



An Introduction to Arduino for Transition Year Students

David O'Callaghan

Caoimhe Conlon

Contents

Introduction.....	2
Day 1	5
Project 1 - Your First Project	5
Project 2 - Improving your first project	9
Project 3 – Traffic Lights	11
Project 4 - Knightrider.....	12
Day 2.....	14
Project 5 – More than just 1s and 0s	14
Project 6 – Analog vs Digital	15
Project 7 – The Dimmer Switch.....	16
Project 8 – The Dimmer Switch – in code!.....	17
Day 3	18
Project 9 - Morse Code Generator.....	18
Project 10 - Randomiser.....	20
Project 11 – Sensor LED	21
Day 4.....	22
Piano Time!	22
Who needs more buttons?	23
Helpful Resources	26
Try Yourself Circuits	27

Introduction

Arduino is a platform for fun electronics projects based on easy-to-use hardware and software.

Arduino boards are able to read inputs (for example, how much light is on a sensor or whether or not a button is pushed) and turn them into outputs (for example, turning on a light, activating a motor or even sending a tweet).

The board that you have is called an Arduino UNO (see *Figure 1***Error! Reference source not found.**). We will use this board along with some electronics components and a breadboard to go through some cool projects.

Here is a link to a good video with more information on the Arduino UNO board:

<https://www.youtube.com/watch?v=09zfRaLEasY>

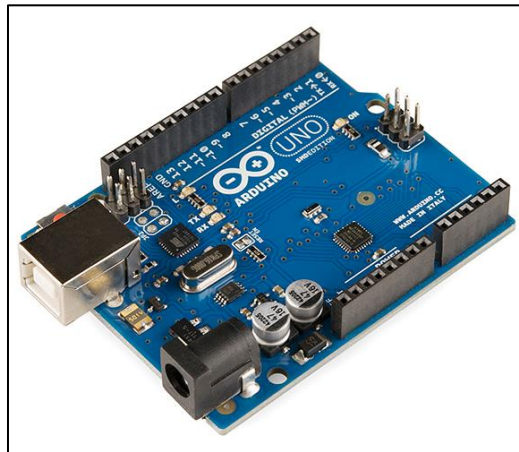


Figure 1: Arduino UNO, the most popular model of Arduino boards

A breadboard is a board used for building circuits without the need for soldering (like welding) components together (see *Figure 2*). This makes building circuits quick and easy!

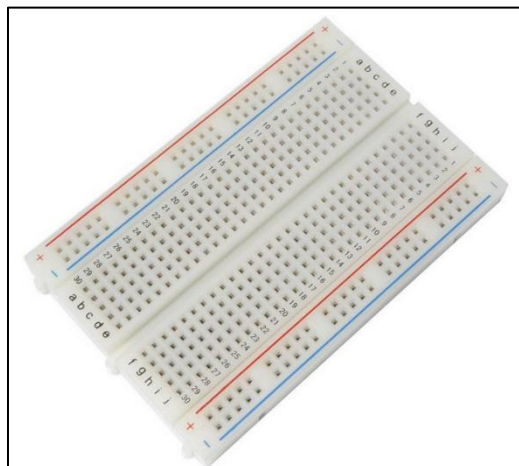


Figure 2: A 400 point breadboard

How a breadboard works

So, you're probably wondering how all of those holes help you build a circuit... Well, the holes in each horizontal row are connected internally within the breadboard. These rows are referred to as the terminal strips.

Similarly, the holes in each vertical column are connected internally. These columns are referred to as the power rails. This is illustrated in *Figure 3* below.

As an example, if you were to insert a wire into the hole at position *b2*, another wire could be inserted into any one of the holes at positions *a2*, *c2*, *d2*, or *e2*, and these wires would be electrically connected. Note that this is assuming that the labelling of rows and columns are the same as those in *Figure 3*.

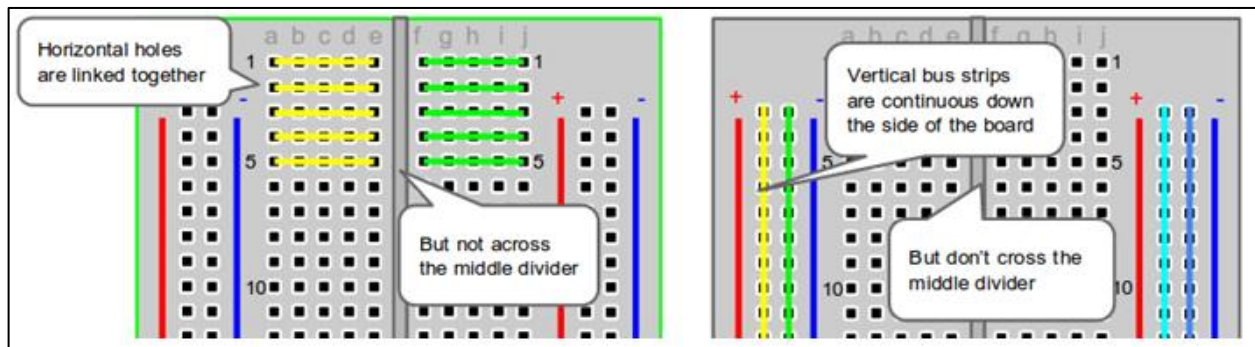








Figure 3: Visualising the strips in a breadboard

A useful resource for more information about breadboards is at:

<https://learn.sparkfun.com/tutorials/how-to-use-a-breadboard>

Circuit Components

	Resistor
	LED (Light Emitting Diode)
	Push button
	Jump wires
	Potentiometer
	Piezo buzzer

Day 1

Project 1 - Your First Project

So... time to get started! Connect your Arduino UNO to a USB port in the computer and then open *Device Manager* from the start menu as shown in the left half of *Figure 4*. Then under the title *Ports (COM & LPT)*, check the number next to the COM illustrated by the red arrow in the right half of *Figure 4*. In this example it's COM7, but yours might be different. It may also change depending on which USB port you connect into, so it's worthwhile checking each time you connect the Arduino.

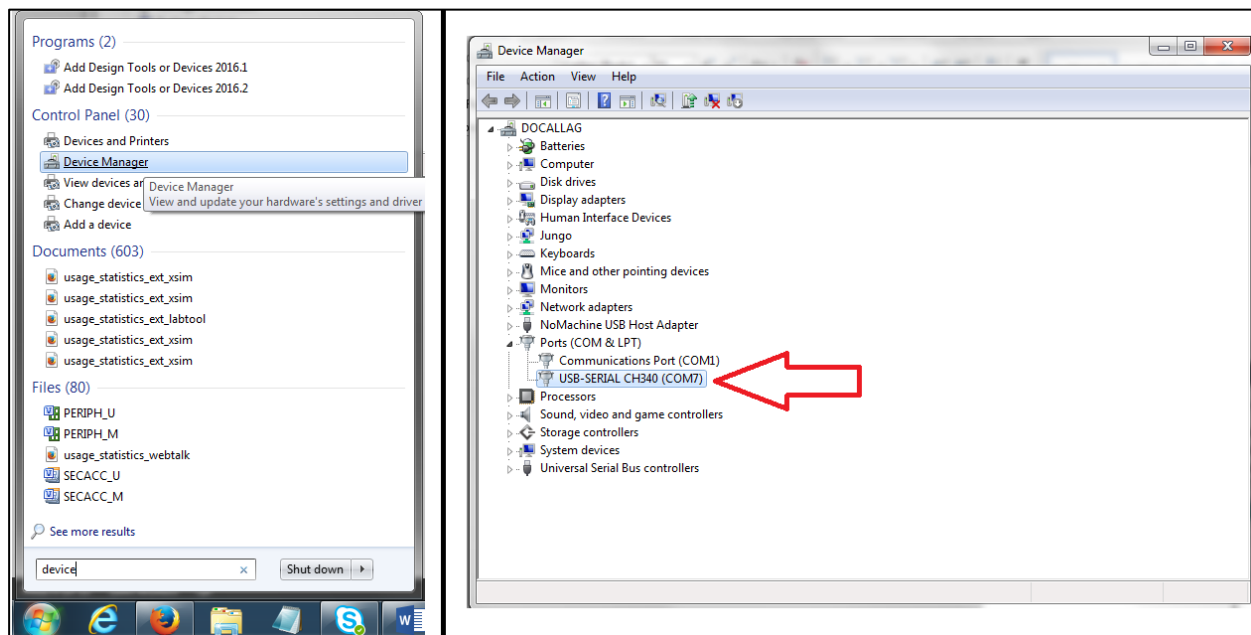


Figure 4: Finding the COM number in Device Manager

Now, open up *Arduino IDE* from the start menu. A window like the one shown in *Figure 5* should appear. We will call this window a 'sketch'. This is where we will write our code and then download it onto the Arduino itself so it can run it.

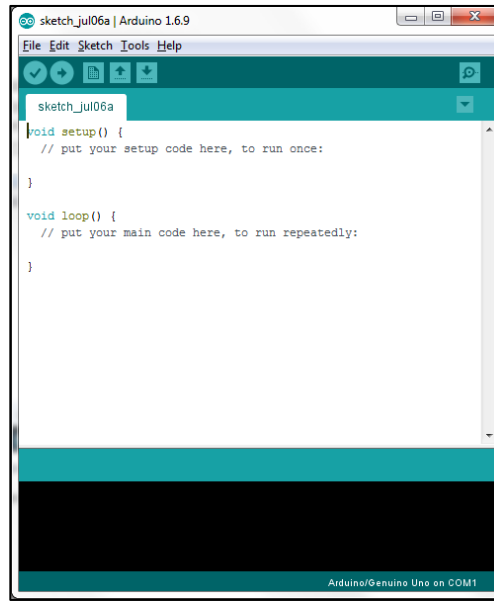


Figure 5: Arduino IDE

Under the *Tools* tab at the top of the window, check that the port is set to the same COM number that you found earlier in *Device Manager*. Now we are ready to start coding (start writing programs)!

You should see some code already written in the sketch window. Lines with the double forward slash at the beginning ('//') are called 'comments' and they do not affect the code itself. They are just used to explain how the code works to make it easier for other people to understand your code and also for you to remind yourself!

Go to the folder labelled *Arduino Sketches* on your desktop (see **Error! Reference source not found.**Figure 6) and open up the sketch labelled *Blink_LED*. A new sketch window should appear with the code shown in Figure 7. Read the comments in the code and see if you understand it (it's okay if you don't!).



Figure 6: Folder on desktop

Now it's time to build the circuit on the breadboard before doing anything with this code.

Set up the circuit as shown in Figure 8. Make sure that the longer leg of the LED is connected to pin 13 and not the shorter one. It is good practice to colour code your wires. For example, always use red for the power supply and blue for ground. This way it is easier to fix your circuit if you make a mistake!

```
Blink_LED
1 /*
2  Blink
3  This program turns on an LED on for one second, then off for one second, repeatedly.
4  */
5
6 // the setup function runs once when you press reset or power the board
7
8 void setup() {
9   // initialize digital pin 13 as an output.
10   pinMode(13, OUTPUT);
11 }
12
13
14 // the loop function runs over and over again forever
15 void loop() {
16   digitalWrite(13, HIGH); // turn the LED on (HIGH is the voltage level)
17   delay(1000);             // wait for a second (1000ms)
18   digitalWrite(13, LOW);  // turn the LED off by making the voltage LOW
19   delay(1000);            // wait for a second
20 }
21
```

Figure 7: Code from 'Blink_LED' sketch

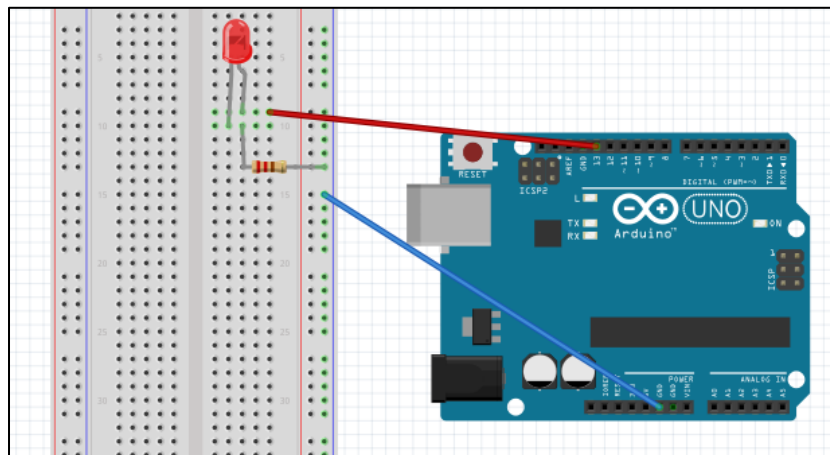


Figure 8: Setup of circuit of 'Blink_LED' sketch

Once the circuit is set up, verify the code by clicking the button with the ‘tick’ symbol at the top of your sketch. This will test your code for any errors. Once you see that there are no errors, upload your code to your Arduino board by clicking the button to the right of the ‘verify’ button (see *Figure 9* for a guide to the buttons).

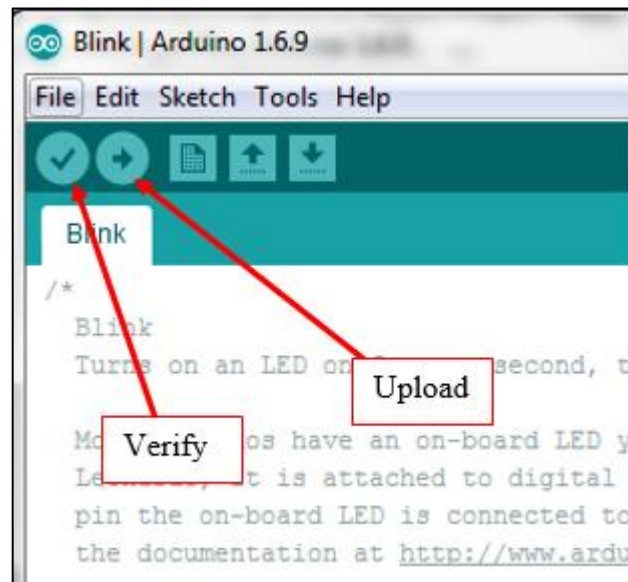


Figure 9: How to verify and upload your code

For you to try:

Now you should see the LED on your breadboard start to blink. Try changing how long the LED is on or off by changing the number inside `delay()` in your code. Don't forget you will need to upload your code to the Arduino board again!

Project 2 - Improving your first project

Here we are going to add a push button into the circuit so that you can use it to turn on the LED whenever you wish. The Arduino will detect the state of the button (whether it is pushed or not) and then send a high (turns the LED on) or low (turns the LED off) signal depending on the state. Make the changes from the circuit in project 1 shown below in *Figure 10*.

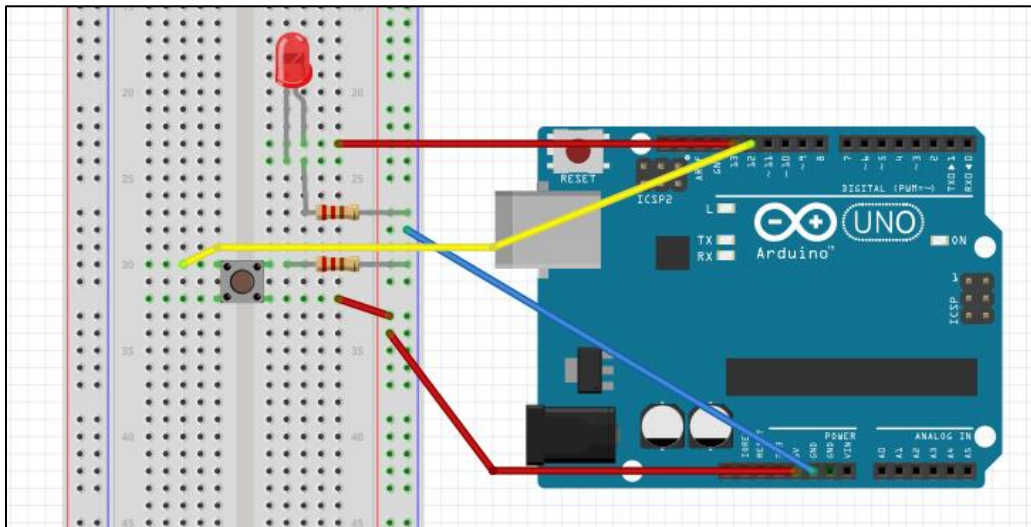


Figure 10: Circuit for Button and LED

We also need to make a few adjustments to our code. So, open up the sketch labelled *Button_and_LED* from the *Arduino Sketches* folder. Now read through the comments in the code again and see if you understand the changes (again, it's okay if you don't!).

The next step is to upload the sketch to the Arduino in the same manner as before (refer back to project 1 if you have forgotten).

For you to try:

Try changing the code so that the LED will be off when the button is pushed and on when the button is not pushed.

Hint: To do this all we will have to do is make a small change to the *if statement*, `if (buttonState == 1)`. Here the program runs the code after the *if statement* if the button is pressed (or the state is 1). Change the number so that it runs this code if it is not pressed (or the state is 0).

```
Button_and_LED
1 /*
2   Button_and_LED
3   This program turns on an LED on whenever the user pushes a button
4 */
5
6 // This time we will define the pins as words so we don't have to remember the numbers
7 int button = 12;    // we initialize pin 2 as the button
8 int led = 13;       // and pin 3 as the LED
9
10 // the setup function runs once when you press reset or power the board
11 void setup() {
12   pinMode(button, INPUT);    // the button is an input (we are 'reading' the signal from it)
13   pinMode(led, OUTPUT);      // and the LED is an output (we are 'writing' a signal to it)
14 }
15
16 // the loop function runs over and over again forever
17 void loop() {
18   int buttonState = digitalRead(button);    // this reads the state (or value) of the button
19   // if it's pressed it will be 1 or HIGH
20   // if not it will be 0 or LOW
21
22   if (buttonState == 1) {    // now the code checks if the button is pressed (1)
23     digitalWrite(led, 1);    // if so the LED turns on (note that 1 is the same as HIGH)
24   }
25   else {    // otherwise
26     digitalWrite(led, 0);    // the LED turns off (note that 0 is the same as LOW)
27   }
28   delay(1);    // the program then waits 1 ms before checking the button again
29 }
30
```

Figure 11: Code for Button and LED sketch

Project 3 – Traffic Lights

Now that you have the hang of the first projects, let's try adding some more LEDs to the board and getting them to work in a traffic light pattern.

Open up the sketch labelled *Traffic_Lights* in the *Arduino Sketches* folder. Read through the comments to understand what is happening in the code.

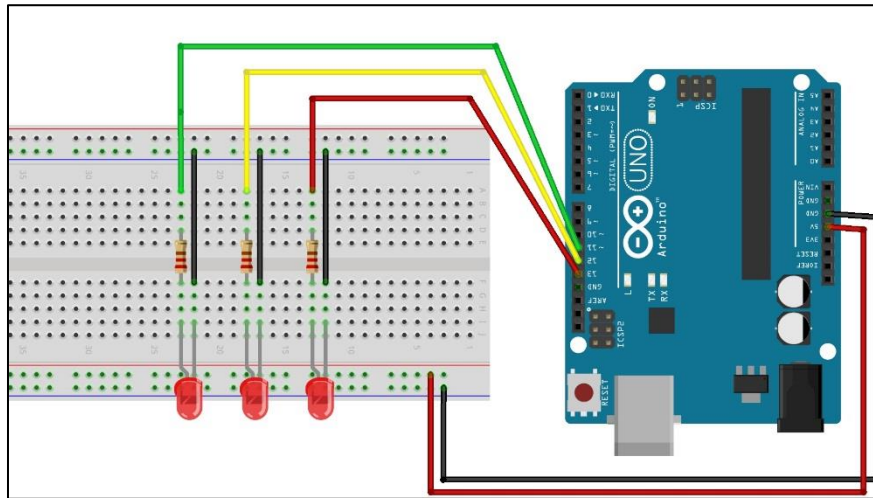


Figure 12: Circuit for Traffic Lights

Once you have connected up the circuit, verified the design and uploaded it to the board as before the LEDs should start to go through the traffic light pattern.

For you to try:

Edit the code to change the order of the lights, the patterns and the length of the delay on each light(s).

Can you add a switch to start the process? Similarly to project 2, you will need to change the code and the circuit to make this work.

Project 4 - Knightrider

This project adds some more LEDs to the design, and works similarly to the Blink project. The light passes over and back along the LEDs, by switching them on one at a time, ensuring the remainder stay off.

Open up the sketch labelled *Knightrider* in the *Arduino Sketches* folder. Read through the comments to understand what is happening in the code.

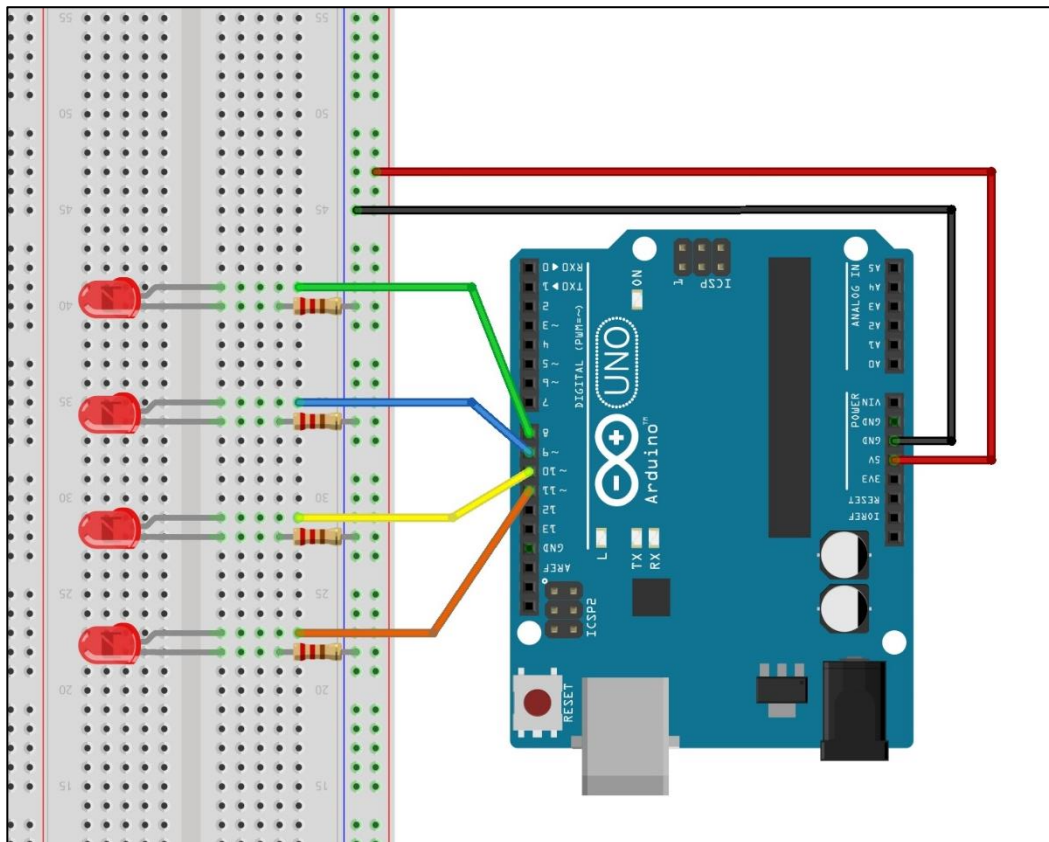


Figure 13: Circuit for Knightrider

Once you have connected up the circuit, verified the design and uploaded it to the board, the lights should start to flash.

For you to try:

Now let's try adding some extra elements. Firstly, the delay instruction is repeated many times, so if we wanted to change the length of each blink it would take lots of time, particularly if we added extra LEDs. So instead of writing the delay value each we can create a variable especially for this purpose, "timer", and give it a value. On the line after `int pin11 = 11;` insert `int timer = 150;`. Don't forget the semi-colon (;) or your sketch won't compile. Now every time there is a (150), replace it with (timer). Once you verify and upload the sketch the LEDs should behave exactly like before. Try changing the value in the timer declaration to see how that impacts the light routine. Now you only need to change one line to change all the delays, much handier than having to do them individually. There is an extra file named *Knightrider_Timer* in the sketches folder which has a working example of this.

Next try adding in some extra LEDs, the circuit and the sketch will need to be changed for this. For each LED added to the board you'll need a resistor and connecting wire to the Arduino. For the sketch you'll need to declare each pin under the other 4 in the section commented `//declare each LED here` (make sure you choose the pins that you just connected the new LEDs to or it won't work properly), eg if you connected an LED to pin 12 you would type `int pin12 = 12;`. Next, set each pin as an output in the setup section, similar to the pins that are already set up. Finally you'll need to add the extra LEDs to the blink routine, make sure to keep them in the right order.

Remember that all the new lines that are added need to finish in a semi-colon (;) or the sketch won't compile!

Day 2

Project 5 – More than just 1s and 0s

You may have noticed that so far that we have only looked at devices with only two states. For instance the LED could only be on or off, and the push button could only be pressed or not pressed. Anything with only two states can be represented by a 1 or a 0. This is the fundamental building block to Digital Electronics.

But what if we want to vary the brightness of the LED or maybe read a value other than just a 1 or a 0? In this project we will start to look at Analog Electronics as opposed to Digital.

Set up the circuit as shown below. Note that the device on the breadboard is a potentiometer which may look a bit different to your one. However, the setup is the same. Just make sure the three pins of the potentiometer are all in different horizontal lines on the breadboard.

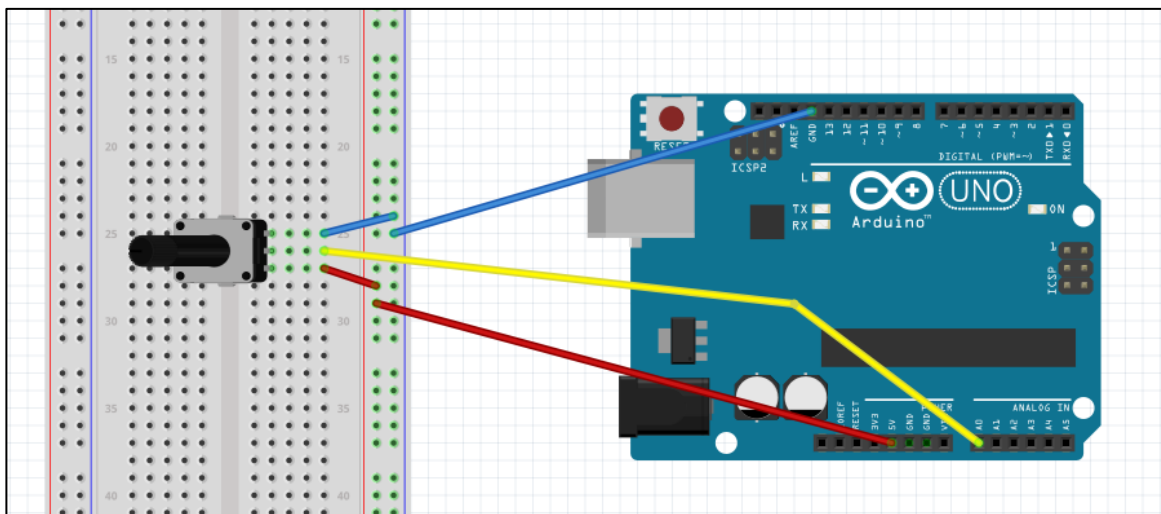


Figure 14: Circuit for Potentiometer

Now, open up the sketch labelled *Potentiometer* in the *Arduino Sketches* folder. As usual, read through the comments and see if you understand much of the code.

Verify and upload the sketch to the Arduino and then open the serial monitor by clicking on the magnifying glass symbol in the top right hand corner of the sketch window (see *Figure 15*).

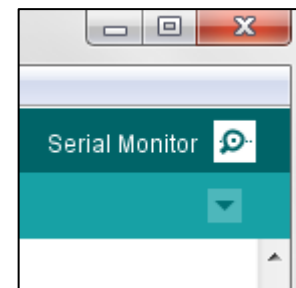


Figure 15: Serial Monitor Tab

Turn the rotor on the potentiometer and observe the change in analog reading on the serial monitor. What's the highest number that will appear? Do you know why this is the maximum value? Google it if you are interested but we will not be covering the reasons here.

You can also open the serial plotter from the *Tools* tab if you want a more graphical representation of the analog reading.

Project 6 – Analog vs Digital

This project might not be as fun as the others (because all of these are super fun) but it is here to help you gain a better understanding of the difference between analog reading and digital readings.

So, set up the circuit below (note that the only difference between this and the circuit in Project 3 is the green wire added to take a digital reading from the potentiometer).

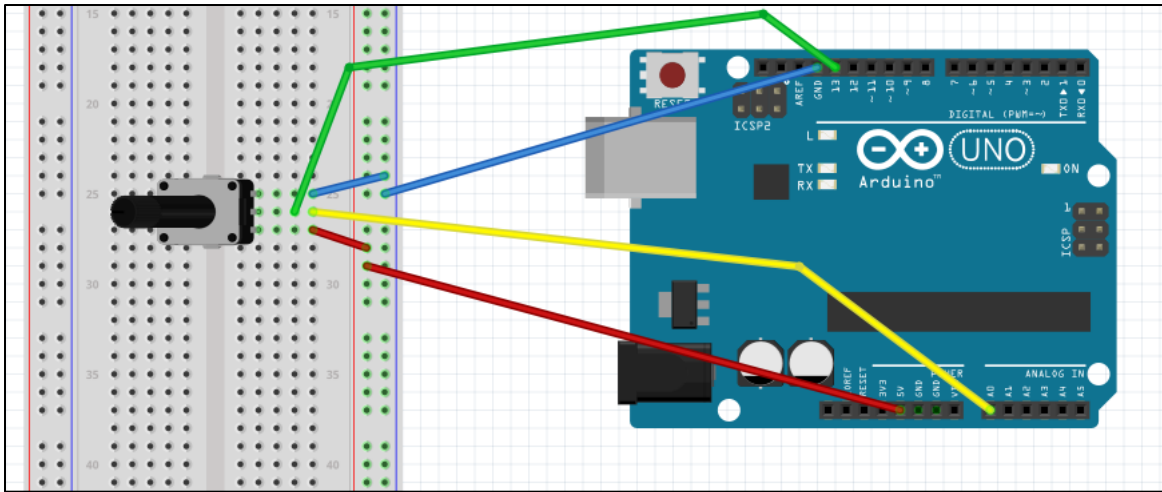


Figure 16: Circuit for Analog vs Digital

Now, open up the sketch called *Analog_vs_Digital* from the *Arduino Sketches* folder. Read through the comments and try to predict what we will see on the serial monitor after the code is uploaded to the Arduino.

Verify the sketch and upload it. Open the serial monitor (a quick way to do this is Ctrl + Shift + M) and observe the changes when you turn the rotor on the potentiometer. Around what analog value does the digital reading change from a 0 to 1?

Project 7 – The Dimmer Switch

Now we are going to tweak our circuit and code from projects 3 and 4 to vary the brightness of an LED using a potentiometer.

Set up the circuit as shown below and open the sketch called *Dimmer_LED* from the *Arduino Sketches* folder. Remember that the longer leg of the LED should be connected to the pin which it is receiving the signal from which is pin 11 in this case (the red wire in the diagram). Verify and upload your code and then turn the rotor on the potentiometer and observe the change in brightness of the LED.

What the `analogWrite()` command actually does is that it turns the LED on (for percentage of time depending on the value between 0 and 255 called in the command) and then off again 500 times per second. This means that the LED isn't actually dimmer, our brain just thinks it is because it is going on and off so fast. For more information (and probably a better explanation) check out the following link:

<https://www.arduino.cc/en/Tutorial/PWM>

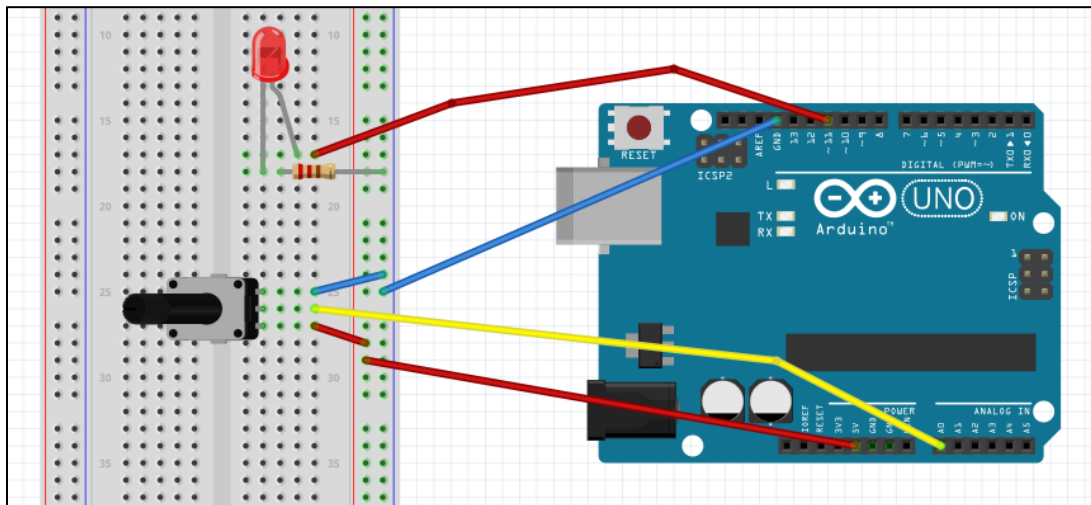


Figure 17: Circuit for Dimmer LED

Project 8 – The Dimmer Switch – in code!

Instead of having to use hardware (switch) and software (the sketch) to create the dimming effect we can use just software. This is what is happening to most devices now, e.g. your phone manufacturer can improve your phone with software updates rather than you having to buy a new phone every time you want new features.

Open up the sketch labelled *Fade* in the *Arduino Sketches* folder. Read through the comments to understand what is happening in the code. This code uses a *for loop*, this is simply a way of repeating a few lines of code many times (here counting from 0 to 255) without you having to write everything out that number of times. Anything which makes for less lines of code is a good thing!

The circuit is exactly the same as the circuit you used in Project 1. Compile and upload it as usual. You should see the LED gradually turning on and off without you having to do anything.

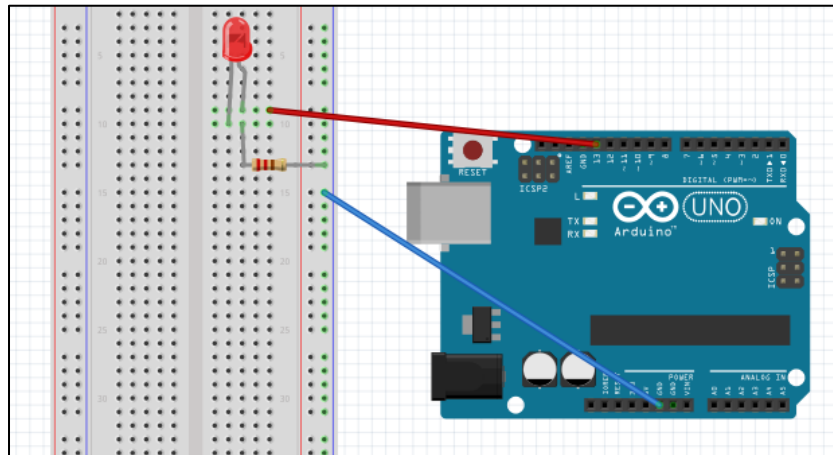


Figure 18: Circuit for Fade

For you to try:

Change the length of the delay in each loop or get rid of them altogether (you can use a comment for this `//delay(200)`, in case you want to put it back in). What effect does it have on the LED?

Next try changing the step size of the brightness. This example increases the brightness by one step on each loop (that's what the `++` stands for). If you want it to increase by 5 on each loop edit the *for loop* to look like this:

```
for (brightness = 0; brightness <= 255; brightness = brightness +5)
```

If you have time try adding a button to turn the light on or off.

Day 3

Project 9 - Morse Code Generator

This project enables you to send messages by Morse code. Again we'll use LEDs to be our signal. Try connecting the circuit without looking at the diagram, you'll need 3 LEDs and 3 resistors. Connect to pins 7, 8 and 9 on the Arduino. (Hint: it's the same idea as the Traffic Lights circuit. The diagram is on the next page if you need it.)

Open up the sketch labelled *Morse_Code* in the *Arduino Sketches* folder. Read through the comments to understand what is happening in the code. It uses a *for loop* like in the fade example. It also contains a *switch statement*. We have seen *if statements* when using a button, and this is somewhat similar. The main difference is that it checks for each character, which is written like this 'a', 'b' etc. Once it finds the character it's looking for it carries out the code for that character. The switch statement saves us from writing lots of *if/else statements*. It also involves a function, this is where the main program calls a function that sits outside of the `main void loop()`. Sometimes, as it does here, it sends some information to the function to be operated on. In our case it sends the current letter in the phrase. Remember to verify and upload the sketch as usual.

For you to try:

First off, an easy one, change the phrase to be sent out in Morse code. Next change the length of the dots (`dotLen`) and dashes (`dashLen`), so the code works faster or slower.

Next we'll try generating a sound as well as the lights. For this we'll replace one of the LEDs with a buzzer. Take out the LED connected to pin 8 and put in the buzzer, being careful to make sure the positive pin is connected the right way around. You'll need to make quite a few changes to the code. Every time LED3 is mentioned you'll need to comment out that line with `//`. Then uncomment all the lines with `audio8` on them, and the `int note` line, they are all marked to uncomment in the sketch. They will be in a number of places: the setting up of the variables, the setup function and the dot, dash and turn off functions.

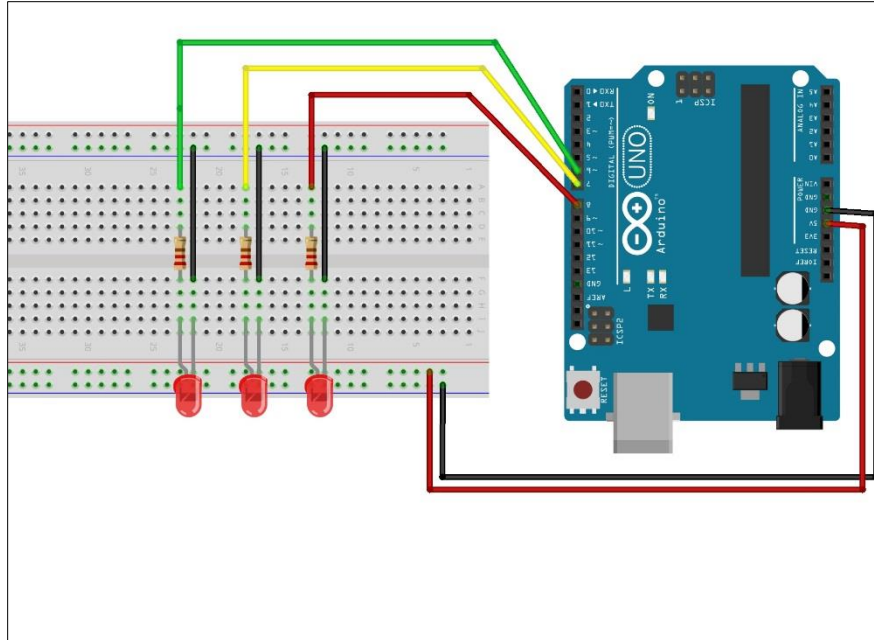


Figure 19: Circuit for Morse Code LEDs

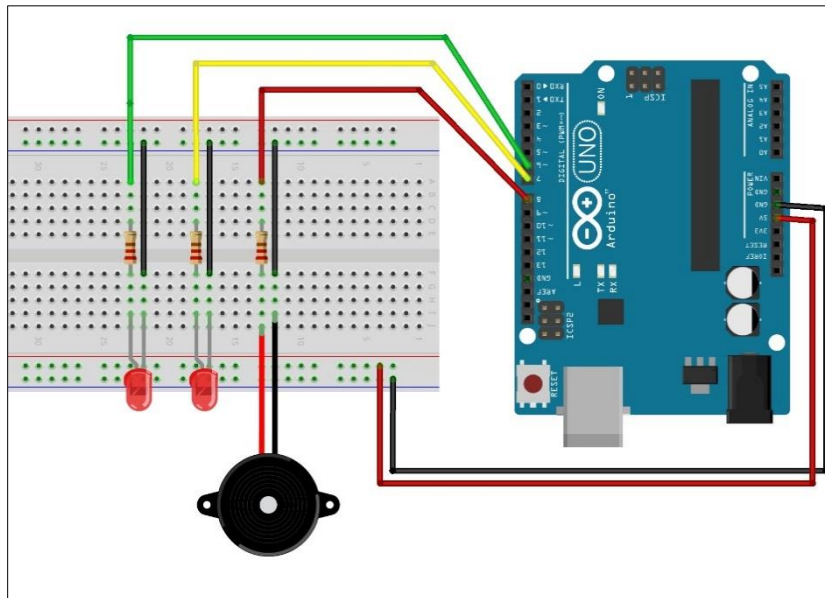


Figure 20: Circuit for Morse Code with Buzzer

Project 10 - Randomiser

This project randomly lights one of six LEDs in turn, starting fast and reducing in speed as it goes through the code. It uses the randomiser function available to the Arduino.

All the LEDs are connected together so be careful of this when making up your circuit, and make sure to add in the resistor. The diagram is below.

Open up the sketch labelled *Randomiser* in the *Arduino Sketches* folder. Read through the comments to see if you can understand what is happening in the code. This one is a little tricky. Remember to verify and upload the sketch as usual.

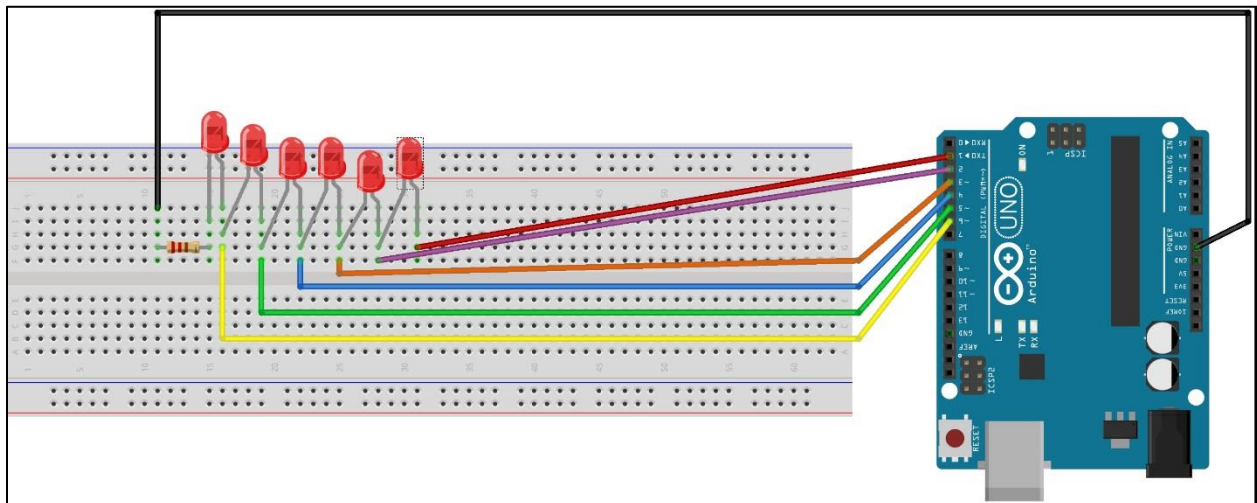


Figure 21: Circuit for Randomiser

For you to try:

Can you add or remove LEDs from the project?

Can you add a button to the project to repeat the process every time the button is pressed? As usual you'll need hardware and software to make this change.

Hints:

Hardware : Extra LED or LEDs depending on how brave you are!

Software : Add the LED(s) in the setup as a pin number and an output.

Be careful with the random number generator.

Project 11 – Sensor LED

Up until now we have been using the LED as an output, but now we're going to use it as an input. This one is also a little complicated to understand as you need to know a little about LEDs, but even if you don't understand every aspect, it's fun to play around with! The circuit contains three LEDs, you're going to need two different colours for this one. Two of the LEDs act as outputs and flash on and off (these will need to be different colours) and the third acts as a sensor to detect how much light is coming from each of the other LEDs. The light level is then going to be seen on the serial monitor, check which one gives a stronger light level.

The circuit is below and you'll need to open up the serial monitor like in Project 5. The sensor LED is the one at the top of the diagram on its own. Be careful when setting up the circuit that you have the connections to the sensor LED the right way around.

Open up the sketch labelled *LED_Sensor* in the *Arduino Sketches* folder. Read through the comments to see if you can understand what is happening in the code. Remember to verify and upload the sketch as usual. When running this project it might help to have some darkness around the board so the sensor LED senses the light from the other LEDs rather than the room, try under the desk!

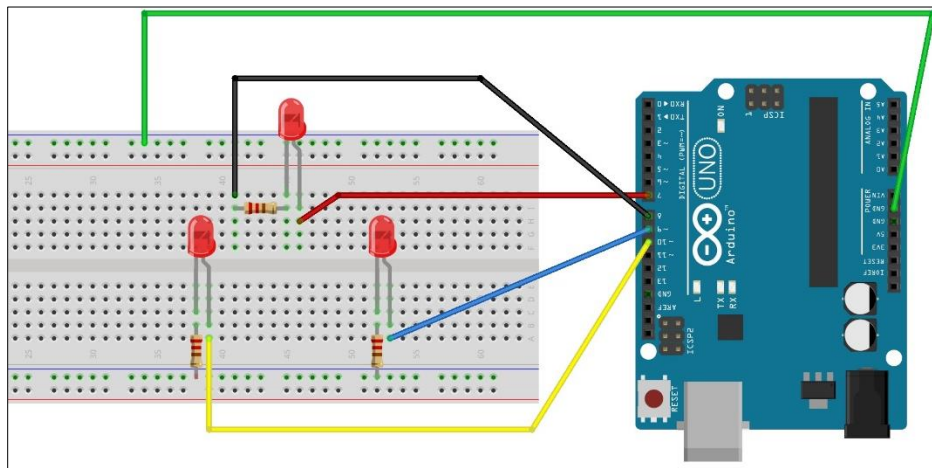


Figure 22: Circuit for Sensor LED

For you to try:

Try changing how fast the output LEDs go from dark to light. Can you figure out where this happens?

Day 4

Piano Time!

Now we are going to design a simple one octave piano keyboard using push buttons and a buzzer. In this project, the Arduino will be reading the signal from 8 different buttons on the breadboard. If a high signal is received from one of the buttons, the Arduino will send a signal to the buzzer with a frequency depending on which button was pushed.

Sound easy, right? Good. Open up the sketch called *Piano1* from the *Arduino Sketches* folder. Although the code here is quite short, it might be difficult to get your head around. Try reading through it if you want and see if you recognize anything similar to what we have coded before.

The line up the top that says `#include 'pitches.h'` just means that we want information from another file. This file contains frequencies for each of the notes. Just think of it as instead of googling the frequency of each note, we can just use this file.

Set up the circuit as shown in *Figure 23* below (it is recommended that you don't power the Arduino while wiring). Reconnect the Arduino to the PC using the USB cable, verify, upload and start playing your piano!

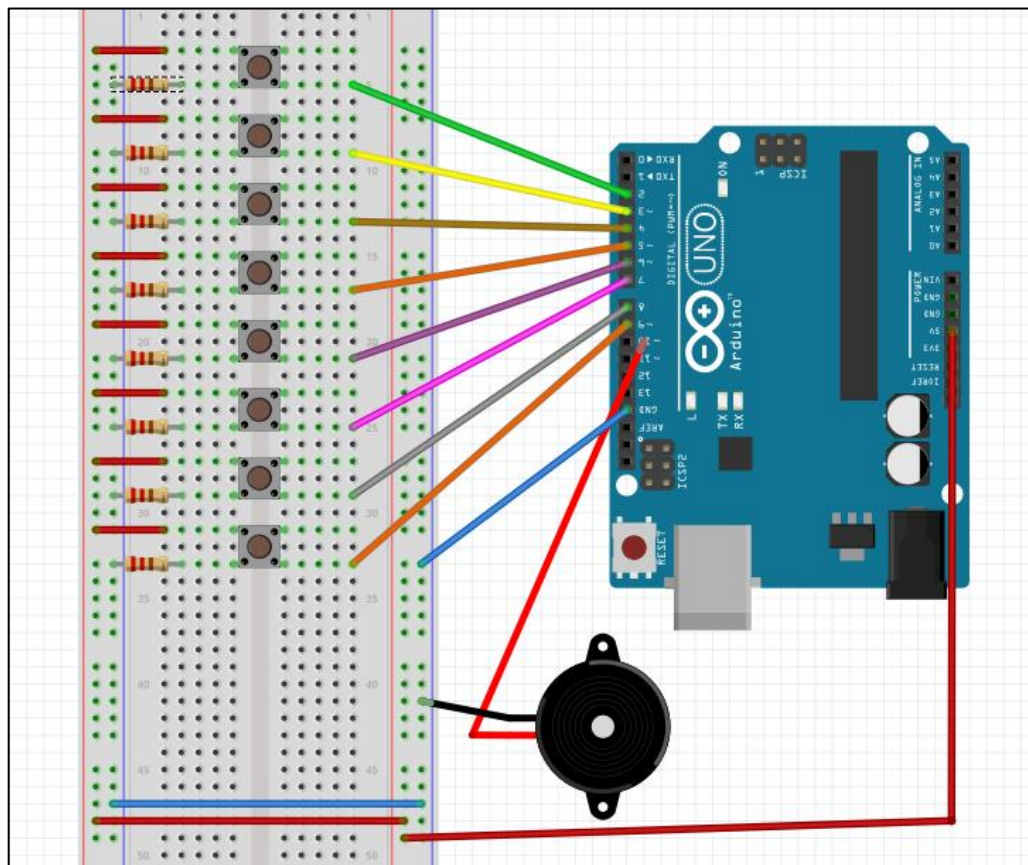


Figure 23: Circuit for Piano

Who needs more buttons?

Let's make a few changes to our piano to make it even cooler (I know what you're thinking: "I didn't think it could get any cooler". Well... it can). Make the changes to your piano circuit that you see in *Figure 24*. Here we're going to use a potentiometer to change the octave of our piano. The LEDs will indicate which of the two octaves you are in.

Open the sketch called *Piano2* from the *Arduino Sketches* folder. Read through it and notice the changes. With only a few slight tweaks we can double the amount of notes of our piano! Verify and upload the code and start playing, turning the potentiometer to change the octave.

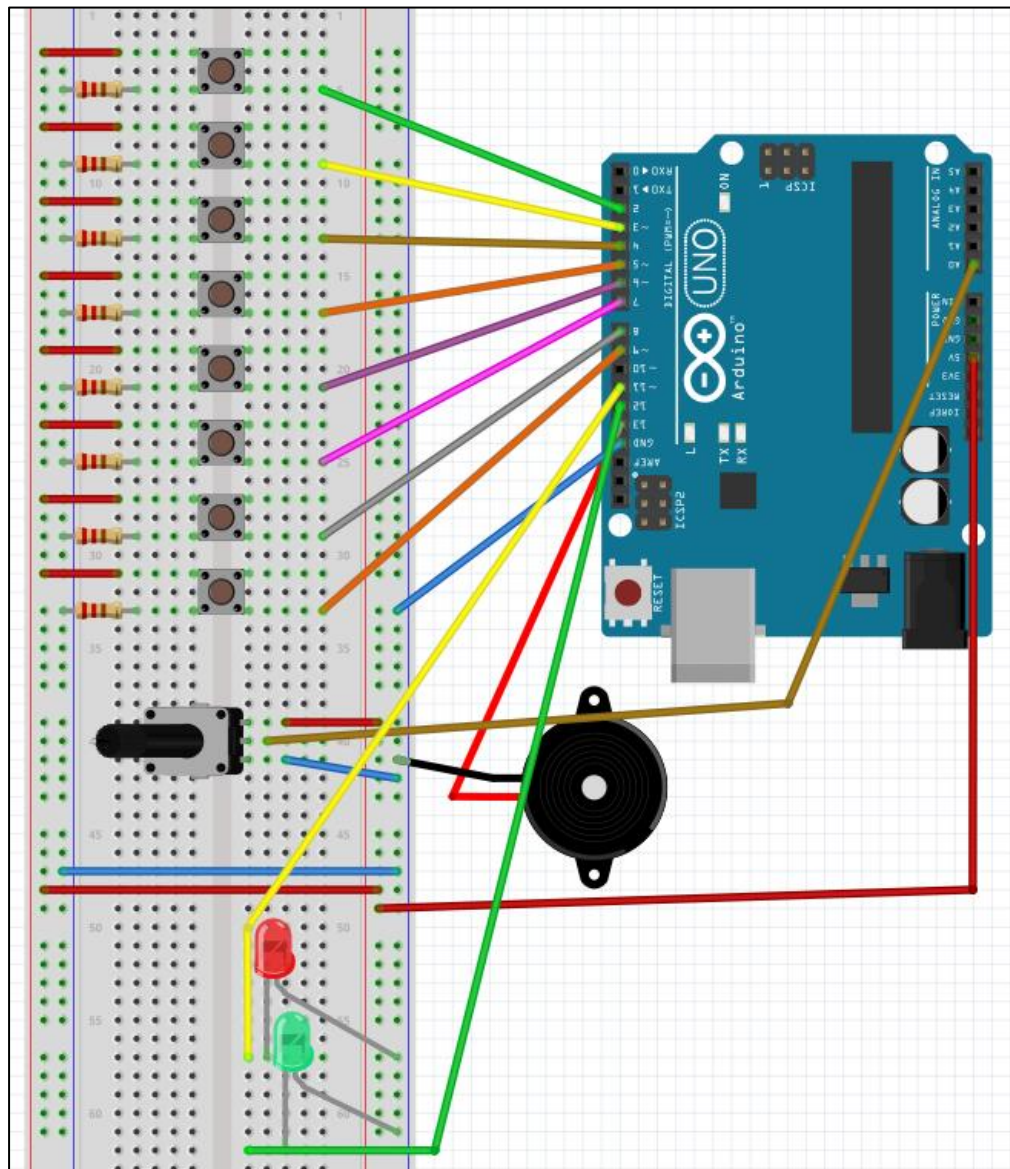


Figure 24: Enhanced piano circuit

For you to try:

Now it's your time to shine. See if you can start a new sketch and just write a program to play a song of your choice. You won't need any push buttons, resistors, LEDs or potentiometers. Just a buzzer.

There's a template below to give you a few hints but that's all you're getting!

Make sure to copy and paste the *pitches.h* file from either the *Piano1* or *Piano2* folder into the same folder as your sketch here!

```
1 int speaker = 13;
2
3 #include "pitches.h"
4
5 void setup()
6 {
7   pinMode(speaker, OUTPUT); // Buzzer
8 }
9
10 void loop()
11 {
12   tone(speaker, NOTE_C4); // Play the note, C
13   delay(240);             // for 240 ms
14   tone(speaker, 0);        // Play nothing
15   delay(10);              // for 10 ms
16
17   tone(speaker, NOTE_D4); // Play the note, D
18   delay(240);             // for 240 ms
19   tone(speaker, 0);        // Play nothing
20   delay(10);              // for 10 ms
21
22   tone(speaker, NOTE_E4); // Play the note, E
23   delay(490);             // for 240 ms
24   tone(speaker, 0);        // Play nothing
25   delay(10);              // for 10 ms
26 }
27
```

Figure 25: Sample code for programming the Arduino to play a piece

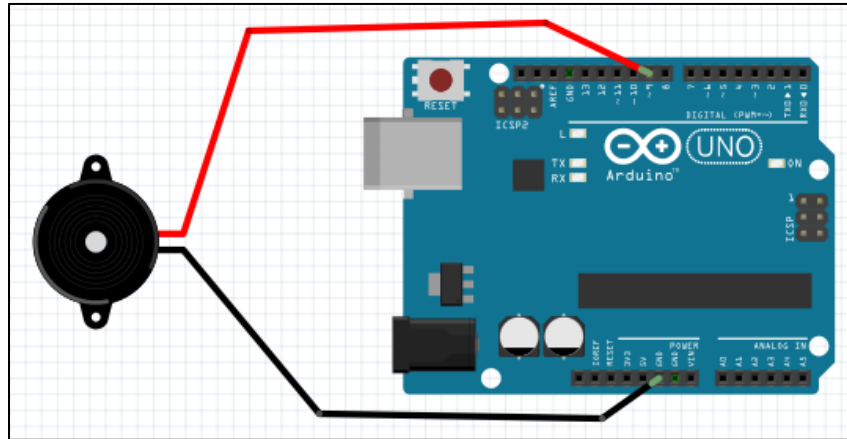


Figure 26: Circuit for Buzzer

Helpful Resources

<https://www.arduino.cc/>

Arduino homepage

<https://www.arduino.cc/en/Reference/HomePage>

A guide to all the terms used in the programs

<http://playground.arduino.cc/Projects/Ideas>

Lots of ideas for fun projects to try

<http://www.instructables.com/howto/arduino/>

Lots more ideas for fun projects!

Try Yourself Circuits

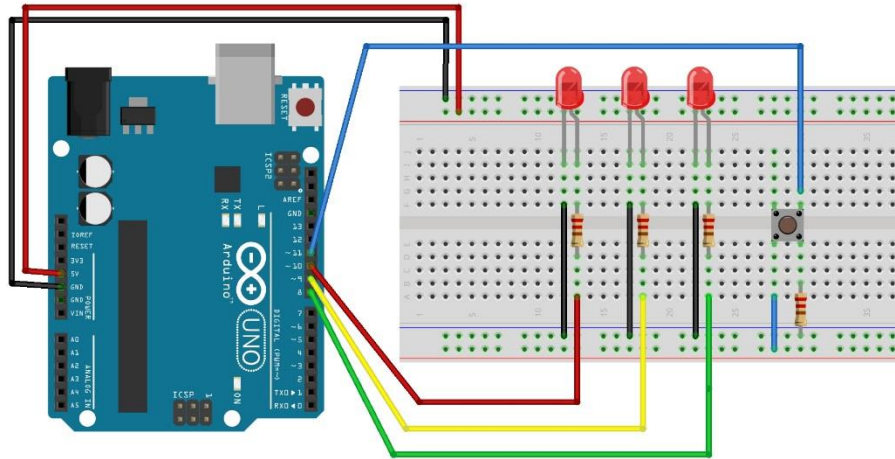


Figure 27 - Traffic Lights with Button

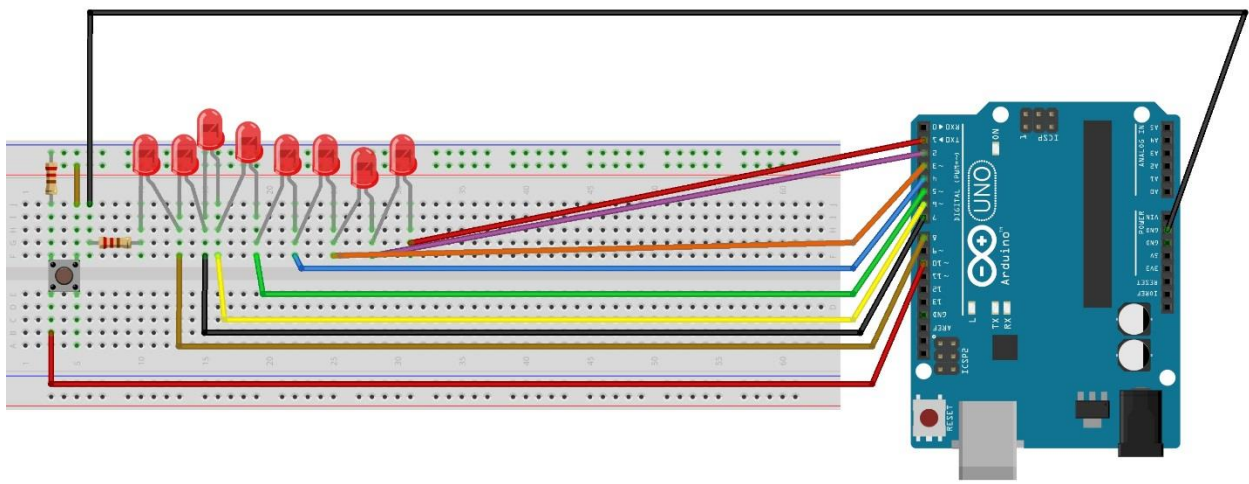


Figure 28 - Randomiser with Extra LEDs and Button