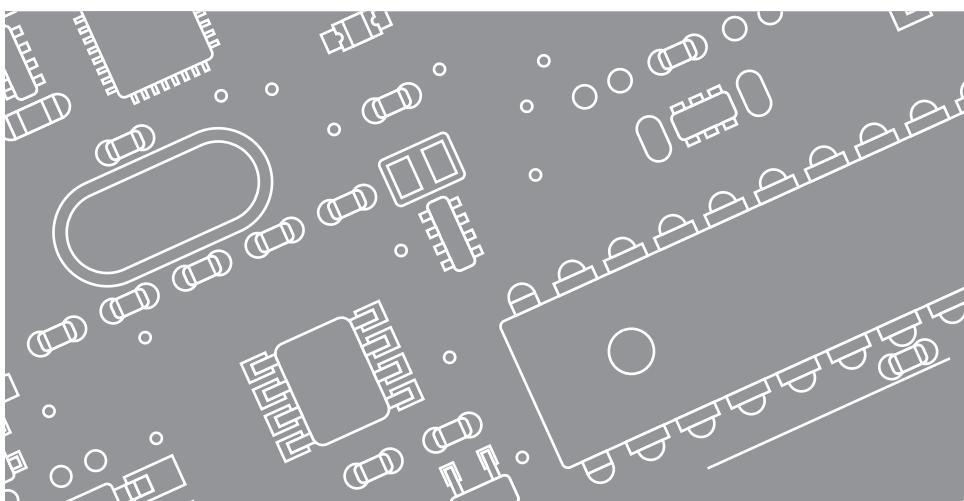


## Quad Store India - Ultimate Kit

Your Guide to the Xtreme Inventors Kit for Arduino



## Table of Contents



# Welcome to the *Inventor's Guide*

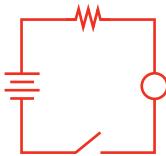
The *Quad Store India* Ultimate Inventor's Guide is your map for navigating the waters of beginning embedded electronics. This booklet contains all the information you will need to explore the 14 circuits of the Ultimate Inventor's Kit for Arduino. At the centre of this manual is one core philosophy - that anyone can (and should) play around with electronics. When you're done with this guide, you'll have the know-how to start creating your own projects and experiments. Now enough talking - let's get inventing!



## Section 1:

### *Getting Started with Arduino*

What is an Arduino?	1
Download Arduino Software (IDE)	3
Install Drivers for Arduino	4
Identify your Arduino	7
Download "SIK Guide Code"	8



## Section 2:

### *Getting Started with Circuits*

The World Runs on Circuits	9
Inventory of Parts	11
Arduino Uno	13
Breadboard	15
Circuit #1 - Your First Circuit: Blinking a LED	17
Circuit #2 - Potentiometer	24
Circuit #3 - RGB LED	28
Circuit #4 - Multiple LEDs	32
Circuit #5 - Push Buttons	36
Circuit #6 - Photo Resistor	40
Circuit #7 - Temperature Sensor	44
Circuit #8 - A Single Servo	48
Circuit #9 - Flex Sensor	52
Circuit #10 - Soft Potentiometer	56
Circuit #11 - Piezo Element	60
Circuit #12 - Spinning a Motor	64
Circuit #13 - Relay	68
Circuit #14 - Shift Register	72

# What is an Arduino?



## The Arduino Revolution

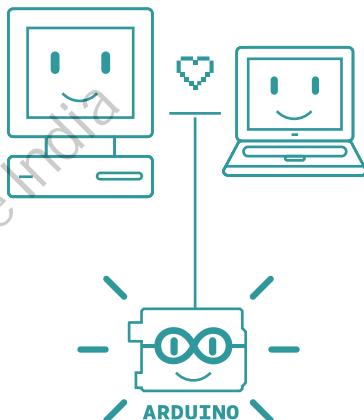
Arduino is an open-source physical computing platform designed to make experimenting with electronics more fun and intuitive. Arduino has its own unique, simplified programming language, a vast support network, and thousands of potential uses, making it the perfect platform for both beginner and advanced DIY enthusiasts.

[arduino.cc](http://arduino.cc)

## A Computer for the Physical World

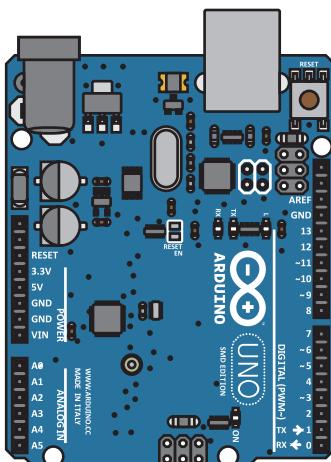
The friendly blue board in your hand (or on your desk) is the Arduino. In some ways you could think of Arduino as the child of traditional desktop and laptop computers. At its roots, the Arduino is essentially a small portable computer. It is capable of taking **inputs** (such as the push of a button or a reading from a light sensor) and interpreting that information to control various **outputs** (like a blinking LED light or an electric motor).

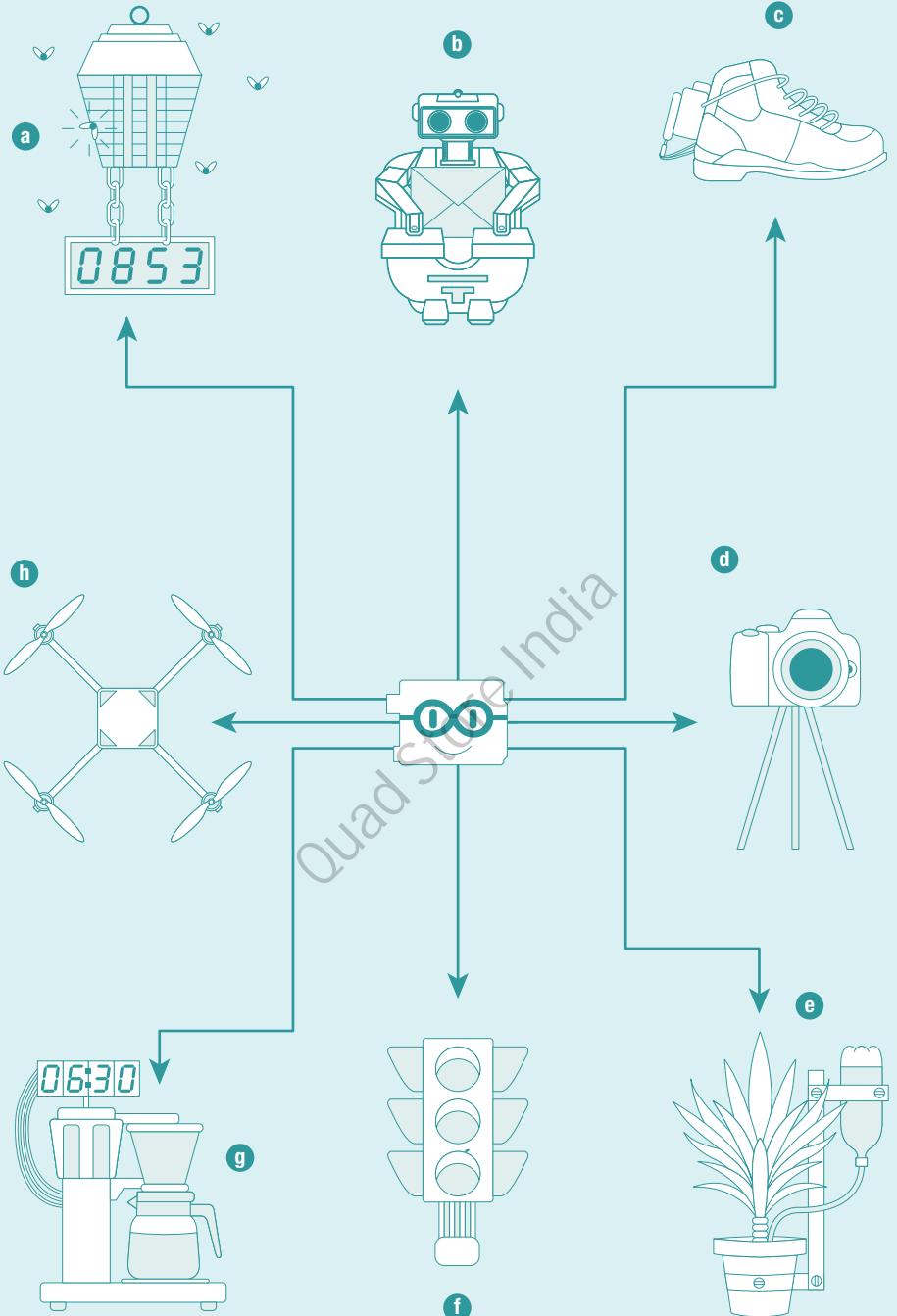
That's where the term "physical computing" is born - an Arduino is capable of taking the world of electronics and relating it to the physical world in a real and tangible way. Trust us - this will all make more sense soon.



### // Arduino UNO SMD R3

The Arduino Uno is one of several development boards based on the ATmega328. We like it mainly because of its extensive support network and its versatility. It has 14 digital input/output pins (6 of which can be PWM outputs), 6 analog inputs, a 16 MHz crystal oscillator, a USB connection, a power jack, an ICSP header, and a reset button. Don't worry, you'll learn about all these later.





- a** Bug Zapper Counter
- b** Old Toy Email Notifier
- c** Power-Lacing High Tops

- d** Camera Time-lapse operation
- e** Auto-Plant Watering
- f** Re-Programmed Traffic Light

- g** Auto-Coffee Maker
- h** Quadcopter

## Download the Arduino IDE (Integrated Development Environment)



### Access the Internet

In order to get your Arduino up and running, you'll need to download some software first from [www.arduino.cc](http://www.arduino.cc) (it's free!). This software, known as the Arduino IDE, will allow you to program the Arduino to do exactly what you want. It's like a word processor for writing programs. With an internet-capable computer, open up your favorite browser and type in the following URL into the address bar:



[arduino.cc/en/Main/Software](http://arduino.cc/en/Main/Software)

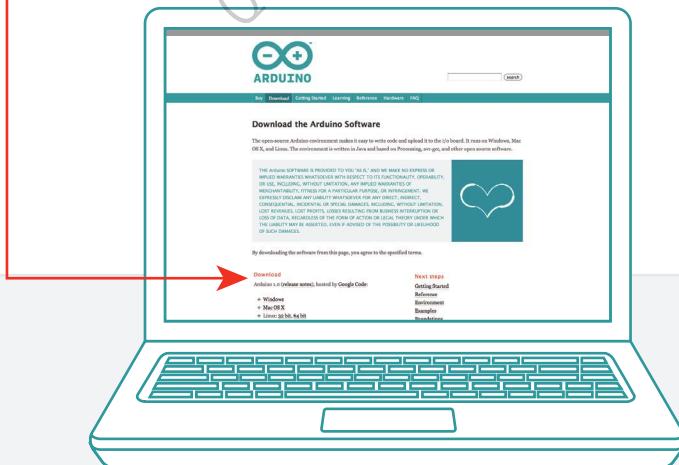
< case sensitive >

1

#### Download

Click on the “+” sign next to your appropriate computer operating system.

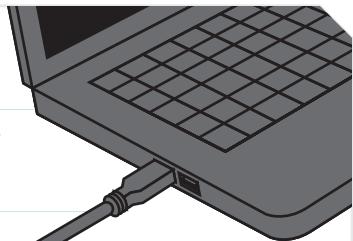
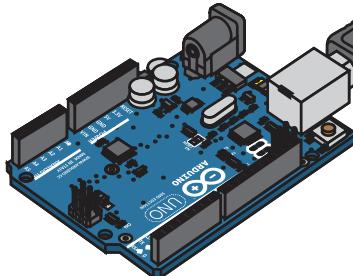
- + Windows
- + Mac OS X
- + Linux: 32 bit, 64 bit
- + source



Choose the appropriate Operating System installation package for your computer.

## // Connect your Arduino Uno to your Computer

Use the USB cable provided in the SIK kit to connect the Arduino to one of your computer's USB inputs.



2

## // Install Drivers

3

Depending on your computer's operating system, you will need to follow specific instructions. Please consult the URLs below for specific instructions on how to install the drivers onto your Arduino Uno.

\* You will need to scroll to the section labeled "Install the drivers".

### **Windows Installation Process**



Go to the web address below to access the instructions for installations on a Windows-based computer.

<http://arduino.cc/en/Guide/Windows>

### **Macintosh OS X Installation Process**



Macs do not require you to install drivers. Enter the following URL if you have questions. Otherwise proceed to next page.

<http://arduino.cc/en/Guide/MacOSX>

### **Linux: 32 bit / 64 bit, Installation Process**



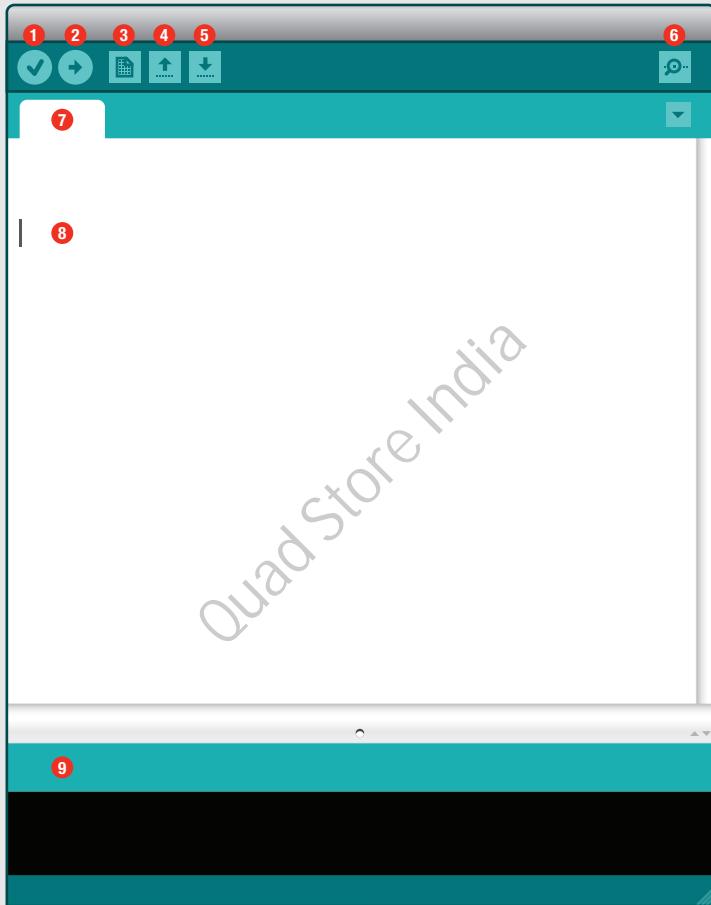
Go to the web address below to access the instructions for installations on a Linux-based computer.

<http://www.arduino.cc/playground/Learning/Linux>



## // Open the Arduino IDE:

Open the Arduino IDE software on your computer. Poke around and get to know the interface. We aren't going to code right away, this is just an introduction. The step is to set your IDE to identify your Arduino Uno.



## GUI (Graphical User Interface)

- 1 **Verify:** Compiles and approves your code. It will catch errors in syntax (like missing semi-colons or parenthesis). // See Diagram Below
- 2 **Upload:** Sends your code to the Arduino board. When you click it, you should see the lights on your board blink rapidly. // See Diagram Below
- 3 **New:** This button opens up a new code window tab.
- 4 **Open:** This button will let you open up an existing sketch. // See Diagram Below
- 5 **Save:** This saves the currently active sketch.
- 6 **Serial Monitor:** This will open a window that displays any serial information your Arduino is transmitting. It is very useful for debugging.
- 7 **Sketch Name:** This shows the name of the sketch you are currently working on.
- 8 **Code Area:** This is the area where you compose the code for your sketch.
- 9 **Message Area:** This is where the IDE tells you if there were any errors in your code.

// The three most important commands for this guide are seen below:



*Open*



*Verify*

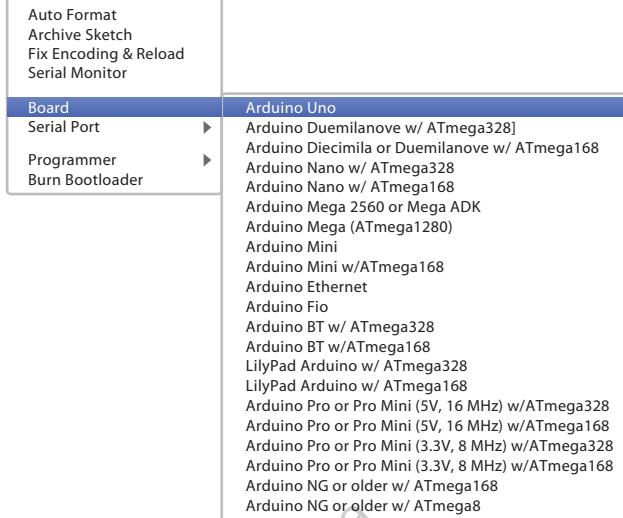


*Upload*

# 4

## // Select your board: Arduino Uno

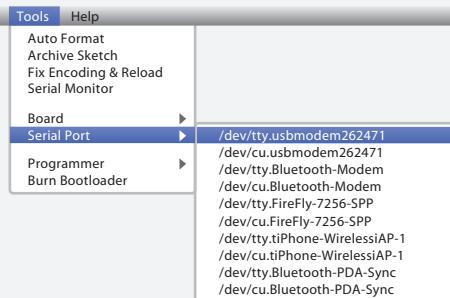
File Edit Sketch Tools Help



Select the serial device of the Arduino board from the Tools | Serial Port menu. This is likely to be **com3 or higher** (COM1 and COM2 are usually reserved for hardware serial ports). To find out, you can disconnect your Arduino board and re-open the menu; the entry that disappears should be the Arduino board. Reconnect the board and select that serial port.



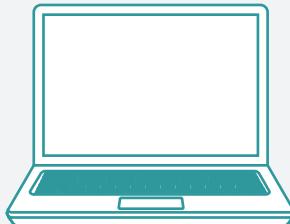
Select the serial device of the Arduino board from the Tools > Serial Port menu. On the Mac, this should be something with **/dev/tty.usbmodem** (for the Uno or Mega 2560) or **/dev/tty.usbserial** (for older boards) in it.



<http://www.arduino.cc/playground/Learning/Linux>

## Download Arduino Code (For use with the circuits in this guide)

# 5



Type in the following URL to download the code:



**quadstore.in/UKit/SIK\_Guide\_Code\_32.zip**

(or)

You can find "SIK Guide Code" Zip file from the CD which came with the kit.



Unzip the file "SIK Guide Code". It should be located in your browser's "Downloads" folder. Right click the zipped folder and choose "unzip".



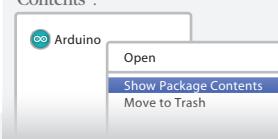
Copy the "SIK Guide Code" folder into Arduino's folder named "examples".



Unzip the file "SIK Guide Code". It should be located in your browser's "Downloads" folder. Right click the zipped folder and choose "unzip".



Find "Arduino" in your applications folder. Right click (ctrl + click) on "Arduino". Select "Show Package Contents".

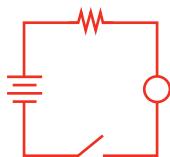


Copy the "SIK Guide Code" folder into Arduino's folder named "examples".



<http://www.arduino.cc/playground/Learning/Linux>

# Getting Started with Circuits



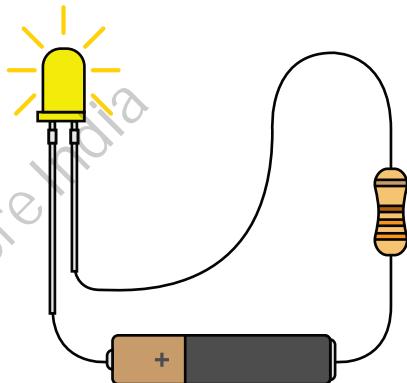
## ***What is an Electrical Circuit?***

A circuit is basically an electronics loop with a starting point and an ending point - with any number of components in between. Circuits can include resistors, diodes, inductors, sensors of all sizes and shapes, motors, and any other handful of hundreds of thousands of components.

Circuits are usually divided into three categories - analog circuits, digital circuits, or mixed-signal circuits. In this guide, you will explore all three sets of circuits.

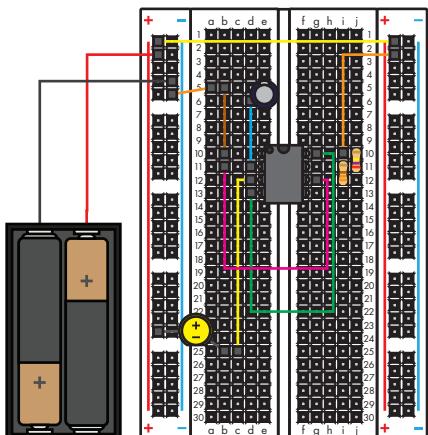
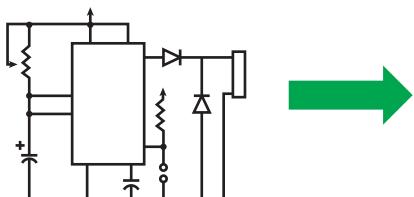
## ***The World Runs on Circuits:***

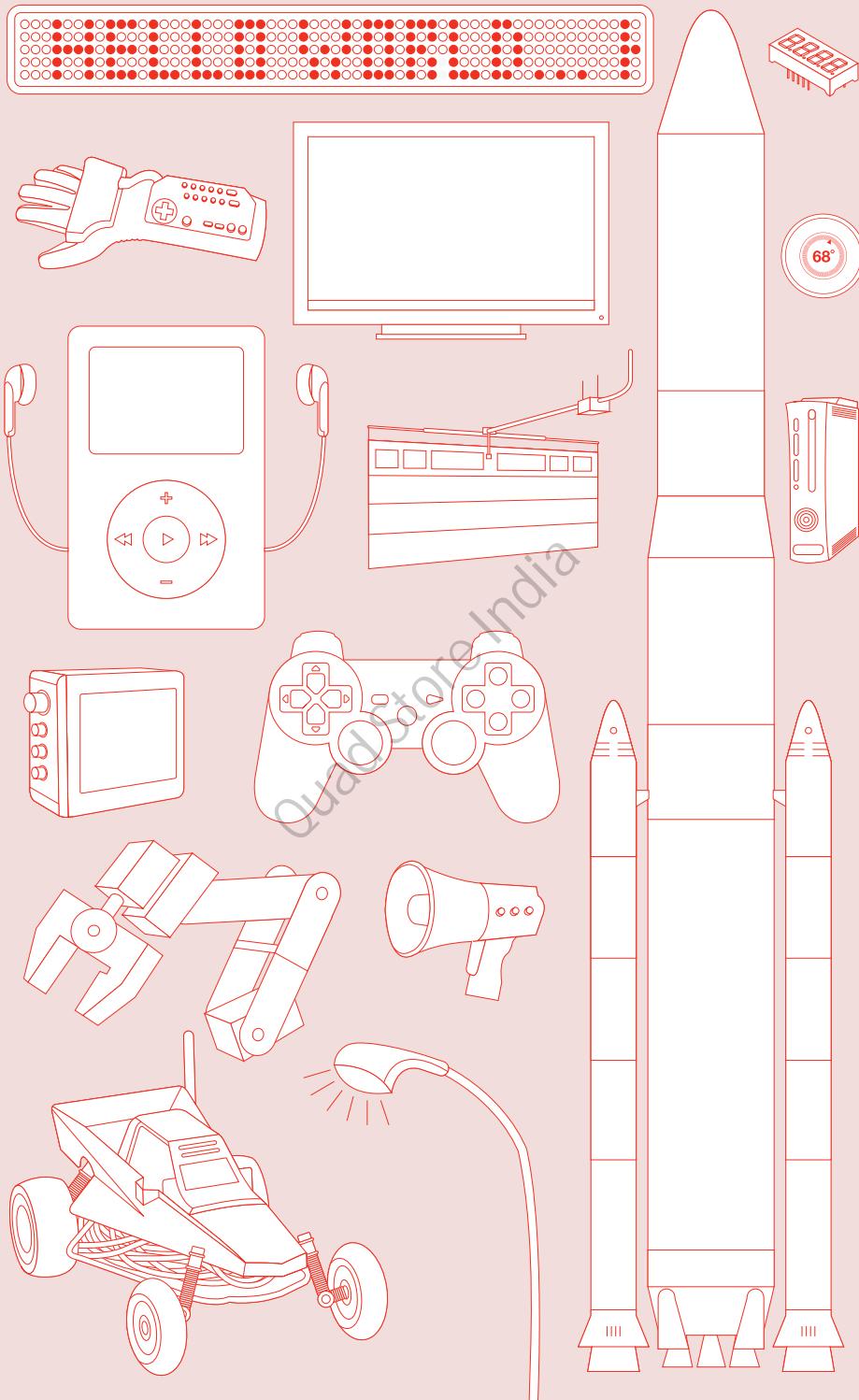
Everywhere you look, you'll find circuits. The cell phone in your pocket, the computer that controls your car's emissions system, your video game console - all these things are chock full of circuits. In this guide, you'll experiment with some simple circuits and learn the gist of the world of embedded electronics.



## // Simple and Complex Circuits

In this guide, you will be primarily exploring simple circuits - but that doesn't mean you can't do amazing things with simple tools! When you've finished the SIK, your knowledge of circuits will enable you to explore amazing projects and unleash the power of your imagination.





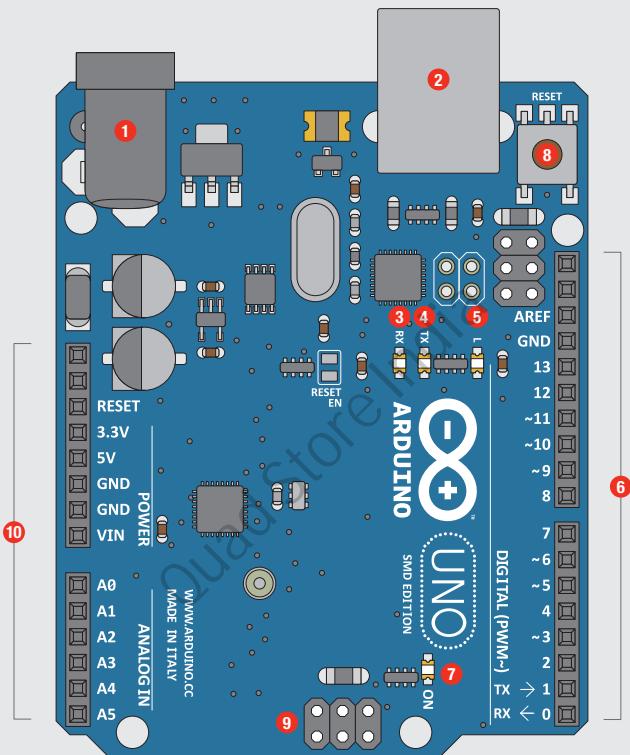
# Inventory of Parts

## Quad Store - Ultimate Arduino Uno R3 Starter to Advance Level Kit

List of components included in the kit

Item No	Description
1	1 x Arduino Uno R3 board + 1 x USB cable
2	1 x GPIO Extension Board + 1 x Connecting Cable
3	1 x Breadboard 830 Tie Points
4	1 x Mini Solderless Prototype Breadboard 170 Tie Point
5	1 x Breadboard Power Supply Module
6	10 x 5mm Red LEDs
7	10 x 5mm Green LEDs
8	10 x 5mm Yellow LEDs
9	10 x 5mm Blue LEDs
10	1 x 5mm RGB LED
11	5 x Tactile Push Button Switch
12	1000 numbers - 30 Different types of Resistors
13	1 x 1k potentiometer
14	1 x 5k potentiometer
15	1 x 10k potentiometer
16	1 x 2.54mm Elbow 40 Pin Header
17	1 x 2.54mm Straight 40 Pin Header
18	1 x 5516 Light Dependent Resistor (Photoresistor)
19	1 x SW-520D Vibration Sensor Metal Ball Tilt Shaking Switch
20	1 x Active Buzzer 5V
21	1 x Passive Buzzer 5V
22	1 x MAX7219 Chip
23	1 x 8*8 Dot Matrix Led display
24	1 x 74HC595 8bit Shift Register
25	1 x LM35 Temperature Sensor
26	1 x Infrared Receiver Type 1838
27	1 x Infrared receiver
28	1 x Infrared Transmitter
29	1 x Npn transistor ss8050
30	1 x Infrared Phototransistor(Flame Sensor)
31	1 x Stepper Motor(28BYJ48 5V)
32	1 x Stepper Motor Driver Board
33	1 x SG90 Servo motor
34	1 x 1602 LCD Blue Backlight
35	1x 4 Digit Display Module
36	1x 4*4 Keyboard Module
37	1 x Sound Sensor Module(With "Digital" and "Analog" output)
38	1 x PS2 Joystick module
39	1 x RTC module(Battery Not Included)
40	1 x 2-Channel Relay Module 5V 10A
41	65pcs x Jump Wires
42	40pcs x 20cm Female-Female Dupont wires
43	40pcs x 20cm Male-Female Dupont wires
44	40pcs x 20cm Male-Male Dupont wires
45	1 x 9V Battery Holder(9V Battery not included)
46	1 x Ultrasonic Module HC-SR04 with Bracket Screws
47	1 x HC-SR501 PIR Motion Sensor
48	1 x RC-522 RFID module
49	1 x S50 Blank White Card
50	1 x S50 Key Ring Card
51	Infrared Remote control
52	1 x FREE !!! Storage Compartment Box

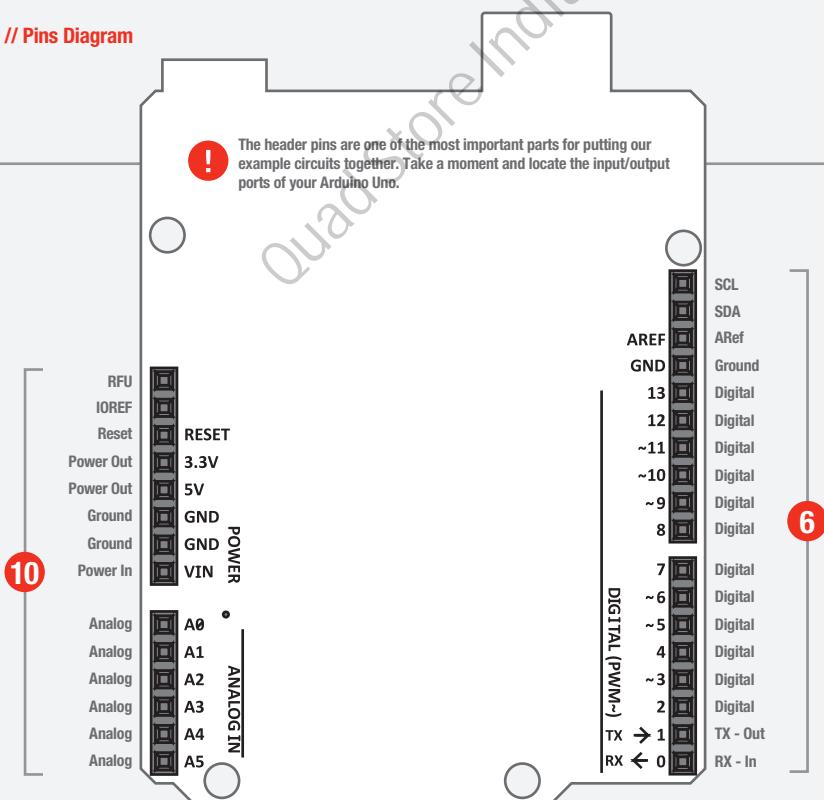
P2N2  
222A  
A18



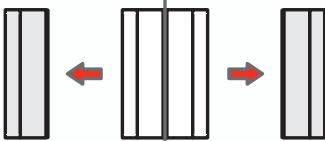
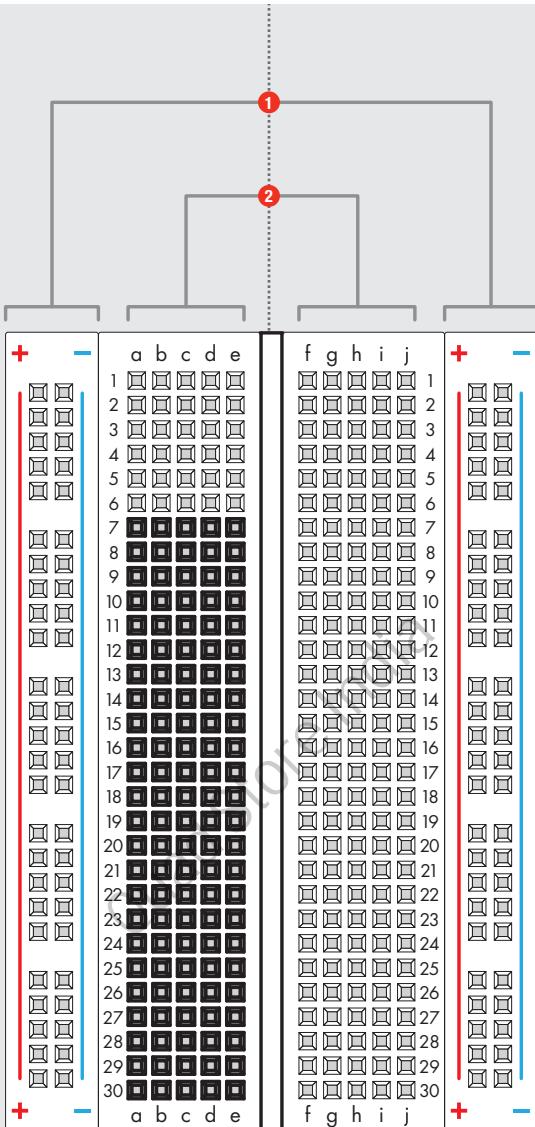
# Arduino Uno

- 1 Power In (Barrel Jack) - Can be used with either a 9V or 12V wall-wart or battery.
- 2 Power In (USB Port) - Provides power and communicates with your board when plugged into your computer via USB.
- 3 LED (RX: Receiving) - This shows when the Arduino is receiving data (such as when being programmed).
- 4 LED (TX: Transmitting) - This shows when your Arduino is transmitting data (such as when running a program).
- 5 LED (Pin 13: Troubleshooting) - This LED is incorporated into your sketch to show if your program is running properly.
- 6 Pins (ARef, Ground, Digital, Rx, Tx) - These various pins can be used for inputs, outputs, power, and ground. // See Diagram Below
- 7 LED (Indicates Arduino is ON) - This is a simple power indicator LED.
- 8 Reset Button - This is a way to manually reset your Arduino, which makes your code restart.
- 9 ICSP Pins (Uploading Code without Bootloader) - This is for "In-Circuit Serial Programming," used if you want to bypass the boot loader.
- 10 Pins (Analog In, Power In, Ground, Power Out, Reset) - These various pins can be used for inputs, outputs, power, and ground. // See Diagram Below

## // Pins Diagram



~ = PWM/Analog out compatible (i.e. ~3)



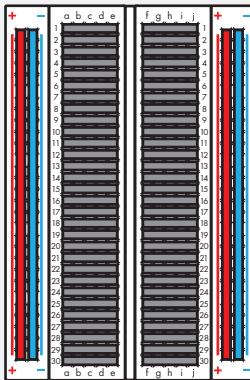
This line divides the board in half, restricting electricity to one half or the other.

# Breadboard

**1** Vertical Connection (+ Power and - Ground // See Diagram Below)

**2** Horizontal Connection (a-e & f-j // See Diagram Below)

How's it all connected?



## Power:

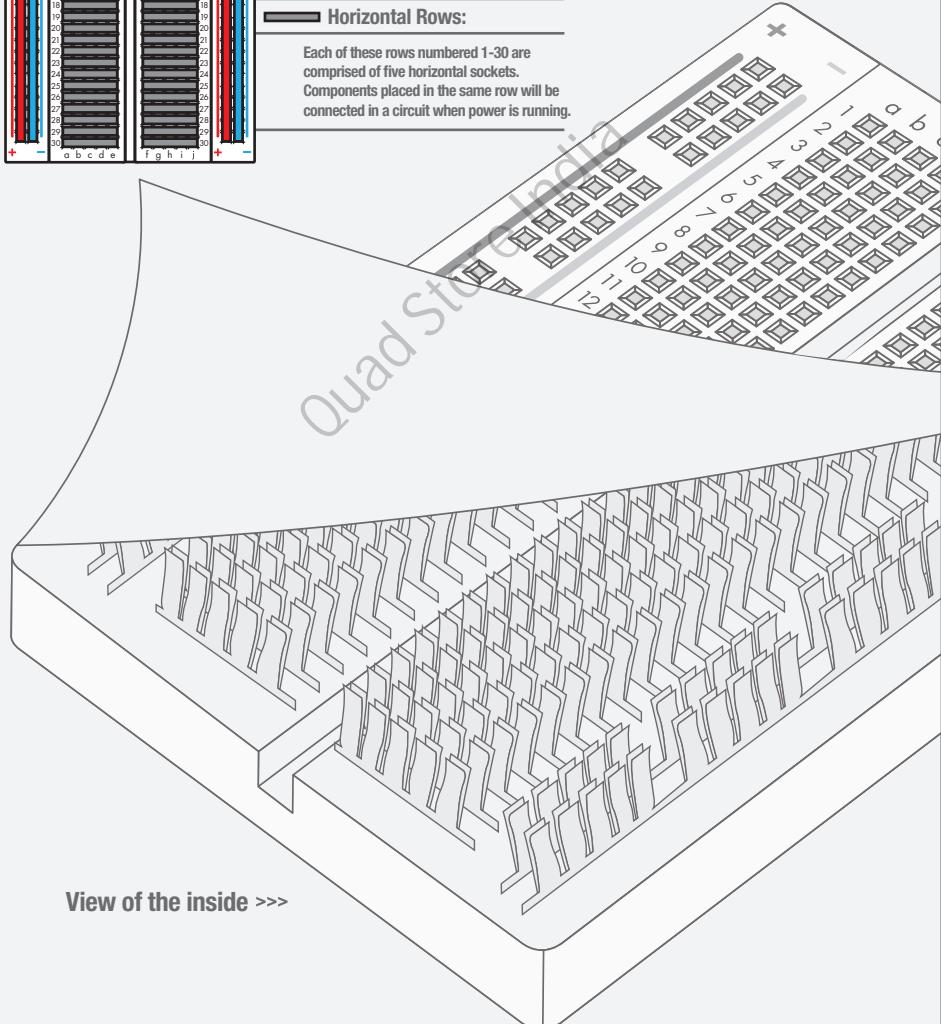
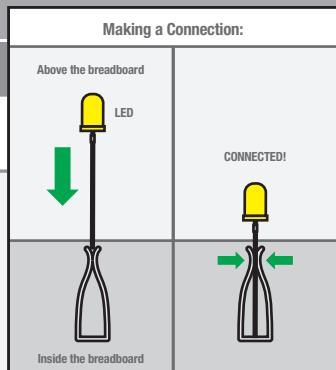
Each + sign runs power anywhere in the vertical column.

## Ground:

Each - sign runs to ground anywhere in the vertical column.

## Horizontal Rows:

Each of these rows numbered 1-30 are comprised of five horizontal sockets. Components placed in the same row will be connected in a circuit when power is running.



View of the inside >>>

## CIRCUIT #1 - Your First Circuit

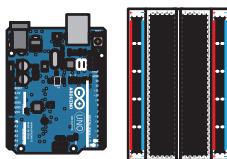
How It Works:

### 1 ASSEMBLE

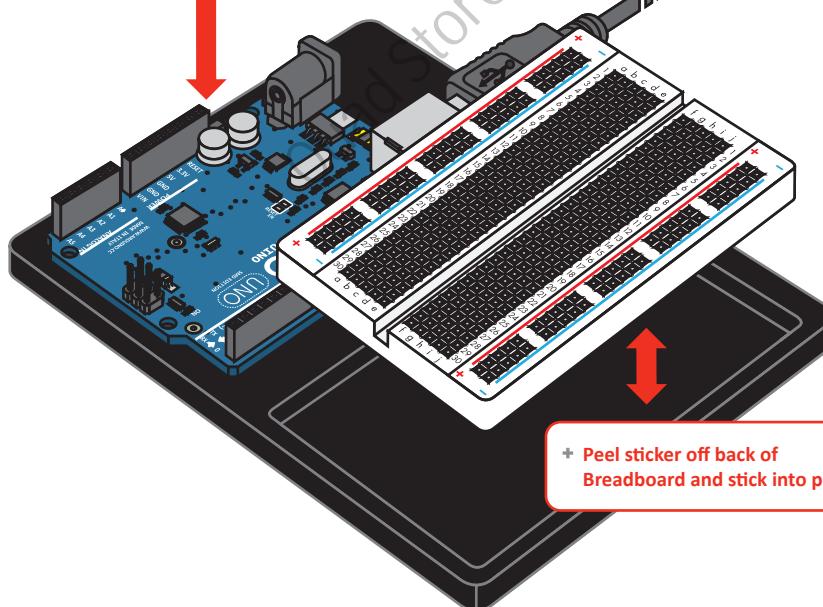
### 2 WRITE

### 3 UPLOAD

- + Make sure the text on the Arduino and Breadboard are facing up so you can read them.



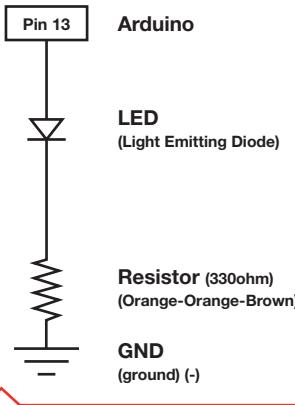
- + Screw the Arduino board down and into place.



- + Connect the USB cable.



**5V Current** Your Arduino runs on five volts. This is the power that will be supplied from your computer via USB and will be the driving force behind any components you use in your circuits. By plugging your Arduino board into your computer, you are supplying it with just the right voltage it needs to thrive! 5V can't hurt you, so don't be afraid to touch anything in your circuit.



+ This is a schematic of your circuit.

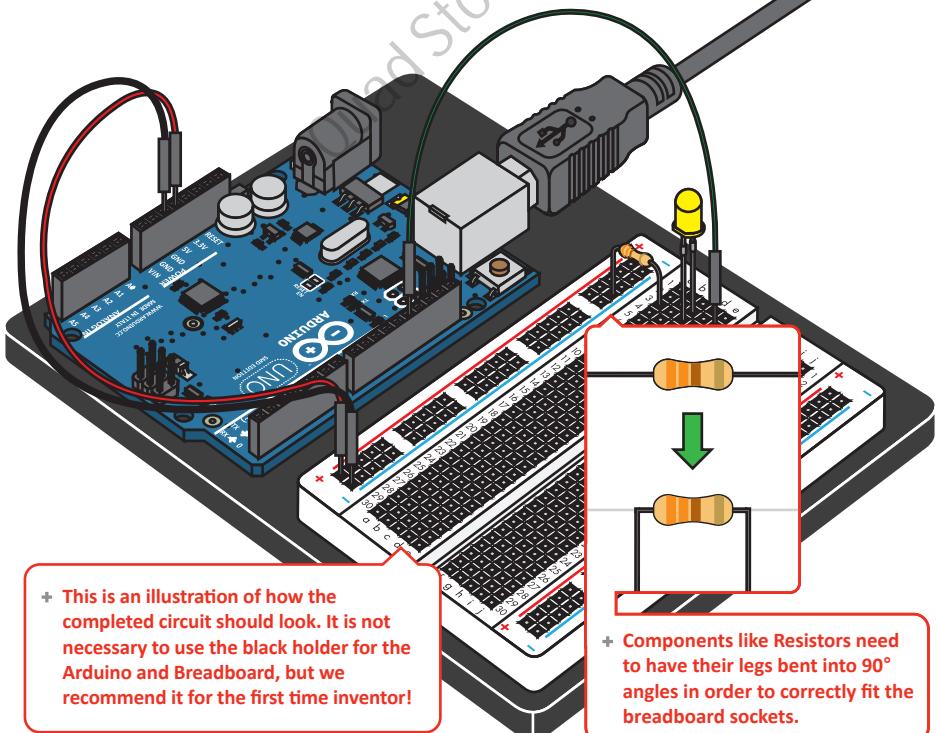
## Blinking a LED

LEDs (light-emitting diodes) are small, powerful lights that are used in many different applications. To start off the SIK, we will work on blinking an LED. That's right - it's as simple as turning a light on and off. It might not seem like much, but establishing this important baseline will give you a solid foundation as we work toward more complex experiments.

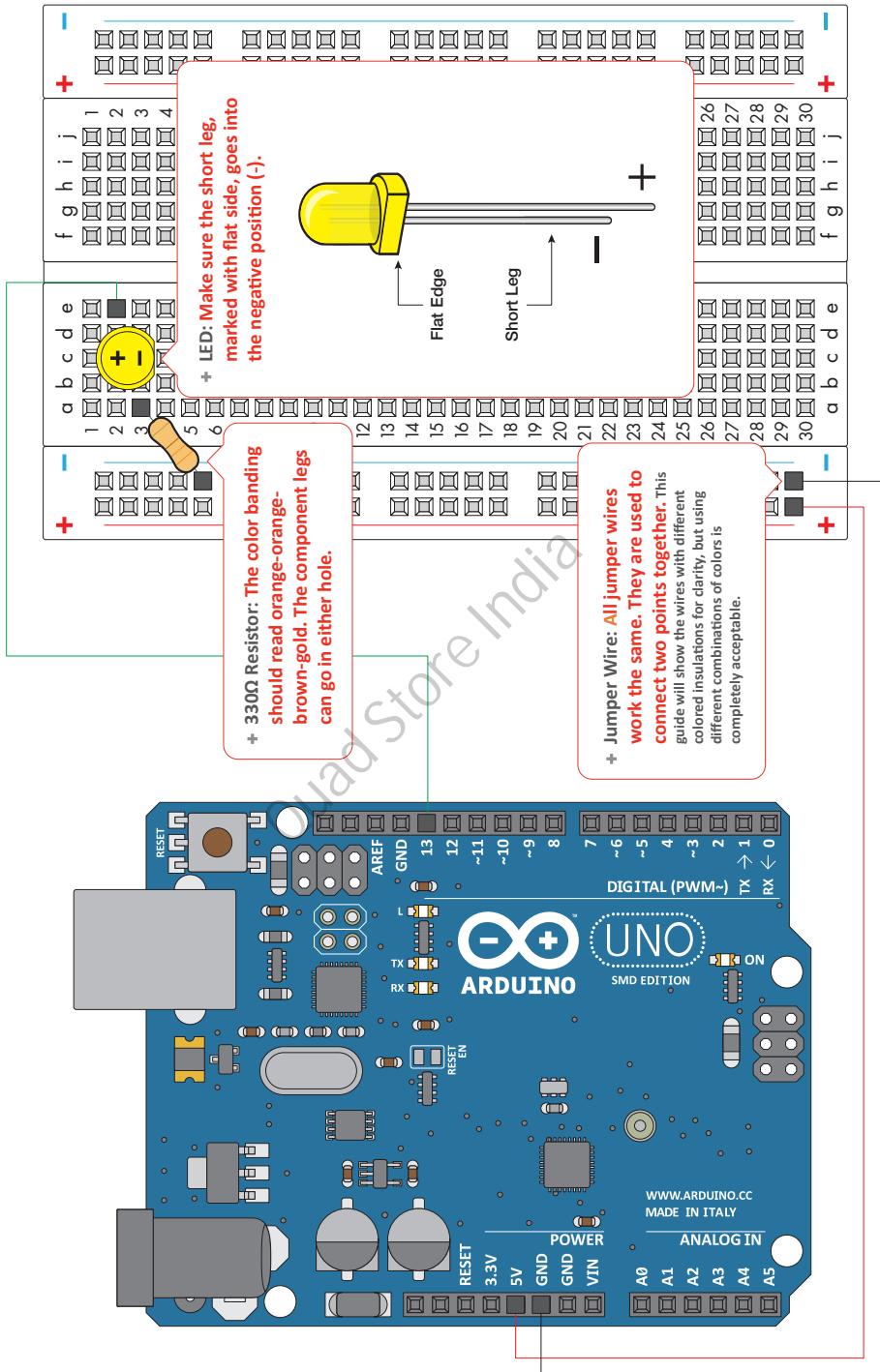
+ Each Circuit begins with a brief description of the what you are putting together and the expected result.



+ This section lists the parts you will need to complete the circuit.



## Circuit 1: Blinking a LED

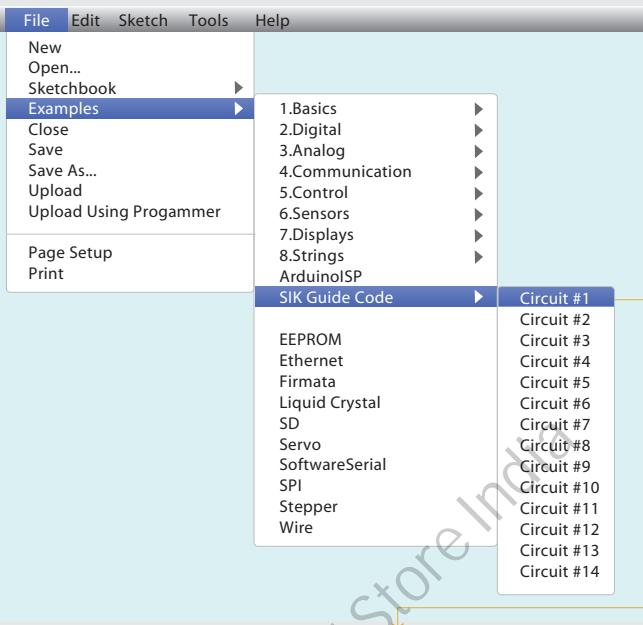


Component:	Image Reference:	
LED (5mm)		c2 - c3 + Components like LEDs are inserted into the Breadboard sockets c2[long leg] / c3[short leg].
330Ω Resistor		a3 - + Resistors are placed in Breadboard sockets only. The “-” symbol represents any socket in its vertical column.
Jumper Wire		GND - + “GND” on the Arduino should be connected to the row marked “-” on the Breadboard.
Jumper Wire		5V + + “5V” on the Arduino connects to the row marked “+” on the Breadboard.
Jumper Wire		Pin 13 e2 + “Pin 13” on the Arduino connects to socket “e2” on the Breadboard.
		+ <b>Breadboard:</b> The white background represents a connection to a breadboard socket.
		+ <b>Arduino:</b> The blue background represents a connection to one of the Arduino header pins.



## Open Your First Sketch:

Open Up the Arduino IDE software on your computer. Coding in the Arduino language will control your circuit. Open the code for Circuit 1 by accessing the “SIK Guide Code” you downloaded and placed into your “Example” folder earlier.



// Circuit #1

```
Circuit #1

/*
Blink
Turns on an LED on for one second,
then off for one second, repeatedly.

This example code is in the public domain.
*/

void setup() {
    // initialize the digital pin as an output.
    // Pin 13 has an LED connected on most Arduino boards
    pinMode(13, OUTPUT);
}

void loop() {
    digitalWrite(13, HIGH);      // set the LED on
    delay(1000);                // wait for a second
    digitalWrite(13, LOW);       // set the LED off
    delay(1000);                // wait for a second
}
```



## Verify

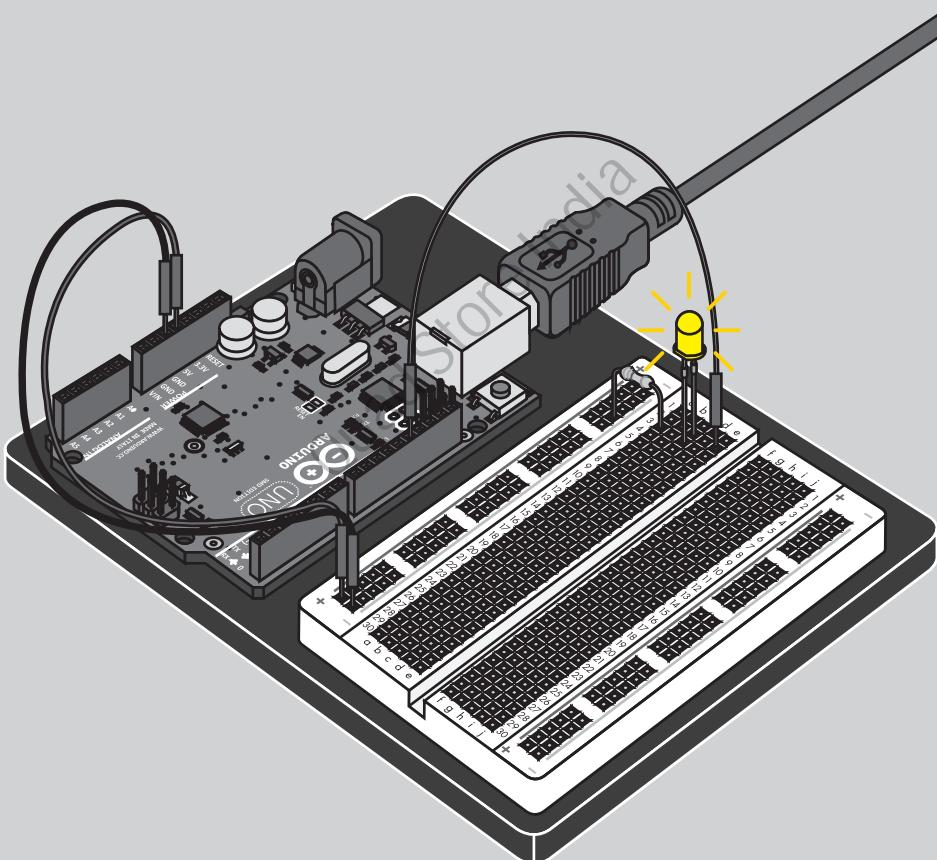
This compiles your code. The IDE changes it from text into instructions the computer can understand.



## Upload

This sends the instructions via the USB cable to the computer chip on the Arduino board. The Arduino will then begin running your code automatically.

*// The result of a completed circuit with correct code after verified and uploaded.*



# 1



**Open Arduino IDE // File > Examples > SIK Guide > Circuit # 1**

**Code to Note:**

- + Begin to understand how the Arduino code works. See below.

- + Remember to Verify and Upload your code.



**pinMode(13, OUTPUT);** →

Before you can use one of the Arduino's pins, you need to tell the Arduino whether it is an INPUT or OUTPUT. We use a built-in "function" called pinMode() to do this.

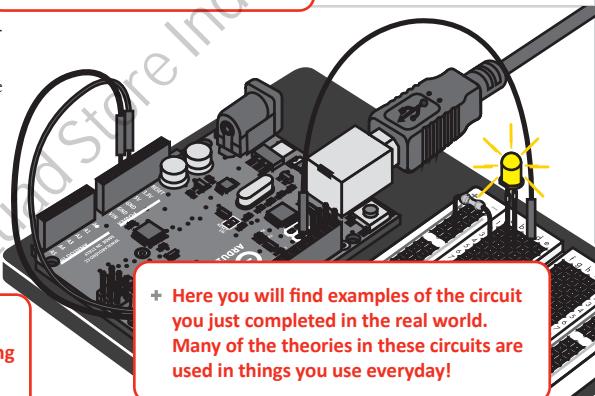
**digitalWrite(13, HIGH);** →

When you're using a pin as an OUTPUT, you can command it to be HIGH (output 5 Volts), or LOW (output 0 Volts).

## What you Should See:

- + See if your circuit is complete and working in this section.

You should see your LED blink on and off. If it isn't, make sure you have assembled the circuit correctly and verified and uploaded the code to your board or see the troubleshooting tips below.



- + This is a section dedicated to the most common mistakes made while assembling the circuit.

- + Here you will find examples of the circuit you just completed in the real world. Many of the theories in these circuits are used in things you use everyday!

## Troubleshooting:

### LED Not Lighting Up?

LEDs will only work in one direction. Try taking it out and twisting it 180 degrees (no need to worry, installing it backwards does no permanent harm).

### Program Not Uploading

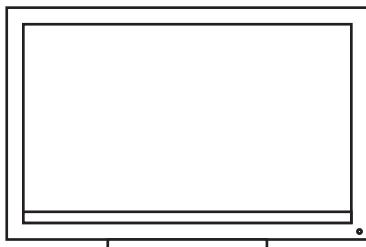
This happens sometimes, the most likely cause is a confused serial port, you can change this in tools>serial port>

### Still No Success?

A broken circuit is no fun, send us an e-mail and we will get back to you as soon as we can: [techsupport@sparkfun.com](mailto:techsupport@sparkfun.com)

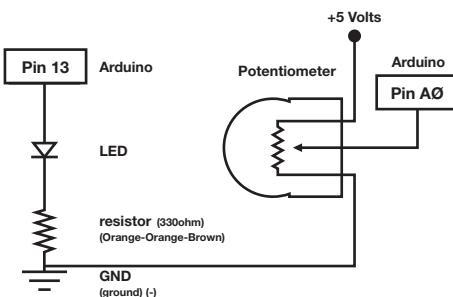
## Real World Application:

Almost all modern flat screen televisions and monitors have LED indicator lights to show they are on or off.



## Potentiometer

In this circuit you'll work with a potentiometer. A potentiometer is also known as a variable resistor. When it's connected with 5 volts across its two outer pins, the middle pin outputs a voltage between 0 and 5, depending on the position of the knob on the potentiometer. In this circuit, you'll learn how to use a potentiometer to control the brightness of an LED.



## PARTS:



x 1



x 1

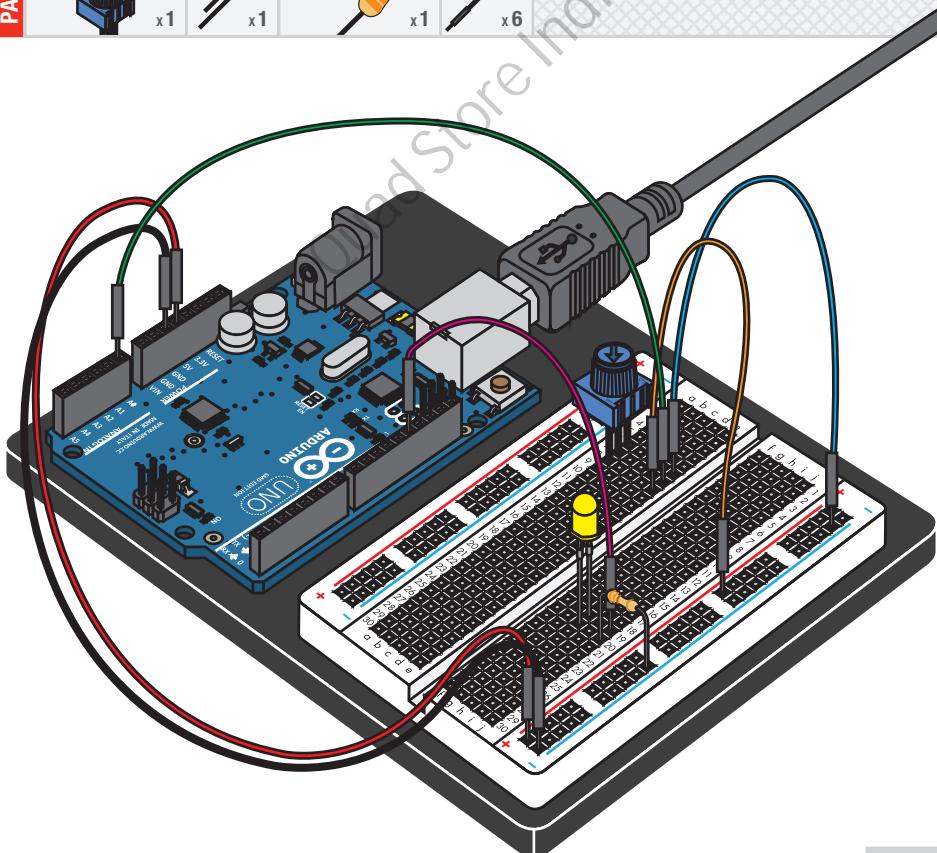


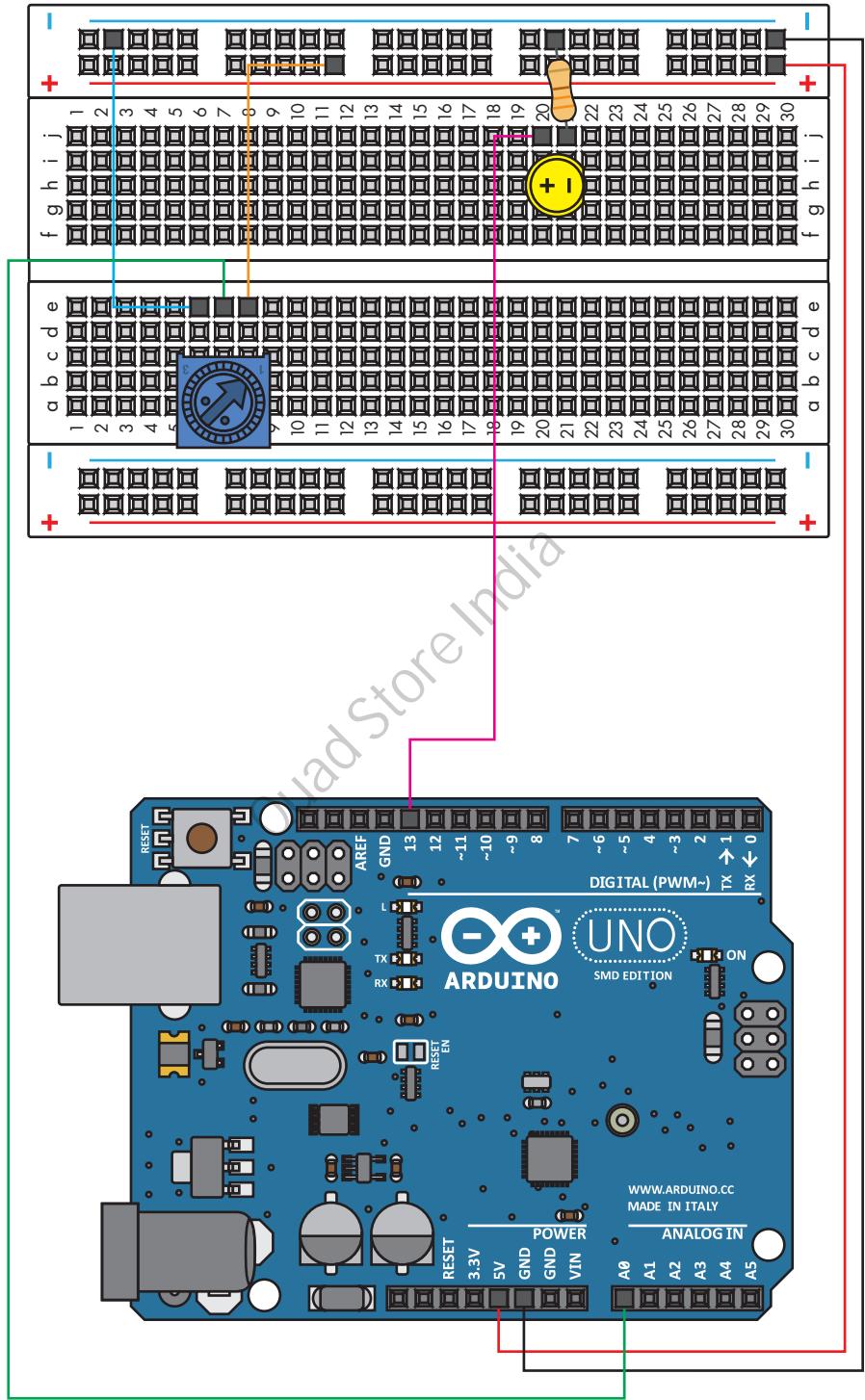
x 1



x 6

ed Store India

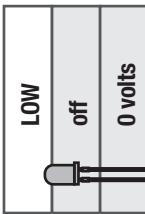
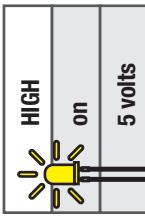




## Digital versus Analog:

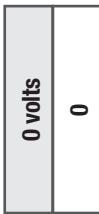
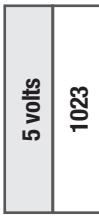
If you look closely at your Arduino, you'll see some pins labeled "DIGITAL", and some labeled "ANALOG". What's the difference?

Many of the devices you'll interface to, such as LEDs and pushbuttons, have only two possible states: on and off, or as they're known to the Arduino, "HIGH" (5 Volts) and "LOW" (0 Volts). The digital pins on an Arduino are great at getting these signals to and from the outside world, and can even do tricks like simulated dimming (by blinking on and off really fast), and serial communications (transferring data to another device by encoding it as patterns of HIGH and LOW).



**DIGITAL**

But there are also a lot of things out there that aren't just "on" or "off". Temperature levels, control knobs, etc. all have a continuous range of values between HIGH and LOW. For these situations, the Arduino offers six analog inputs that translate an input voltage into a number that ranges from 0 (0 Volts) to 1023 (5 Volts). The analog pins are perfect for measuring all those "real world" values, and allow you to interface the Arduino to all kinds of things.



**ANALOG**

Component	Image Reference:	Icon	Icon
Potentiometer			
LED (5mm)			
330Ω Resistor			
Jumper Wire			



**Open Arduino IDE // File > Examples > SIK Guide > Circuit # 2**

Code to Note:



**int sensorValue;**



A "variable" is a number you've given a name to. You must introduce, or "declare" variables before you use them; here we're declaring a variable called `sensorValue`, of type "int" (integer). Don't forget that variable names are case-sensitive!

**sensorValue = analogRead(sensorPin);**



We use the `analogRead()` function to read the value on an analog pin. `analogRead()` takes one parameter, the analog pin you want to use ("sensorPin"), and returns a number ("sensorValue") between 0 (0 Volts) and 1023 (5 Volts).

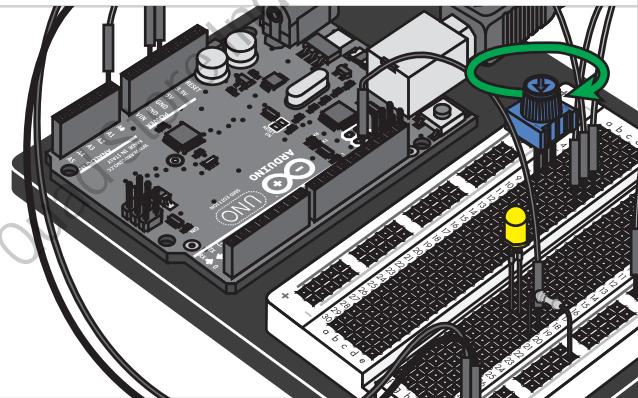
**delay(sensorValue);**



The Arduino is very very fast, capable of running thousands of lines of code each second. To slow it down so that we can see what it's doing, we'll often insert delays into the code. `Delay()` counts in milliseconds; there are 1000 ms in one second.

## What you Should See:

You should see the LED blink faster or slower in accordance with your potentiometer. If it isn't working, make sure you have assembled the circuit correctly and verified and uploaded the code to your board or see the troubleshooting tips below.



## Troubleshooting:

### Sporadically Working

This is most likely due to a slightly dodgy connection with the potentiometer's pins. This can usually be conquered by holding the potentiometer down.

### Not Working

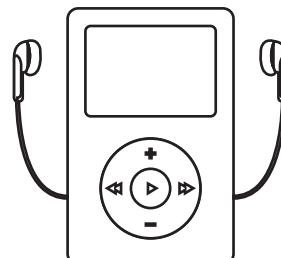
Make sure you haven't accidentally connected the potentiometer's wiper to digital pin 2 rather than analog pin 2. (the row of pins beneath the power pins).

### Still Backward

You can try operating the circuit upside down. Sometimes this helps.

## Real World Application:

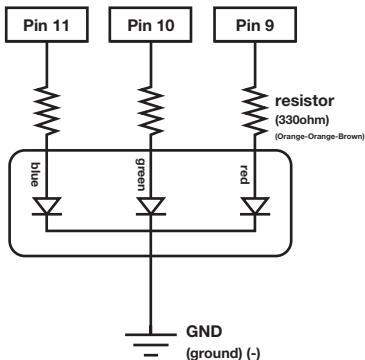
MP3 players' volume control is an example of a potentiometer in action.



## CIRCUIT #3

# 3

### RGB LED



You know what's even more fun than a blinking LED? A colored one. RGB, or red-green-blue, LEDs have three different color-emitting diodes that can be combined to create all sorts of colors. In this circuit, you'll learn how to use an RGB LED to create unique color combinations. Depending on how bright each diode is, nearly any color is possible!

#### PARTS:



x1

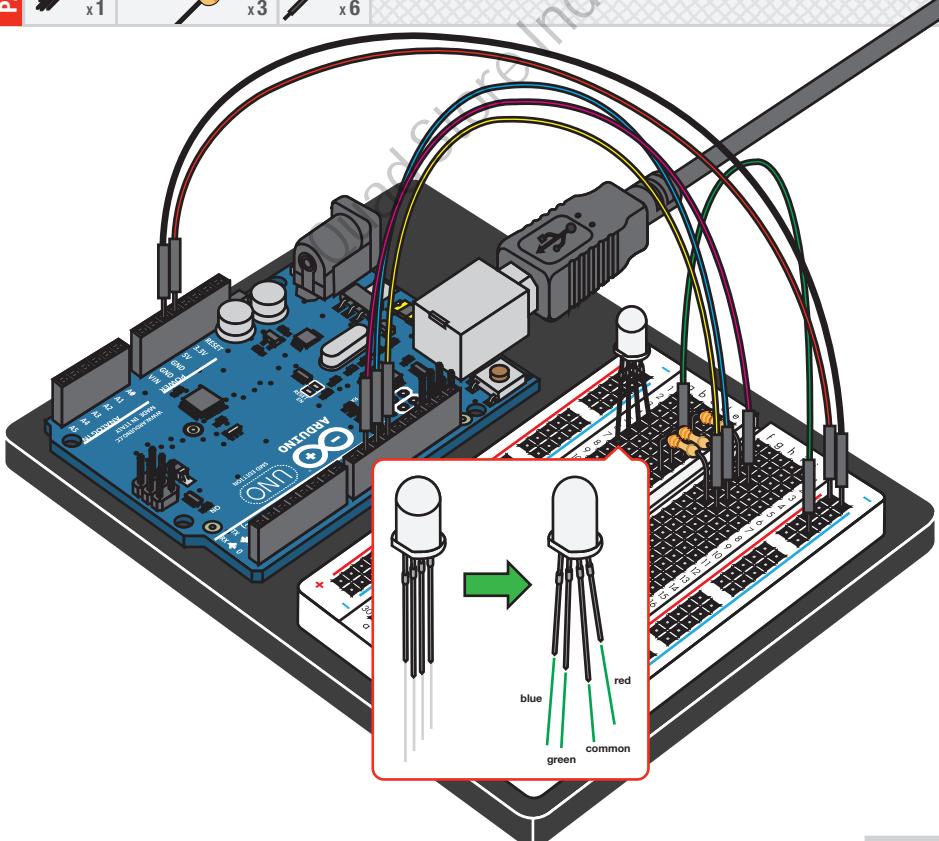


330Ω  
Resistor

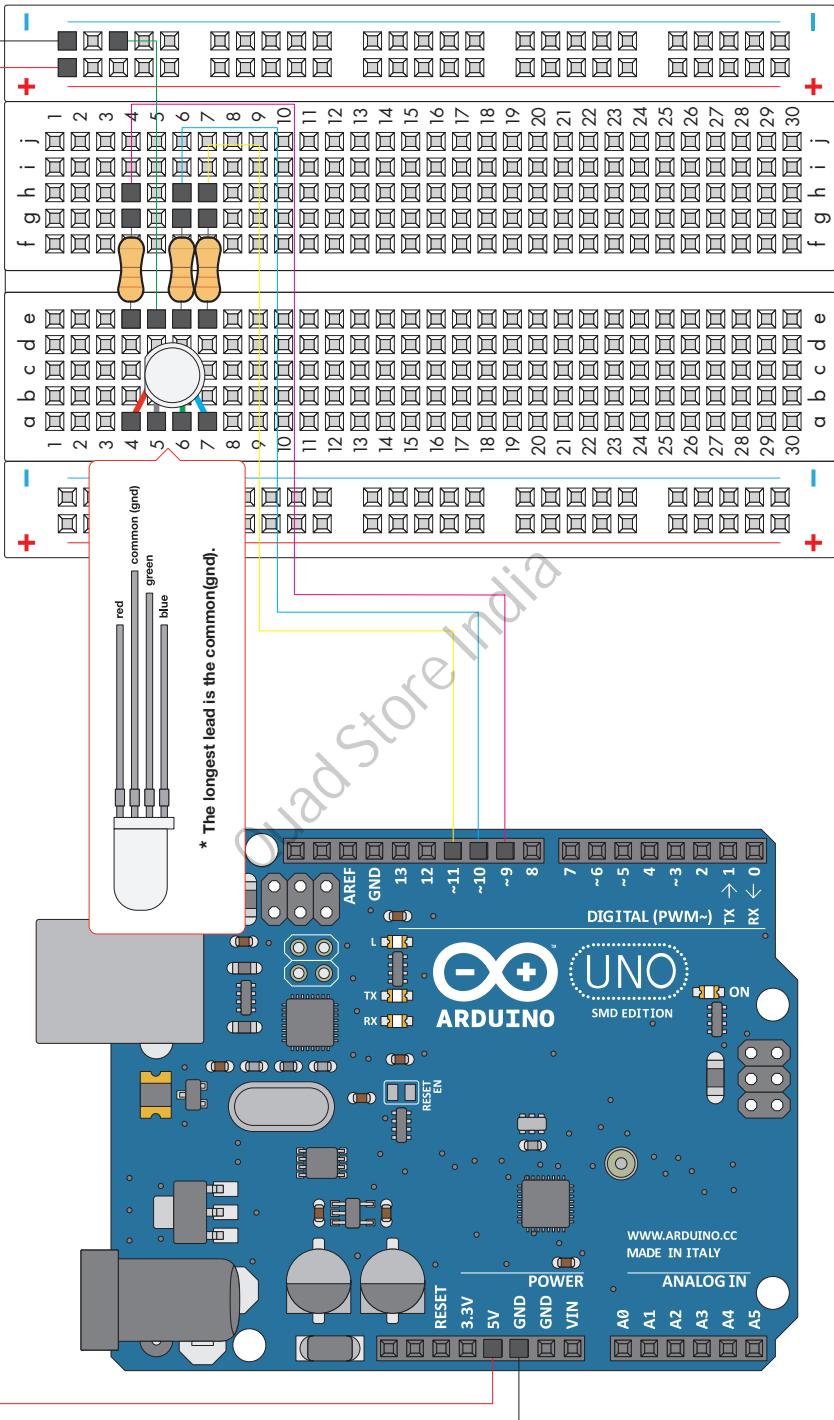


Wire  
x6

ARDUINO India



Circuit 3: RGB LED



The shocking truth behind analogWrite()	
<p>We've seen that the Arduino can read analog voltages (voltages between 0 and 5 Volts) using the <code>analogRead()</code> function. Is there a way for the Arduino to output analog voltages as well?</p> <p>The answer is no... and yes. The Arduino does not have a true analog voltage output. But, because the Arduino is so fast, it can fake it using something called <b>PWM ("Pulse-Width Modulation")</b>.</p> <p>The Arduino is so fast that it can blink a pin on and off almost 1000 times per second. PWM goes one step further by varying the amount of time that the blinking pin spends HIGH vs. the time it spends LOW. If it spends most of its time HIGH, a LED connected to that pin will appear bright. If it spends most of its time LOW, the LED will look dim. Because the pin is blinking much faster than your eye can detect, the Arduino creates the illusion of a "true" analog output.</p>	

Component:	Image Reference:	Diagram
RGB LED (5mm)		
330Ω Resistor		
330Ω Resistor		
330Ω Resistor		
Jumper Wire		
5V		
GND		
Jumper Wire		
Jumper Wire		
Jumper Wire		

# 3



Open Arduino IDE // File > Examples > SIK Guide > Circuit # 3

Code to Note:



**for (x = 0; x < 768; x++)**      ➔

A for() loop is used to step a number across a range, and repeatedly runs code within the brackets {}. Here the variable "x" starts at 0, ends at 767, and increases by one each time ("x++").

**if (x <= 255)**  
{}  
**else**  
{}

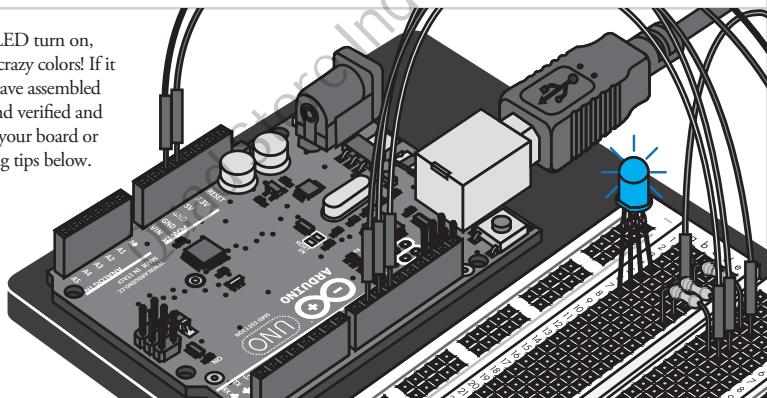
"If / else" statements are used to make choices in your programs. The statement within the parenthesis () is evaluated; if it's true, the code within the first brackets {} will run. If it's not true, the code within the second brackets {} will run.

**delay(sensorValue);**      ➔

The Arduino is very very fast, capable of running thousands of lines of code each second. To slow it down so that we can see what it's doing, we'll often insert delays into the code. Delay() counts in milliseconds; there are 1000 ms in one second.

## What you Should See:

You should see your LED turn on, but this time in new, crazy colors! If it isn't, make sure you have assembled the circuit correctly and verified and uploaded the code to your board or see the troubleshooting tips below.



## Troubleshooting:

### LED Remains Dark or Shows Incorrect Color

With the four pins of the LED so close together, it's sometimes easy to misplace one. Double check each pin is where it should be.

### Seeing Red

The red diode within the RGB LED may be a bit brighter than the other two. To make your colors more balanced, use a higher ohm resistor. Or adjust in code.

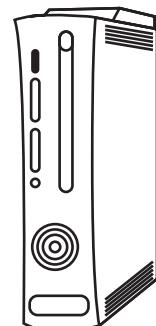
```
analogWrite(RED_PIN, redIntensity);
```

to

```
analogWrite(RED_PIN, redIntensity/3);
```

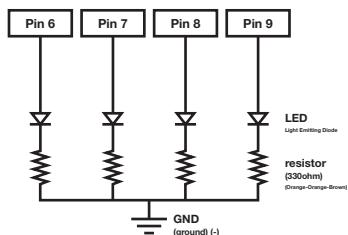
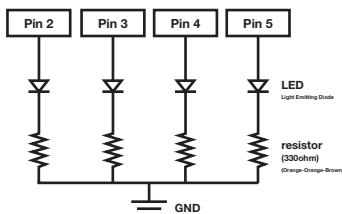
## Real World Application:

Many electronics such as videogame consoles use RGB LEDs to have the versatility to show different colors in the same area. Often times the different colors represent different states of working condition.



## CIRCUIT #4

# 4



## Multiple LEDs

So you have gotten one LED to blink on and off – fantastic! Now it's time to up the stakes a little bit – by connecting EIGHT LEDs AT ONCE. We'll also give our Arduino a little test by creating various lighting sequences. This circuit is a great setup to start practicing writing your own programs and getting a feel for the way Arduino works.

Along with controlling the LEDs, you'll learn about a couple programming tricks that keep your code neat and tidy:

**for() loops** - used when you want to run a piece of code several times

**arrays[ ]** - used to make managing variables easier by grouping them together

### PARTS:



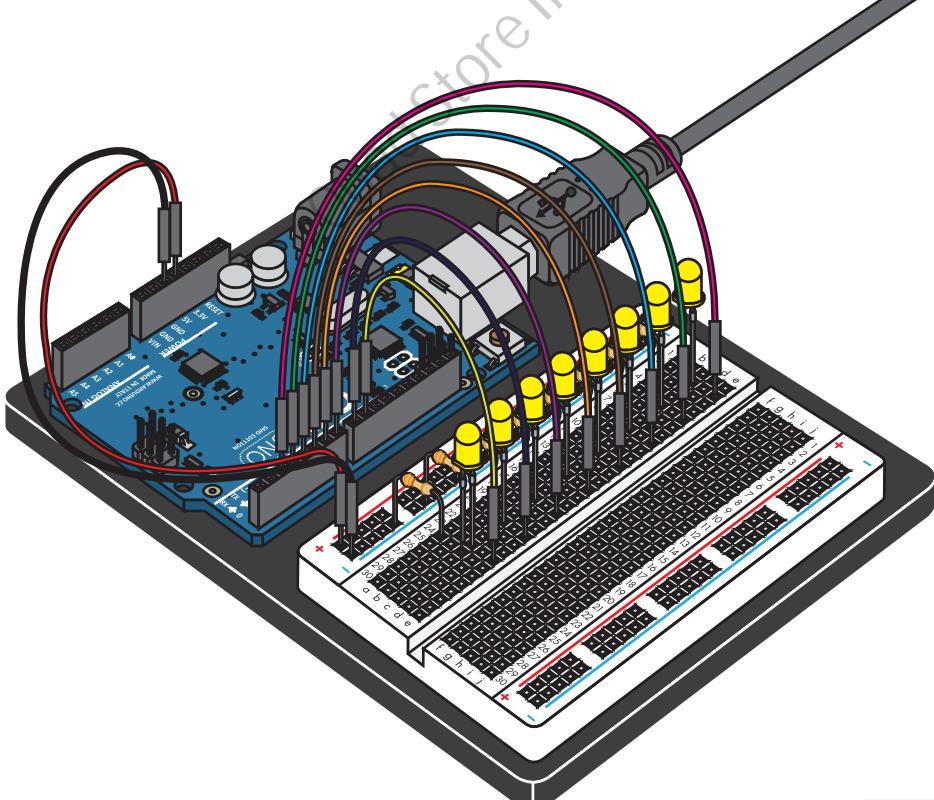
330Ω Resistor

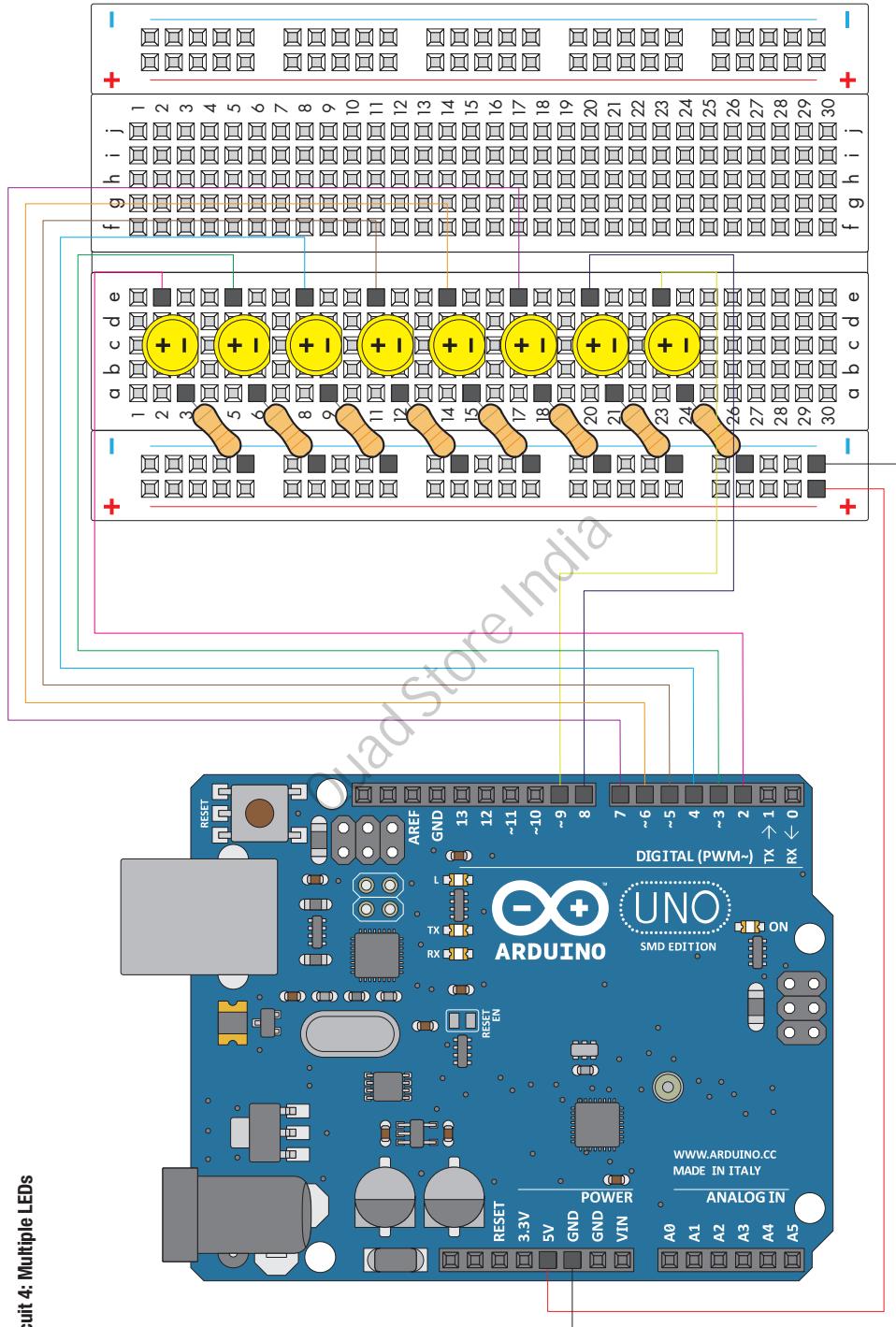


Wire



x 10





Circuit 4: Multiple LEDs

Component:	Image Reference:	Component:	Image Reference:
LED (5mm)		c2   c3 + -	330Ω Resistor 
LED (5mm)		c5   c6 + -	330Ω Resistor 
LED (5mm)		c8   c9 + -	330Ω Resistor 
LED (5mm)		c11   c12 + -	Jumper Wire 
LED (5mm)		c14   c15 + -	Jumper Wire 
LED (5mm)		c17   c18 + -	Jumper Wire 
LED (5mm)		c20   c21 + -	Jumper Wire 
LED (5mm)		c23   c24 + -	Jumper Wire 
330Ω Resistor		c23   c24 + -	Jumper Wire 
330Ω Resistor		GND   a6 + -	Jumper Wire 
330Ω Resistor		GND   a9 + -	Jumper Wire 
330Ω Resistor		GND   a12 + -	Jumper Wire 
330Ω Resistor		GND   a15 + -	Jumper Wire 
			GND 



Code to Note:

`int ledPins[] = {2,3,4,5,6,7,8,9};`

When you have to manage a lot of variables, an "array" is a handy way to group them together. Here we're creating an array of integers, called ledPins, with eight elements.

`digitalWrite(ledPins[0], HIGH);`

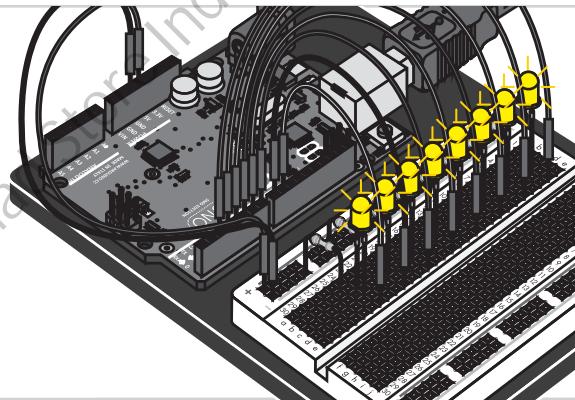
You refer to the elements in an array by their position. The first element is at position 0, the second is at position 1, etc. You refer to an element using "ledPins[x]" where x is the position. Here we're making digital pin 2 HIGH, since the array element at position 0 is "2".

`index = random(8);`

Computers like to do the same things each time they run. But sometimes you want to do things randomly, such as simulating the roll of a dice. The random() function is a great way to do this. See <http://arduino.cc/en/Reference/Random> for more information.

## What you Should See:

This is similar to circuit number one, but instead of one LED, you should see all the LEDs blink. If they aren't, make sure you have assembled the circuit correctly and verified and uploaded the code to your board or see the troubleshooting tips below.



## Troubleshooting:

### Some LEDs Fail to Light

It is easy to insert an LED backwards. Check the LEDs that aren't working and ensure they are the right way around.

### Operating out of sequence

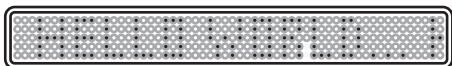
With eight wires it's easy to cross a couple. Double check that the first LED is plugged into pin 2 and each pin thereafter.

### Starting Afresh

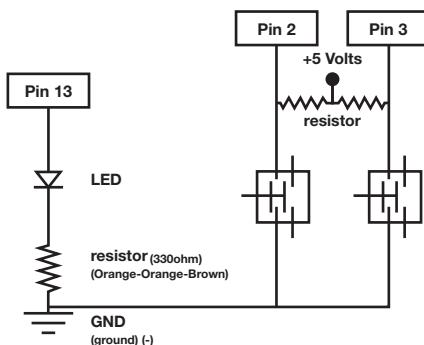
It's easy to accidentally misplace a wire without noticing. Pulling everything out and starting with a fresh slate is often easier than trying to track down the problem.

## Real World Application:

Scrolling marquee displays are generally used to spread short segments of important information. They are built out of many LEDs.



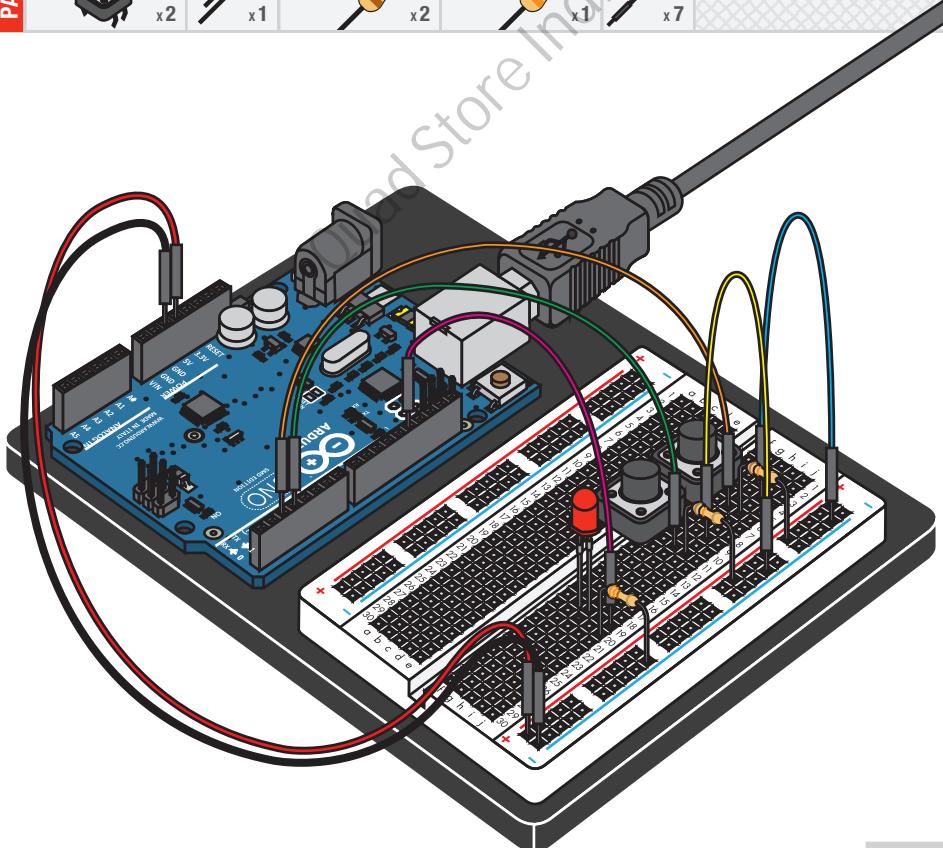
## Push Buttons

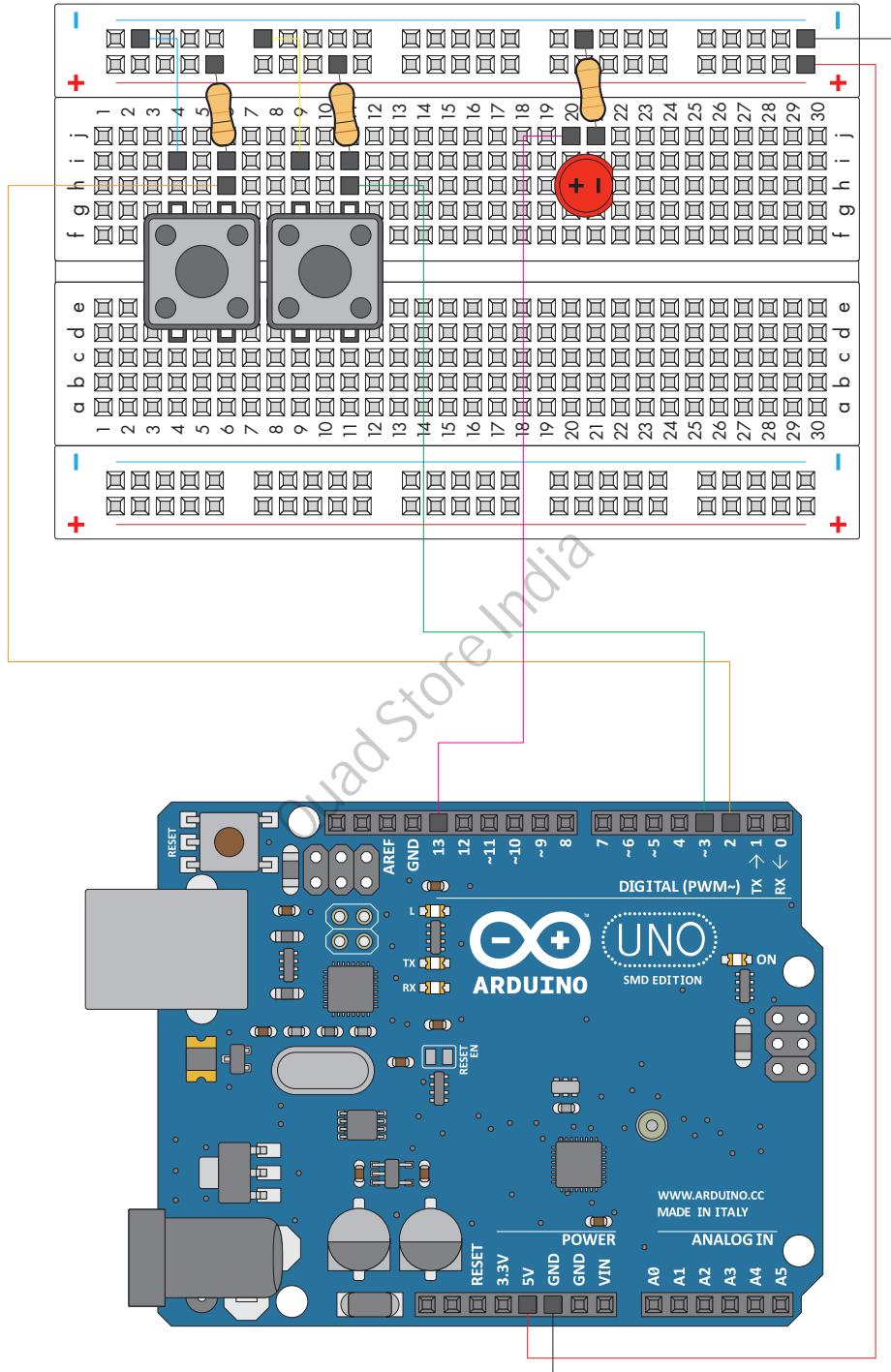


Up until now, we've focused solely on outputs. Now we're going to go to the other end of spectrum and play around with inputs. In this circuit, we'll be looking at one of the most common and simple inputs – a push button. The way a push button works with Arduino is that when the button is pushed, the voltage goes LOW. The Arduino reads this and reacts accordingly. In this circuit, you will also use a pull-up resistor, which helps clean up the voltage and prevents false readings from the button.

**PARTS:**

Push Button		x 2	LED		x 1	10KΩ Resistor		x 2	330Ω Resistor		x 1	Wire		x 7
-------------	--	-----	-----	--	-----	---------------	--	-----	---------------	--	-----	------	--	-----





Circuit 5: Push Buttons

How to use logic like a Vulcan:																	
Component:	Image Reference:																
Push Button																	
Push Button																	
LED (5mm)																	
10KΩ Resistor																	
10KΩ Resistor																	
330Ω Resistor																	
Jumper Wire																	
Jumper Wire																	
Jumper Wire																	
Jumper Wire																	
Jumper Wire																	
One of the things that makes the Arduino so useful is that it can make complex decisions based on the input it's getting. For example, you could make a thermostat that turns on a heater if it gets too cold, a fan if it gets too hot, waters your plants if they get too dry, etc.																	
In order to make such decisions, the Arduino provides a set of logic operations that let you build complex "if" statements. They include:																	
<table border="1"> <thead> <tr> <th></th><th>==</th><th>EQUIVALENCE</th><th>A == B is true if A and B are the SAME.</th></tr> </thead> <tbody> <tr> <td>!=</td><td>DIFFERENCE</td><td>A != B is true if A and B are NOT THE SAME.</td></tr> <tr> <td>&amp;&amp;</td><td>AND</td><td>A &amp;&amp; B is true if BOTH A and B are TRUE.</td></tr> <tr> <td>  </td><td>OR</td><td>A    B is true if A or B or BOTH are TRUE.</td></tr> <tr> <td>!</td><td>NOT</td><td>!A is TRUE if A is FALSE, and FALSE if A is TRUE.</td></tr> </tbody> </table>			==	EQUIVALENCE	A == B is true if A and B are the SAME.	!=	DIFFERENCE	A != B is true if A and B are NOT THE SAME.	&&	AND	A && B is true if BOTH A and B are TRUE.		OR	A    B is true if A or B or BOTH are TRUE.	!	NOT	!A is TRUE if A is FALSE, and FALSE if A is TRUE.
	==	EQUIVALENCE	A == B is true if A and B are the SAME.														
!=	DIFFERENCE	A != B is true if A and B are NOT THE SAME.															
&&	AND	A && B is true if BOTH A and B are TRUE.															
	OR	A    B is true if A or B or BOTH are TRUE.															
!	NOT	!A is TRUE if A is FALSE, and FALSE if A is TRUE.															
You can combine these functions to build complex if() statements.																	
For example:																	
<pre>if ((mode == heat) &amp;&amp; ((temperature &lt; threshold)    (override == true))) {     digitalWrite(HEATER, HIGH); }</pre>																	
...will turn on a heater if you're in heating mode AND the temperature is low, OR if you turn on a manual override. Using these logic operators, you can program your Arduino to make intelligent decisions and take control of the world around it!																	



Code to Note:

`pinMode(button2Pin, INPUT);`

The digital pins can be used as inputs as well as outputs. Before you do either, you need to tell the Arduino which direction you're going.

`button1State = digitalRead(button1Pin);`

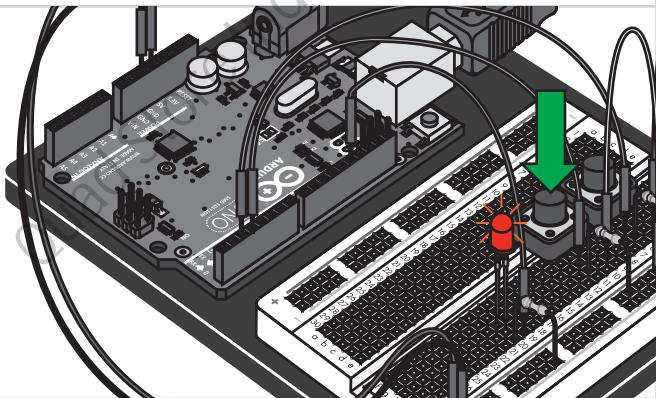
To read a digital input, you use the `digitalRead()` function. It will return HIGH if there's 5V present at the pin, or LOW if there's 0V present at the pin.

`if (button1State == LOW)`

Because we've connected the button to GND, it will read LOW when it's being pressed. Here we're using the "equivalence" operator ("`==`") to see if the button is being pressed.

## What You Should See:

You should see the LED turn on and off as you press the button. If it isn't working, make sure you have assembled the circuit correctly and verified and uploaded the code to your board or see the troubleshooting tips below.



## Troubleshooting:

### Light Not Turning On

The pushbutton is square, and because of this it is easy to put it in the wrong way. Give it a 90 degree twist and see if it starts working.

### Light Not Fading

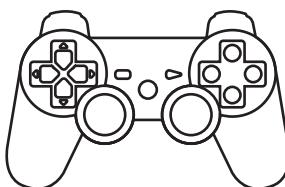
A bit of a silly mistake we constantly made, when you switch from simple on off to fading, remember to move the LED wire from pin 13 to pin 9.

### Underwhelmed

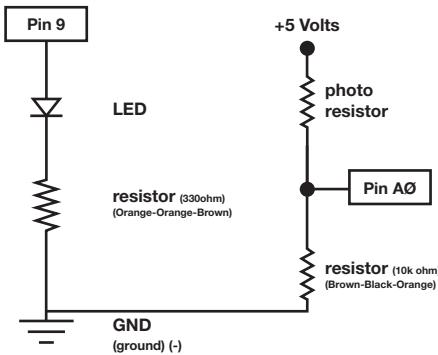
No worries, these circuits are all super stripped down to make playing with the components easy, but once you throw them together the sky is the limit.

## Real World Application:

The buttons we used here are similar to the buttons in most videogame controllers.

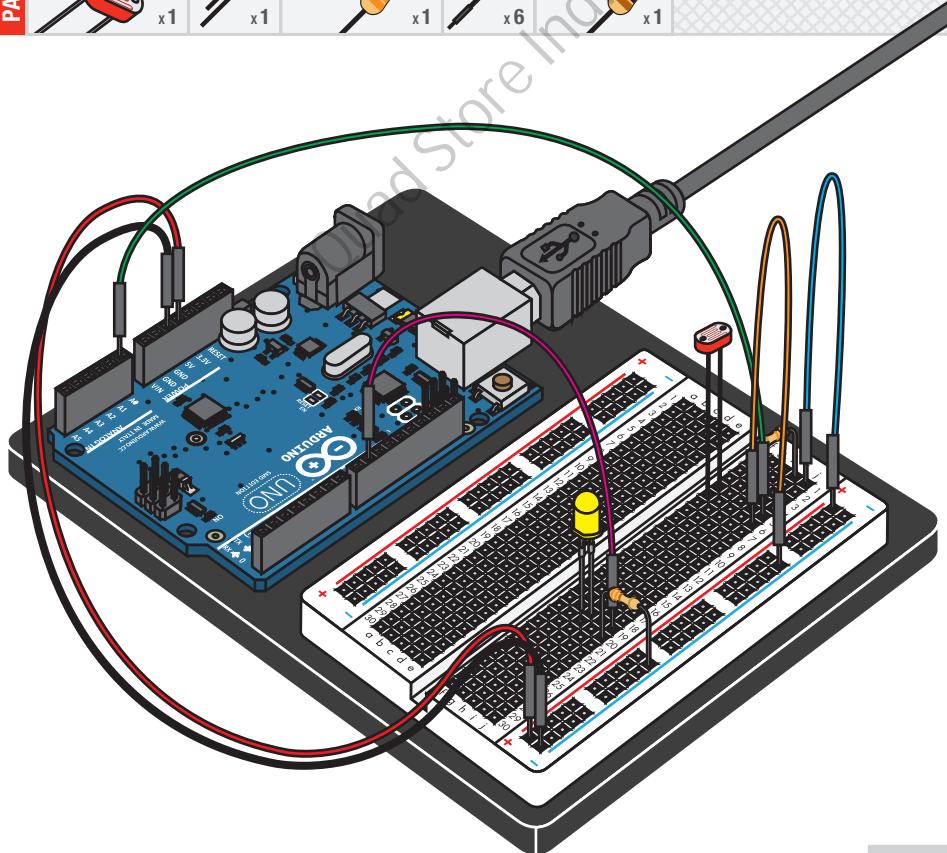


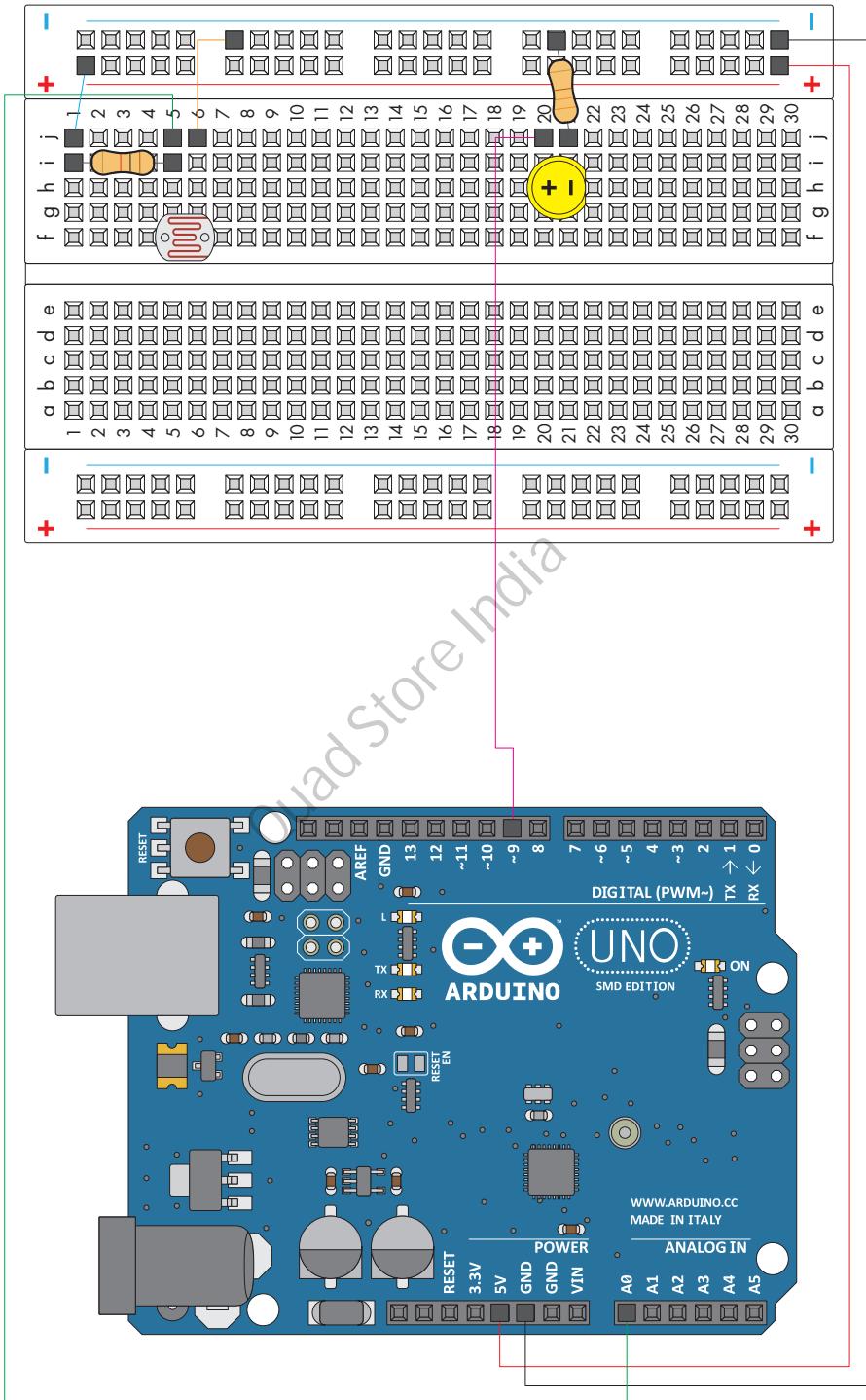
## Photo Resistor



So you've already played with a potentiometer, which varies resistance based on the twisting of a knob. In this circuit, you'll be using a photo resistor, which changes resistance based on how much light the sensor receives. Since the Arduino can't directly interpret resistance (rather it reads voltage), we use a voltage divider to use our photo resistor. This voltage divider will output a high voltage when it is getting a lot of light and a low voltage when it is not.

## PARTS:





Circuit 6 : Photo Resistor

## Measuring resistive sensors:

	<b>Image Reference:</b>	
Photo Resistor		
<b>LED (6mm)</b>		
330Ω Resistor (sensor)		
10kΩ Resistor		
Jumper Wire		



## *Code to Note:*



```
lightLevel = map(lightLevel, 0, 1023, 0, 255);
```

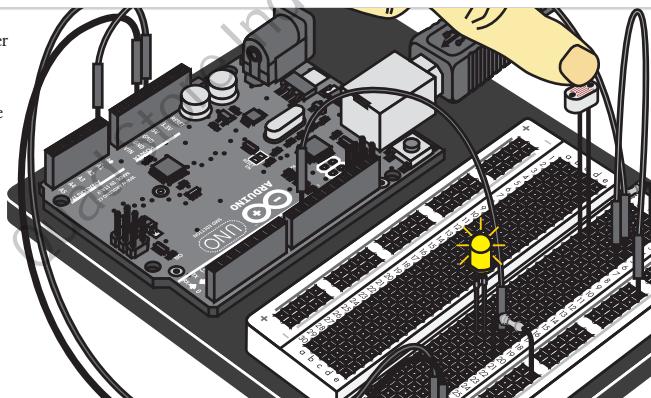
When we read an analog signal using `analogRead()`, it will be a number from 0 to 1023. But when we want to drive a PWM pin using `analogWrite()`, it wants a number from 0 to 255. We can "squeeze" the larger range into the smaller range using the `map()` function.

```
lightLevel = constrain(lightLevel, 0, 255);
```

→ Because map() could still return numbers outside the "to" range, we'll also use a function called constrain() that will "clip" numbers into a range. If the number is outside the range, it will make it the largest or smallest number. If it is within the range, it will stay the same.

## **What You Should See:**

You should see the LED grow brighter or dimmer in accordance with how much light your photoresistor is reading. If it isn't working, make sure you have assembled the circuit correctly and verified and uploaded the code to your board or see the troubleshooting tips below.



## Troubleshooting:

### **LED Remains Dark**

This is a mistake we continue to make time and time again, if only they could make an LED that worked both ways. Pull it up and give it a twist.

### **It Isn't Responding to Changes in Light**

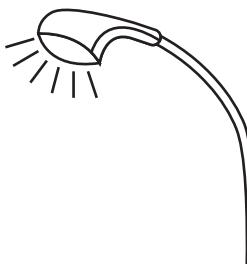
Given that the spacing of the wires on the photo-resistor is not standard, it is easy to misplace it. Double check it's in the right place.

## Still Not Quite Working

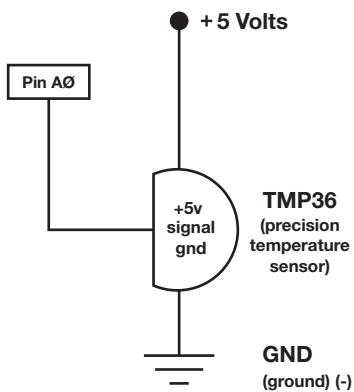
You may be in a room which is either too bright or dark. Try turning the lights on or off to see if this helps. Or if you have a flashlight near by give that a try.

## Real World Application:

A street lamp uses a light sensor to detect when to turn the lights on at night.



## Temperature Sensor

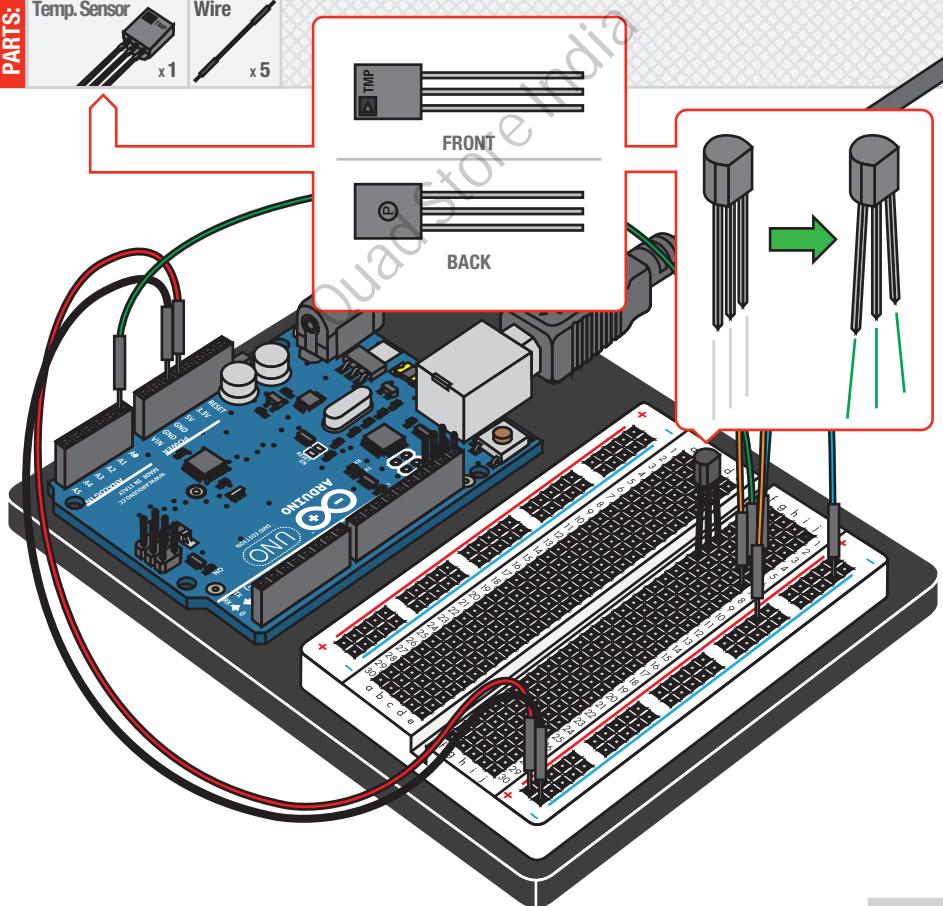


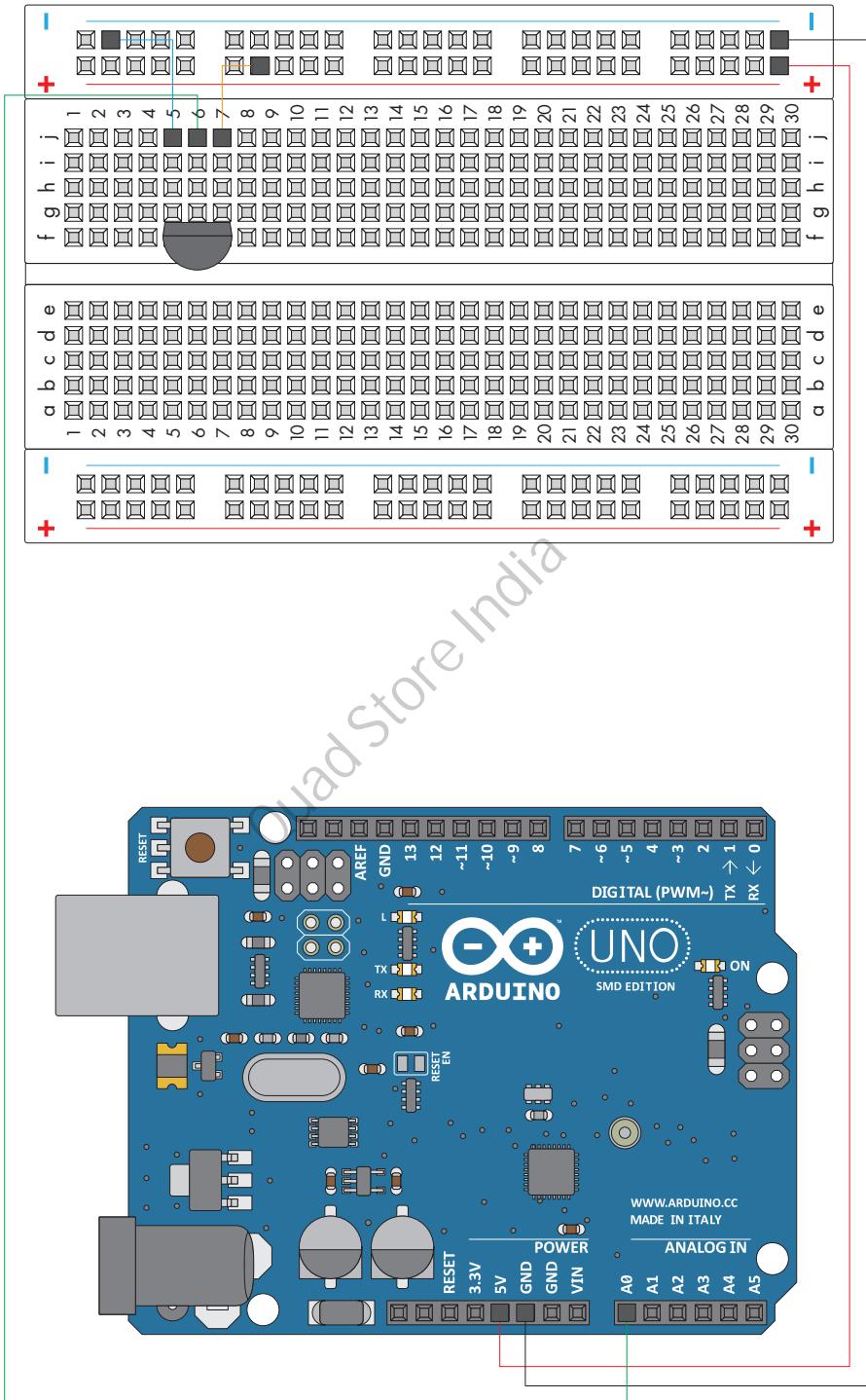
A temperature sensor is exactly what it sounds like – a sensor used to measure ambient temperature. This particular sensor has three pins – a positive, a ground, and a signal. For every centigrade degree it reads, it outputs 10 millivolts. In this circuit, you'll learn how to integrate the temperature sensor with your Arduino, and use the Arduino IDE's debug window to display the temperature.



When you're building the circuit be careful not to mix up the temperature sensor and the transistor, they're almost identical.

## PARTS:

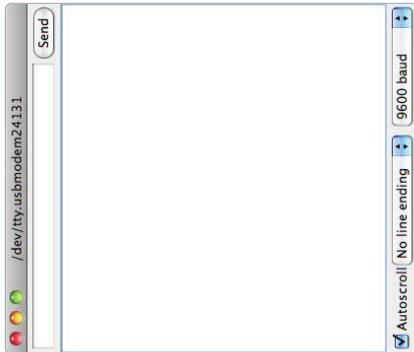
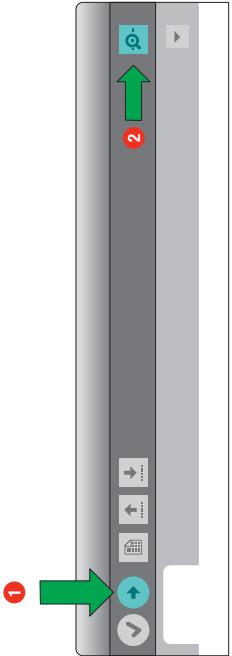




Circuit 7: Temperature Sensor

## Opening your serial monitor:

This circuit uses the Arduino IDE's **serial monitor**. To open this, first upload the program then click the button which looks like a magnifying glass in a square.



Component:	Image Reference:
Temperature Sensor	 
Jumper Wire	
Jumper Wire	
Jumper Wire	
Jumper Wire	
Jumper Wire	
Jumper Wire	
A0	
i6	
i5	
-	
16	
17	
+	
17	
-	
5V	
GND	



## Open Arduino IDE // File > Examples > SIK Guide > Circuit # 7

Code to Note:



`Serial.begin(9600);`



Before using the serial monitor, you must call `Serial.begin()` to initialize it. 9600 is the "baud rate", or communications speed. When two devices are communicating with each other, both must be set to the same speed.

`Serial.print(degreesC);`



The `Serial.print()` command is very smart. It can print out almost anything you can throw at it, including variables of all types, quoted text (AKA "strings"), etc.

*See <http://arduino.cc/en/Serial/Print> for more info.*

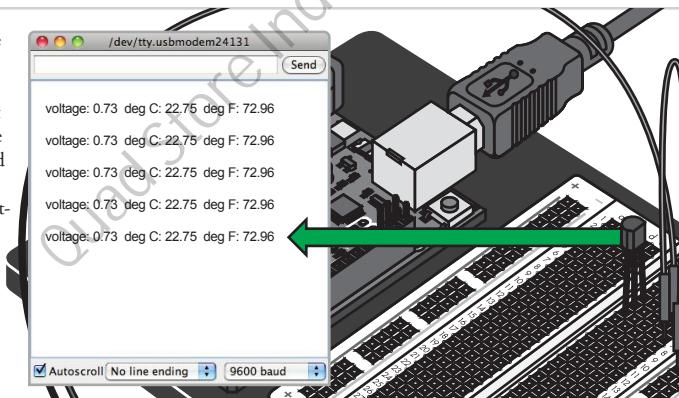
`Serial.println(degreesF);`



`Serial.print()` will print everything on the same line. `Serial.println()` will move to the next line. By using both of these commands together, you can create easy-to-read printouts of text and data.

## What You Should See:

You should see be able to read the temperature your temperature sensor is detecting on the serial monitor in the Arduino IDE. If it isn't working, make sure you have assembled the circuit correctly and verified and uploaded the code to your board or see the troubleshooting tips below.



## Troubleshooting:

### Nothing Seems to Happen

This program has no outward indication it is working. To see the results you must open the Arduino IDE's serial monitor (instructions on previous page).

### Gibberish is Displayed

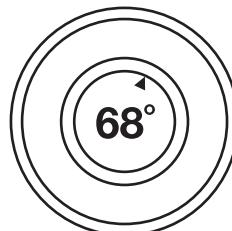
This happens because the serial monitor is receiving data at a different speed than expected. To fix this, click the pull-down box that reads "\*\*\* baud" and change it to "9600 baud".

### Temperature Value is Unchanging

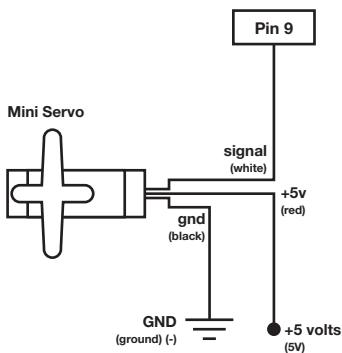
Try pinching the sensor with your fingers to heat it up or pressing a bag of ice against it to cool it down.

## Real World Application:

Building climate control systems use a temperature sensor to monitor and maintain their settings.

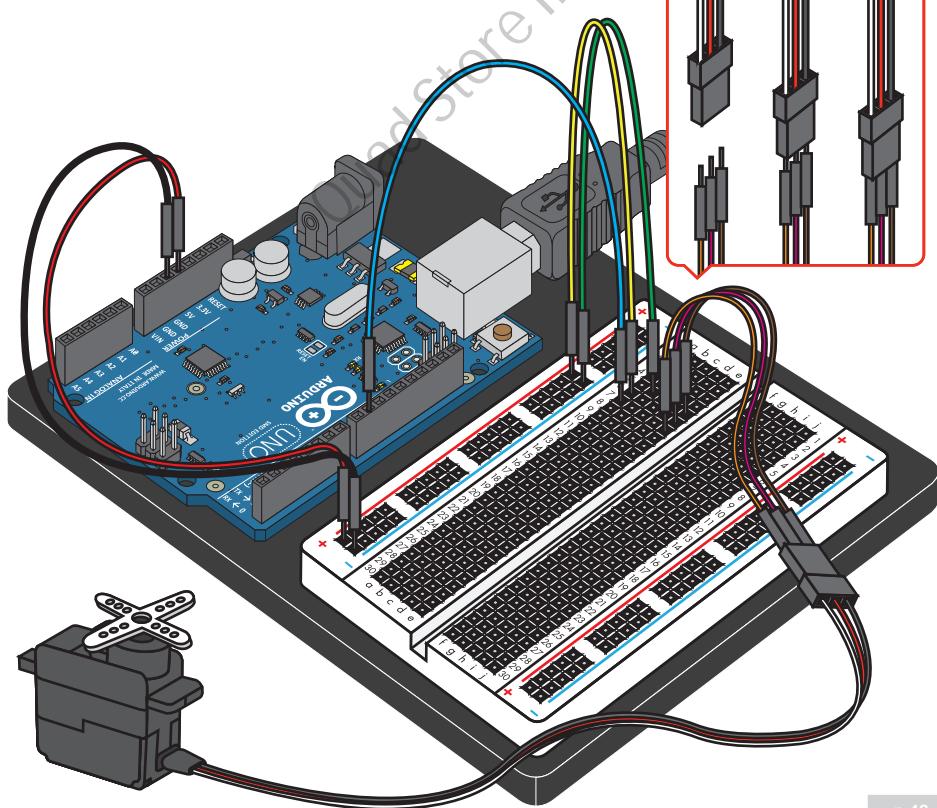


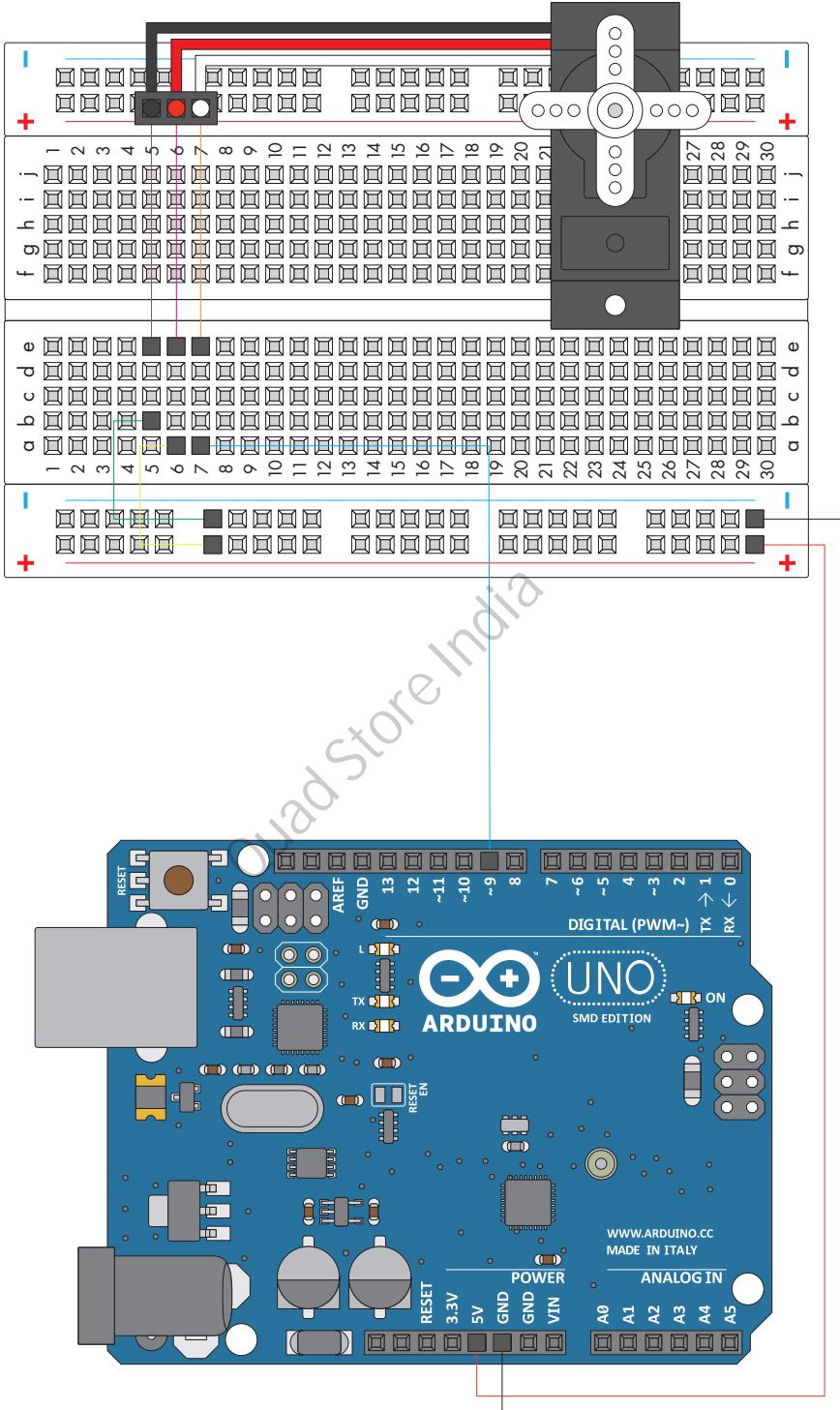
## A Single Servo



Servos are ideal for embedded electronics applications because they do one thing very well that spinning motors cannot – they can move to a position accurately. By varying the pulse of voltage a servo receives, you can move a servo to a specific position. For example, a pulse of 1.5 milliseconds will move the servo 90 degrees. In this circuit, you'll learn how to use PWM (pulse width modulation) to control and rotate a servo.

## PARTS:





## Circuit 8: A Single Servo

## Expand your horizons using Libraries:



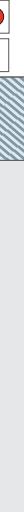
Servo



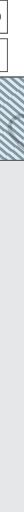
Jumper Wire



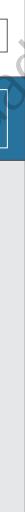
Jumper Wire



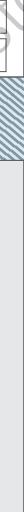
Jumper Wire



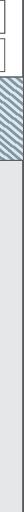
Jumper Wire



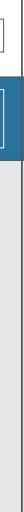
Jumper Wire



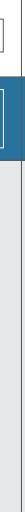
Jumper Wire



Jumper Wire



Jumper Wire

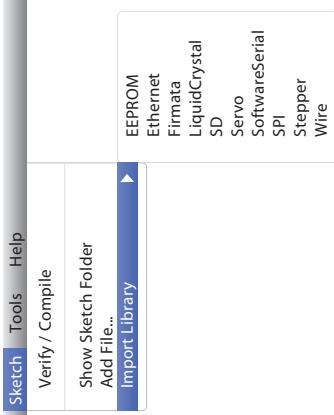


Jumper Wire

See <http://arduino.cc/en/Reference/Libraries> for a list of the standard libraries and information on using them.

But anyone can create a library, and if you want to use a new sensor or output device, chances are that someone out there has already written one that interfaces that device to the Arduino. Many of SparkFun's products come with Arduino libraries, and you can find even more using Google and the Arduino Playground at <http://arduino.cc/playground/>. And when YOU get the Arduino working with a new device, consider making a library for it and sharing it with the world!

To use a library in a sketch, select it from Sketch > Import Library.





## Open Arduino IDE // File > Examples > SIK Guide > Circuit #8

Code to Note:



**#include <Servo.h>**



#include is a special "preprocessor" command that inserts a library (or any other file) into your sketch. You can type this command yourself, or choose an installed library from the "sketch / import library" menu.

**Servo servo1;**



**servo1.attach(9);**

The servo library adds new commands that let you control a servo. To prepare the Arduino to control a servo, you must first create a Servo "object" for each servo (here we've named it "servo1"), and then "attach" it to a digital pin (here we're using pin 9).

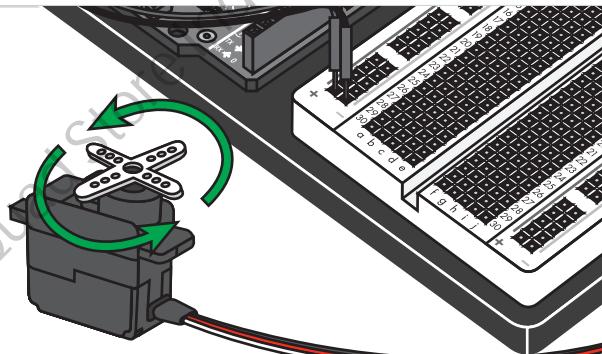
**servo1.write(180);**



Servos don't spin all the way around, but they can be commanded to move to a specific position. We use the servo library's write() command to move a servo to a specified number of degrees(0 to 180). Remember that the servo requires time to move, so give it a short delay() if necessary.

## What You Should See:

You should see your servo motor move to various locations at several speeds. If the motor doesn't move, check your connections and make sure you have verified and uploaded the code, or see the troubleshooting tips below.



## Troubleshooting:

### Servo Not Twisting

Even with colored wires it is still shockingly easy to plug a servo in backward. This might be the case.

### Still Not Working

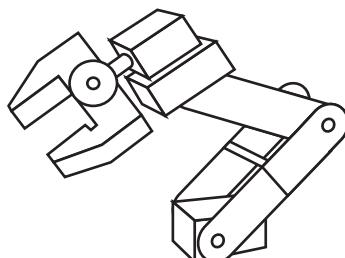
A mistake we made a time or two was simply forgetting to connect the power (red and brown wires) to +5 volts and ground.

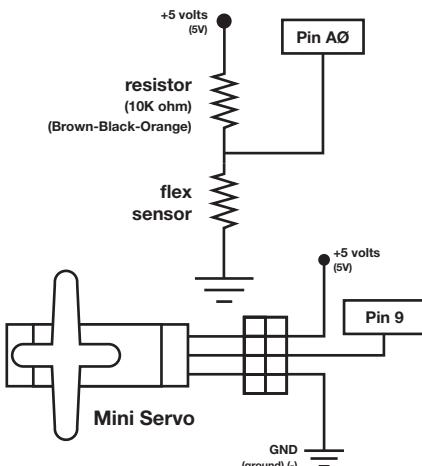
### Fits and Starts

If the servo begins moving then twitches, and there's a flashing light on your Arduino board, the power supply you are using is not quite up to the challenge. Using a wall adapter instead of USB should solve this problem.

## Real World Application:

Robotic arms you might see in an assembly line or sci-fi movie probably have servos in them.

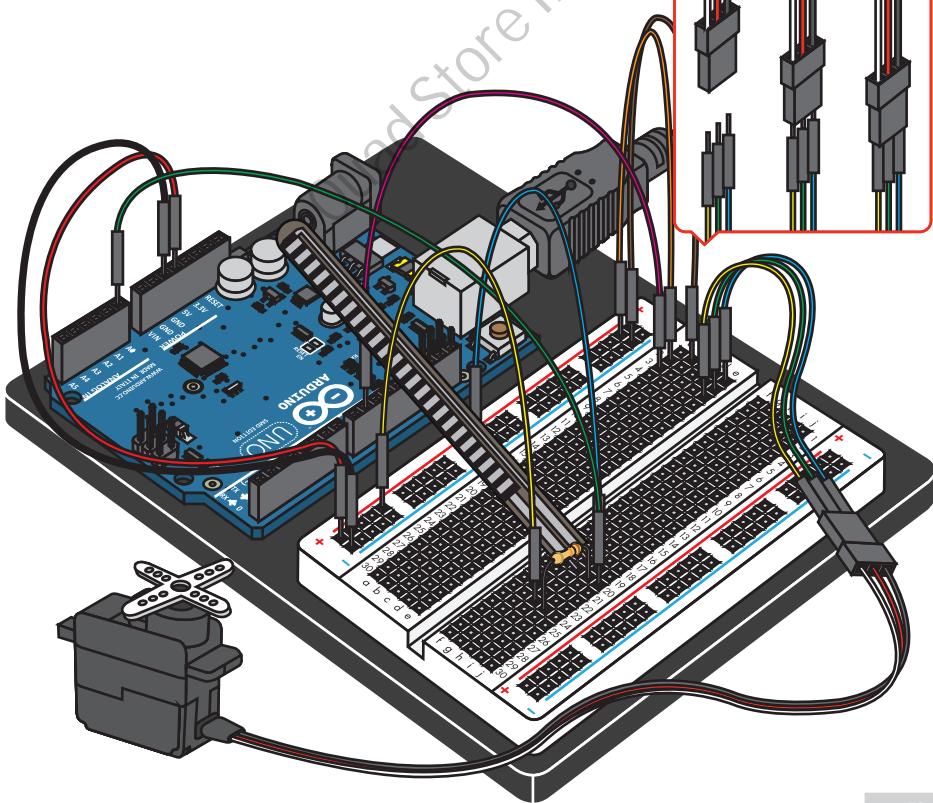


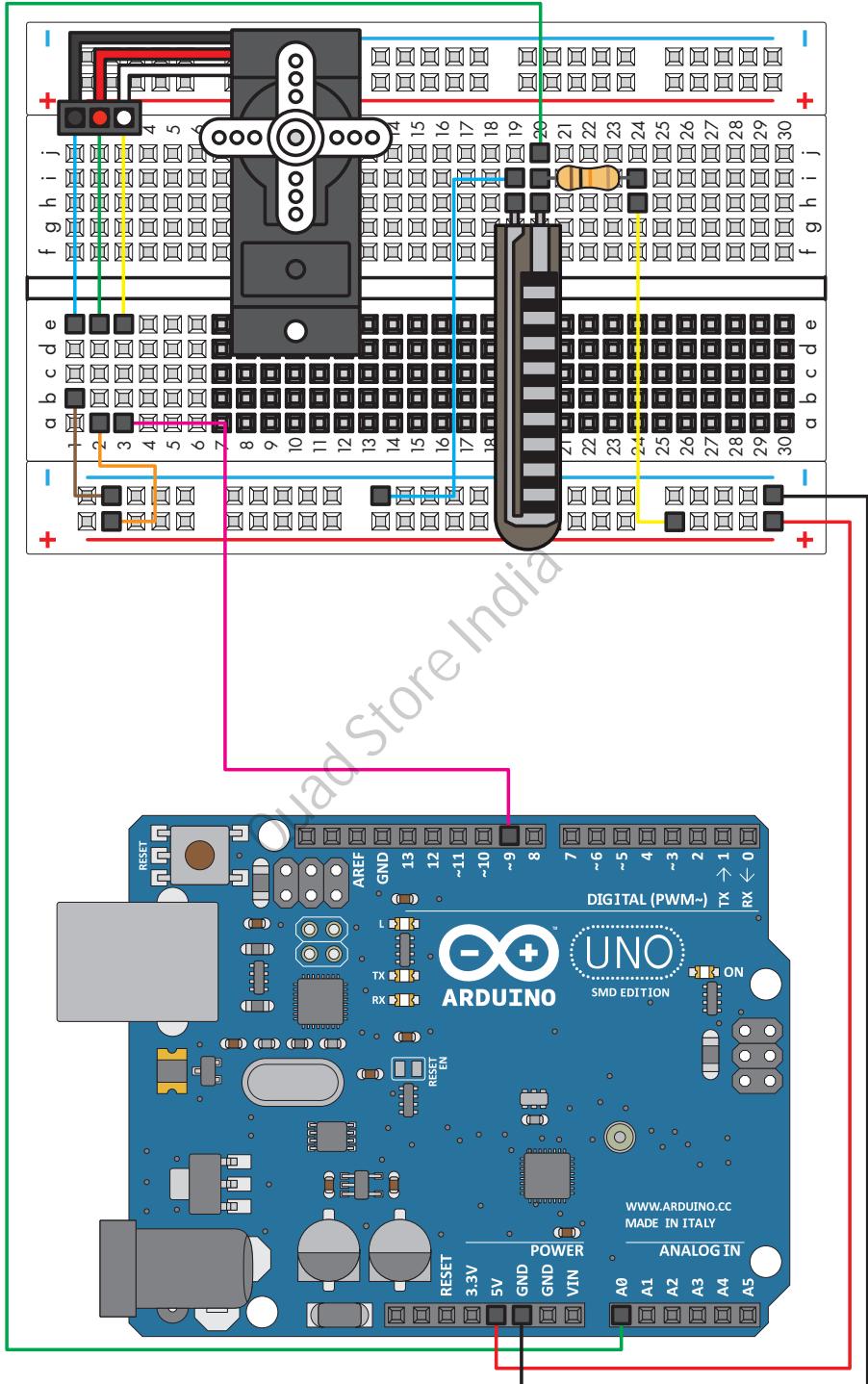


## Flex Sensor

In this circuit, we will use a flex sensor to measure, well, flex! A flex sensor uses carbon on a strip of plastic to act like a variable resistor, but instead of changing the resistance by turning a knob, you change it by flexing (bending) the component. We use a "voltage divider" again to detect this change in resistance. The sensor bends in one direction and the more it bends, the higher the resistance gets; it has a range from about 10K ohm to 35K ohm. In this circuit we will use the amount of bend of the flex sensor to control the position of a servo.

### PARTS:





Circuit 9: Flex Sensor

## Debugging your sketches using the Serial Monitor:

Component	Image Reference:		
Servo		e1   e2   e3	
Jumper Wire		e1   ●   -	
Jumper Wire		e2   ●   -	
Jumper Wire		e3   ○   -	
Flex Sensor		h19   h20	
10KΩ Resistor		20   24   -	
Jumper Wire		19   -	
Jumper Wire		20   -	
Jumper Wire		A0   -	
Jumper Wire		24   +	
Jumper Wire		61   -	
Jumper Wire		62   +	
Jumper Wire		Pin 9   -	
Jumper Wire		5V   -	
Jumper Wire		GND   -	

It happens to everyone - you write a sketch which successfully compiles and uploads, but you can't figure out why it's not doing what you want it to. Larger computers have screens, keyboards, and mice that you can use to debug your code, but tiny computers like the Arduino have no such things.

The key to visibility into a microcontroller is output. This can be almost anything, including LEDs and buzzers, but one of the most useful tools is the serial monitor. Using `Serial.print()` and `println()`, you can easily output human-readable text and data from the Arduino to a window back on the host computer. This is great for your sketch's final output, but it's also incredibly useful for debugging.

```
for (x = 0; x < 8; x++)  
{  
    Serial.print(x);  
}
```

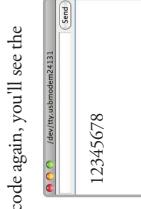
Let's say you wanted a `for()` loop from 1 to 8, but your code just doesn't seem to be working right. Just add `Serial.begin(9600)` to your `setup()` function, and add a `Serial.println()` or `println()` to your loop.



You wanted 1 to 8, but the loop is actually giving you 0 to 7. Whoops! Now you just need to fix the loop.

```
for (x = 1 ; x < 9 ; x++)  
{  
    Serial.print(x);  
}
```

And if you run the code again, you'll see the output you wanted:





Code to Note:



```
servoposition = map(flexposition, 600, 900, 0, 180);
```

⇒ Because the flex sensor / resistor combination won't give us a full zero to five-volt range, we're using the map() function as a handy way to reduce that range. Here we've told it to only expect values from 600 to 900, rather than 0 to 1023.

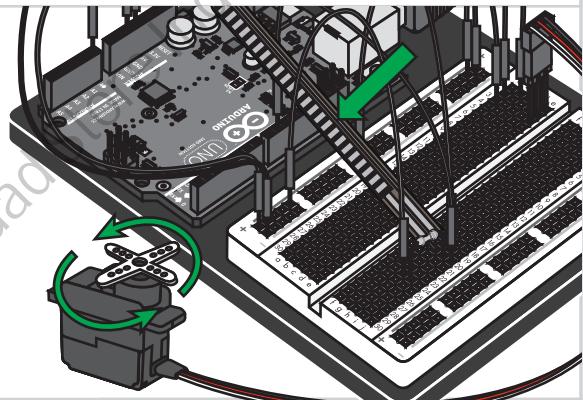
```
Serial.print("sensor: ");  
Serial.print(flexposition);  
Serial.print(" servo: ");  
Serial.println(servoposition);
```



Because map() could still return numbers outside the "to" range, we'll also use a function called constrain() that will "clip" numbers into a range. If the number is outside the range, it will make it the largest or smallest number. If it is within the range, it will stay the same.

## What You Should See:

You should see the servo motor move in accordance with how much you are flexing the flex sensor. If it isn't working, make sure you have assembled the circuit correctly and verified and uploaded the code to your board or see the troubleshooting tips below.



## Troubleshooting:

### Servo Not Twisting

Even with colored wires it is still shockingly easy to plug a servo in backwards. This might be the case.

### Servo Not Moving as Expected

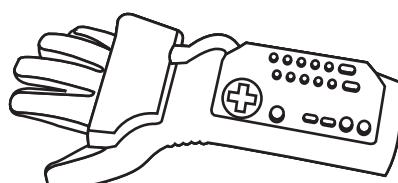
The sensor is only designed to work in one direction. Try flexing it the other way (where the striped side faces out on a convex curve).

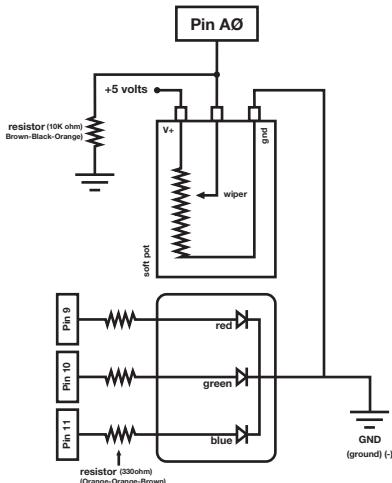
### Servo Doesn't Move very Far

You need to modify the range of values in the call to the map() function.

## Real World Application:

Controller accessories for videogame consoles like Nintendo's "Power Glove" use flex-sensing technology. It was the first video game controller attempting to mimic hand movement on a screen in real time.

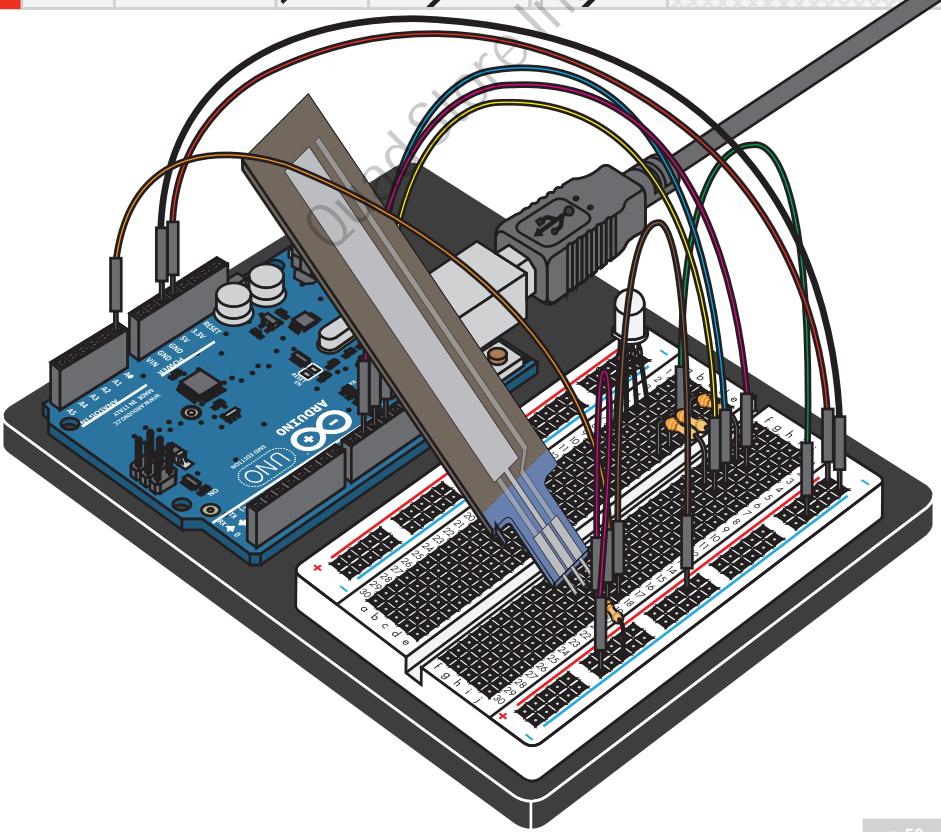




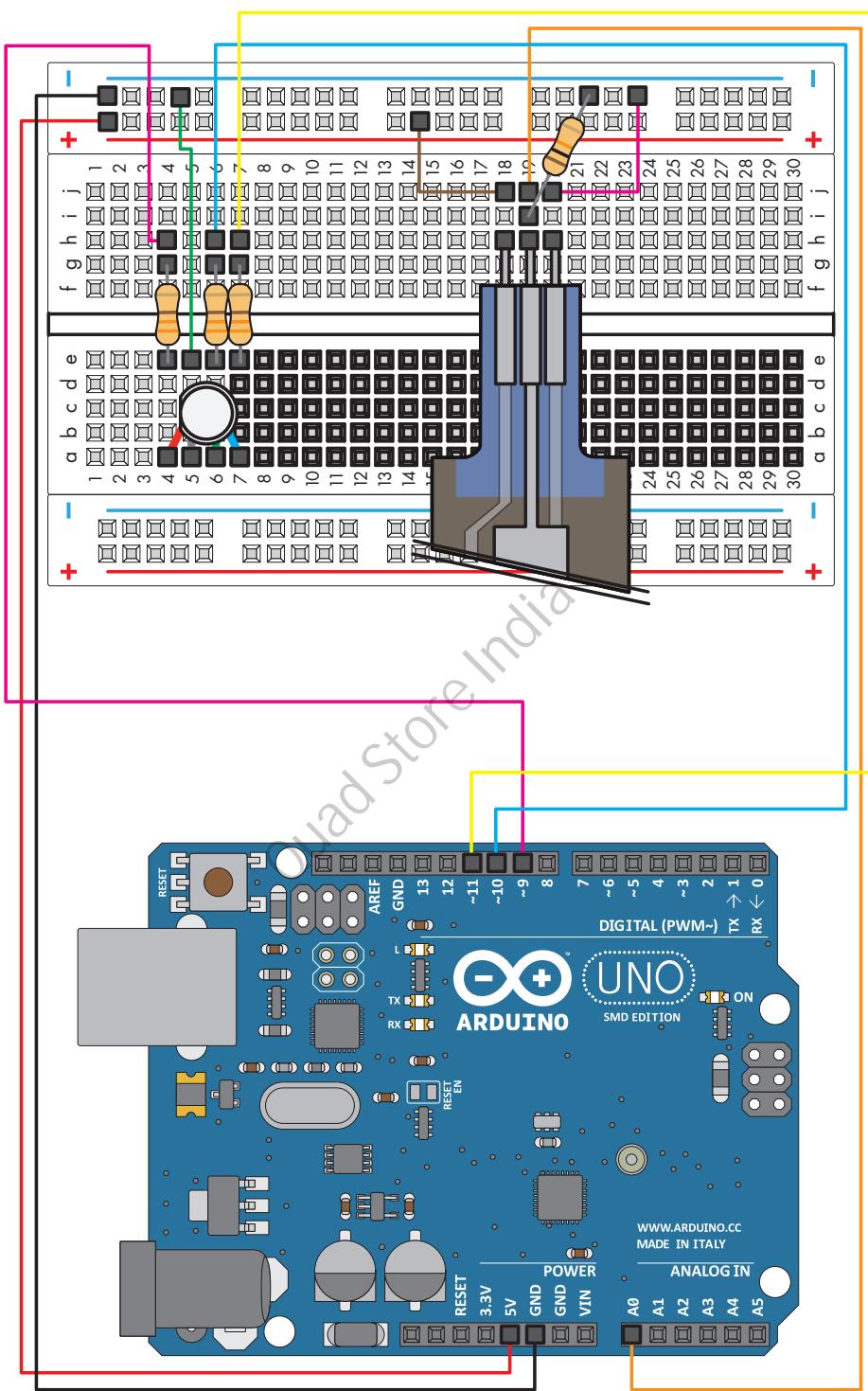
## Soft Potentiometer

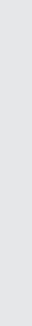
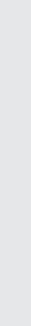
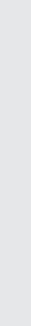
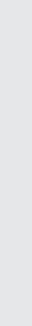
In this circuit, we are going to use yet another kind of variable resistor – this time, a soft potentiometer (or soft pot). This is a thin and flexible strip that can detect where pressure is being applied. By pressing down on various parts of the strip, you can vary the resistance from 100 to 10K ohms. You can use this ability to track movement on the soft pot, or simply as a button. In this circuit, we'll get the soft pot up and running to control an RGB LED.

### PARTS:



## Circuit 10: Soft Potentiometer



Component:	Image Reference:	Component:	Image Reference:
RGB LED (5mm) 		Jumper Wire	
Soft Potentiometer		Jumper Wire	
330Ω Resistor		Jumper Wire	
330Ω Resistor		Jumper Wire	
330Ω Resistor		Jumper Wire	
10KΩ Resistor		Pin 9	
Jumper Wire		h4	
Jumper Wire		e5	
Jumper Wire		h6	
Jumper Wire		h+	
Jumper Wire		i18	
Jumper Wire		A0	
Jumper Wire		i20	



Code to Note:



```
redValue = constrain(map(RGBposition, 0, 341, 255, 0), 0, 255)
+ constrain(map(RGBposition, 682, 1023, 0, 255), 0, 255);
```

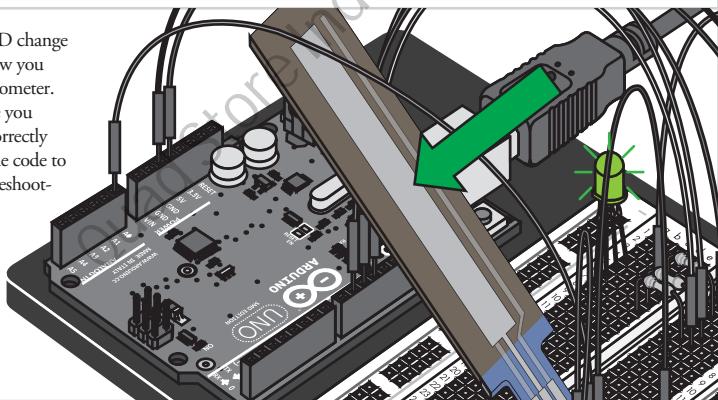
```
greenValue = constrain(map(RGBposition, 0, 341, 0, 255), 0, 255)
- constrain(map(RGBposition, 341, 682, 0, 255), 0, 255);
```

```
blueValue = constrain(map(RGBposition, 341, 682, 0, 255), 0, 255)
- constrain(map(RGBposition, 682, 1023, 0, 255), 0, 255);
```

These big, scary functions take a single Value (RGBposition) and calculate the three RGB values necessary to create a rainbow of color. The functions create three "peaks" for the red, green, and blue values, which overlap to mix and create new colors. Even if you're not 100% clear how it works, you can copy and paste this (or any) function into your own code and use it yourself.

## What You Should See:

You should see the RGB LED change colors in accordance with how you interact with the soft potentiometer. If it isn't working, make sure you have assembled the circuit correctly and verified and uploaded the code to your board, or see the troubleshooting tips below.



## Troubleshooting:

### LED Remains Dark or Shows Incorrect Color

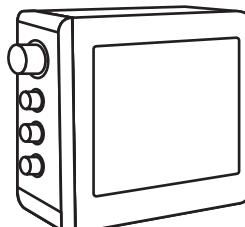
With the four pins of the LED so close together, it's sometimes easy to misplace one. Try double checking each pin is where it should be.

### Bizarre Results

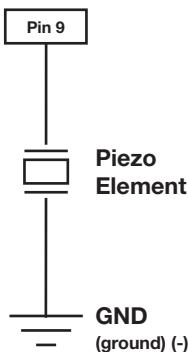
The most likely cause of this is if you're pressing the potentiometer in more than one position. This is normal and can actually be used to create some neat results.

## Real World Application:

The knobs found on many objects, like a radio for instance, are using similar concepts to the one you just completed for this circuit.



## Buzzer

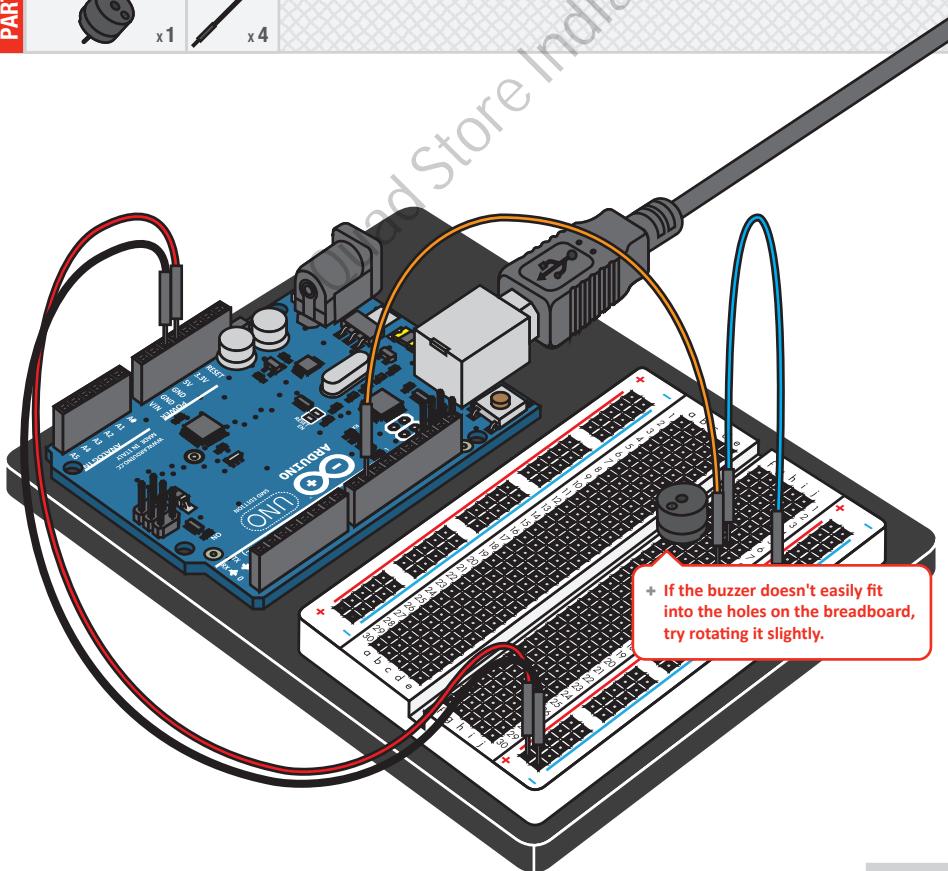


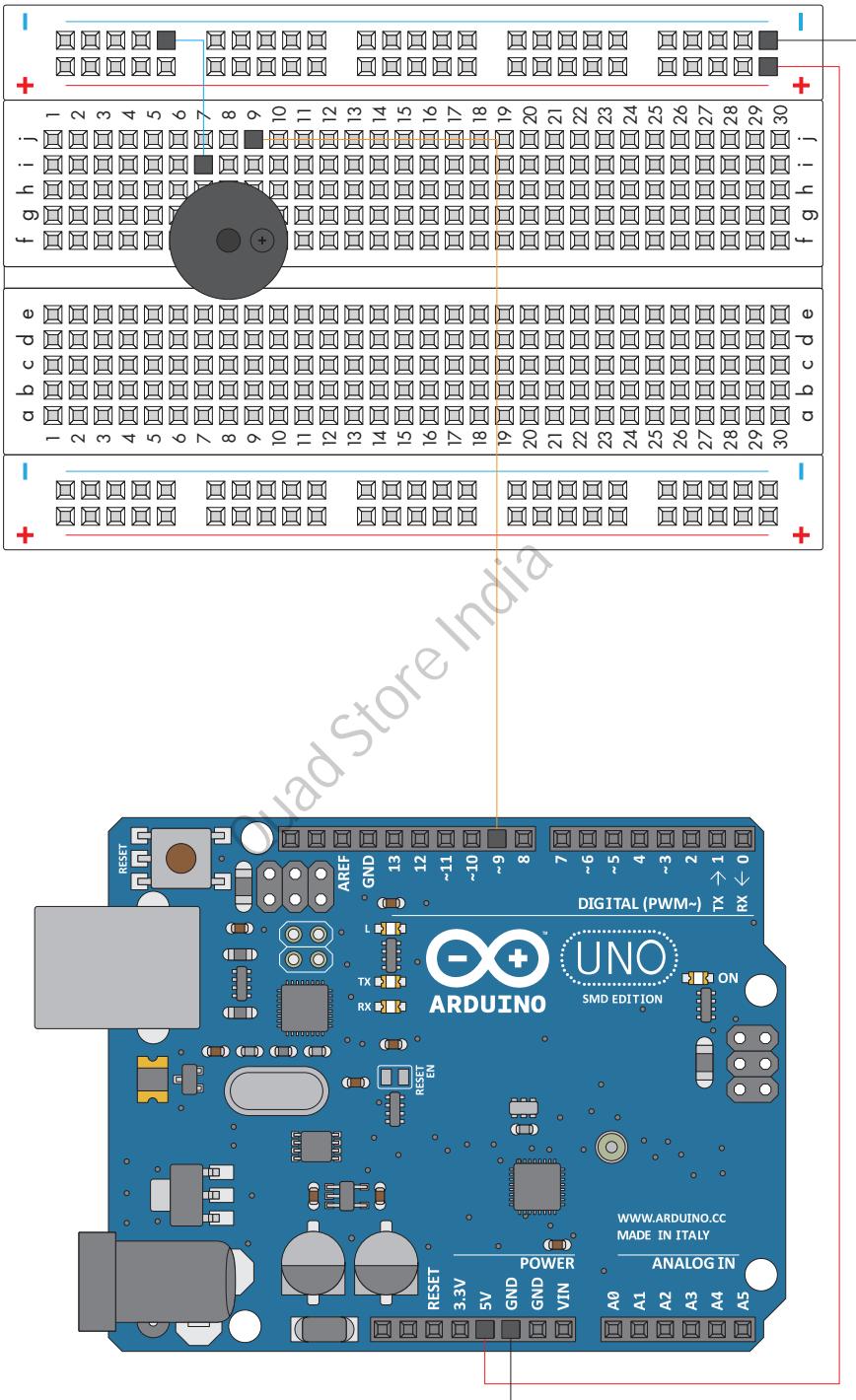
In this circuit, we'll again bridge the gap between the digital world and the analog world. We'll be using a buzzer that makes a small "click" when you apply voltage to it (try it!). By itself that isn't terribly exciting, but if you turn the voltage on and off hundreds of times a second, the buzzer will produce a tone. And if you string a bunch of tones together, you've got music! This circuit and sketch will play a classic tune. We'll never let you down!

**PARTS:**

Piezo Element  
x1

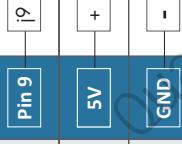
Wire  
x4





Circuit 11: Piezo Elements

## Creating your own functions:



Arduino contains a wealth of built-in functions that are useful for all kinds of things. (See <http://arduino.cc/en/Reference> for a list.) But you can also easily create your own functions. Here's a simple example named "add", which adds two numbers together and returns the result. Let's break it down.

```
int add(int parameter1, int parameter2)
{
    int x;
    x = parameter1 + parameter2;
    return(x);
}
```

Your functions can take in values ("parameters"), and return a value, as this one does. But you can also do either or none of those things, if you wish.

If you'll be passing parameters **to** your function, put them (and their types) in the parentheses after the function name. If you won't be giving your function any parameters, just use an empty parenthesis () after the name.

If you'll be returning a value **from** your function, put the type of the return value in front of the function name. Then in your function, when you're ready to return the value, put in a **return()** statement. If you won't be returning a value, put "void" in front of the function name (just like you've already seen for the **setup()** and **loop()** functions).

When you write your own functions, you make your code neater and easier to re-use.

Component:	Image Reference:				
Piezo Element		19	17	+	-
Jumper Wire		17	19	-	
Jumper Wire		5V	Pin 9	+	
Jumper Wire		GND		-	
Jumper Wire					



**Open Arduino IDE // File > Examples > SIK Guide > Circuit # 11**

Code to Note:



```
char notes[] = "cdfda ag cdfdg gf ";
```



```
char names[] = {'c','d','e','f','g','a','b','C'};
```

Up until now we've been working solely with numerical data, but the Arduino can also work with text. Characters (single, printable, letters, numbers and other symbols) have their own type, called "char". When you have an array of characters, it can be defined between double-quotes (also called a "string"), OR as a list of single-quoted characters.

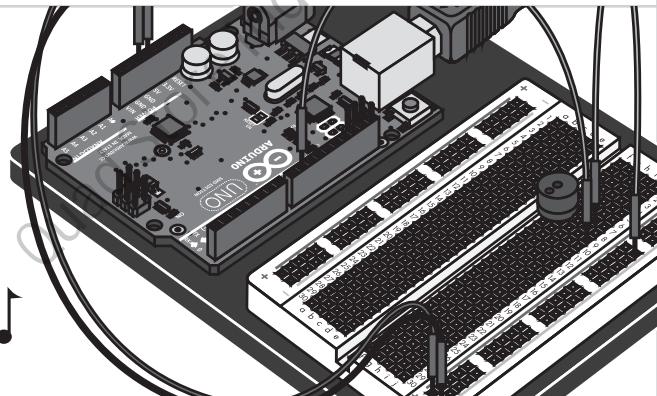
```
tone(pin, frequency, duration);
```



One of Arduino's many useful built-in commands is the tone() function. This function drives an output pin at a certain frequency, making it perfect for driving buzzers and speakers. If you give it a duration (in milliseconds), it will play the tone then stop. If you don't give it a duration, it will keep playing the tone forever (but you can stop it with another function, noTone() ).

## What You Should See:

You should see - well, nothing! But you should be able to hear your piezo element playing "Twinkle, Twinkle Little Star" (or possibly, "The ABCs"). If it isn't working, make sure you have assembled the circuit correctly and verified and uploaded the code to your board or see the troubleshooting tips below.



## Troubleshooting:

### No Sound

Given the size and shape of the piezo element it is easy to miss the right holes on the breadboard. Try double checking its placement.

### Can't Think While the Melody is Playing

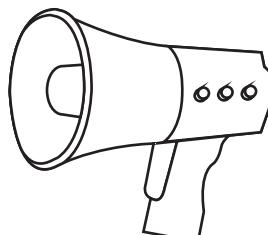
Just pull up the piezo element whilst you think, upload your program then plug it back in.

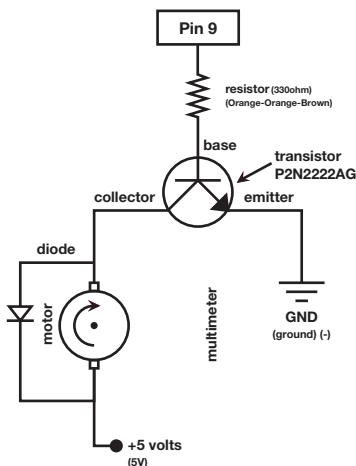
### Tired of Twinkle Twinkle Little Star

The code is written so you can easily add your own songs.

## Real World Application:

Many modern megaphones have settings that use a loud amplified buzzer. They are usually very loud and quite good at getting people's attention.





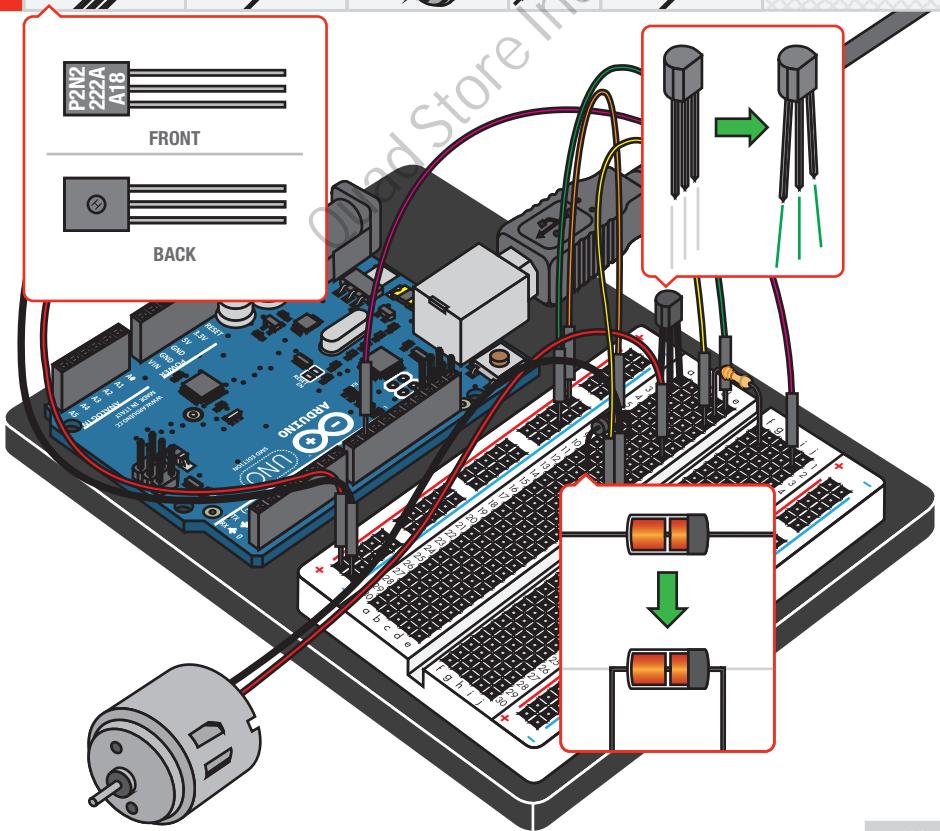
## Spinning a Motor

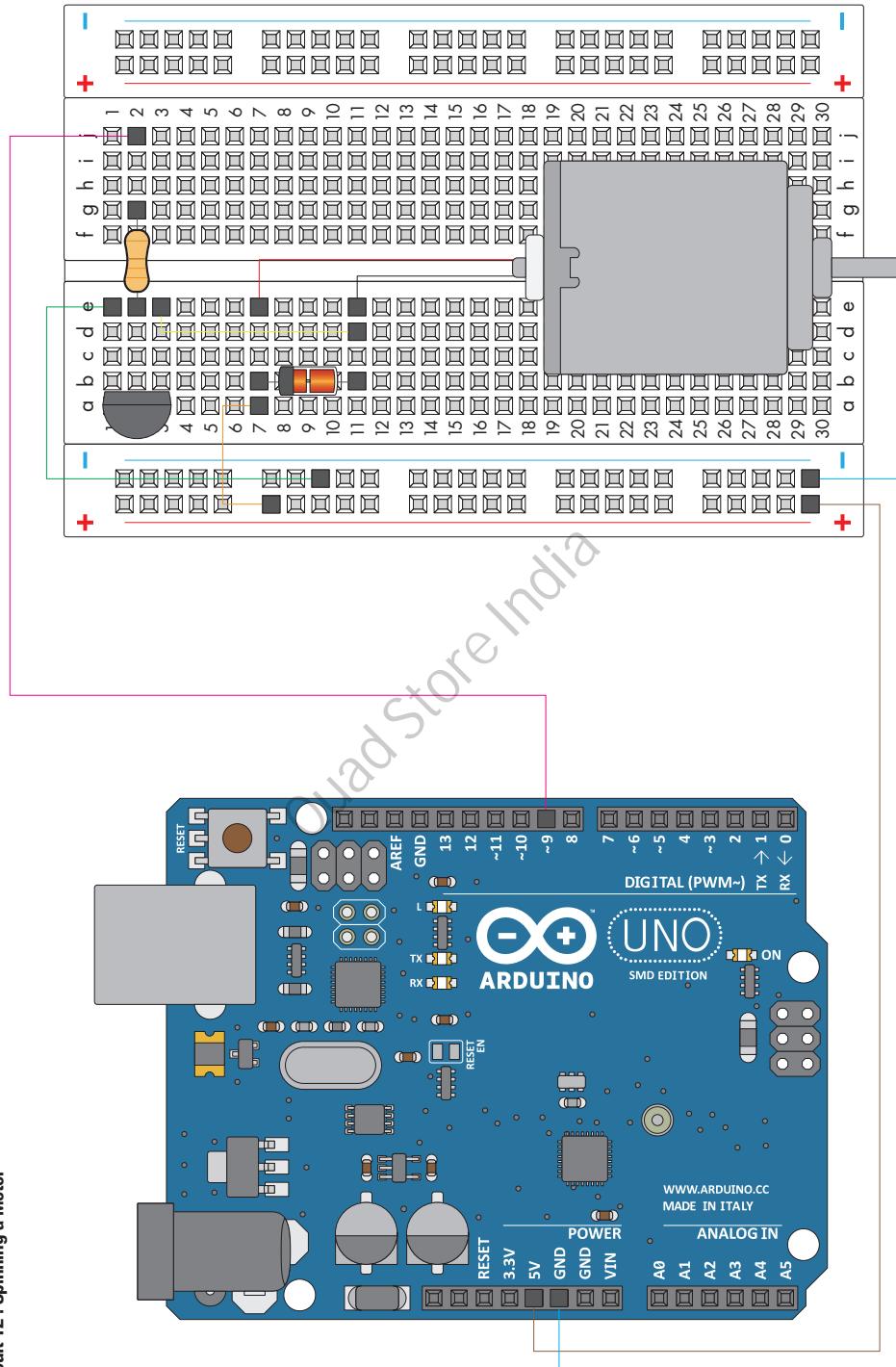
Remember before when you played around with a servo motor? Now we are going to tackle spinning a motor. This requires the use of a transistor, which can switch a larger amount of current than the Arduino can. When using a transistor, you just need to make sure its maximum specs are high enough for your use. The transistor we are using for this circuit is rated at 40V max and 200 millamps max – perfect for our toy motor!



When you're building the circuit be careful not to mix up the transistor and the temperature sensor, they're almost identical.

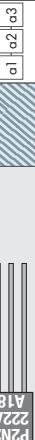
## PARTS:





Circuit 12 : Spinning a Motor

## Putting it all together:



Transistor P2N2222A



Diode 1N4148



DC Motor



330Ω Resistor



Jumper Wire



Jumper Wire



Jumper Wire



Jumper Wire



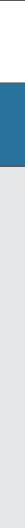
Jumper Wire



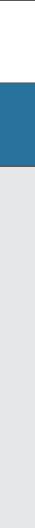
Jumper Wire



Jumper Wire



Pin 9



Pin 10

At this point you're probably starting to get your own ideas for circuits that do fun things, or help solve a real problem. Excellent! Here are some tips on programming in general.

Most of the sketches you write will be a loop with some or all of these steps:

1. Perform some sort of input
2. Make some calculations or decisions
3. Perform some sort of output
4. Repeat! (Or not!)

We've already shown you how to use a bunch of different input sensors and output devices (and we still have a few more to go). Feel free to make use of the examples in your own sketches - this is the whole idea behind the "Open Source" movement.

It's usually pretty easy to pull pieces of different sketches together, just open them in two windows, and copy and paste between them. This is one of the reasons we've been promoting "good programming habits". Things like using constants for pin numbers, and breaking your sketch into functions, make it much easier to re-use your code in new sketches. For example, if you pull in two pieces of code that use the same pin, you can easily change one of the constants to a new pin. (Don't forget that not all of the pins support `analogWrite()`; the compatible pins are marked on your board.)

If you need help, there are internet forums where you can ask questions. Try Arduino's forum at [arduino.cc/forum](http://arduino.cc/forum), and SparkFun's at [forum.sparkfun.com](http://forum.sparkfun.com). When you're ready to move to more advanced topics, take a look at Arduino's tutorials page at [arduino.cc/en/Tutorial](http://arduino.cc/en/Tutorial). Many of SparkFun's more advanced products were programmed with Arduino, (allowing you to easily modify them), or have Arduino examples for them. See our product pages for info.

Finally, when you create something really cool, consider sharing it with the world so that others can learn from your genius. (And be sure to let us know so we can put it on our home page<sup>e</sup>)



Code to Note:

**while (Serial.available() > 0)**

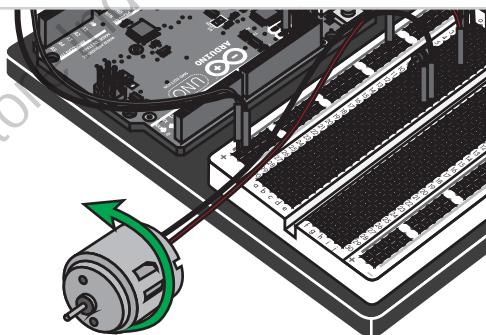
The Arduino's serial port can be used to receive as well as send data. Because data could arrive at any time, the Arduino stores, or "buffers" data coming into the port until you're ready to use it. The Serial.available() command returns the number of characters that the port has received, but haven't been used by your sketch yet. Zero means no data has arrived.

**speed = Serial.parseInt();**

If the port has data waiting for you, there are a number of ways for you to use it. Since we're typing numbers into the port, we can use the handy Serial.parseInt() command to extract, or "parse" integer numbers from the characters it's received. If you type "1" "0" "0" to the port, this function will return the number 100.

## What You Should See:

The DC Motor should spin if you have assembled the circuit's components correctly, and also verified/uploaded the correct code. If your circuit is not working check the troubleshooting section below.



## Troubleshooting:

### **Motor Not Spinning**

If you sourced your own transistor, double check with the data sheet that the pinout is compatible with a P2N2222AG (many are reversed).

### **Still No Luck**

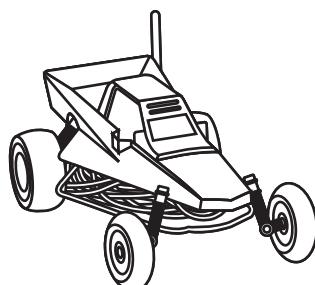
If you sourced your own motor, double check that it will work with 5 volts and that it does not draw too much power.

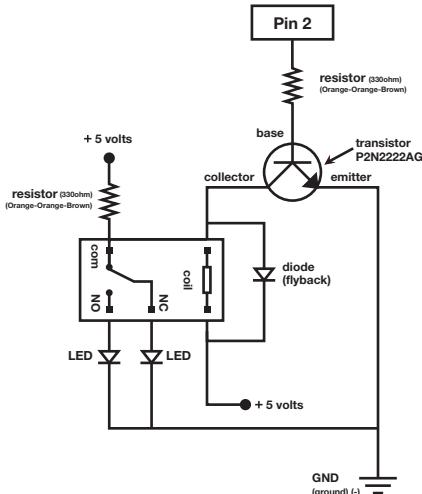
### **Still Not Working**

Sometimes the Arduino board will disconnect from the computer. Try un-plugging and then re-plugging it into your USB port.

## Real World Application:

Radio Controlled(RC) cars use Direct Current(DC) motors to turn the wheels for propulsion.





## Relays

In this circuit, we are going to use some of the lessons we learned in circuit 12 to control a relay. A relay is basically an electrically controlled mechanical switch. Inside that harmless looking plastic box is an electromagnet that, when it gets a jolt of energy, causes a switch to trip. In this circuit, you'll learn how to control a relay like a pro – giving your Arduino even more powerful abilities!



When the relay is off, the COM (common) pin will be connected to the NC (Normally Closed) pin. When the relay is on, the COM (common) pin will be connected to the NO (Normally Open) pin.

### PARTS:

Relay



x1

Transistor  
P2N2222AG

x1

Diode  
1N4148

x1

330Ω  
Resistor

x2

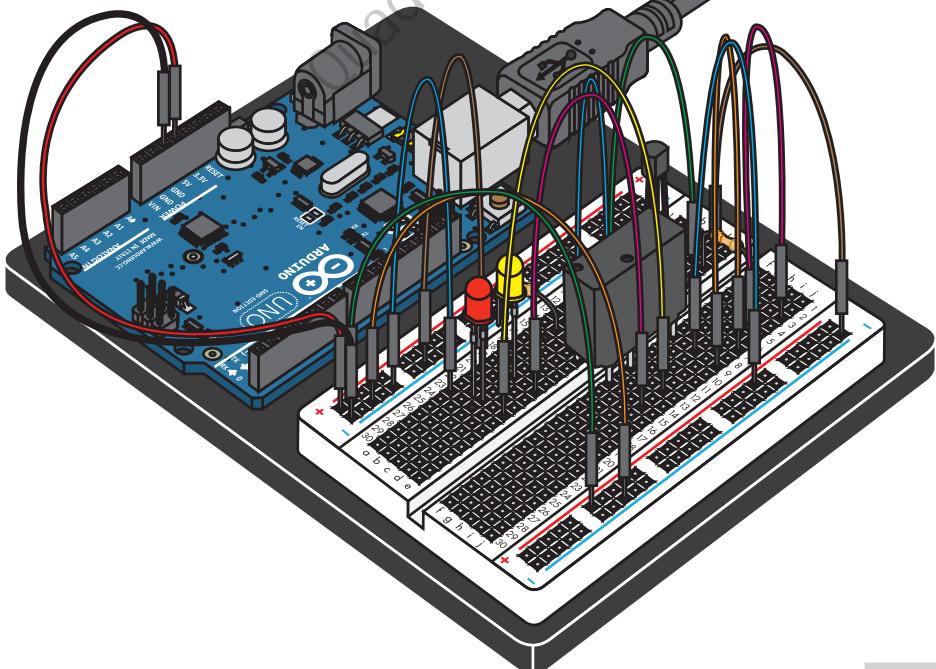


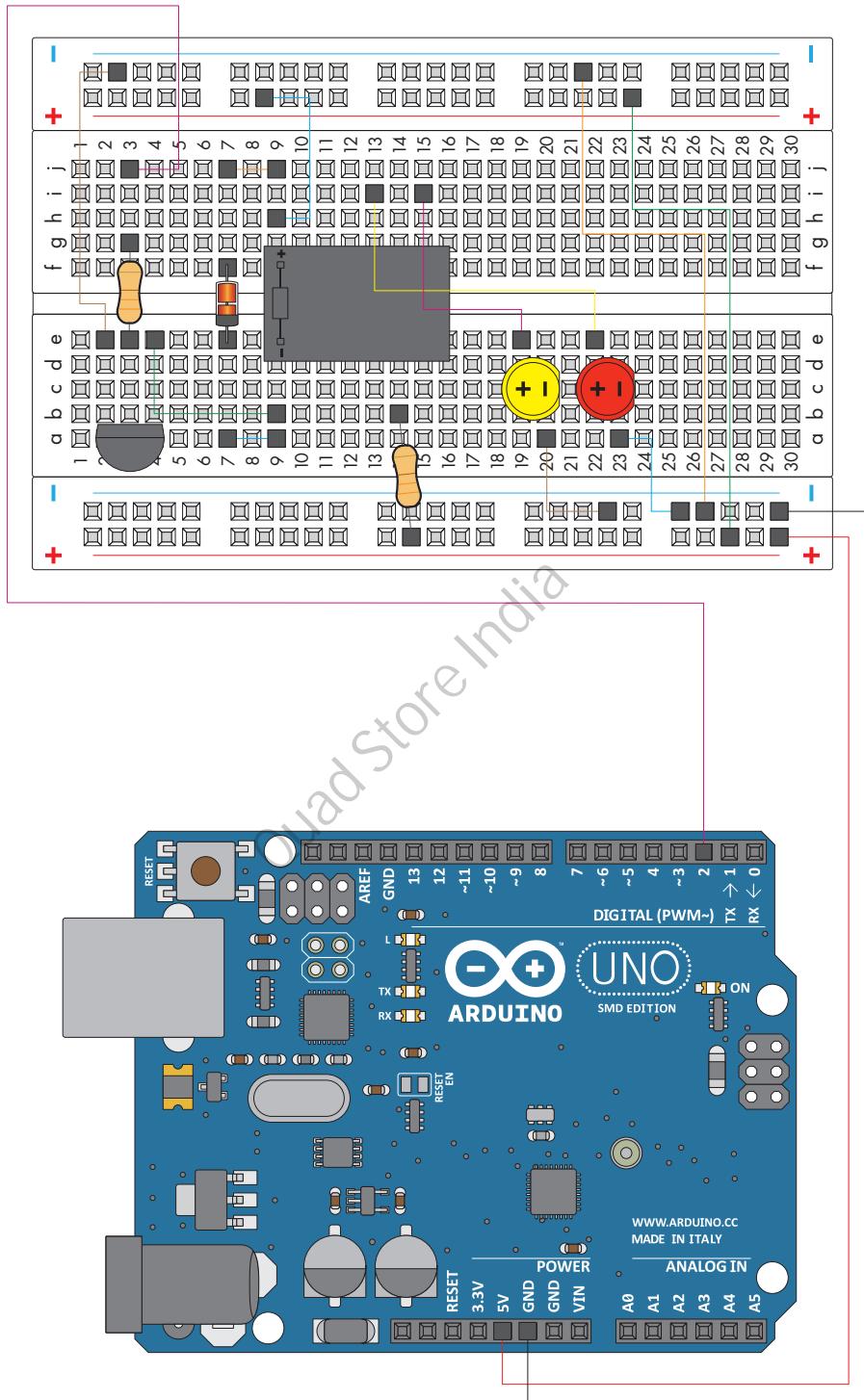
x2

Wire



x14





Component:	Image Reference:	Component:	Image Reference:
Relay		Jumper Wire	
Transistor P2N2222A		Jumper Wire	
LED (5mm)		Jumper Wire	
LED (5mm)		Jumper Wire	
Diode 1N4148		Jumper Wire	
330Ω Resistor		Jumper Wire	
330Ω Resistor		Jumper Wire	
Jumper Wire		Jumper Wire	
Jumper Wire		Jumper Wire	
Jumper Wire		Jumper Wire	
Jumper Wire		Jumper Wire	
Jumper Wire		Jumper Wire	
		Pin 2	
		i3	
		i7	
		i9	
		h9	
		i13	
		e22	
		i15	
		e19	



Code to Note:

**`digitalWrite(relayPin, HIGH);`**

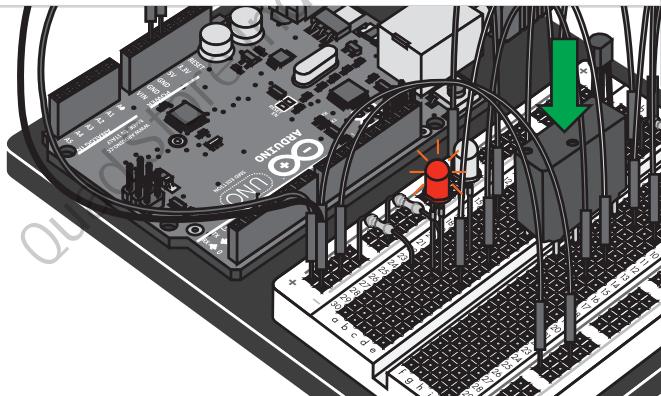
When we turn on the transistor, which in turn energizes the relay's coil, the relay's switch contacts are closed. This connects the relay's COM pin to the NO (Normally Open) pin. Whatever you've connected using these pins will turn on. (Here we're using LEDs, but this could be almost anything.)

**`digitalWrite(relayPin, LOW);`**

The relay has an additional contact called NC (Normally Closed). The NC pin is connected to the COM pin when the relay is OFF. You can use either pin depending on whether something should be normally on or normally off. You can also use both pins to alternate power to two devices, much like railroad crossing warning lights.

## What You Should See:

You should be able to hear the relay contacts click, and see the two LEDs alternate illuminating at 1-second intervals. If you don't, double-check that you have assembled the circuit correctly, and uploaded the correct sketch to the board. Also, see the troubleshooting tips below.



## Troubleshooting:

### LEDs Not Lighting

Double-check that you've plugged them in correctly. The longer lead (and non-flat edge of the plastic flange) is the positive lead.

### No Clicking Sound

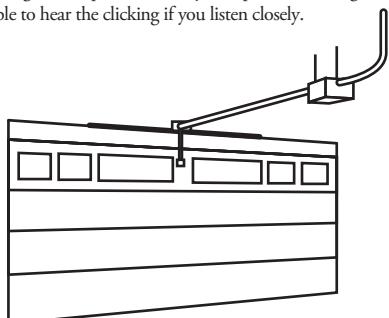
The transistor or coil portion of the circuit isn't quite working. Check the transistor is plugged in the right way.

### Not Quite Working

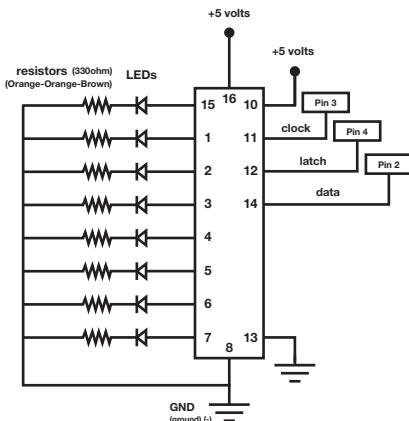
The included relays are designed to be soldered rather than used in a breadboard. As such you may need to press it in to ensure it works (and it may pop out occasionally).

## Real World Application:

Garage door openers use relays to operate. You might be able to hear the clicking if you listen closely.

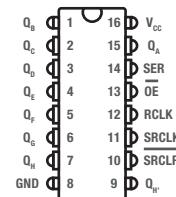
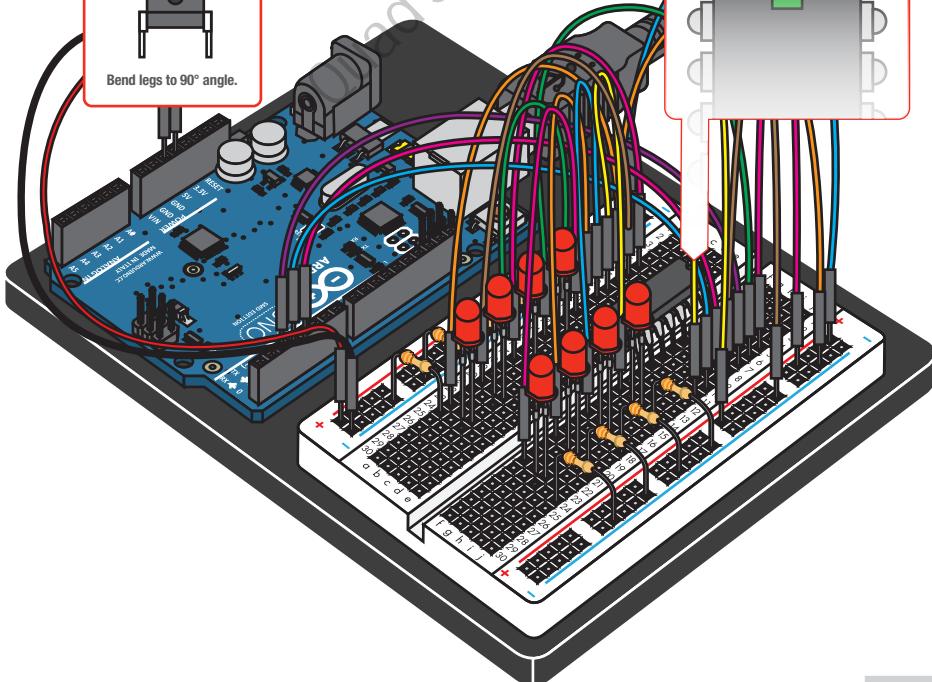


## Shift Register

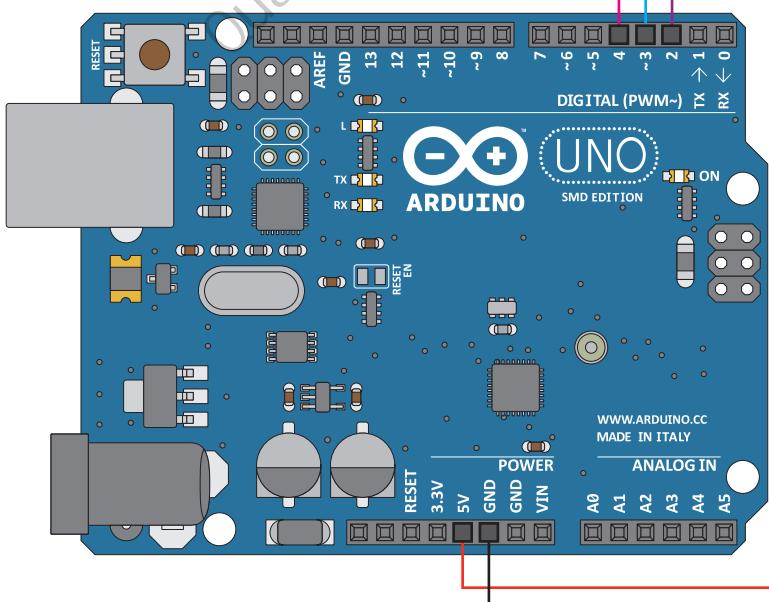
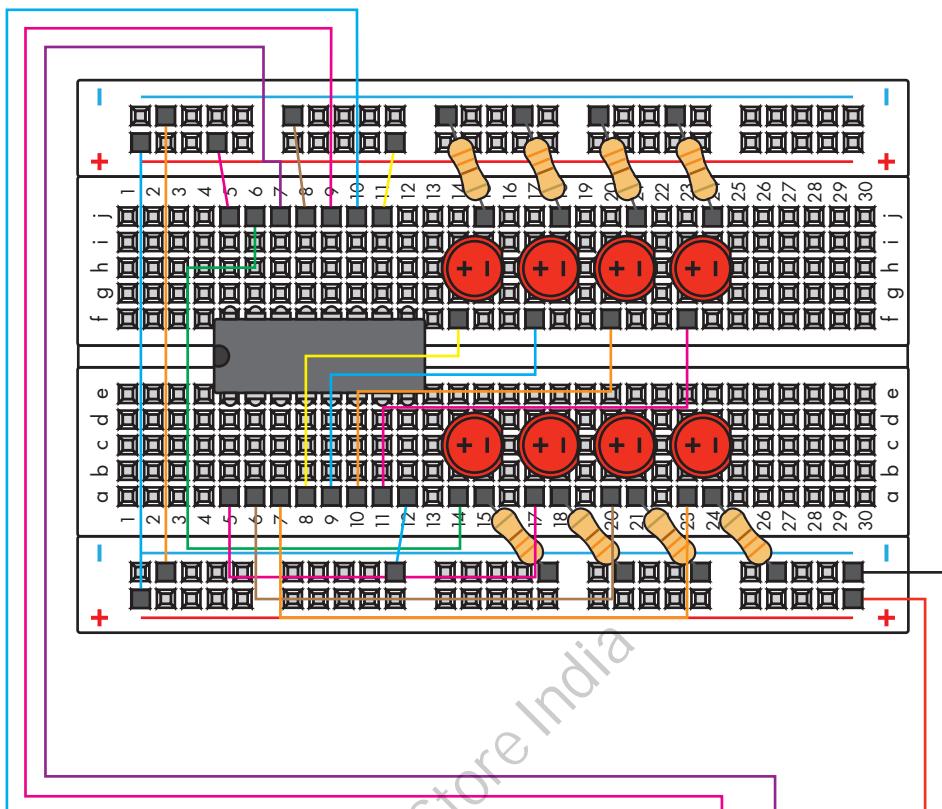


Now we are going to step into the world of ICs (integrated circuits). In this circuit, you'll learn all about using a shift register (also called a serial-to-parallel controller). The shift register will give your Arduino an additional eight outputs, using only three pins on your board. For this circuit, you'll practice by using the shift register to control eight LEDs.

## PARTS:



Align notch on top, inbetween "e5" and "f5" on the breadboard.



Circuit 14: Shift Register

Component:	Image Reference:	Component:	Image Reference:	Component:	Image Reference:
IC		Jumper Wire		Jumper Wire	
LED (5mm)		Jumper Wire		Jumper Wire	
LED (5mm)		Jumper Wire		Jumper Wire	
LED (5mm)		Jumper Wire		Jumper Wire	
LED (5mm)		Jumper Wire		Jumper Wire	
LED (5mm)		Jumper Wire		Jumper Wire	
LED (5mm)		Jumper Wire		Jumper Wire	
LED (5mm)		Jumper Wire		Jumper Wire	
330Ω Resistor		Jumper Wire		Jumper Wire	
330Ω Resistor		Jumper Wire		Jumper Wire	
330Ω Resistor		Jumper Wire		Jumper Wire	
330Ω Resistor		Jumper Wire		Jumper Wire	
330Ω Resistor		Jumper Wire		Jumper Wire	
330Ω Resistor		Jumper Wire		Jumper Wire	
5V		Jumper Wire		Jumper Wire	
GND		Jumper Wire		Jumper Wire	



Code to Note:

**shiftOut(datapin, clockpin, MSBFIRST, data);**

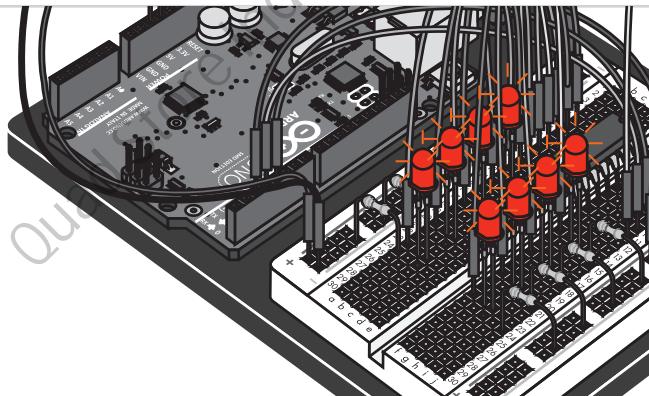
You'll communicate with the shift register (and a lot of other parts) using an interface called SPI, or Serial Peripheral Interface. This interface uses a data line and a separate clock line that work together to move data in or out of the Arduino at high speed. The MSBFIRST parameter specifies the order in which to send the individual bits, in this case we're sending the Most Significant Bit first.

**bitWrite(data,desiredPin,desiredState);**

Bits are the smallest possible piece of memory in a computer; each one can store either a "1" or a "0". Larger numbers are stored as arrays of bits. Sometimes we want to manipulate these bits directly, for example now when we're sending eight bits to the shift register and we want to make them 1 or 0 to turn the LEDs on or off. The Arduino has several commands, such as `bitWrite()`, that make this easy to do.

## What You Should See:

You should see the LEDs light up similarly to in circuit 4 (but this time, you're using a shift register). If they aren't, make sure you have assembled the circuit correctly and verified and uploaded the code to your board, or see the troubleshooting tips below.



## Troubleshooting:

### The Arduino's power LED goes out

This happened to us a couple of times, it happens when the chip is inserted backward. If you fix it quickly nothing will break.

### Not Quite Working

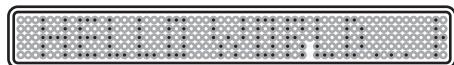
Sorry to sound like a broken record but it is probably something as simple as a crossed wire.

### Frustration

Shoot us an e-mail, this circuit is both simple and complex at the same time. We want to hear about problems you have so we can address them in future editions: [techsupport@sparkfun.com](mailto:techsupport@sparkfun.com)

## Real World Application:

Similar to circuit #4, a scrolling marquee display delivers a message with multiple LEDs. Essentially the same task the shift register achieves here in Circuit #14.





### Explore More:

This is just the beginning of your exploration into embedded electronics and coding. So start to explore more using additional components provided in the kit and online tutorials to plan ahead for your next project!

### NOTES:

Quad Store India