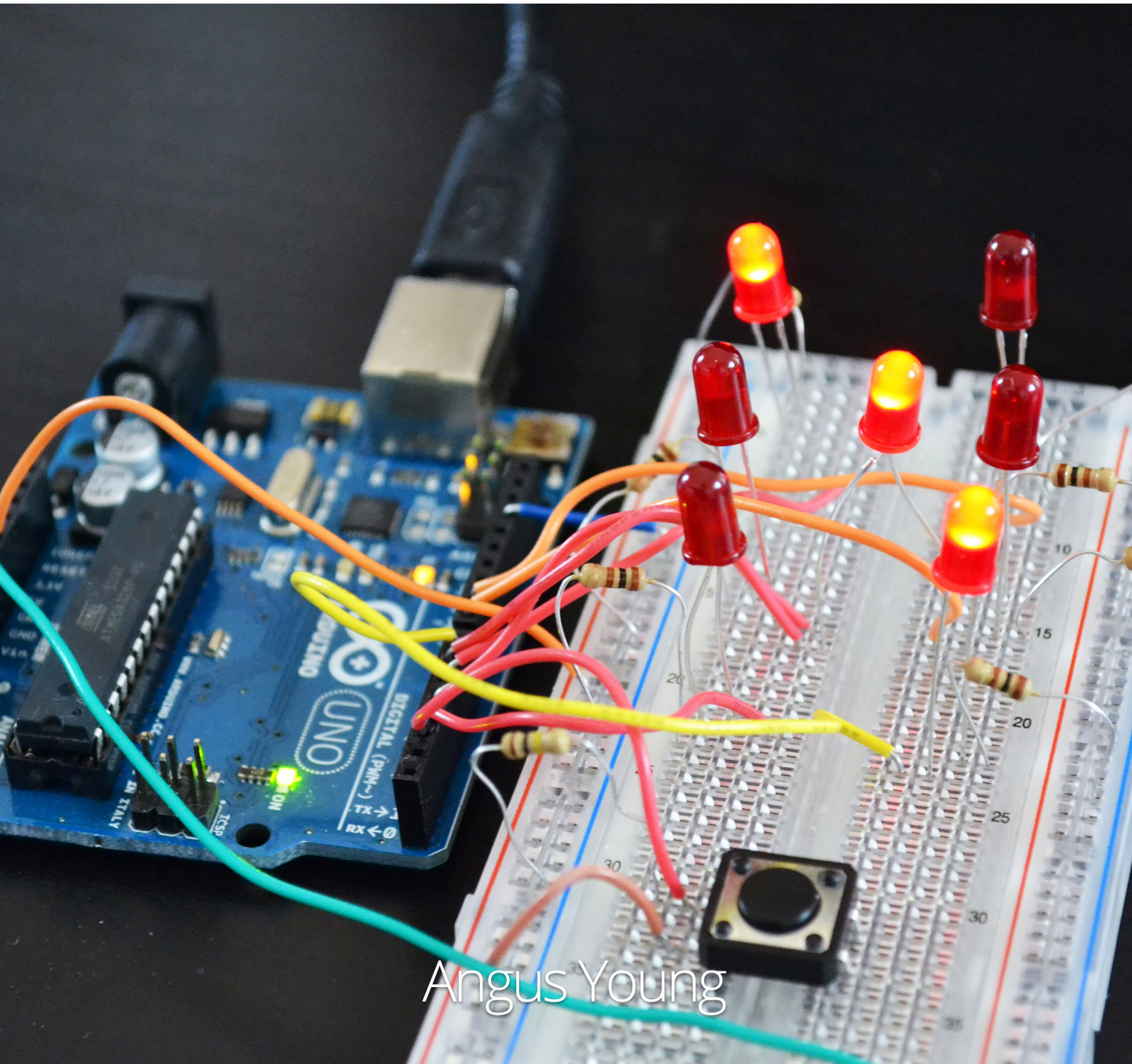
Ultimate Book of **Arduino**



Angus Young

Copyright © 2018 Pi My Life Up, All Rights Reserved

No parts of this book may be reproduced in any form or by any electronic means, including information storage and retrieval systems without permission from the copyright holder.

For permission contact Pi My Life Up at:

support@pimylifeup.com

The tutorials in this book are for general information purpose only.

While we strive to keep our information up to date, we do not make any warranties about the completeness, reliability and accuracy of this information.

Any action you take upon the information on within this book is strictly at your own risk. We will not be liable for any losses and damages in connection with the use of the information within this book

Arduino is a trademark of Arduino AG, https//www.arduino.cc
All trademarks are the propety of their resperctive owners.

Contents

Getting Started
With the Arduino

Quick rundown on the Arduino

The Arduino is an open-source microcontroller board developed by the company of the same name. The Arduino serves as an open-source prototyping platform that allows you to create and write software that can interact with hardware through its various analog input pins and digital input/output pins.

The analog pins that are available on the Arduino makes it a lot easier to deal with various sensors and other devices that provide an analog output. This is unlike the Raspberry Pi for example where you would need to convert the analog signal to digital before being able to deal with it.

Using the Arduino, you can interact with various sensors, motors, lights and any other electronic device that allow physical interactions. You can have the Arduino run independently or communicate with another software package running on a connected device; this will enable you to run more complicated tasks that the Arduino itself usually wouldn't be able to handle due to its limited processing power.

To interact with the Arduino, you typically are going to be utilizing its programming language. The Arduino programming language is merely a specific implementation of a set of C/C++ functions, so it is straightforward to pick up if you have basic knowledge of any programming language. Don't worry, even if you are not used to programming at all, we will walk you through our script to give you an idea of what they are doing.

Writing your programs for your Arduino is a reasonably simple process thanks to the Arduino IDE. You will be utilizing the Arduino IDE either on your Windows, Mac or Linux PC since the Arduino itself isn't powerful enough to act like a desktop, you can't write programs directly on the device.

The Arduino is unlike the Raspberry Pi which features a fully fledged operating system. Once you are finished with writing your program, you can directly upload your application to the board by using this same IDE.

There are many different versions of the Arduino, with one of the most popular and most will doccumented versions of these being the Arduino Uno.

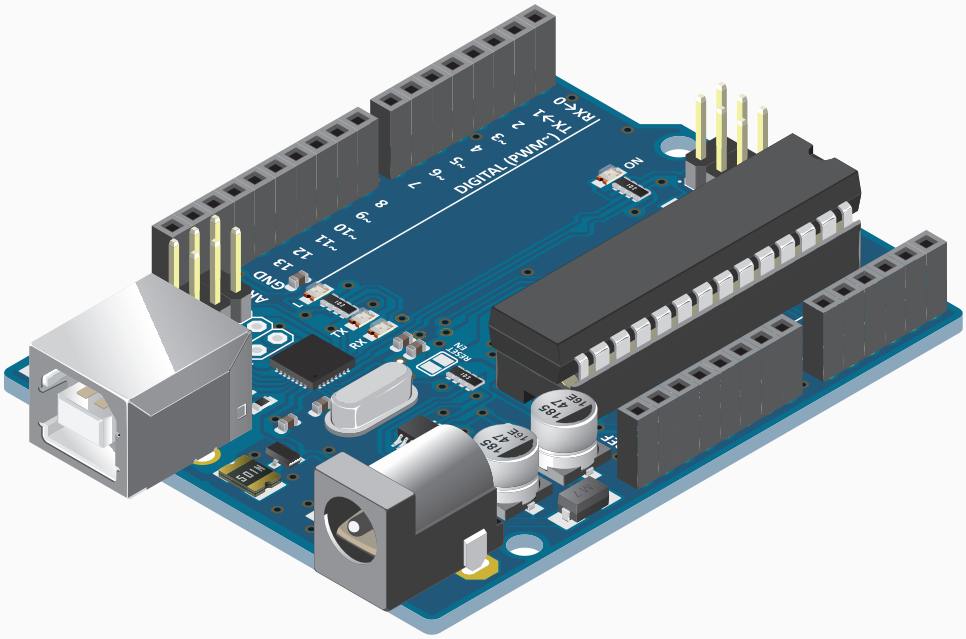
We will be utilizing the Arduino Uno board in all of our tutorials within this book as we found it to be the easiest board to get started with, especially with the ability to power the device off of USB and also write programs to it easily via the USB port.

Please note that to get the most out of your Arduino Uno board you will have to buy various different shields for it, these "shields" allow you add additional functionality to the base board such as giving it an ethernet connection.

Getting Started with the Arduino

1. Of course, the very first thing you must do before you get started with the Arduino is to obviously purchase one. If you have already bought one, then you can just skip onto our next step.

For our tutorials we decided to make use of the Arduino Uno, this is the most doccumented board out of the whole Arduino family. As an added bonous it is regarded as the best Arduino board to get started with electronics and coding and is perfect for first timers.



When purchasing your Arduino, we highly recommend you stick with the official ones, while you can get clone boards for lower prices they often skimp on certain parts and don't hold up to the same quality the official boards are known for.

As a bonus, you will also be helping support the company that developed the IDE we utilize and the hardware platform that we are going to be using.

2. Once you have your Arduino, you can then proceed to download the Arduino environment. The Arduino environment is the software package where you will end up spending most of your time when dealing with the Arduino.

You can obtain this software by going to the official download page on Arduino.cc, or you can go to it directly by going to the following link, https://www.arduino.cc/en/Main/Software.

Make sure you download the correct version for your operating system. Fortunately, the Arduino IDE supports Windows, Mac, and Linux operating systems.

Once downloaded, proceed to install it on your computer, if you are installing on Windows make sure you keep the box ticked for installing the USB drivers. These USB drivers will allow you to push programs to your Arduino over its USB connection, without the drivers the computer will fail to detect the correct interface.

3. Now connect your Arduino to your computer, if your using the Arduino UNO like we do this is as simple as plugging in a USB cable from the Arduino to the computer. If you are utilizing an older Arduino that doesn't feature USB support your may need to look into setting up a serial connection.

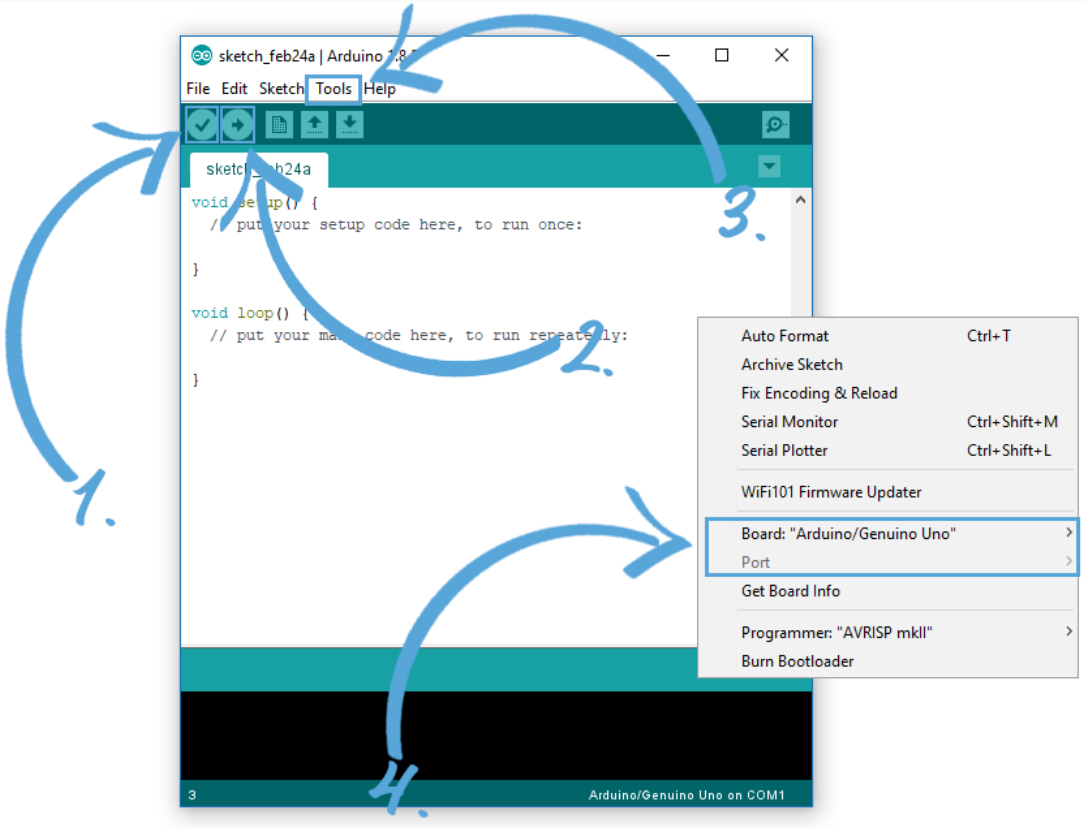
If a succesful connection has been made you should see the Arduino's power LED switch on and if you are on Windows you will likely see the "Found New Hardware Wizard" running in the background.

Getting Started with the Arduino

4. With your Arduino now connected to your computer lets go ahead and launch up the Arduino IDE.

The Arduino IDE is where you will be doing most of your work for programming the Arduino and is the best tool to push code to the device as it is actively maintained by Arduino themselves and has built-in support for all of their boards.

Below we run through a few things you should keep an eye out for within the Arduino environment, and what you will likely end up commonly using.



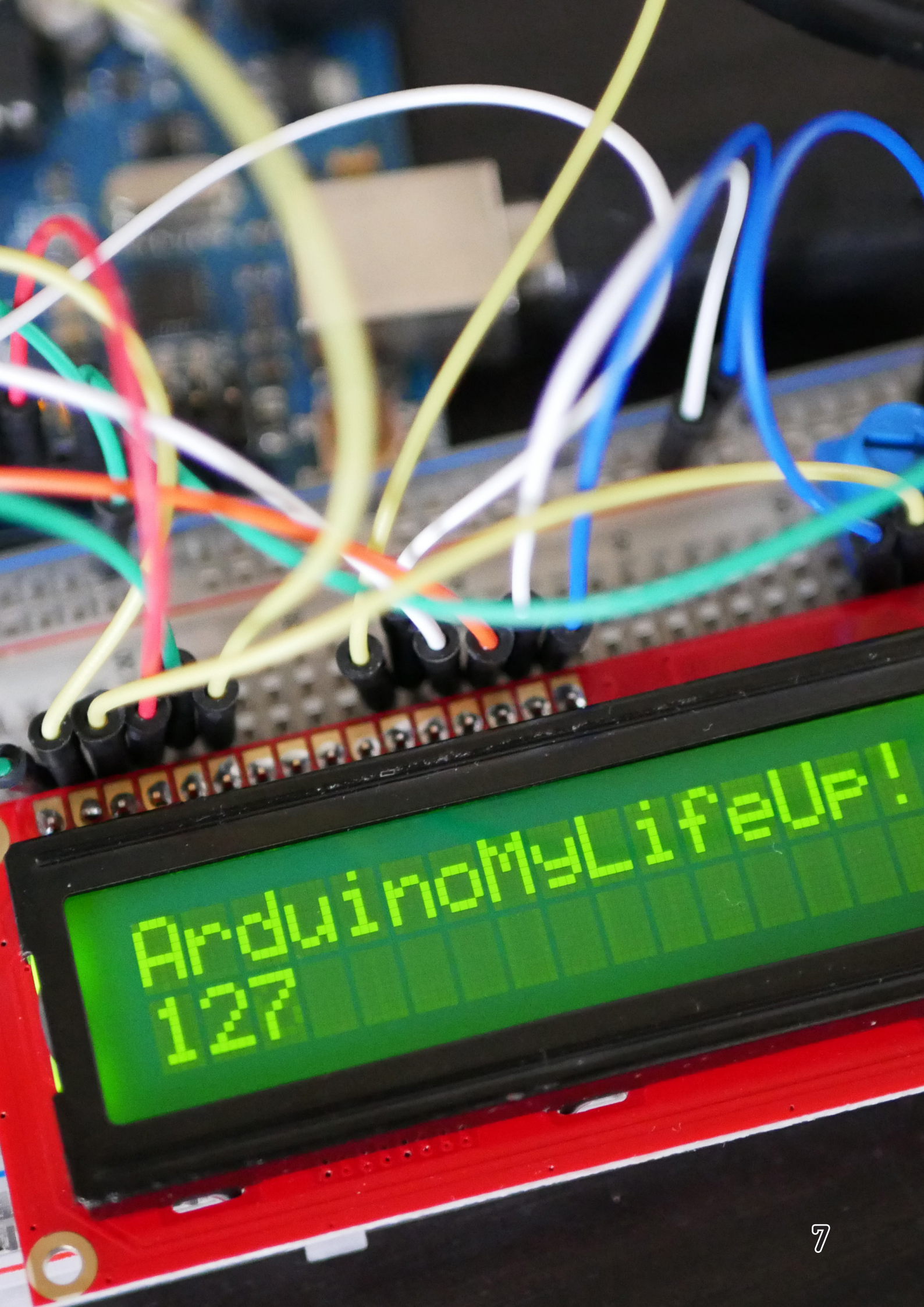
The button with the tick on it **(1.)** is the button you will utilize to verify your code is correct. This button will compile your code and verify that it will actually in fact run. However, this will of course not find any bugs in your code but will give you a good idea if what you wrote will even be able to be pushed to the Arduino.

The button directly next to it **(2.)** is the button you will press when you finally want to push your code to your Arduino. Pressing this button will begin the process of verifying and compiling your code then delivering it to the specified port to your Arduino.

In some cases this may fail if the Arduino IDE was unable to find the correct port to connect to your Arduino, if this occurs, you should see the error "**Problem Uploading to Board.**" Don't worry the next two points will show you where you can change the board or the port.

In the top menu, you can select "**Tools**" **(3.)**, this will bring up a sub-menu where you can find a variety of different tools.The two were are paying attention in this section are highlighted in the image above **(4.)**.

Here you can set the board that you are currently using, and also set the port that you want to utilize to connect to your Arduino. If your code doesn't push correctly and you see the error "**Problem Uploading to Board,**" then try changing the port till you find the correct one.


ArduinoMyLifeUP!
127

Arduino
Serial Monitor

Project Description

In this tutorial, we will be going through on how to setup the Arduino IDE's serial monitor so you can debug and interact with a program running on the Arduino.

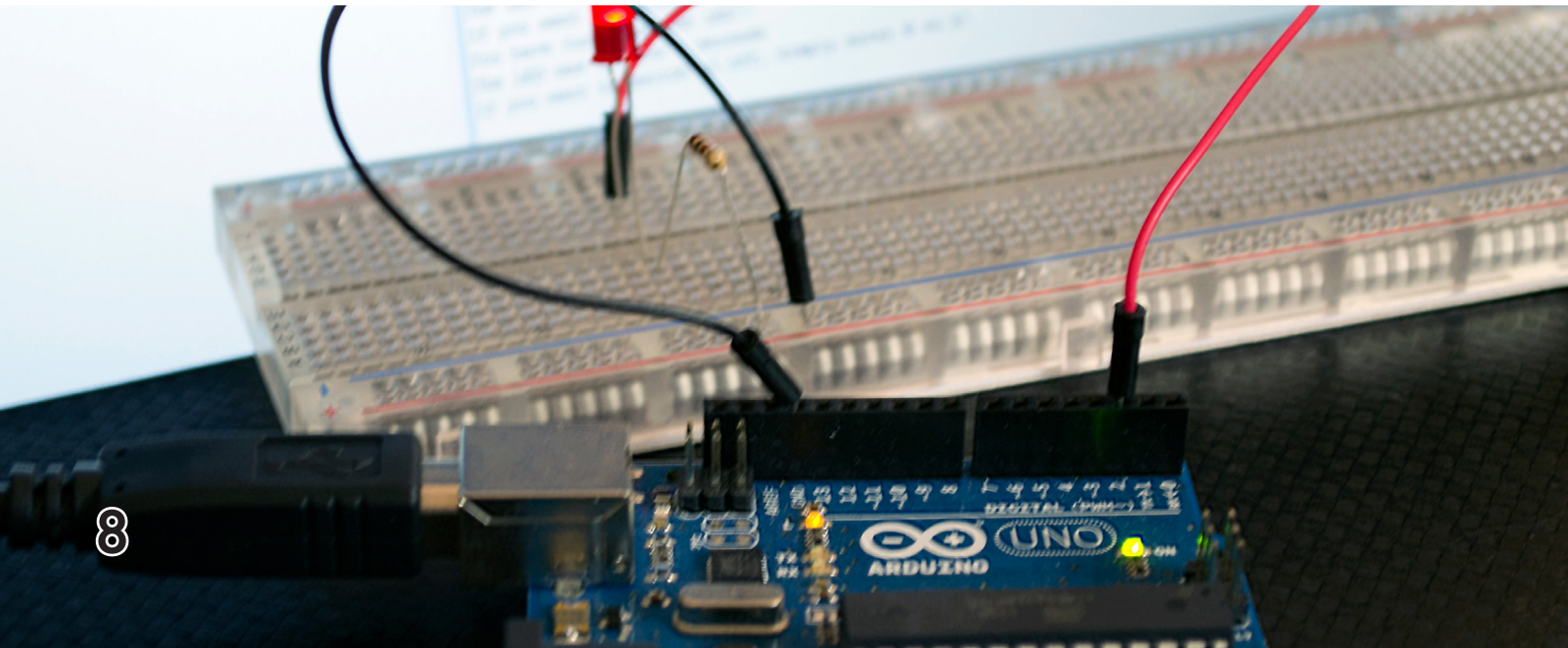
Setting up an Arduino serial monitor is a pretty simple process, but it can be a bit confusing when first starting out. This process is especially complicated if you are relatively new to programming and new to the Arduino platform.

As part of this tutorial, we will be building a straightforward circuit to demonstrate how you can both receive and send commands over the Arduino's serial monitor.

Equipment

Required

- Arduino Uno
- 1 x 100-ohm resistor
- 1 x Red LED
- Breadboard
- Breadboard Wire



Building our simple circuit

The circuit that we will be building for this is super simple and will consist of a single LED and a resistor. We will be able to control this LED pair by utilizing the serial monitor from the Arduino IDE.

This circuit is purely just there to act as an example and show both sending and receiving data through the serial monitor.

To setup, this circuit simply follow the next couple of steps, its simple as connecting the led and resistor to certain pins on the Arduino.

If you're having trouble, please refer to the circuit diagram below to see how it should be wired up to the Arduino UNO.

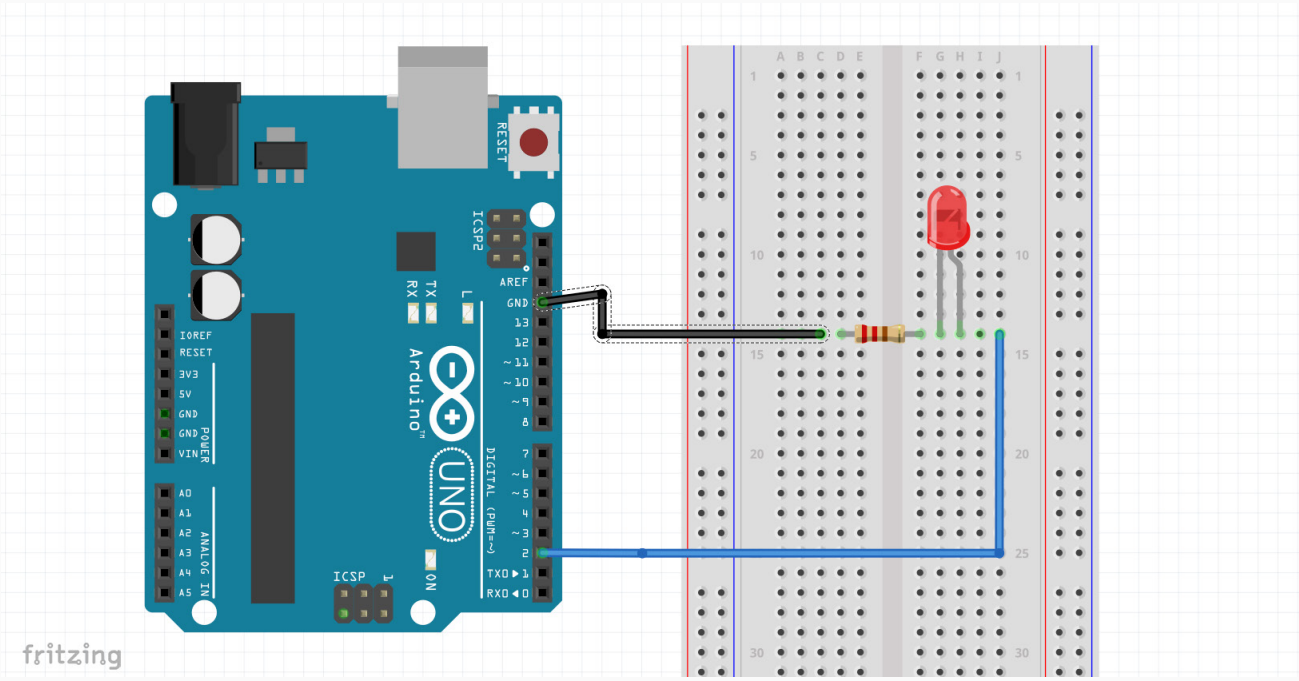
1. To begin with simply connect the **postive lead** of the **red LED** to the **pin 2** of your Arduino, this should be marked on the board with just **2** next to it.

Make sure you utilize the pin in the "**DIGITAL**" section of the pins and not the "**ANALOG IN**" pins.

2. Next connect a **100-ohm resistor** from the **negative lead** of the **red LED** to the **ground rail** on the Arduino, on the Arduino UNO this pin is marked with the text "**GND**" next to it.

3. You should now have the simple circuit all setup, if you are unsure if you have setup the circuit correctly then view our diagram that we have displayed below. This should help give an idea of how everything is meant to be wired.

Once you are sure you have this circuit setup correctly, proceed onto the next page to begin writing the sample code.



The Serial Monitor Code

The code that we are utilizing is relatively simple. The codes primary purpose is to show the basics of how you can use the Arduino Serial monitor to view and send data. I will explain each of the lines that we are using and how you can use it in your next project to debug or interact with the Arduino.

1. To begin with, simply open up the Arduino environment and begin writing a new file, in this, we will be entering in all the code we plan on pushing to the Arduino. If you already have code there, save it by pressing **Ctrl + S** and then create a new file by pressing **Ctrl + N**.

2. With the Arduino sketch file now ready, let's begin writing the following lines of code, we recommend you read through the explanations, so you have a proper understanding of how everything works.

To begin we initialize an integer variable for our LED, this is the pin number leading to the LED. We also initialize a variable called count; this will store the amount of time the LED is on or off.

```
int redLedPin = 2;
int count = 0;
```

In the setup function, we first set the red LED to act as an output so we can turn it off and on.

We then call the **serial.begin** function with a parameter value of **9600**. The value we're setting is known as the **bits per second (baud)** rate. The **baud rate** can be set to quite a large range of values, but we will use **9600** as this seems to be the default and will work with most devices without any issues. It is essential to call this function. If you fail to call this functiuon you won't be able to communicate with the Arduino.

Next, we wait for the serial interface to initialize and connect to the computer/device. We then call a function called **Serial.println()**. This function allows us to print a line to the Arduino serial monitor. You will notice this will print the text and then start on a new line for the next input/output.

```
void setup() {
  pinMode(redLedPin, OUTPUT);
  Serial.begin(9600);
  while (! Serial);
  Serial.println("Enter Y to turn on the LED:");
}
```

Lastly, we have the loop. This loop function will continually loop through until either a new program is uploaded or the Arduino is switched off.

Inside the loop function, we have an if statement checking to see if there is data waiting in the serial buffer**(Serial.Available).**

In simple terms, we are checking to see if we have sent a command to it. If there is data, we enter into that if statement.

```
void loop(){
  if (Serial.available()){
```

Next, we create a variable called **ch** and call **Serial.read()** which will get us the data currently waiting within the serial buffer.

Keep in mind this function will only get us the first byte of the data that is incoming to the Arduino. If you want to get an entire string, you will need to use something like **Serial.readString**.

```
    char ch = Serial.read();
```

The Serial Monitor Code

If the byte of data is either **y** or **n**, we then enter the relevant if statement. In here you will notice we change the **output** of the **LED pin high** or **low**.

We then print out a series of statements. There are two things you should notice here. **Serial.println** will print the data and go to a new line while **Serial.print** will print the data but stay on the same line when outputting.

You will also notice you can print variables such as the count example in the code below. Doing this will allow you to be able to debug values when it comes to data processing.

```
        if (ch == 'y' || ch == 'Y'){
          digitalWrite(redLedPin, HIGH);
          Serial.println("You have turned on the LED!!");
          Serial.print("The LED was off for ");
          Serial.print(count);
          Serial.println(" seconds");
          Serial.println("If you want to switch it off, simply enter N or n!");
          count = 0;
        }
        if (ch == 'n' ||      ch == 'N'){
          digitalWrite(redLedPin, LOW);
          Serial.println("You have turned off the LED!!");
          Serial.print("The LED was on for ");
          Serial.print(count);
          Serial.println(" seconds");
          Serial.println("If you want to switch it on, simply enter Y or y!");
          count = 0;
        }
      }
```

Lastly, we delay the script for a second **(1000 ms)** and add **1** to the count **variable**. This is just so we can show youan example of how to add data to a variable then print it via the serial output.

```
      delay(1000);
      count += 1;
    }
```

3. Now that we have gone through all the code you can check over the full code block on the next page. Check to see if you have made any mistakes while typing in your code.

Since the code block is to large to fit onto this page we have included on the next page over.

The Serial Monitor Code

If the byte of data is either **y** or **n**, we then enter the relevant if statement. In here you will notice we change the **output** of the **LED pin high** or **low**.

We then print out a series of statements. There are two things you should notice here. **Serial.println** will print the data and go to a new line while **Serial.print** will print the data but stay on the same line when outputting.

You will also notice you can print variables such as the count example in the code below. Doing this will allow you to be able to debug values when it comes to data processing.

```
int redLedPin = 2;
int count = 0;

void setup() {
   pinMode(redLedPin, OUTPUT);
   Serial.begin(9600);
   while (! Serial);
   Serial.println("Enter Y to turn on the LED:");
}

void loop(){
   if (Serial.available()){
      if (ch == 'y' || ch == 'Y') {
         digitalWrite(redLedPin, HIGH);
         Serial.println("You have turned on the LED!!");
         Serial.print("The LED was off for ");
         Serial.print(count);
         Serial.println(" seconds");
         Serial.println("If you want to switch it off, simply enter N or n!");
         count = 0;
      }
      if (ch == 'n' ||      ch == 'N') {
         digitalWrite(redLedPin, LOW);
         Serial.println("You have turned off the LED!!");
         Serial.print("The LED was on for ");
         Serial.print(count);
         Serial.println(" seconds");
         Serial.println("If you want to switch it on, simply enter Y or y!");
         count = 0;
      }
   }
  delay(1000);
   count += 1;
}
```

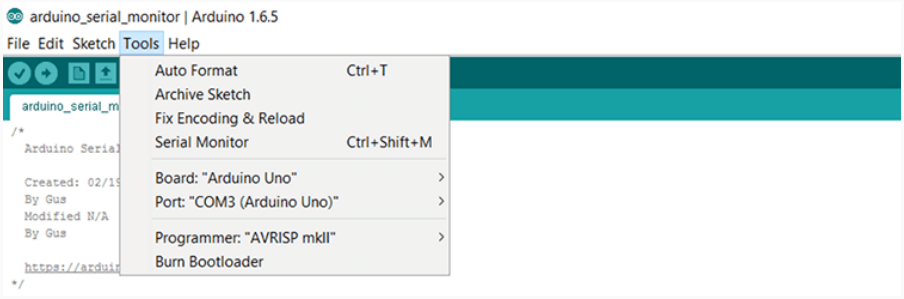
4. Once you are certain you have typed in all the code correctly you can go ahead and push the data to your Arduino by pressing the "**Upload**" button within the Arduino UNO. This will verify, compile and upload the code to your Arduino.

We can now proceed onto actually connecting, recieving and sending data to the Adruino.

Connecting, Receiving and Sending Data to the Arduino

Now that the code has been uploaded to the Arduino we will need to open up the Arduino serial monitor. This is pretty easy, and there are only a few options that we will need to explain.

To open up the serial monitor click "**Tools**" in the top menu and then select "**Serial Monitor**". Alternatively you can press, **CTRL+SHIFT+M**, this will bring up the same window. Make sure that you have hooked your computer hooked up to the Arduino otherwise the window won't open up.



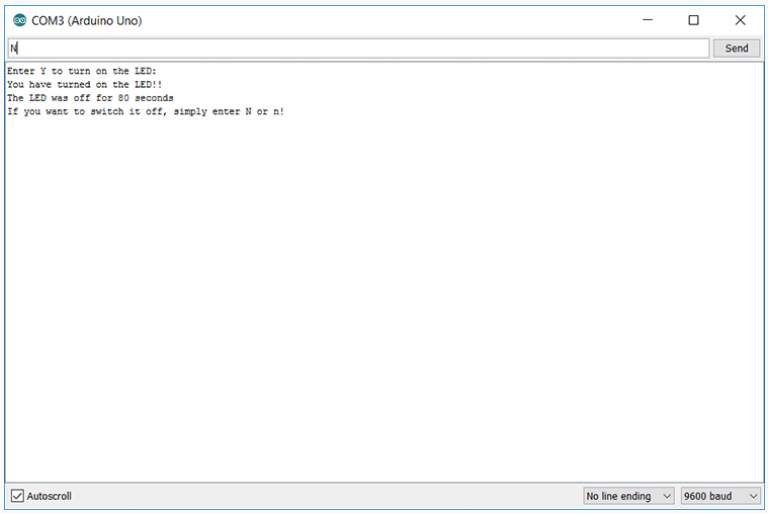
You should now have a window open that looks similar to the one below.
At the top of this screen, there should be an input area. This input area is where you can type and send text data to the Arduino. To transmit the text to the Arduino that you have typed just press the send button.

Below the textbox is the output window, all the data that is sent to us from the Arduino will be displayed here. If you're using the example code that we wrote earlier, you will be able to see the text that is sent through the serial by utilizing the **Serial.println** functions.

On the bottom toolbar, we have three different options. The first is pretty self-explanatory, the auto-scroll on the output window will be disabled and enabled by checking and unchecking the Autoscroll checkbox.

Second, we have the line ending option. You can set the monitor to automatically append a line ending after what you enter/send to the Arduino.

Thirdly we have the baud rate. Make sure this matches to what you have set in the code. If it is different to what we set within the code than the output text will appear as gibberish. If you're receiving nonsense, then this is a likely cause of this.



This is basically everything you need to know to understand the basics of the serial monitor. There are a few more things you can learn about such as using strings via the serial interface and more but this probably all you need to know for now.

Arduino
Traffic Light Circuit

Project Description

In this tutorial, we will be showing you how to set up a very simple traffic light circuit utilizing your Arduino as the brain of it all.

This project will utilize a little bit of simple code that we will thoroughly explain so you can understand everything that is going on.

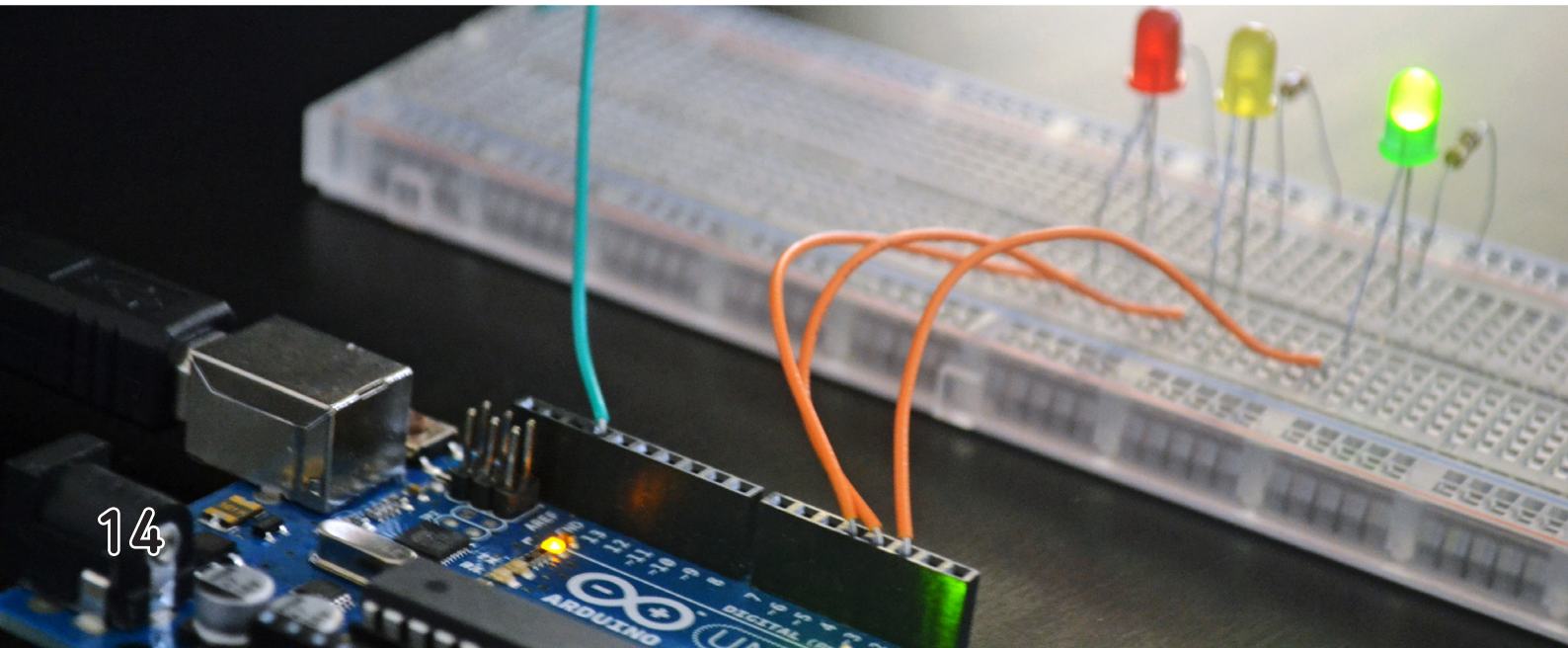
Additionally, you will be setting up a straightforward circuit with the LED's all connected to various pins on the Arduino.

This project is a fantastic way to start out for those who are just getting started with the Arduino as it introduces some fundamental elements.

Equipment

Required

- Arduino Uno
- 3 x 100-ohm resistor
- 1x Red LED
- 1x Yellow LED
- 1x Yellow LED
- Breadboard
- Breadboard Wire

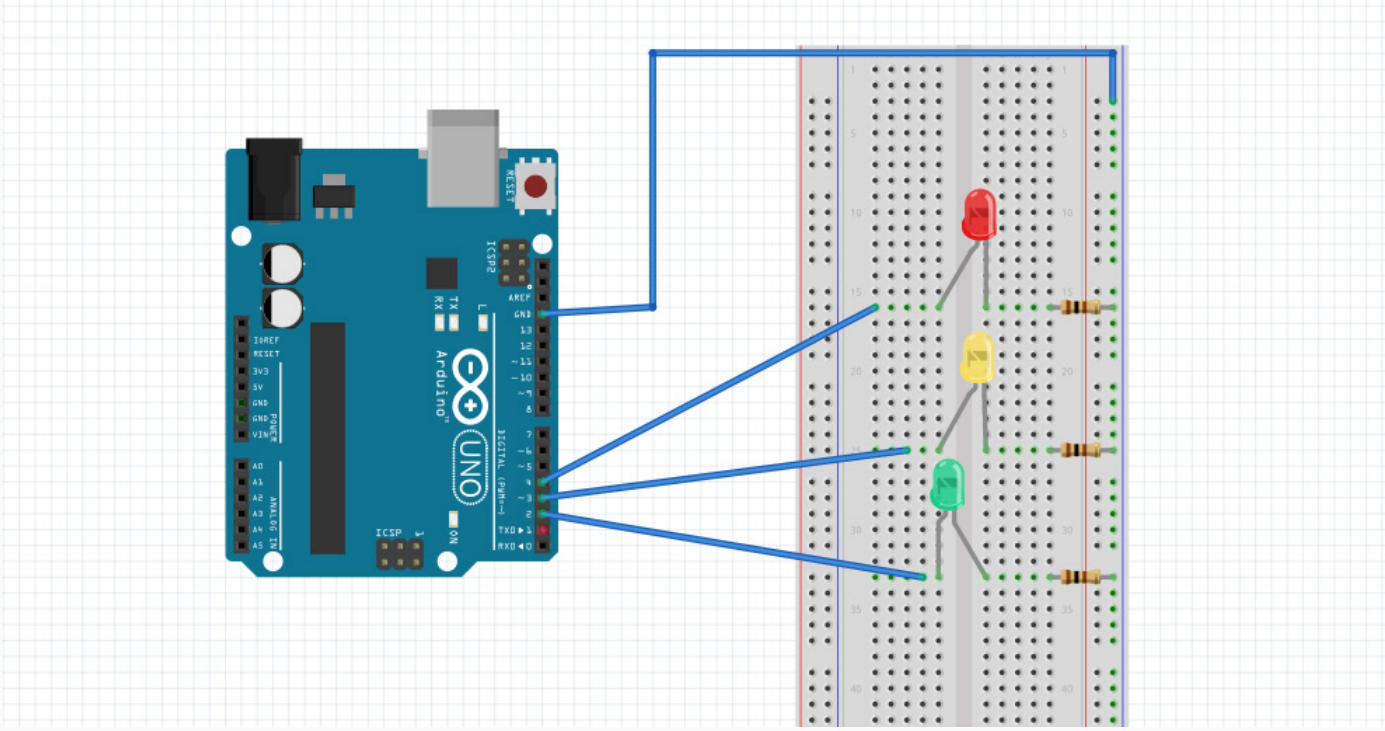


Building our traffic light circuit

The circuit that we will be building in this tutorial is very simple to setup. It shouldn't take you very long to get up and running correctly.

Follow the following steps to learn how to setup this circuit, alternatively just follow the diagram that we have included below.

1. Connect the **postive lead** of the **red LED** to the "**Digital**" **pin 4** of your Arduino.
2. Connect the **postive lead** of the **yellow LED** to the "**Digital**" **pin 3** of your Arduino.
3. Connect the **postive lead** of the **green LED** to the "**Digital**" **pin 2** of your Arduino.
4. Next we need to connect a **100-ohm resistor** from the **negative lead** of each **LED** to the negative rail on your breadboard. You should have one resistor per LED.
5. Now finally you need to connect a wire from the negative rail on your breadboard to the GND pin on the Arduino, this is marked by the GND text on the Arduino board.



Writing our traffic light circuit code

Now we will write some code to bring our traffic lights to life. If you have programming experience, then you will find this code incredibly basic. If you are new to programming then don't worry as this is a great way to start learning.

1. To start off, open up the Arduino environment and begin writing a new scratch file. If you already have code that you are working with, save it by pressing **Ctrl + S** and then create a new file by pressing **Ctrl + N**.

2. Now that we are working with a new sketch file, let's write the following lines of code to the file. We recommend you read through the explanations, so you have a proper understanding of how everything works.

First, we will need to declare our variables. We create an integer(numerical value) variable for each LED, for instance, we use the **GREEN** variable to represent the pin that the green LED is currently connected to.

We utilize the **DELAY_** integer variables are the amount of time in milliseconds that we will delay the program from moving forward.

```
// variables
int GREEN = 2;
int YELLOW = 3;
int RED = 4;
int DELAY_GREEN = 5000;
int DELAY_YELLOW = 2000;
int DELAY_RED = 5000;
```

Now we need to set up the pins so that they act as an output and not as an input. We do this by utilizing the **Pinmode(pin,mode)** function.

This pinMode function accepts two parameters. The first being the pin number and the second being the mode we want to set. The **pinMode** mode can either be set to **OUTPUT** or **INPUT**.

```
// basic functions
void setup()
{
  pinMode(GREEN, OUTPUT);
  pinMode(YELLOW, OUTPUT);
  pinMode(RED, OUTPUT);
}
```

This next function creates a loop that the Arduino will automatically run through. Within this function, we include our main logic.

In our case, this will just call a couple of functions that we will implement a bit further down the track. It will also run the "**delay(milliseconds)**" function to pause the script temporarily.

```
void loop()
{
  green_light();
  delay(DELAY_GREEN);
  yellow_light();
  delay(DELAY_YELLOW);
  red_light();
  delay(DELAY_RED);
}
```

Writing our traffic light circuit code

Finally we need to write a function for each of the lights, there is cleaner ways to handle this but for simplistic sake we are going to keep it simple. Each function has a very simple purpose, and that is simply to write values to our pins by setting them either **HIGH** or **LOW**.

To be able to write to the pins we make use of the **digitalWrite(Pin Number, Value)** function. This function will send data over the specified pin. For dealing with digital pins this data value should just be **HIGH** or **LOW**.

Each function will turn its respective LED on while turning the other two LED's off by sending **LOW** to them and the **HIGH** to the LED we want to turn on.

```
void green_light()
{
   digitalWrite(GREEN, HIGH);
   digitalWrite(YELLOW, LOW);
   digitalWrite(RED, LOW);
}

void yellow_light()
{
   digitalWrite(GREEN, LOW);
   digitalWrite(YELLOW, HIGH);
   digitalWrite(RED, LOW);
}

void red_light()
{
   digitalWrite(GREEN, LOW);
   digitalWrite(YELLOW, LOW);
   digitalWrite(RED, HIGH);
}
```

3. Now that we have finished writing in all the code we recommend you check it to ensure you have typed everything in correctly.

You can see the full final version of the code on the page as it is too large to fit onto this page. If you are sure your code is correct you can move onto the next step.

4. Once you have finished entering all the code, you can verify it by pressing the verify button; this is the button next to the upload button and the one with a tick on it.

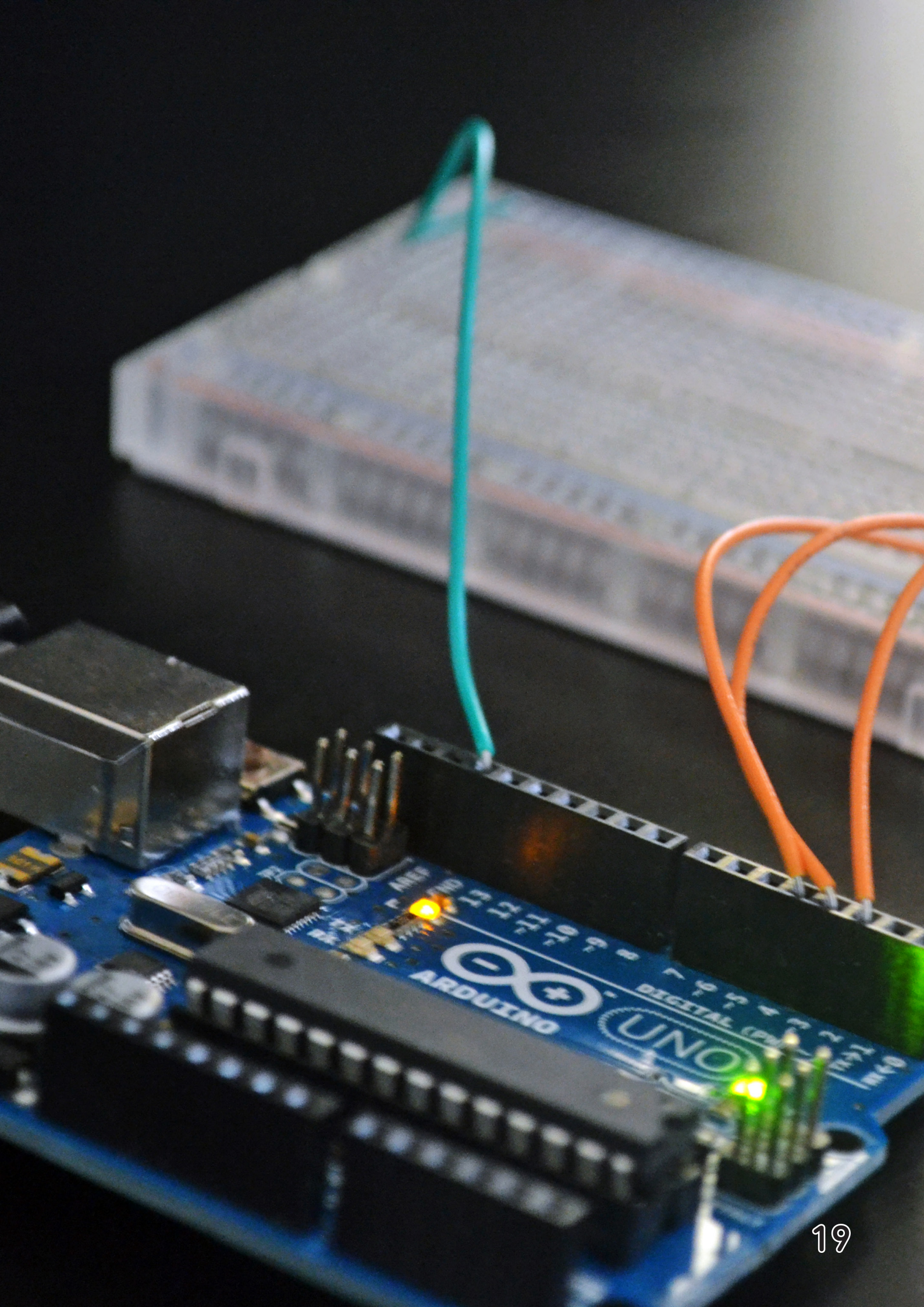
If your code successfully verifies you can go ahead and press the upload button. This will push the code to your Arduino.

If everything is working correctly you should see your lights start to cycle in a pattern, with the green light starting first, then the yellow, then the red light. If everything is working, then congratulations you have setup your first simple circuit and pushed some simple code.

You can now move onto some of the more complicated tutorials featured in this book. If for some reason nothing occurs, make sure you have the circuit connected correctly and have entered all the code correctly.

Writing our traffic light circuit code

```
// variables
int GREEN = 2;
int YELLOW = 3;
int RED = 4;
int DELAY_GREEN = 5000;
int DELAY_YELLOW = 2000;
int DELAY_RED = 5000;

// basic functions
void setup()
{
   pinMode(GREEN, OUTPUT);
   pinMode(YELLOW, OUTPUT);
   pinMode(RED, OUTPUT);
}

void loop()
{
   green_light();
   delay(DELAY_GREEN);
   yellow_light();
   delay(DELAY_YELLOW);
   red_light();
   delay(DELAY_RED);
}

void green_light()
{
   digitalWrite(GREEN, HIGH);
   digitalWrite(YELLOW, LOW);
   digitalWrite(RED, LOW);
}

void yellow_light()
{
   digitalWrite(GREEN, LOW);
   digitalWrite(YELLOW, HIGH);
   digitalWrite(RED, LOW);
}

void red_light()
{
   digitalWrite(GREEN, LOW);
   digitalWrite(YELLOW, LOW);
   digitalWrite(RED, HIGH);
}
```


ARDUINO
UNO

Arduino Dice
A Simple Dice Circuit

Project Description

In this tutorial, we are going to be expanding our earlier Arduino traffic lights project. We are going to be doing this by setting up our very own Arduino dice circuit. This is a simple project that will introduce you to some new elements.

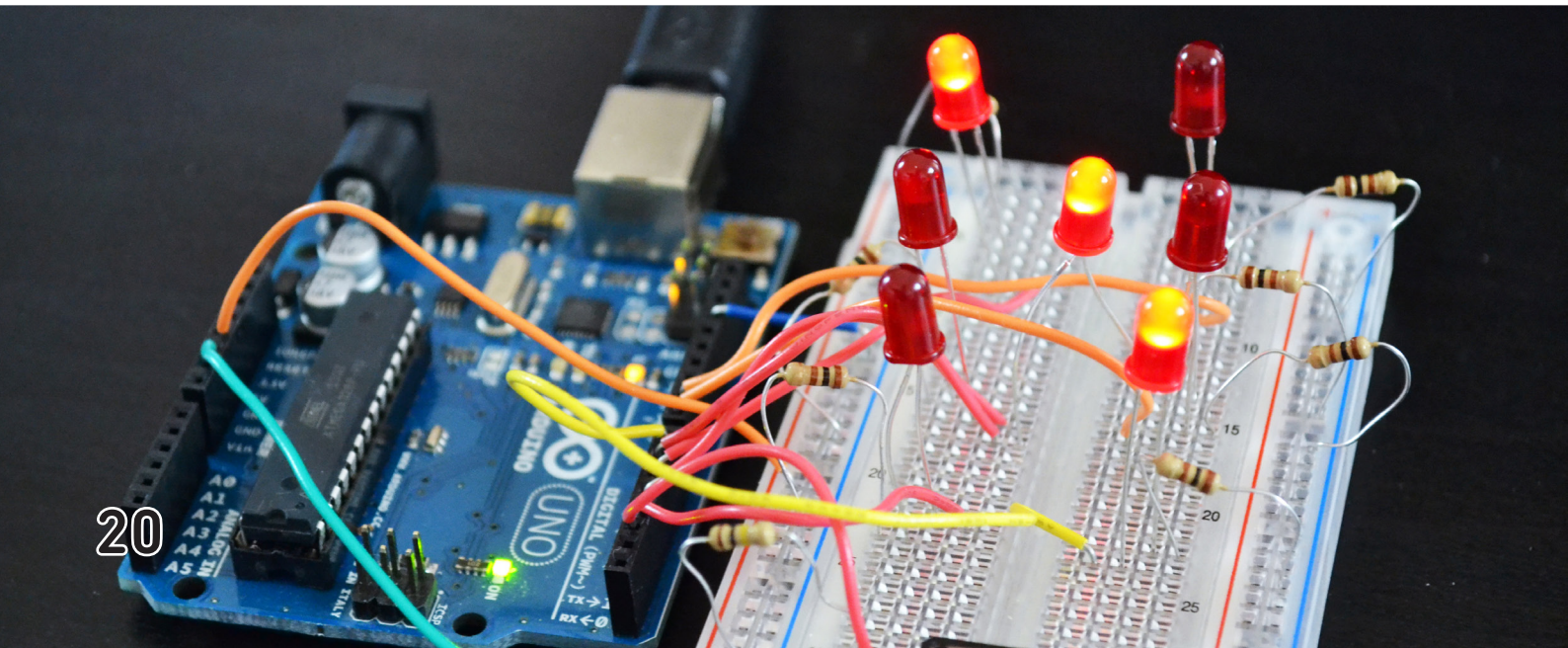
In case of this project, we will be showing you how you can utilize a push button which will run a random number generator on the Arduino itself to pick a number between 1 and 6. Based on the result it will then output to each of the LED's, high or low.

This circuit is much like the Arduino traffic lights project this project uses a variety of LEDs and resistors to make up the circuit.

Equipment

Required

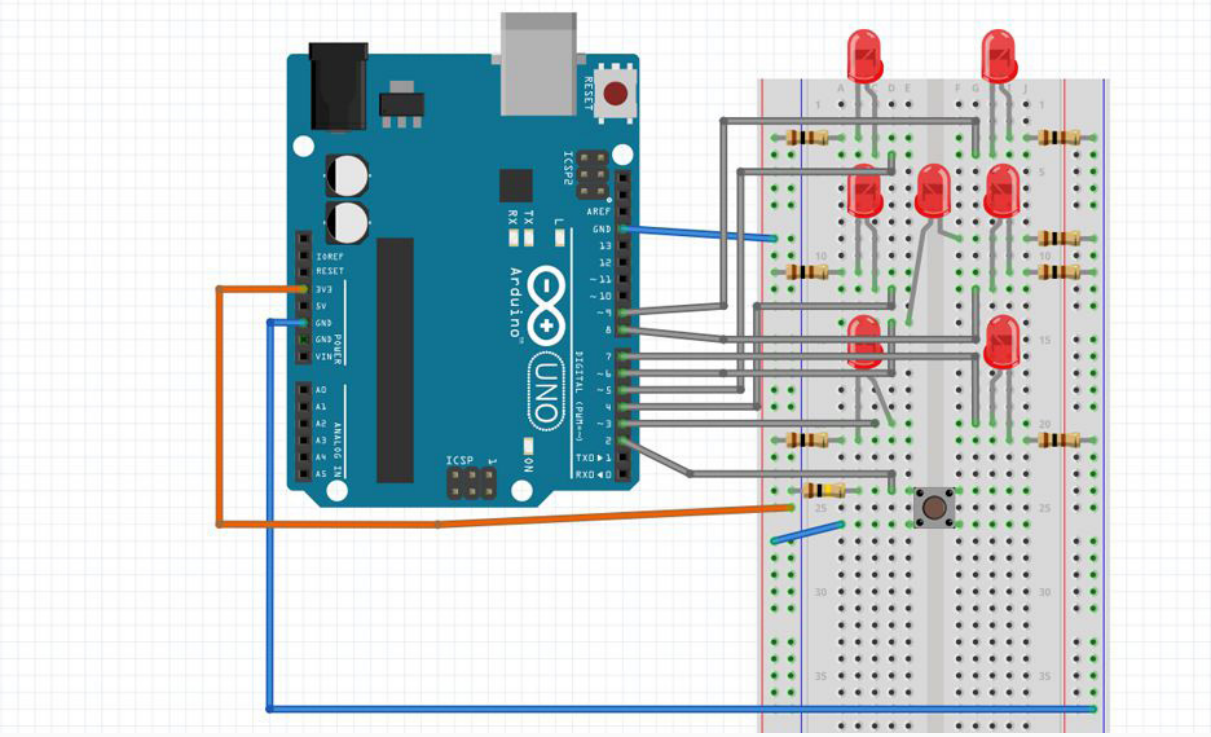
- Arduino Uno
- 7 x LED's
- 7 x 100 Ohm Resistors (Brown Black Brown)
- 7 x 100k Ohm Resistors (Brown Black Yellow)
- Physical switch/push button
- Breadboard
- Breadboard wire



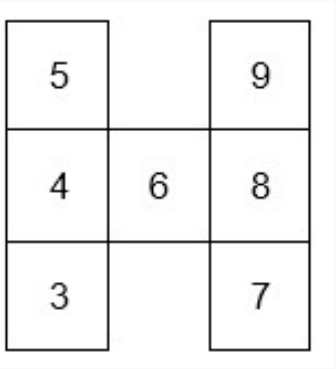
Building the Arduino Dice Circuit

The circuit that we are going to be building looks a lot more complicated then it really is, basically we will be setting up our LED's in a H pattern with each LED being connected back to the Arduino board .

You can either put the circuit together based on the diagram below or follow our instructions.



Each of the LEDs needs to be positioned to resemble a die face. We have spaced these out a bit more, so it's easier to see on the diagram circuit above. You can place yours much closer together if you would like it to look much more like a proper die.



With the LED's positioned in an **H pattern** use our diagram to the right of this text to connect each positive lead on the LED's back to their respective pin.

For example, the top left LED should be connected to the **"Digital" pin 5** on the Arduino board.

Once you have connected all the positive pins of the LED's to their correct pin on the Arduino you can move onto connecting up the negative side.

To every **negative pin** of the LED's **add** one of the **100-ohm resistors**, with each of these resistors connected to the **negative rails** of your breadboard. Once you have all the resistors plugged in proceed to use a bit of breadboard wire to connect the **negative rail** on your breadboard to the **GND pin** on your Arduino.
Now for the final bit of the circuit we need to place our push button onto the breadboard. Run a wire from the **3v3 connection** on your Arduino to the **postive rail** on your **breadboard**.

From the **postive rail** add a **100k resitor** between it and one of the pins of the push button as we display in our diagram above.Now from that same pin we need to add a breadboard wire to connect from that to **pin 2** on the Arduino.

Finally connect the other pin on the psuh button to the **negative rail** of the breadboard. What this will do is that when the push button is pressed it will complete the circuit and trigger the input on pin 2.

Writing our dice circuit code

Now lets bring our dice to life with a bit of code, this code will assume you used the same pins on the Arduino as we did in our own circuit.

1. We need to startup a new file by launching the Arduino Enviornment. If you have a previous project within the Arduino Enviornment, save it by pressing **Ctrl + S** and then create a new file by pressing **Ctrl + N**.

2. Once we have a new sketch file we need to write in the following lines of code, this is quite a long project. Though it is simple we do reccomend you read through all our comments to ensure that you have a proper understanding of what is happening.

In the example below we declare all the global variables that we intend on utilizing. **BottomLeft** refers to the location of the LED this makes it easier to identify when editing the script. The value that we are setting these variables to are the pins that the they are connected to.

The **state** and **randNumber** variables will be used for storing the random numbers we generate for the dice. You will see us make use of these shortly.

```
int button = 2;
//LED for DICE
int bottomLeft = 3;
int middleLeft = 4;
int upperLeft = 5;
int middle = 6;
int bottomRight = 7;
int middleRight = 8;
int upperRight = 9;

int state = 0;
long randNumber;
```

In the next lot of code we create our **setup()** function. Within this function is where we assign pins to be either an **OUTPUT** or an **INPUT**, to do this we utilize the **pinMode(pin, mode)** function. We use **randomSeed()** to give our number generator a sense of randomness.

We set all of the pins but the button to be output, we also utilize this function to also set the serial **BAUD** rate.

```
//Initial setup
void setup(){
  pinMode(bottomLeft, OUTPUT);
  pinMode(middleLeft, OUTPUT);
  pinMode(upperLeft, OUTPUT);
  pinMode(middle, OUTPUT);
  pinMode(bottomRight, OUTPUT);
  pinMode(middleRight, OUTPUT);
  pinMode(upperRight, OUTPUT);

  pinMode(button, INPUT);
  Serial.begin(9600);
  randomSeed(analogRead(0));
}
```

Writing our dice circuit code

The loop part of our script continues to loop through until the Arduino is either turned off or a new set of code is uploaded to the Arduino. At the start of our loop we check to see if the switch has been pressed by seeing if it has a **HIGH** input.

If the conditions are met, then we change the state to 1 and get a random number between 1 and 7. Once we have the random number we check to see what it is and run the relevant function. Once the function has completed we wait for 4 seconds and then turn the LEDS off and return to our normal state of 0 for the next loop.

```
void loop(){
  //Read our button if high then run dice
  if (digitalRead(button) == HIGH && state == 0){
    state = 1;
    randNumber = random(1, 7);
    delay(100);
    Serial.println(randNumber);

    if (randNumber == 6){
      six();
    }

    if (randNumber == 5){
      five();
    }

    if (randNumber == 4){
      four();
    }

    if (randNumber == 3){
      three();
    }

    if (randNumber == 2){
      two();
    }

    if (randNumber == 1){
      one();
    }

    delay(4000);
    clearAll();
    state = 0;
  }
}
```

Writing our dice circuit code

Now we need to write the functions that we utilize within out loop. We are writing a function for each possible dice roll, which means in our case we have six possible functions.

Each function will merely turn a certain amount of the LED's on by sending a "**HIGH**" output. These LED's will represent the current dice roll. For instance, a roll of six would have 3 LED's on each side switched on, with the middle LED being turned off.

```
void six()
{
   digitalWrite(bottomLeft, HIGH);
   digitalWrite(middleLeft, HIGH);
   digitalWrite(upperLeft, HIGH);
   digitalWrite(bottomRight, HIGH);
   digitalWrite(middleRight, HIGH);
   digitalWrite(upperRight, HIGH);
}

void five()
{
   digitalWrite(upperLeft, HIGH);
   digitalWrite(bottomLeft, HIGH);
   digitalWrite(middle, HIGH);
   digitalWrite(upperRight, HIGH);
   digitalWrite(bottomRight, HIGH);
}

void four()
{
   digitalWrite(upperLeft, HIGH);
   digitalWrite(bottomLeft, HIGH);
   digitalWrite(upperRight, HIGH);
   digitalWrite(bottomRight, HIGH);
}

void three()
{
   digitalWrite(upperLeft, HIGH);
   digitalWrite(middle, HIGH);
   digitalWrite(bottomRight, HIGH);
}

void two()
{
   digitalWrite(bottomRight, HIGH);
   digitalWrite(upperLeft, HIGH);
}

void one(){
   digitalWrite(middle, HIGH);
}
```

Writing our dice circuit code

Now we have our final function, this function is very simple. All it does is utilize **digitalWrite** to set each of the pins back to low.

This ensures when the loop runs again that everything is switched off, it also helps keeps all of our functions shorter as we dont have to worry about turning any of the LED's off.

```
void clearAll(){
  digitalWrite(bottomLeft, LOW);
  digitalWrite(middleLeft, LOW);
  digitalWrite(upperLeft, LOW);
  digitalWrite(middle,LOW);
  digitalWrite(bottomRight, LOW);
  digitalWrite(middleRight, LOW);
  digitalWrite(upperRight, LOW);
}
```

3. With our code now completed we can proceed onto verifying and pushing the code to the Arduino. If you would like to check your code against the full version you can look over the next couple of pages, this will contain all the required code in one blob without explanations.

If you would prefer to just proceed with your current code, then go to the next step.

4. Now before we push our code to the Arduino, make sure that you have actually got your Arduino connected. With it connected, press the "**Verify**" button, this is the button with a tick on it.

This will compile and check your code to ensure it will actually run on the Arduino.

5. Finally if everything compiled fine we can press the "**Upload**" button, this will push the code to the Arduino. This is the button located next to the "**Verify**" button in the Arduino IDE.

6. If the code pushed fine, you should now be able to make use of your circuit! To test this push the push button down, this will cause the code to run and generate a random number. One of the six possible outcomes should occur, and your LED's should light up for six seconds

If everything works correctly, you should now have a working die set up with your Arduino.

The final dice circuit code

```
int button = 2;
//LED for DICE
int bottomLeft = 3;
int middleLeft = 4;
int upperLeft = 5;
int middle = 6;
int bottomRight = 7;
int middleRight = 8;
int upperRight = 9;

int state = 0;
long randNumber;

void setup(){
  pinMode(bottomLeft, OUTPUT);
  pinMode(middleLeft, OUTPUT);
  pinMode(upperLeft, OUTPUT);
  pinMode(middle, OUTPUT);
  pinMode(bottomRight, OUTPUT);
  pinMode(middleRight, OUTPUT);
  pinMode(upperRight, OUTPUT);

  pinMode(button, INPUT);
  Serial.begin(9600);
  randomSeed(analogRead(0));
}

void loop(){
  if (digitalRead(button) == HIGH && state == 0){
    state = 1;
    randNumber = random(1, 7);
    delay(100);
    Serial.println(randNumber);
    if (randNumber == 6){
      six();
    }
    if (randNumber == 5){
      five();
    }
    if (randNumber == 4){
      four();
    }

    if (randNumber == 3){
      three();
    }

    if (randNumber == 2){
      two();
    }
    if (randNumber == 1){
      one();
    }
    delay(4000);
    clearAll();
    state = 0;
  }
}
```

The final dice circuit code

```
void six()
{
   digitalWrite(bottomLeft, HIGH);
   digitalWrite(middleLeft, HIGH);
   digitalWrite(upperLeft, HIGH);
   digitalWrite(bottomRight, HIGH);
   digitalWrite(middleRight, HIGH);
   digitalWrite(upperRight, HIGH);
}

void five()
{
   digitalWrite(upperLeft, HIGH);
   digitalWrite(bottomLeft, HIGH);
   digitalWrite(middle, HIGH);
   digitalWrite(upperRight, HIGH);
   digitalWrite(bottomRight, HIGH);
}

void four()
{
   digitalWrite(upperLeft, HIGH);
   digitalWrite(bottomLeft, HIGH);
   digitalWrite(upperRight, HIGH);
   digitalWrite(bottomRight, HIGH);
}

void three()
{
   digitalWrite(upperLeft, HIGH);
   digitalWrite(middle, HIGH);
   digitalWrite(bottomRight, HIGH);
}

void two()
{
   digitalWrite(bottomRight, HIGH);
   digitalWrite(upperLeft, HIGH);
}

void one(){
   digitalWrite(middle, HIGH);
}

void clearAll(){
   digitalWrite(bottomLeft, LOW);
   digitalWrite(middleLeft, LOW);
   digitalWrite(upperLeft, LOW);
   digitalWrite(middle,LOW);
   digitalWrite(bottomRight, LOW);
   digitalWrite(middleRight, LOW);
   digitalWrite(upperRight, LOW);
}
```

Arduino
Light Sensor

Project Description

In this Arduino light sensor tutorial, we will be going through the basics of setting up a photoresistor and showing you how you can utilize it to detect changes in light easily. This little device can be convenient in a lot of projects where measuring the amount of light is essential.

This tutorial is incredibly simple but will hopefully explain and show how you can use a photoresistor in your next project. You won't need very much equipment with most of it being fundamental parts that you would probably already have if you bought an electronics starter kit.

This tutorial also shows why the Arduino's analog pins are so great in comparision to the Raspberry Pi's as the circuit is a lot simpler.

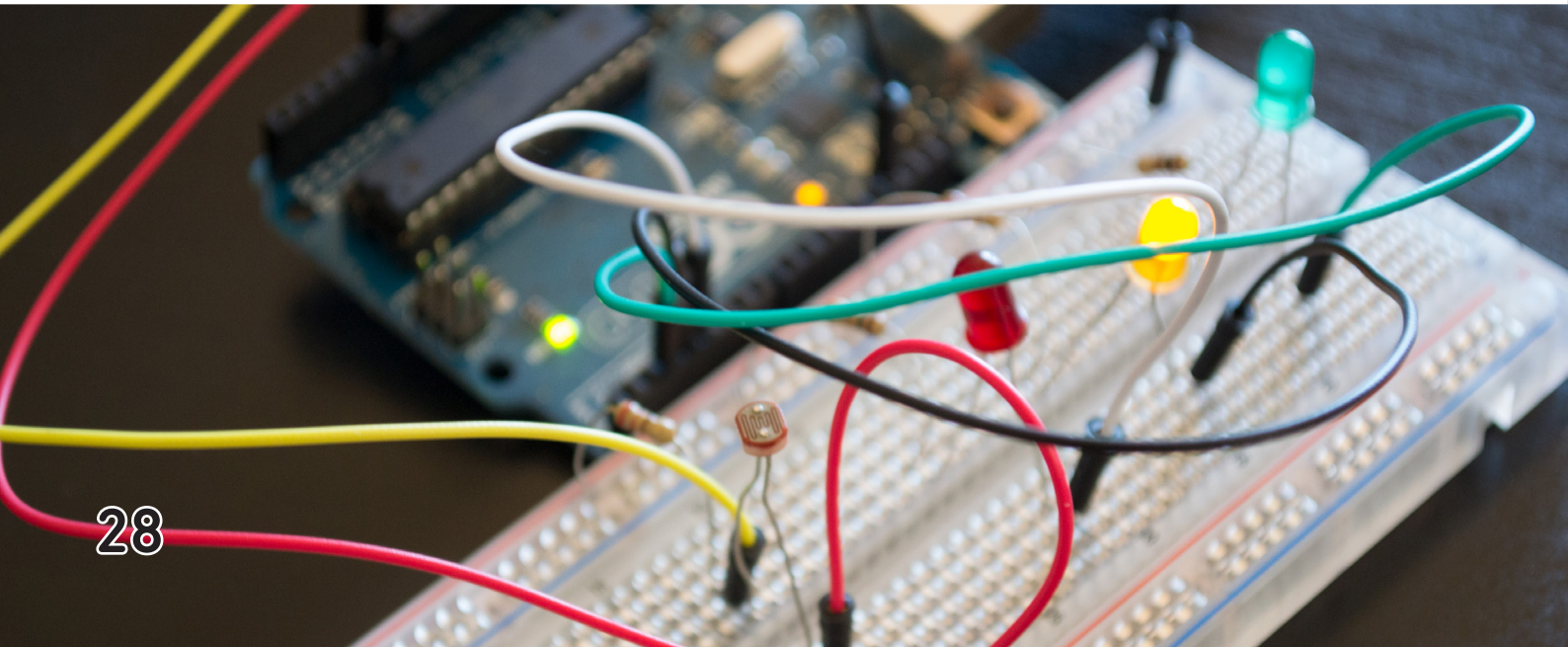
Equipment

Required

- Arduino Uno
- 1 x Photo Resistor
- 1 x 220 Ohm Resistor
- Breadboard
- Breadboard wire

Optional

- 3 x 100 Ohm Resistor
- 1 x Red LED
- 1 x Green LED
- 1 x Yellow LED



The Arduino Light Sensor Circuit

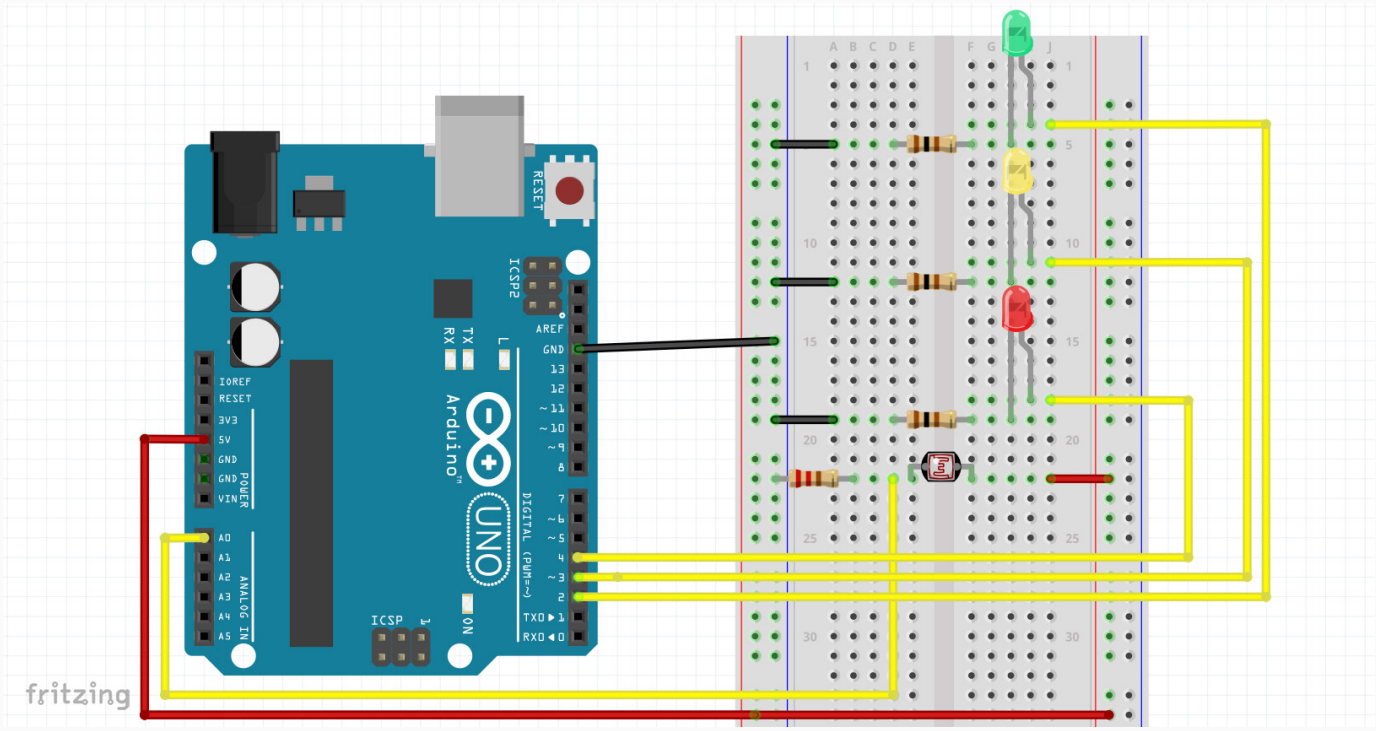
The circuit we need to build is pretty basic, and you shouldn't have too much trouble setting it up. I will briefly mention each of the parts that are in it and finally how to put it all together.

The light sensor or what is also known as a photoresistor is the piece of equipment we will be utilizing in our circuit to tell how light or dark it is. When it is dark, the resistor will have a very high resistance of up to 10 megohms. When it is light, it will probably have only a few hundred ohms of resistance.

You can often find out roughly the resistance by looking at the device datasheet. It is likely to refer to lux the unit of illuminance. It will give you an approximate resistance at a particular lux amount.

The LED's in our circuit will represent the current amount of resistance across the photoresistor. Green will be when it is at low resistance (Lots of light). Yellow will be when there is medium resistance across the LDR (Shady). Finally, red will represent for when it's at high resistance (In the dark).

1. First connect the **5v pin** from the Arduino to the **positive rail** on the breadboard.
2. Next connect the **ground(GND) pin** from the Adruino to the **negative rail** on your breadboard.
3. Next place the photoresistor on to the breadboard.
   - Connect one end of the **photoresistor** to the **positive rail**.
   - On the end other have the wire go back to the **A0** Analogue input on the **Arduino Board**.
   - Finally on the same end that you have the wire that goes back to the analogue input add a 220 ohm resistor the connects to the **negative rail** on the breadboard.
4. Now place 3 LED's on to the breadboard, these being one each of the Green, Yellow and Red LED's that we mention in the equipment list.
   - On each of these LED's add a **100-ohm resistor** to the **negative pin** of the LED and have it connect back to the negative rail on your breadboard.
   - Now finally place a wire that goes back to the Arduino from each of the LED's positive pins to a certain pin on the Arduino. For this tutorial the **Red LED** should go to **pin 4**, the **Yellow LED** to **pin 3** and the **Green LED** to **pin 2**.



Coding for our Arduino Light Sensor Circuit

1. Start off by launching the Arduino IDE. If there is a project opened already within the Arduino IDE, save it by pressing **Ctrl + S** and then create a new file by pressing **Ctrl + N**.

2. Within this new sketch file begin writing in the code below, we will explain each section of the code so you have an understanding on how everything works and how you can possibly change that for your own needs.

Before we start doing anything we first need to set up all our variables. For this program, we will need four variables to store our pin numbers and one variable to store the value of the analog pin in. All of our variables are integers as we only need to store numerical data.

```
int greenLedPin = 2;
int yellowLedPin = 3;
int redLedPin = 4;

int lightSensorPin = A0;
int analogValue = 0;
```

With this next section of code we set all the pins for the LEDS to act as outputs by utilizing the **pinMode()** function. You don't need to worry about setting up the analogue pin.

```
void setup() {
  pinMode(greenLedPin, OUTPUT);
  pinMode(yellowLedPin,OUTPUT);
  pinMode(redLedPin,OUTPUT);
}
```

The loop function is pretty simple and shouldn't be too hard to understand what's going on. We first get the value from the analog pin this is the photoresistor. Once we have the value, we compare and turn on the relevant LED. For example, the RED led will be when it's dark, yellow for shady and finally green for light. After this, we delay for 200ms turn all the LEDs to low and check again.

```
void loop(){
  analogValue = analogRead(lightSensorPin);
  if(analogValue < 50){
    digitalWrite(redLedPin, HIGH);
  }
  else if(analogValue >= 50 && analogValue <= 100){
    digitalWrite(yellowLedPin, HIGH);
  }
  else{
    digitalWrite(greenLedPin, HIGH);
  }
  delay(200);
  digitalWrite(greenLedPin, LOW);
  digitalWrite(yellowLedPin, LOW);
  digitalWrite(redLedPin, LOW);
}
```

3. Now that our code is completed we can now upload it to the Adruino, you can do this easily by pressing the "**Upload**" button with the Adruino IDE. This is the button next to the "**Verify**" button (Button with the tick) if you are unsure which one it is.

If you want to see what the code looks like when its fully written out look on the next page.

Coding for our Arduino Light Sensor Circuit

4. With the code pushed to your Arduino you should now start seeing your circuit react to different light levels. if you don't see any reactions you can try fiddling the comparision values that we use within our script.

If you want, you can even implement some serial monitoring to check what values that the photoresistor is returning.

The final code for our Arduino Light Sensor Circuit

```
int greenLedPin = 2;
int yellowLedPin = 3;
int redLedPin = 4;

int lightSensorPin = A0;
int analogValue = 0;

void setup() {
   pinMode(greenLedPin, OUTPUT);
   pinMode(yellowLedPin,OUTPUT);
   pinMode(redLedPin,OUTPUT);
}

void loop(){
   analogValue = analogRead(lightSensorPin);
   if(analogValue < 50){
      digitalWrite(redLedPin, HIGH);
   }
   else if(analogValue >= 50 && analogValue <= 100){
      digitalWrite(yellowLedPin, HIGH);
   }
   else{
      digitalWrite(greenLedPin, HIGH);
   }
   delay(200);
   digitalWrite(greenLedPin, LOW);
   digitalWrite(yellowLedPin, LOW);
   digitalWrite(redLedPin, LOW);
}
```

Arduino
Motion Sensor

Project Description

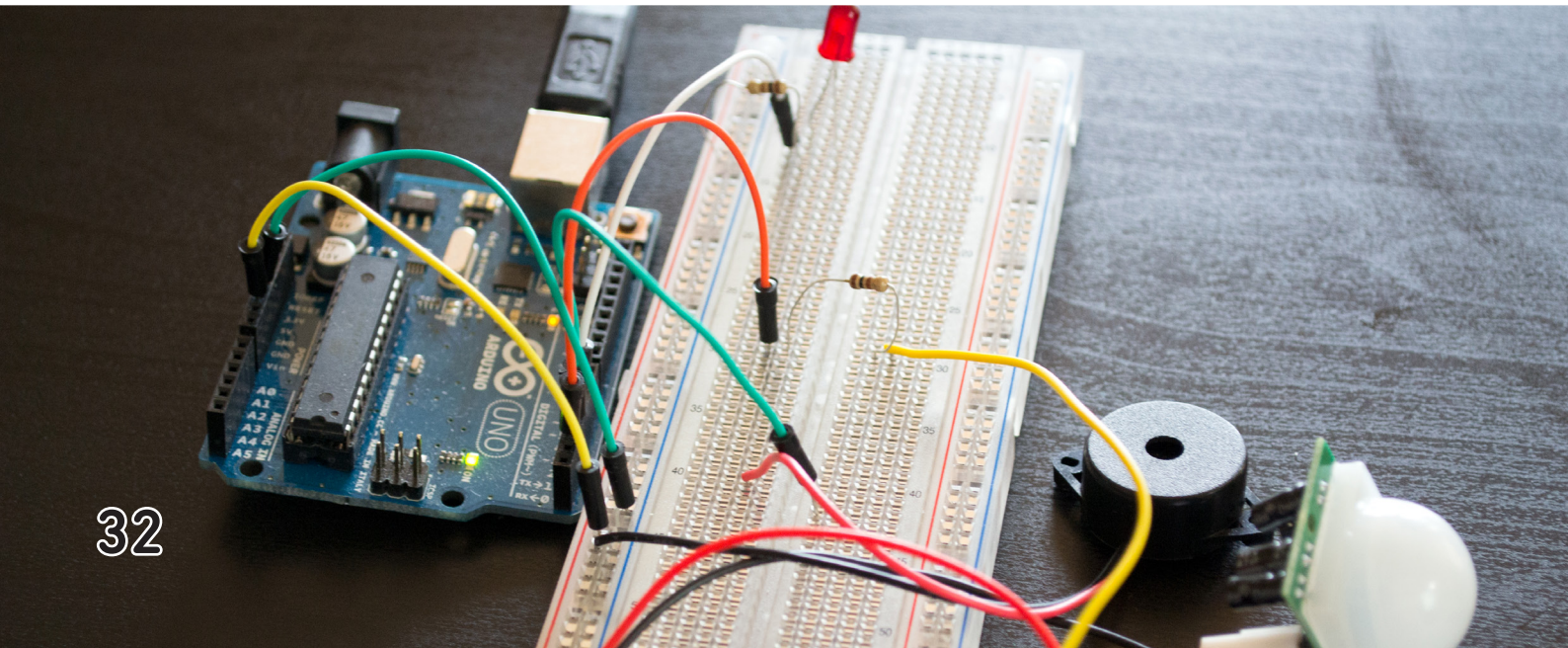
In this tutorial, we will be building a simple Arduino motion sensor that makes use of a PIR sensor to be able to detect motion. This simple circuit can be extended to do some pretty cool stuff, for example, activate lights, a speaker and much more.

In this particular tutorial, we will be making use of both a LED and piezo buzzer to show when motion has been detected through the PIR motion sensor.

Equipment

Required

- Arduino Uno
- 1 x LED
- 2 x 100 Ohm Resistor (Brown Black Brown)
- 1 x Piezo Buzzer
- 1 x PIR Sensor
- Breadboard
- Breadboard wire

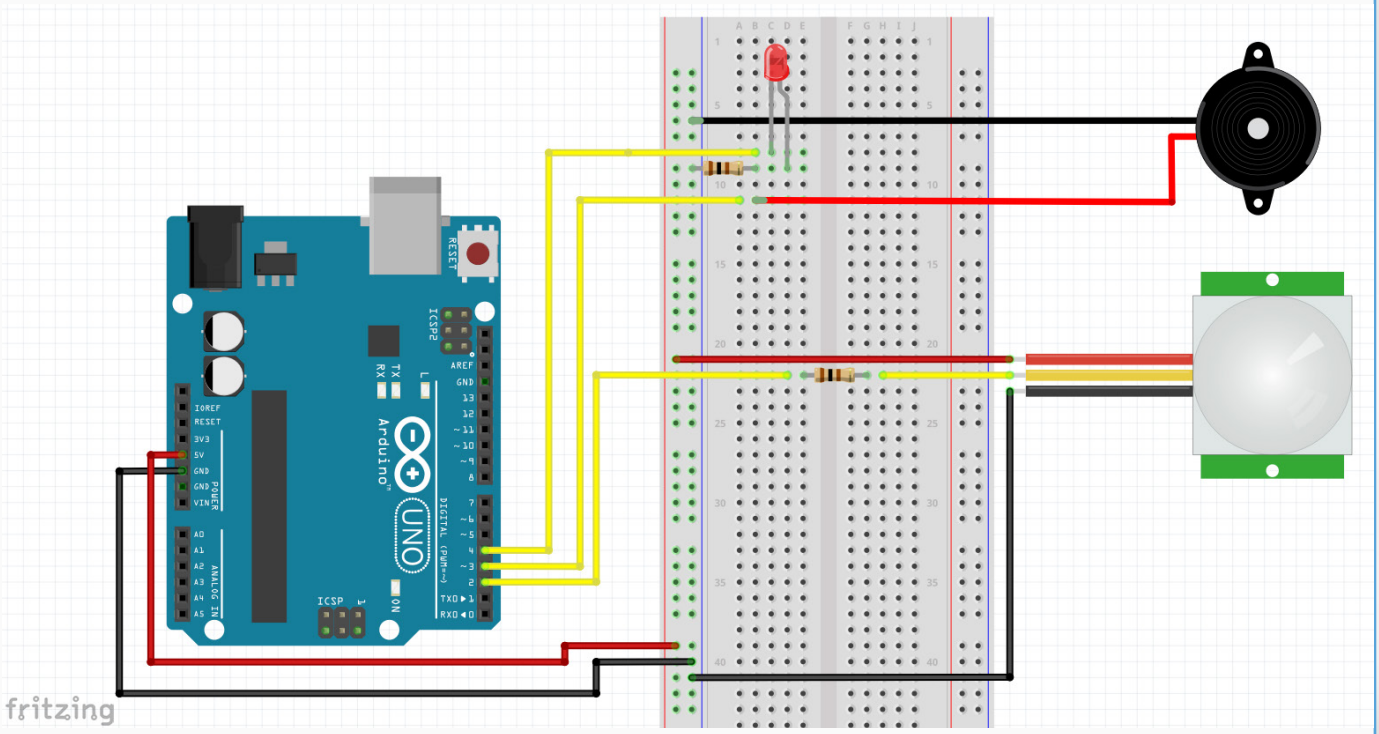


The Arduino Motion Sensor Circuit

The PIR sensor is probably the most complicated part we have in our circuit which showcases how simple it really is. This device has three wires coming from it. The Red is a positive power source (5v). The black is for ground. Finally, the yellow wire is the output and turns live whenever motion is detected. On the PIR sensor, you can adjust the sensitivity and time for the detection of motion.

The Piezo buzzer is a super simple speaker. It has a ground wire and a positive wire. We will connect the positive wire to a pin on the Arduino. Whenever the pin outputs a voltage (High), the speaker will make a sound.

Below we have included the PIR motion sensor Arduino circuit diagram. Underneath the circuit diagram we go into step by step on how to put this device together.



1. First connect the **5v pin** from the Arduino to the **positive rail** on the breadboard.
2. Next connect the **ground(GND) pin** from the Adruino to the **negative rail** on your breadboard.
3. For the PIR Sensor, follow the following steps:
   - Run the black wire from the sensor to the negative rail on the breadboard.
   - Run the red wire from the sensor to the positive rail on the breadboard.
   - Place a 100-ohm resistor onto the middle of the breadboard as shown in the diagram.
   - Run the yellow wire from the sensor to one end of the resistor, on the other end run a wire to pin 2 on the Arduino board.
4. Now for the Piezo buzzer follow the following steps:
   - Run the red wire to pin 3 on the Arduino
   - Now wire the black wire to the ground rail on the breadboard.
5. Now for the red LED follow the following steps:
   - Run a 100-ohm resistor from the ground rail on the breadboard to a spot on the breadbaord.
   - Connect the negative end of the LED to the resistor, and the other end of the LED into somewhere else on the breadboard.
   - Finally place a wire from pin 4 on the Arduino to the positive side of your LED.

Before we move on you may want to double check your connections to make sure they're all done properly. The circuit diagram above is probably the best thing to refer to.

Coding for the Arduino Motion Sensor Circuit

1. Begin writing for this project by opening up the Arduino IDE, save it by pressing **Ctrl + S** and then create a new file by pressing **Ctrl + N**.

2. In this new sketch file begin writing the following lines of code.
In this first block of code, we set 4 different variables. The first three are setting pin numbers to a variable. These variables are so we can quickly reference and recognize that pin further down in the code.

The fourth variable is where we store the state of our PIR sensor. Low means that no motion has been detected while high means it has been detected. We will keep this variable as low, to begin with.

```
int ledPin = 4; // Pin LED is connected to
int piezoBuzzerPin = 3; // Pin Piezo Buzzer is connected to
int pirSensorPin = 2; // PIN PIR Sensor is connected to
int motionDetected = LOW; // Start MotionDetected as low (No motion detected)
```

In this next block of code we set up all our pins and anything else, we may need to initialize. For both the piezo buzzer and the LED we set these to be OUTPUT as we want to send these HIGH or LOW to be able to switch them on and off.

For the PIR sensor, we want to listen to it for when motion is detected so we will set it to be an input.

Finally, we set the **Serial.begin**, so we can debug our code if required. **Serial.println("hello")** will allow you to print debug messages to the console. Finally, we have a delay for 5 seconds to allow for the PIR sensor to initialize. You can tinker with this value for the shortest delay without the sensor throwing false positives.

```
void setup() {
  pinMode(ledPin, OUTPUT);      // declare LED as output
  pinMode(pirSensorPin, INPUT);     // declare the PIR sensor as input
  pinMode(piezoBuzzerPin, OUTPUT); //declare buzzer as output
  Serial.begin(9600); //Set serial out if we want debugging
  delay(5000); //Allow time for the PIR Sensor to calibrate
}
```

Now we have the core of our program. The loop function will continue to loop through until power is disconnected from the Arduino or a new program is uploaded.

Firstly, we check to see if the PIR sensor is high or not. If it is high, then we enter the if statement otherwise we keep the LED and the piezo buzzer off.

If the sensor is high, then we turn the LED on and set the buzzer to be at a certain pitch by using an analog output. We then wait for 100 Ms then turn the LED off and change the pitch of the Piezo buzzer. We repeat this until the PIR sensor goes back to low (No motion detected).

```
void loop(){
  motionDetected = digitalRead(pirSensorPin);  // Read the PIR sensor
  if(motionDetected == HIGH) //If motion detected
  {
    digitalWrite(ledPin, HIGH);
    analogWrite(piezoBuzzerPin, 200);
    delay(100);
    analogWrite(ledPin, LOW);
    analogWrite(piezoBuzzerPin, 25);
    delay(100);
  }
  digitalWrite(ledPin, LOW);
  digitalWrite(piezoBuzzerPin,LOW);
}
```

Coding for the Arduino Motion Sensor Circuit

3. With the project now written to our sketch file we need to simply push the code to our Arduino. We can do this by pressing the "**Upload**" button which is right next to the "**Verify**" button (The button with the tick).

If you would like to compare your code against the full version of the code, you can look below after the final steps.

4. You can now test your circuit by simply making motion in front of the sensor, if everything is working correctly the LED should appear to flash and you should hear a buzz from the piezo buzzer.

Final code for the Arduino Motion Sensor Circuit

```
int ledPin = 4; // Pin LED is connected to
int piezoBuzzerPin = 3; // Pin Piezo Buzzer is connected to
int pirSensorPin = 2; // PIN PIR Sensor is connected to
int motionDetected = LOW; // Start MotionDetected as low (No motion detected)

void setup() {
   pinMode(ledPin, OUTPUT);      // declare LED as output
   pinMode(pirSensorPin, INPUT);     // declare the PIR sensor as input
   pinMode(piezoBuzzerPin, OUTPUT); //declare buzzer as output
   Serial.begin(9600); //Set serial out if we want debugging
   delay(5000); //Allow time for the PIR Sensor to calibrate
}

void loop(){
   motionDetected = digitalRead(pirSensorPin);  // Read the PIR sensor
   if(motionDetected == HIGH) //If motion detected
   {
      digitalWrite(ledPin, HIGH);
      analogWrite(piezoBuzzerPin, 200);
      delay(100);
      analogWrite(ledPin, LOW);
      analogWrite(piezoBuzzerPin, 25);
      delay(100);
   }
   digitalWrite(ledPin, LOW);
   digitalWrite(piezoBuzzerPin,LOW);
}
```

Arduino
LCD Display

Project Description

In this Arduino LCD tutorial, I will take you through the steps to connecting a simple 16×2 LCD up to the Arduino.

There is a ton that you're able to do with an LCD (liquid crystal display), so it's a good little device to learn how to connect and communicate with.

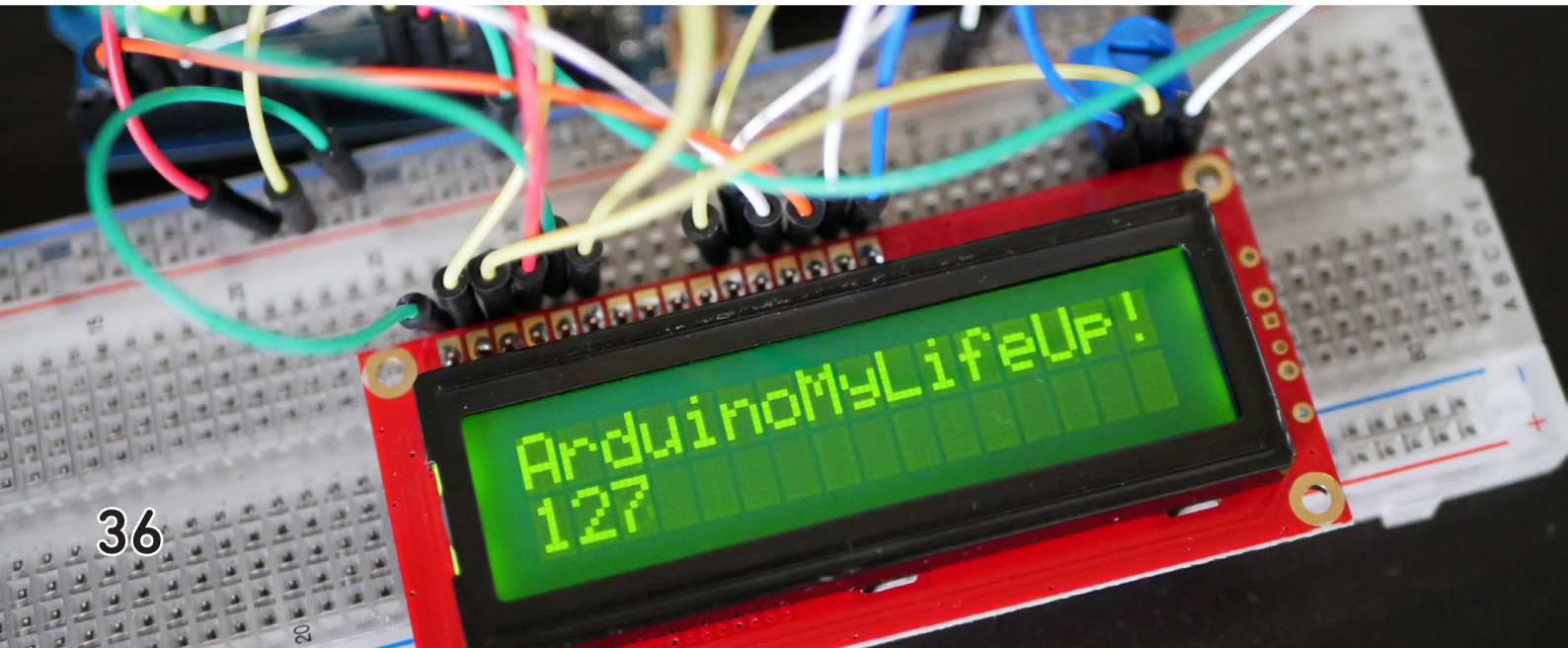
You will find that most LCD boards are not assembled with header pins so these will need to be soldered on. If you have header pins, then this isn't too much of a task. You can probably skip soldering but getting a good connection to the board will be difficult.

The potentiometer you find in the circuit is used to control the brightness of the screen. If you find nothing is displaying or it doesn't look right, then try turning this up or down.

Equipment

Required

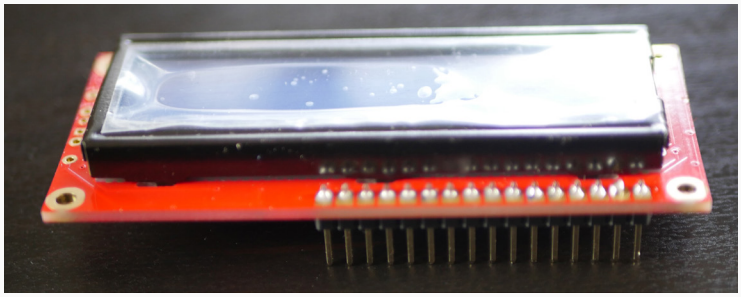
- Arduino Uno
- 16 x 2 LCD Board
- 16 x Header Pins (If LCD Board has none)
- 1 x Breadboard
- 1 x Breadboard Wire
- 10k ohm Potentiometer



The Arduino LCD Display Circuit

The circuit for the Arduino liquid crystal display is surprisingly easy. You will find that the most off-putting thing about it is how many wires there are that needs to be hooked up.

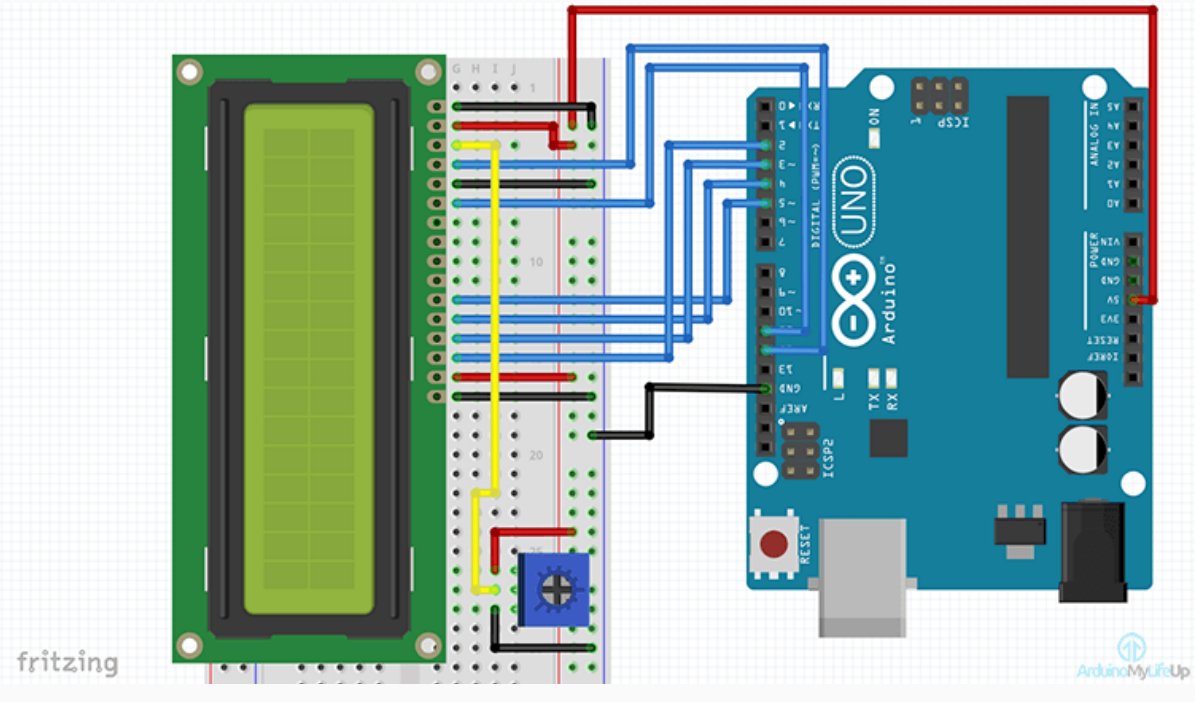
It's important to note that you are likely going to need to solder header pins onto the LCD. This is a pretty straightforward process.



1. Place the header pins so the short side sticks up through the holes on the display.
2. Now solder the pins one by one ensuring you don't accidentally connect two up. If you do connect two up then melt the solder and suck it up using a solder sucker.
3. Once done it's ready for use.

Below are the steps that you will need to follow for hooking up the display. You can find the full circuit diagram right underneath the steps if you rather follow that.

1. First connect the **5V pin** from the Arduino to the **positive line** on the **breadboard**.
2. Connect the **GND pin** from the Arduino to the **negative line** on the **breadboard**.
3. Connect the **potentiometer** to the breadboard and wire the **positive pin** to the breadboards **positive rail**. Also wire the **ground pin** of the **potentiometer** to the **ground rail** on the breadboard.
4. Connect the following wires to the LCD screen, with pin 1 being the closest to the edge of the board.
   - Ground rail to **pin 1**, **pin 5** and **pin 16** on the LCD Board.
   - Positive rail to **pin 2** and **pin 16** of the LCD Board.
   - Middle wire on the **potentiometer** to **pin 3** on the LCD Board.
5. Wire the following Arduino Digital pins to the LCD Board.
   - **Pin 12** on the **Arduino** to **Pin 4** on the **LCD Board**.
   - **Pin 11** on the **Arduino** to **Pin 6** on the **LCD Board**.
   - **Pin 5** on the **Arduino** to **Pin 11** on the **LCD Board**.
   - **Pin 4** on the **Arduino** to **Pin 12** on the **LCD Board**.
   - **Pin 3** on the **Arduino** to **Pin 13** on the **LCD Board**.
   - **Pin 2** on the **Arduino** to **Pin 14** on the **LCD Board**.



Coding to communicate with the LCD Display

The code for communicating with the display is pretty straightforward. You may have thought that communicating with the the LCD board in the code would be complicated but gladly it's not thanks to a helpful library by Limor Fried that's already included in Arduino main library collection.

It's important to know that there is are already some code examples within the Arduino software. To find them simply go to **File -> Examples -> LiquidCrystal**

In here you will see a range of different examples with each showing you how to use certain methods of the liquid crystal library. To begin, I suggest taking a look at the hello world example. The code for this example is below.

```
// include the library code:
#include

//initialise the library with the numbers of the interface pins
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);

void setup() {
  // set up the LCD's number of columns and rows:
  lcd.begin(16, 2);
  // Print a message to the LCD.
  lcd.print("hello, world!");
}

void loop() {
  // set the cursor to column 0, line 1
  lcd.setCursor(0, 1);
  // print the number of seconds since reset:
  lcd.print(millis() / 1000);
}
```

Coding to communicate with the LCD Display

In the code you will need to setup a variable of type LiquidCrystal. As you can see below.

```
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);
```

The numbers that are passed as parameters represent the pin number on the Arduino we have connected to the relevant pin on the display. Below is an example of the function again but instead it contains the corresponding pins on the LCD.

```
LiquidCrystal(rs, enable, d4, d5, d6, d7)
```

The above function can be extended to read from 8 data lines in total.

```
LiquidCrystal(rs, enable, d0, d1, d2, d3, d4, d5, d6, d7)
```

The **lcd.begin** function call located in the setup function in the code above is used to initialize the interface to the LCD. Here is where you need to specify how many columns and rows there are on the display.

For example, my display has **16 columns** and **2 rows**. This means we need to set it up with 16 as the first parameter and 2 as the 2nd parameter **lcd.begin(cols, rows)**

Basic Methods to deal with the LCD

If you ever need to clear the display then simply call the method **lcd.clear();** This will wipe the display so that you can display a new value on there.

Make sure you have a delay beforehand so you're able to see whatever is being displayed on the screen. You can do this by utilizing the **delay(millisecconds)** function.

If you ever just want to move the cursor back to home rather than clearing the display then **lcd.home();** will move the **cursor** to **position 1**.

If you want to print a string on the LCD display then the print method is what you need. Just simply call it like this **lcd.print("My String");**

The last basic method I will mention is write. This method is used to write a character to the screen. Very much like print simply call it like this: **lcd.write('I');**

Extra Functions to deal with the LCD Display

- **Auto Scroll:** Shift text right and left.
  autoscroll(); This will move the text one space to the left whenever a new character is added.
  noAutoScroll(); Turns off auto scrolling.
- **Cursor:** This allows you to turn on and off the underscore cursor.
  noCursor(); Turns off the underscore cursor.
  Cursor(); Turns on the underscore cursor.
- **Blink:** Turns on and off the blinking of the cursor.
  noBlink(); Turns off the blinking cursor.
  blink(); Turns on the blinking cursor;
- **Display:** Make the display go blank without losing the current text.
  noDisplay(); Turns off the display.
  display(); Turns on the display.
- **Scroll:** Allows you to scroll the text both left and right.
  scrollDisplayLeft(); Will scroll one position to the left.
  scrollDisplayRight(); Will scroll one position to the right.
- **Serial Display:** You can setup the board to accept serial input from something like the serial monitor.
  You can print the text sent through the serial port on the screen.

  write(Serial.read())
- **Set Cursor:** Sets the cursor to a specific position. (Location, Line)
  setCursor(0, 0); Will set the cursor to be at the top left.
  setCursor(15, 1); Will set the cursor to be at the bottom right.
- **Text Direction:** Allows you to tell which way the text should flow from the cursor.
  rightToLeft(); Forces the text to flow from the left of the cursor.
  leftToRight(); Forces the text to flow from the right of the cursor.

Arduino Temperature Sensor

Project Description

In this tutorial, we will be looking at how to setup the Arduino DS18b20 temperature sensor and anything else you need to know about it.

This project is pretty cool if you want to set up a data logger or just something to monitor the temperatures of a certain room. You could combine this tutorial with another to create a pretty cool smart sensor. You could use something like a piezo buzzer to alert you if the temperature falls outside a certain range for example.

In this tutorial, we will be showing you the basics of connecting the DS18B20 to the Arduino correctly. We will also be adding some LEDs to just demonstrate the circuit & code functionality. If you want to stay up to date with all my projects and much more, then be sure to follow me on any of the major social networks.

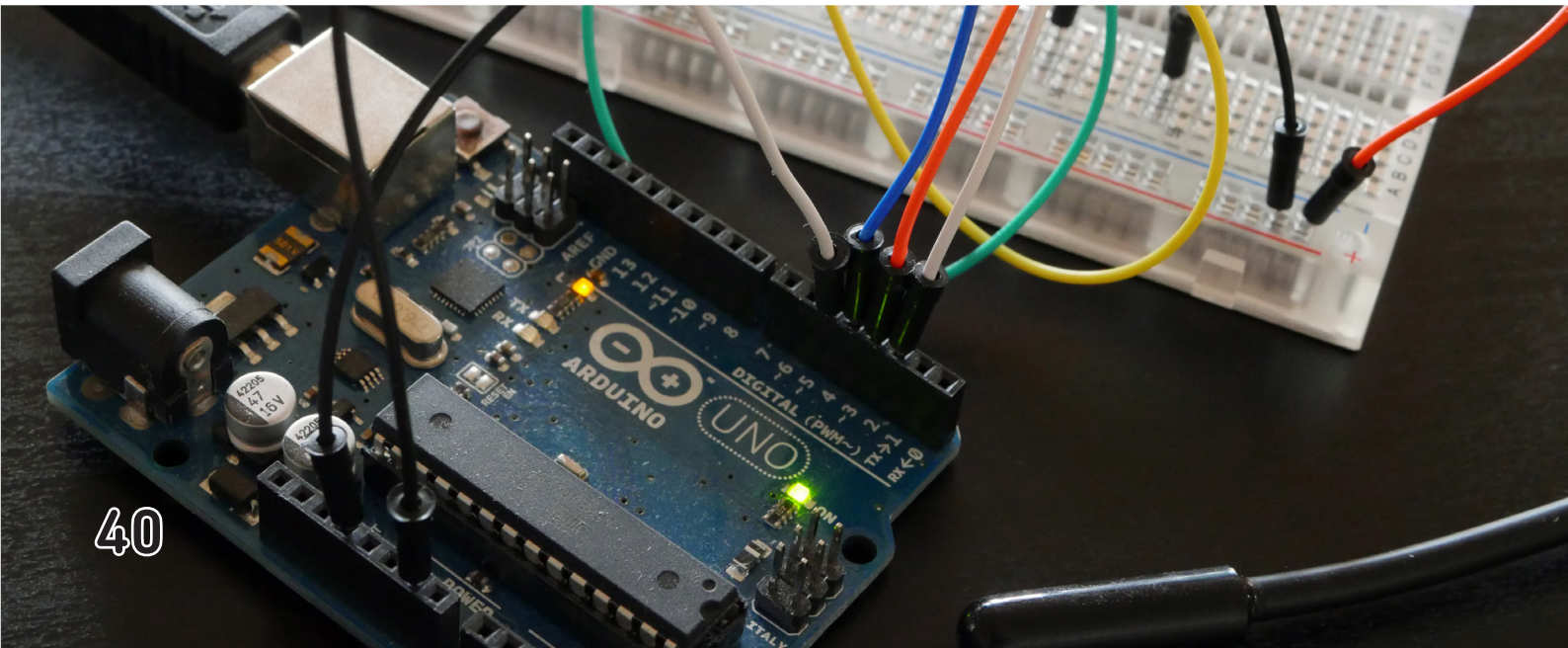
Equipment

Required

- Arduino Uno
- 1 x DS18B20 Temperature sensor
- 1 x 4.7k-ohm Resistor
- Breadboard
- Breadboard Wire

Optional

- 3 x 100-ohm resistors
- 1 x Red LED
- 1 x Green LED
- 1 x Yellow LED



The Arduino Temperature Sensor Circuit

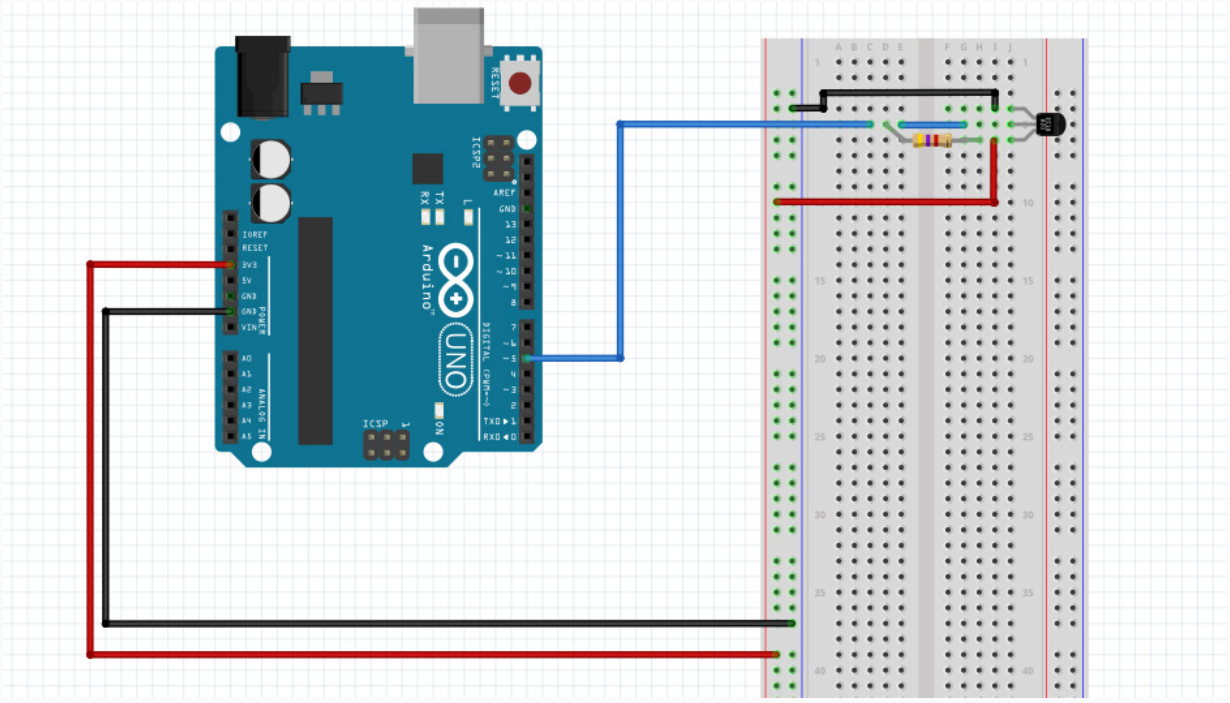
The circuit for the Arduino DS18B20 sensor is pretty simple and if you don't want the LEDs just leave that part of the circuit out. I will just quickly describe the temperature sensor below however the rest of equipment is pretty straightforward and doesn't need explaining.

The DS18B20 is a digital temperature sensor that is capable of reading temperatures within 0.05°C. It also has one wire support which means it's capable of sharing a single wire with other devices that also support one wire.

In this tutorial, we use a waterproof version of the DS18B20 so it simply just looks like a long thick wire with three wires sticking out of 1 end. If you get the device without any extras, it simply looks like a normal transistor.

Now let's move on to putting the circuit together.

1. First connect the **3v3 pin** from the Arduino to the **positive rail** & a **ground pin** to the **ground rail** on the breadboard.
2. Now place the **DS18B20 sensor** on to the breadboard.
3. Place a **4.7k ohm resistor** between the **positive lead** (**Red Wire**) and the **output lead** (**White Wire**) of the sensor.
4. Next, place a wire from the **positive lead** (**Red Wire**) to the **positive 3v3 rail**.
5. Place a wire from the **output lead** back to **pin #5** on the Arduino. Place a wire from the **ground lead** (**black wire**) to the **ground rail**.
6. Now the next few steps are optional and are only required if you want the LEDS.
7. Place a wire from the **ground rail** to the **ground rail** on the **opposite side** of the **breadboard**.
8. Place the 3 LEDs onto the breadboard. (**Red**, **Yellow** and **Green**)
9. Connect a **100-ohm resistor** to **each** LED and have this go to the **ground rail**.
10. Now have a wire come from the following Arduino pins: **pin 2** to the **green LED**, **pin 3** to a **yellow LED** and finally the **red LED** to the **pin 4**.
11. Now that's the circuit all setup, it's now time to move onto the code.



Installing one wire support

Now by default the Arduino doesn't have one wire support so we will need to download and install the library for this. If you want some extensive information on what is one wire, then be sure to check Arduinos official website. Now this is a pretty straightforward process that we will take you through now.

1. Firstly, download the latest version of the One Wire library, you can download it from here: https://github.com/PaulStoffregen/OneWire/archive/master.zip
2. Once downloaded load up the Arduino IDE.
3. Within the Arduino IDE, click **Sketch -> Include Library -> Add .zip Library**
4. Find the .zip file that you downloaded in step one and select it.
5. It should now say something like "**Library added to Libraries. Check 'include Libarary' menu**".
6. So now again click **Sketch** -> **Include Library** and then under contributed libraries you will find **onewire**, Click on it.
7. This will add a **#include** at the top of your file and will be added to your Adruino at the next upload.

Installing the Dallas Temperature Library

To make things a lot easier, we are going to suggest installing a library to handle the data that comes from the sensor. This will make it a lot easier than to write the data processing code yourself. If you want to do it yourself without an additional library, then be sure to check out this one wire page for more information. Alternatively, you can look at an analog sensor such as the TMP36 which is super easy to get going on the Arduino.

If you're happy using the dallas temperature sensor library then simply do the following.

1. First download the Dallas Temperature Sensor Library, you can download it from: https://github.com/milesburton/Arduino-Temperature-Control-Library/archive/master.zip
2. Now once that has downloaded go into the Arduino IDE and repeat the steps we did before for the onewire library.
3. Go to **Sketch** -> **Include Library** -> "**Add .zip library**".
4. Find the .zip file that you downloaded in step one and select it.
5. It should now say something like "**Library added to Libraries. Check 'include Libarary' menu**".
6. So now again click **Sketch** -> **Include Library** and then under contributed libraries you will find **DallasTemperature**, Click on it.
7. This will automatically add a **#include** at the top of the script. The library will be synced with your Arduino when you upload it.

Now let's move onto writing the rest of the code to both get the temperature from the sensor and turn on the correct LED.

Coding for the DS18B20 on the Arduino

This next part of the tutorial I will go through the code for reading the sensor. It's pretty straight forward however if you're new to coding it may be a little overwhelming.

Remember to write this all into a new Arduino Sketch file.

To begin we need to include both of the **onewire** and the **dallasTemperature** headers so we can use these within the code. These should already be there if you followed the steps above.

```
#include <OneWire.h>
#include <DallasTemperature.h>
```

Next we declare all the variables that we will need to use throughout the script. The first of these represent the pin numbers that the devices are connected to.

The float temperature variable is where we will be storing our temperature value.

The **lowerlimit** and **upperlimit** variables represent our threshold temperatures. Anything below the lower limit will trigger the yellow LED to turn on. Anything above the upper limit will turn on the red LED. Anything between these will turn on the green LED.

```
int greenLedPin = 2;
int yellowLedPin = 3;
int redLedPin = 4;

int temp_sensor = 5;

float temperature = 0;
int lowerLimit = 15;
int higherLimit = 35;
```

Next we create a OneWire object by using our pin we defined earlier (5). If you have a large amount of 1 wire sensors, it's suggested you split these up across multiple pins. For each just make a new **Onewire** object. For example, **OneWire oneWirePin2(temp_sensor2);**

Now we create our Dallas Temperature sensor object by passing our OneWire reference into the class.

```
OneWire oneWirePin(temp_sensor);
DallasTemperature sensors(&oneWirePin);
```

In our setup function we activate the Arduino serial interface so we can monitor the output lines in the code. This is also where we need to setup our LEDs to all act as outputs. Lastly we make a call to **sensors.begin()**, this will set up our sensor so we can start requesting data from it.

```
void setup(void){
  Serial.begin(9600);

  //Setup the LEDS to act as outputs
  pinMode(redLedPin,OUTPUT);
  pinMode(greenLedPin,OUTPUT);
  pinMode(yellowLedPin,OUTPUT);

  sensors.begin();
}
```

Coding for the DS18B20 on the Arduino

In this segment we have a few output lines to just let us know when it has successful requested all the temperatures of the sensors connected to our onewire pin (5).

We next store the value from the **sensors.getTempCByIndex(0)** into the temperature variable. If you want to get the fahrenheit temperature simply change **C** to **F**. For example **sensors.getTempFByIndex(0).** Also the **0** refers to which sensor we want information from. **0 = the first sensor**, if you had a 2nd sensor then it would be **1**.

```
void loop(){
  Serial.print("Requesting Temperatures from sensors: ");
  sensors.requestTemperatures();
  Serial.println("DONE");

  temperature = sensors.getTempCByIndex(0);
```

We then turn all the LEDS to low so that only the correct LED remains on when we run the next segment of code. We also print two statements the first being a simple string that says "**Temperature is**" and then the temperature itself. We finish off this text in the next segment of code described below.

```
  digitalWrite(redLedPin, LOW);
  digitalWrite(greenLedPin, LOW);
  digitalWrite(yellowLedPin, LOW);

  Serial.print("Temperature is ");
  Serial.print(temperature);
```

This last section compares our temperature values with our predefined values. Depending on the result it will turn the relevant LED on. For example, the first if statement says if the temperature is lower or equal to the lowerlimit value then do this.

Lastly we also delay for 500ms before looping through the process again.

```
  //Setup the LEDS to act as outputs
  if(temperature <= lowerLimit){
    Serial.println(", Yellow LED is Activated");
    digitalWrite(yellowLedPin, HIGH);
  }
  else if(temperature > lowerLimit && temperature < higherLimit){
    Serial.println(", Green LED is Activated");
    digitalWrite(greenLedPin, HIGH);
  }
  else if(temperature >= higherLimit){
    Serial.println(", Red LED is Activated");
    digitalWrite(redLedPin, HIGH);
  }
  delay(500);
}
```

With all the code written in you can now push it to your Arduino by pressing the "Upload" code button. If you want to see what the full code is meant to look like to make sure you havent entered any mistakes then checkout the next page.

Full code for the DS18B20 on the Arduino

```
#include <OneWire.h>
#include <DallasTemperature.h>

int greenLedPin = 2;
int yellowLedPin = 3;
int redLedPin = 4;

int temp_sensor = 5;

float temperature = 0;
int lowerLimit = 15;
int higherLimit = 35;

OneWire oneWirePin(temp_sensor);
DallasTemperature sensors(&oneWirePin);

void setup(void){
   Serial.begin(9600);

   //Setup the LEDS to act as outputs
   pinMode(redLedPin,OUTPUT);
   pinMode(greenLedPin,OUTPUT);
   pinMode(yellowLedPin,OUTPUT);

   sensors.begin();
}

void loop(){
   Serial.print("Requesting Temperatures from sensors: ");
   sensors.requestTemperatures();
   Serial.println("DONE");

   temperature = sensors.getTempCByIndex(0);

   digitalWrite(redLedPin, LOW);
   digitalWrite(greenLedPin, LOW);
   digitalWrite(yellowLedPin, LOW);

   Serial.print("Temperature is ");
   Serial.print(temperature);
  //Setup the LEDS to act as outputs
   if(temperature <= lowerLimit){
      Serial.println(", Yellow LED is Activated");
      digitalWrite(yellowLedPin, HIGH);
   }
   else if(temperature > lowerLimit && temperature < higherLimit){
      Serial.println(", Green LED is Activated");
      digitalWrite(greenLedPin, HIGH);
   }
   else if(temperature >= higherLimit){
      Serial.println(", Red LED is Activated");
      digitalWrite(redLedPin, HIGH);
   }
   delay(500);
}
```

Arduino
Simple Battery Tester

Project Description

The Arduino battery tester is a pretty cool nifty tool that you can use to check how charged a battery is. This is perfect for anyone who wants to know roughly how long a battery has left before it runs out of charge.

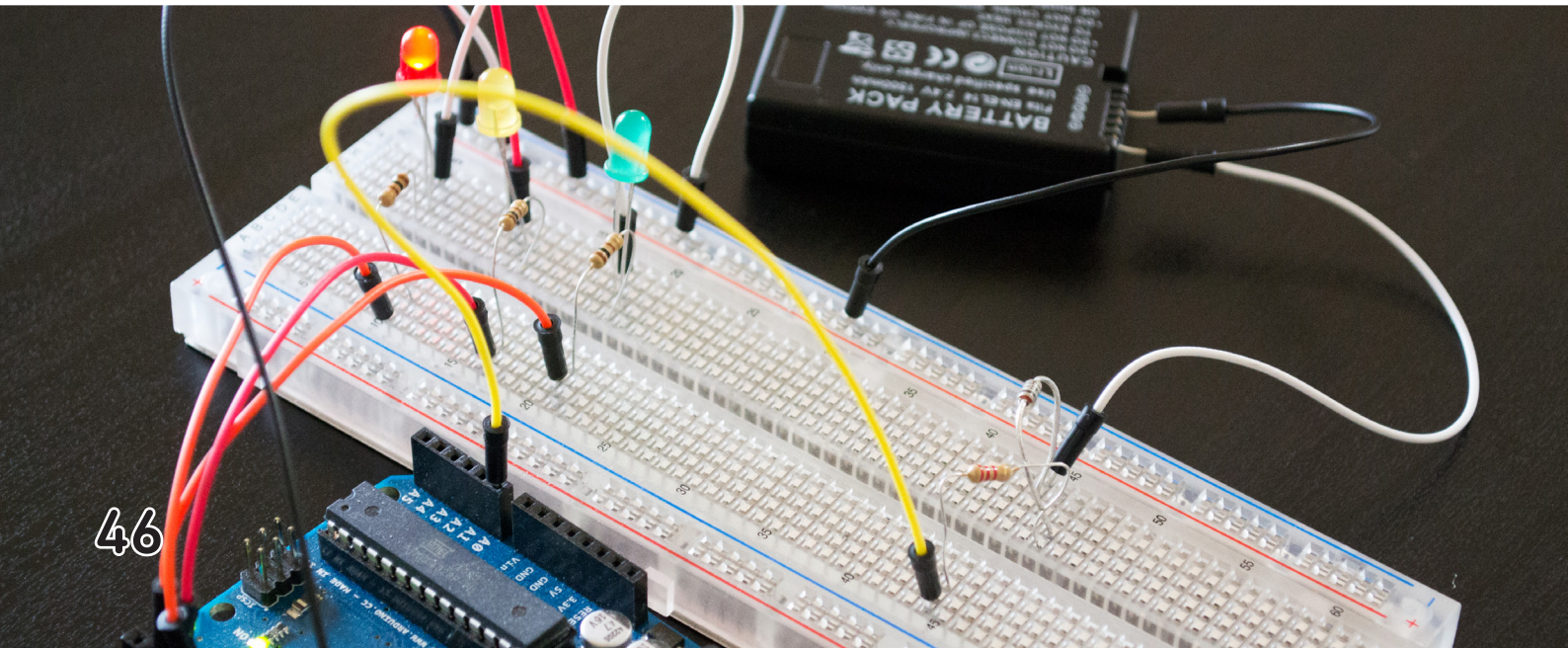
This is a pretty barebones beginner Arduino project but it could be extended to be a proper setup. We will be utilizing a small amount of code and some really simple components that you can obtain incredibly cheapily.

Best of all you could have this up and running within almost 30 minutes.

Equipment

Required

- Arduino Uno
- 1 x Green LED
- 1 x Yellow LED
- 1 x Red LED
- 1 x 5.1k Zener Diode
- 3 x 100-ohm resistors
- 1 x 2.2k-ohm resistors
- Breadboard
- Breadboard Wire



The Simple Battery Tester Circuit

This battery tester circuit is pretty easy to put together circuit. If you have been following all of the tutorials, then you will notice that we are using a new component called the Zener diode.

The Zener diode will allow you to test batteries that have a voltage greater than 8 volts. A Zener diode works by allowing current to flow in one direction until it hits the breakdown voltage (limit on the diode, in our case it is 5.1 volts). Once it hits this limit, it allows voltage to go in the reverse direction. This can help protect parts that can only handle a certain amount of voltage.

The 2.2k ohm resistor reduces the current coming from the battery to something that the Arduino will be able to take in. If your current is too high, then it may damage your Arduino.

The circuit also has three different LEDs, each of these LEDs represents roughly how much charge there is left in the battery. Red will represent the battery is low/almost dead. Yellow will represent the battery being roughly half used up. Finally, the green LED will represent the battery being full.

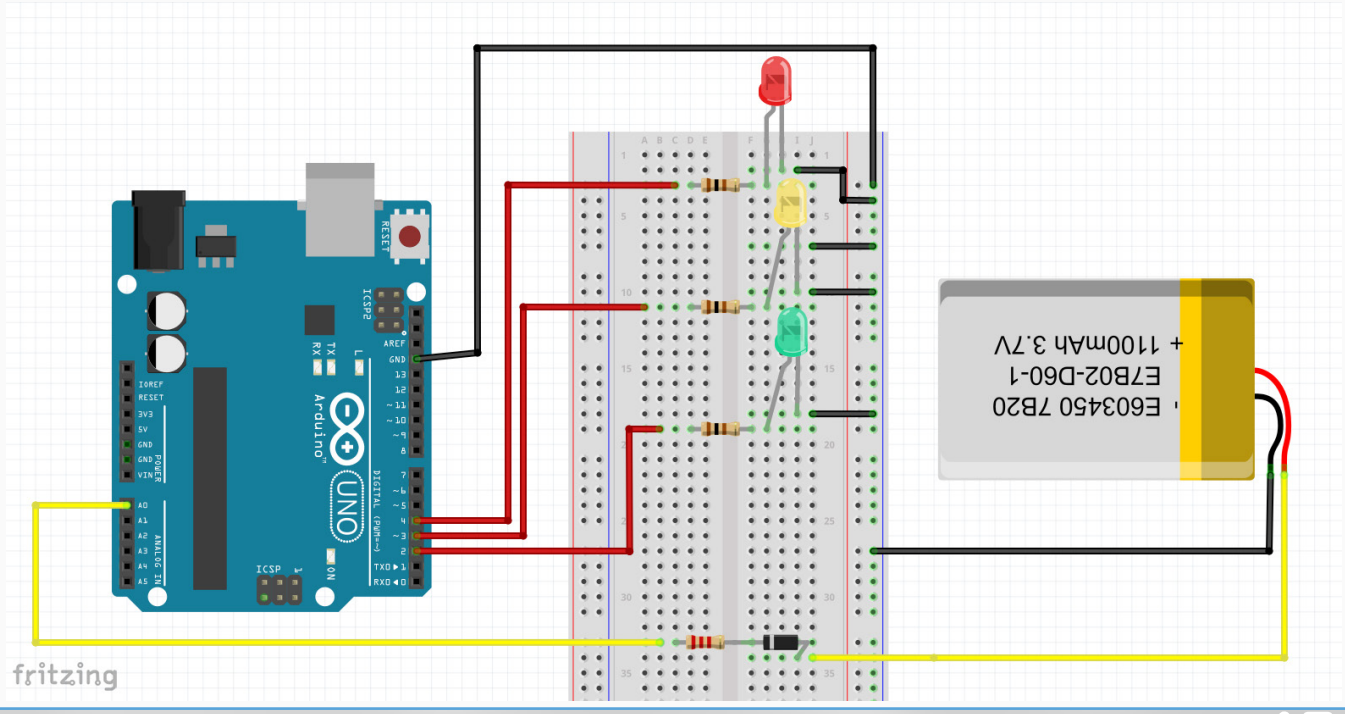
I will now quickly go through the steps of putting this together below. Underneath the instructions you will find a circuit diagram if you find it easier just following that.

1. Wire the **ground pin** on the Arduino to the **ground rail** on the breadboard.
2. On the breadboard place the **green**, **red** and **yellow** LED's. Connect the **ground pins** to the **ground rail**.
3. Place a 100-ohm resistor onto positive end of the LEDS then hook a wire from resistor to the relevant pins on the Arduino.

The following LEDS should connect to the relevant pin numbers.

- **Red LED** to **Pin 4** on the **Arduino**
- **Yellow LED** to **Pin 3** on the **Arduino**
- **Green LED** to **Pin 2** on the **Arduino**

4. Now connect from analogue pin 0 (A0) to the breadboard. After this add a 2.2k resistor and the Zener diode (with the line on Zener diode facing towards the Arduino). Finally have a loose wire coming from the other end to diode.
5. Finally have a loose wire connected to the ground rail



Coding For The Simple Battery Tester Circuit

Firstly, we need to set up all our variables; the LED'S variables are all assigned to their relevant pin number mentioned earlier.

The analog value variable is where we will be storing the value that comes from the analog input. We then do a calculation (Which I will explain later) and store the resulting number into a variable called voltage.

Finally, **ledDelay** is how long you want the LEDs to remain on before switching off.

```
int greenLed = 2;
int yellowLed = 3;
int redLed = 4;

int analogValue = 0;
float voltage = 0;
int ledDelay = 1000;
```

The setup function is called once and the perfect place to set up all our pins. In this particular program, we only need to set up all our LED pins as outputs.

```
void setup()
{
   pinMode(greenLed, OUTPUT);
   pinMode(yellowLed,OUTPUT);
   pinMode(redLed,OUTPUT);
}
```

Inside the loop function, we will do a couple of things. Firstly, we read the analog pin, the value of this pin will be between **0-1023**. We will need to do a calculation to convert this to a variable, so we simply multiply the a**nalogueValue** by **0.0048** to do this.

```
void loop()
{
   analogValue = analogRead(A0);
   voltage = 0.0048*analogValue;
```

This last part we compare our calculated voltage and compare this to our defined voltage values. Whenever the voltage falls between a set of values, we simply turn the relevant LED. You can change voltage variables to get the LED's to display how you like.

```
   if( voltage >= 1.6 )
      digitalWrite(greenLed, HIGH);
   else if (voltage > 1.2 && voltage < 1.6)
      digitalWrite(yellowLed, HIGH);
   else if( voltage <= 1.2)
      digitalWrite(redLed, HIGH);

   delay(ledDelay);
   digitalWrite(redLed, LOW);
   digitalWrite(yellowLed, LOW);
   digitalWrite(greenLed, LOW);
}
```

Once you are done with the circuit simply plug the Arduino into a computer and upload the code. Once this is done we can continue onto testing and checking that it is working correctly. If you want to compare what you have written to the full version of the script then go to the end of this tutorial.

Testing the Simple Battery Tester

When you first turn on and deploy the new code to the Arduino, you will notice that it keeps jumping between the leds.

This is because the analog input wire is floating and picking up noise causing our program to detect false positives.

To stop this, you can simply ground the analog wire to the ground rail when not in use. You can also try grounding the other analog pins to help reduce the amount of noise it's picking up.

To test it, all you need to do is hook up a battery to the two wires. Place the ground wire to the negative end of the battery and the positive wire to the positive end of the battery. The Arduino should pick up the voltage and tell the relevant LED to light up.

If it doesn't then have a look at adding some debug lines for the battery input. This should tell you if there is something wrong with reading the input or there is simply a mistake for displaying the LEDs.

The Final Code for the Battery Tester

```
int greenLed = 2;
int yellowLed = 3;
int redLed = 4;

int analogValue = 0;
float voltage = 0;
int ledDelay = 1000;

void setup()
{
   pinMode(greenLed, OUTPUT);
   pinMode(yellowLed,OUTPUT);
   pinMode(redLed,OUTPUT);
}

void loop()
{
   analogValue = analogRead(A0);
   voltage = 0.0048*analogValue;
  if( voltage >= 1.6 )
      digitalWrite(greenLed, HIGH);
   else if (voltage > 1.2 && voltage < 1.6)
      digitalWrite(yellowLed, HIGH);
   else if( voltage <= 1.2)
      digitalWrite(redLed, HIGH);

   delay(ledDelay);
   digitalWrite(redLed, LOW);
   digitalWrite(yellowLed, LOW);
   digitalWrite(greenLed, LOW);
}
```

Arduino
Getting Started With Cayenne

Project Description

This tutorial goes through all the steps to getting started with Arduino Cayenne. It's an easy way to enable a powerful IoT software for the Arduino. You should find it is a great way to quickly integrate sensors and more with your board. It will also drastically reduce the amount of coding you will need to do.

There are many ways you're able to connect the Arduino to the Cayenne platform. The easiest way is to use either a WiFi or ethernet shield. Other methods include using the serial/USB connection but can be tricky to get working.

You can use MyDevices Cayenne with a whole range of different Arduino devices. This range includes the Arduino Duo, Leonardo, Mega, Mini, Nano, Pro Micro, Pro Mini, Uno and the Yun. The range of devices is a pretty hefty list, and they all follow the same setup steps.

Equipment

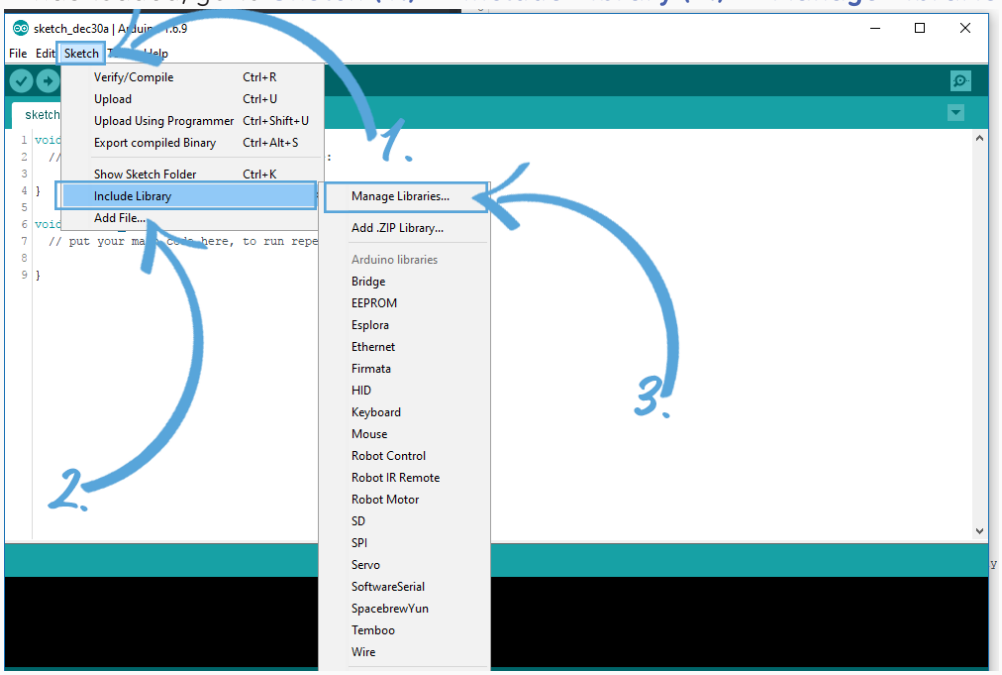
Required

- Arduino Uno
- Ethernet Shield, WiFi Shield or Similar
- Ethernet Cord (If using Ethernet Shield)
- 1 x DS18B20 Temperature Sensor
- 1 x 4.7k-ohm Resistor
- Breadboard
- Breadboard Wire

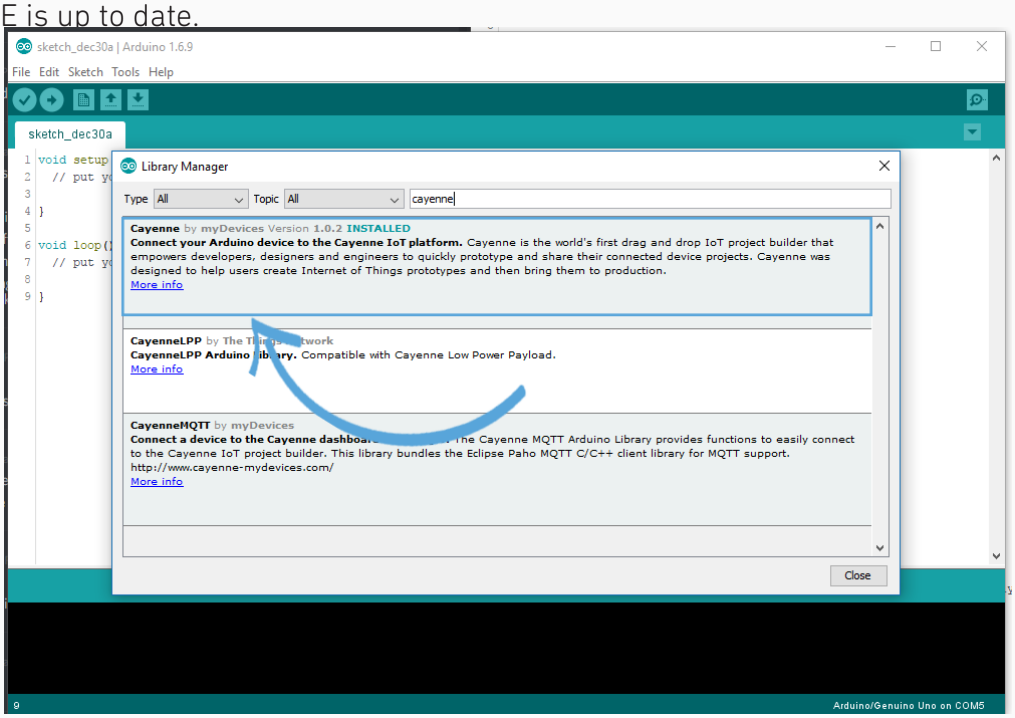
Setting up Cayenne for the Arduino

The process of setting up Cayenne for the Arduino is pretty straightforward. You will need internet access to the Arduino; this can be achieved by using an ethernet shield, WiFi shield, and some other methods.

1. Add an ethernet shield or your chosen connection method onto the Arduino.
2. Next, add an ethernet cord to the ethernet shield.
3. Now connect the Arduino to your PC using the USB Cable.
4. Open up the Arduino IDE software; if you don't have it installed, then you can grab it over at the official Arduino website.
5. Once the IDE has loaded, go to **Sketch (1.)** -> **Include Library (2.)** -> **Manage Libraries (3.)**



6. In this window, search for cayenne and install the Cayenne package. If it doesn't show make sure the Arduino IDE is up to date.



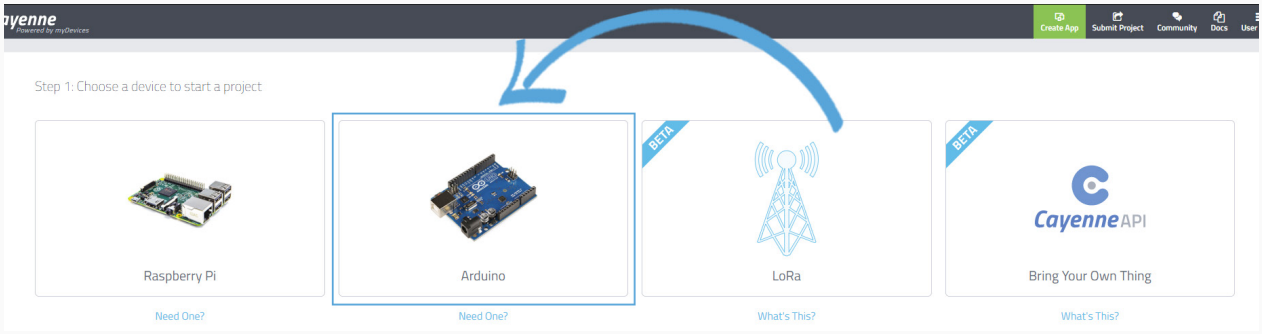
That's all we need to do before moving onto signing up and configuring Arduino Cayenne.

Signing up and Configuring Cayenne

1. Begin by heading to the myDevices Cayenne website and signup for a free account. You can find their website at: https://arduinomylifeup.com/out/cayennetutorial

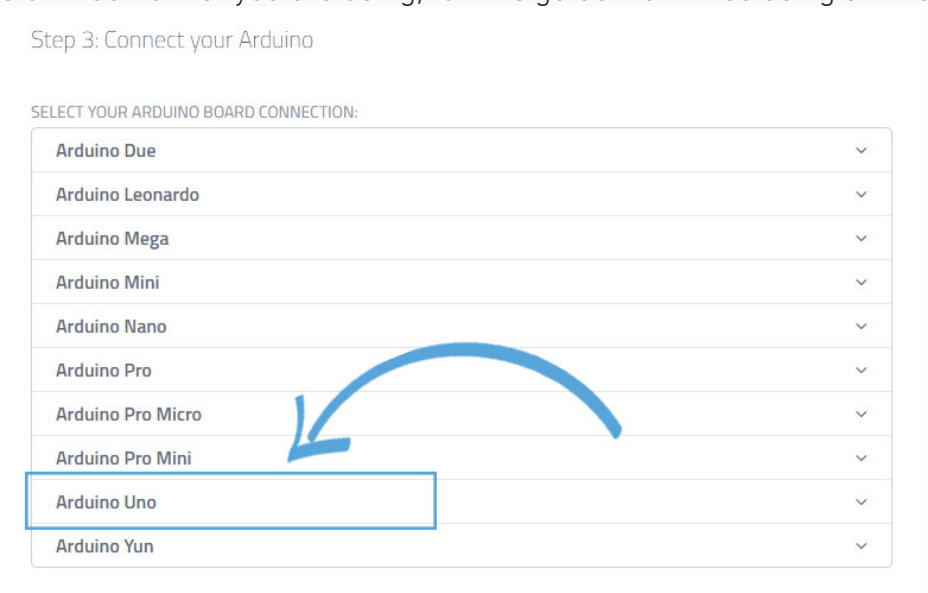
2. Once you are at the setup screen, you will need to connect the Arduino to the Cayenne platform now.

3. On this screen, we must select Arduino out of the list of devices that should be displayed.

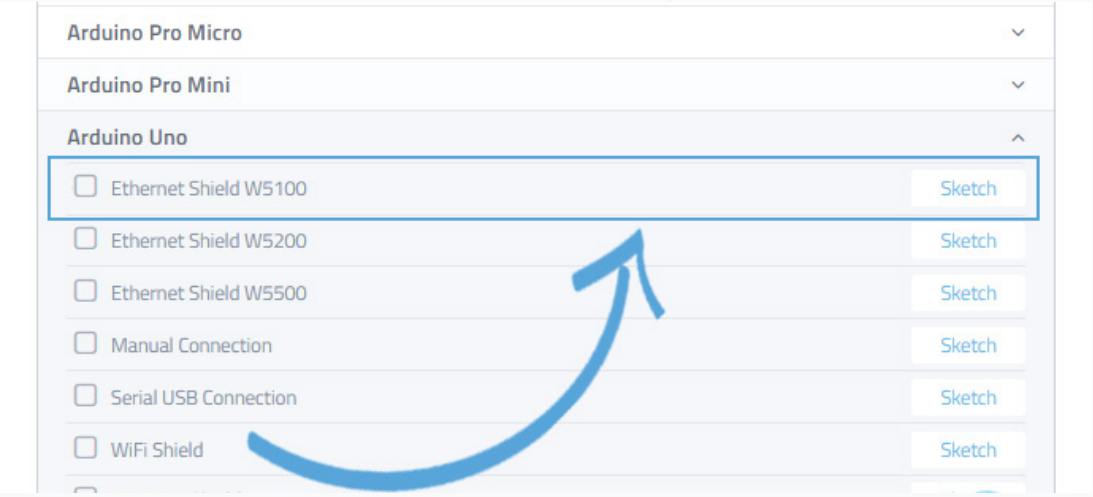


4. On the next screen simply click the "Next" button this is now displayed.

5. Select the type of Arduino that you are using, for this guide we will be using an Arduino Uno.



6. Now select the type of connection you're going to use. I highly recommend using an ethernet or WiFi shield as the others may be tricky to use. Some of the options will require a little bit more setting up. If you're unsure of the type of ethernet shield you're using, then select "**Ethernet Shield W5100**".

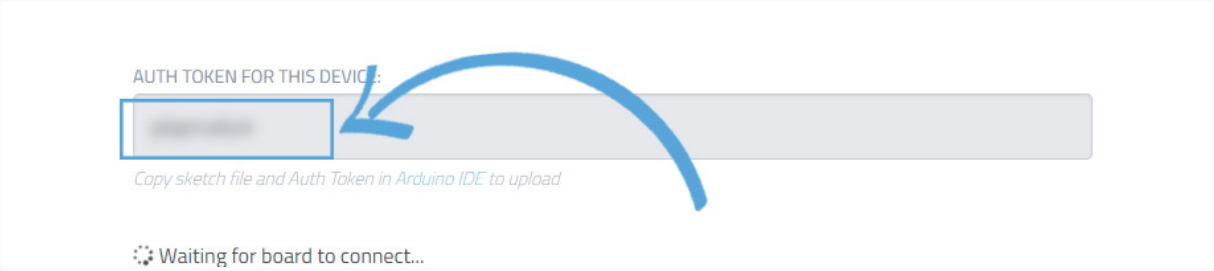


Signing up and Configuring Cayenne

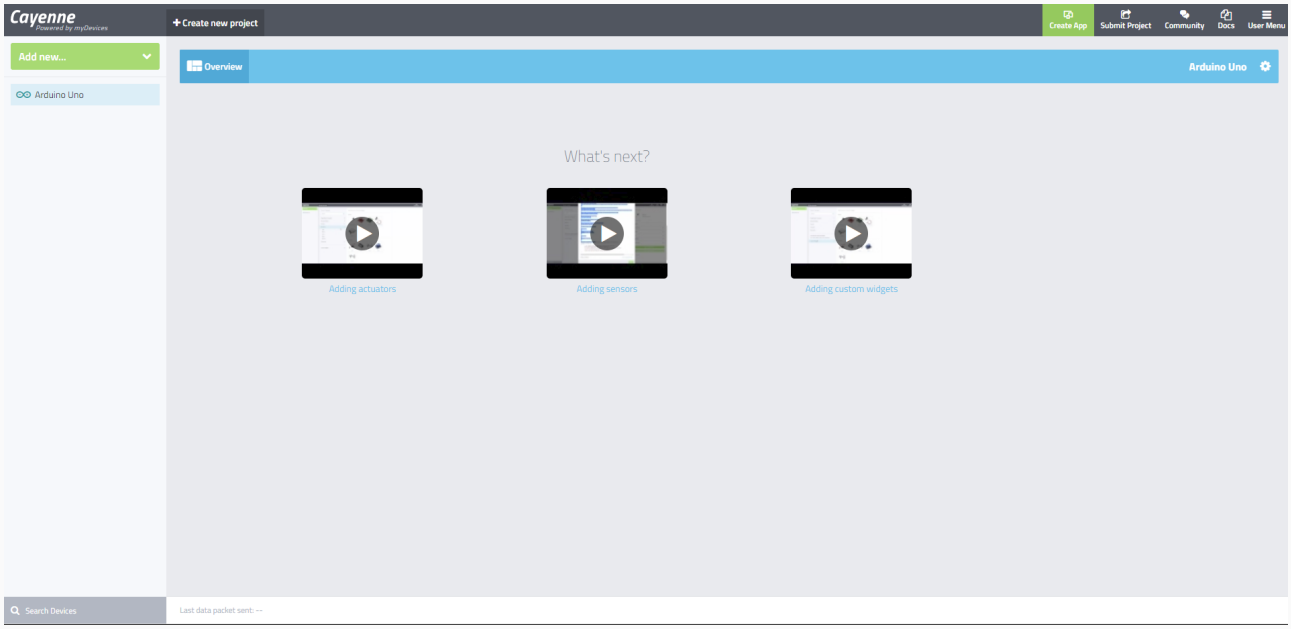
7. Now copy and paste the code shown in your browser into the Arduino IDE.

8. Once copied, upload it to the Arduino by pressing the "Upload" button, this may take a minute or two as it is a large amount of code.

9. The Arduino board should now be able to connect to Cayenne; this should appear not long after uploading it to the Arduino. If it takes longer than a few minutes, then there might be something wrong.

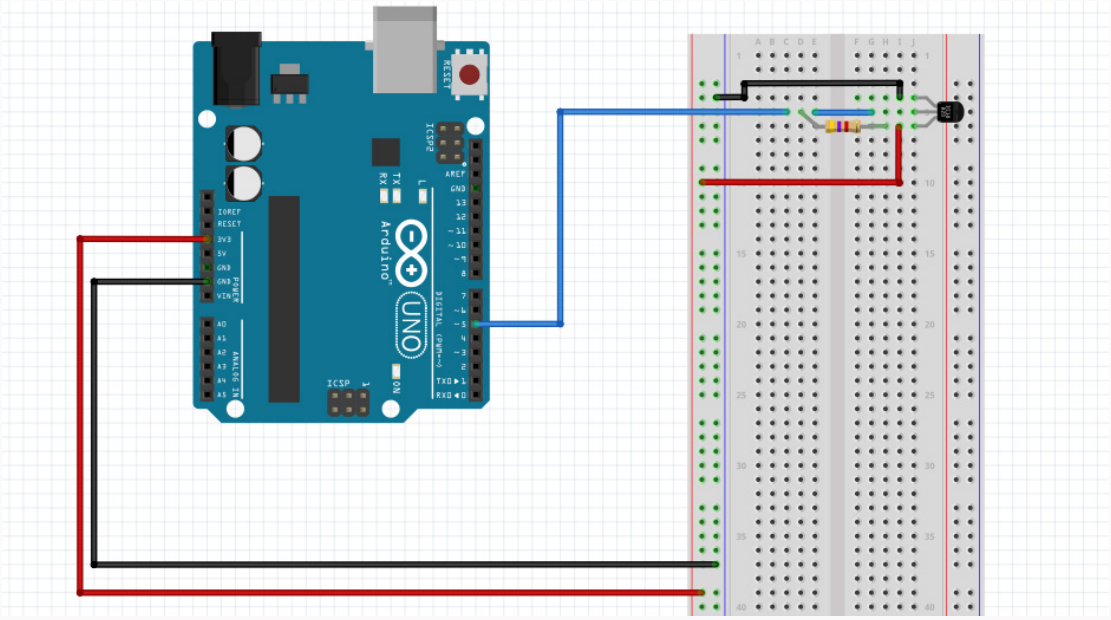


10. Once connected it will redirect you to the dashboard that should look something similar to the one shown below.

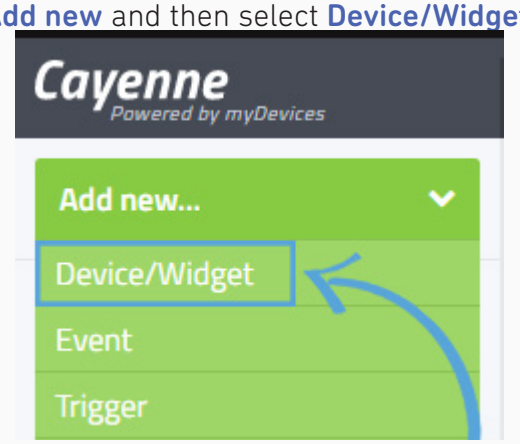


Setting up your first sensor with Cayenne

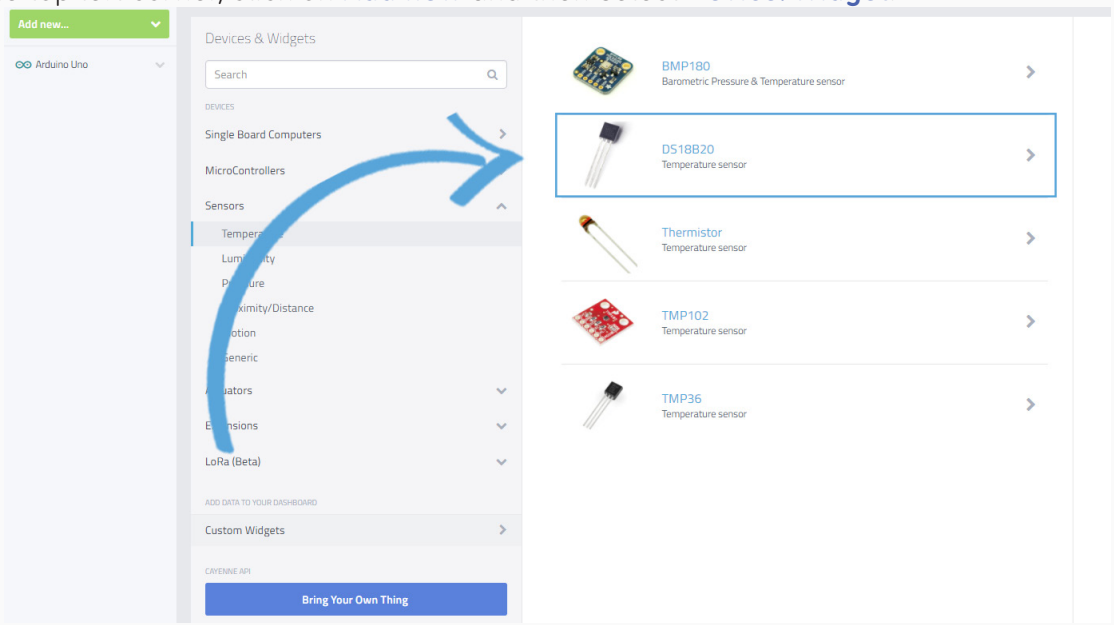
To quickly show you how to set up a sensor I will go through the steps of setting up a the DS18B20 temperature sensor. Firstly, set up the temperature sensor as shown in the diagram below. If you need more information, you can head over to the DS18B20 tutorial. It will take you through all the steps to set up this sensor.



1. Once your sensor is all wired up, go to the Cayenne dashboard.
2. In the **top left corner**, click on **Add new** and then select **Device/Widget**.



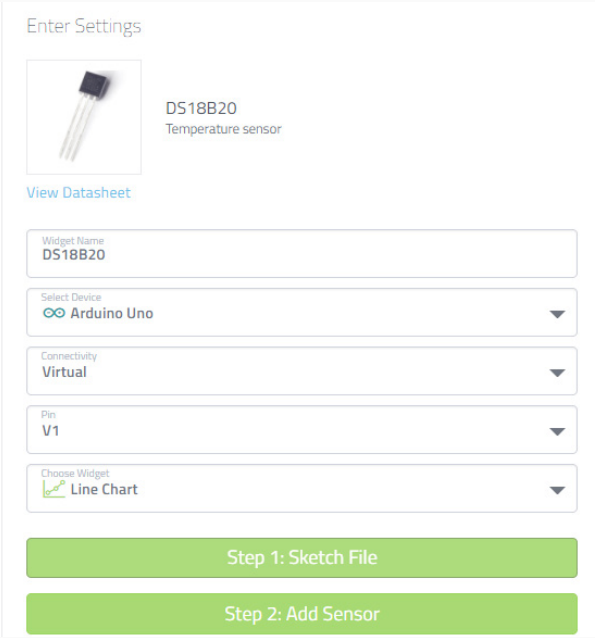
3. In the top left corner, click on **Add new** and then select **Device/Widget**.



Setting up your first sensor with Cayenne

4. Now enter the settings required for the **DS18B20 sensor** as shown below.

- **Widget name:** DS18B20
- **Select Device:** Arduino Uno
- **Pin:** V1
- **Choose Widget:** Line Chart



5. Now click on sketch file, this will give you all the code you need to add to your sketch file. You may need to change some values in this file. For example, if you set the **virtual pin** to something different or the pin of your temperature sensor is different to **2** (**tmpPin**), we use **pin 3** in the diagram above.

It's best to only copy the bits you need from the code into your sketch file.

```
#define CAYENNE_PRINT Serial  // Comment this out to disable prints and save space
#include <OneWire.h>
#include <DallasTemperature.h>

// If you're not using the Ethernet W5100 shield, change this to match your connection type. See Communications exam
#include <CayenneEthernet.h>

// Virtual Pin of the DS18B20 widget.
#define VIRTUAL_PIN V1

// Digital pin the DS18B20 is connected to. Do not use digital pins 0 or 1 since those conflict with the use of Seri
const int tmpPin = 2;

OneWire oneWire(tmpPin);
DallasTemperature sensors(&oneWire);

// Cayenne authentication token. This should be obtained from the Cayenne Dashboard.
char token[] = "[redacted]";

void setup()
{
        Serial.begin(9600);
        Cayenne.begin(token);
        sensors.begin();
}

void loop()
{
        Cayenne.run();
}

// This function is called when the Cayenne widget requests data for the Virtual Pin.
CAYENNE_OUT(VIRTUAL_PIN)
{
        // Send the command to get temperatures.
        sensors.requestTemperatures();
        // This command writes the temperature in Celsius to the Virtual Pin.
        Cayenne.celsiusWrite(VIRTUAL_PIN, sensors.getTempCByIndex(0));
        // To send the temperature in Fahrenheit use the corresponding code below.
        //Cayenne.fahrenheitWrite(VIRTUAL_PIN, sensors.getTempFByIndex(0));
}
```

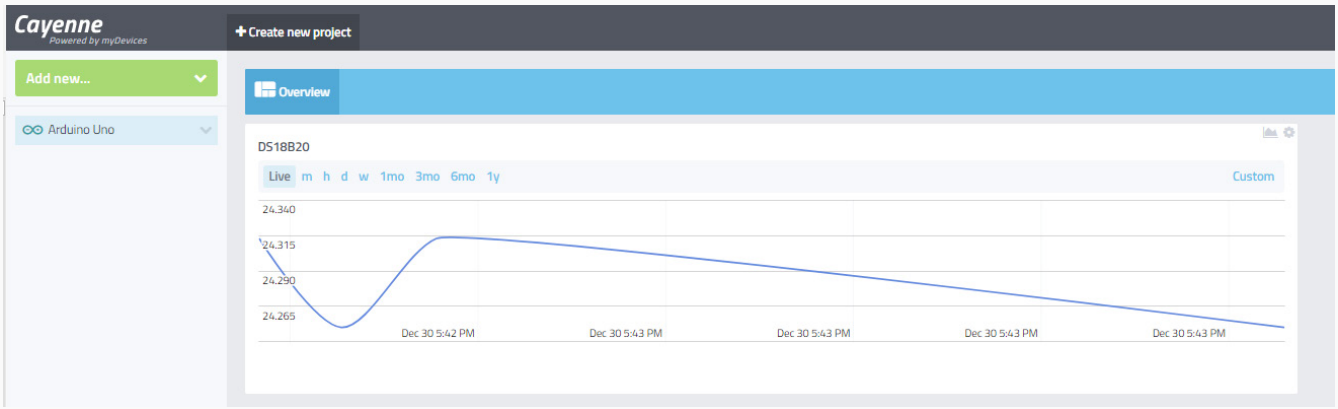
Setting up your first sensor with Cayenne

6. Secondly, to get the temperature sensor working you will need to install the **onewire library** and the **DallasTemperature** library. If you followed my DS18B20 tutorial, then these should be already installed. If you don't have them installed, then the cayenne code has instructions on how to install the required packages.

```
/*
Cayenne DS18B20 Example

This sketch shows how to send temperature data to a DS18B20 Sensor in the Cayenne Dashboard.

The Cayenne Library is required to run this sketch. If you have not already done so you can install it from the Ardu

Steps:
1. Install the OneWire library (http://www.pjrc.com/teensy/td_libs_OneWire.html) from the Arduino Library Manager.
2. Install the DallasTemperature library (http://milesburton.com/Main_Page?title=Dallas_Temperature_Control_Library#
3. In the Cayenne Dashboard add a new DS18B20 widget.
4. Set the widget to Value Display.
5. Select Virtual Pins and a virtual pin number.
6. Set VIRTUAL_PIN to the pin number you selected.
7. Attach a DS18B20 to an digital pin on your Arduino.
   Schematic:
   [Ground] -- [DS18B20] -- [4.7k resistor] -- [5V]
                  |_____________|
                  |
             Digital Pin
8. Set the tmpPin variable to match the pin used to connect the DS18B20.
9. Set the token variable to match the Arduino token from the Dashboard.
10. Compile and upload this sketch.
11. Once the Arduino connects to the Dashboard it should automatically update the DS18B20 widget with data.
*/
```

7. Once you have copied over the sketch code and uploaded it to the Arduino, select **Add Sensor**.

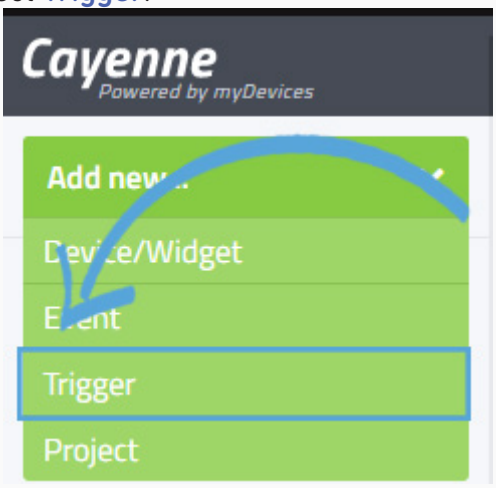
8. It should now be displayed on the dashboard. It may take a few minutes for data to start displaying, if it takes any longer than there is likely a setting wrong somewhere.



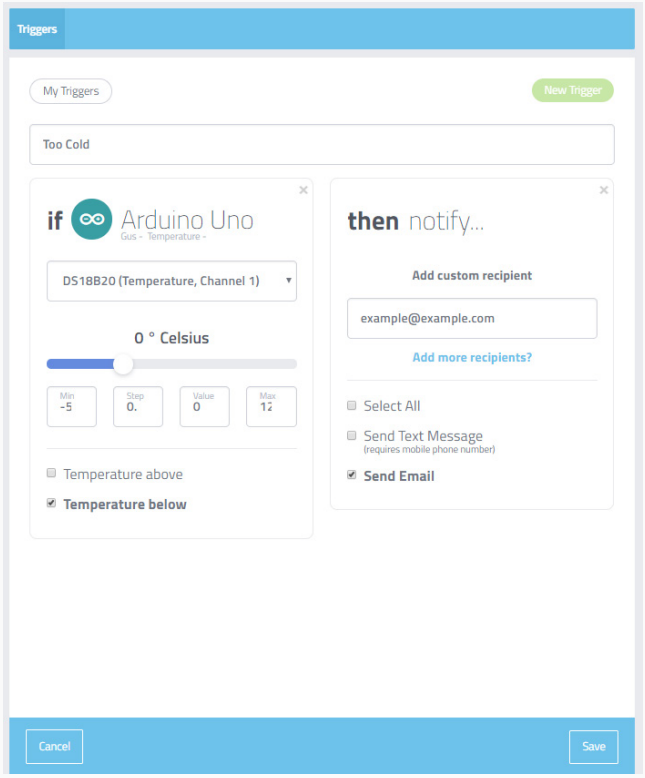
To add more sensors, you basically follow the same process but don't simply copy and paste the code. You will need to only copy the bits you need to integrate that sensor correctly. Copying and pasting all the code will override your previous set devices.

Setting up Cayenne Triggers

1. First, go to **Add New** then select **Trigger**.



2. Selecting "Trigger" will pop up with a screen to create your own trigger.
3. Next, add a name for your trigger. I'm going to call mine **Too cold**.
4. Now, drag the **Arduino Uno** from the **left column** into the **if**.
5. Select the **DS18B20** from the **dropdown box** called **Select Trigger**.
6. Set the degrees to **0°c** and select **temperature below**.
7. Once that is done, you can proceed to set up the then statement. You can either drag a device or set up a notification. For example, if it drops below 0, you can have Cayenne switch another device on, something like a heater. I'm going to simply set a notification, and have it send an email to me.



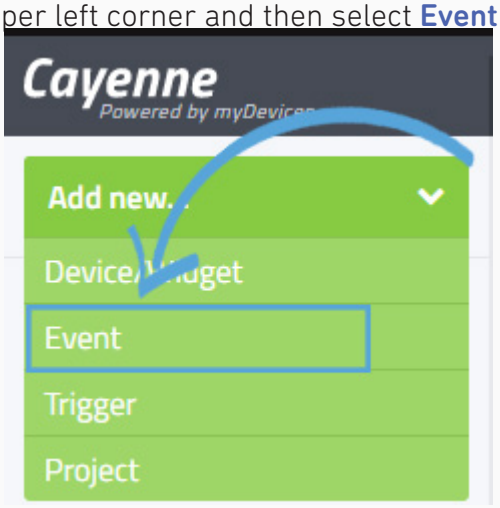
8. Once you have finished with it simply hit the "**Save**" button.
9. You can view the triggers you ahve set up by selecting "**My Triggers**".

That's a basic overview of triggers, as you can see these can be extremely powerful with setting up automation or alerts.

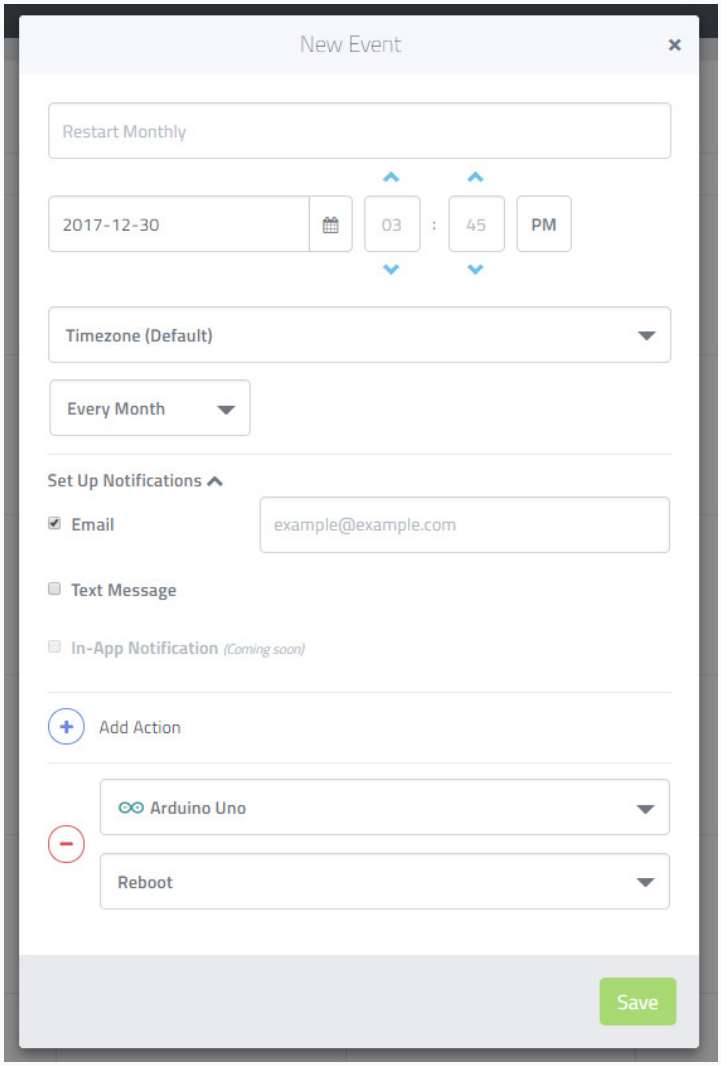
Setting up Cayenne Events

Events within cayenne are very similar to trigger but depend on time as the triggers rather than changes in sensors and other devices. These are super easy to setup and perfect if you want to make sure a certain task is performed at your chosen time.

1. First, go to **Add New** in the upper left corner and then select **Event** from the drop-down box.



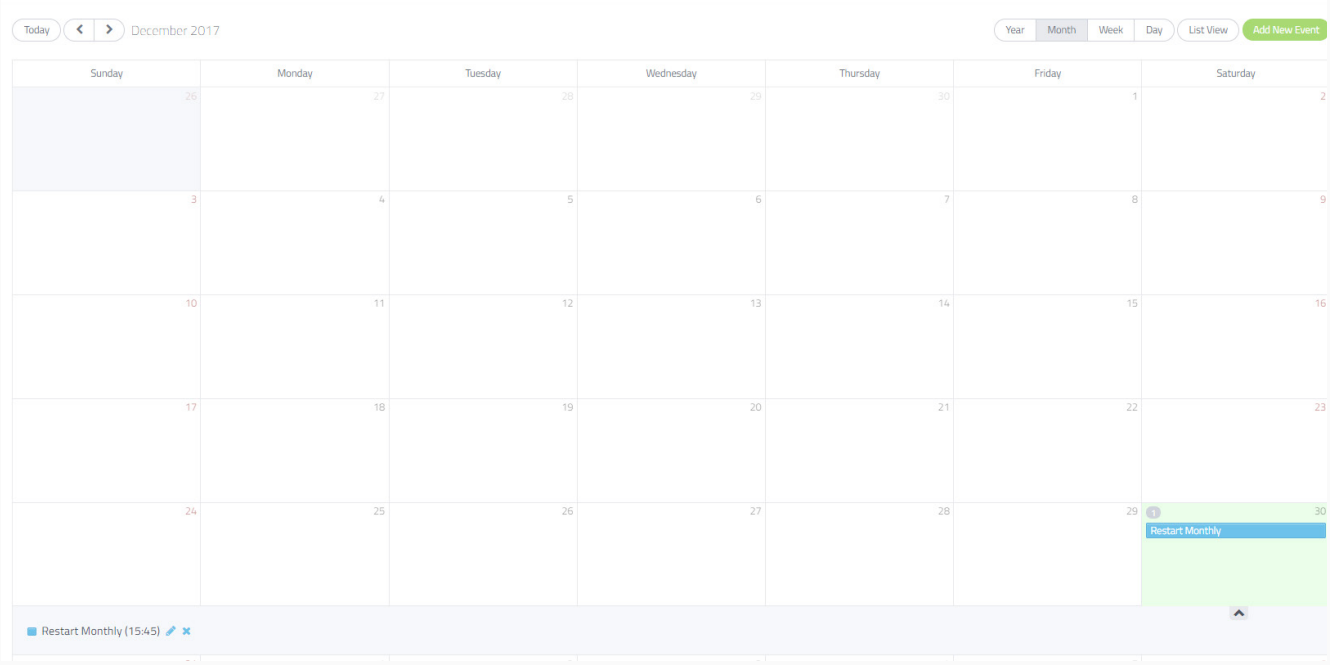
2. This will load a screen with a calendar and the new event screen should popup.
3. The settings on this screen are pretty straight forward, as you can see below.



Setting up Cayenne Events

4. I set the **name** to be restart monthly. The **date** to be 30/12/2017, and the **time** to be 3:45PM. I have set it to **repeat monthly** and for email alerts to be sent to **example@example.com**. Lastly it performs an action to **reboot the Arduino**.

5. Once saved, you can see the event in the calendar.



Basic Arduino Web Server

Project Description

This tutorial goes through the steps to making your own Arduino web server. It's a straightforward process that covers the process of getting it up and running. I will also cover some core concepts such as using ajax to update rather than refreshing the page over and over.

You will need to know some basic HTML which is incredibly easy to grasp the concepts of. If you have never done any HTML then I recommend hitting up w3schools for a basic introduction.

This server won't be able to do anything too fancy but it's incredibly useful for anyone who wants to display data from devices connected to the pins or would like to interact with a circuit from a webpage. The more powerful Raspberry Pi web server might interest you if this doesn't really cover what you're after.

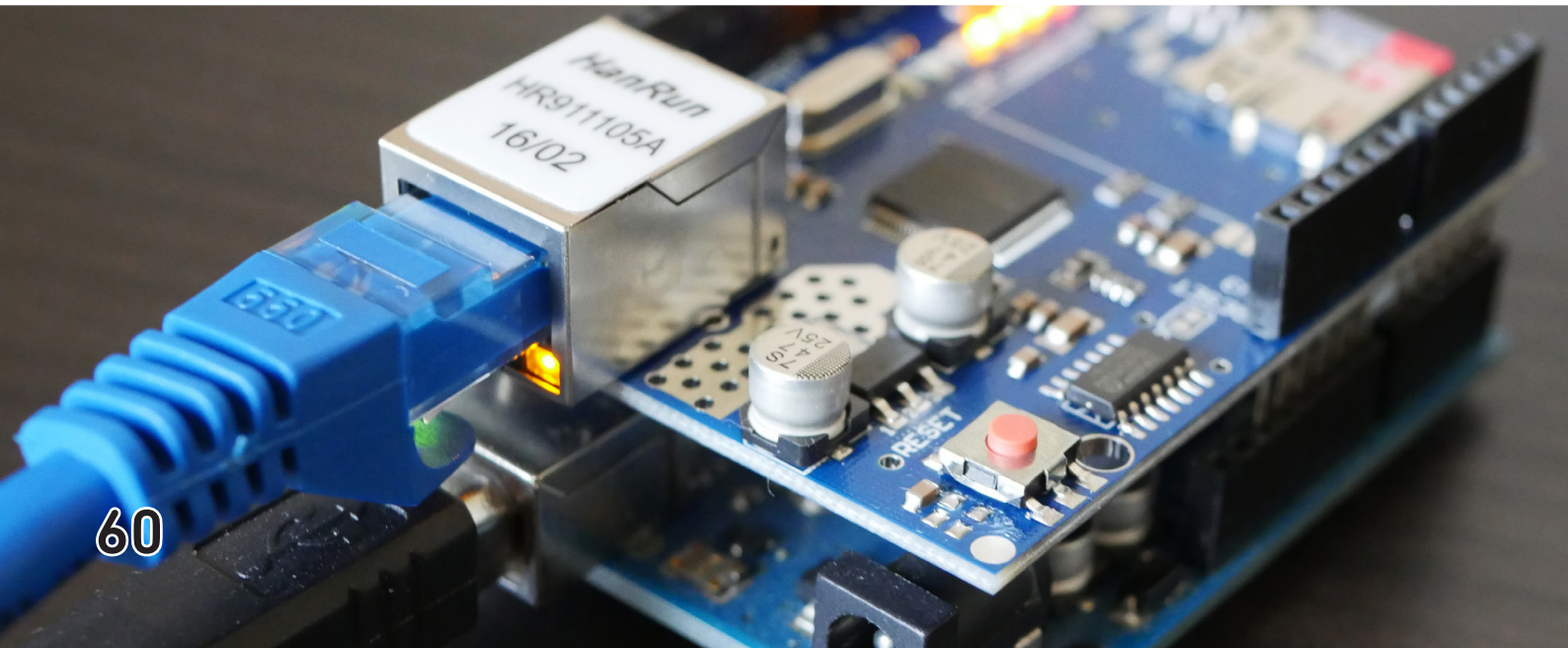
Equipment

Required

- Arduino Uno
- Ethernet Shield or similar
- Ethernet cord

Optional

- Micro SD Card



Initialisation for the Arduino Web Server

To begin you will first need initialise the web server by defining a few variables then calling a function.

Firstly, you need to define a mac address the one below should be fine on most home networks. If you know exactly what you're after then don't hesitate to change it.

```
byte mac[] = {
  0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED
};
```

Secondly, you will need to define an IP address; this is done by simply calling the IPAddress class with your chosen IP. Make sure you update this to an IP address that won't conflict on your network.

```
IPAddress ip(192, 168, 1, 177);
```

Thirdly, you will need to set the port number that you would like the server to listen on. By default, plain http runs over port 80. If you change this then you will need to define the port when you go to access the web page in your browser. For example if your port was **14213** then the address will look like this: **192.168.1.177:14213**

```
EthernetServer server(80);
```

Lastly in the setup function you will need to initialize the Ethernet device. Once this is initialised simply make a call to server.begin(), once this is done you should now be able to listen for connections and respond with data when appropriate.

```
Ethernet.begin(mac, ip);
 server.begin();
```

Detecting a new connection for the Arduino Web Server

For a browser to display the webpage correctly, we first need to reply with an HTML response header. It doesn't need to be anything complicated; the example below is more than enough to get things working correctly. If you're after more information on the header fields, then this incredibly handy wiki page explains everything pretty well.

```
client.println("HTTP/1.1 200 OK");
client.println("Content-Type: text/html");
client.println("Connection: close");
client.println();
```

Outputting HTML from the Arduino Web Server

Outputting HTML can simply be done by calling client.println() or client.print() with the text/html passed as a parameter. This is straightforward but as you can see in the examples further down this page it really does bloat out the code.

Below is a basic example of outputting some HTML.

```
client.println("<!DOCTYPE HTML>");
client.println("<html>");
client.print("<h1>Analogue Values</h1>");
```

The other option is to have the HTML files stored on an SD card which you're able to access and load from. You can then perform AJAX requests to update elements on the web page, so they're showing the correct data. You can also do this to get the Arduino to perform actions such as turn an LED on and off.

Closing the connection from the Arduino Web Server

Once you're done processing the web page, you should delay for a short amount of time to ensure the client gets the data. After a second or so is up simply close the connection to the client.

```
    delay(1);
      client.stop();
```

Coding your Arduino Web Server

If you're not using the SD card, then it's incredibly straightforward to get a web server up and running.

It's important to note that if you do have a SD card inserted but it's not in use then it can cause issues with sketch communicating with the Arduino. To avoid this from happening add the following two lines in the setup function.

```
    pinMode(4, OUTPUT);
    digitalWrite(4, HIGH);
```

To quickly get an Arduino web server up and going simply open sketch and copy and paste the code below. You will recognise several parts of this code as we talked about it earlier in this tutorial. Please note that due to the sheer size of the code it had to be split onto the next page, so make sure you copy that as well.

```
#include <SPI.h>
#include <Ethernet.h>

byte mac[] = {
  0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED
};
IPAddress ip(192, 168, 1, 177);

EthernetServer server(80);

void setup() {
  Serial.begin(9600);
  while (!Serial) {
    ; // wait for serial port to connect. Needed for native USB port only
  }

  Ethernet.begin(mac, ip);
  server.begin();
  Serial.print("server is at ");
  Serial.println(Ethernet.localIP());
}

void loop() {
  // listen for incoming clients
  EthernetClient client = server.available();
  if (client) {
    Serial.println("new client");
    // an http request ends with a blank line
    boolean currentLineIsBlank = true;
```

Coding your Arduino Web Server

```
        while (client.connected()) {
         if (client.available()) {
           char c = client.read();
           Serial.write(c);
           if (c == '\n' && currentLineIsBlank) {
            client.println("HTTP/1.1 200 OK");
            client.println("Content-Type: text/html");
            client.println("Connection: close");
            client.println("Refresh: 5");
            client.println();
            client.println("<!DOCTYPE HTML>");
            client.println("<html>");
            client.print("<h1>Analogue Values</h1>");
            for (int analogChannel = 0; analogChannel < 6; analogChannel++) {
              int sensorReading = analogRead(analogChannel);
              client.print("analog input ");
              client.print(analogChannel);
              client.print(" is ");
              client.print(sensorReading);
              client.println("<br />");
            }
            client.println("</html>");
            break;
           }
           if (c == '\n') {
            currentLineIsBlank = true;
           } else if (c != '\r') {
            currentLineIsBlank = false;
           }
         }
        }
        delay(1);
        // close the connection:
        client.stop();
        Serial.println("client disconnected");
      }
    }
```

Once you have uploaded this code to the Arduino, you should have a very basic web server up and going. It will refresh every 5 seconds updating the values of the Arduino's analog pins (This will just be random values unless you have an actual device/sensor connected to them). You can also monitor the serial monitor for any debug lines that are located throughout the code.

Coding your Arduino Web Server - Adding SD Card Support

If you decide to go the SD card route for the Arduino web server, then the HTML files will need to be created on your computer and then copied to the SD card before it is inserted into the Arduino.

A pro with loading the webpage from the SD card is that you could have more complex/heavy pages without needing to write hundreds of "**client.write**" lines. It also helps prevent any low memory situations that usually occur when you end up having too much code running on the Arduino.

Initialising the SD Card

Before you are able to access the SD card, you will need to import the SD package and initialize it in the **setup** function. The check in the setup function will see if it can access the card by utilizing the **SD.begin()** function. otherwise it will throw an error to the serial monitor.

Insert the following code to initialise your SD Card on setup.

Find

```
Ethernet.begin(mac, ip);
server.begin();
Serial.print("server is at ");
Serial.println(Ethernet.localIP());
```

Add above

```
Serial.println("Checking SD card is accessible...");
if (!SD.begin(4)) {
  Serial.println("ERROR - SD card initialization failed!");
  return;    // init failed
}
Serial.println("SUCCESS - SD card initialized.");
```

Loading the html file from the sd card

When you are ready to load the file simply use **sd.open(filename)** to open the file. Use an **if statement** to make sure the file exists and then loop through the contents of the file until it reaches the end. The code we use below simply opens up a file named index.html and outputs the contents of the file.

Find

```
client.println("<!DOCTYPE HTML>");
client.println("<html>");
client.print("<h1>Analogue Values</h1>");
for (int analogChannel = 0; analogChannel < 6; analogChannel++) {
  int sensorReading = analogRead(analogChannel);
  client.print("analog input ");
  client.print(analogChannel);
  client.print(" is ");
  client.print(sensorReading);
  client.println("<br />");
}
```

Replace With

```
webPage = SD.open("index.htm"); // open web page file
if (webPage) {
   while (webPage.available()) {
   client.write(webPage.read()); // send web page to client
}
webPage.close();
```

Final version of the Arduino Web Server SD Card code

Below is the final version of the Arduino Web Server SD Card code, you can check your changes against this or move onto our next section on handling AJAX requests.

```
#include <SPI.h>
#include <Ethernet.h>

byte mac[] = {
  0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED
};
IPAddress ip(192, 168, 1, 177);

EthernetServer server(80);

void setup() {
  Serial.begin(9600);
  while (!Serial) {
    ; // wait for serial port to connect. Needed for native USB port only
  }

 Serial.println("Checking SD card is accessible...");
  if (!SD.begin(4)) {
    Serial.println("ERROR - SD card initialization failed!");
    return;    // init failed
  }
  Serial.println("SUCCESS - SD card initialized.");

  Ethernet.begin(mac, ip);
  server.begin();
  Serial.print("server is at ");
  Serial.println(Ethernet.localIP());
}

void loop() {
  // listen for incoming clients
  EthernetClient client = server.available();
  if (client) {
    Serial.println("new client");
    // an http request ends with a blank line
    boolean currentLineIsBlank = true;
    while (client.connected()) {
     if (client.available()) {
        char c = client.read();
        Serial.write(c);
        if (c == '\n' && currentLineIsBlank) {
          client.println("HTTP/1.1 200 OK");
          client.println("Content-Type: text/html");
          client.println("Connection: close");
          client.println("Refresh: 5");
          client.println();
```

```
          webPage = SD.open("index.htm"); // open web page file
          if (webPage) {
             while (webPage.available()) {
             client.write(webPage.read()); // send web page to client
          }
          webPage.close();
          break;
        }
        if (c == '\n') {
          currentLineIsBlank = true;
        } else if (c != '\r') {
          currentLineIsBlank = false;
        }
      }
    }
    delay(1);
    // close the connection:
    client.stop();
    Serial.println("client disconnected");
  }
}
```

Coding your Arduino Web Server - Adding AJAX Support

There may be a time when you want to control a device or sensor via the web page. In this section, I will look at performing requests and updates using AJAX. This means you won't see the page load over and over, but it will still be kept updated when required.

AJAX or Asynchronous JavaScript and XML is a little too complex to cover completely in this tutorial, but I will go through the basics of using it on the Arduino. You should be able to adjust the examples to suit your needs without too many issues.

Ajax script example

AJAX works simply by sending a request to the server; the server will then check to see if the string "**ajaxrefresh**" or "**ledstatus**" exists in the header (For this example). If it does then, it will generate the relevant data and return it. The JavaScript then finds the relevant HTML element with the correct id then replaces the **innerHTML** with the response from the **AJAX request**.

The example below sends a request to the web server (Arduino) with the GET variable **ajaxrefresh**. If the server returns data, it will replace the **innerHTML** of the element that has the id **analogue_data**. This script runs every **5 seconds** but can be adjusted to better suit your needs.

If you are using the SD Card example then you should add this following text block into your index.html file.

```
<script>window.setInterval(function(){
 nocache = "&nocache=" + Math.random() * 10;
 var request = new XMLHttpRequest();
 request.onreadystatechange = function() {
  if (this.readyState == 4) {
   if (this.status == 200) {
    if (this.responseText != null) {
     document.getElementById("analoge_data").innerHTML = this.responseText;
    }
   }
  }
 }
 request.open("GET", "ajaxrefresh" + nocache, true);
 request.send(null);
 }, 5000);
</script>
```

Of course if you are running the non SD card version of this script you will have to include all of that code into the loop, printing it out line by line. This means be prepared to add a ton more lines into your code shortly.

But before we do that we will be going over the two functions that we need to write to allow the AJAX scripts to actually interact with the Arduino and return relevant data.

Arduino Ajax Functions

The two functions below are called whenever the relevant AJAX request comes through to the server. They're very simple with the first one reading the analog pins and returning all the relevant data. It does this by simply looping through all 6 possible channels and returning the values.

The second function will turn the RED LED on or off depending on its current status. It does this by reading the current state utilizing digitalRead then using digitalWrite to set the LED status to the opposite. It will also return the status of the LED so that AJAX can update the value on the webpage.

You should add both of those functions to the bottom of your current script.

```
void ajaxRequest(EthernetClient client)
{
  for (int analogChannel = 0; analogChannel < 6; analogChannel++) {
    int sensorReading = analogRead(analogChannel);
    client.print("analog input ");
    client.print(analogChannel);
    client.print(" is ");
    client.print(sensorReading);
    client.println("<br />");
  }
}

void ledChangeStatus(EthernetClient client)
{
  int state = digitalRead(RED);
  Serial.println(state);
  if (state == 1) {
    digitalWrite(RED, LOW);
    client.print("OFF");
  }
  else {
    digitalWrite(RED, HIGH);
    client.print("ON");
  }
}
```

With those two functions added to your script you can now proceed onto modifying the loop. The changes of course vary whether you are utilizing the SD card example or the non SD card example. If you are using the non SD Card example then please be prepared to have to add a fair few lines.

If you are using the SD Card example then skip past the next couple of pages as there is a considerable amount of code that is required to be added.

AJAX with HTML files on SD Card

This is the most simple way to add ajax support since you can easily add more HTML by just modifying your **index.html** file.

Lets begin by adding the following couple of lines of code, this basically ensures that we capture the incoming HTTP request. Without capturing it we wont be able to see what the AJAX request is for.

Find

```
while (client.connected()) {
  if (client.available()) {
    char c = client.read();
```

Add Below

```
if ( HTTP_req.length() < 80)
      HTTP_req += c;
```

Now we need to add the extra bit of code that checks what is in the http request and fires the correct functionn for that request. We do this by simply using **indexOf** to see whether a certain text exists within the request that we stored.

If index of returns anything but 0 then we run the relevent function and break out of the if statement.

Find

```
if (c == '\n' && currentLineIsBlank) {
  client.println("HTTP/1.1 200 OK");
  client.println("Content-Type: text/html");
  client.println("Connection: close");
  client.println();
```

Add Below

```
if (HTTP_req.indexOf("ajaxrefresh") >= 0 ) {
  ajaxRequest(client);
  break;
}
else if (HTTP_req.indexOf("ledstatus") >= 0 ) {
  ledChangeStatus(client);
  break;
}
else {
```

Find

```
if (c == '\n') {
  currentLineIsBlank = true;
} else if (c != '\r') {
  currentLineIsBlank = false;
}
```

Add Below

```
}
```

With these additions your code should now be ready to go. Go ahead and push this to the Arduino and try it out. In fact that is the end of the tutorial if you are just using the SD Card example. You can view what the final version of the code should look like by going over the next two pages.

Arduino Ajax Loop changes - SD Card example

```
#include <SPI.h>
#include <Ethernet.h>

byte mac[] = {
  0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED
};
IPAddress ip(192, 168, 1, 177);

EthernetServer server(80);

void setup() {
  Serial.begin(9600);
  while (!Serial) {
    ; // wait for serial port to connect. Needed for native USB port only
  }

 Serial.println("Checking SD card is accessible...");
  if (!SD.begin(4)) {
    Serial.println("ERROR - SD card initialization failed!");
    return;    // init failed
  }
  Serial.println("SUCCESS - SD card initialized.");

  Ethernet.begin(mac, ip);
  server.begin();
  Serial.print("server is at ");
  Serial.println(Ethernet.localIP());
}

void loop() {
  // listen for incoming clients
  EthernetClient client = server.available();
  if (client) {
    Serial.println("new client");
    // an http request ends with a blank line
    boolean currentLineIsBlank = true;
    while (client.connected()) {
     if (client.available()) {
        char c = client.read();
        if ( HTTP_req.length() < 80)
          HTTP_req += c;
        Serial.write(c);
        if (c == '\n' && currentLineIsBlank) {
          client.println("HTTP/1.1 200 OK");
          client.println("Content-Type: text/html");
          client.println("Connection: close");
```

Arduino Ajax Loop changes - SD Card example

```
          client.println("Refresh: 5");
          client.println();
         if (HTTP_req.indexOf("ajaxrefresh") >= 0 ) {
           ajaxRequest(client);
           break;
         }
         else if (HTTP_req.indexOf("ledstatus") >= 0 ) {
           ledChangeStatus(client);
           break;
         }
         else {
           webPage = SD.open("index.htm"); // open web page file
           if (webPage) {
               while (webPage.available()) {
               client.write(webPage.read()); // send web page to client
           }
           webPage.close();
           break;
         }
         if (c == '\n') {
           currentLineIsBlank = true;
         } else if (c != '\r') {
           currentLineIsBlank = false;
         }

       }
     }
     delay(1);
     // close the connection:
     client.stop();
     Serial.println("client disconnected");
   }
 }
```

Arduino Ajax Loop changes - Non SD Card example

The example below has all the HTML & JavaScript completely generated by the Arduino. The JavaScript makes AJAX requests to the Arduino server which then updates the page depending on the results it provides. Just copy the code below to replace your loop function. If you want an explanation of some of the additions check out the example for the SD Card.

One issue I noticed by generating all the code on the Arduino is that it started to run low on space and give stability warnings. This could be an issue if you want tons of sensors and options on the page. You might want to look at storing the HTML on an SD Card and grabbing it from there instead.

```
void loop() {
  EthernetClient client = server.available();
  if (client) {
    Serial.println("new client");
    boolean currentLineIsBlank = true;
    while (client.connected()) {
      if (client.available()) {
        char c = client.read();
        if( HTTP_req.length() < 120)
          HTTP_req += c;  // save the HTTP request 1 char at a time
        Serial.write(c);
        if (c == '\n' && currentLineIsBlank) {
          // send a standard http response header
          client.println("HTTP/1.1 200 OK");
          client.println("Content-Type: text/html");
          client.println("Connection: close");
          client.println();
          Serial.println(HTTP_req);
          if (HTTP_req.indexOf("ajaxrefresh") >= 0 ) {
            // read switch state and analog input
            ajaxRequest(client);
            break;
          }
          else if (HTTP_req.indexOf("ledstatus") >= 0 ) {
            // read switch state and analog input
            ledChangeStatus(client);
            break;
          }
          else {
            client.println("<!DOCTYPE HTML>");
            client.println("<html lang=\"en\">");
            client.println("<script>window.setInterval(function(){");
            client.println("nocache = \"&nocache=\" + Math.random() * 10;");
            client.println("var request = new XMLHttpRequest();");
            client.println("request.onreadystatechange = function() {");
            client.println("if (this.readyState == 4) {");
            client.println("if (this.status == 200) {");
            client.println("if (this.responseText != null) {");
            client.println("document.getElementById(\"analoge_data\").innerHTML = this.responseText;");
            client.println("}}}}");
            client.println(
              "request.open(\"GET\", \"ajaxrefresh\" + nocache, true);");
            client.println("request.send(null);");
            client.println("}, 5000);");
            client.println("function changeLEDStatus() {");
            client.println(
              "nocache = \"&nocache=\" + Math.random() * 10;");
```

Arduino Ajax Loop changes - Non SD Card example

```
          client.println("var request = new XMLHttpRequest();");
          client.println("request.onreadystatechange = function() {");
          client.println("if (this.readyState == 4) {");
          client.println("if (this.status == 200) {");
          client.println("if (this.responseText != null) {");
          client.println("document.getElementById(\"led_status\").innerHTML = this.responseText;");
          client.println("}}}}");
          client.println("request.open(\"GET\", \"?ledstatus=1\" + nocache, true);");
          client.println("request.send(null);");
          client.println("}");
          client.println("</script></head>");
          // output the value of each analog input pin
          client.print("<h1>Analogue Values</h1>");
          client.println("<div id=\"analoge_data\">Arduino analog input values loading.....</div>");
          client.println("<h1>Arduino LED Status</h1>");
          client.println("<div><span id=\"led_status\">");
          if(digitalRead(RED) == 1)
           client.println("On");
          else
            client.println("Off");
          client.println("</span> | <button onclick=\"changeLEDStatus()\">Change Status</button> </div>");
          client.println("</html>");
          break;
        }
      }
      if (c == '\n') {
        // you're starting a new line
        currentLineIsBlank = true;
      } else if (c != '\r') {
        // you've gotten a character on the current line
        currentLineIsBlank = false;
      }
    }
  }
  delay(1);
  client.stop();
  HTTP_req = "";
  Serial.println("client disconnected");
 }
}
```