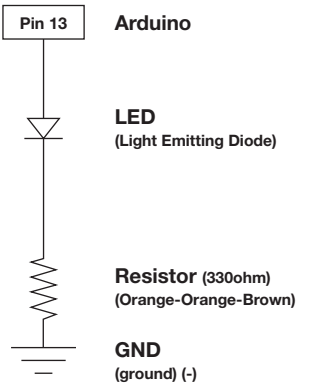
1

Blinking a LED



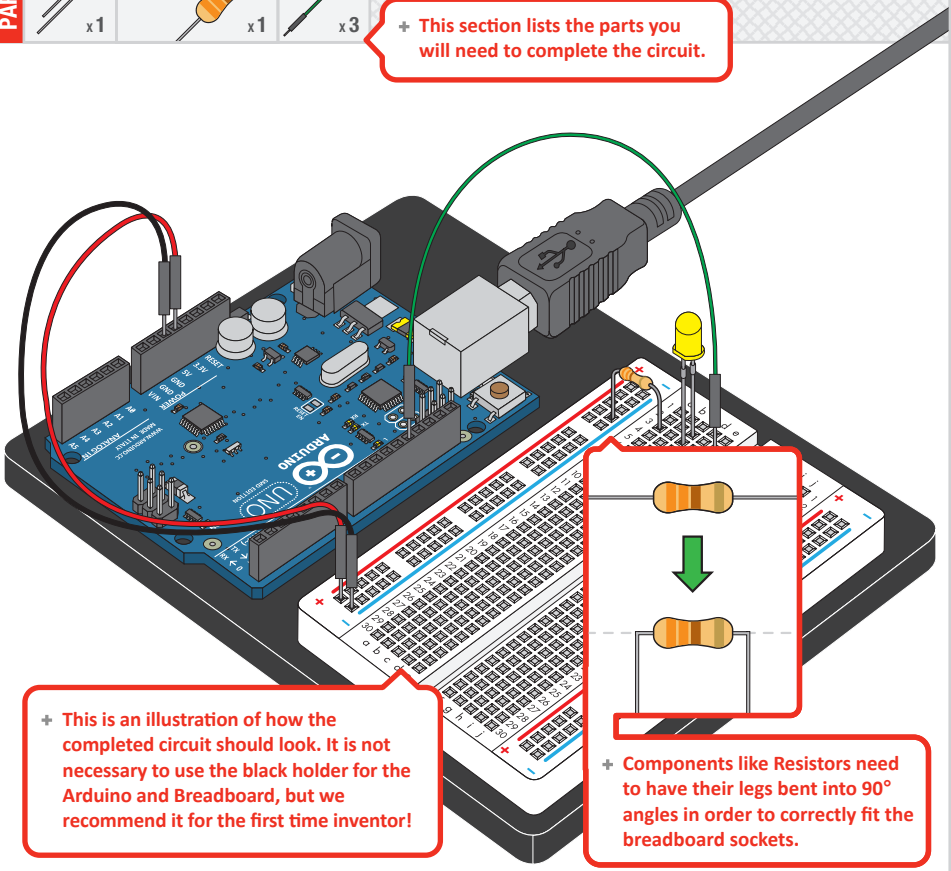
+ This is a schematic of your circuit.

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.

PARTS:

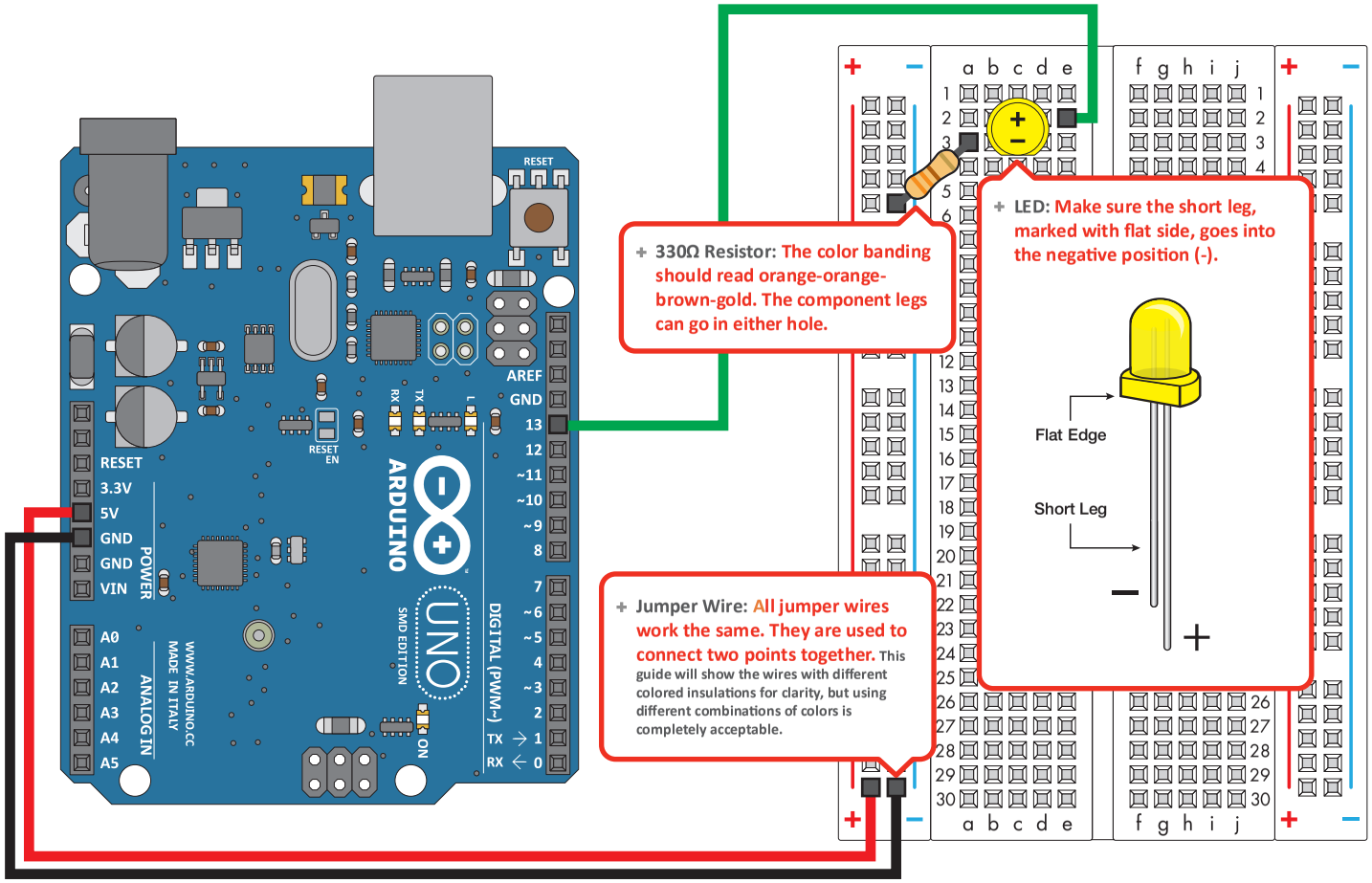
+ This section lists the parts you will need to complete the circuit.

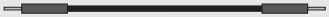
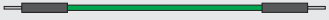


+ This is an illustration of how the completed circuit should look. It is not necessary to use the black holder for the Arduino and Breadboard, but we recommend it for the first time inventor!

+ Components like Resistors need to have their legs bent into 90° angles in order to correctly fit the breadboard sockets.

Circuit 1: Blinking a LED



Component:	Image Reference:		
LED (5mm) ⚠			c2 (+) c3 (-)
330Ω Resistor			a3 -
Jumper Wire		GND	-
Jumper Wire		5V	+
Jumper Wire		Pin 13	e2

+ Components like LEDs are inserted into the Breadboard sockets c2(long leg) c3(short leg).

+ Resistors are placed in Breadboard sockets only. The "-" symbol represents any socket in its vertical column.

+ "GND" on the Arduino should be connected to the row marked "-" on the Breadboard.

+ "5V" on the Arduino connects to the row marked "+" on the Breadboard.

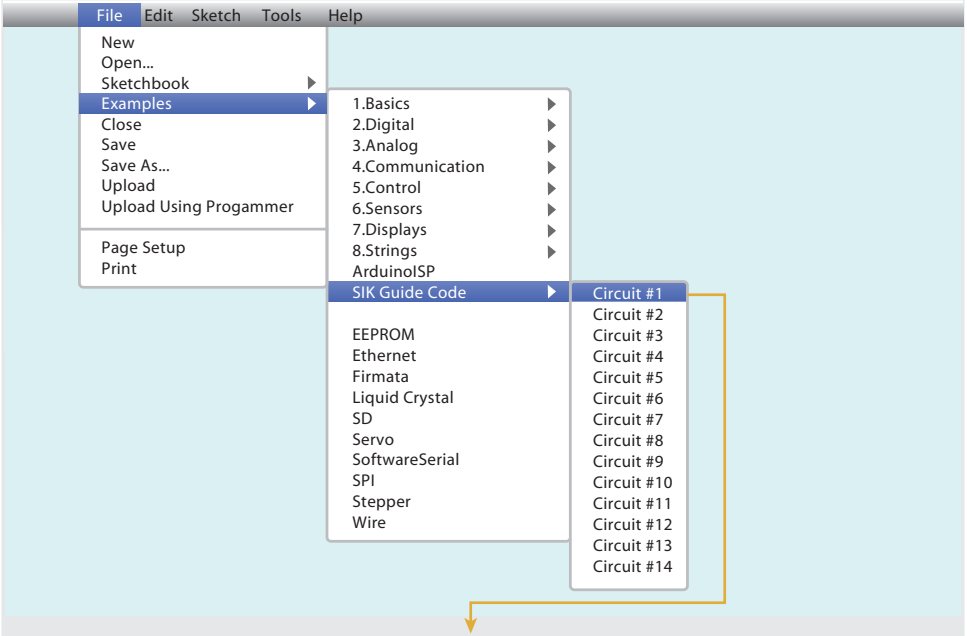
+ "Pin 13" on the Arduino connects to socket "e2" on the Breadboard.

+ **Arduino:** The blue background represents a connection to one of the Arduino header pins.

+ **Breadboard:** The white background represents a connection to a breadboard socket.

Open Your First Sketch:

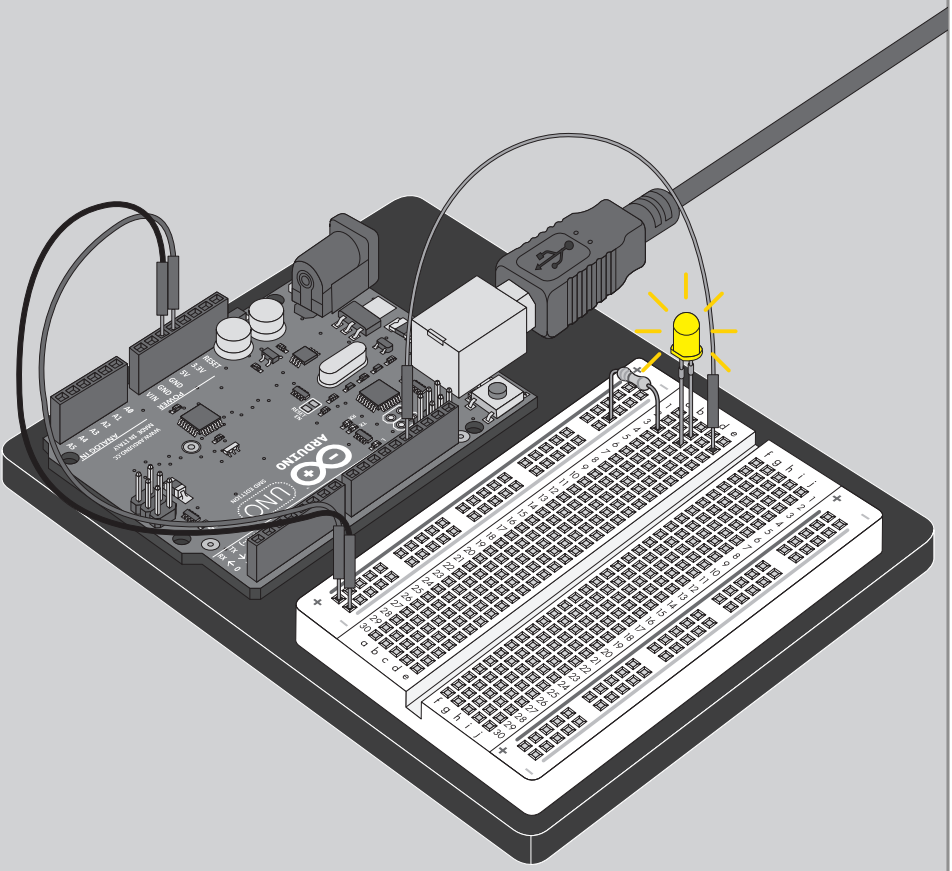
Open Up the Arduino IDE software on your computer. Coding in the Arduino language will control your circuit. Open the code for Circuit 1 by accessing the "SIK Guide Code" you downloaded and placed into your "Example" folder earlier.



// Circuit #1

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

+ This is where you will find the Arduino code for each circuit.

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

Code to Note:

+ Remember to Verify and Upload your code.

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

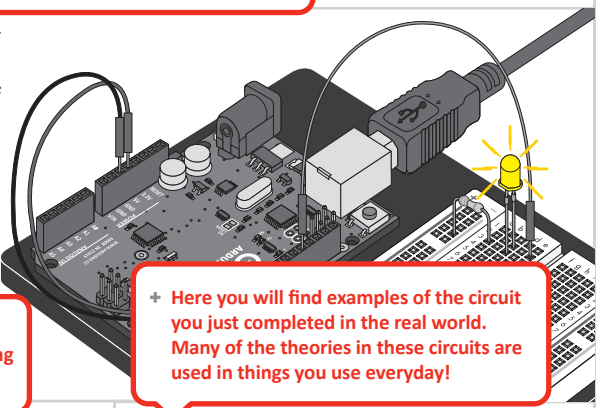
pinMode(13, OUTPUT); ⇨ Before you can use one of the Arduino's pins, you need to tell the Arduino whether it is an INPUT or OUTPUT. We use a built-in "function" called pinMode() to do this.

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.



+ 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!

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

Troubleshooting:

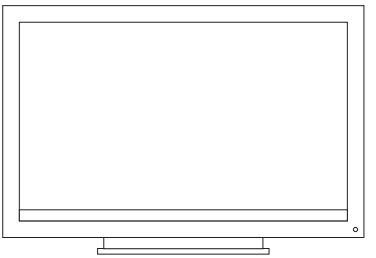
LED Not Lighting Up?
LEDs will only work in one direction. Try taking it out and twisting it 180 degrees (no need to worry, installing it backwards does no permanent harm).

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**

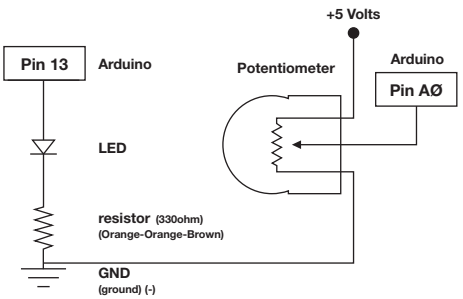
Real World Application:

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



2

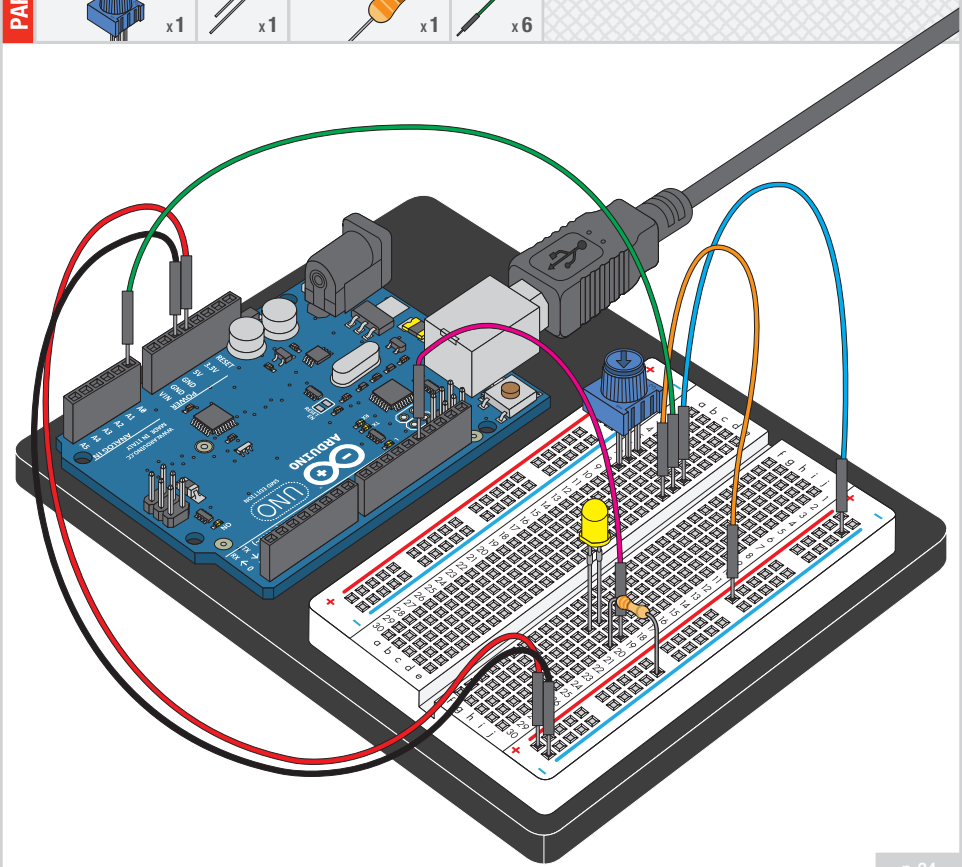
Potentiometer



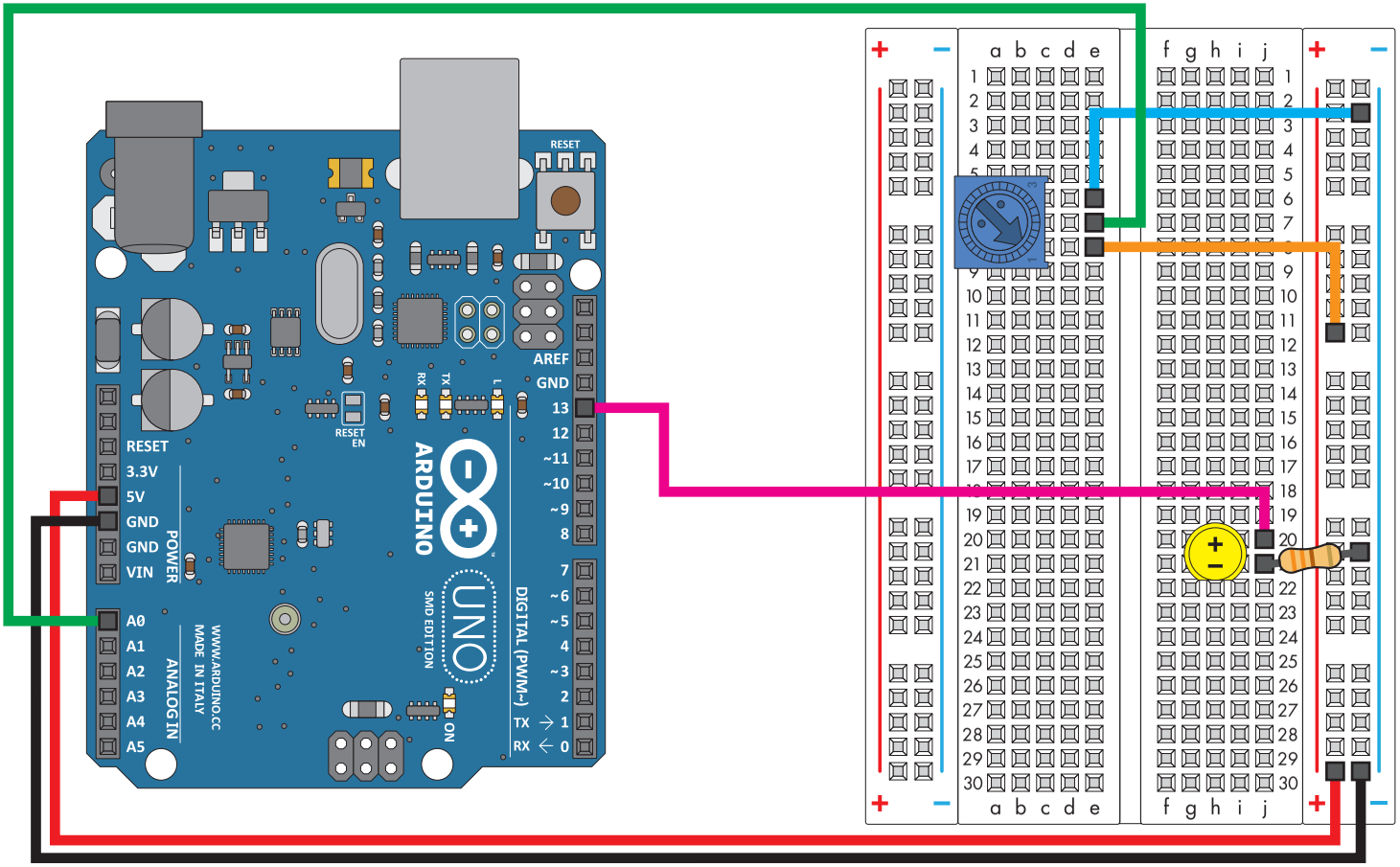
In this circuit you'll work with a potentiometer. A potentiometer is also known as a variable resistor. When it's connected with 5 volts across its two outer pins, the middle pin outputs a voltage between 0 and 5, depending on the position of the knob on the potentiometer. In this circuit, you'll learn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

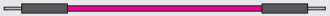
PARTS:

Potentiometer	LED	330Ω Resistor	Wire
x1	x1	x1	x6



Circuit 2: Potentiometer



Component:	Image Reference:		
Potentiometer			a6 a7 a8
LED (5mm) ⚠			h20 (+) h21 (-)
330Ω Resistor			i21 -
Jumper Wire			e6 -
Jumper Wire		A0	e7
Jumper Wire			e8 +
Jumper Wire		Pin 13	i20
Jumper Wire		5V	+
Jumper Wire		GND	-

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

LOW	to	HIGH
off		on
0 volts		5 volts

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

0 volts	to	5 volts
0		1023

2

Arduino Code:

Code to Note:

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

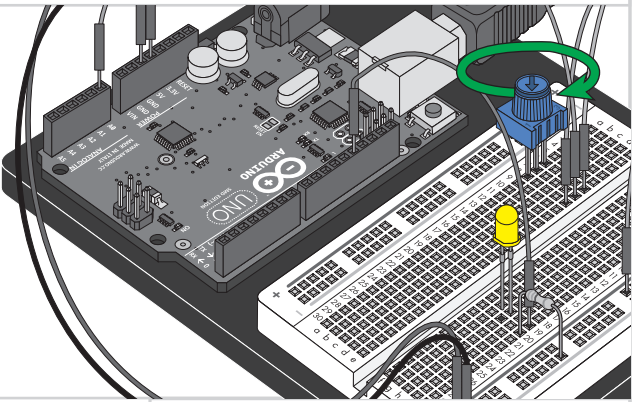
int sensorValue; ⇨ A "variable" is a number you've given a name to. You must introduce, or "declare" variables before you use them; here we're declaring a variable called sensorValue, of type "int" (integer). Don't forget that variable names are case-sensitive!

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:

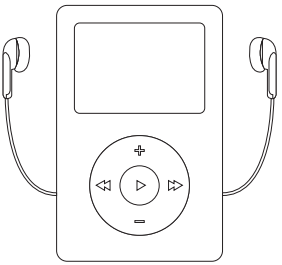
Sporadically Working
This is most likely due to a slightly dodgy connection with the potentiometer's pins. This can usually be conquered by holding the potentiometer down.

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.

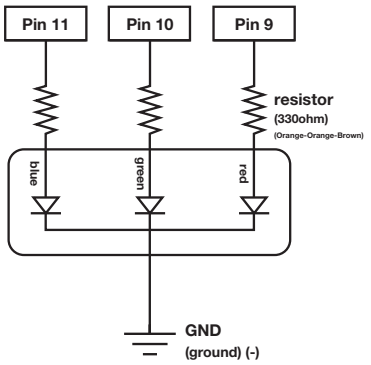
Real World Application:

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



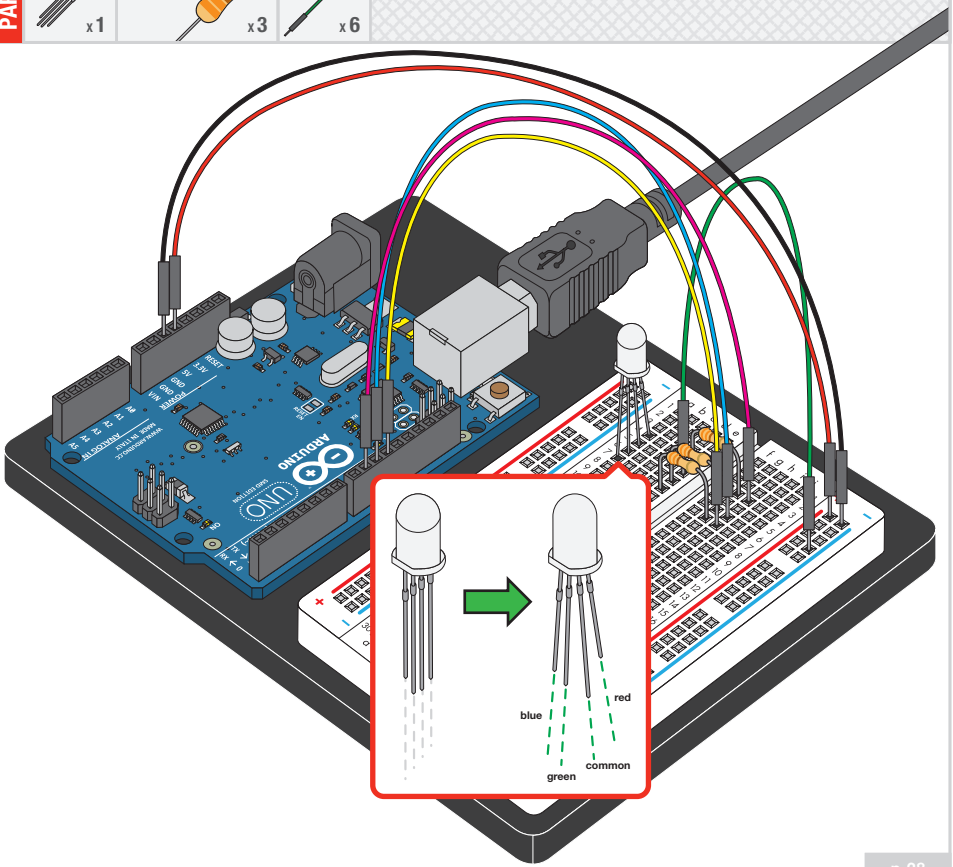
3

RGB LED

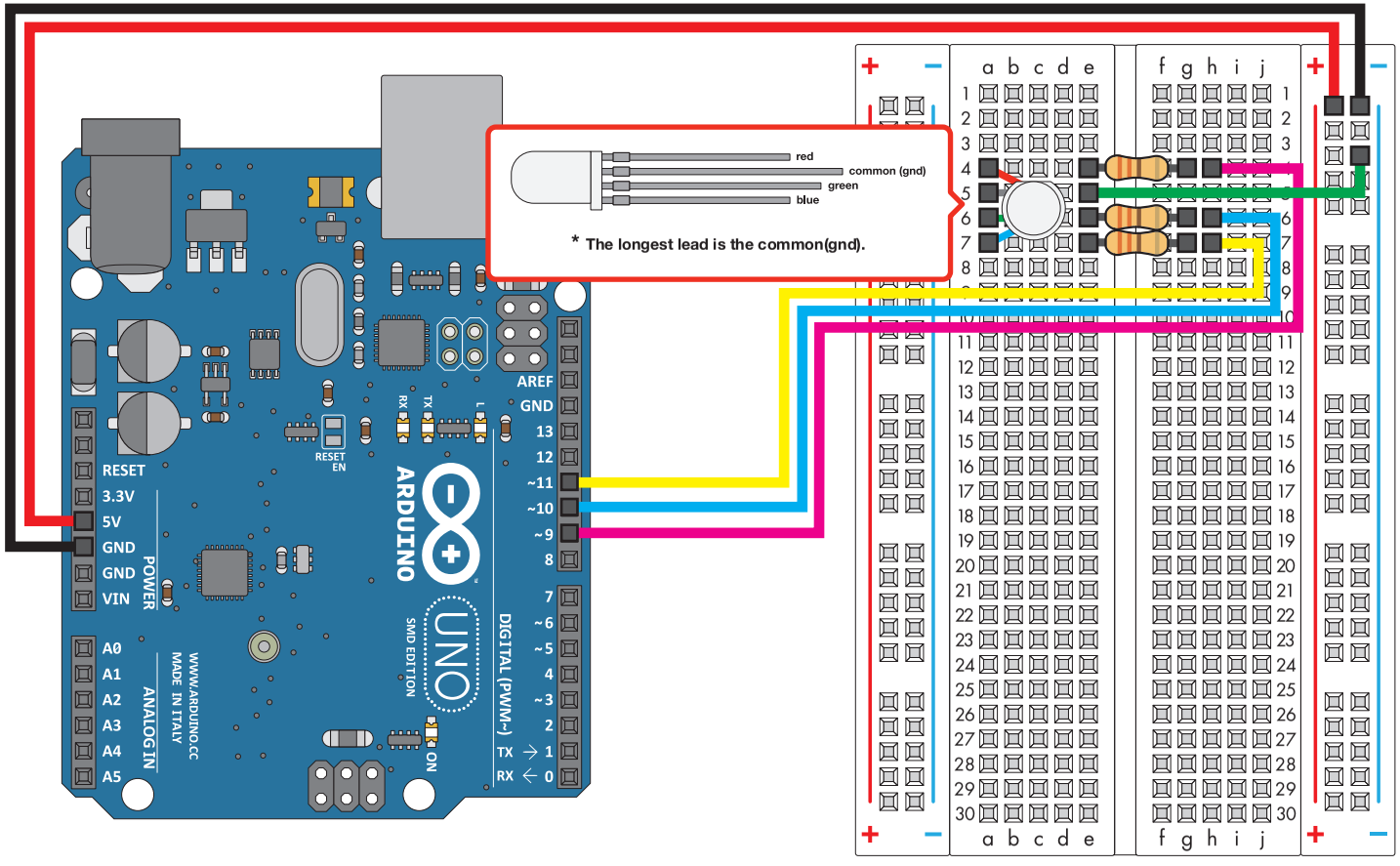


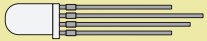
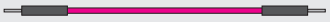
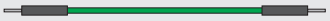
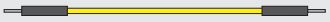
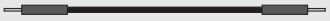
You know what's even more fun than a blinking LED? A colored one. RGB, or red-green-blue, LEDs have three different color-emitting diodes that can be combined to create all sorts of colors. In this circuit, you'll learn how to use an RGB LED to create unique color combinations. Depending on how bright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

PARTS:



Circuit 3: RGB LED



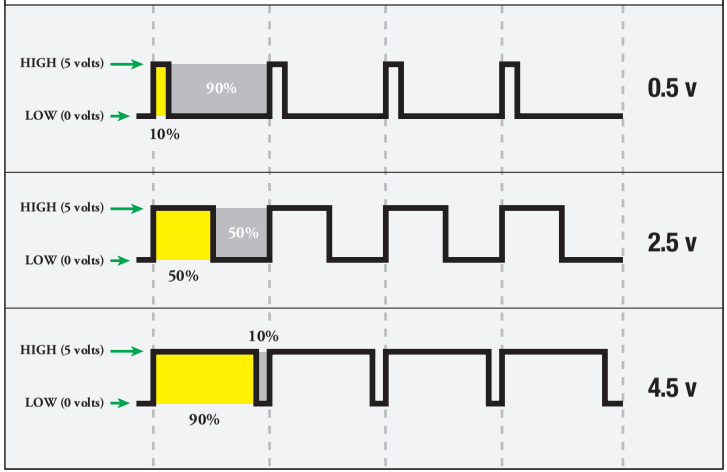
Component:	Image Reference:		
RGB LED (5mm) ⚠			a4 a5 a6 a7
330Ω Resistor			e4 - g4
330Ω Resistor			e6 - g6
330Ω Resistor			e7 - g7
Jumper Wire		Pin 9	h4
Jumper Wire			e5 - -
Jumper Wire		Pin 10	h6
Jumper Wire		Pin 11	h7
Jumper Wire		5V	+
Jumper Wire		GND	-

The shocking truth behind analogWrite():

We've seen that the Arduino can read analog voltages (voltages between 0 and 5 Volts) using the **analogRead()** function. Is there a way for the Arduino to output analog voltages as well?

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 **PWM ("Pulse-Width Modulation")**.

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.



3

Arduino Code:

Code to Note:

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

```
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 a 0, ends at 767, and increases by one each time ("x++").

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

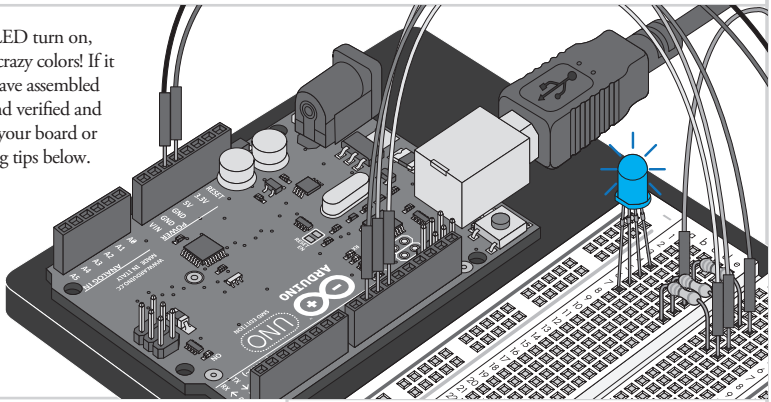
"If / else" statements are used to make choices in your programs. The statement within the parenthesis () is evaluated; if it's true, the code within the first brackets {} will run. If it's not true, the code within the second brackets {} will run.

```
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.

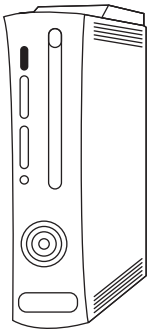
analogWrite(RED_PIN, redIntensity);

to

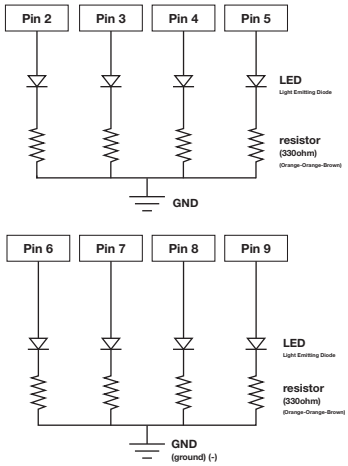
analogWrite(RED_PIN, redIntensity/3);

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 diffent colors represent different states of working condition.



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

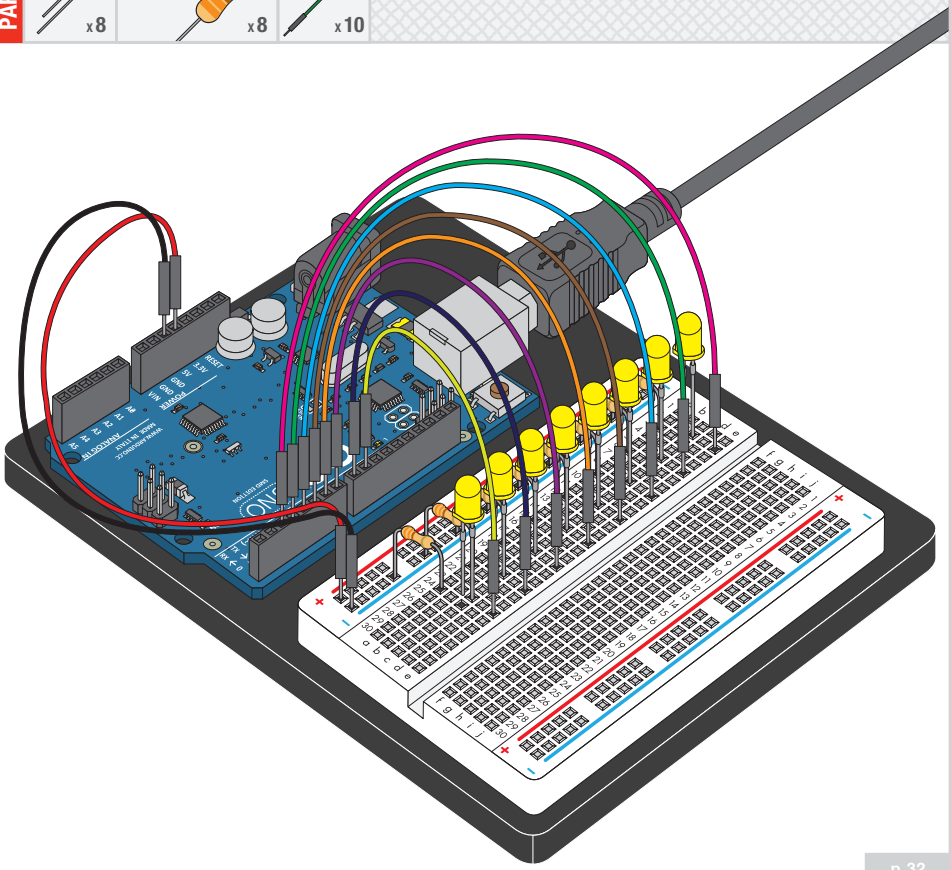
arrays[ ] - used to make managing variables easier by grouping them together

PARTS:

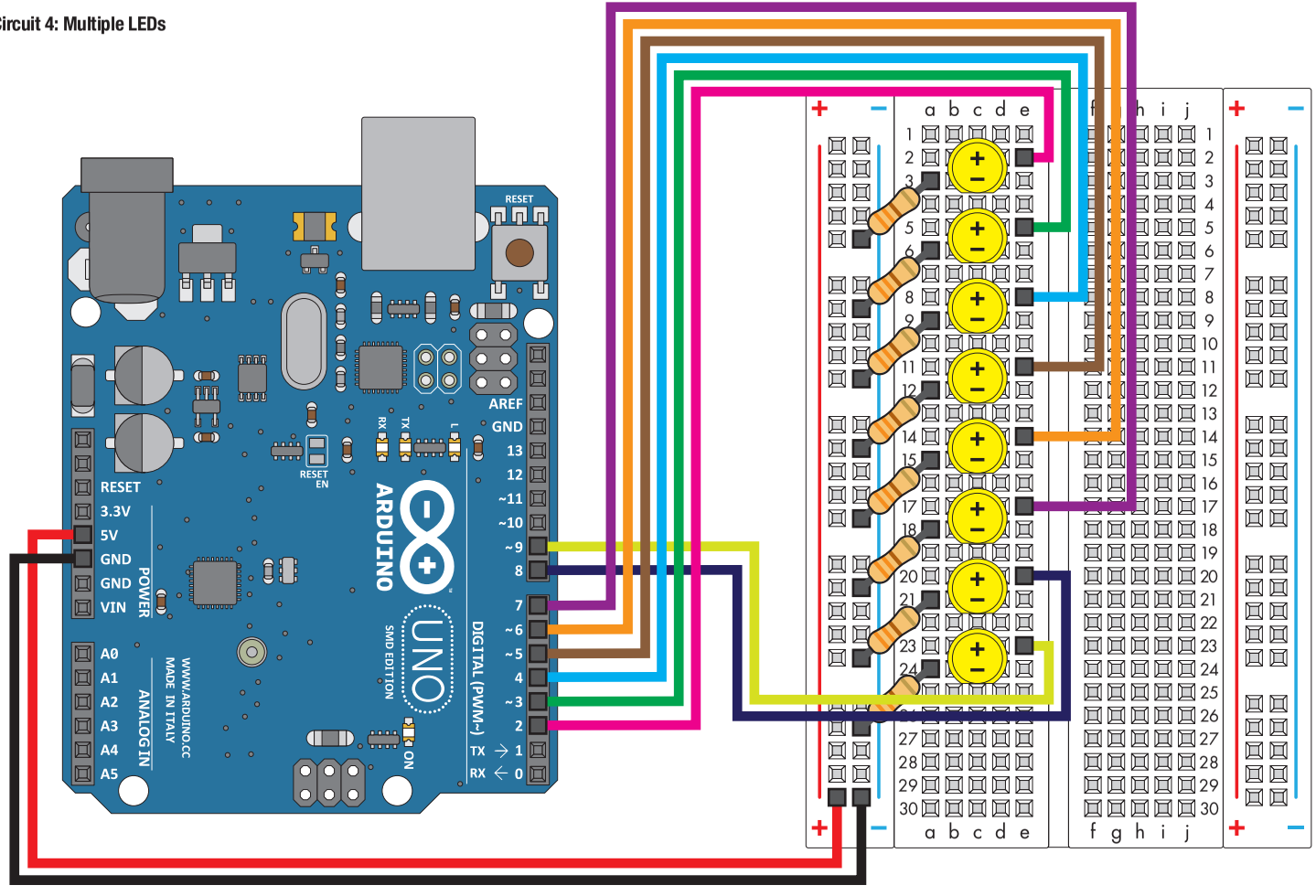

LED x8

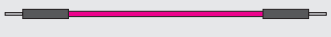
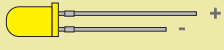
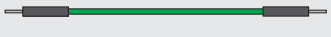
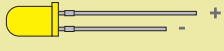
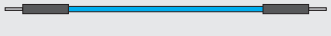
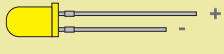
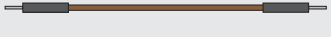
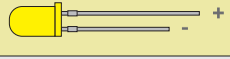
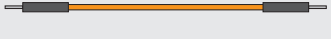
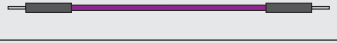
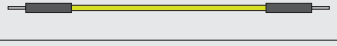

330Ω Resistor x8


Wire x10



Circuit 4: Multiple LEDs



Component:	Image Reference:			Component:	Image Reference:		
LED (5mm) ⚠			c2 + - c3	330Ω Resistor			GND - a3
LED (5mm) ⚠			c5 + - c6	330Ω Resistor			GND - a21
LED (5mm) ⚠			c8 + - c9	330Ω Resistor			GND - a24
LED (5mm) ⚠			c11 + - c12	Jumper Wire		Pin 2	e2
LED (5mm) ⚠			c14 + - c15	Jumper Wire		Pin 3	e5
LED (5mm) ⚠			c17 + - c18	Jumper Wire		Pin 4	e8
LED (5mm) ⚠			c20 + - c21	Jumper Wire		Pin 5	e11
LED (5mm) ⚠			c23 + - c24	Jumper Wire		Pin 6	e14
330Ω Resistor			c23 - c24	Jumper Wire		Pin 7	e17
330Ω Resistor			GND - a6	Jumper Wire		Pin 8	e20
330Ω Resistor			GND - a9	Jumper Wire		Pin 9	e23
330Ω Resistor			GND - a12	Jumper Wire		Pin 3	+
330Ω Resistor			GND - a15	Jumper Wire		GND	-

4

Arduino Code:

Open Arduino IDE // File > Examples > SIK Guide > Circuit # 4

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);

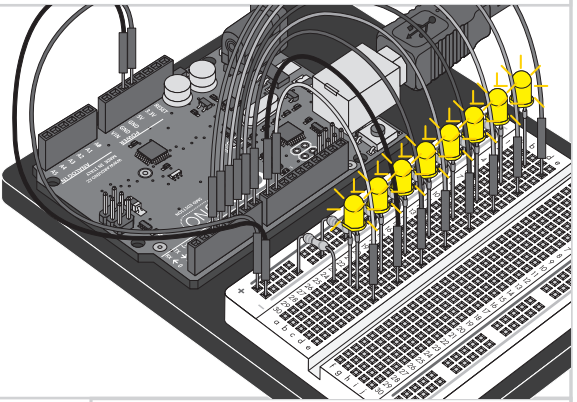
You refer to the elements in an array by their position. The first element is at position 0, the second is at position 1, etc. You refer to an element using "ledPins[x]" where x is the position. Here we're making digital pin 2 HIGH, since the array element at position 0 is "2".

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:

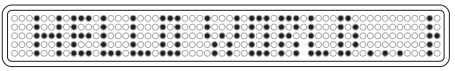
Some LEDs Fail to Light
It is easy to insert an LED backwards. Check the LEDs that aren't working and ensure they the right way around.

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 there after.

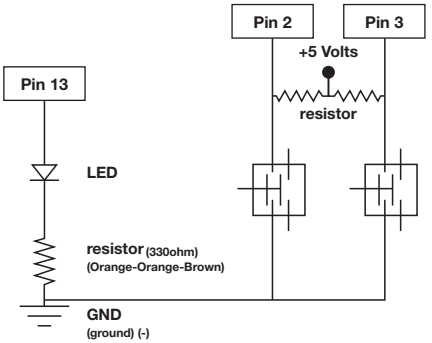
Starting Afresh
Its easy to accidentally misplace a wire without noticing. Pulling everything out and starting with a fresh slate is often easier than trying to track down the problem.

Real World Application:

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



5

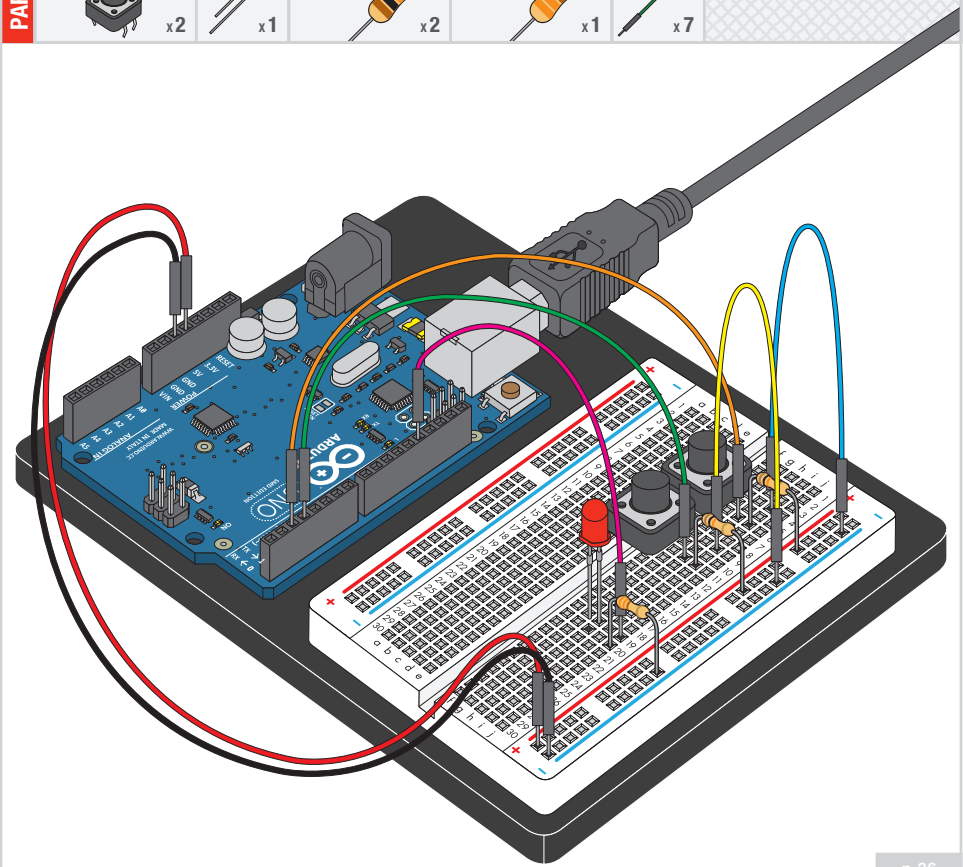


Push Buttons

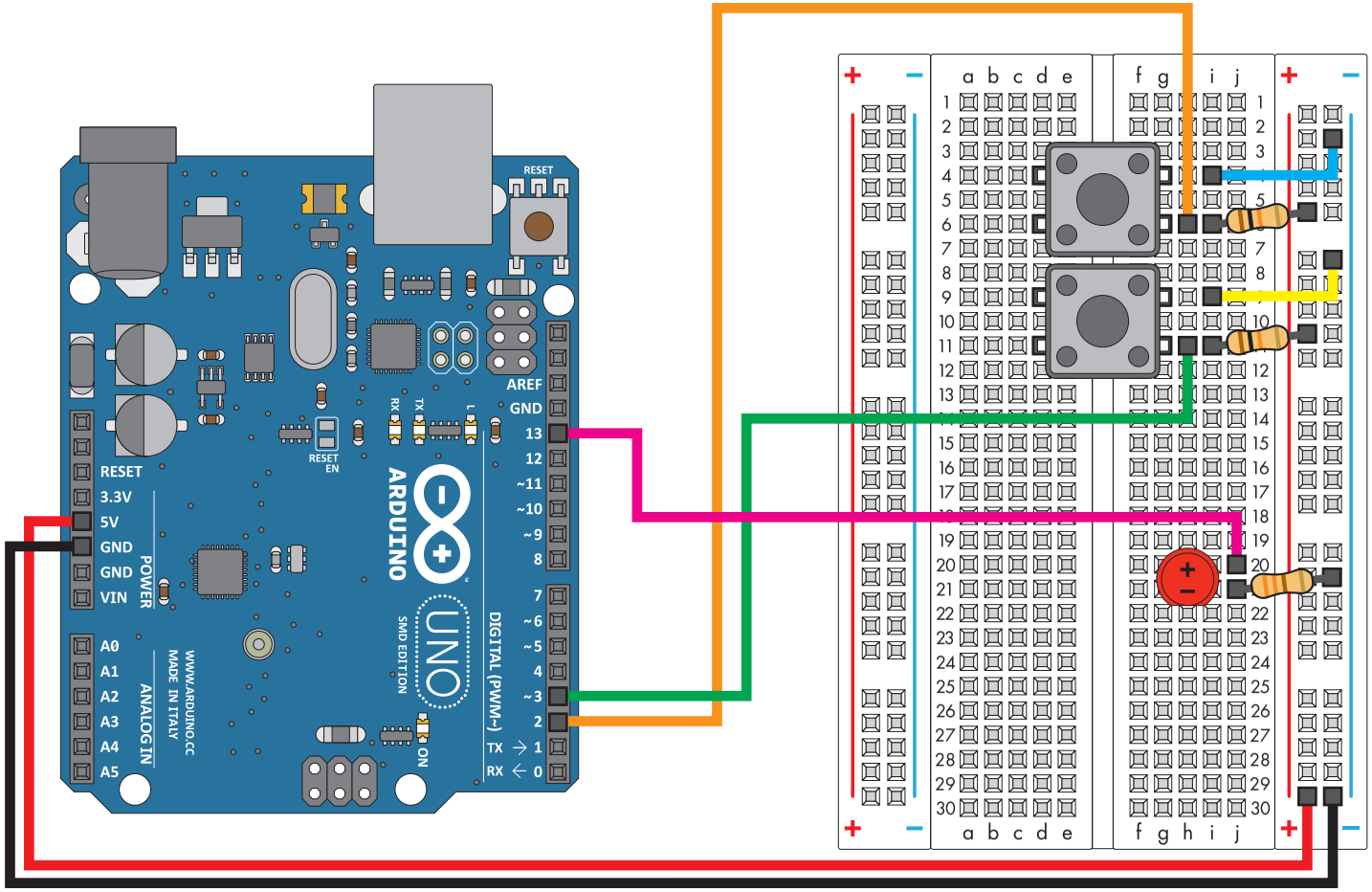
Up until now, we've focused solely on outputs. Now we're going to go to the other end of spectrum and play around with inputs. In this circuit, we'll be looking at one of the most common and simple inputs – a push button. The way a push button works with Arduino is that when the button is pushed, the 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.

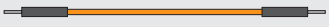
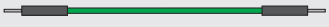
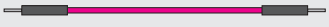
PARTS:

Push Button	LED	10KΩ Resistor	330Ω Resistor	Wire
x2	x1	x2	x1	x7



Circuit 5: Push Buttons



Component:	Image Reference:		
Push Button			d4 g4 d6 g6
Push Button			d9 g9 d11 g11
LED (5mm) ⚠			h20 - h21 + + -
10KΩ Resistor			i6 +
10KΩ Resistor			i11 +
330Ω Resistor			j21 -
Jumper Wire			i4 -
Jumper Wire			i9 -
Jumper Wire		Pin 2	h6
Jumper Wire		Pin 3	h11
Jumper Wire		Pin 13	j20

How to use logic like a Vulcan:

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:

==	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:

```
if ((mode == heat) && ((temperature < threshold) || (override == true)))
{
digitalWrite(HEATER, HIGH);
}
```

...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!

5

Arduino Code:

Code to Note:

Open Arduino IDE // File > Examples > SIK Guide > Circuit # 5

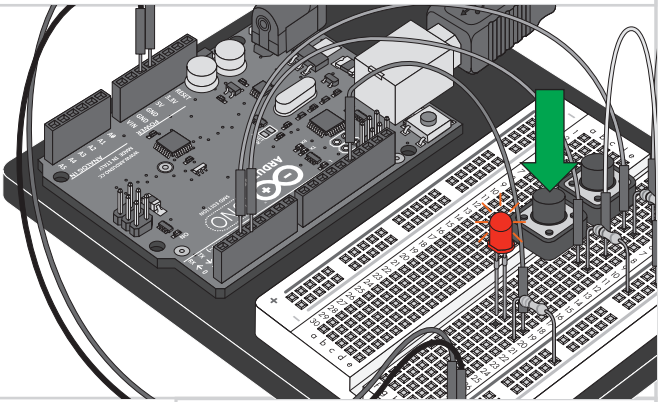
pinMode(button2Pin, INPUT); ⇨ The digital pins can be used as inputs as well as outputs. Before you do either, you need to tell the Arduino which direction you're going.

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:

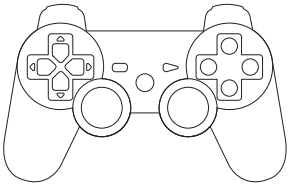
Light Not Turning On
The pushbutton is square, and because of this it is easy to put it in the wrong way. Give it a 90 degree twist and see if it starts working.

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.

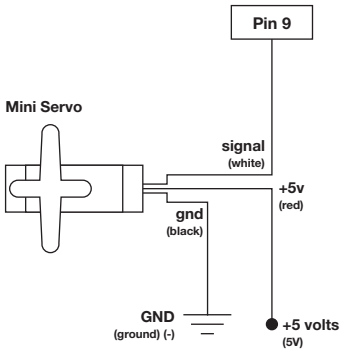
Real World Application:

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



8

A Single Servo



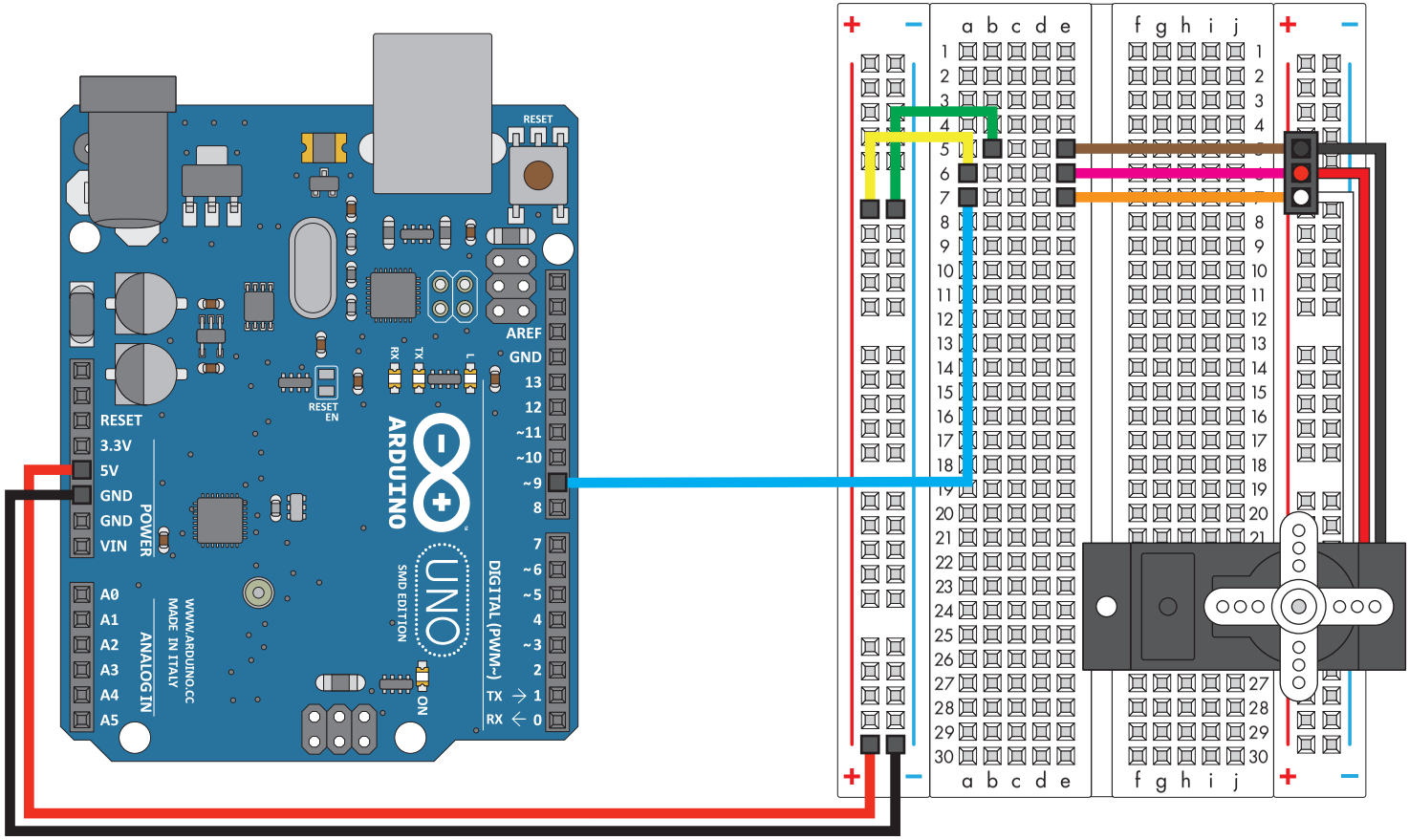
Servos are ideal for embedded electronics applications because they do one thing very well that spinning motors cannot – they can move to a position accurately. By varying the pulse of voltage a servo receives, you can move a servo to a specific position. For example, a pulse of 1.5 milliseconds will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

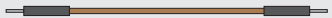
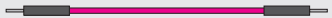
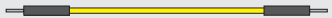
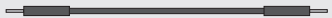
PARTS:


Servo x1


Wire x8

Circuit 8: A Single Servo



Component:	Image Reference:		
Servo			e5 e6 e7
Jumper Wire			e5
Jumper Wire			e6
Jumper Wire			e7
Jumper Wire		Pin 9	a7
Jumper Wire			b5 -
Jumper Wire			a6 +
Jumper Wire		5V	+
Jumper Wire		GND	-

Expand your horizons using Libraries:

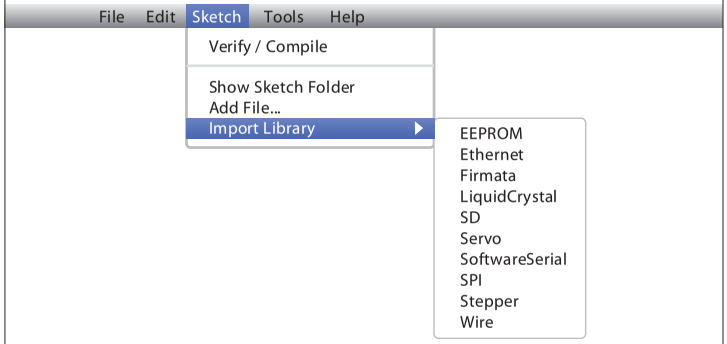
Arduino gives you a very useful set of built-in commands for doing basic input and output, making decisions using logic, solving math problems, etc. But the real power of Arduino is the huge community using it, and their willingness to share their work.

Libraries are collections of new commands that have been packaged together to make it easy to include them in your sketches. Arduino comes with a handful of useful libraries, such as the servo library used in this example, that can be used to interface to more advanced devices (LCD displays, stepper motors, ethernet ports, etc.)

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**.



8

Arduino Code:

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.

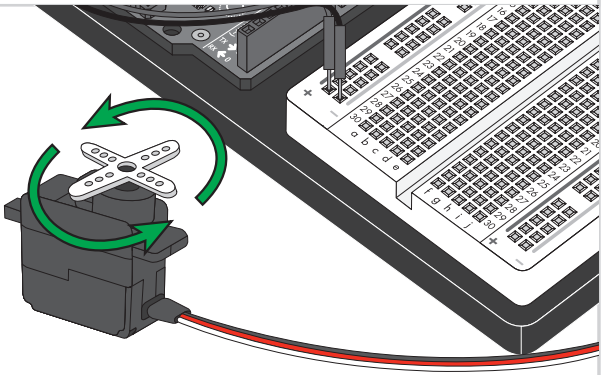
Servo servo1;

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:

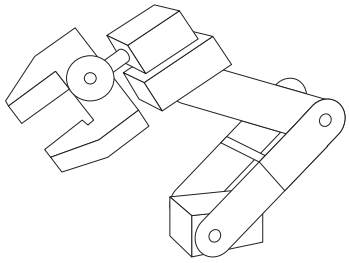
Servo Not Twisting
Even with colored wires it is still shockingly easy to plug a servo in backward. This might be the case.

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.

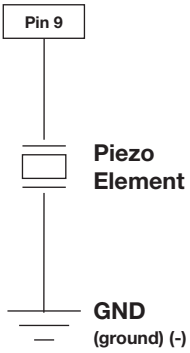
Real World Application:

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



11

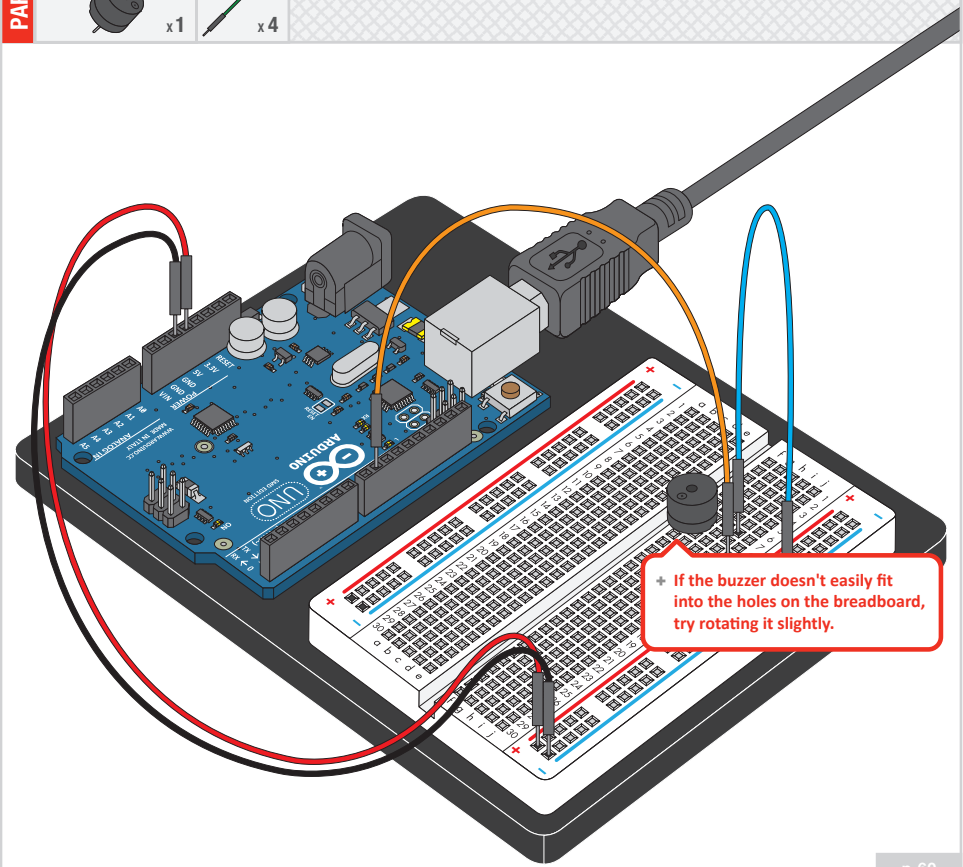
Buzzer



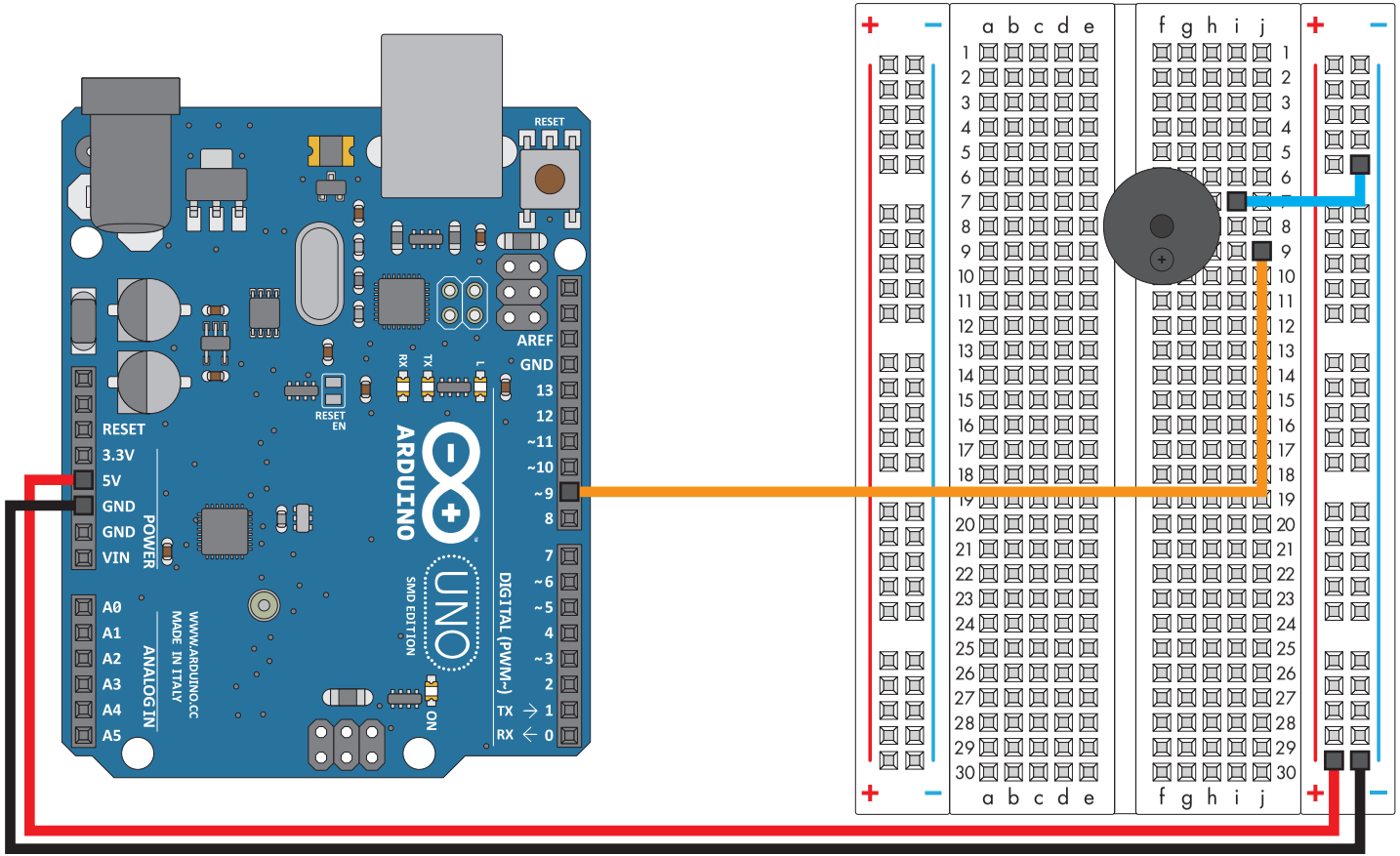
In this circuit, we'll again bridge the gap between the digital world and the analog world. We'll be using a buzzer that makes a small "click" when you apply voltage to it (try it!). By itself that isn't terribly exciting, but if you turn the voltage on and off hundreds of times a second, the buzzer 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!

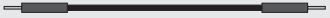
PARTS:

- Piezo Element x1
- Wire x4



Circuit 11: Piezo Elements



Component:	Image Reference:		
Piezo Element			j9 + j7 -
Jumper Wire			j7 -
Jumper Wire		Pin 9	j9
Jumper Wire		5V	+
Jumper Wire		GND	-

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.

11

Arduino Code:

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'};
```

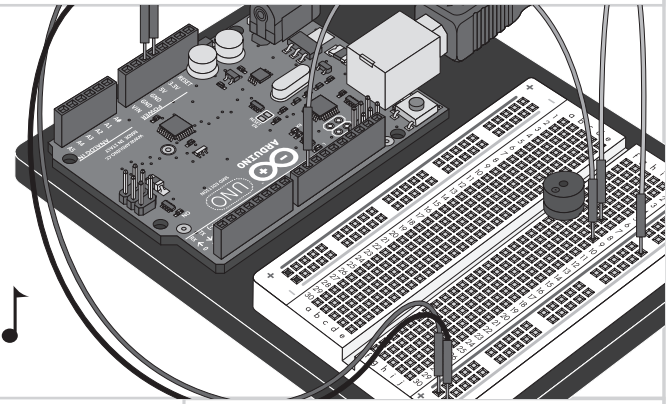
Up until now we've been working solely with numerical data, but the Arduino can also work with text. Characters (single, printable, letters, numbers and other symbols) have their own type, called "char". When you have an array of characters, it can be defined between double-quotes (also called a "string"),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

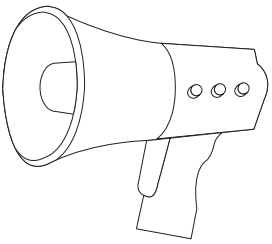
No Sound
Given the size and shape of the piezo element it is easy to miss the right holes on the breadboard. Try double checking its placement.

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:

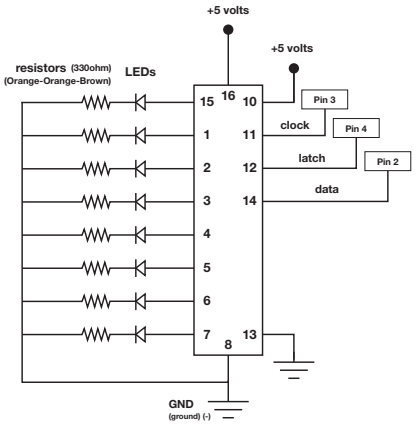
Many modern megaphones have settings that use a loud amplified buzzer. They are usually very loud and quite good at getting people's attention.



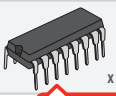
14

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.



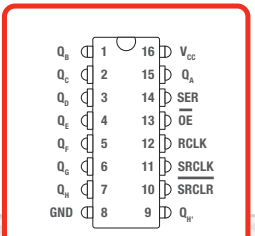
PARTS:


IC x1

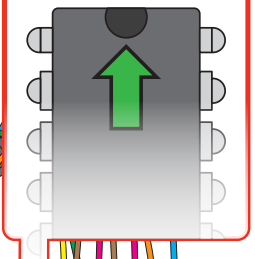

LED x8

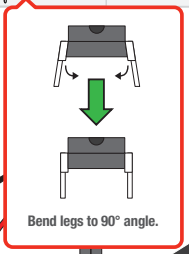

330Ω Resistor x8


Wire x19

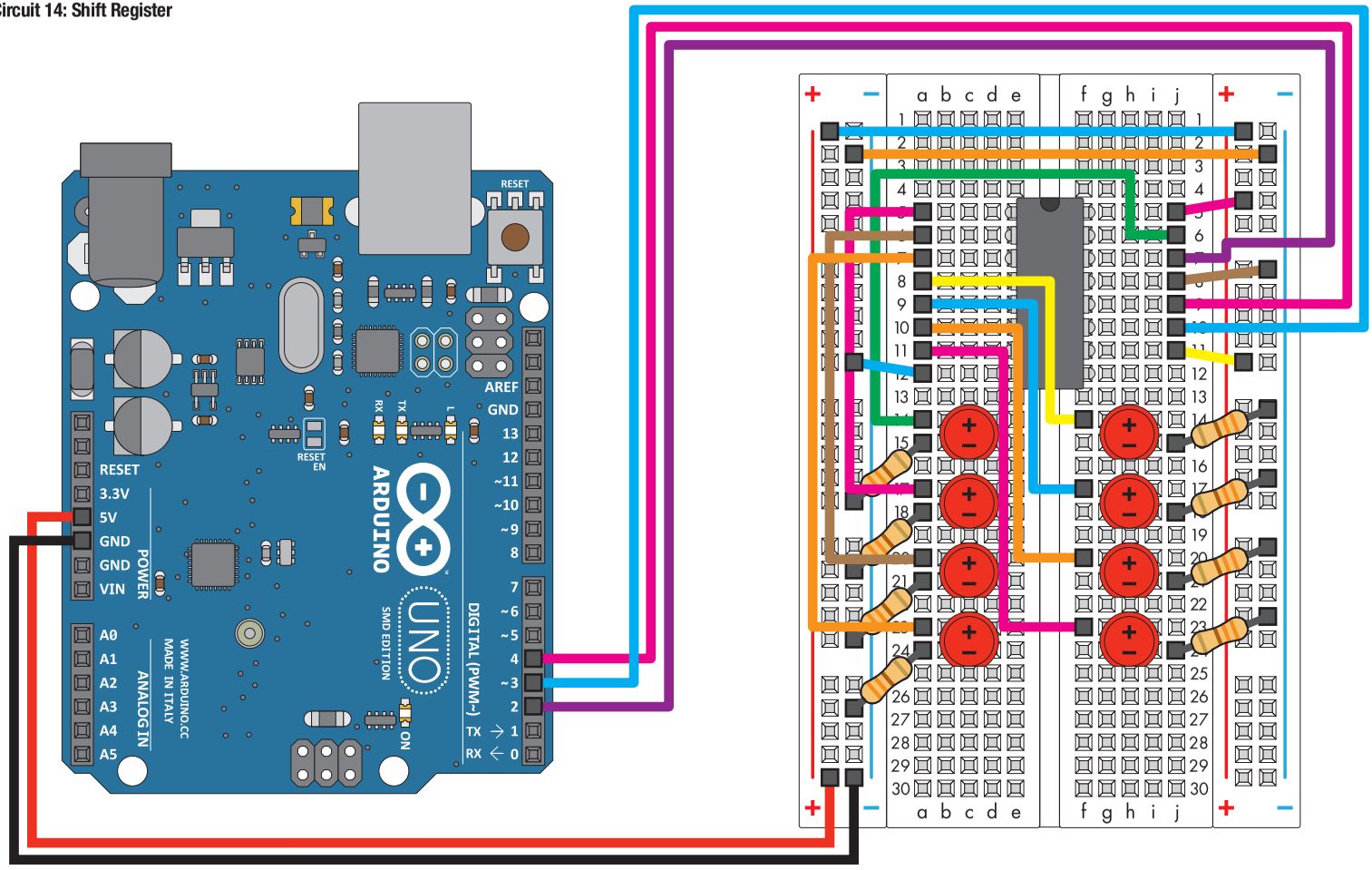


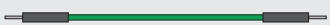
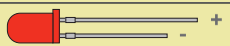
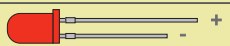
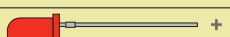
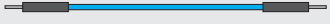
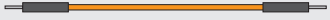
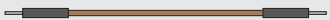
Align notch on top, inbetween "e5" and "f5" on the breadboard.




Bend legs to 90° angle.

Circuit 14: Shift Register



Component:	Image Reference:			Component:	Image Reference:		
IC		e5 e6 e7 e8 e9 e10 e11 e12 f5 f6 f7 f8 f9 f10 f11 f12		Jumper Wire			- -
LED (5mm)			c14 c15 + -	Jumper Wire			j5 +
LED (5mm)			c17 c18 + -	Jumper Wire			j6 a14
LED (5mm)			c20 c21 + -	Jumper Wire		Pin 2	j7
LED (5mm)			c23 c24 + -	Jumper Wire			j8 -
LED (5mm)			h14 h15 + -	Jumper Wire		Pin 4	j9
LED (5mm)			h17 h18 + -	Jumper Wire		Pin 3	j10
LED (5mm)			h20 h21 + -	Jumper Wire			j11 +
LED (5mm)			h23 h24 + -	Jumper Wire			f14 a8
330Ω Resistor			- a15	Jumper Wire			f17 a9
330Ω Resistor			- a18	Jumper Wire			f20 a10
330Ω Resistor			- a21	Jumper Wire			f23 a11
330Ω Resistor			- a24	Jumper Wire			a23 a7
330Ω Resistor			j15 -	Jumper Wire			a20 a6
330Ω Resistor			j18 -	Jumper Wire			a17 a5
330Ω Resistor			j21 -	Jumper Wire			a14 j6
330Ω Resistor			j24 -	Jumper Wire		5V	+
Jumper Wire			+ +	Jumper Wire		GND	-

14

Arduino Code:

Open Arduino IDE // File > Examples > SIK Guide > Circuit # 14

Code to Note:

shiftOut(datapin, clockpin, MSBFIRST, data);

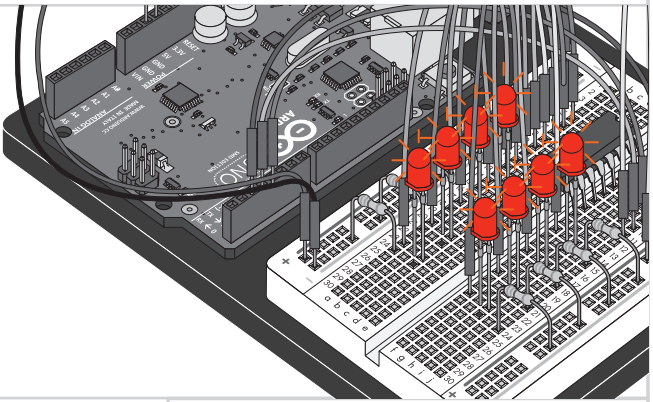
You'll communicate with the shift register (and a lot of other parts) using an interface called SPI, or Serial Peripheral Interface. This interface uses a data line and a separate clock line that work together to move data in or out of the Arduino at high speed. The MSBFIRST parameter specifies the order 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**

Real World Application:

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