Connected Community HackerSpace

Arduino Beginners Guide

February 24, 2015

Contents

1 Getting Started **2**

1.1 What is an Arduino? 2

1.1.1 Uploading 2

1.1.2 The Blink Sketch 3

1.2 Digital Output 4

1.2.1 Pin as Source 4

1.2.2 Pin as Sink 5

1.2.3 Software 6

1.2.4 Hardware 6

1.3 Digital Input 8

1.3.1 Pull-up Resistor 9

1.3.2 Pull-down Resistor 10

1.3.3 Tristate Logic 10

1.3.4 Software 10

1.3.5 Internal Pulldown 10

1.3.6 Hardware 11

1.4 Analogue Output 11

1.4.1 Hardware 12

1.4.2 Pulse Width Modulation 12

1.4.3 Software 12

1.5 Analogue Input 12

1.5.1 Hardware 13

1.5.2 Software 13

A All the Physics You Will Need **14**

B Circuit Symbols **15**

C Source Code **16**

Chapter 1

Getting Started

1.1 What is an Arduino?

At its simplest an Arduino is a tiny computer. Unlike personal computers which hide their internals to prevent them from being damanged Arduinos expose all their wiring to the outside world. While your laptop can only use USB to communicate with outside devices an Arduino can send and receive binary voltage signals. This ability makes an Arduino an ideal platform for physical programming. For example in the CCHS workshop Arduinos are used to control the small C&C mill, the laser cutter and all the 3D printers you see.

At a more technical level the Arduino is an 8bit **micro controller** placed on a printed circuit board which safely exposes all the pins of the chip. There is already a primitive operating system on the chip which any software you write will interface with. This makes writing software much easier as you only need to explicitly control the hardware which you want to, for example writing and uploading your first program to blink an LED will take about 5 minutes, doing the same for a naked micro controller would take about 6 weeks in an introductory university course.

The real strength of the Arduino comes from using **shields**, these are custom designed modular circuit boards which fit on top of the Arduino and each other. They range complexity from safe high current switches all the way up to working gsm modules which allow the Arduino to be controlled by sms. Each comes with custom software which allows them to “just work” with the Arduino.

You won’t be using shields in this tutorial since each shield has it’s own quirks and gotchas that only distract you from learning the basics of the Arduino that are universal across all projects.

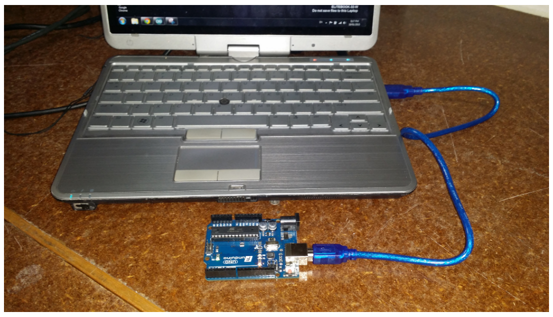
1.1.1 Uploading

Getting the Arduino to work with your personal computer takes some effort and is different for every computer. The CCHS Arduinos are already set up to work with the CCHS Windows laptops, and the software which you will use are already installed on the laptop.

If you wish to use your own computer then the rest of this section will not work for you. You can copy the examples from this tutorial directly from the software appendix.

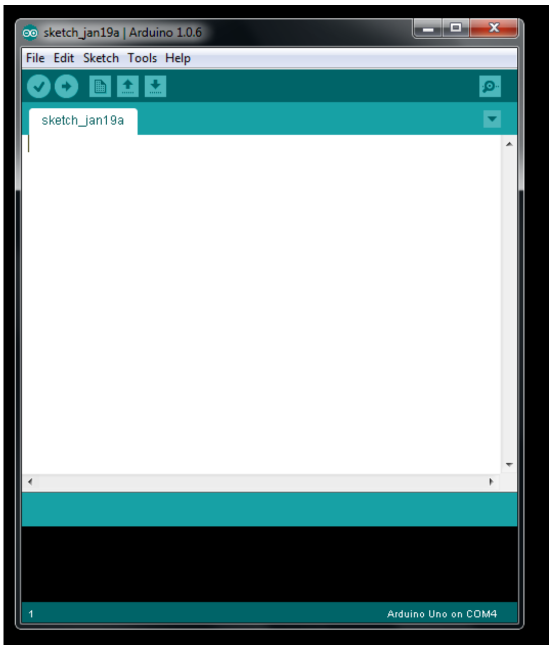
After the computer starts there maybe some updates that need to be installed. Leave them running for a few minutes, if it takes more than that ask someone to check on the computer.

Before you do anything else connect the Arduino and laptop using the provided USB cable:

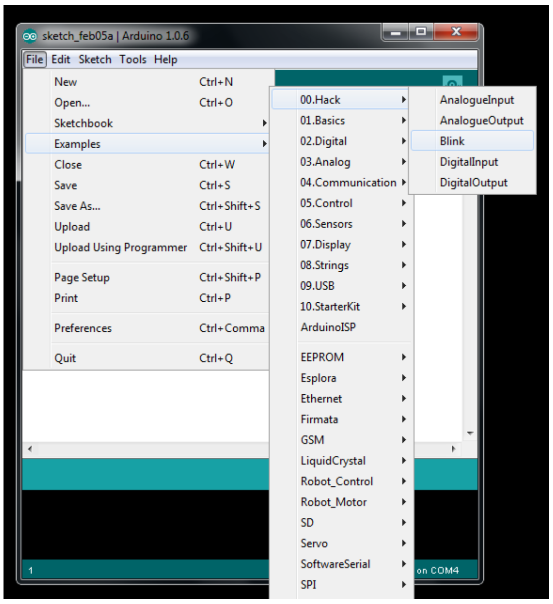


Next launch the **Arduino Integrated Development Environment (IDE)** by clicking its icon on the desktop.

The Arduino IDE will then launch with an empty project.

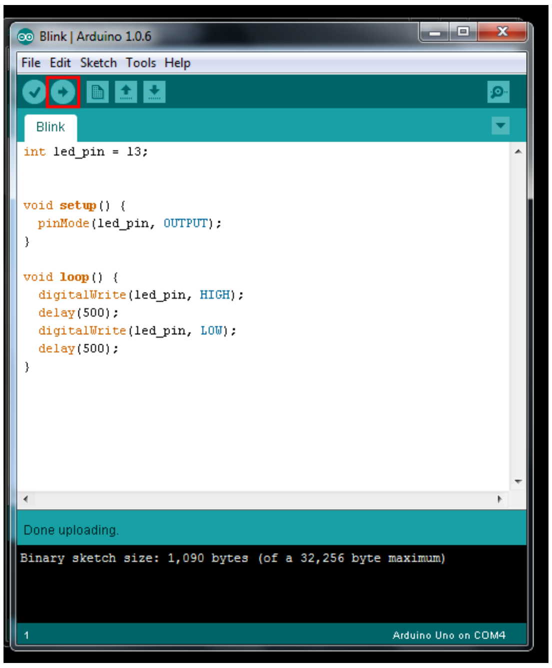


To check that everything works fine with the IDE and Arduino we will upload a program, called a **sketch**, to the Arduino. The most basic sketch is `Blink` as the name suggests it blinks something on the Arduino. You can find it under: `File>Examples>00.Hack>Blink`.



A new window with some code should open. We are not interested in the code right now, just in getting the sketch running on the Arduino.

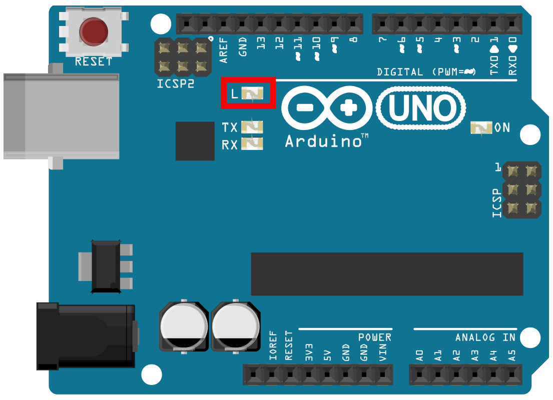
To upload the sketch click on the highlighted arrow button. The IDE will think for a bit and seem to do nothing. When it has finished uploading the sketch a `Done Uploading` message will appear underneath the code along with some technical details.



```
int led_pin = 13;


void setup() {
  pinMode(led_pin, OUTPUT);
}

void loop() {
  digitalWrite(led_pin, HIGH);
  delay(500);
  digitalWrite(led_pin, LOW);
  delay(500);
}
```

```
Done uploading.
Binary sketch size: 1,090 bytes (of a 32,256 byte maximum)
```

If everything worked you will now see the highlighted LED, labeled L on the Arduino, blinking once a second.



Congratulations! You just got your first Arduino program running!

If it didn't and there was an error message than most likely Windows has guessed that the Arduino is connected on the wrong COM port, it should be connected on COM 4,5 or 6 depending on where it was plugged in. If you don't know what that means ask someone for help, don't worry we are a friendly bunch :-)

1.1.2 The Blink Sketch

This sketch has three parts, the declarations at the top, the setup and loop. Of setup and loop are universal to every Arduino sketch.

The `setup` is run once when the sketch is uploaded to the Arduino, when the reset button is pressed, or when the Arduino is powered off and on. The `loop` first runs right after set up has finished. Once the Arduino has done all the instructions in the loop it goes straight back at the top and starts over. The name is somewhat of a giveaway.

```
int led_pin = 13;
```

The Arduino language is quite close to C in syntax. You need to end every line with a semicolon or else you **will** get errors. In the above line you declare an `int`, this means that you reserve an `int` sized space in the Arduinos memory this means a whole number between about -30,000 to 30,000. We then name that space `led_pin` and set the value there to 13.

```
void setup() {
  pinMode(led_pin, OUTPUT);
}
```

In proper C `setup` would be a function and would need to be embedded in a larger `main` function to run. The minimum operating system already on the Arduino will take care of running the setup and loop function.

The `void` keyword tells the Arduino that it will get no information from the function on what it is doing.

`pinMode` tells the Arduino to set up the hardware in such a way that pin 13 can send out strong electrical signals to the outside world. In the CCHS Arduino pin 13 is also connected to an LED on the board, so every time it is activated you will see the L LED turn on.

```
void loop() {
  digitalWrite(led_pin, HIGH);
  delay(500);
  digitalWrite(led_pin, LOW);
  delay(500);
}
```

`digitalWrite` sets a given pin either high or low, in this case it's pin 13. The `HIGH` and `LOW` are known as macros and come from the C programming language. In this case they stand for the voltages we want out of pin 13: 0V and 4.7V respectively. In general macros can stand for any text you do not want to type out, or as in our case implementation details you want someone else to take care of somewhere.

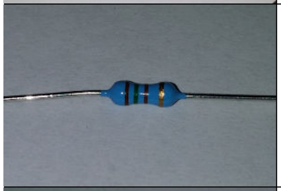
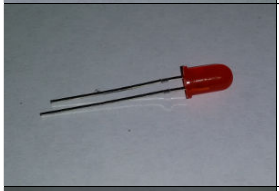
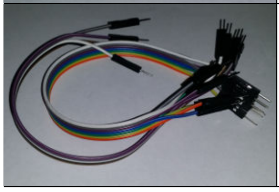
`delay` is function that makes the Arduino wait for a given time in thousands of a second. In this case 500, or half a second.

1.2 Digital Output

In this section you will use the `DigitalOutput` sketch. It is located in the same folder as `Blink`, to open it go to `Examples>00.Hack>DigitalOutput`. It is exactly the same as the `Blink` sketch. The difference is that you will build two circuits to see how the basic digital pins of the Arduino can be used to control electronics not on the board.

Go ahead and upload it in exactly the same way as you did blink.

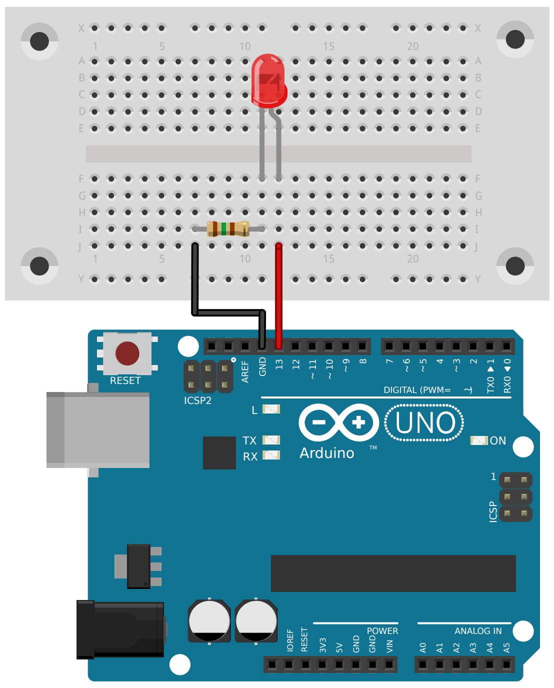
You will need the following from the kit:

	1 × Bread Board
	1 × 150Ω Resistor
	1 × 5mm Red LED
	2 × Jumper Cables

1.2.1 Pin as Source

First you will use the pin as the source of current that drives the circuit. This is the easier of the two ways of using the digital pins on an Arduino to turn things on and off.

Wire the components listed above as show in the diagram bellow. You really should disconnect the Arduino from the USB cable, while the voltages and currents from an Arduino will never be dangerous touching stray cables with power in them while putting

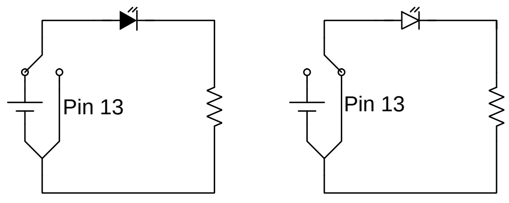


Note carefully the holes in which the components and wires are plugged in! Breadboards will only work if you follow their internal wiring. In this case two components are connected if they are plugged into the same column in the middle of the board and not separated by the ridge. For more details on how breadboards work see their section 1.2.4.

Also note that LEDs only allow current to flow in one direction, to get the LEDs wired right you need to have one with the long leg connected to pin 13 and the short leg connected to the ground (GND) pin.

You should now see both the built in L LED and the external red LED blinking in time with each other.

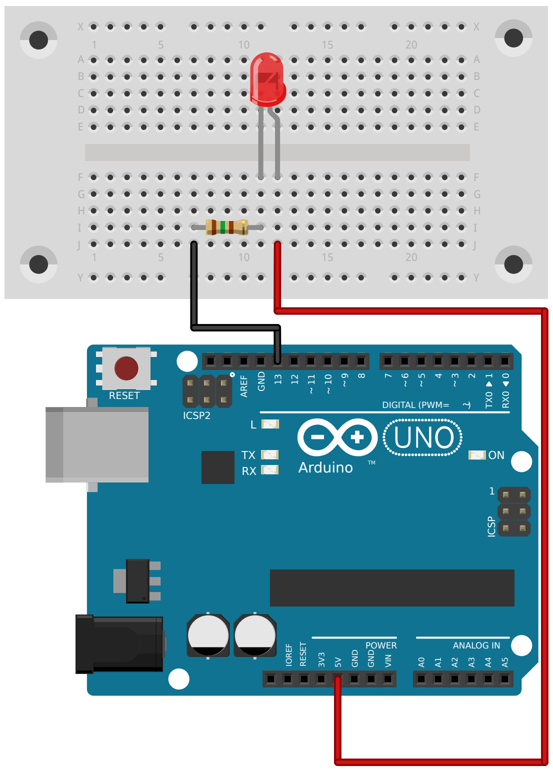
The circuit you build looks like the bellow diagram. If you are no familiar with circuit diagrams and are interested in seeing how it works in detail see the section on Physics and Hardware.



As you can see, turning pin 13 on connects a source to the circuit and current flows. Turning pin 13 off removes the source from the circuit and everything turns off.

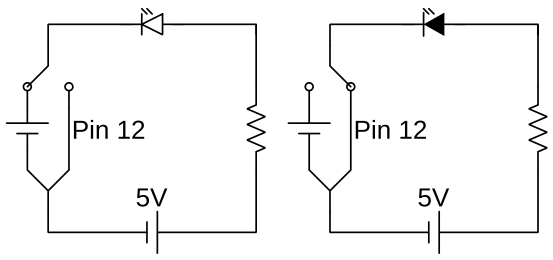
1.2.2 Pin as Sink

The circuit in this section uses the same code as the last one, the difference is that the pin will be used as ground.



Note that even though the circuits look similar they are very different. The red wire is live and the black is ground.

If you got everything wired up the right way around you should see the L LED and red external LED blink out of sink with each other.



In the circuit above you see that when pin 13 is on the voltage it is connected to counteracts the voltage from the power pin. When the pin 13 is off the circuit is closed and the external LED turns off.

It is common to use digital pins as both sources and sinks of current in circuits, which you choose to use depends on exactly what part you want to control.

1.2.3 Software

The Sketch

```
int led_pin = 13;


void setup() {
  pinMode(led_pin, OUTPUT);
}

void loop() {
  digitalWrite(led_pin, HIGH);
  delay(500);
  digitalWrite(led_pin, LOW);
  delay(500);
}
```

This sketch is essentially the same as the `Blink` sketch you looked at line by line in the last section.

Instead a few words about programming: It is always good to giver discriptive names to variables so you remember what they refer to. Naked numbers in the body of a program are generally a bad idea, especially in long ones, in the above program you might want to keep the LED on pin 13 high for a second, since you have just two 500s you need to read the whole program to figure out which one you need to change.

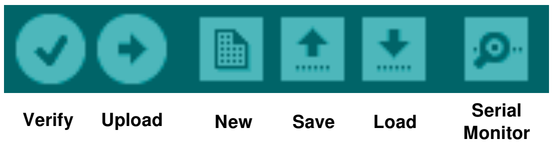
Other than using variables for every number, something that can get tedious too, we can use comments to remind you of things you wrote yourself. Everything between `/* */` and after `//` is a comment. This is text ignored by the computer and is only used to explain what software does to the people.

Compiling, Uploading and Executing

To get a sketch running on the Arduino is it needs to be translated from human readable text to numbers the computer can understand. This is done by **compiling** it. Once you have the compiled sketch on your computer it is then sent by the USB cable to the Arduino which **executes** it.

At a lower level this is done by the IDE for you, it first expands the macros we met before into code that makes sense, runs the GCC cross-compiler for ARV, and uploads the resulting binary into a predefined place in the memory of the Arduino, then restarts it. Once this happens the operating system on the Arduino takes over. It runs some basic diagnostic tests, then jumps to the part of memory where setup is located, runs it, jumps to where loop is located, and keeps running that so long as the Arduino has power.

You have already compiled sketches multiple times by using the upload button. Using the buttons on the top row of the IDE is the simplest way to write and edit sketches for the Arduino.



Verify: Compile your sketch without uploading it. Useful for error checking.

Upload: Compile and upload your sketch. The Arduino begins to execute it immediately.

New: Discard everything you have done and start in a blank sketch.

Save: Save your sketch to the computer. You should keep in mind that even if you have the sketch uploaded to the Arduino you can't get back the code from it. If you want to access a sketch in the future you **need** to save it.

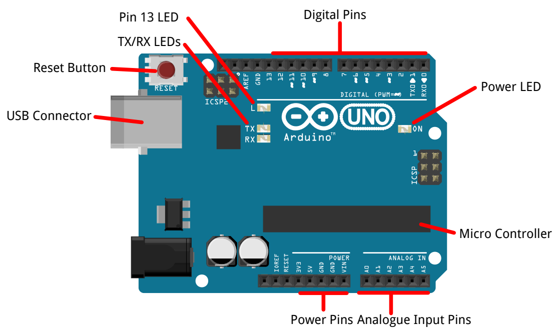
Load: Open a previously saved sketch. Again, you can't load a sketch *from* the Arduino, only sketches that you have saved on your computer.

Serial Monitor: A way for the Arduino to talk back to your computer through the USB cable. We will see how this works in detail in the Digital Input section.

1.2.4 Hardware

The Arduino

We already talked about what an Arduino is in general, however looking at the parts which we will be using more closely:



Pin 13 LED: You have already used this extensively, the only other thing is that it is used by the Arduino for diagnostic purposes during upload and reboot. To you this will look like fast blinking which you didn't tell it to do for about 1 second after either an upload or reboot.

TX/RX LED: These are LEDs that activate when the computers talks to the Arduino (RX), such as when it uploads a sketch, or when the Arduino talks to the computer (TX), such as when running a serial monitor or connecting through the USB cable.

Reset Button: This resets the Arduino to the state it had when the program was first uploaded. This is useful when a sketch you write stops working after something it didn't expect happens.

USB Cable: The most common way to connect to the Arduino and to give power to it. Because of the limitations of the USB standard the Arduino is limited to 5 volts and a few hundred milliamp, much less than even a AAA battery, such tiny currents and voltages make it completely safe but not able to drive more power hungry components like DC motors directly.

Power Pins: These are pins which can be used when you need just a simple battery like source, either in 3 volts or 5 volts.

Analogue Input Pins: A specialized set of pins that can be used to read continuously varying voltages, such as light detectors. Limited to voltages between 0V to 5V.

Micro Controller: The heart of the Arduino. All the code lives here.

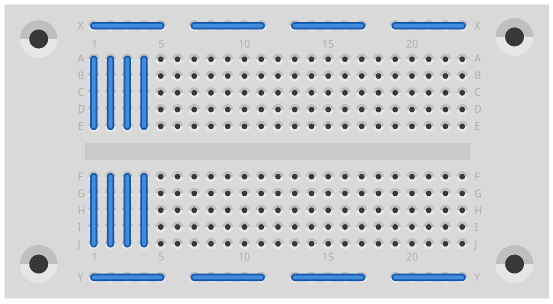
Power LED: The first thing you should look at if your Arduino isn't working.

Digital Pins: A set of pins which can can be used for input or output. They can only send out and receive high or low signals. Pins with a $\sim$ next to the number can approximate an analogue output by quickly turning on and off see analogue output section.

The advantage of having a micro controller over other arrangements, such is that Raspberry Pi, is that it is quite easy to replace the micro controller with another chip and use it as is in one of your projects. At between one to three dollars a piece even small hardware projects can have multiple Arduinos embedded within them at a negligible cost and extra power.

The Bread Board

A breadboard is the circuit equivalent of a Lego set, and like a Lego set where you put anything has a huge impact on how things get wired.



The holes in the middle of the board are connected to each other in half columns separated by a ridge in the middle. On the outside the holes are connected by rows in the same way. Each blue line above shows which holes are connected together in a circuit. If you connect a wire to any of them it is as if you had connected it to all of them.

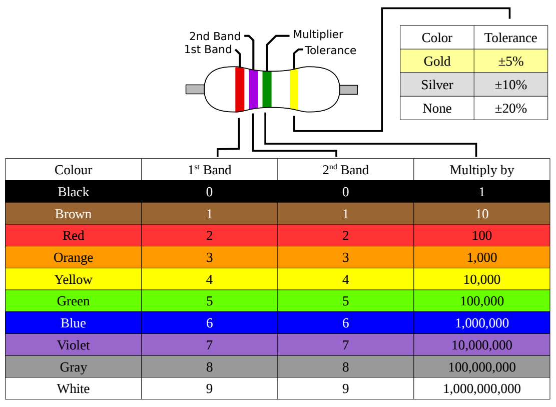
The breadboard you have with the kit is somewhat larger than the one in the diagrams here but is wired on exactly the same principles.

While it isn't as robust as a soldered circuit or printed circuit board and making sure that every wire is threaded into the right hole might seem like a pain, the flexibly it adds to prototyping makes it more than a worth while trade off. Every electronic project in the space started off on a breadboard somewhere.

The Resistor

Resistors are electrical components which are used to limit the current through, or voltage across, other more sensitive electronic components. For the circuits which you will be using here they can be considered indestructible. For an explanation on what current and voltage mean please see the Physics Appendix A.

Every resistor is colour coded, the ones in the kit are 3 band and the following info graphics explains how to read them:

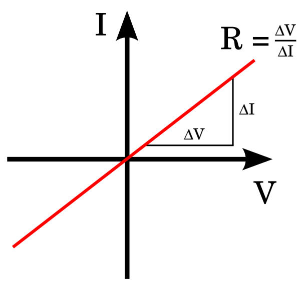

2nd Band
1st Band
Multiplier
Tolerance

Color	Tolerance
Gold	±5%
Silver	±10%
None	±20%

Colour	1st Band	2nd Band	Multiply by
Black	0	0	1
Brown	1	1	10
Red	2	2	100
Orange	3	3	1,000
Yellow	4	4	10,000
Green	5	5	100,000
Blue	6	6	1,000,000
Violet	7	7	10,000,000
Gray	8	8	100,000,000
White	9	9	1,000,000,000

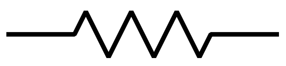
In the above example we have red, purple, green, gold. This translates to 2, 7, 100,000 ±5% which is $27 \times 100,000 = 2,700,000\Omega \pm 5\%$. This is a very large resistance, one which you will not be using in any project here. The resistors you will deal with will be between 10,000Ω and 100Ω.

At the level of physics a resistor is an anything that follows the rule $V = IR$ and the equivalent $I = \frac{V}{R}$, $R = \frac{V}{I}$. Bellow is a current voltage characteristic plot which shows how the current changes in response to changes in voltage for a given idealized resistor.



Unlike LEDs they have no preferred direction and will work equally well whichever way they are plugged in.

In circuits the resistor is most often represented by

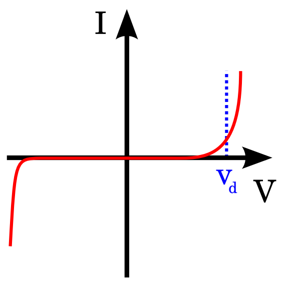


The LED

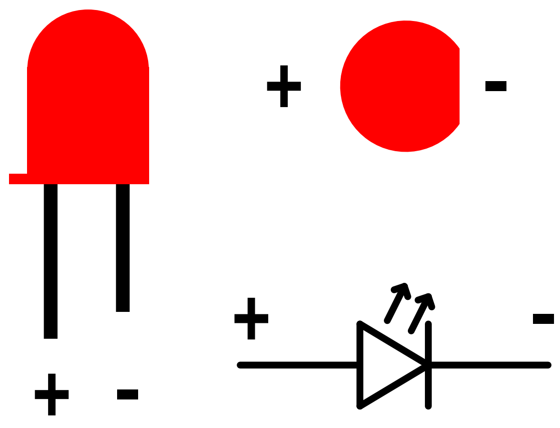
LEDs are semiconductor devices, to understand how they work in detail requires quite involved quantum mechanics, however to use them successfully in circuits you only need to remember a few simple rules:

- They will not work bellow a given voltage, usually between 1.2-3.4V depending on their colour. This is called the voltage drop v_d.
- They need a resistor to limit current through them so they do not burnout.
- During normal operation they will only let current flow in one direction, if you supply a high enough voltage in the other direction the LED will burnout.

The voltage-current characteristic for an LED, as you can see there isn't a simple relationship between voltage and current like there is for the resistor. You will always want to run the LED close to the v_d spot on the diagram.



Bellow is a diagram on how to wire an LED properly. It's worth remembering that positive in the direction of the source and negative in the direction of ground.

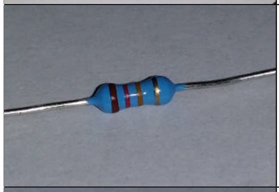
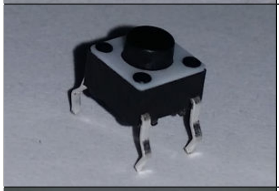
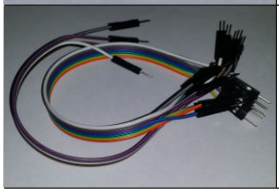


The last symbol above is how LEDs are most often represented in diagrams.

1.3 Digital Input

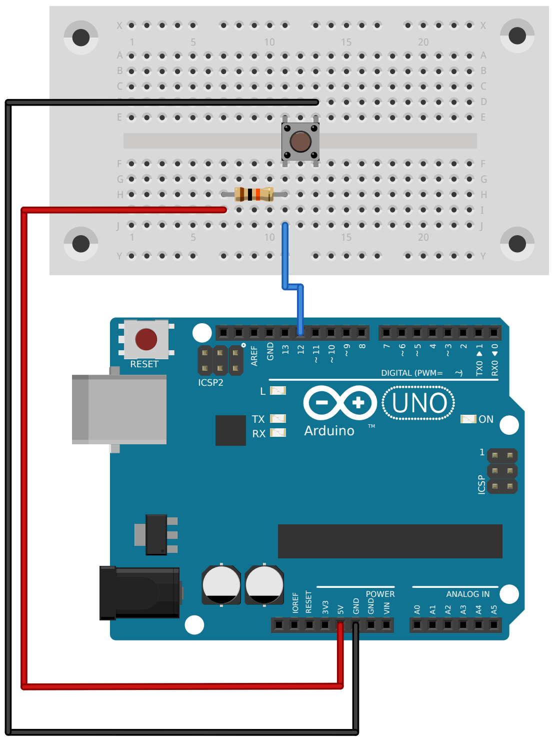
In this section you will let the outside world talk to the Arduino and let the Arduino talk to the computer. The first of these is done by taking a pin and telling the Arduino to set it aside as an input pin. The second is done by using the USB cable and telling the Arduino to print text to it. You will also see tristate logic and why buttons will never work like you expect them to.

You will need the following parts:

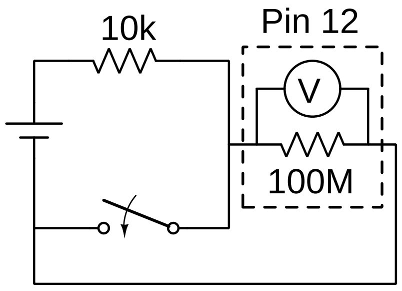
	1 × Bread Board
	1 × 12KΩ Resistor
	1 × Push Button
	3 × Jumper Cables

1.3.1 Pull-up Resistor

Like in the preceding section you will make several circuits to see the different ways of achieving the same thing.

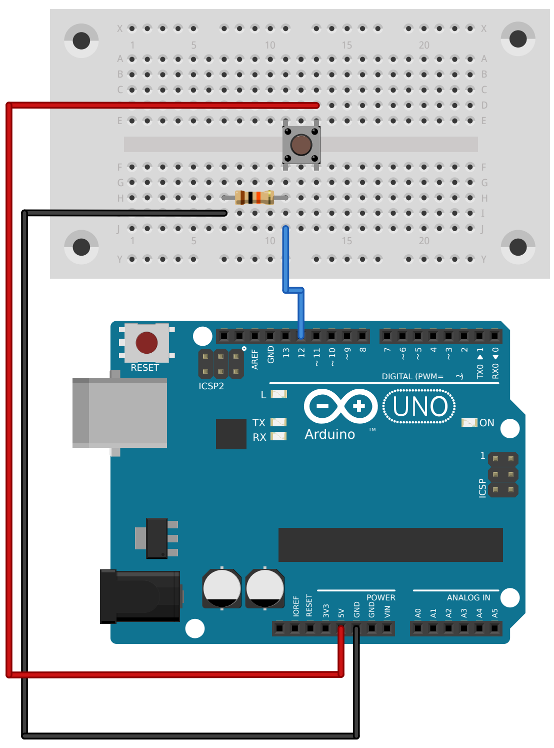


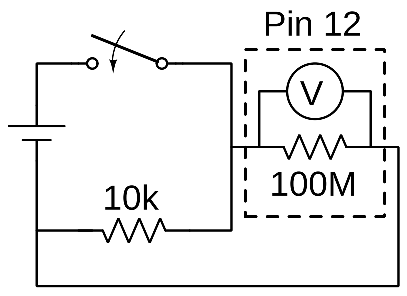
The first of these is the more "normal" circuit. When the button is not pressed the circuit is "off" as you would imagine a switch not being pressed should be.



The above shows the schematic of the circuit. Note that the internal resistor has a higher resistance than anything you have seen so far.

1.3.2 Pull-down Resistor



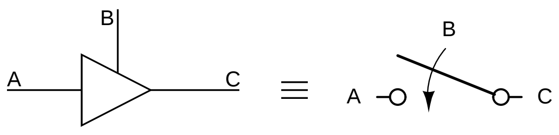


1.3.3 Tristate Logic

To understand why you bothered with the weird circuits above you need know about tristate, or floating logic. That combined with the demands of building a detector that doesn't disturb the circuit during normal operation means you need to "ground" all input pins to stop yourself from having erratic readings.

In the above circuit, for example, if you didn't use one of the pull up or pull down patterns and just connected the live button to the live wire an input pin you can get a button pressed even just by hovering your finger over the button without touching it at all.

Without having at least an acquaintance with it nothing after this will make sense.



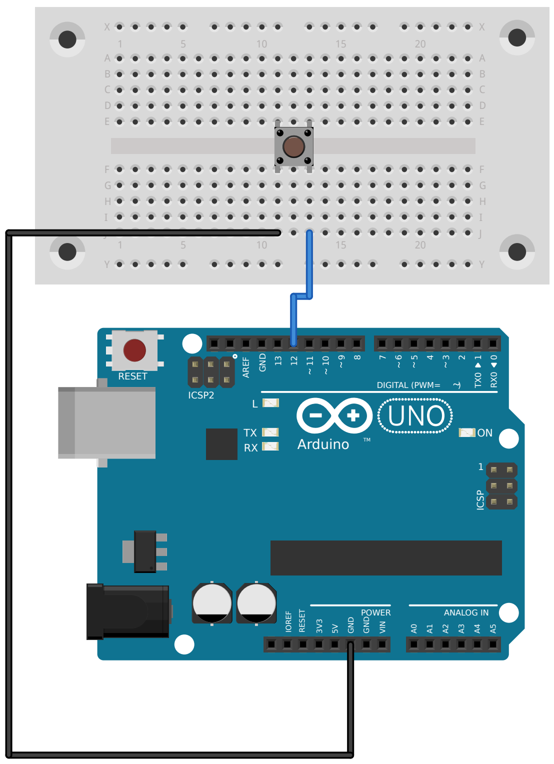
1.3.4 Software

```
int in 12;

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

void loop(){
  if (digitalRead(in)== HIGH) {
    Serial.write("I'm on\n");
    delay(1000);
  }
  else {
    Serial.write("I'm off\n");
    delay(1000);
  }
}
```

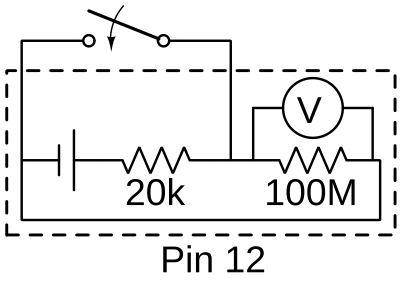
You are using pin 12 here instead of pin 13 because pin 13 has connected to it an LED, as you learned in the last section an LED causes a voltage drop of around 2-3V across it, since the logical ON here is 5V this may cause the board to not register a button press.

1.3.5 Internal Pulldown

Because input is so common the Arduino comes with a build in pull down resistor. You can connect the Arduinos output pin and ground directly to the switch.



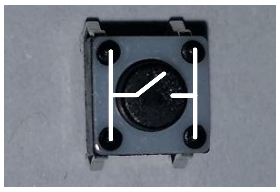
The internals of the circuit look like:



```
int in 12;

void setup(){
    Serial.begin(9600);
    pinMode(in, INPUT);
    digitalWrite(in, HIGH);
    Serial.begin(9600);
}

void loop(){
    if (digitalRead(in) == HIGH) {
        Serial.write("I'm on\n");
        delay(1000);
    }else{
        Serial.write("I'm off\n");
        delay(1000);
    }
}
```

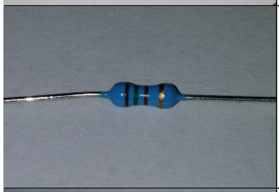
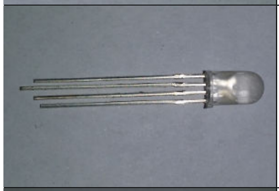
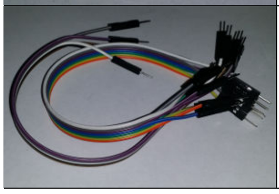
Here we see the usefullness of macros, `HIGH` keyword here means that we turn on the internall pullup.

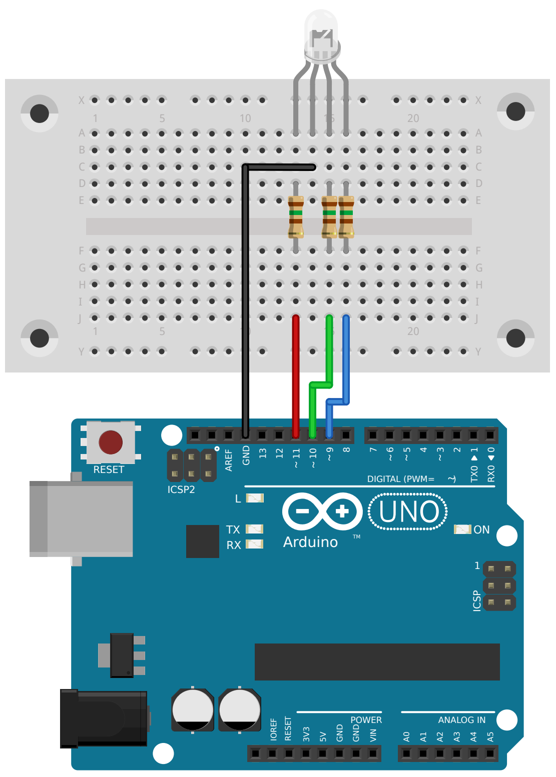
1.3.6 Hardware



The push button is an instananeous on which stays on so long as it is pressed. Note the internal wiring.

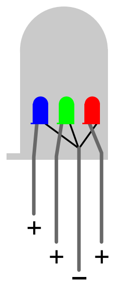
1.4 Analogue Output

	1 × Bread Board
	3 × 150Ω Resistor
	1 × RGB LED
	4 × Jumper Cables



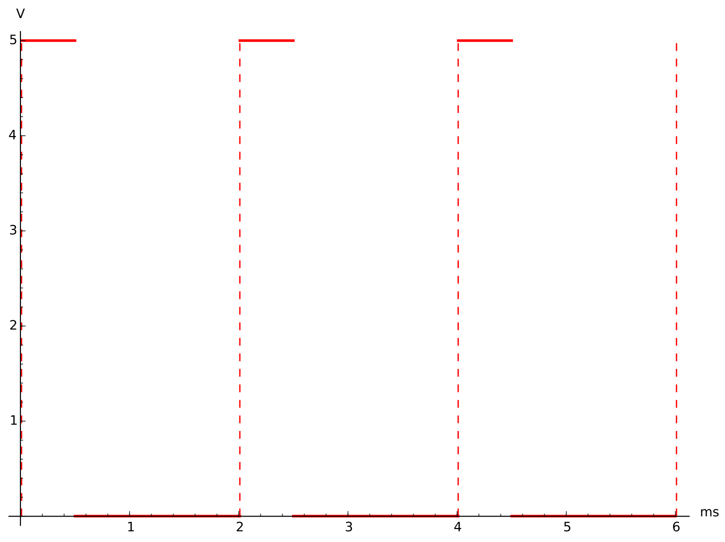
The reason why you should use a resistor for every color is that

1.4.1 Hardware



A RGB LED can be thought of as 3 separate LEDs which happen to share a common ground in terms of a circuit.

1.4.2 Pulse Width Modulation



The cycle for PWM is 2ms. It varies between 0 and 255 with 0 being fully off and 255 being fully off. Show in the diagram is somewhere around 64.

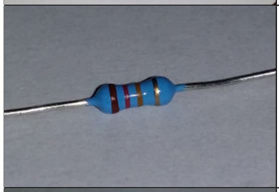
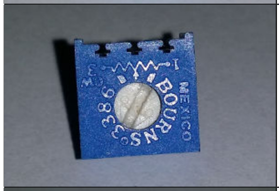
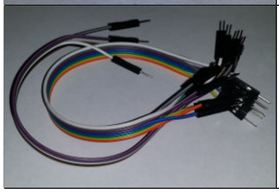
1.4.3 Software

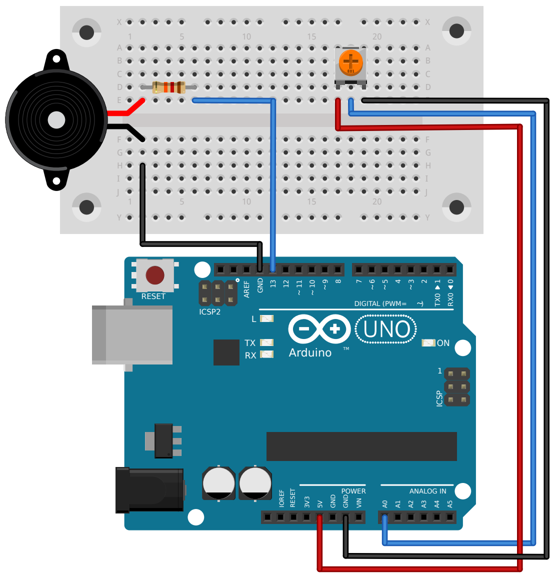
```
int redLedPin = 9;
int greenLedPin = 10;
int blueLedPin = 11;

void setup() {
  pinMode(redLedPin, OUTPUT);
  pinMode(greenLedPin, OUTPUT);
  pinMode(blueLedPin, OUTPUT);
}
void loop() {
  analogWrite(blueLedPin, random(0, 255));
  analogWrite(greenLedPin, random(0, 255));
  analogWrite(redLedPin, random(0, 255));
  delay(100);
}
```

random gives a random number between 0 and 255, meaning any value for any channel from fully off to fully on.

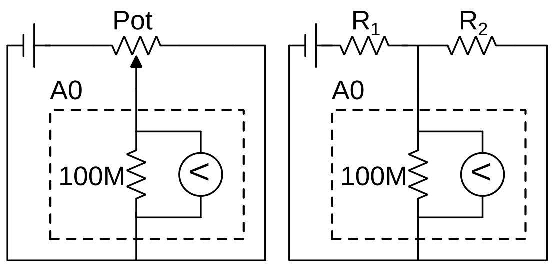
1.5 Analogue Input

In this section you will learn how to read in analogue input. The numbers aren't true analogue, just 1024 possible values. Here you will be reading off a voltage compared to 5V.

	1 × Bread Board
	1 × 12KΩ Resistor
	1 × Piezo
	1 × Potentiometer
	2 × Jumper Cables



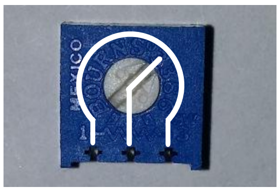
The important circuit here is the one with the pot. The other one with the piezo is just there to give you feedback on the value that's being read.



The equivalent circuit from which you will read off the voltage. Explain more of the physics.

1.5.1 Hardware

Potentiometer



The potentiometer, pot for short, is a variable resistor. By connecting the thing you already ground it so you can't get tristate logic, if you just connect live and input you will only get noise even when its off.

You control it by turning the knob in the middle.

Piezo

This is an element that shrinks and expends when a voltage is applied. This produces noise.

1.5.2 Software

```
int piezo = 8;

void setup()
{
    Serial.begin(9600);
    pinMode(piezo,OUTPUT);
    pinMode(A0,INPUT);
}
void loop()
{
    int sensorValue = analogRead(A0);
    Serial.println(sensorValue);
    tone(piezo, 4*sensorValue);
}
```

Tone is half duty cycle unsigned unsigned int, can overflow.

Appendix A

All the Physics You Will Need

There are two measurable quantities that all electrical circuits share: **current** I and **voltage** V.

Current is a way to count the number of electrons flowing past a point in a wire during some period of time, for what we will be doing here you can imagine them as tiny tennis balls flying inside the wire.

Voltage is a measure of how much energy each ball has, thinking of tennis balls again you can imagine it as how much it will hurt to be hit by a ball served during the Australian open verses one gently rolled across the table: the higher the voltage the more it will hurt.

Notice that the two are independent: how many electrons flow past a point is not related to their energy, nor the other way around.

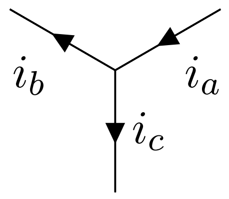
In all electrical circuits we use electricity to deliver **power** P to something. The power delivered is the current times the voltage: $P = VI$. Circuit design is about giving every element as much power as it needs, but not so much that it burns out.

What "too much" power means depends on how much power an element converts to heat, and how efficient at dissipating heat it is.

Knowing only that you and looking at the current voltage characteristics you can see why a resistor is a lot more robust than an LED: a tiny change in the voltage around v_d will cause a hugely disproportionate increase in the power delivered, quite probably more than enough to burn it out. For the resistor, by contrast there, would have to be a huge increase in the voltage for it to burnout, since the power delivered to it increases much more evenly.

To deal specifically with circuits you need a further two laws: Kirchhoff's current and voltage law.

The current law is a way of saying that current can't be created out of nothing: As much current must leave any junction in a circuit as enters it. You will also need two laws for dealing with circuits: Kirchhoff's circuit laws.



In the above we must have $i_a = i_c + i_b$

Redo the so it doesn't make your eyes bleed, add a dot in

Appendix B

Circuit Symbols

Appendix C

Source Code

Blink/Digital Output

```
int led_pin = 13;


void setup() {
  pinMode(led_pin, OUTPUT);
}

void loop() {
  digitalWrite(led_pin, HIGH);
  delay(500);
  digitalWrite(led_pin, LOW);
  delay(500);
}
```

Digital Input

External

```
int in 12;

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

void loop(){
  if (digitalRead(in)== HIGH) {
    Serial.write("I’m on\n");
    delay(1000);
  }
  else {
    Serial.write("I’m off\n");
    delay(1000);
  }
}
```

Internal

```
int in 12;

void setup(){
    Serial.begin(9600);
    pinMode(in, INPUT);
    digitalWrite(in, HIGH);
    Serial.begin(9600);
}

void loop(){
    if (digitalRead(in) == HIGH) {
        Serial.write("I’m on\n");
        delay(1000);
    }else{
        Serial.write("I’m off\n");
        delay(1000);
    }
}
```

Analog Output

```
int redLedPin = 9;
int greenLedPin = 10;
int blueLedPin = 11;

void setup() {
  pinMode(redLedPin, OUTPUT);
  pinMode(greenLedPin, OUTPUT);
  pinMode(blueLedPin, OUTPUT);
}
void loop() {
  analogWrite(blueLedPin, random(0, 255));
  analogWrite(greenLedPin, random(0, 255));
  analogWrite(redLedPin, random(0, 255));
  delay(100);
}
```

Analog Input

```
int piezo = 8;

void setup()
{
    Serial.begin(9600);
    pinMode(piezo,OUTPUT);
    pinMode(A0,INPUT);
}
void loop()
{
    int sensorValue = analogRead(A0);
    Serial.println(sensorValue);
    tone(piezo, 4*sensorValue);
}
```