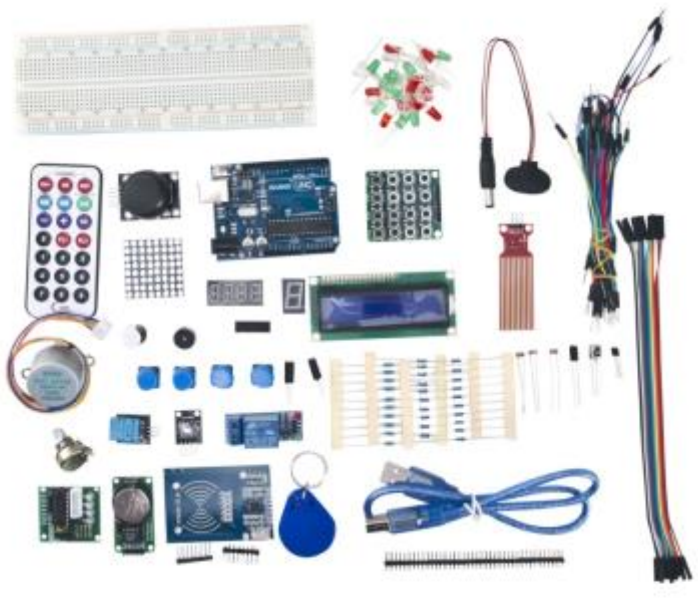
Arduino Guide and Project book



Arduino Guide and Project Book
Tyson Popynick
Prot3us1@gmail.com

About the Arduino

The Arduino is a development board based on the ATMega microprocessors. It is fairly powerful as far as processors go, and suits most hobby purposes.

There are multiple versions of the Arduino, including the Micro, Uno, Mega, Teensy and Zero. There are also plenty of other variants out there but these are the main boards freely available.

The Arduino is an open source project, which means many companies have free access to the layout and specifications of the board, and are legally allowed to produce their own versions for sale, with – or without modifications. These clones are not illegal, and are freely available for you to purchase and use with the genuine Arduino Integrated Development Environment. (IDE).

In the projects that follow we will be using the Arduino UNO. We will be using an Iduino branded Uno…However any brand and version will be fine to use.

About the guide

This guide is written from a hobbyist perspective, aiming to deliver the reader with sample code and a basic but functional understanding of how the Arduino works, and how to use it.
The projects aim to be informative and interesting, and most importantly the code should be as simple as possible to follow, and easy to modify for your own projects!

If you have any queries or comments please email the author (Tyson) at prot3us1@gmail.com. I will respond ASAP and do my best to help you with your problems.

Safety

Personal safety is not generally an issue while operating the Arduino from USB or battery. Be aware of the voltages you are working with and always stay within your own abilities. None of the projects that follow are dangerous, however if you decide to use separate power (for instance, to drive motors) you should be aware that incorrectly wiring or running your project while plugged into USB can potentially damage your USB ports. Always follow the instructions and check your wiring before plugging your Arduino in. Also remember the Arduino will begin running code as soon as it is powered. So if you wish to start a new project it is always good practice to upload a blank sketch to the Arduino before wiring any new modules, to ensure the IO ports are not doing anything that will compromise your new project.

What you need

For the projects in this guide we will make use of many commonly purchased modules and components. I recommend purchasing one of the starter kits available from www.auselectronicsdirect.com.au to give your collection a boost, and set you up to make many fun projects. I will try to explain choices on components as we go along so you can make your own judgments on potential replacement parts, however if you are new to electronics it is best to simply use exactly what is specified to ensure straightforward use.

At minimum you will need:

- Arduino Board
- IDE (Integrated Development Environment) Software (https://www.arduino.cc/en/Main/Software)
- A computer that will run the software
- A breadboard
- Components or modules that you need for your desired project

Contents

Getting Started

First download the Arduino IDE and install it. The latest version can be found at www.arduino.cc/en/Main/Software.

Once this is installed, plug your Arduino into a spare USB port on your computer and the drivers will be automatically installed.

At this point your Arduino is running a blank sketch. A sketch is the Arduino term for a program. Every sketch consists of 2 main functions, and runs immediately once adequate power is provided to the Arduino board.

An example of a sketch will follow, and I will explain each part, the red text is NOT part of the sketch. The blue text IS:

//COPY FROM THIS LINE (You may include this line in the copy operation, but it is not necessary to do so) <- This line tells you to begin copying from here.
//Global Variables go here <- This is a single line comment.
Global variables are like containers to hold data that the entire sketch can access.
void setup() {
/*Runs once at startup*/ <- As mentioned, the code inside the startup() function will only run once, making it the perfect place to set up serial communications and other instructions that you need to run a single time before the main loop starts.

//You can also place LOCAL variables inside functions, only the function they are inside can access them however, and as soon as the function completes they are lost.
}

void loop() {
/*Runs immediately after setup(), and once the end is reached, begins again at the start of loop().*/
//This function is where your main code will go. It is run from top to bottom in the exact order it is written, and once the end is reached, it returns to the top and starts again. The Arduino will run this loop as fast as it possibly can. However it is free-running, so as you add more instructions it will run slower (slightly).

//A quick word on local vs global variables – If I were to try to access a global variable from here, it would succeed. I could edit or read it without a problem. However if I tried to access a variable that was declared inside setup() it would fail, seeing as loop() is outside setup. This may seem confusing right now but you will understand more as you progress.
}
//Additional Functions
Once you get into more advanced programming you will need to add your own functions, for instance – you may routinely flash an LED to indicate something. You could make a function here that would do so, then call this function from inside loop. This saves you having to repeat the same code over and over inside loop.
//COPY TO THIS LINE (You may include this line in the copy operation, but it is not necessary to do so) <- This indicates the end of a sketch. If you are copy/pasting you should stop copying at this line.

Debugging and Error tracking

It is easy to get lost in code and create errors. If you forget a part of syntax such as a bracket or comment, the IDE will likely tell you about the problem…but if the code is written correctly, but doesn't function correctly – how do you find it?

The Serial Monitor!

This window allows us to have the Arduino send text back to the PC, so we can create a log of what is happening, or even relay data back to ourselves as the Arduino processes live.

The serial monitor is a 2 way connection, we can send and receive data to and from the Arduino. Of course most projects are not intended to be plugged into the PC forever…so generally buttons etc are a better method of interacting with the Arduino.

Questions and troubleshooting

Please don't hesitate to contact me at prot3us1@gmail.com with comments or questions regarding the Arduino, I will do my best to reply promptly and sort any issues out. This service is offered free of charge for troubleshooting and simple help. There is a service available where I can assist you or take the lead on writing sketches for you for a modest fee. Feel free to enquire at the above email address if this sounds like something you require.

Arduino Programming

Arduino uses a proprietary coding language based on C# or MONO, the syntax should be familiar to anyone who has programmed in these languages before.

Functions do not need to be declared, but variables do. Variables do not need to be explicitly initialized, although you should always do so to avoid problems later.

Syntax:

All lines of code should be appended with a colon to tell the compiler it is the end of an instruction.

All functions are called by name, followed by () with any parameters inside the brackets.

All functions including if-else statements etc should be followed by curly braces to keep code clean and minimize errors. For example:

```
If (x == 1) {

//instructions to execute if true

}

//Instructions to execute if false.
```

Variables:

Declare the type, then the name, then the value as follows variables are case sensitive so x is not the same as X:

```
Int temp = 1;

Bool x = 0;

String temp = “Hello World”;
```

Integers can be increased or decreased using the shorthand ++ as follows:

```
Int temp = 0; //Declare a integer called temp, and initialize it to 0.

temp++; //Adds 1 to temp. temp is now 1.

temp--; //Subtracts 1 from temp. temp is now 0.
```

Functions:

Functions do not need to be declared like in some languages, instead you simply write the function itself and call it by name.

To write the function you declare the type of data to be returned, or void if nothing is returned. You then declare the name, then any parameters that would need to be used also. An example follows:

```
void flashLED(int numFlashes) { <- void tells Arduino there is nothing returned. Int numFlashes
tells Arduino there should be an integer provided.
	for (int x = 0; x < numFlashes; x++) {
		digitalWrite(9, HIGH);
		delay(1000);
		digitalWrite(9, LOW);
	}
Serial.write(“Flashed LED “);
Serial.write(numFlashes);
Serial.writeln(“ times.”);
}
```

To use this code we would do the following in the main loop or another function:
flashLED(10); <- Would flash the LED 10 times.

Alternatively, if you need to return data, for instance if you are processing data, you would use the following:

```
Int addFive(int numberIn) { //We want to return an integer.
numberIn + 5; //take the number provided, and add 5.
return numberIn; //return the answer.
}
```

This would be called in the same way, however it would return a value…see the example that follows:

```
Int number = 7; //We have an integer with a value of 7 stored in the variable “number”.
number = addFive(number); //number equaled 7. The function then added 5 to it. So number
now is equal to 12. Look at it as if the result replaces the function name, so this looks like:
number = Result of Function. And the result of 5 + 7 = 12.
```

IO Ports:

Accessing the IO ports is quite easy on the Arduino, they are referred to by their number. Arduino in most cases will automatically configure the pin between input and output based on the function you use, however in some cases you may wish to manually set the pin up, for instance if you need to set up the internal pull up or pull down resistors etc. If you have need of this you should go to www.arduino.cc, all of the available functions and variables are listed there and this guide is simply meant to give the newcomer enough experience to get started and understand enough about the device to then go on to their own research.

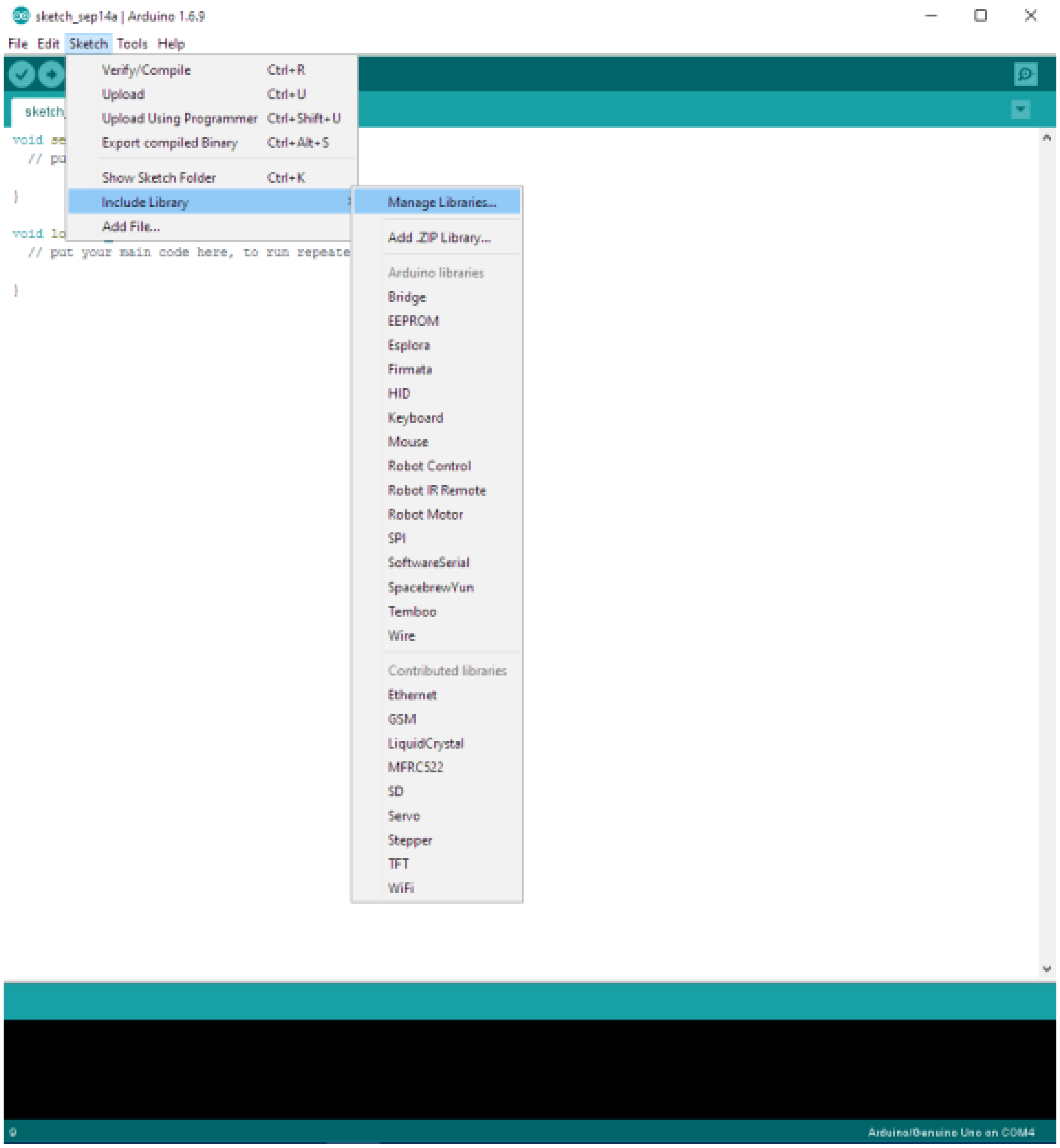
Libraries

During your adventures with Arduino there are often premade programs used to interact with modules and sensors. These are often top quality and there is no point to rewriting them. These are called libraries, they will allow you to use your modules quickly and easily.

You can google search the libraries and find plenty of support and instructions on using them (above and beyond what I provide).

A library allows you to call functions that are already written, for instance...the RFID library contains all of the raw code to access the hardware, all we need to do is initialize it and call the functions. If it wasn’t for this library we would need to manually code the protocol and data lines. A library does all the heavy lifting.

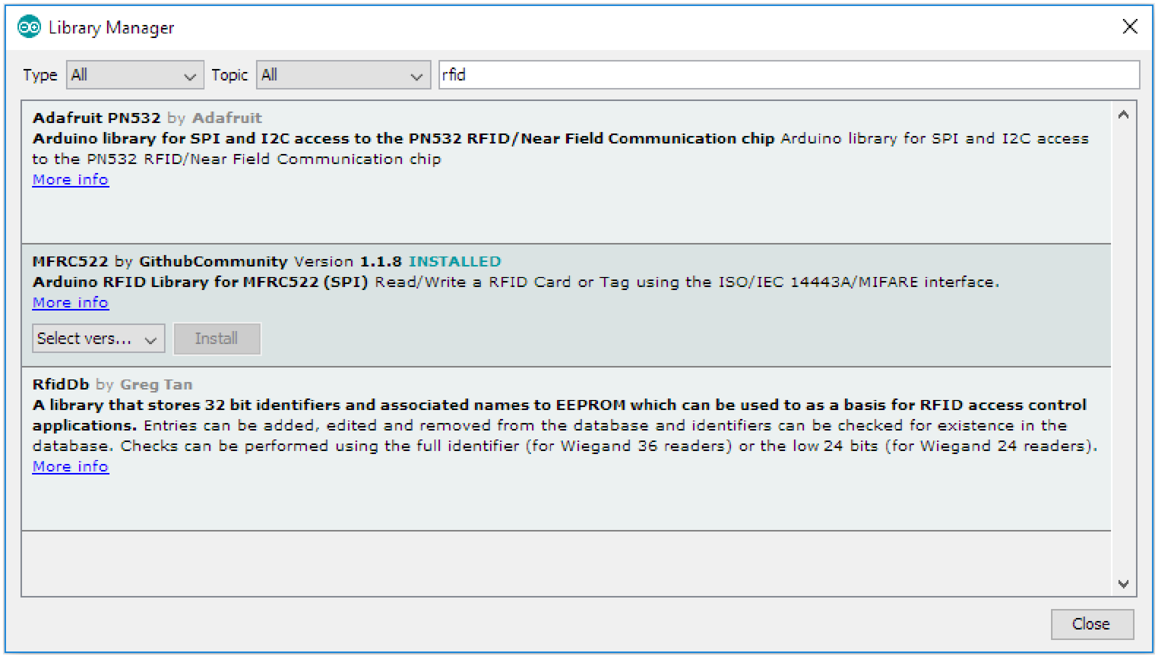
The method of installing and using these libraries follows on the next page.



```
// put your main code here, to run repeate
```

First you should navigate to the Sketch -> Include Library Menu.

Click on Manage Libraries and the following window will appear:



Adafruit PN532 by Adafruit
Arduino library for SPI and I2C access to the PN532 RFID/Near Field Communication chip Arduino library for SPI and I2C access to the PN532 RFID/Near Field Communication chip
More info

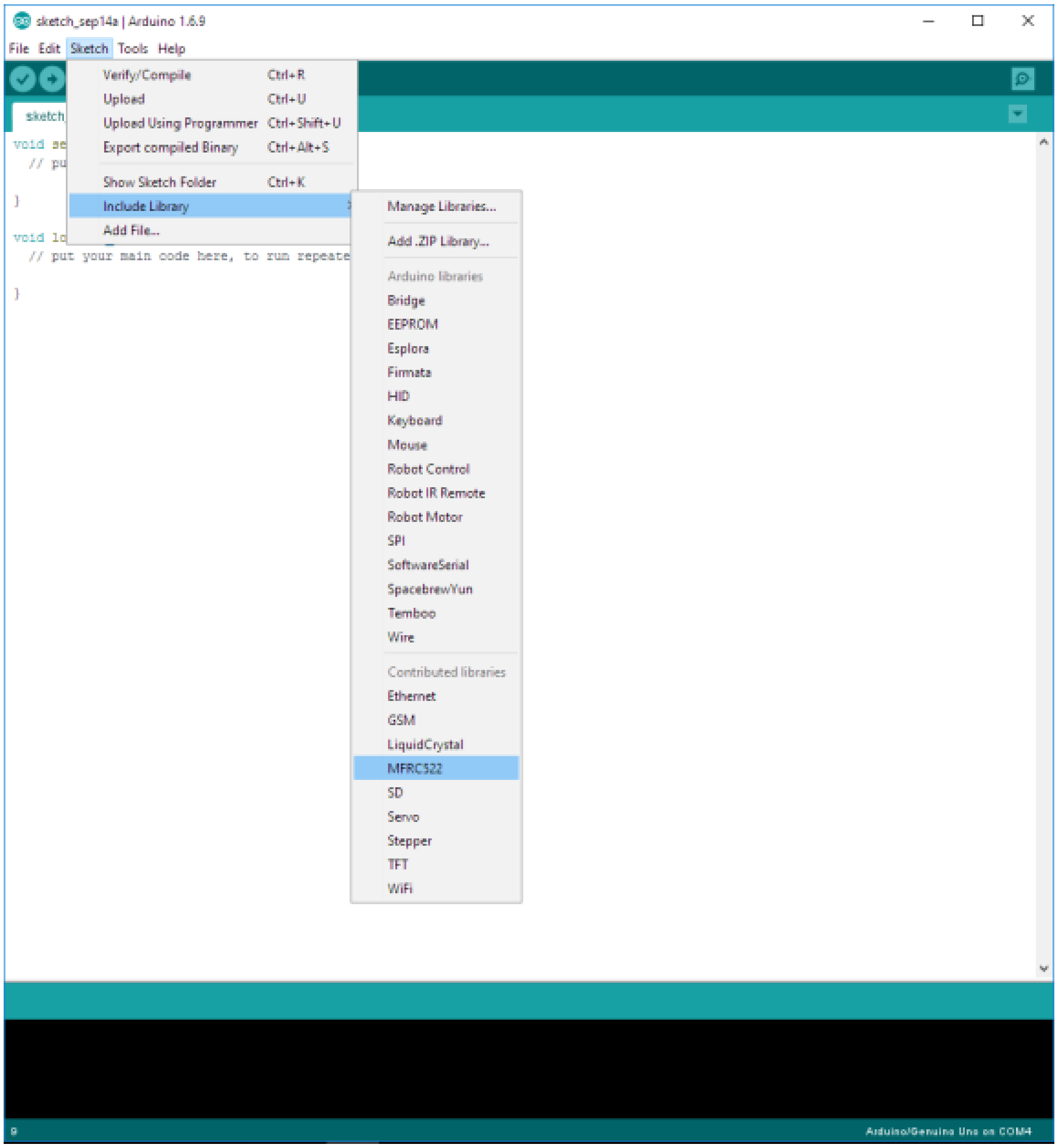
MFRC522 by **GithubCommunity** Version **1.1.8** **INSTALLED**
Arduino RFID Library for MFRC522 (SPI) Read/Write a RFID Card or Tag using the ISO/IEC 14443A/MIFARE interface.
More info

RfidDb by Greg Tan
A library that stores 32 bit identifiers and associated names to EEPROM which can be used to as a basis for RFID access control applications. Entries can be added, edited and removed from the database and identifiers can be checked for existence in the database. Checks can be performed using the full identifier (for Wiegand 36 readers) or the low 24 bits (for Wiegand 24 readers).
More info

You can then search by keyword for a library. In the example above I have searched for RFID as I am looking for an RFID library.

I am presented with 3 results. I choose the middle result and click the install button. If there is a drop down box with multiple versions, I choose the latest version. The reason I have chosen the middle option is because the RFID module I have is based on the MFRC522 chip and uses SPI communication. You will be faced with similar options in the future and can always refer to the documentation from the manufacturer or place of purchase to help you choose the correct option.

This will now install the library to your Arduino directory. You have one final step to add it to your project, and that is to select it from the list of libraries. You only need to install each library once and then it is kept automatically in your Arduino directory for you to use any time.



In this example I have selected the MFRC522 Library we just installed. If you are installing a different library, please select the appropriately named option.

Project 1

(Serial Communication)

Project Aim/Description:

This project will show you how to communicate with the Arduino, including sending data to the Arduino as well as getting data back from the Arduino.

What you need:

- Arduino

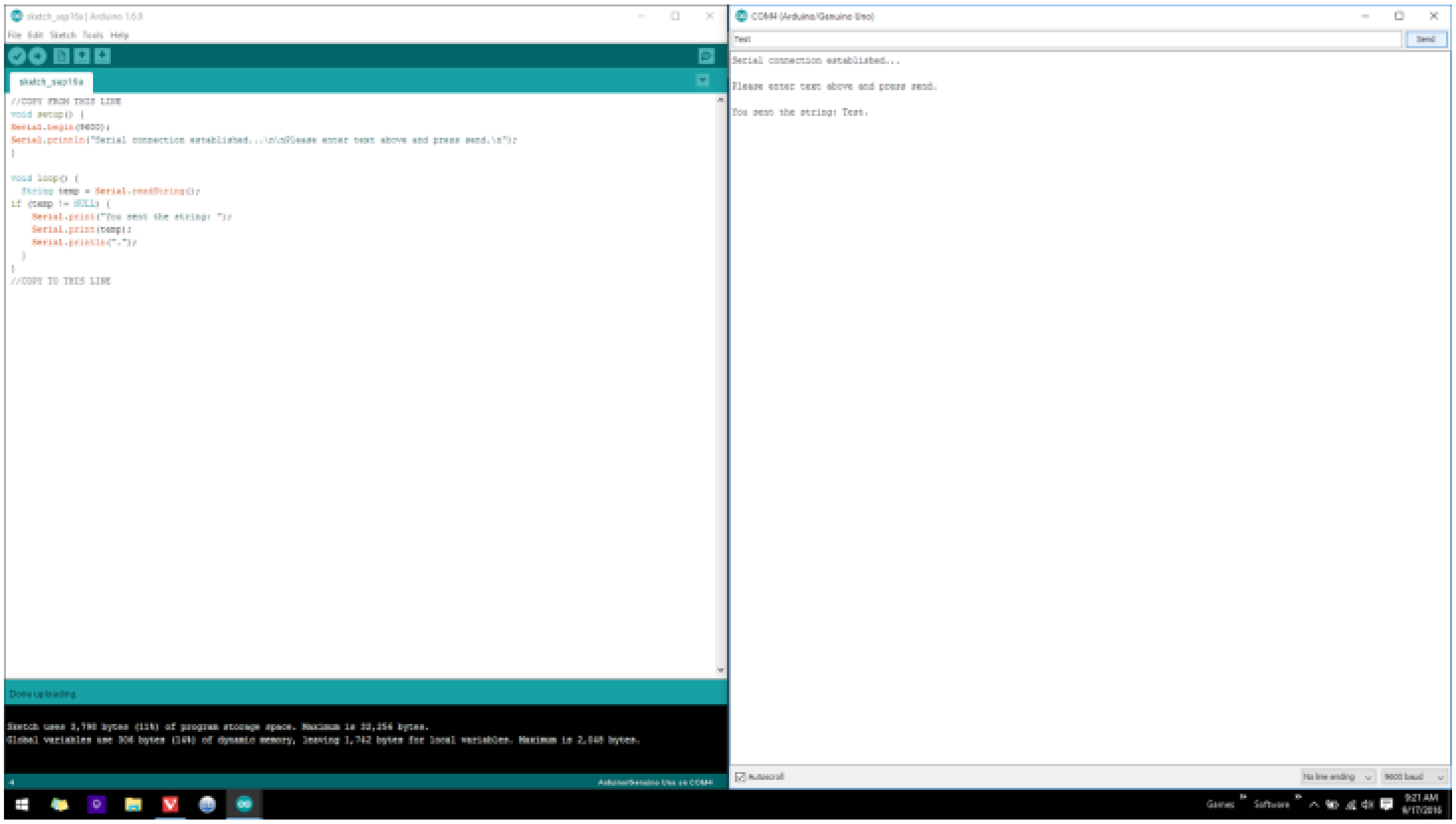
Expected outcome:

You will be able to send a string of text to the Arduino, as well as have the Arduino send a string of text back.

Procedure:

This project is very simple, we will upload a sketch to the Arduino, open the serial monitor and use the textbox to send a string to the board. The Arduino will then send the Text back, after reformatting it.

To begin, copy the code below into the IDE window (left) as illustrated in the image that follows, you will also want to bring up the SERIAL MONITOR (right in image). To do this, open the TOOLS menu, and select SERIAL MONITOR.



```
//COPY FROM THIS LINE
void setup() {
Serial.begin(9600);
Serial.println("Serial connection established...\n\nPlease enter text above and press send.\n");
}

void loop() {
  String temp = Serial.readString();
if (temp != NULL) {
    Serial.print("You sent the string: ");
    Serial.print(temp);
    Serial.println(".");
  }
}
//COPY TO THIS LINE
```

Done uploading.

```
Sketch uses 3,798 bytes (11%) of program storage space. Maximum is 32,256 bytes.
Global variables use 306 bytes (14%) of dynamic memory, leaving 1,742 bytes for local variables. Maximum is 2,048 bytes.
```

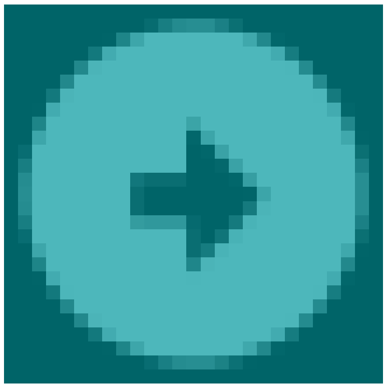
```
Serial connection established...

Please enter text above and press send.

You sent the string: Test.
```

CODE:

```
//COPY FROM THIS LINE
void setup() {
Serial.begin(9600);
Serial.println("Serial connection established...\n\nPlease enter text above and press send.\n");
}

void loop() {
  String temp = Serial.readString(); /*This is a LOCAL variable, it will store any text we send TO
the Arduino FROM the PC.*/
if (temp != NULL) { //If the variable is NOT empty run the next instructions
    Serial.print("You sent the string: "); //Print the string
    Serial.print(temp); //then print the text we sent to the Arduino
    Serial.println("."); //Finally print a fullstop. This is data FROM the Arduino TO the PC.
  }
//If there is no text entered, the Arduino will continue processing from here…But as there are no
further instructions, it will return to the top and start again. If you were to put something here
such as a Serial.print() command, it would fire constantly until you provide a string of text. Feel
free to try it!
}
//COPY TO THIS LINE
```

With the code copied into the IDE window (replacing all the text that was previously in there), you now simply need to click the UPLOAD button, and the Arduino will receive the code and start executing it!

The upload button can be found in the top left of the IDE window and looks like this:



Press the button and you will see the progress at the bottom of the IDE window. Once it completes you will be greeted with the following text in the serial monitor:

Serial connection established…

Please enter text above and press send.

You will notice a text box at the top of the serial monitor window, with a send button next to it. Type some text into the box, and press enter or click the send button.

You should now see:

You sent the string: <TEXT YOU ENTERED>.

Looking at the code we uploaded again, can you see what is happening?

Notes:

In this project we learned:

How to enter and upload code to the Arduino.

How to send and receive data to and from the Arduino.

How to bring up and use the serial monitor.

The Serial.print function will print text without adding a new line, so the text will continue on as if it was a single sentence.

The Serial.println function does the same thing, except it adds a newline after the text, as if you had pressed enter after typing a sentence.

Project 2
(PWM – Pulse-Width Modulation)

Project Aim/Description:

In this project we will look at Pulse-Width Modulation, how to use the Arduino to generate it, and what it does.

What you need:

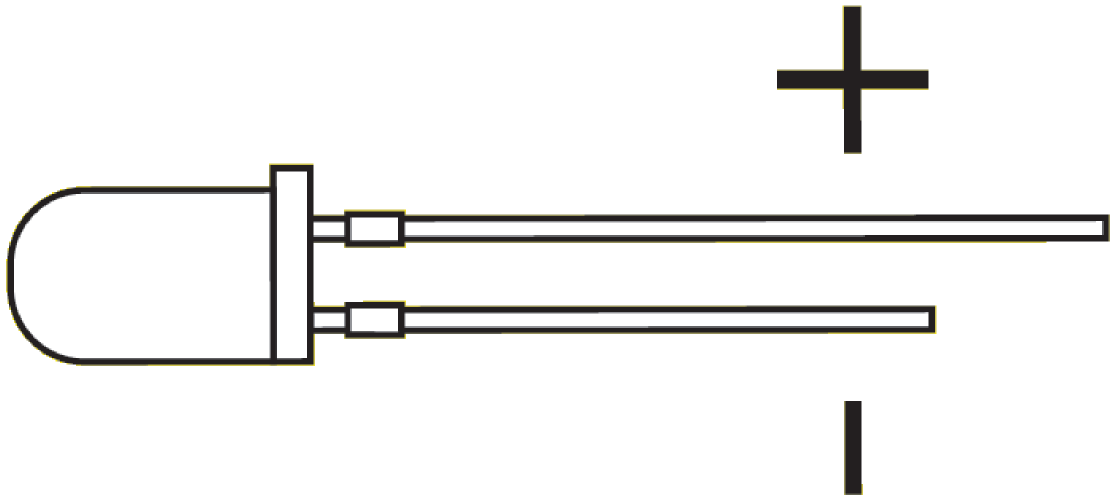
- Arduino
- Breadboard
- LED
- Jumper Wires (Male to Male)
- 1x 1k Resistor
  (Brown, Black, Black, Brown, Brown)

Expected outcome:

You will understand what PWM is, and how to use the Arduino to generate it. We will do this by dimming an LED. An LED is a Light Emitting Diode, and it can only be connected one way. If you connect it in reverse, it will not work at all.

You can distinguish the polarity of an LED in two ways. Firstly, there will be a longer leg and a shorter leg on the LED. The longer leg is positive and the shorter leg is negative. Sometimes however, the leg length may not be easily visible. For instance, if you have cut the legs shorter for a different project, or bent them for a breadboard. In this case, you can use the second method.

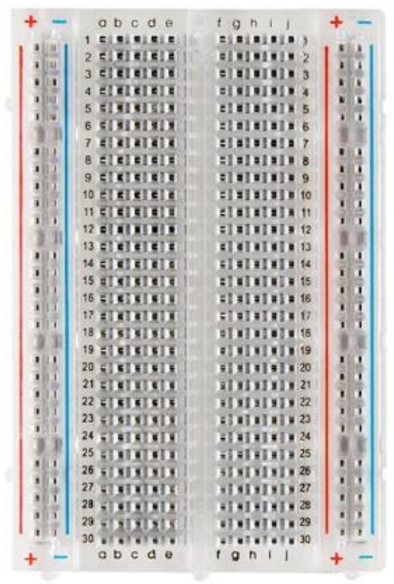
The second method is to look at the plastic body of the LED. You will notice one side of it is FLAT. This signifies the NEGATIVE side of the LED.

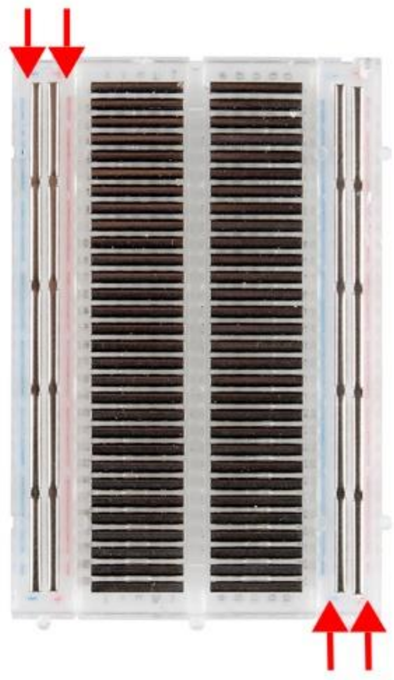


Procedure:

In this project we will be wiring a circuit on the breadboard. A breadboard is a very handy prototyping board, which allows you to make non-permanent connections between components very fast, you can change components and connections without needing to bond anything together.

A breadboard has internal connections that seem strange at first, but once you have used one a few times you should see the beauty of it.

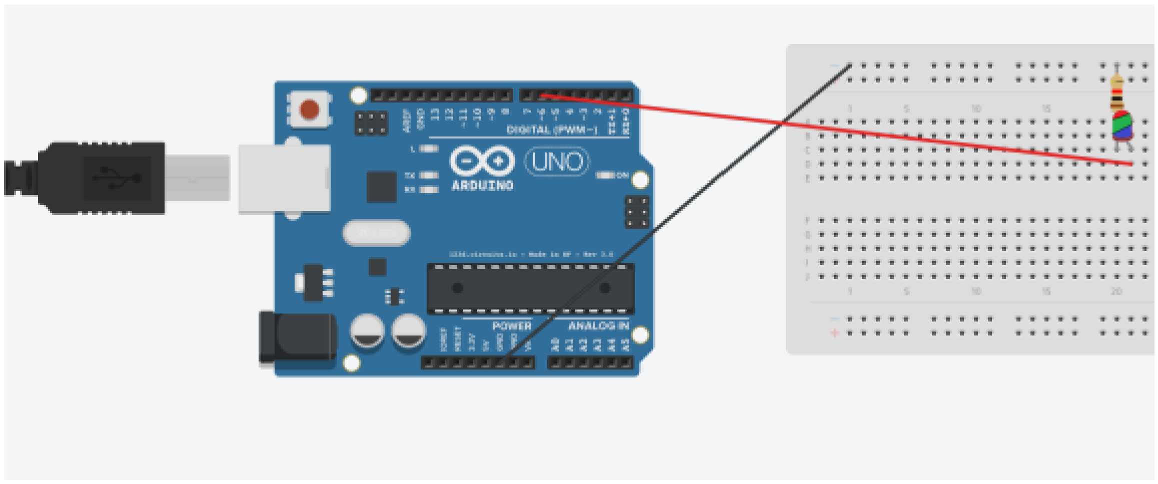




This is the inside of a breadboard, the dark lines are copper clips, the white area is insulating plastic. Notice how the edges of the board are connected in 2 separate lines, and the inside is connected in short lines with a break down the middle. You can always refer back to this image to see what is going on inside the breadboard later. It will seem confusing for now, but it will clear up once you have some experience. The edges are handy for running power, as they are accessible from anywhere on the board, and the inner lines are great for components. The gap in the middle is specifically designed to allow most microchip packages to fit across it, allowing access to each pin easily.

In this project we will be connecting an LED to a pin on the Arduino. If we were to connect it to the 5v output it would shine at full brightness. However we will be using the PWM abilities of the Arduino to “chop” the 5 volts up. Essentially we will turn 5v on and off rapidly, which will make the LED turn on and off. The longer we allow the pulses to be, the brighter the LED will seem. The slower we make them, the dimmer it will seem.

Please wire the circuit as indicated in the image below:



Once you have done this, please upload the following code:

CODE:

```
//----Begin Code (copy from here)----

//Variables
int pwmVal  = 255; //PWM range is 0 - 255. Change this value and reupload to see the difference!

void setup() {
  // put your setup code here, to run once:
pinMode(ledPin, OUTPUT); //Set the pin we chose above as OUTPUT. This is actually automatic however I included it to give an example of where you might use this feature.
analogWrite(6, pwmVal); // the 6 is the pin on the Arduino the LED is connected to, pwmVal is the variable containing the value for the PWM duration. analogWrite tells the Arduino this is a PWM pin.
}

void loop() {
  // put your main code here, to run repeatedly:
}

//----End Code (copy to here)----
```

Notes:

The BLUE text is code, the RED text is comments. As you can see there is very little code required for this project, however I am trying to give fairly detailed explanations in comments, which makes it look longer than it is.

Project 3

(Light Sensor <Photoresistor/Light Dependent Resistor>)

Project Aim/Description:

This project aims to show the reader how to use an LDR, also known as a photoresistor in their projects. The component itself varies its resistance based on the amount of light that hits the surface, we can measure the resistance and use that to determine the level of light present.

What you need:

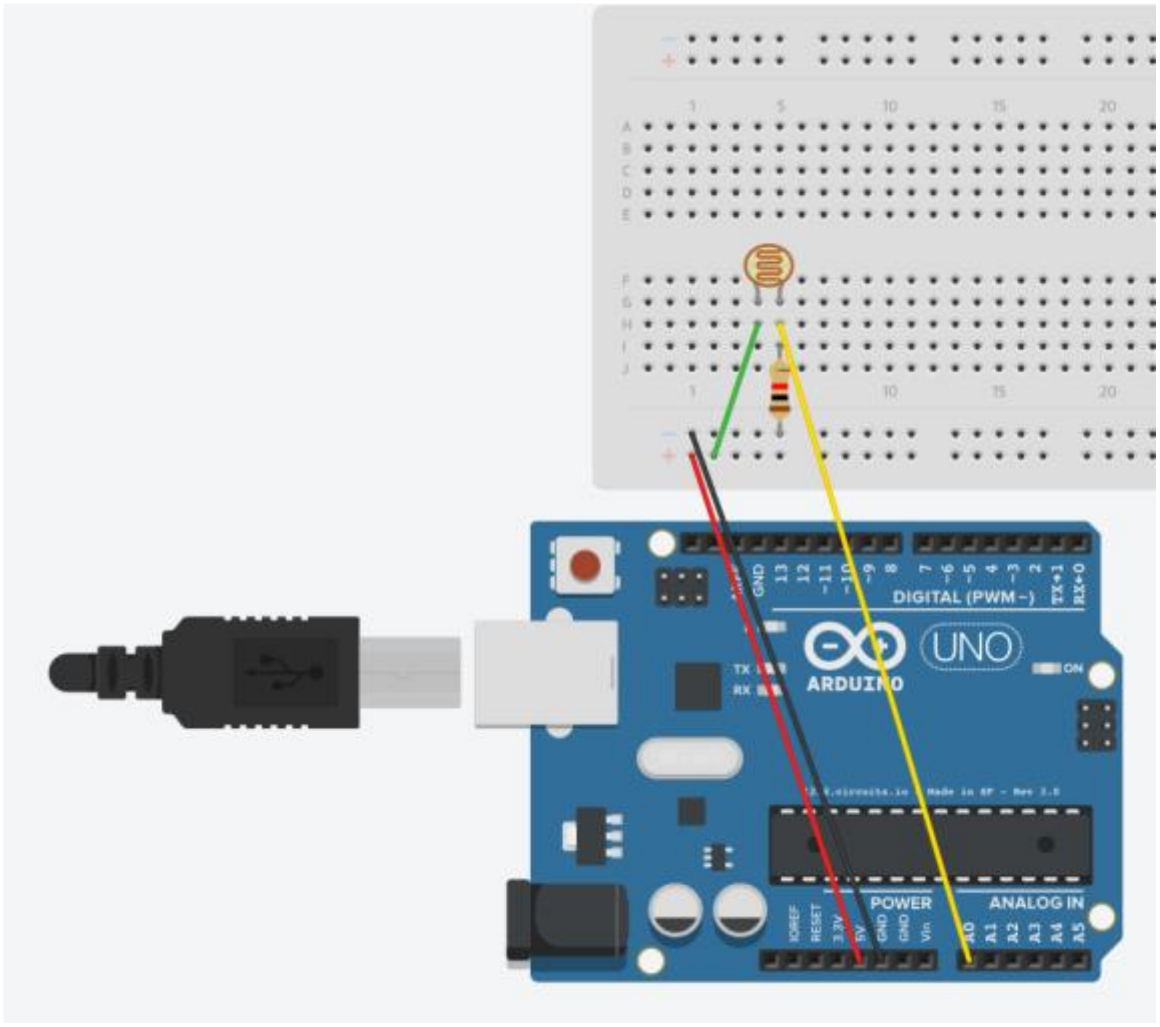
- Arduino
- Breadboard
- Jumper Wires (Male to Male)
- Photoresistor
- 1x 220 Ohm resistor
  (Brown, Black, Black, Red, Red)

Expected outcome:

The reader will be able to connect and read a photoresistor, and use the code in their future projects!

Procedure:

Wire the circuit as follows:



CODE:

```
//----Begin Code (copy from here)----
//Variables
int inPin = A0; //Pin the sensor is connected to
int sensorVal = 0; //Variable to store sensor data

void setup() {
  // put your setup code here, to run once:
Serial.begin(9600);
Serial.println("Serial Communication started...\n");
}
void loop() {
  // put your main code here, to run repeatedly:
sensorVal = analogRead(inPin); //analogRead will read the voltage on the pin specified and
return it as a value between 0 and 1024.
Serial.println(sensorVal); //Print the sensor reading to the serial window so we can view the
data.
}
//----End Code (copy to here)----
```

Notes:

The LDR varies its resistance based on light hitting it, and the Arduino technically only reads voltages, so how do we measure resistance?

We are not REALLY measuring the resistance in this circuit, instead we are creating a resistor divider network between the LDR and the resistor, which will vary the voltage directly based on the LDRs value. We then read the voltage out of the divider network and print that value. Google resistor divider network for more information.

Project 4
(Flame Sensor)

Project Aim/Description:

This project uses a Flame Sensor component to detect the intensity and presence of a flame. The sensor is designed to isolate and read specific bands of the IR and UV wavelengths to determine if a pattern corresponding to a flame is present. It is quite an accurate reading, only a real flame should trigger it.

What you need:

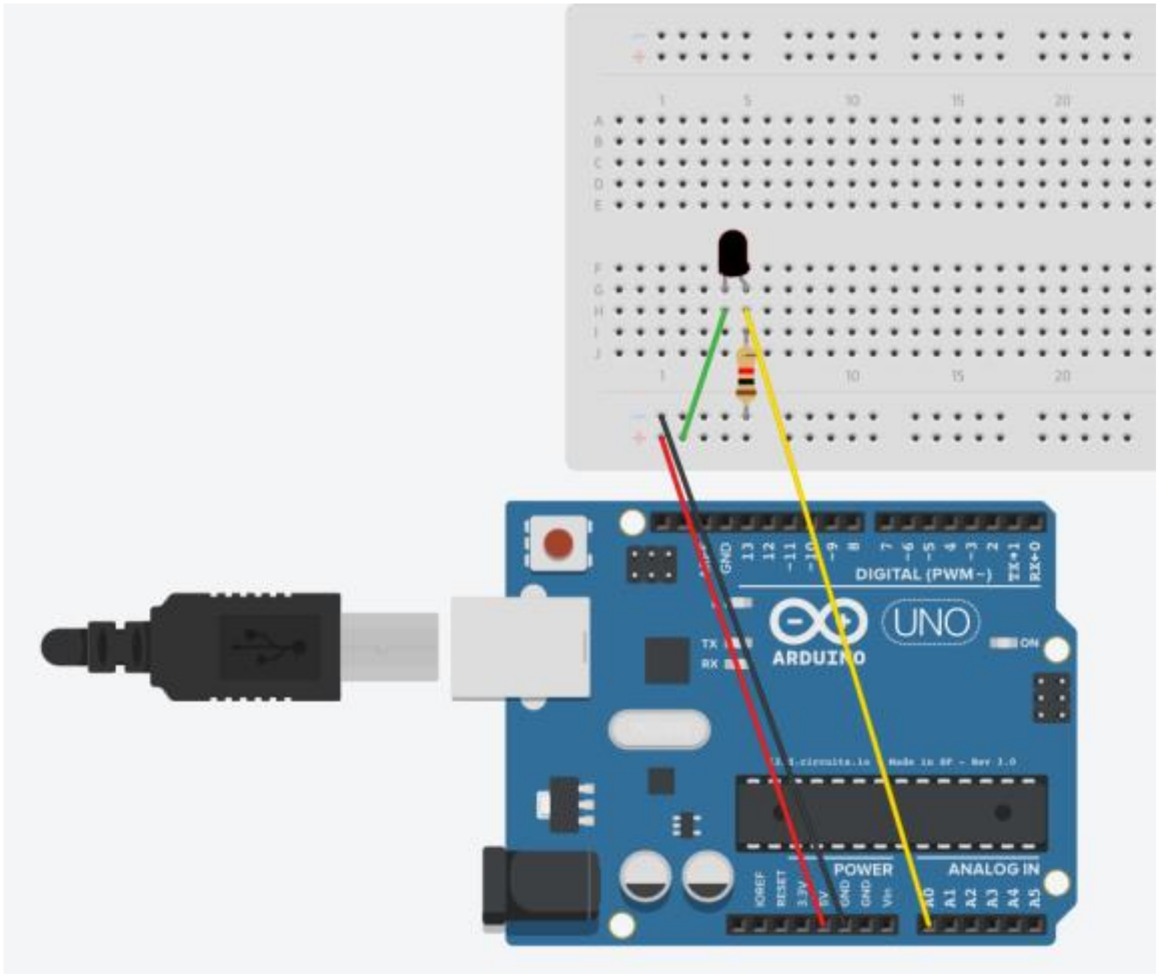
- Arduino
- Breadboard
- Jumper Wires (Male to Male)
- Flame Sensor
- 1x 220 Ohm Resistor.
  (Brown, Black, Black, Red, Red)

Expected outcome:

The user will have the ability to read and use a flame sensor component in their projects, as well as the code to view the data.

Procedure:

Wire the circuit as follows (You may notice it is identical to the previous project, with only the sensor changed. It works in the same way…resistance varies based on input):



CODE:

```
//----Begin Code (copy from here)----
//Variables
int inPin = A0; //Pin the sensor is connected to
int sensorVal = 0; //Variable to store sensor data

void setup() {
  // put your setup code here, to run once:
Serial.begin(9600);
Serial.println("Serial Communication started...\nReady to detect Flame.");

}

void loop() {
  // put your main code here, to run repeatedly:
sensorVal = analogRead(inPin); //analogRead will read the voltage on the pin specified and
return it as a value between 0 and 1024.
if (sensorVal < 1000) {
//Flame
Serial.print("Flame: ");
Serial.println(sensorVal);
}
else {
//Uncomment the lines below to view the raw sensor data.
//You can change the "if statement" above to reflect the difference in sensitivity and ambient
values
//Serial.print("Sensor Value: ");
//Serial.println(sensorVal);
}
}
//----End Code (copy to here)----
```

Notes:

Most of the 2 pin sensors use resistance as a form of reference, therefore they are interchangeable without changing the code. I have added slightly more complex code for this project to try to help you start to see how code flows and functions.

Project 5

(IR Sensor / Remote)

Project Aim/Description:

What you need:

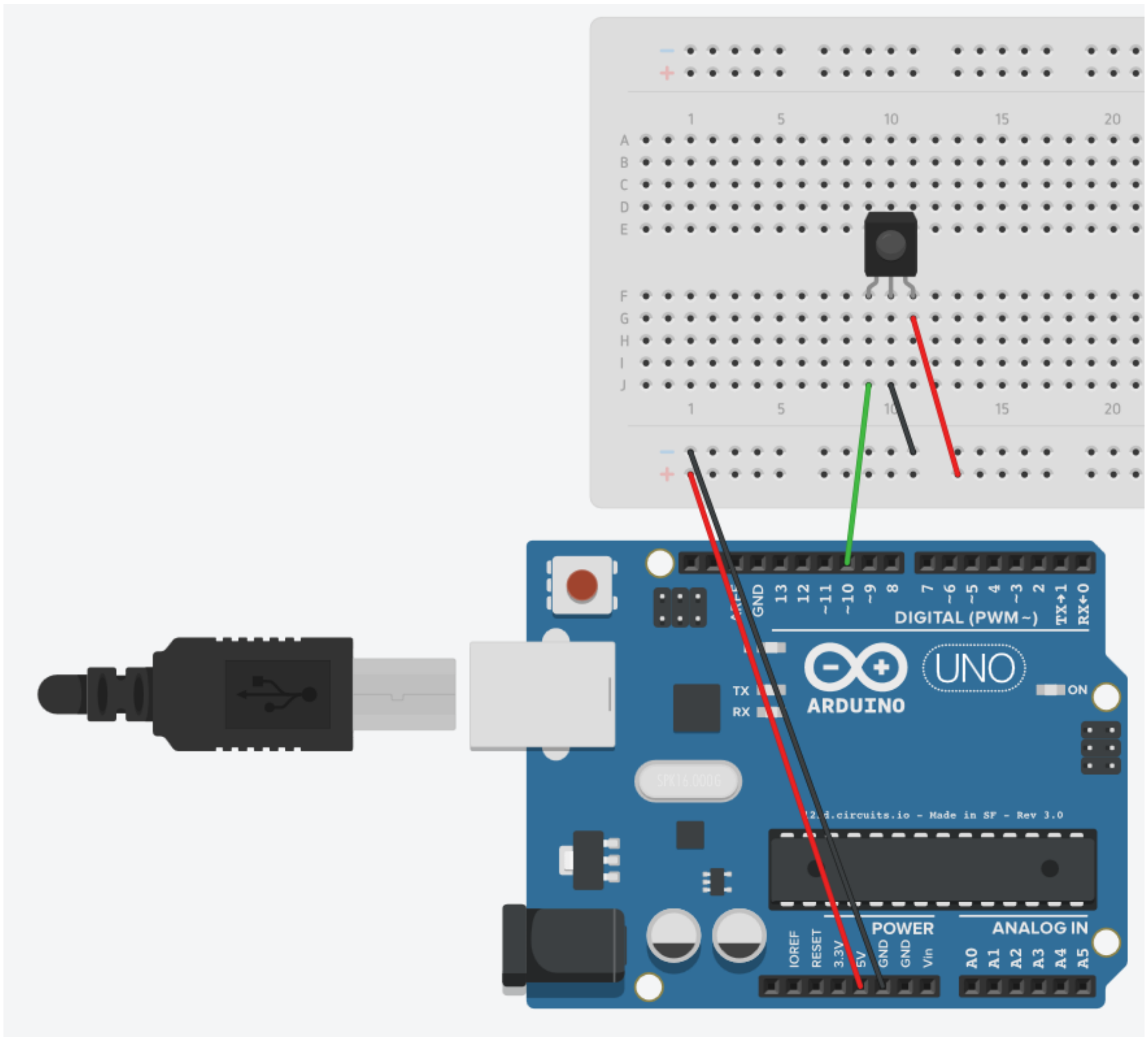
- Arduino
- Breadboard
- Jumper Wires (Male to Male)
- IR Sensor
- Remote Emitter
- IRremote library (Type IR Remote into the library manager as described at the start of the guide) It is the IRremote library by sheriff.

Expected outcome:

The reader will be able to interface with, demodulate and read the signals from IR remote controls.

Procedure:

Wire the circuit as follows:



CODE:

Notes:

This is a 3 pin sensor, and has no resistor! This sensor actually sends data in the form of pulses to the Arduino, the IRremote library decodes these pulses and presents you with this demodulated data.

Project 6
(Tilt Switch)

Project Aim/Description:

This project shows the reader how to interface with a tilt switch sensor, it is the same method used to interface with a button, as this sensor is essentially a button in the way that it works.

What you need:

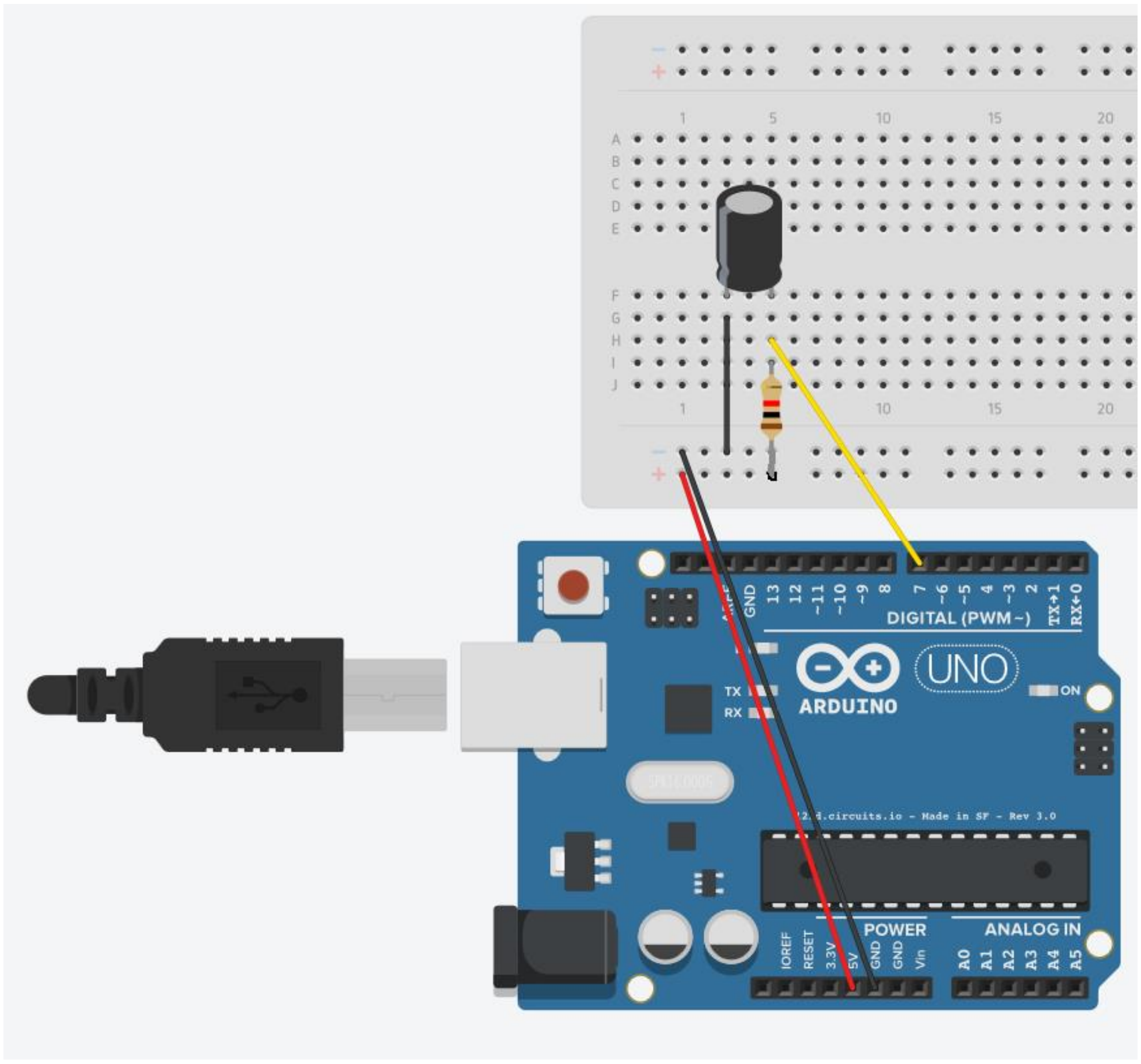
- Arduino
- Breadboard
- Jumper Wires (Male to Male)
- Tilt Switch Sensor
- 1x 220 Ohm Resistor.
  (Brown, Black, Black, Red, Red)

Expected outcome:

The reader will know how to use a tilt switch in their future projects, as well as the code to interface with it.

Procedure:

Wire the circuit as follows:



CODE:

```
//----Begin Code (copy from here)----
//Variables:
void setup() {
  // put your setup code here, to run once:
Serial.begin(9600);
Serial.println("Serial Established..\nTilt board to continue.");
pinMode (7, INPUT); //Set pin 7 to input for reading the sensor.
}
void loop() {
  // put your main code here, to run repeatedly:
if (digitalRead(7) == true)
{
Serial.println("Tilted!");
}
else {
Serial.println("Upright!");
}
}
//----End Code (copy to here)----
```

Notes:

The tilt sensor is a ball bearing inside a tube, at one end there is a wire and there is a wire running along the entire length of the tube. When the ball rolls to the end with the wire present, the ball conducts electricity through both wires, completing the circuit…
When the ball is at the other end, there is no physical connection and the circuit breaks.

Project 7
(RGB LED)

Project Aim/Description:

The reader will be able to use an RGB LED to create many colors and effects in their projects. We will use PWM to control each of the channels in the LED (Red, Green and Blue). By mixing these channels we can achieve any color.

What you need:

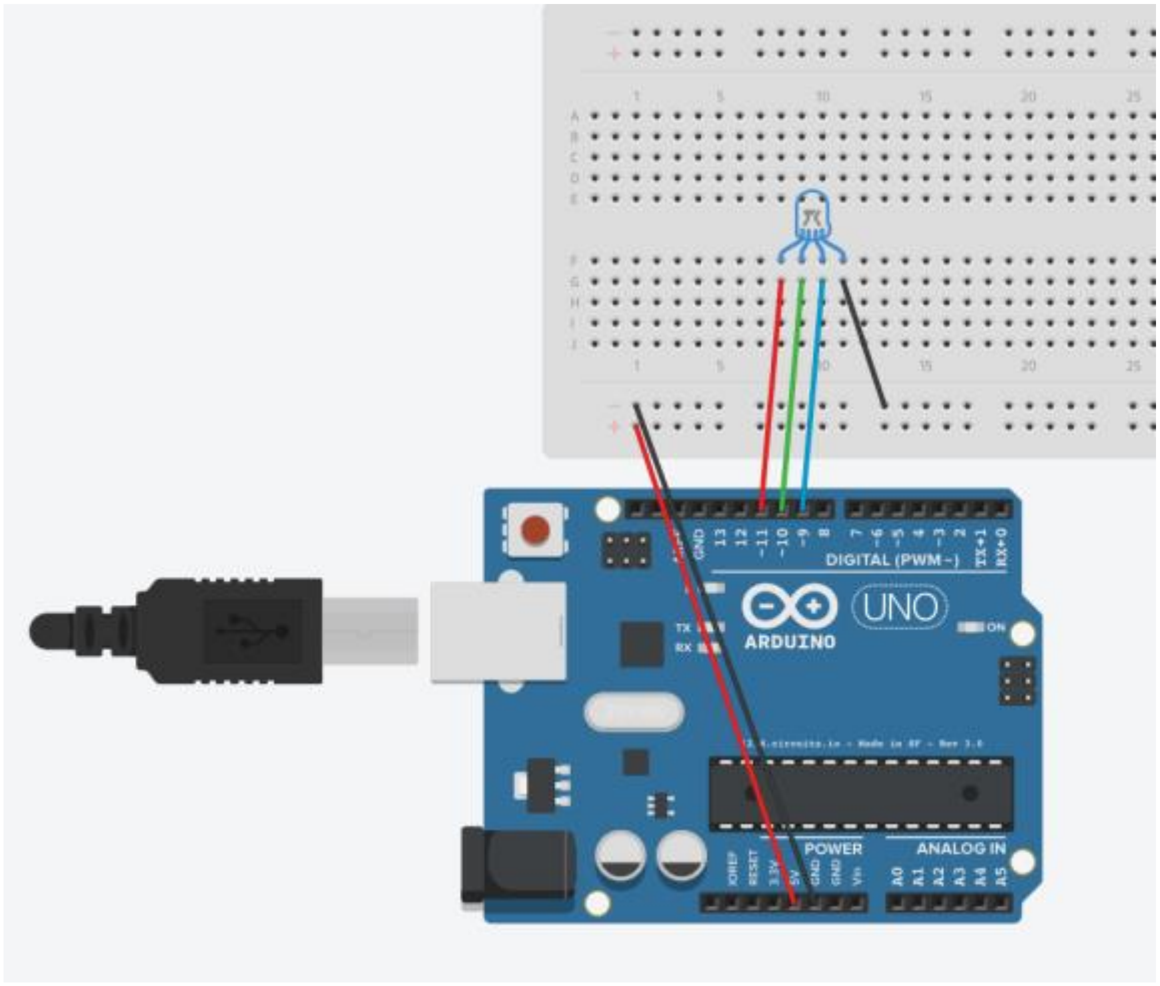
- Arduino
- Breadboard
- Jumper Wires (Male to Male)
- RGB LED Module

Expected outcome:

You will be able to use PWM to control an RGB LED and create any color you wish! This is extremely useful for many projects, from status indicators to mood lighting or color matching with sensors!

Procedure:

Wire the circuit as follows:



CODE:

```
//----Begin Code (copy from here)----
//Variables:
int rPin = 11;
int gPin = 10;
int bPin = 9; //Set the PWM pins to be used on the arduino
int rVal = 0;
int gVal = 0;
int bVal = 0; //Set the values to 0 to begin with
void setup() {
  // put your setup code here, to run once:
//No setup for this project.
}
void loop() {
  // put your main code here, to run repeatedly:
analogWrite(rPin, rVal);
analogWrite(bPin, bVal);
analogWrite(gPin, gVal); //Apply PWM output to each leg of the RGB LED, with the value stored
in the corresponding variable.
rVal = random(0,255);
gVal = random(0,255);
bVal = random(0,255); //Randomise the variables to get a random color each time
delay(500); //Delay before changing colors, so we can see each change.
}
//----End Code (copy to here)----
```

Notes:

This code randomly changes the channels values so the light constantly changes color. You could also manually set the channels by changing the following code:

```
rVal = random(0, 255);

gVal = random(0, 255);

bVal= random(0, 255);
```

to the following:

```
rVal = 0; //Change the 0 to the value you desire.

gVal = 0; //Change the 0 to the value you desire.

bVal = 0; //Change the 0 to the value you desire.
```

Project 8
(Read a potentiometer)

Project Aim/Description:

The reader will be able to read the value from a potentiometer and use them in their projects!

What you need:

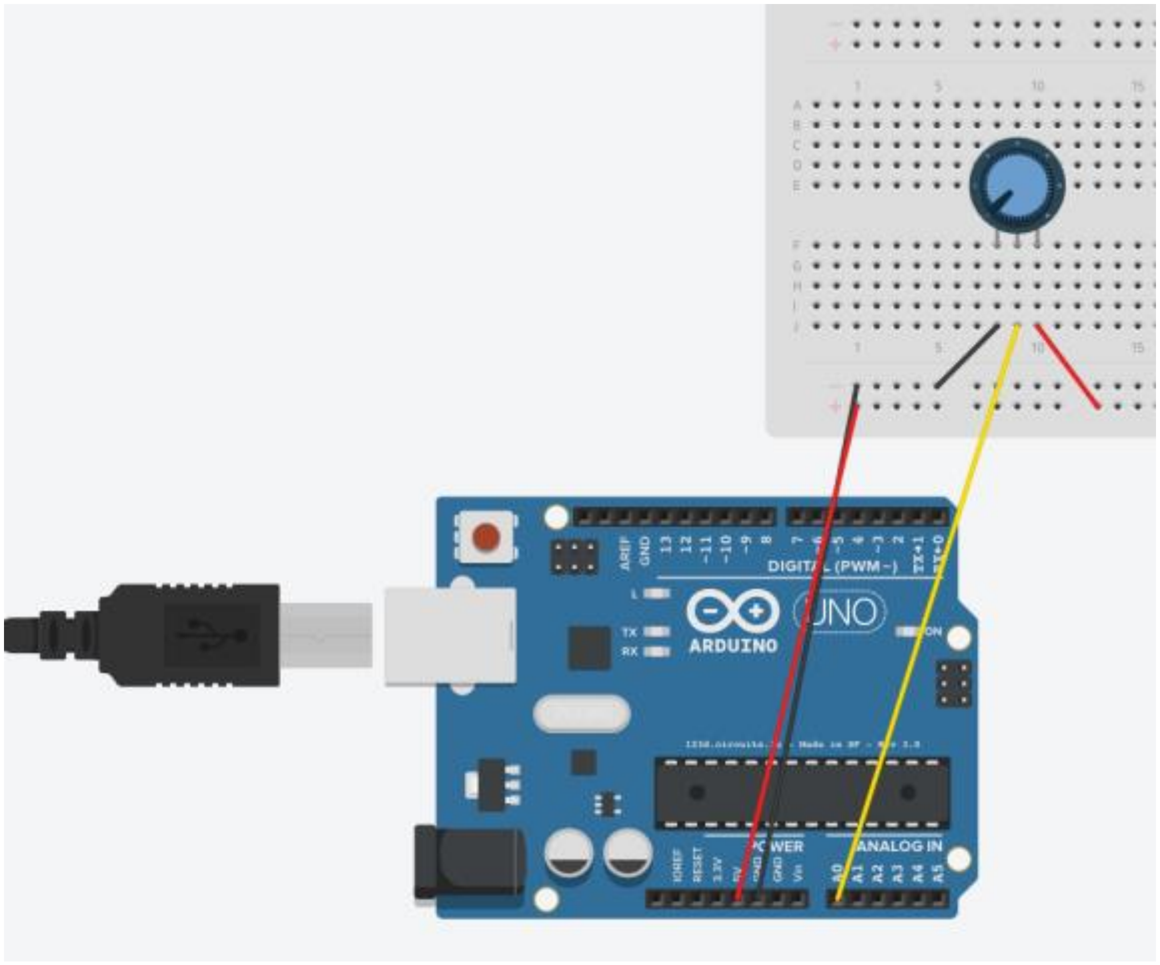
- Arduino
- Breadboard
- Jumper Wires (Male to Male)
- 1x Potentiometer
- 1x 220 Ohm Resistor
  (Brown, Black, Black, Red, Red)

Expected outcome:

The reader will be able to wire and use potentiometer values in their circuits. This is extremely handy to use as an input for brightness, volume, and even as a control input to move values in a selected range. I have seen projects use these as joysticks to control paddles in pong – type games even!

Procedure:

Wire the circuit as follows:



CODE:

```
//----Begin Code (copy from here)----
//Variables:
int varPin = A0;
int val = 0;

void setup() {
  // put your setup code here, to run once:
Serial.begin(9600);
Serial.println("Serial connection established..\nAdjust the Potentiometer to see the value
change!");
}
void loop() {
  // put your main code here, to run repeatedly:
Serial.print("Potentiometer Value: ");
val = analogRead(varPin);
Serial.println(val);
}
//----End Code (copy to here)----
```

Notes:

This is sort of a manual LDR, the resistance changes based on the position of the potentiometer rather than the amount of light on the sensor, but the rest of the circuit is identical. Did you notice this as you were building it?

Project 8
(7-Segment Display)

Project Aim/Description:

The reader will be able to display numbers and letters on a 7-Segment display. These are great for status outputs that require a bit more verbosity than a simple light.

What you need:

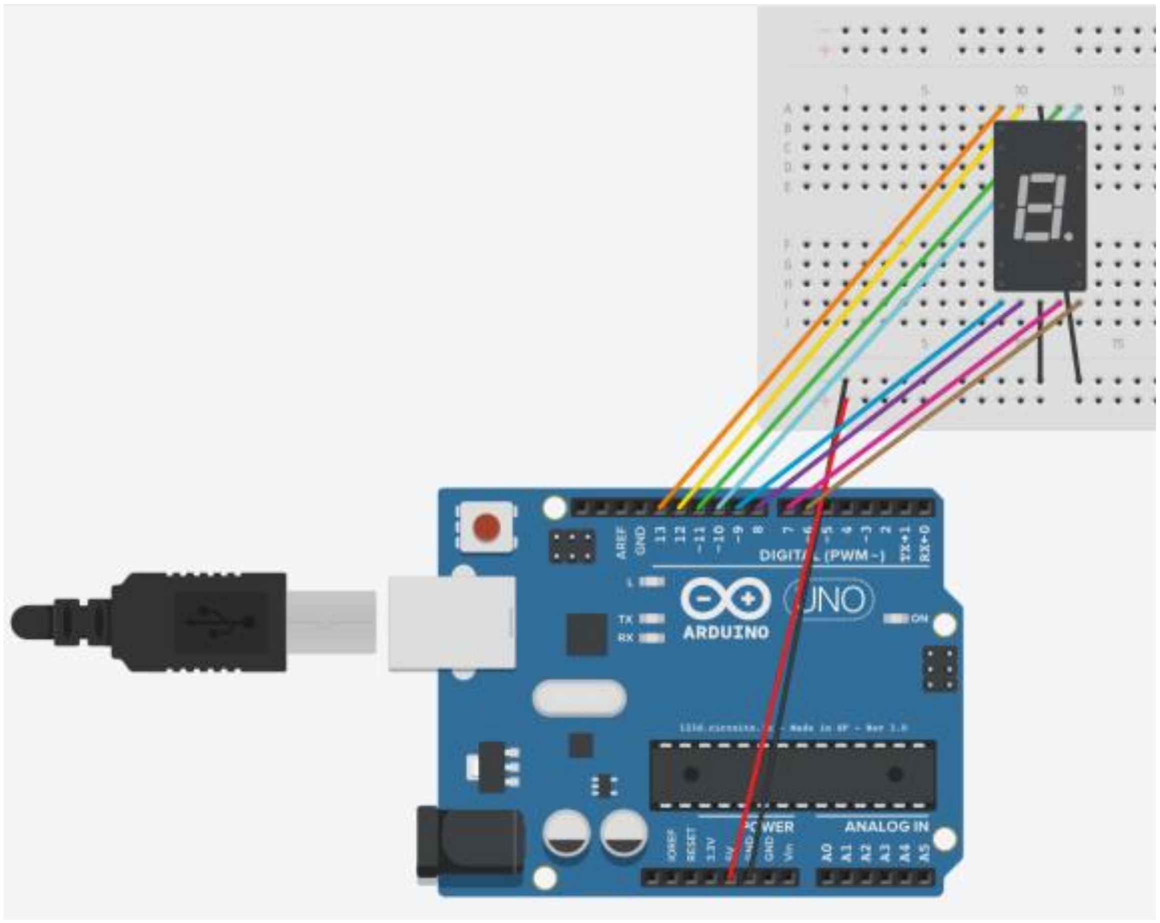
- Arduino
- Breadboard
- Jumper Wires (Male to Male)
- 7-Segment Display

Expected outcome:

As we move into more complex wiring and code the reader will start to get familiar with using the breadboard and IDE, and should be feeling more confident overall using the kit. We are adding another tool to their library this time, in the form of a 7-segment display.

Procedure:

Wire the circuit as follows:



CODE:

```
//----Begin Code (copy from here)----
//Variables:
int pin_a = 6;
int pin_b = 7;
int pin_c = 8;
int pin_d = 9;
int pin_e = 10;
int pin_f = 11;
int pin_g = 12;
int pin_h = 13;
int delayVar = 500;
void setup() {
  // put your setup code here, to run once:
pinMode(pin_a, OUTPUT);
pinMode(pin_b, OUTPUT);
pinMode(pin_c, OUTPUT);
pinMode(pin_d, OUTPUT);
pinMode(pin_e, OUTPUT);
pinMode(pin_f, OUTPUT);
pinMode(pin_g, OUTPUT);
pinMode(pin_h, OUTPUT);
}

void loop() {
 // put your main code here, to run repeatedly:
ch_0();
delay(delayVar);
ch_1();
delay(delayVar);
ch_2();
delay(delayVar);
ch_3();
delay(delayVar);
ch_4();
delay(delayVar);
ch_5();
delay(delayVar);
ch_6();
delay(delayVar);
ch_7();
delay(delayVar);
ch_8();
delay(delayVar);
ch_9();
delay(delayVar);
ch_a();
delay(delayVar);
ch_b();
delay(delayVar);
ch_c();
delay(delayVar);
ch_d();
delay(delayVar);
ch_e();
delay(delayVar);
ch_f();
delay(delayVar);
delay(delayVar);
}
void ch_a()
{
digitalWrite(pin_a, LOW);
digitalWrite(pin_b, HIGH);
digitalWrite(pin_c, LOW);
digitalWrite(pin_d, HIGH);
digitalWrite(pin_e, HIGH);
digitalWrite(pin_f, HIGH);
digitalWrite(pin_g, HIGH);
digitalWrite(pin_h, HIGH);
}
void ch_b()
{
digitalWrite(pin_a, LOW);
digitalWrite(pin_b, HIGH);
digitalWrite(pin_c, HIGH);
digitalWrite(pin_d, HIGH);
digitalWrite(pin_e, HIGH);
digitalWrite(pin_f, HIGH);
digitalWrite(pin_g, HIGH);
digitalWrite(pin_h, HIGH);
}
void ch_c()
{
digitalWrite(pin_a, LOW);
digitalWrite(pin_b, LOW);
digitalWrite(pin_c, HIGH);
digitalWrite(pin_d, HIGH);
digitalWrite(pin_e, LOW);
digitalWrite(pin_f, HIGH);
digitalWrite(pin_g, HIGH);
digitalWrite(pin_h, LOW);
}
void ch_d()
{
digitalWrite(pin_a, LOW);
digitalWrite(pin_b, HIGH);
digitalWrite(pin_c, HIGH);
digitalWrite(pin_d, HIGH);
digitalWrite(pin_e, HIGH);
digitalWrite(pin_f, LOW);
digitalWrite(pin_g, LOW);
digitalWrite(pin_h, HIGH);
}
void ch_e()
{
digitalWrite(pin_a, LOW);
digitalWrite(pin_b, LOW);
digitalWrite(pin_c, HIGH);
digitalWrite(pin_d, HIGH);
digitalWrite(pin_e, LOW);
digitalWrite(pin_f, HIGH);
digitalWrite(pin_g, HIGH);
digitalWrite(pin_h, HIGH);
}
void ch_f()
```

```
{
digitalWrite(pin_a, LOW);
digitalWrite(pin_b, LOW);
digitalWrite(pin_c, LOW);
digitalWrite(pin_d, HIGH);
digitalWrite(pin_e, LOW);
digitalWrite(pin_f, HIGH);
digitalWrite(pin_g, HIGH);
digitalWrite(pin_h, HIGH);
}
void ch_1()
{
digitalWrite(pin_a, LOW);
digitalWrite(pin_b, HIGH);
digitalWrite(pin_c, LOW);
digitalWrite(pin_d, LOW);
digitalWrite(pin_e, HIGH);
digitalWrite(pin_f, LOW);
digitalWrite(pin_g, LOW);
digitalWrite(pin_h, LOW);
}
void ch_2()
{
digitalWrite(pin_a, LOW);
digitalWrite(pin_b, LOW);
digitalWrite(pin_c, HIGH);
digitalWrite(pin_d, HIGH);
digitalWrite(pin_e, HIGH);
digitalWrite(pin_f, HIGH);
digitalWrite(pin_g, LOW);
digitalWrite(pin_h, HIGH);
}
void ch_3()
{
digitalWrite(pin_a, LOW);
digitalWrite(pin_b, HIGH);
digitalWrite(pin_c, HIGH);
digitalWrite(pin_d, LOW);
digitalWrite(pin_e, HIGH);
digitalWrite(pin_f, HIGH);
digitalWrite(pin_g, LOW);
digitalWrite(pin_h, HIGH);
}
void ch_4()
{
digitalWrite(pin_a, LOW);
digitalWrite(pin_b, HIGH);
digitalWrite(pin_c, LOW);
digitalWrite(pin_d, LOW);
digitalWrite(pin_e, HIGH);
digitalWrite(pin_f, LOW);
digitalWrite(pin_g, HIGH);
digitalWrite(pin_h, HIGH);
}
void ch_5()
{
digitalWrite(pin_a, LOW);
digitalWrite(pin_b, HIGH);
digitalWrite(pin_c, HIGH);
digitalWrite(pin_d, LOW);
digitalWrite(pin_e, LOW);
digitalWrite(pin_f, HIGH);
digitalWrite(pin_g, HIGH);
digitalWrite(pin_h, HIGH);
}
void ch_6()
{
digitalWrite(pin_a, LOW);
digitalWrite(pin_b, HIGH);
digitalWrite(pin_c, HIGH);
digitalWrite(pin_d, HIGH);
digitalWrite(pin_e, LOW);
digitalWrite(pin_f, HIGH);
digitalWrite(pin_g, HIGH);
digitalWrite(pin_h, HIGH);
}
void ch_7()
{
digitalWrite(pin_a, LOW);
digitalWrite(pin_b, HIGH);
digitalWrite(pin_c, LOW);
digitalWrite(pin_d, LOW);
digitalWrite(pin_e, HIGH);
digitalWrite(pin_f, HIGH);
digitalWrite(pin_g, LOW);
digitalWrite(pin_h, LOW);
}
void ch_8()
{
digitalWrite(pin_a, LOW);
digitalWrite(pin_b, HIGH);
digitalWrite(pin_c, HIGH);
digitalWrite(pin_d, HIGH);
digitalWrite(pin_e, HIGH);
digitalWrite(pin_f, HIGH);
digitalWrite(pin_g, HIGH);
digitalWrite(pin_h, HIGH);
}
```

```
void ch_9()
{
digitalWrite(pin_a, LOW);
digitalWrite(pin_b, HIGH);
digitalWrite(pin_c, HIGH);
digitalWrite(pin_d, LOW);
digitalWrite(pin_e, HIGH);
digitalWrite(pin_f, HIGH);
digitalWrite(pin_g, HIGH);
digitalWrite(pin_h, HIGH);
}
void ch_0()
{
digitalWrite(pin_a, LOW);
digitalWrite(pin_b, HIGH);
digitalWrite(pin_c, HIGH);
digitalWrite(pin_d, HIGH);
digitalWrite(pin_e, HIGH);
digitalWrite(pin_f, HIGH);
digitalWrite(pin_g, HIGH);
digitalWrite(pin_h, LOW);
}
//----End Code (copy to here)----
```

Notes:

In this project we have defined each letter as a function, we then call the functions in the main loop in order to display the character we want to see. This is directly translatable to the readers projects and gives a simple to use template for further symbols and signs. Experiment with the code to make your own features!

Project 10
(Passive Buzzer)

Project Aim/Description:

The reader will be able to wire up and use a passive buzzer, which is a small speaker. The passive buzzer cannot just be fed a DC voltage, it requires an AC voltage to work, and we will use a PWM signal to simulate an AC voltage to swing the diaphragm. In this example we will generate random pitches, however it is not difficult to experiment and find a tone you would like.

What you need:

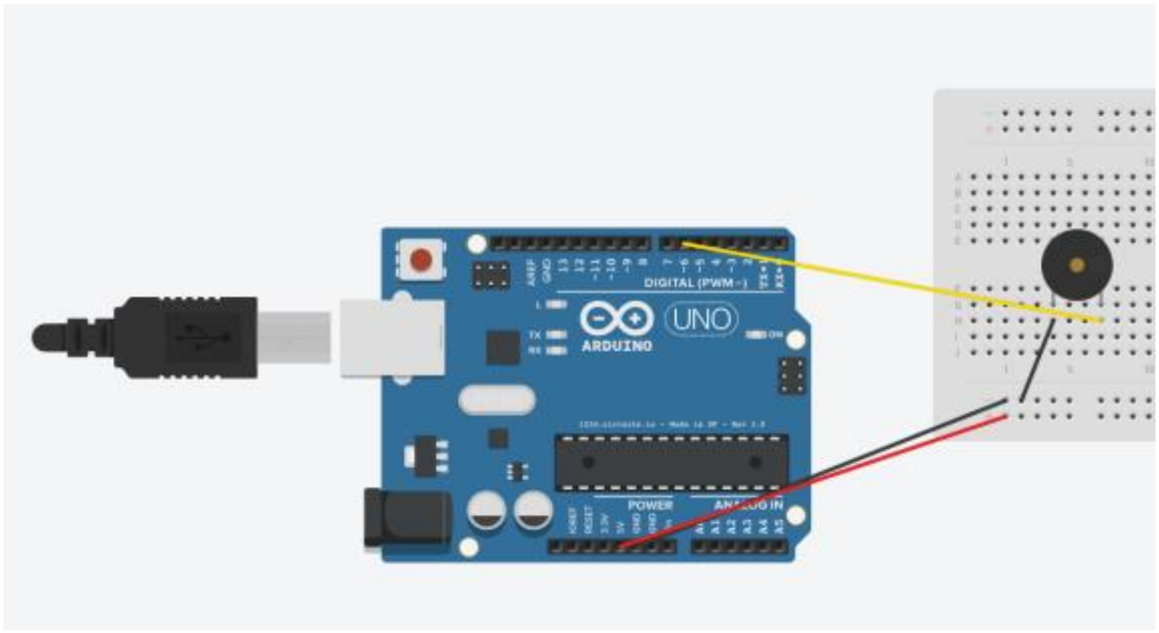
- Arduino
- Breadboard
- Jumper Wires (Male to Male)
- Passive buzzer (If you apply 5v to it and it just makes a small CLICK, it is passive. If you apply 5v to it and it emits a steady sound, it is an active buzzer).

Expected outcome:

The reader will be able to generate tones with a passive buzzer using PWM signals to emulate an ac signal.

Procedure:

Wire the circuit as follows:



CODE:

```
//----Begin Code (copy from here)----
//Variables:

void setup() {
  // put your setup code here, to run once:
pinMode(6, OUTPUT);
}

void loop() {
  // put your main code here, to run repeatedly:
analogWrite(6, random(0,255)); //Send a random PWM signal to pin 6
delay(500); //wait half a second before changing pitch
}

//----End Code (copy to here)----
```

Notes:

This is not quite good enough to be used to playback music, but it is certainly useful enough to emit tones for warnings, or simple MIDI sounds!

Project 10
(Active Buzzer)

Project Aim/Description:

The reader will be able to use an active buzzer in their projects. An active buzzer has an oscillator inside which resonates without the need for an AC voltage to be applied. You can simply connect it to a power source and it will emit a steady tone.

What you need:

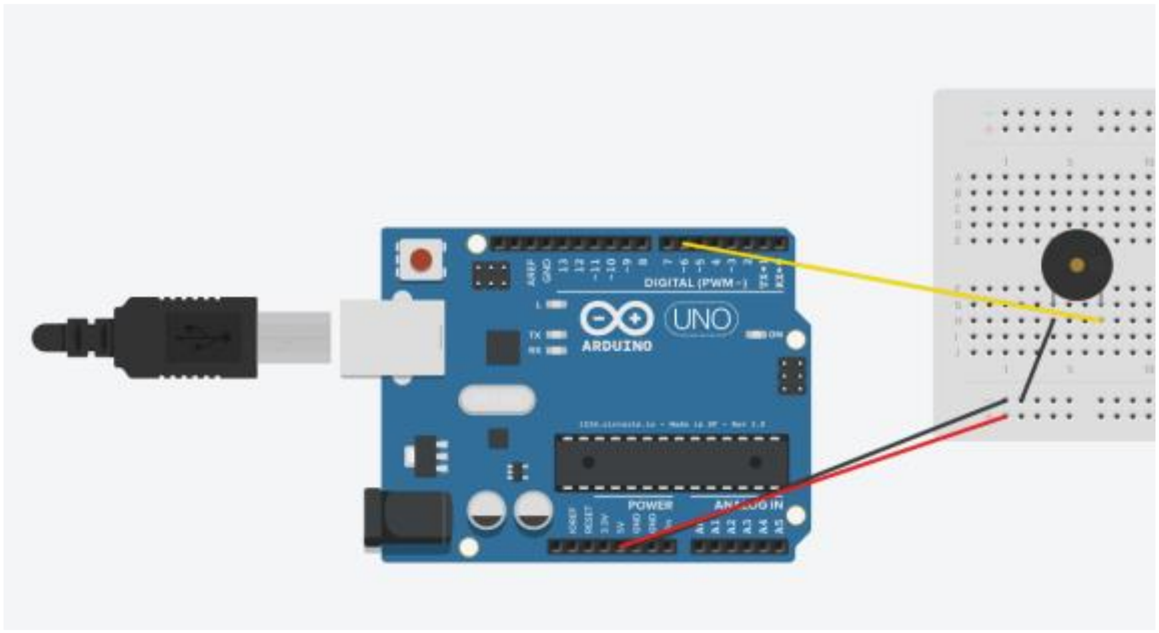
- Arduino
- Breadboard
- Jumper Wires (Male to Male)
- Active Buzzer (If you apply 5v to it and it just makes a small CLICK, it is passive. If you apply 5v to it and it emits a steady sound, it is an active buzzer).

Expected outcome:

The reader will be able to use an active buzzer to emit tones for their projects, as well as see the difference between the last project and this one to gain an understanding of the difference between the 2 types of buzzers.

Procedure:

Wire the circuit as follows:



CODE:

```
//----Begin Code (copy from here)----
//Variables:
void setup() {
  // put your setup code here, to run once:
pinMode(6, OUTPUT);
}
void loop() {
  // put your main code here, to run repeatedly:
digitalWrite(6, HIGH);
delay(500);
digitalWrite(6, LOW);
delay(500);
}
//----End Code (copy to here)----
```

Notes:

We simply turn the buzzer on and off repeatedly in this project, it is quite loud! An active buzzer is handy when you don't have the need for specific tones, or no spare PWM pins left but you still need to make sounds. You CAN apply a PWM signal to it to change the tone in the same way dimming an LED works, why don't you try that?

Project 12
(Button)

Project Aim/Description:

Buttons seem like a simple idea, press them and you see a change in the circuit right? This project will demonstrate how to use a button properly. It is a little more complex than you would assume, we need to use a pull down resistor to ensure the button does not change state on its own…As well as "debounce" the button, so when you press it we get a clean "on" and "off" state change, rather than rapidly changing states during the switches operation.

What you need:

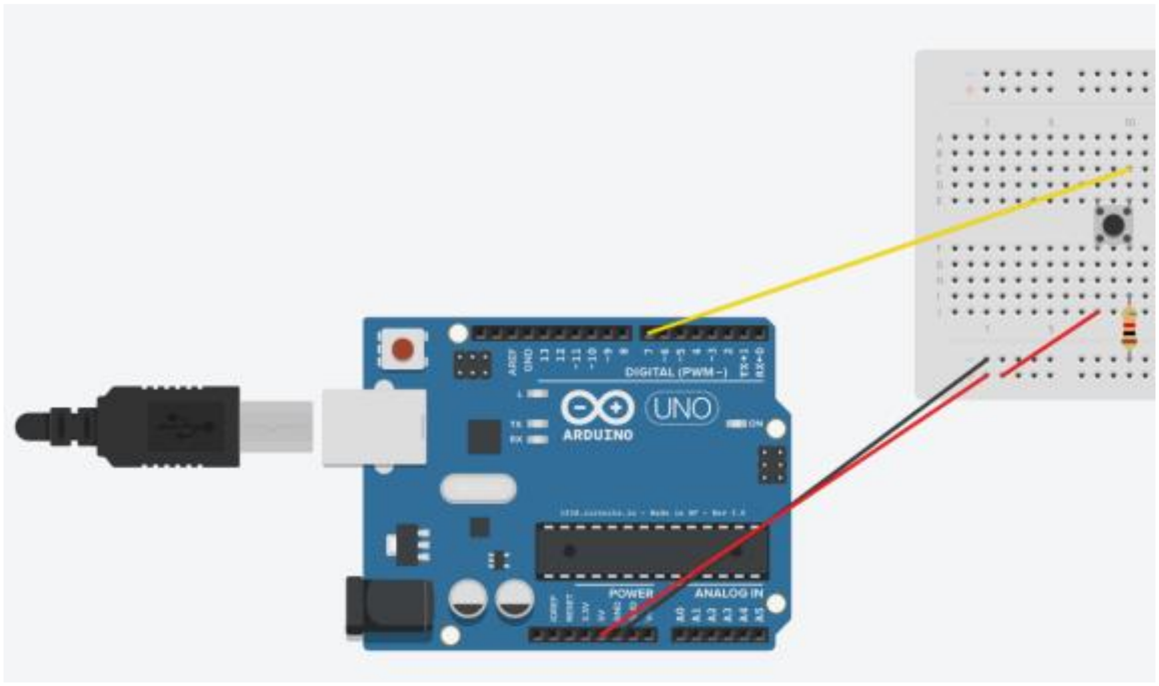
- Arduino
- Breadboard
- Jumper Wires (Male to Male)
- Button
- 1x 220 Ohm Resistor
  (Brown, Black, Black, Red, Red)

Expected outcome:

Buttons can be tricky, they have a physical connection inside that can rapidly flicker between on and off when pressed. This is called bouncing, and can be difficult to fix. There are 2 ways to tackle the problem…Either in circuitry or in code. In this project we will use a coded solution rather than a component based solution. Feel free to google using a capacitor to debounce a switch if you would rather a non coded debounce method.

Procedure:

Wire the circuit as follows:



CODE:

```
//----Begin Code (copy from here)----
//Variables:
int buttonPin = 7;          //Pin button is connected to
int buttonState;            //Variable to store button state
int lastButtonState = LOW;   //Variable to store last state
long lastDebounceTime = 0;   //Variable to store time button was last pressed
long debounceDelay = 50;     //Minimum amount of time in milliseconds to wait between
presses. (1000 milliseconds per second)
void setup() {
  // put your setup code here, to run once:
Serial.begin(9600);
pinMode(buttonPin, INPUT);
Serial.println("Serial communication established...\nPress Button to test.");
}
void loop() {
  // put your main code here, to run repeatedly:
int reading = digitalRead(buttonPin);
if (reading != lastButtonState) {
lastDebounceTime = millis();
}
if ((millis() - lastDebounceTime) > debounceDelay) {
if (reading != buttonState) {
buttonState = reading;
if (buttonState == HIGH) {
```

```
//Button pressed
buttonPressed();
}
}
}
lastButtonState = reading;
}
void buttonPressed()
{
Serial.println("Press!");
}
//----End Code (copy to here)----
```

Notes:

In this project we create a timer that checks if the last time the switch changed states was sooner than X amount of seconds. If it was we ignore it, otherwise we know the press was genuine, and trigger the state change in software.

Project 13

(LM35 Temperature Sensor)

Project Aim/Description:

This project uses an LM35 temperature sensor to communicate to the Arduino the ambient temperature. It is an extremely accurate sensor and is surprisingly easy to use! We also use a bit of math to derive the temperature in Celsius!

What you need:

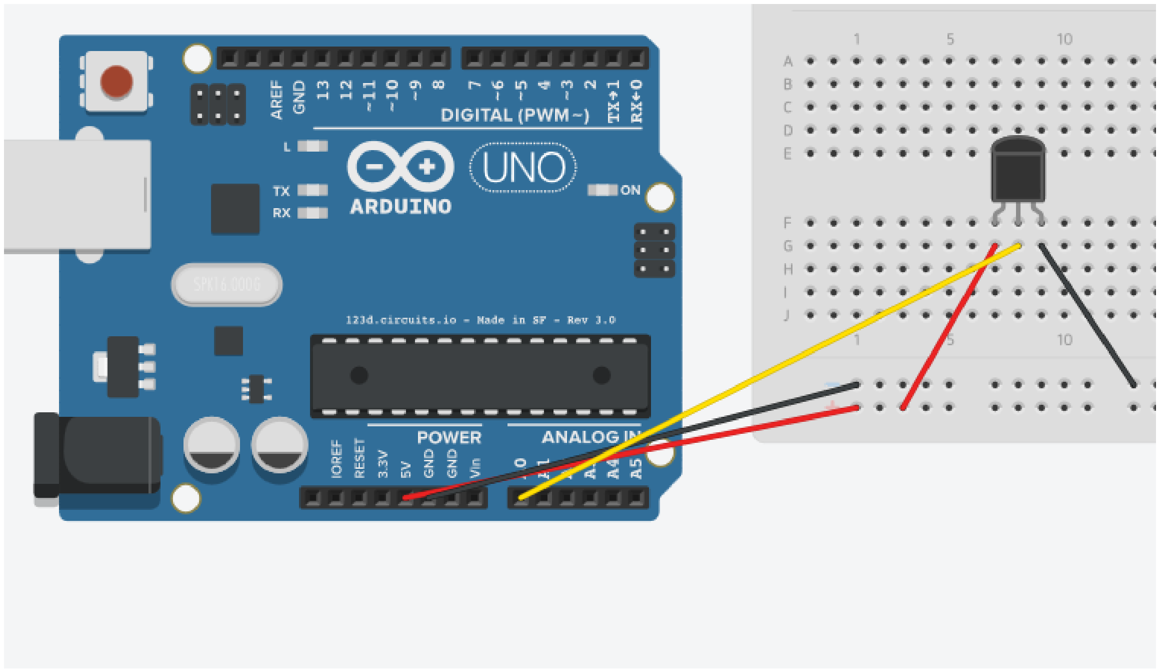
- Arduino
- Breadboard
- Jumper Wires (Male to Male)
- LM35 Temperature Sensor

Expected outcome:

The reader will be able to use an LM35 based temperature sensor to read the ambient temperature, this project creates a real working temperature logger!

Procedure:

Wire the circuit as follows:



CODE:

```
//----Begin Code (copy from here)----
//Variables:
int varPin = A0; //A0 is the analog in pin
int val = 0;     //This will store the current value from the sensor
int oldVal = 0;  //This will store the last value from the sensor to compare.

void setup() {
  // put your setup code here, to run once:
Serial.begin(9600);
Serial.println("Serial connection established..\nTemperature logging started!");
}

void loop() {
  // put your main code here, to run repeatedly:
oldVal = val; //Store the last value in the variable
val = analogRead(varPin); //Store the new value in its variable
if (oldVal != val)       //If the temperature has changed, tell us the new temp
{
Serial.print("Temp: ");
Serial.print(5.0 * val * 100 / 1024); //please see note at bottom regarding this formula.
Serial.println(" Degrees(Celsius)");
}
delay(500); //Wait half a second, then do it again. (This allows the line to settle between reads.
Not really required, but it does remove a bit of jitter
}

//----End Code (copy to here)----
```

Notes:

The formula used to convert to temperature was derived from the data sheet. In this case we have the following elements:
5.0 is the voltage we are inputting.
100 is the multiplier to convert from mV to V
1024 is the maximum resolution of the analog input on the arduino.
Essentially we know from the data sheet that 10mV = 1 degree, which means if we convert and clean the reading using the above formula, we get the temperature in Degrees Celsius -/+ half a degree.

You will notice the temperature will display in the serial monitor. If you touch the sensor you will see the reading go up, and once you let go you will see it drop back down as the sensor cools.

Project 14

(Sound Sensor/Microphone)

Project Aim/Description:

This project uses a microphone module to detect and measure sound. As sound waves travel through the air they cause everything they come into contact with to vibrate. Using this sensor we can measure these vibrations. The Arduino by itself is not the best solution for sound processing and recording, although with a cheap module it certainly can do it. For this project we will look at measuring the loudness of the sound. You could also use it to measure the pitch of the sound with a little more programming. (We will be measuring the raw volume, to measure pitch you would want to count the variance over time to work out the cycles a second, then compare this to a database of hz – pitch).

What you need:

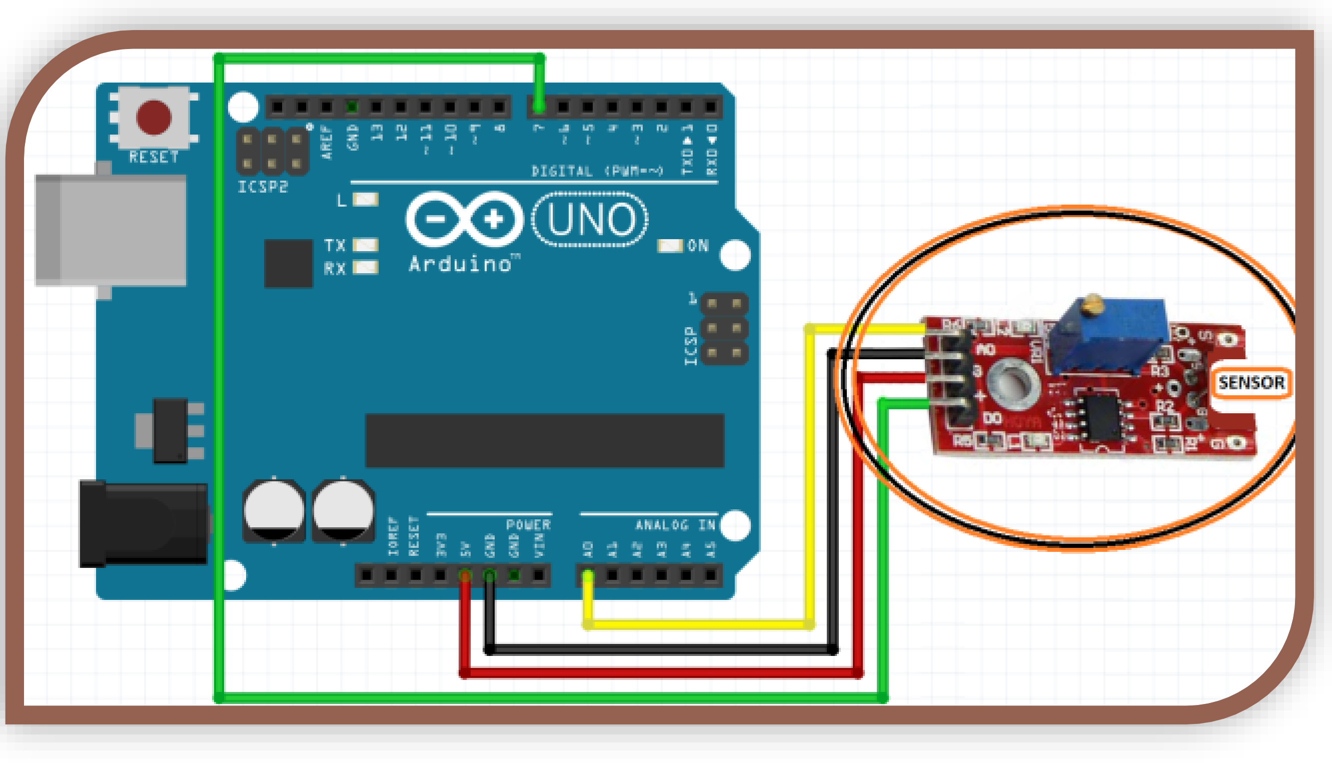
- Arduino
- Jumper Wires (Male to Female)
- Sound Sensor Module

Expected outcome:

The reader will be able to interface a sound sensor module to their Arduino and use it in their projects. In the configuration shown we will be measuring raw sound levels.

Procedure:

Wire the circuit as follows:



CODE:

```
//----Begin Code (copy from here)----
//Variables:
int val = 0;     //This will store the current value from the sensor
int dval = 0;    //This will store the current value from the onboard comparator.
int oldval = 0;  //Variable to store the old analog value
int olddval = 0; //variable to store the old digital value

void setup() {
  // put your setup code here, to run once:
pinMode(7, INPUT); //Set pin 7 to digital input. Analog pins are automatically input.
Serial.begin(9600);
Serial.println("Serial connection established...\nVolume logging started!");
}

void loop() {
  // put your main code here, to run repeatedly:
oldval = val; //Store the last analog value in oldval
olddval = dval; //store the lastdigital value in olddval
val = analogRead(A0); //Store the new value in its variable
dval = digitalRead(7); //Store the new digital value in its variable
```

```
if (oldval != val) {
Serial.print("Analog Read: ");
Serial.println(val); //Print the sensor value.
Serial.print("Digital Read: ");
Serial.println(dval); //Print the differential comparator result
}
delay(100); //Wait half a second, then do it again. (This allows the line to settle between reads.
Not really required, but it does remove a bit of jitter
}
//Remember you can always comment out the analog or digital values above to see only the
result you want.
//----End Code (copy to here)----
```

Notes:

The digital comparator takes an average value and compares the microphone read to it. You can adjust the value by turning the screw, I had to "unscrew" mine about 15 turns before it was at the correct threshold. The indicator LED next to the adjustment screw will be on or off. You should turn the adjustment screw until the LED is right on the edge of changing state, then from there you can fine tune it as you wish…you might have this set to a certain level to alert you if the volume is too loud, or perhaps set very low in a silent room to alert you of activity etc.

Project 15
(Moisture Level Sensor)

Project Aim/Description:

This project will demonstrate the use of the moisture level sensor. This sensor is used to detect the level of moisture in soil or other medium. This will not tell you volume or humidity in air. This of it as a digital measuring device to tell you when to water your plants etc.

What you need:

- Arduino
- Breadboard
- Jumper Wires (Male to Male)
- Moisture Sensor

Expected outcome:

The reader will be able to integrate a moisture sensor into their Arduino projects. The code is easily modifiable to be used in future projects, such as a plant watering reminder, drought monitor or skin hydration level sensor.

Procedure:

Wire the circuit as follows:

You should connect the sensor as follows:

SENSOR PIN	ARDUINO PIN
S	A0
+	3.3V
-	GND

CODE:

```
//----Begin Code (copy from here)----
//Variables:
int val = 0;     //This will store the current value from the sensor
int oldval = 0;  //Variable to store the old analog value

void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);
  Serial.println("Serial connection established...\nMoisture logging started!");
}

void loop() {
  // put your main code here, to run repeatedly:
  oldval = val; //Store the last analog value in oldval
  val = analogRead(A0); //Store the new value in its variable
  if (oldval != val) {
  Serial.print("Moisture level: ");
  Serial.println(val); //Print the sensor value.
  }
}
//----End Code (copy to here)----
```

Notes:

I had some fun while putting this project together. If you moisten your finger and wipe it on the sensor, you can actually watch the value drop as the moisture evaporates off the sensor…This is a very cool sensor despite its simplicity.

Project 16
(RTC – Real Time Clock Module)

Project Aim/Description:

This project will interface the Arduino with a RTC module based on the DS1302 chip. The reader will be able to set a time and date, and the module will keep it up to date from there, allowing calendar and clock access easily in all future projects.

What you need:

- Arduino
- Breadboard
- Jumper Wires (Male to Male)
- RTC Module

Expected outcome:

The reader will install a library manually, interface with an RTC chip and set the time. The chip will then keep the time very accurately indefinitely. If this chip was left unplugged for 10 years, then re integrated (without re-setting the time) it would still display the correct time, with a small margin for error.)

Useful for applications when date and time are needed, as the Arduino only knows how many seconds it has run since powered on, and has no idea of time outside its own scope.

Procedure:

Download the library manually from:
https://github.com/msparks/arduino-ds1302/archive/master.zip
You should extract all files in this zip folder into a folder named RTC inside your Arduino Libraries folder. Default location is:
C:\Program Files (x86)\Arduino\libraries
Navigate to the libraries folder, create a folder called RTC and extract all files from the archive there.

If you are having trouble, you can use google to search: Manually installing Arduino Library.

Wire the circuit as follows:

RTC MODULE PIN	Arduino PIN
VCC	3.3V
GND	GND
CLK	6
DAT	7
RST	8

CODE:

```
//COPY FROM THIS LINE (You may include this line in the copy operation, but it is not necessary
to do so)

#include <stdio.h>
#include <DS1302.h>

namespace {
//This section is used to define certain parameters for the library. This is not usually needed,
however the author of this particular library has coded it in this way.
//This will create the object we use to interact with the RTC module
DS1302 RTC(8, 7, 6);
//Now we will create a function to format the data as strings for easier use in our projects.
String dayAsString(const Time::Day day) {
  switch (day) {
    case Time::kSunday: return "Sunday";
    case Time::kMonday: return "Monday";
    case Time::kTuesday: return "Tuesday";
    case Time::kWednesday: return "Wednesday";
    case Time::kThursday: return "Thursday";
    case Time::kFriday: return "Friday";
    case Time::kSaturday: return "Saturday";
  }
  return "(invalid day)";
}
//And finally a function to get the data from the module, and format it for printing to the serial
window.
void PrintRTC() {
  Time t = RTC.time();
  String day = dayAsString(t.day);
  char tmp[50];
  snprintf(tmp, sizeof(tmp), "%s %04d-%02d-%02d %02d:%02d:%02d", day.c_str(), t.yr, t.mon,
t.date, t.hr, t.min, t.sec);
  Serial.println(tmp);
}
}
void setup() {
Serial.begin(9600);
RTC.writeProtect(false);
RTC.halt(false);
Time t(2016, 1, 2, 1, 2, 3, Time::kTuesday); //Sets Date to 1 Feb, 1:02:03am also sets Day to
Tuesday. Set this to the current time.
RTC.time(t); //Sends the time as you specified above to the RTC module, and begins the clock.
}

void loop() {
PrintRTC(); //Prints the current time as formatted above.
delay(1000); //Delays for one second, then returns to start of loop.
}
//COPY TO THIS LINE (You may include this line in the copy operation, but it is not necessary to
do so)
```

Notes:

An RTC is a Real Time Clock. This device is a trickle charge real time clock chip, with the oscillator and battery integrated into the PCB. This is an all-in-one solution and communicates using serial protocols with the Arduino.

The library was written by a 3rd party, however it works very well in this application. Feel free to email me with questions and I will answer if possible.

Project 17

(Stepper Motor and Driver)

Project Aim/Description:

A stepper motor is a motor that can very precisely move, rather than just spinning randomly it can move a certain number of degrees for instance. These motors are used in 3D printing systems and many other high quality projects. A very handy piece of hardware to have.

What you need:

- Arduino
- Breadboard
- Jumper Wires (Male to Male)
- Stepper Motor (28BYJ-48 – 5 wires)
- Stepper Controller (ZC-A0591)

Expected outcome:

The reader will be able to access and control a stepper motor with their Arduino. Personally I intend on making a cool clock out of mine, that will use the RTC module to keep time and the stepper motor and servo to control hands for seconds and hours.

Procedure:

Download the “CheapStepper” Library through library manager as explained at the start of this guide.

Wire the circuit as follows:

Plug the Stepper motor connector into the connector on the ZC-A0591 board.

Connect the ZX-A0591 board to the Arduino as follows:

ZC-A0591 PIN	ARDUINO PIN
1N1	8
1N2	9
1N3	10
1N4	11
-	GND
+	+5 – 12V

CODE:

```
//COPY FROM THIS LINE (You may include this line in the copy operation, but it is not necessary
to do so)
#include <CheapStepper.h>
CheapStepper motorObj (8,9,10,11); //Create the motor object and assign to the pins we used.
bool moveClockwise = true;
void setup() {
 // put your setup code here, to run once:
motorObj.setRpm(12);
Serial.begin(9600);
moveDegrees(180);
}
void loop() {
 // put your main code here, to run repeatedly:
 int stepsLeft = motorObj.getStepsLeft(); //Ask the controller how many steps remain
 if (stepsLeft == 0) {
  Serial.println("Move complete!"); //Print completion status to serial monitor and don’t run
motor
 }
 else {
 motorObj.run(); //Run motor as more steps are needed
 }
}
void moveDegrees(int degIn) {
 //4096 = full turn.
 int result = map(degIn, 0, 360, 0, 4096); //Convert input to steps
 motorObj.newMoveTo(moveClockwise, result); //Move correct number of steps
}
```

//COPY TO THIS LINE (You may include this line in the copy operation, but it is not necessary to do so)

Notes:

Project 14

(Servo Motor)

Project Aim/Description:

What you need:

- Arduino
- Breadboard
- Jumper Wires (Male to Male)

Expected outcome:

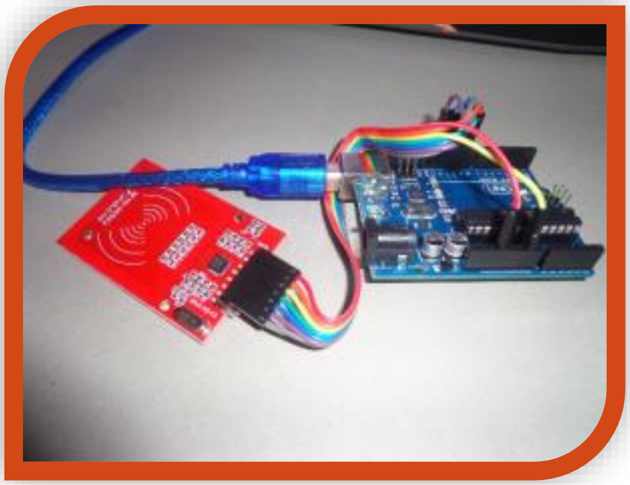
Procedure:

Wire the circuit as follows:

CODE:

Notes:

Project N
(RFID Module <TAG Dump>)



Project Aim/Description:

This project will show you how to wire the RFID module, as well as provide code that will simply dump the contents of the cards memory to the Serial Monitor.

What you need:

- Arduino
- RFID Module
- TAGs or Cards compatible with your module

Expected outcome:

You will be able to wire an RFID module to your Arduino, and access the memory on TAGs and Cards that are compatible with your reader.

Note: RFID TAGs are secured, the protocol has encryption and the keys are not stored on the card itself. I will include the ability to use codes to decrypt and encrypt your card, however if you do not have access to the private key there is no way to simply bypass it.

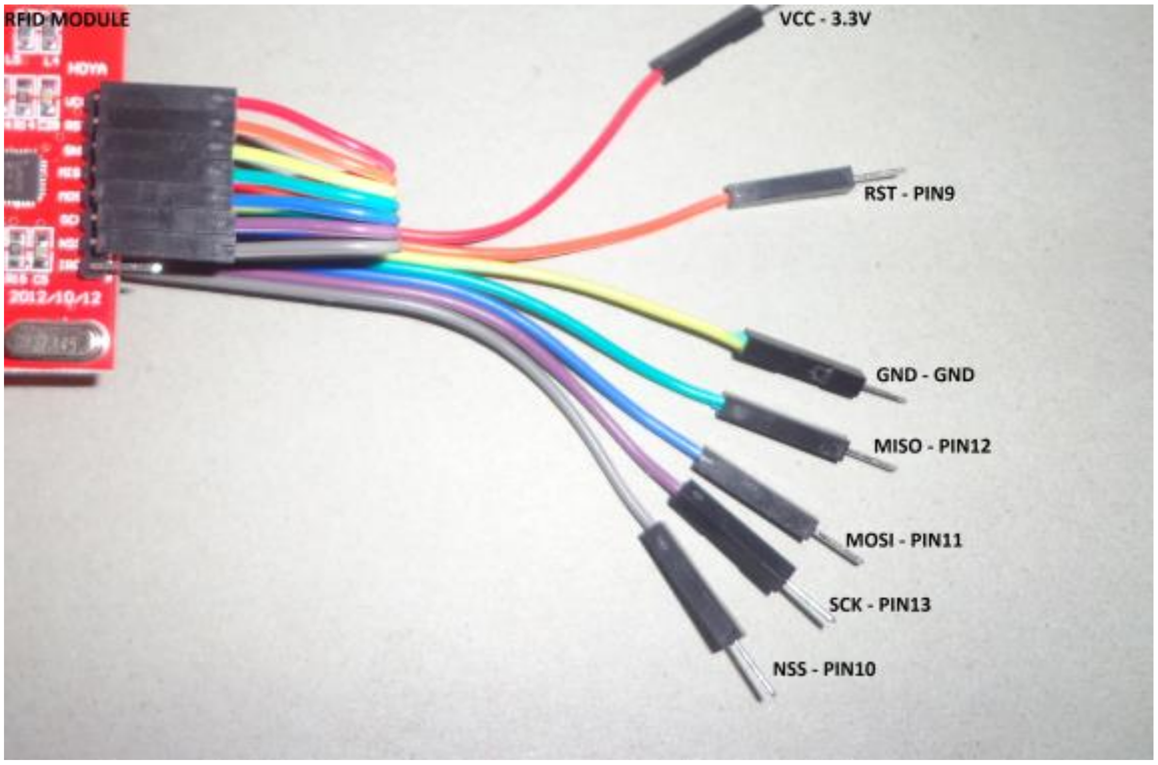
Procedure:

Begin by installing the MFRC522 library as instructed at the beginning of this guide. This will install the necessary files on your computer to tell your Arduino how to use and communicate with the module.

Next you should wire the module as follows:

MODULE PIN	ARDUINO PIN
VCC (+ Voltage)	3.3v
RST (Reset)	9
GND (Ground)	GND
MISO (Master in Slave out)	12
MOSI (Master out Slave in)	11
SCK (Serial Clock)	13
NSS (Negative Slave Select)	10
IRQ	NOT CONNECTED

Please see the image below for an example of the connections:



Once you have done that, upload the following code to the Arduino, and open the serial monitor. As instructed in the serial output, introduce your tag or card to the reader and you will see the data stored on it in a table.

CODE:

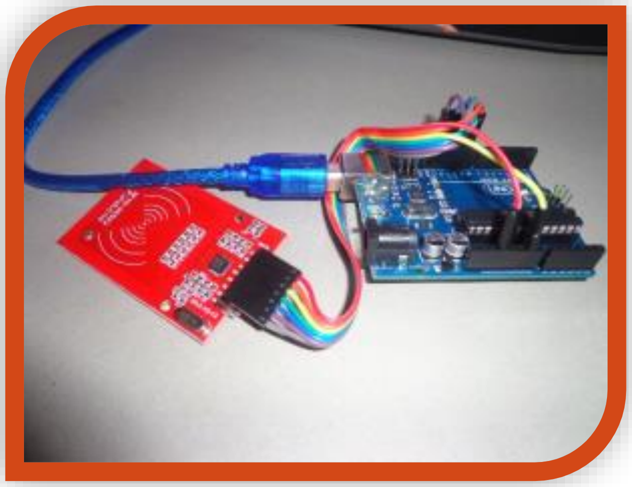
```
//***COPY FROM HERE***
//(C) Tyson Popynick for Aus Electronics Direct 2016
/*
Pin connections for the RFID Module:
VCC (Voltage +)
RST (RESET)
GND (Voltage -)
MISO (Master input - Slave Output)
MOSI (Master Output - Slave Input)
SCK (Serial Clock)
NSS (Negative Slave Select)
IRQ (Interrupt ReQuest)

RFID  -  Arduino
VCC  - 3.3V
RST  - PIN 9
GND  - GND
MISO - PIN 12
MOSI - PIN 11
SCK  - PIN 13
NSS  - PIN 10
IRQ  - NO CONNECTION
*/
#include <SPI.h>           //Tell the Arduino it will use the SPI interface
#include <MFRC522.h>       //Tell the Arduino it will use the MFRC522 protocol
MFRC522 RFIDModule(10, 9);  //Create the reader instance

void setup() {
  Serial.begin(9600); // Init serial to PC
  SPI.begin();      // Init serial to reader
  RFIDModule.PCD_Init(); //Init reader interface
  Serial.println("Scan Card or FOB to read data...");
}

void loop() {
  //This instruction tells the Arduino to keep scanning until it sees a new card or fob
  if ( ! RFIDModule.PICC_IsNewCardPresent()) {
    return;
  }

  //This instruction tells the Arduino to select and read the card or fob once it finds one
  if ( ! RFIDModule.PICC_ReadCardSerial()) {
    return;
  }

  //Finally, if a card or fob is found, this instruction dumps the data over serial to the PC
  RFIDModule.PICC_DumpToSerial(&(RFIDModule.uid));
}
//Copy to here.
//Tyson Popynick - Aus Electronics Direct. - Prot3us1@gmail.com
```

Notes:

This project simply blindly dumps all the data on the card. Please see the next project in order to write data. Please not the procedure for writing is a bit more complex due to the need to have safety measures in place to stop you accidentally overwriting important areas of the card.

Project N

(RFID Module <Read & Write>)



Project Aim/Description:

This project follows on from the previous project (RFID Module <TAG Dump>. This time we will write our own custom data to the tags. Remember that you can corrupt the tags by writing data to the wrong areas. I have coded in protection to a reasonable extent, while trying to keep the code simple. Please beware that it is possible to render a card unusable if you are not careful. Cards and tags are not expensive however, so it is always a good idea to grab some spares!

What you need:

- Arduino
- RFID Module
- TAGs or Cards compatible with your module

Expected outcome:

You will be able to read and write to your RFID tags. This project aims to demonstrate R&W capabilities for you to use in your own projects.

Procedure:

Follow the previous project, however upload the code that follows instead.

Follow the instructions in the comment at the top of the code. You will need to uncomment code//comment code depending on what you wish to do.

Feel free to google search or email me for additional help as needed. My email address is on the title page of this document.

CODE:

```
//***COPY FROM HERE***
//(C) Tyson Popynick for Aus Electronics Direct 2016
/*
Pin connections for the RFID Module:
VCC (Voltage +)
RST (RESET)
GND (Voltage -)
MISO (Master input - Slave Output)
MOSI (Master Output - Slave Input)
SCK (Serial Clock)
NSS (Negative Slave Select)
IRQ (Interrupt ReQuest)

RFID  -  Arduino
VCC  - 3.3V
RST  - PIN 9
GND  - GND
MISO - PIN 12
MOSI - PIN 11
SCK  - PIN 13
NSS  - PIN 10
IRQ  - NO CONNECTION

HOW TO USE:
1. If there is a security key assigned to the card or fob, enter it belkow in place of 0xFF.
2. Select the block we are interested in accessing. In the demo we will use block 2.
3. If you would like to write to the card, enter the 16 bytes of data in place of AusElecDirect__
below.
4. If you would like to delete a block, comment the AusElecDirect line, and uncomment the 0,0,0
etc line.
5.

*/
#include <SPI.h>            //Tell the Arduino it will use the SPI interface
#include <MFRC522.h>         //Tell the Arduino it will use the MFRC522 protocol
MFRC522 RFIDModule(10, 9);     //Create the reader instance
MFRC522::MIFARE_Key keyStore;  //Create a store for the information on the card.
```

```
void setup() {
  Serial.begin(9600); // Init serial to PC
  SPI.begin();      // Init serial to reader
  RFIDModule.PCD_Init(); //Init reader interface
  Serial.println("Scan Card or FOB to read data...");
  /*
   We need to prepare the data store for use. If this is a brand new fob or card the security key
will be 0xFF by default.
   This will be used for reading and writing.
   If the card was written to by another party, change 0xFF to the key below.
   */
   for (byte i = 0; i < 6; i++) {
    keyStore.keyByte[i] = 0xFF;
   }
}

int block = 2;                                    //Which block should be written. There are 63 on a 1kb
card.
byte blockcontent[16] = {"AusElecDirect__"};              //16 byte array to write.
//byte blockcontent[16] = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0}; //Array of 0. Used to 'delete' a block.
byte readbackblock[18];//DO NOT MODIFY                    18 byte array used to store data that is
read. DO NOT MODIFY.
int delayTime = 1000; //Miliseconds to wait between reads.
void loop() {
  //This instruction tells the Arduino to keep scanning until it sees a new card or fob
  if ( ! RFIDModule.PICC_IsNewCardPresent()) {
    return;
  }

  //This instruction tells the Arduino to select and read the card or fob once it finds one
  if ( ! RFIDModule.PICC_ReadCardSerial()) {
    return;
  }

  //Finally, if a card or fob is found the next instructions will run:
  //RFIDModule.PICC_DumpToSerial(&(RFIDModule.uid)); //this instruction dumps the data over
serial to the PC.
  //writeBlock(block, blockcontent); //Writes the blockcontent array to the block we selected.

  readBlock(block, readbackblock); //Read selected block into the array we created.
  //Uncomment the write above if you wish to write to the fob or card.

  Serial.print("Block Contents: ");  //The following will display the data we read into the array in
the serail monitor.
  for (int x = 0; x < 16; x++) { //Loop through blocks and print contents.
    Serial.write(readbackblock[x]);
  }

  Serial.print("\nRead Complete.\n\n");    //print a new line for readability.
  delay(delayTime);      //Set the delay above as needed. Default is 1 second.
  RFIDModule.PCD_Init(); //This will reinitialize the reader after each set of commands, allowing
you to perform multiple operations without resetting the Arduino.
  Serial.println("Scan Card or FOB to read data...");
}
//Additional Functions used to authenticate and read/write to/from blocks. If you are getting to
this point of programming you should read the
//.CPP file, and work from there.
//http://cache.nxp.com/documents/data_sheet/MF1S50YYX_V1.pdf << Datasheet for the
hardware
//https://github.com/miguelbalboa/rfid/blob/master/MFRC522.cpp << CPP for functions.
int writeBlock(int blockNumber, byte arrayAddress[])
{
  int largestModulo4Number=blockNumber/4*4;
  int trailerBlock=largestModulo4Number+3;//determine trailer block for the sector
  if (blockNumber > 2 && (blockNumber+1)%4 == 0){Serial.print(blockNumber);Serial.println(" is
a trailer block:");return 2;}
  //Authenticate to access block
  byte status = RFIDModule.PCD_Authenticate(MFRC522::PICC_CMD_MF_AUTH_KEY_A,
trailerBlock, &keyStore, &(RFIDModule.uid));
  if (status != MFRC522::STATUS_OK) {
       return 3;
  }
  //Write block
  status = RFIDModule.MIFARE_Write(blockNumber, arrayAddress, 16);
  if (status != MFRC522::STATUS_OK) {
         return 4;
  }
  Serial.println("block was written");
}
int readBlock(int blockNumber, byte arrayAddress[])
{
  int largestModulo4Number=blockNumber/4*4;
  int trailerBlock=largestModulo4Number+3;
  //Authenticate to access block
  byte status = RFIDModule.PCD_Authenticate(MFRC522::PICC_CMD_MF_AUTH_KEY_A,
trailerBlock, &keyStore, &(RFIDModule.uid));
  if (status != MFRC522::STATUS_OK) {
       return 3;
  }
  //Read block
  byte buffersize = 18;
  status = RFIDModule.MIFARE_Read(blockNumber, arrayAddress, &buffersize);
  if (status != MFRC522::STATUS_OK) {
        return 4;
  }
}
//Copy to here.
//Tyson Popynick - Aus Electronics Direct. - Prot3us1@gmail.com
```

Notes:

This code can be used to write many cards by setting up the data to be written, then simply swiping each card in turn.
You can then use the previous project to write a sketch to verify if the cards are correct and perform some task as needed!

Thank you for reading!