then there is no data available to be read.

```
void loop() {  
    if (GPS.available()) {  
        // read GPS values  
        float latitude = GPS.latitude();  
        float longitude = GPS.longitude();  
        float altitude = GPS.altitude();  
        int satellites = GPS.satellites();  
        Serial.print("LAT: ");  
        Serial.println(latitude, 7);  
        Serial.print("LON: ");  
        Serial.println(longitude, 7);  
        Serial.print("ALT: ");  
        Serial.print(altitude);  
        Serial.println("m");  
        Serial.print("SAT: ");  
        Serial.println(satellites);  
    }  
}
```

PRO TIP

If the GPS Shield cannot see any satellites, it will not return any errors—it will simply return False from **GPS.available()**. You may be able to get a signal near a big window, but you may have to go outside to get a good signal. We had to use a battery pack and go for a 5 minute walk.

6

As well as outputting everything to the serial monitor, it is a good idea show the most important data (the latitude and longitude) on the LCD screen so that you can check your position when running from a battery.

```
lcd.setCursor(0,0);  
lcd.print("LAT:");  
lcd.print(latitude, 7);  
Serial.print("LON: ");  
Serial.println(longitude, 7);  
lcd.setCursor(0,1);  
lcd.print("LON:");  
lcd.print(longitude, 7);
```

7

It is easy to use your GPS device to create a game. Hide objects at several different locations, and write down the GPS coordinates of each location. Ask a partner to use your GPS device to navigate to the locations and find the objects.

For even more fun, instead of objects, hide words encrypted using a Caesar cipher. The words could make up a message, which your friend has to decrypt.

PRO TIP

You may need to do this activity outside. Remember that accuracy may vary, so keep an eye on how much your readings are fluctuating when choosing your locations.

Testing

There is a really simple way to test your GPS location. If you copy the latitude and longitude, separated by a comma, into the search bar of Google Maps, you should see your current location highlighted. If the device isn't working as expected, compare the results of both the serial monitor and the LCD display to identify where the problem might be.

Stretch tasks

- Latitude identifies how far North or South you are from the Equator; longitude identifies how far East or West you are from the Prime Meridian. You can compare the target location and the current location to identify the direction and distance between them. Try making the LCD screen alternately display current locations and direction information.
- You will notice that the current location fluctuates even when you are standing still. You can measure the variation over several readings and use an RGB LED to identify the stability of the signal: the more stable the signal is the less fluctuation there will be and, correspondingly, the smaller the area of potential error will be.
- Allow a user to enter a GPS location over the serial monitor and store it into memory (research into PROGMEM). Use Pythagoras' theorem to identify the distance to the target location.
- Add a servo with an arrow on to point left/right/centrally to direct the user towards the given location.

Final thoughts

Geocaching is a popular activity, which encourages individuals, groups, and families to go out into nature, get some exercise, and solve problems together. You can find out more at geocaching.com. You could use your device to take part in geocaching activities, building your experience and improving your fitness. You could even set up your own geocaching activities for your family and friends.

19. LOCK BOX

Setting the scene

This chapter combines several concepts that you have studied in previous chapters, and also introduces some new methods of solving problems. You will use a servo as an *actuator* (an output device that moves part of a machine or device) to lock or unlock a box. The servo arm will move into place to stop the door opening and rotate out of the way to allow the door to swing open. To activate the servo and unlock the box, the user must enter a code on a series of buttons. To lock the box again, they must enter a new code. Each button will have a corresponding dual-purpose red LED to indicate a press or an error; a green LED will indicate a correct code. This project will encourage you to think creatively to solve problems by considering how sub-systems can interact.

1

The lock box can be as simple or as complex as you wish. There are numerous online tutorials with instructions for building a servo-operated lock box. You could use wood, card, foam board, or any other material you wish. Make sure your design includes a servo arm which can be deployed to stop the door opening and can also be rotated out of the way to allow the door to swing open.

Success criteria

- Design a lock box.
- Build a circuit built with buttons, LEDs to indicate error and OK, and a servo.
- Declare the variables and constants.
- Initialize the device.
- Write helper routines to reset the entered code, lock and unlock the box, clear the red LEDs, and show the current digits entered.
- Write a helper routine written to check the code entered and act accordingly.
- Write a helper routine to deal with the buttons being pressed and to check for the button press in the main loop.

Box

Servo
Door

2

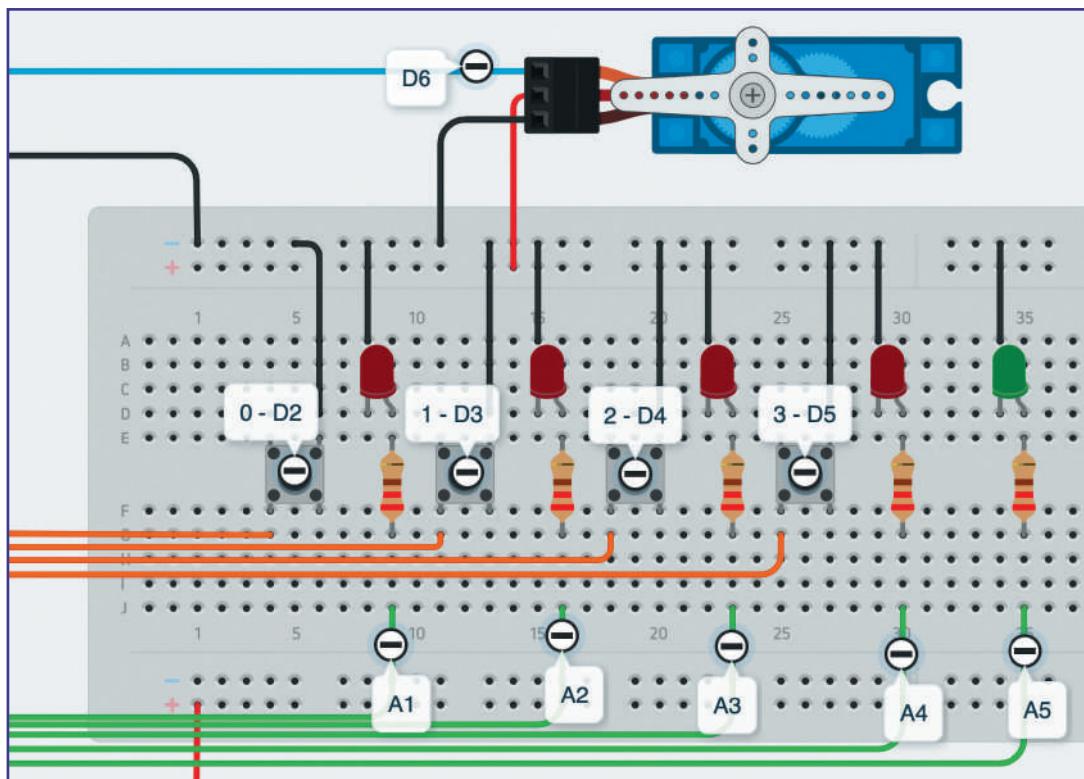
The circuit is similar to many others you have explored in this book. You will need:

- four red LEDs and one green LED, all with $220\ \Omega$ resistors, connected to A1-A4 (for the error red LEDs 0-3) and A5 (for the OK green LED)

- four buttons connected between ground and D2-D5 (D2 for button 0 and D5 for button 3)

(Note: The Arduino includes built-in pullup resistors so there is no need to use resistors to pull the buttons down to ground as in Chapter 3. However, be aware that pullup resistors mean that the button is “LOW” when pressed and “HIGH” when unpressed, which is counterintuitive.)

- a servo connected to D6 for its signal



3

You will need to use several constants (the servo positions) and variables (such as whether the box is locked or not). There are a number of new things introduced here: for example, **Locked** is a *bool* (short for Boolean), which can have either a “True” or “False” value. A concept re-introduced is arrays; **Buttons[]** is an *array* of integers and has been initialized to **{2,3,4,5}**. This means that Buttons has four slots, identified by their index number: **Buttons[0]** contains 2, **Buttons[1]** contains 3, and so on.

Note that in Computing we start counting at 0 not 1, so **Buttons[0]** is the first element of the Buttons array.

```
int ErrorLEDs[] = {A1,A2,A3,A4};  
int OKLED = A5;  
int Buttons[] = {2,3,4,5};  
int ServoSignal = 6;  
const int LockedPos = 90;  
const int UnlockedPos = 0;  
Servo Lock;  
bool Locked = true;  
int CorrectCode[] = {0,1,2,3};  
int EnteredCode[] = {-1,-1,-1,-1};  
int CurrentDigit = 0;
```

4

Using arrays allows you to loop or iterate over a group of variables. As you know how many elements are in your arrays, you can use a **for** loop.

For example, in the setup routine, you can declare an integer, p (for pin), and repeat while p is less than or equal to 3 (≤ 3), incrementing p by 1 each time around. Inside the loop, you can use p instead of the index number to access elements of the array. This allows you very quickly to set the button pins to be pullup inputs and the LED pins to be outputs. You will be using a lot more of these **for** loops in this chapter!

Of course, you also need to include the “OK” LED and servo in the setup.

```
Serial.begin(9600);
for (int p = 0;p <= 3;p++) {
    pinMode(Buttons[p],INPUT_PULLUP);
    pinMode(ErrorLEDs[p],OUTPUT);
}
pinMode(OKLED,OUTPUT);
Lock.attach(ServoSignal);
Lock.write(LockedPos);
```

PRO TIP

Arrays are a complex concept. Think of them as being like a book, with the variable name as the book name, and each indexed position as a page that can hold some information.

5

This is quite a complex project. Instead of using lots of code written multiple times in one long loop, it is sensible to separate the code into *helper routines*. Helper routines are just like the main loop, but you give each helper routine a meaningful name of its own:

- **void resetEntered()** can use a **for** loop to reset the elements of the **EnteredCode** array to -1 (-1 is used to identify nothing entered for that digit);
- **void lock()** and **void unlock()** can set the **Locked** variable to True or False, and write the correct position to the servo;
- **void clearErrorLEDs()** can use a for loop to go through the **ErrorLEDs** array and set each one to “LOW”;
- **void showCurrentDigits()** can use a **for** loop to go through the **EnteredCode** array and output each digit to the serial monitor for checking.

```
void showCurrentDigits() {
    Serial.print("Currently entered: ");
    for (int p = 0;p <= 3;p++) {
        Serial.print(EnteredCode[p]);
    }
    Serial.println("");
}
```

PRO TIP

Decomposition is the term for breaking problems down into smaller chunks that are easier to deal with—it is a fundamental concept in Computing.

6

The routine to check the code (**void checkCode()**) needs to carry out a number of tasks:

- Firstly, it has to loop through and check each element of the **EnteredCode** against the same element of **CorrectCode**.
- If any elements of the codes are different, the corresponding LED needs to be lit, and a flag set to identify that the codes are different.
- If the codes are the same, the “OK” LED must be lit for a short while and the unlock routine must be called. If the codes are not the same, then the error LEDs need to remain lit for a while, before the LEDs are reset.

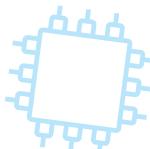
```
void checkCode() {  
    bool codeCorrect = true;  
    for (int p = 0; p <= 3; p++) {  
        if (EnteredCode[p] != CorrectCode[p]) {  
            codeCorrect = false;  
            digitalWrite(ErrorLEDs[p], HIGH);  
        }  
    }  
    resetEntered();  
    if (codeCorrect) {  
        Serial.println("Correct code!");  
        digitalWrite(OKLED, HIGH);  
        unlock();  
        delay(500);  
        digitalWrite(OKLED, LOW);  
    } else {  
        Serial.println("Incorrect code!");  
        delay(1000);  
        clearErrorLEDs();  
    }  
}
```

7

The most important routine in this code is **void ButtonPressed**, which takes a parameter, **buttonID**. A *parameter* is a piece of information that is passed into a routine for it to use—this means that you can use the same routine regardless of which button is pressed.

When a button is pressed, the corresponding LED is lit temporarily and the corresponding button number is put into the **EnteredCode** array at the **CurrentDigit** position. If the current digit is 4, then four digits have been entered. In that case, if the box is currently locked, then the **checkCode()** routine is called. If the box is not locked, then the entered code is copied one digit at a time into the correct code array, and the **lock()** routine is called.

```
void buttonPressed(int buttonID) {  
    Serial.print("Button ");  
    Serial.print(buttonID);  
    Serial.println(" pressed.");  
    digitalWrite(ErrorLEDs[buttonID], HIGH);  
    EnteredCode[CurrentDigit] = buttonID;  
    CurrentDigit++;  
    showCurrentDigits();  
    delay(500);  
    clearErrorLEDs();  
    if (CurrentDigit == 4) {  
        if (Locked) {  
            checkCode();  
        } else {  
            for (int p = 0; p <= 3; p++) {  
                CorrectCode[p] = EnteredCode[p];  
            }  
            resetEntered();  
            lock();  
        }  
        CurrentDigit = 0;  
    }  
}
```



Testing

This project involves both physical testing and software testing. Make sure you watch the serial monitor carefully, noticing what button presses are registered when the buttons are being pressed. The most common error is to forget that you must identify array elements by their index, and that indexes start at 0.

The problem with the physical lock box is that, when it is locked, you can't see inside it and fix any problems. It may be worth building the box in such a way that you can temporarily remove one of the sides or the top until you are confident the lock will work reliably and smoothly.

Stretch tasks

- In an earlier chapter you learned how to use multiple inputs on a single pin. Could you add more buttons to have the full range of digits 0–9?
- At the moment, the lock box is a normal device. Try turning it into an IoT device by logging all attempts to enter the box to the Cloud. You could also have a remote unlock button in case you forget your pin code.
- At the moment, it would take only a short time for someone to try every possible code and eventually get into the lock box. Consider what you could add to make it significantly more difficult to “hack” the box.

Final thoughts

As mentioned in “Setting the scene” above, the servo is a type of actuator. Physical devices often need to move to enable an action or display a variation. Motors are another type of actuators, and there is a wide range, such as screw jacks for producing linear motion.

Although the lock box device you built in this chapter is quite simple, the concept is used worldwide to operate delivery lockers used by companies such as Amazon. Consider the environmental benefits of using delivery lockers rather than delivering every parcel to individual addresses.

20. FINAL PROJECT

Setting the scene

The chapters in this book have introduced many new concepts, from simple inputs and outputs using buttons and LEDs, to more complex input sensors and physical outputs using motors, buzzers, and servos. Completing all these projects should give you a great sense of achievement, but now you have come to the end it is time to put all your skills into practice and complete a project of your own. You will find some ideas below, along with some helpful tips; see also <https://www.kaggle.com/>.

To give you a starting point, try to design a project that meets the following success criteria.

Success criteria

Create a device that:

- has at least two inputs and two outputs
- is an IoT device, using the Internet either for input or for displaying and logging data
- uses as many recycled products or materials as possible
- has a practical purpose, either for you or for someone you know.

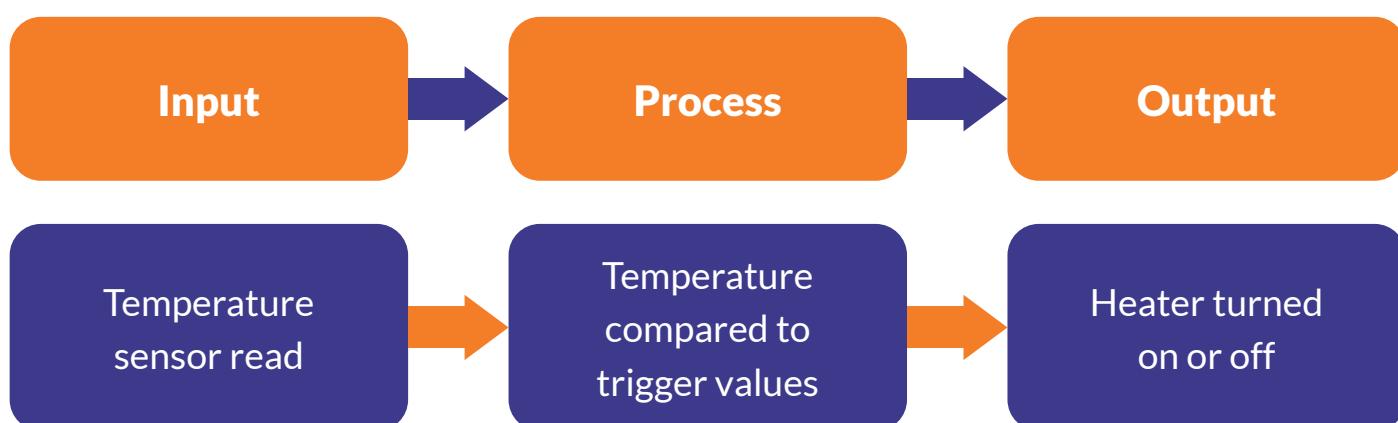
1

When you are designing your project, consider the IPO: inputs, processes, and outputs.

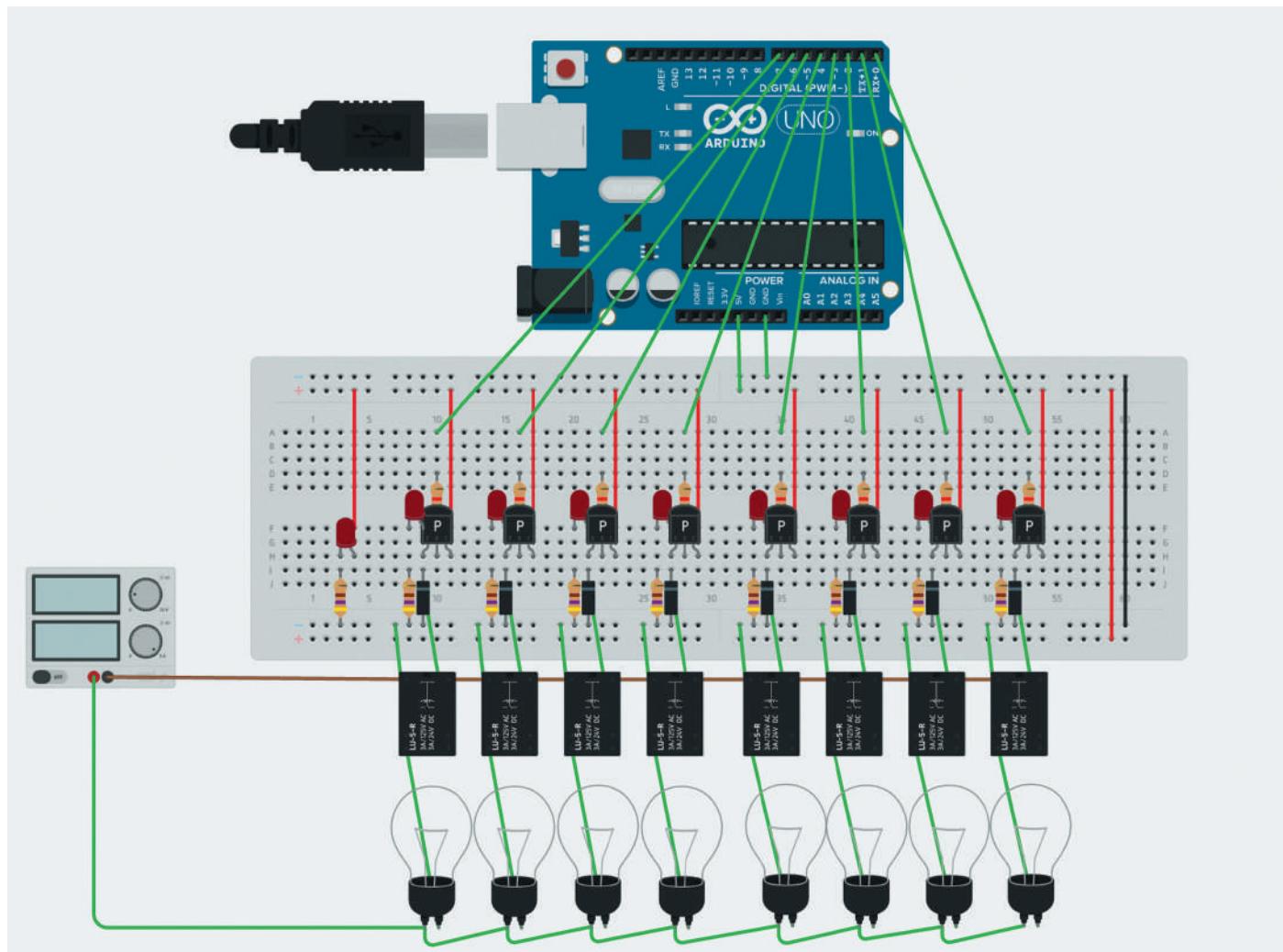
- What do you want to happen? These are the outputs.
- What will cause the outputs to happen? These are the inputs.
- In what situations will the inputs cause the outputs to happen? These are the processes.

The diagram gives an example of each.

Remember that inputs and outputs are not just buttons being pressed, LEDs lighting, or fans turning on, but also data being entered into a spreadsheet online or an Internet switch being flicked.



When you design your circuit, try the inputs and the outputs individually by echoing data to the serial monitor or sending a direct message to the output pins. Once each is working as expected, it is time to move on to the coding.



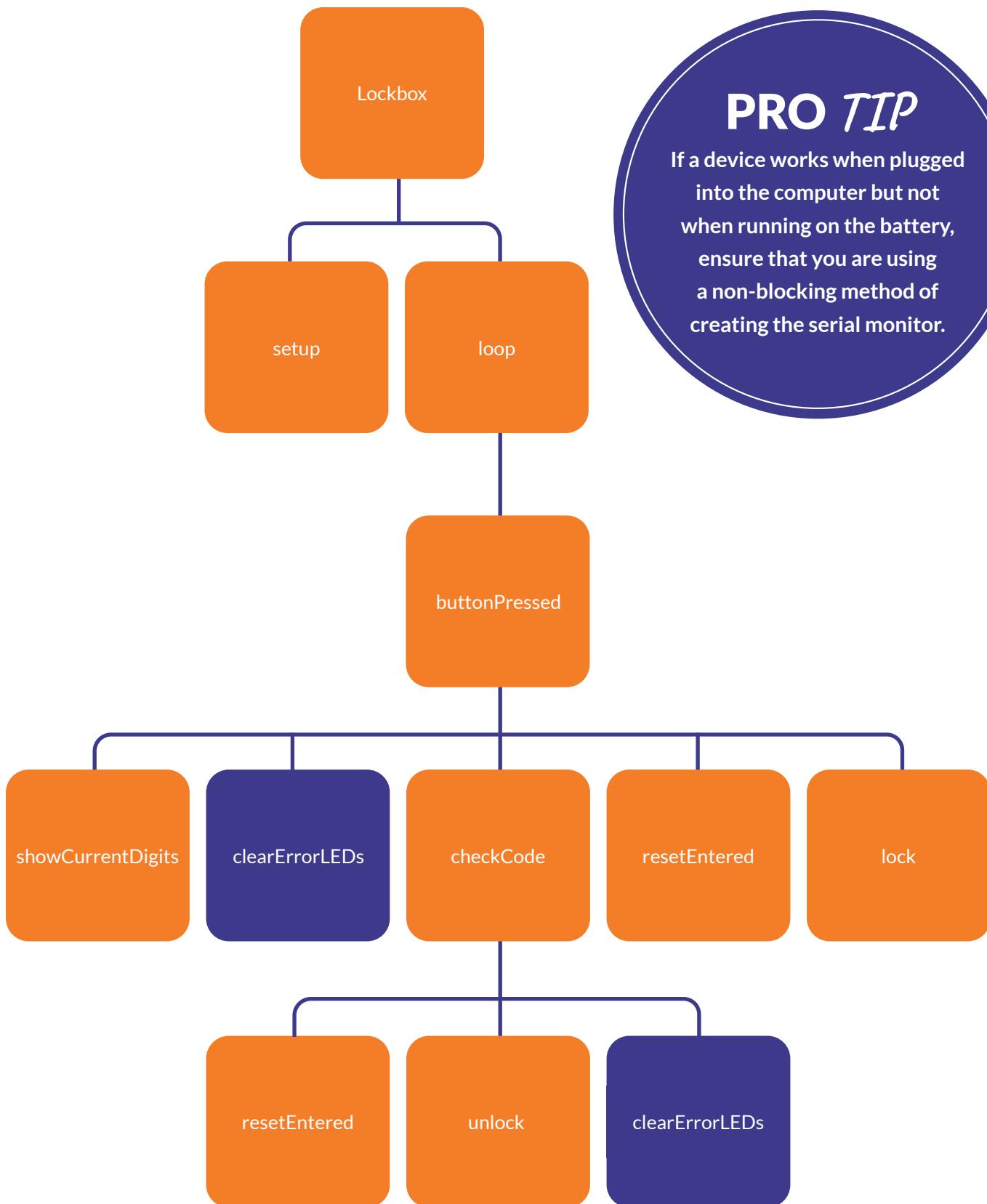
PRO TIP

Sometimes certain pins on the Arduino are withheld for different purposes. In Chapter 18, you had to move two of the LCD pins as the GPS module was using those pins. If something isn't working as expected, try moving the code and the connection to a different pin.



3

Decomposition is the programmer's best friend. One method of decomposition is *top-down design*. This is a hierarchical design methodology. At the top is your main program. Each "thing" the program needs to carry out becomes a sub-routine of its own below the top level. If a sub-routine becomes complex, split it again into more sub-routines until each sub-routine is simple enough to program.



PRO TIP

If a device works when plugged into the computer but not when running on the battery, ensure that you are using a non-blocking method of creating the serial monitor.

When you are programming the designed solution, program one piece at a time. In Chapter 19, you started by making the lock work, then the buttons and indicator LEDs, then the comparison of codes, and so on. You should have tested each section of code fully before moving on, and you may have saved the code using a different version number at each point. Using version numbers means that, if something goes wrong, you can easily find the last code that you know worked OK.

Testing

Formal testing takes some time. For each possible input, you need carry out the following tests:

- “Normal” tests involve inputs that should work—pressing a button, for example.
- “Erroneous” tests involve inputs that should not work—for example (in normal programming) typing a letter when the program expects a number; (in physical computing) **not** pressing a button when the program expects you to.
- “Edge cases” involve inputs that should (or should not) work, but only just—pressing two buttons at once or pressing a button really fast or really slowly, for example.
- The most important thing is to be thorough. Test each possible input and fix any problems.

Stretch tasks

- Creating a device once is an interesting project for you. Creating a device that can be reproduced could be useful for many people, so consider writing up your project to share with the wider world. [Instructables.com](#) is a popular website for makers to share their project stories.
- Once your project is complete, consider what extensions you could add to it. If your project involves logging data, can you analyze the data using either spreadsheet tools or Machine Learning techniques?
- Is your device fully IoT enabled? Does it transmit data both to and from the device? If not, add this as an extension.
- Part of documenting a project is completing a full circuit diagram. Draw up your circuit as a diagram using the proper symbols. You might like to use a site such as [circuit-diagram.org](#) to help you.

Final thoughts

The use of microcontrollers, and the IoT in particular, is a hugely growing field. Many homes now have several Internet-enabled devices and they may have many more microcontroller-based devices that are not (yet) online. The projects in this book have given you a taster of many concepts and encouraged you to continue exploring. Now it is over to you to dream big, solve problems, and use the skills you have developed to come up with new concepts to benefit the world around you.

