

**Introduction to Arduino**  
SparkFun Electronics Summer Semester

## Installing Arduino

Mac platform	PC platform
<p>1. Double click the file <b>arduino-0022.dmg</b> inside the folder \SIK Applications\Mac\</p> <p>2. Go to “Arduino” in the devices section of the finder and move the “Arduino” application to the “Applications” folder.</p> <p>3. Go to the “Arduino” device, double click and install:  <b>“FTDI drivers for Intel Macs 0022.pkg”</b>  or  <b>“FTDI drivers for PPC Macs 0022.pkg”</b>  then Restart your computer.</p> <p>4. Plug your Arduino board into a free USB port using the USB cord provided.</p>	<p>1. Unzip the file <b>arduino-0022</b> inside the folder \SIK Applications\PC\. We recommend unzipping to your c:\Program Files\ directory.</p> <p>2. Open the folder containing your unzipped Arduino files and create a shortcut to Arduino.exe. Place this on your desktop for easy access.</p> <p>3. Plug your Arduino board into a free USB port using the USB cord provided. Wait for a pop up box about installing drivers.</p> <p>4. Skip “searching the internet”. Click “Install from a list or specific location” in the advanced section. Choose the location c:\program files\arduino-0022\drivers\Arduino Uno\   <b>(You may have to do this last step more than once)</b>  <b>(If you are using the Duemilanove you will have to choose the sub-directory, FTDI USB Drivers and you will have to do this twice)</b></p>

**Issues with Java? Make sure you have the latest version of Java installed.**

**Otherwise....**

**Ta da! You're ready to open the Arduino programming environment.**

**(For Linux info go to <http://ardx.org/LINU>)**

## Introduction to Arduino

### SparkFun Electronics Summer Semester

# A few notes about Arduino setup

## Selecting your board:

You are using the Arduino Uno board with an ATmega 328 micro-controller. This means you will need to select “Arduino Uno” as your board. To do this you click on the “Tools” menu tab, then click the “Board” tab and select “Arduino Uno”. If you are using a different board you will need to select the correct model in order to properly upload to your board.

## Selecting your com port:

Another option that is necessary to change occasionally is your “Serial Port”. This can also be found under the “Tools” menu tab. When you click on this tab you should be presented with at least one com port labeled “COM1” (or “COM2, etc....”) This indicates which USB port your board is plugged into. Sometimes you will need to make sure you are using the correct com ports for your Arduino, here is some information on your com ports depending on which platform you are using:

Mac	PC
The Mac version of Arduino refreshes your com port list every time you plug in a device. For this reason all you really need to do is select the com port called “/dev/cu.usbserial-XXXX” where XXXX will be a value that changes.	The PC version of Arduino creates a new com port for every distinct board you plug into your computer. You will need to find out which com port is the board you are currently trying to use. 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.

### A few more tidbits that will help to know....

There are seven buttons that look like this at the top of your Arduino window:



Here is what they do from left to right:

Compile	Stop	New	Open	Save	Upload	Serial Monitor
This checks your code for errors.	This stops the program.	This creates a new sketch.	This opens an existing sketch.	This saves the open sketch.	This uploads the sketch to your board.	Used to display Serial Communication.

For any words or phrases you are unfamiliar with try the Glossary in the back.

## Basic Arduino Reference Sheet

**Installation:** Arduino: <http://www.arduino.cc/en/Guide/HomePage>

**Support:** Arduino: <http://www.arduino.cc>, <http://www.freeduino.org>, [google.com](http://google.com)

**Forums:** Arduino: <http://forum.sparkfun.com/viewforum.php?f=32>

### Basic Arduino code definitions:

**setup( ):** A function present in every Arduino sketch. Run once before the loop( ) function. Often used to set pinmode to input or output. The setup( ) function looks like:

```
void setup( ){
    //code goes here
}
```

**loop( ):** A function present in every single Arduino sketch. This code happens over and over again. The loop( ) is where (almost) everything happens. The one exception to this is setup( ) and variable declaration. ModKit uses another type of loop called “forever( )” which executes over Serial. The loop( ) function looks like:

```
void loop( ) {
    //code goes here
}
```

**input:** A pin mode that intakes information.

**output:** A pin mode that sends information.

**HIGH:** Electrical signal present (5V for Uno). Also ON or True in boolean logic.

**LOW:** No electrical signal present (0V). Also OFF or False in boolean logic.

**digitalRead:** Get a HIGH or LOW reading from a pin already declared as an input.

**digitalWrite:** Assign a HIGH or LOW value to a pin already declared as an output.

**analogRead:** Get a value between or including 0 (LOW) and 1023 (HIGH). This allows you to get readings from analog sensors or interfaces that have more than two states.

**analogWrite:** Assign a value between or including 0 (LOW) and 255 (HIGH). This allows you to set output to a PWM value instead of just HIGH or LOW.

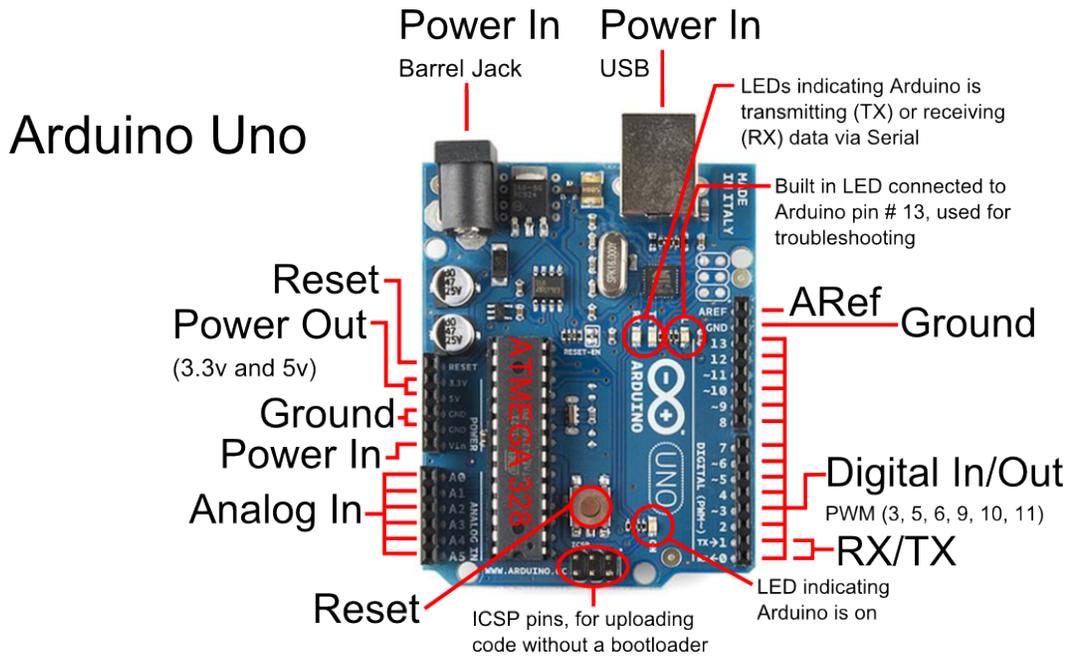
**PWM:** Stands for Pulse-Width Modulation, a method of emulating an analog signal through a digital pin. A value between or including 0 and 255. Used with analogWrite.

### Arduino Uno pin type definitions: (Take a look at your Arduino board)

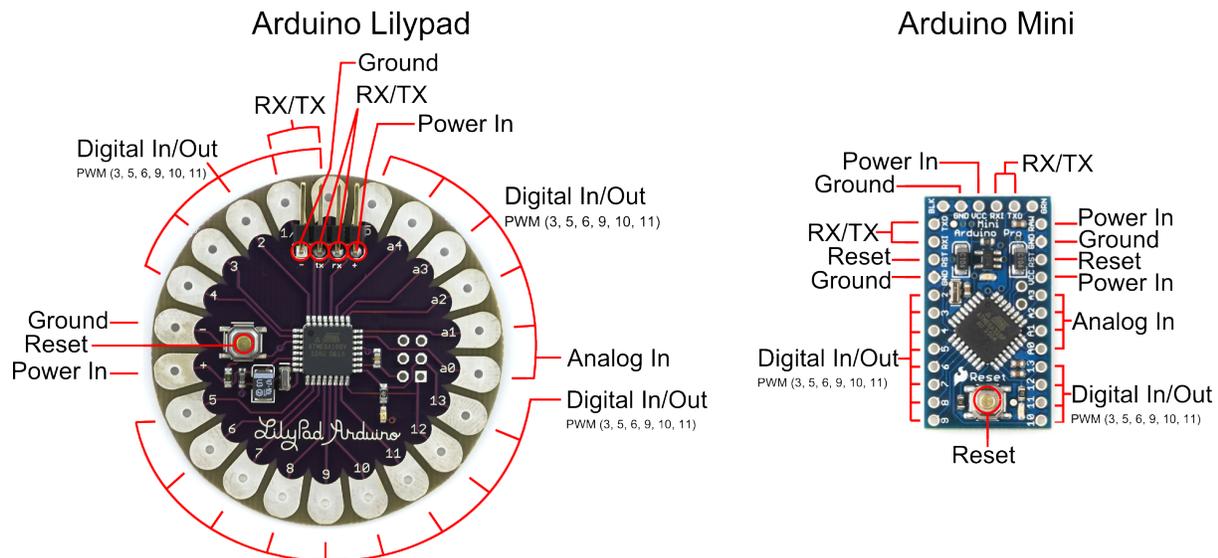
Reset	3v3	5v	Gnd	Vin	Analog In	RX/TX	Digital	PWM(~)	AREF
Resets Arduino sketch on board	3.3 volts in and out	5 volts in and out	Ground	Voltage in for sources over 7V (9V - 12V)	Analog inputs, can also be used as Digital	Serial comm. Receive and Transmit	Input or output, HIGH or LOW	Digital pins with output option of PWM	External reference voltage used for analog

**Introduction to Arduino**  
 SparkFun Electronics Summer Semester

## Basic Arduino Pin Reference Sheet



These boards below use the same micro-controller, just in a different package. The Lilypad is designed for use with conductive thread instead of wire and the Arduino Mini is simply a smaller package without the USB, Barrel Jack and Power Outs. Other boards in the Arduino family can be found at <http://arduino.cc/en/Main/Hardware>.



## Introduction to Arduino

### SparkFun Electronics Summer Semester

## Analog

A continuous stream of information with values between and including 0% and 100%.

Humans perceive the world in analog. Everything we see and hear is a continuous transmission of information to our senses. The temperatures we perceive are never 100% hot or 100% cold, they are constantly changing between our ranges of acceptable temperatures. (And if they are out of our range of acceptable temperatures then what are we doing there?) This continuous stream is what defines analog data. Digital information, the complementary concept to Analog, estimates analog data using only ones and zeros.

In the world of Arduino an Analog signal is simply a signal that can be HIGH (on), LOW (off) or anything in between these two states. This means an Analog signal has a voltage value that can be anything between 0V and 5V (unless you mess with the Analog Reference pin). Analog allows you to send output or receive input about devices that run at percentages as well as on and off. The Arduino does this by sampling the voltage signal sent to these pins and comparing it to a voltage reference signal (5V). Depending on the voltage of the Analog signal when compared to the Analog Reference signal the Arduino then assigns a numerical value to the signal somewhere between 0 (0%) and 1023 (100%). The digital system of the Arduino can then use this number in calculations and sketches.

To receive Analog Input the Arduino uses Analog pins # 0 - # 5. These pins are designed for use with components that output Analog information and can be used for Analog Input. There is no setup necessary, and to read them use the command:

```
analogRead(pinNumber);
```

where pinNumber is the Analog In pin to which the the Analog component is connected. The analogRead command will return a number including or between 0 and 1023.

The Arduino also has the capability to output a digital signal that acts as an Analog signal, this signal is called Pulse Width Modulation (PWM). Digital Pins # 3, # 5, # 6, # 9, # 10 and #11 have PWM capabilities. To output a PWM signal use the command:

```
analogWrite(pinNumber, value);
```

where pinNumber is a Digital Pin with PWM capabilities and value is a number between 0 (0%) and 255 (100%). On the Arduino UNO PWM pins are signified by a ~ sign. For more information on PWM see the PWM worksheets or S.I.K. circuit 12.

### Examples of Analog:

Values: Temperature, volume level, speed, time, light, tide level, spiciness, the list goes on....

Sensors: Temperature sensor, Photoresistor, Microphone, Turntable, Speedometer, etc....

### Things to remember about Analog:

Analog Input uses the Analog In pins, Analog Output uses the PWM pins

To receive an Analog signal use: *analogRead*(pinNumber);

To send a PWM signal use: *analogWrite*(pinNumber, value);

Analog Input values range from 0 to 1023 (1024 values because it uses 10 bits,  $2^{10}$ )

PWM Output values range from 0 to 255 (256 values because it uses 8 bits,  $2^8$ )

## Digital

An electronic signal transmitted as binary code that can be either the presence or absence of current, high and low voltages or short pulses at a particular frequency.

Humans perceive the world in analog, but robots, computers and circuits use Digital. A digital signal is a signal that has only two states. These states can vary depending on the signal, but simply defined the states are ON or OFF, never in between.

In the world of Arduino, Digital signals are used for everything with the exception of Analog Input. Depending on the voltage of the Arduino the ON or HIGH of the Digital signal will be equal to the system voltage, while the OFF or LOW signal will always equal 0V. This is a fancy way of saying that on a 5V Arduino the HIGH signals will be a little under 5V and on a 3.3V Arduino the HIGH signals will be a little under 3.3V.

To receive or send Digital signals the Arduino uses Digital pins # 0 - # 13. You may also setup your Analog In pins to act as Digital pins. To set up Analog In pins as Digital pins use the command:

```
pinMode(pinNumber, value);
```

where pinNumber is an Analog pin (A0 – A5) and value is either INPUT or OUTPUT. To setup Digital pins use the same command but reference a Digital pin for pinNumber instead of an Analog In pin. Digital pins default as input, so really you only need to set them to OUTPUT in pinMode. To read these pins use the command:

```
digitalRead(pinNumber);
```

where pinNumber is the Digital pin to which the Digital component is connected. The digitalRead command will return either a HIGH or a LOW signal. To send a Digital signal to a pin use the command:

```
digitalWrite(pinNumber, value);
```

where pinNumber is the number of the pin sending the signal and value is either HIGH or LOW.

The Arduino also has the capability to output a Digital signal that acts as an Analog signal, this signal is called Pulse Width Modulation (PWM). Digital Pins # 3, # 5, # 6, # 9, # 10 and #11 have PWM capabilities. To output a PWM signal use the command:

```
analogWrite(pinNumber, value);
```

where pinNumber is a Digital Pin with PWM capabilities and value is a number between 0 (0%) and 255 (100%). For more information on PWM see the PWM worksheets or S.I.K. circuit 12.

### Examples of Digital:

Values: On/Off, Men's room/Women's room, pregnancy, consciousness, the list goes on....

Sensors/Interfaces: Buttons, Switches, Relays, CDs, etc....

### Things to remember about Digital:

Digital Input/Output uses the Digital pins, but Analog In pins can be used as Digital

To receive a Digital signal use: `digitalRead(pinNumber);`

To send a Digital signal use: `digitalWrite(pinNumber, value);`

Digital Input and Output are always either HIGH or LOW

## Input Signals

A signal entering an electrical system, in this case a micro-controller. Input to the Arduino pins can come in one of two forms; Analog Input or Digital Input.

Analog Input enters your Arduino through the Analog In pins # 0 - # 5. These signals originate from analog sensors and interface devices. These analog sensors and devices use voltage levels to communicate their information instead of a simple yes (HIGH) or no (LOW). For this reason you cannot use a digital pin as an input pin for these devices. Analog Input pins are used only for receiving Analog signals. It is only possible to read the Analog Input pins so there is no command necessary in the `setup( )` function to prepare these pins for input. To read the Analog Input pins use the command:

```
analogRead(pinNumber);
```

where `pinNumber` is the Analog Input pin number. This function will return an Analog Input reading between 0 and 1023. A reading of zero corresponds to 0 Volts and a reading of 1023 corresponds to 5 Volts. These voltage values are emitted by the analog sensors and interfaces. If you have an Analog Input that could exceed  $V_{cc} + .5V$  you may change the voltage that 1023 corresponds to by using the Aref pin. This pin sets the maximum voltage parameter your Analog Input pins can read. The Aref pin's preset value is 5V.

Digital Input can enter your Arduino through any of the Digital Pins # 0 - # 13. Digital Input signals are either HIGH (On, 5V) or LOW (Off, 0V). Because the Digital pins can be used either as input or output you will need to prepare the Arduino to use these pins as inputs in your `setup( )` function. To do this type the command:

```
pinMode(pinNumber, INPUT);
```

inside the curly brackets of the `setup( )` function where `pinNumber` is the Digital pin number you wish to declare as an input. You can change the `pinMode` in the `loop( )` function if you need to switch a pin back and forth between input and output, but it is usually set in the `setup( )` function and left untouched in the `loop( )` function. To read the Digital pins as inputs use:

```
digitalRead(pinNumber);
```

where `pinNumber` is the Digital Input pin number.

Input can come from many different devices, but each device's signal will be either Analog or Digital, it is up to the user to figure out which kind of input is needed, hook up the hardware and then type the correct code to properly use these signals.

### Things to remember about Input:

Input is either Analog or Digital, make sure to use the correct pins depending on type.

To take an Input reading use `analogRead(pinNumber);` (for analog)

Or `digitalRead(pinNumber);` (for digital)

Digital Input needs a `pinMode` command such as `pinMode(pinNumber, INPUT);`

Analog Input varies from 0 to 1023

Digital Input is always either HIGH or LOW

### Examples of Input:

Push Buttons, Potentiometers, Photoresistors, Flex Sensors

## Output Signals

A signal exiting an electrical system, in this case a micro-controller.

Output to the Arduino pins is always Digital, however there are two different types of Digital Output; regular Digital Output and Pulse Width Modulation Output (PWM). Output is only possible with Digital pins # 0 - # 13. The Digital pins are preset as Output pins, so unless the pin was used as an Input in the same sketch, there is no reason to use the `pinMode` command to set the pin as an Output. Should a situation arise where it is necessary to reset a Digital pin to Output from Input use the command:

```
pinMode(pinNumber, OUTPUT);
```

where `pinNumber` is the Digital pin number set as Output. To send a Digital Output signal use the command:

```
digitalWrite(pinNumber, value);
```

where `pinNumber` is the Digital pin that is outputting the signal and `value` is the signal. When outputting a Digital signal value can be either HIGH (On) or LOW (Off).

Digital Pins # 3, # 5, # 6, # 9, # 10 and #11 have PWM capabilities. This means you can Output the Digital equivalent of an Analog signal using these pins. To Output a PWM signal use the command:

```
analogWrite(pinNumber, value);
```

where `pinNumber` is a Digital Pin with PWM capabilities and `value` is a number between 0 (0%) and 255 (100%). For more information on PWM see the PWM worksheets or S.I.K. circuit 12.

Output can be sent to many different devices, but it is up to the user to figure out which kind of Output signal is needed, hook up the hardware and then type the correct code to properly use these signals.

### Things to remember about Output:

Output is always Digital

There are two kinds of Output: regular Digital or PWM (Pulse Width Modulation)

To send an Output signal use `analogWrite(pinNumber, value);` (for analog) or `digitalWrite(pinNumber, value);` (for digital)

Output pin mode is set using the `pinMode` command: `pinMode(pinNumber, OUTPUT);`

Regular Digital Output is always either HIGH or LOW

PWM Output varies from 0 to 255

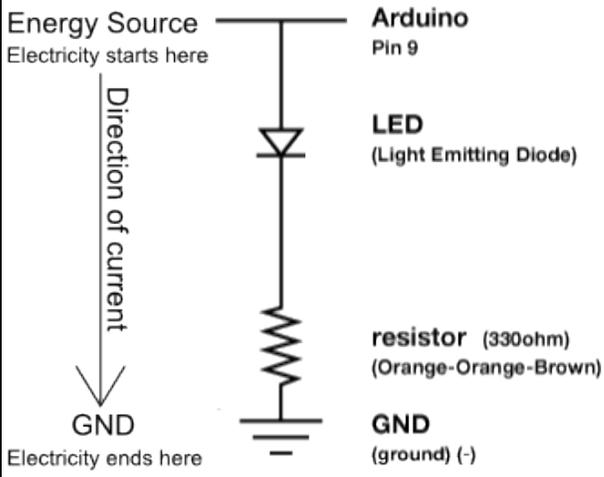
### Examples of Output:

Light Emitted Diodes (LED's), Piezoelectric Speakers, Servo Motors

## Introduction to Arduino

### SparkFun Electronics Summer Semester

## Circuit 1

<p><b>Explanation:</b></p> <p>This circuit takes electricity from digital Pin # 9 on the Arduino. Pin # 9 on the Arduino has Pulse Width Modulation capability allowing the user to change the brightness of the LED when using analogWrite. The LED is connected to the circuit so electricity enters through the anode (+, or longer wire) and exits through the cathode (-, or shorter wire). The resistor dissipates current so the LED does not draw current above the maximum rating and burn out. Finally the electricity reaches ground, closing the circuit and allowing electricity to flow from power source to ground.</p>	<p><b>Schematic:</b></p> 
<p><b>Components:</b></p> <p>Arduino Digital Pin # 9: Power source, PWM (if code uses analogWrite) or digital (if code uses digitalWrite) output from Arduino board.</p> <p>LED: As in other diodes, current flows easily from the + side, or anode (longer wire), to the - side, or cathode (shorter wire), but not in the reverse direction. Also lights up!</p> <p>330 Ohm Resistor: A resistor resists the current flowing through the circuit. In this circuit the resistor reduces the current so the LED does not burn out.</p> <p>Gnd: Ground</p>	<p><b>Code:</b></p> <pre>int ledPin = 3; void setup() {   pinMode(ledPin, OUTPUT); } void loop() {   digitalWrite(ledPin, HIGH); //LED on   delay(1000); // wait second   digitalWrite(ledPin, LOW); //LED off   delay(1000); // wait second } </pre> <p>or for PWM output loop could read :</p> <pre>int ledPin = 3; void setup() {   pinMode(ledPin, OUTPUT); } void loop() {   analogWrite(ledPin, 255); // LED on   delay(1000); // wait second   analogWrite(ledPin, 0); // LED off   delay(1000); // wait second } </pre>

This first circuit is the simplest form of output in the kit. You can use the LED to teach both analog and digital output before moving on to more exciting outputs. There is an LED built into your Arduino board which corresponds to Digital Pin # 13.

## Introduction to Arduino

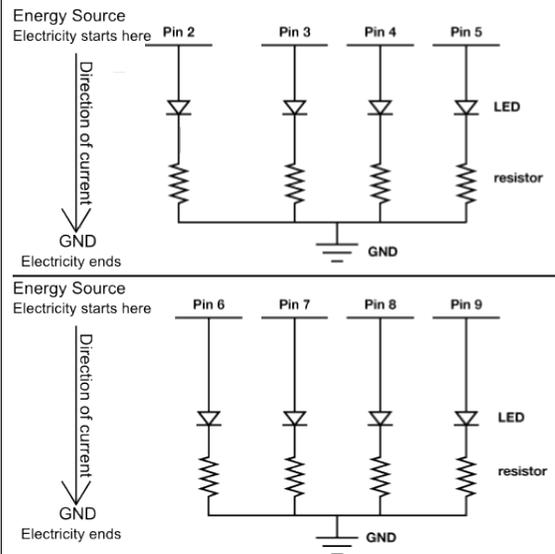
### SparkFun Electronics Summer Semester

## Circuit 2

### Explanation:

This circuit takes electricity from Pin # 2 through Pin # 9 on the Arduino. The LEDs are connected to the circuit so electricity enters through the anode (+, or longer wire) and exits through the cathode (-, or shorter wire). The resistor dissipates current so the LEDs do not draw current above the maximum rating and burn out. Finally the electricity reaches ground, closing the circuit and allowing electricity to flow from power source to ground.

### Schematic:



### Components:

Arduino Digital Pins # 2 - # 9: Power source, analog (if code uses analogWrite, only possible on pins 3, 5, 6, & 9) or digital (if code uses digitalWrite) output from Arduino board.

LEDs: As in other diodes, current flows easily from the + side, or anode (longer wire), to the - side, or cathode (shorter wire), but not in the reverse direction. Also lights up!

330 Ohm Resistor: The resistors resist the current flowing through the circuit. In this circuit the resistors reduce the current so the LEDs do not burn out.

Gnd: Ground

### Code:

```
//this line below declares an array
int ledPins[ ] = {2,3,4,5,6,7,8,9};
void setup( ) {
  //these two lines set Digital Pins # 0
  // - 8 to output
  for(int i = 0; i < 8; i++){
    pinMode(ledPins[i],OUTPUT);
  }
  void loop( ) {
    //these lines turn LEDs on and then off
    for(int i = 0; i <= 7; i++){
      digitalWrite(ledPins[i], HIGH);
      delay(delayTime);
      digitalWrite(ledPins[i], LOW);
    }
  }
}
```

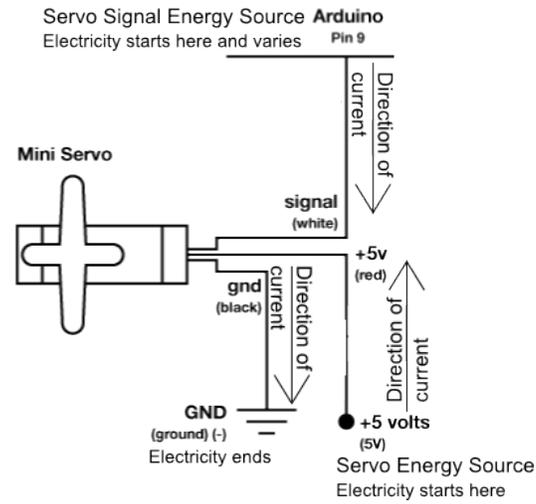
The code examples in the S.I.K get a little complicated for the second circuit, but don't worry, it's just more outputs. Some of the code examples use "for" loops to do something a number of times, if you're not familiar with "for" look it up because it is a key programming concept.



## Introduction to Arduino

### SparkFun Electronics Summer Semester

## Circuit 4

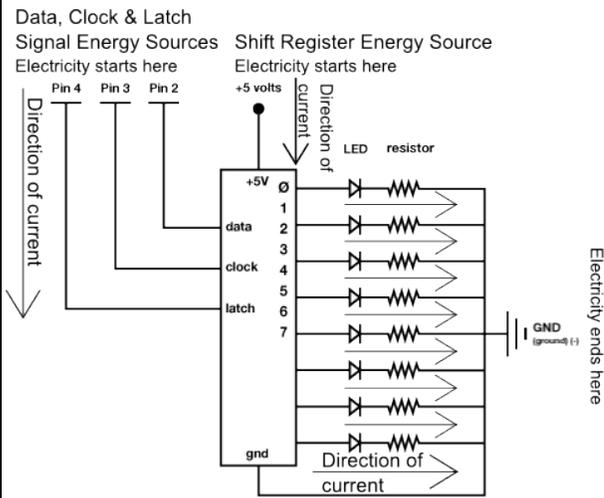
<p><b>Explanation:</b></p> <p>The servo in this circuit takes electricity from 5V on the Arduino. Pin # 9 on the Arduino supplies a PWM signal that sets the position of the servo. Each voltage value has a distinct correlating position. Finally the electricity reaches ground, closing the circuit and allowing electricity to flow from power source to ground.</p>	<p><b>Schematic:</b></p>  <p>Servo Signal Energy Source Arduino Electricity starts here and varies Pin 9</p> <p>Direction of current</p> <p>Mini Servo</p> <p>signal (white)</p> <p>+5v (red)</p> <p>Direction of current</p> <p>gnd (black)</p> <p>Direction of current</p> <p>GND (ground) (-)</p> <p>Electricity ends</p> <p>+5 volts (5V)</p> <p>Servo Energy Source Electricity starts here</p>
<p><b>Components:</b></p> <p>Arduino Digital Pin #9: Signal power source for servo.</p> <p>Servo: Sets the position of the servo arm depending on the voltage of the signal received.</p> <p>+5V: Five Volt power source.</p> <p>Gnd: Ground</p>	<p><b>Code:</b></p> <pre>//include the servo library for use #include &lt;Servo.h&gt; Servo myservo; //create servo object int pos = 0;  void setup() {   myservo.attach(9); }  void loop() {   //moves servo from 0° to 180°   for(pos = 0; pos &lt; 180; pos += 1) {     myservo.write(pos);     delay(15);   }   // moves servo from 180° to 0°   for(pos = 180; pos &gt;= 1; pos -= 1) {     myservo.write(pos);     delay(15);   } }</pre>

Remember, this is just slightly more complicated output, same as the motor and LED.

## Introduction to Arduino

### SparkFun Electronics Summer Semester

## Circuit 5

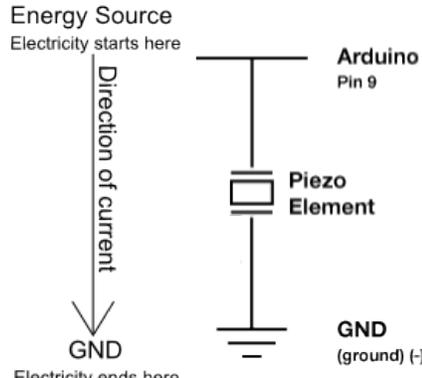
<p><b>Explanation:</b></p> <p>The shift register in this circuit takes electricity from 5V on the Arduino. Pin # 2, # 3 and # 4 on the Arduino supply a digital value. The latch and clock pins are used to allow data into the shift register. The shift register sets the eight output pins to either HIGH or LOW depending on the values sent to it via the data pin. The LEDs are connected to the circuit so electricity enters through the anode (+, or longer wire) and exits through the cathode (-, or shorter wire) if the shift register pin is HIGH. The resistor dissipates current so the LEDs do not draw current above the maximum rating and burn out. Finally the electricity reaches ground, closing the circuit and allowing electricity to flow from power source to ground.</p>	<p><b>Schematic:</b></p>  <p style="text-align: right; font-size: small;">Electricity ends here</p>
<p><b>Components:</b></p> <p>Arduino Digital Pin # 2, # 3 and # 4: Signal power source for data, clock and latch pins on shift register.</p> <p>Shift register: Allows usage of eight output pins with three input pins, a power and a ground. Link to datasheet.</p> <p>LED: As in other diodes, current flows easily from the + side, or anode (longer wire), to the - side, or cathode (shorter wire), but not in the reverse direction. Lights up!</p> <p>330 Ohm Resistor: A resistor resists the current flowing through the circuit. In this circuit the resistor reduces the current so the LED does not burn out.</p> <p>+5V: Five Volt power source.</p> <p>Gnd: Ground</p>	<p><b>Code:</b></p> <pre> int data = 2; int clock = 3; int latch = 4;  int ledState = 0; const int ON = HIGH; const int OFF = LOW;  void setup() {   pinMode(data, OUTPUT);   pinMode(clock, OUTPUT);   pinMode(latch, OUTPUT); }  void loop(){   for(int i = 0; i &lt; 256; i++) {     updateLEDs(i);     delay(25);   } }  void updateLEDs(int value) {   digitalWrite(latch, LOW);   shiftOut(data, clock, MSBFIRST, value);   digitalWrite(latch, HIGH); } </pre>

For more advanced components you will need to read Datasheets to figure out how to use them. Any documentation is good as long as you can get the correct information out of it.

## Introduction to Arduino

### SparkFun Electronics Summer Semester

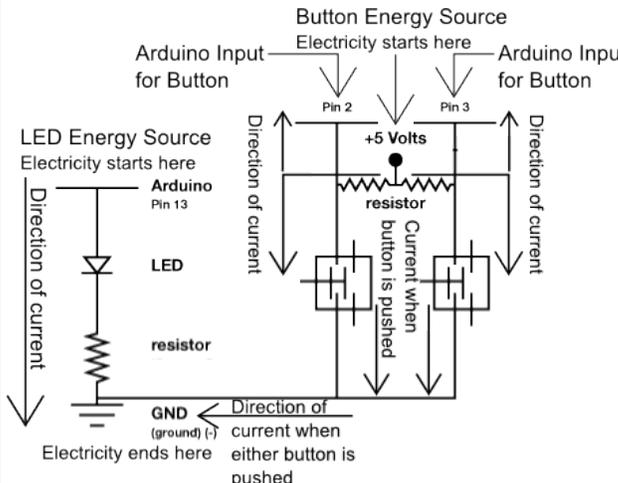
## Circuit 6

<p><b>Explanation:</b></p> <p>This circuit gets electricity from Arduino Pin # 9. The Piezo element plays different musical notes depending on the speed and duration of the electrical signal sent from Pin # 9. Finally the electricity reaches ground, closing the circuit and allowing electricity to flow from power source to ground.</p>	<p><b>Schematic:</b></p>  <p>The schematic shows a vertical circuit. At the top is 'Energy Source' with the note 'Electricity starts here'. A vertical line goes down to 'Arduino Pin 9'. From there, another vertical line goes down to a 'Piezo Element' represented by a rectangle with a diagonal line. Below the piezo element is another vertical line going down to 'GND (ground) (-)'. To the left of this circuit, a vertical arrow points downwards with the text 'Direction of current' and 'Electricity ends here' at the bottom.</p>
<p><b>Components:</b></p> <p>Arduino Digital Pin # 9: Power source, digital output from Arduino board. (If changed to PWM output this creates distortion of note, not a change in volume.)</p> <p>Piezo element: A tiny speaker with a magnetic coil that responds to electrical current by moving more or less depending on the current. The coil is attached to a diaphragm that moves air and causes the noise we hear.</p> <p>Gnd: Ground</p>	<p><b>Code:</b></p> <pre> /* this section contains only the two functions needed to make the piezo play a note of a given duration. These functions are called in the loop ( ) function. */ void playTone(int tone, int duration) {   for (long i = 0; i &lt; duration * 1000L; i += tone * 2) {     digitalWrite(speakerPin, HIGH);     delayMicroseconds(tone);     digitalWrite(speakerPin, LOW);     delayMicroseconds(tone);   } } void playNote(char note, int duration) {   char names[ ] = { 'c', 'd', 'e', 'f', 'g', 'a', 'b', 'C' };   int tones[ ] = { 1915, 1700, 1519, 1432, 1275, 1136, 1014, 956 };   for (int i = 0; i &lt; 8; i++) {     if (names[i] == note) {       playTone(tones[i], duration);     }   } } </pre>

This is another way to use digital pins to create analog output.

## Introduction to Arduino SparkFun Electronics Summer Semester

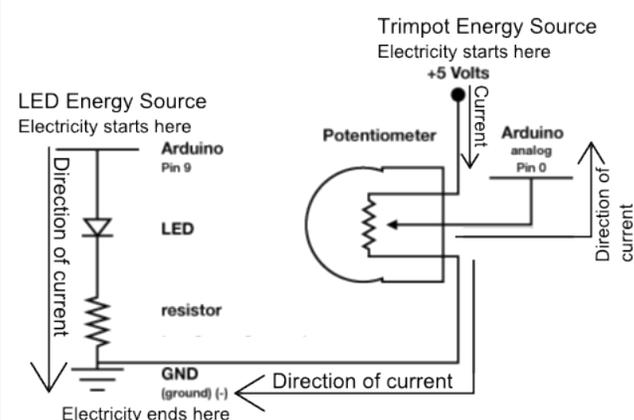
### Circuit 7

<p><b>Explanation:</b></p> <p>This circuit is actually two different circuits. One circuit for the buttons and another for the LED. See 'How the Circuits Work', Circuit 1 for an explanation of the LED circuit. The button circuit gets electricity from the 5V on the Arduino. The electricity passes through a pull up resistor, causing the input on Arduino Pins # 2 and # 3 to read HIGH when the buttons are not being pushed. When a button is pushed it allows the current to flow to ground, causing a LOW reading on the input pin connected to it. This LOW reading is then used in the code you load onto the Arduino and effects the power signal in the LED circuit.</p>	<p><b>Schematic:</b></p> 
<p><b>Components:</b></p> <p>Arduino Digital Pin # 13: Power source, PWM (if code uses analogWrite) or digital (if code uses digitalWrite) output from Arduino board.</p> <p>Arduino Digital Pin # 2 and # 3: Digital input to Arduino board.</p> <p>330 &amp; 10K Ohm Resistors: Resistors resist the current flowing through the circuit. In the LED circuit the 330 ohm resistor reduces the current so the LED in the circuit does not burn out. In the button circuits the 10K's ensure that the buttons will read HIGH when they are not pressed.</p> <p>LED: As in other diodes, current flows easily from the + side, or anode (longer wire), to the - side, or cathode (shorter wire), but not in the reverse direction. Lights up!</p> <p>Button: A press button which is open (or disconnected) when not in use and closed (or connected) when pressed. This allows you to complete a circuit when you press a button.</p> <p>+5V: Five Volt power source.</p> <p>Gnd: Ground</p>	<p><b>Code:</b></p> <pre>const int buttonPin = 2; const int ledPin = 13; int buttonState = 0;  void setup() {   pinMode(ledPin, OUTPUT);   //this line below declares the button   pinMode(buttonPin, INPUT); }  void loop(){   //this line assigns whatever the   Digital Pin 2 reads to buttonState   buttonState = digitalRead(buttonPin);   if (buttonState == HIGH) {     digitalWrite(ledPin, HIGH);   }   else {     digitalWrite(ledPin, LOW);   } }</pre>

In this circuit you are using a Digital Pin, but you are using it as input rather than output.

## Introduction to Arduino SparkFun Electronics Summer Semester

### Circuit 8

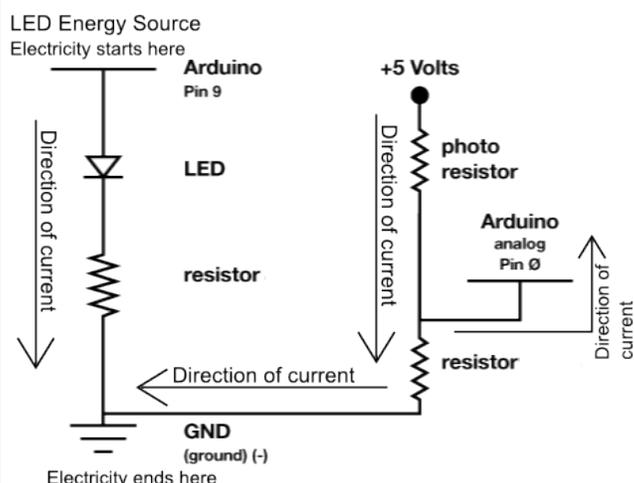
<p><b>Explanation:</b></p> <p>This circuit is actually two different circuits. One circuit for the potentiometer and another for the LED. See How the Circuits Work, Circuit 1 for an explanation of the LED circuit. The potentiometer circuit gets electricity from the 5V on the Arduino. The electricity passes through the potentiometer and sends a signal to Analog Pin # 0 on the Arduino. The value of this signal changes depending on the setting of the dial on the potentiometer. This analog reading is then used in the code you load onto the Arduino and effects the power signal in the LED circuit. Finally the electricity reaches ground, closing the circuit and allowing electricity to flow from power source to ground.</p>	<p><b>Schematic:</b></p> 
<p><b>Components:</b></p> <p>Arduino Digital Pin # 13: Power source, PWM (if code uses analogWrite) or digital (if code uses digitalWrite) output from Arduino board.</p> <p>Arduino Analog Pin # 0: Analog input to Arduino board.</p> <p>330 Ohm Resistor: A resistor resists the current flowing through the circuit. In the LED circuit it reduces the current so the LED in the circuit does not burn out.</p> <p>LED: As in other diodes, current flows easily from the + side, or anode (longer wire), to the - side, or cathode (shorter wire), but not in the reverse direction.</p> <p>Potentiometer: A voltage divider which outputs an analog value.</p> <p>+5V: Five Volt power source.</p> <p>Gnd: Ground</p>	<p><b>Code:</b></p> <pre>int sensorPin = 0; int ledPin = 13; int sensorValue = 0;  void setup() {   pinMode(ledPin, OUTPUT); }  void loop() {   //this line assigns whatever the   //Analog Pin 0 reads to sensorValue    sensorValue = analogRead(sensorPin);    digitalWrite(ledPin, HIGH);   delay(sensorValue);   digitalWrite(ledPin, LOW);   delay(sensorValue); }</pre>

This is another example of input, only this time it is Analog. Circuits 7 and 8 in the S.I.K. introduce you to the two kinds of input your board can receive: Digital and Analog.

## Introduction to Arduino

### SparkFun Electronics Summer Semester

## Circuit 9

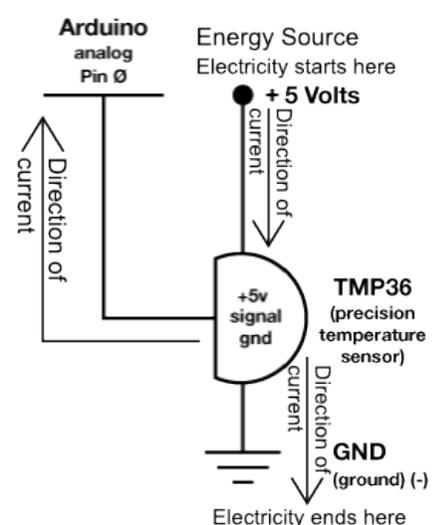
<p><b>Explanation:</b></p> <p>This circuit is actually two different circuits. One circuit for the photoresistor and another for the LED. See How the Circuits Work, Circuit 1 for an explanation of the LED circuit. The photoresistor circuit gets electricity from the 5V on the Arduino. The electricity passes through the photoresistor and sends a signal to Analog Pin # 0 on the Arduino. The value of this signal changes depending on the amount of sunlight. This analog reading is then used in the code you load onto the Arduino and affects the power signal in the LED circuit. The resistor below the Analog Pin connection creates the voltage divider necessary to measure the resistance of the photoresistor. Finally the electricity reaches ground, closing the circuit and allowing electricity to flow from power source to ground.</p>	<p><b>Schematic:</b></p> 
<p><b>Components:</b></p> <p>Arduino Digital Pin # 13: Power source, PWM (if code uses analogWrite) or digital (if code uses digitalWrite) output from Arduino board.</p> <p>Arduino Analog Pin # 0: Analog input to Arduino.</p> <p>330 Ohm Resistor: A resistor resists the current flowing through the circuit. In the LED circuit it reduces the current so the LED in the circuit does not burn out. In the photoresistor circuit the resistor completes the voltage divider.</p> <p>LED: As in other diodes, current flows easily from the + side, or anode (longer wire), to the - side, or cathode (shorter wire), but not in the reverse direction. Lights up!</p> <p>Photoresistor: A resistor with a resistance value that corresponds to the light hitting the sensor.</p> <p>+5V: Five Volt power source.</p> <p>Gnd: Ground</p>	<p><b>Code:</b></p> <pre>int lightPin = 0; int ledPin = 9;  void setup() {   pinMode(ledPin, OUTPUT); }  void loop() {   int lightLevel =   analogRead(lightPin);   lightLevel = map(lightLevel, 0, 900,   0, 255);   lightLevel = constrain(lightLevel, 0,   255);   analogWrite(ledPin, lightLevel); }</pre>

This circuit is another example of Analog input. It is also a perfect example of a voltage divider. Don't worry about the "map" and "constrain" functions, they are explained in the glossary.

## Introduction to Arduino

### SparkFun Electronics Summer Semester

## Circuit 10

<p><b>Explanation:</b></p> <p>This circuit takes electricity from the 5V on the Arduino. The temp sensor sends an analog value to Arduino Analog Pin # 0. Then the electricity reaches ground, closing the circuit and allowing electricity to flow from power source through the sensor to ground. Finally Arduino uses it's Serial monitor to display the temperature reading.</p>	<p><b>Schematic:</b></p> 
<p><b>Components:</b></p> <p>Arduino Analog Pin # 0: Analog input to Arduino board.</p> <p>Temperature Sensor: Provides a voltage value depending on the temperature. Some math is then required to convert this value to Celsius or Fahrenheit.</p> <p>+5V: Five Volt power source.</p> <p>Gnd: Ground</p>	<p><b>Code:</b></p> <pre>int temperaturePin = 0; void setup() {   //Serial comm. at a Baud Rate of 9600   Serial.begin(9600); } void loop() {   //Calls the function to read the sensor pin   float temp = getVoltage(temperaturePin);   //Below is a line that compensates for an   //offset (see datasheet)   temp = (temp - .5) * 100;   //This line displays the variable   temperature after all the math   Serial.println(temp);   delay(1000); } //function that reads the Arduino pin and //starts to convert it to degrees float getVoltage(int pin) {   return (analogRead(pin) * .004882814); }</pre>

There is a lot of math involved in the code section of this circuit and it all has a reason. But, how would you know you need to offset the temperature reading by .5 unless you had read the Datasheet? Also, pay attention to the code lines that enable Serial communication.

## Introduction to Arduino SparkFun Electronics Summer Semester

### Circuit 11

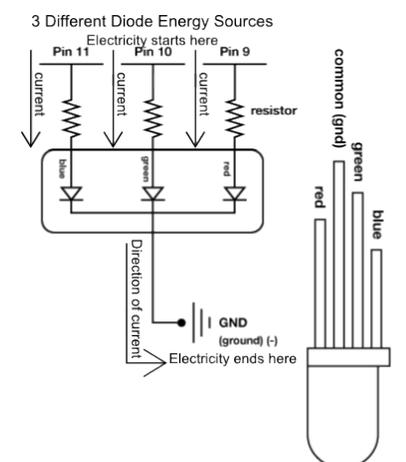
<p><b>Explanation:</b></p> <p>The relay circuit gets electricity from the 5V on the Arduino. The electricity always passes through the relay communication line which is switched to either NO (Normally Open) or NC (Normally Closed), lighting up one of the two LEDs. The transistor gets electricity from Arduino Digital Pin # 2 with a resistor to prevent burn out. In this case the transistor receives a digital signal. The transistor closes the circuit when it is sent a HIGH value, allowing electricity to flow through the relay coil, into the collector, out the emitter and to ground, completing the circuit. The energized coil sets the relay switch to NO. The transistor opens or breaks the circuit when it is sent a LOW value, so no electricity passes through the coil and the relay switch is set to NC. The flyback diode connected close to the motor is simply to protect the motor in the rare case that electricity flows from the transistor towards the motor. This only happens if the transistor is shut off suddenly.</p>	<p><b>Schematic:</b></p> <p><b>* Current will flow into the relay coil only if the transistor base is HIGH so the transistor emitter is providing a path to GND</b></p> <p><b>** Relay switches connection only if there is current running through the coil</b></p> <p><b>Code:</b></p> <pre>int ledPin = 2;  void setup() {   pinMode(ledPin, OUTPUT); }  void loop() {   //set the transistor on   digitalWrite(ledPin, HIGH);   // wait for a second   delay(1000);   // set the transistor off   digitalWrite(ledPin, LOW);   // wait for a second   delay(1000); }</pre>
<p><b>Components:</b></p> <p>Arduino Digital Pin # 2: Power source, digital output from Arduino board.</p> <p>Relay: The relay acts as an electrically operated switch between the two LED's.</p> <p>Transistor: A semiconductor which can be used as an amplifier or a switch. In this case the amount of electricity supplied to the base corresponds to the amount of electricity allowed through from the collector to the emitter.</p> <p>330 Ohm &amp; 10K Resistors: A resistor resists the current flowing through the circuit. In the transistor circuit it reduces the current so the transistor in the circuit does not burn out.</p> <p>Flyback Diode: As in other diodes, current flows easily from the + side, or anode, to the - side, or cathode, but not in the reverse direction. In this case the diode is being used to prevent current from 'flying back' to the relay in case the transistor is suddenly turned off.</p>	

Transistors can be used as switches or amplifiers and they are often called the most important invention of the 20<sup>th</sup> century. The relay is also a great control component, but it needs something to activate it, hence the transistor.

## Introduction to Arduino

### SparkFun Electronics Summer Semester

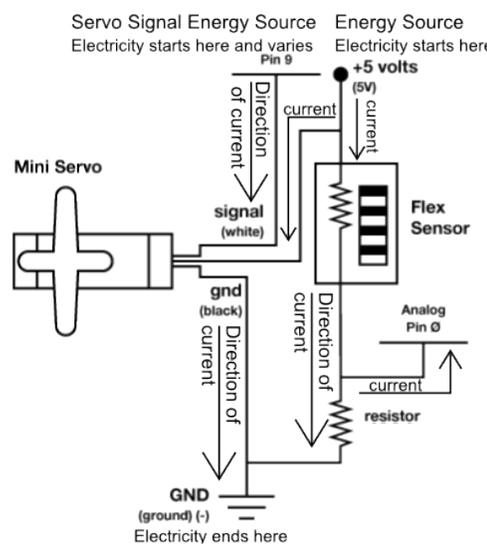
## Circuit 12

<p><b>Explanation:</b></p> <p>This circuit is pretty straight forward. The Digital Arduino Pins # 9, # 10 and # 11 supply a PWM value to each of the three different LEDs within the Tri-Color LED (Red, Green, and Blue). The LEDs are connected to the circuit so electricity enters through the anode (+, or longer wire) and exits through the cathode (-, or shorter wire). The resistors dissipate current so the LEDs do not draw current above the maximum rating and burn out. Finally the electricity reaches ground, closing the circuit and allowing electricity to flow from power source to ground. By supplying different values to just these three Digital Pins you can mix 16,777,216 different colors!</p>	<p><b>Schematic:</b></p> 
<p><b>Components:</b></p> <p>Arduino Digital Pin # 9, # 10 and # 11: Power source, PWM output from Arduino board.</p> <p>RGB LED: Unlike single color LED's, on RGB (Also called 'Tri-Color') LED's, the cathode (or ground wire) is the longest wire and each color (Red, Green, and Blue) gets its own lead. (See the schematic for details).</p> <p>330 Ohm Resistor: A resistor resists the current flowing through the circuit. In this circuit the resistor reduces the current so the LEDs do not burn out.</p> <p>Gnd: Ground</p>	<p><b>Code:</b></p> <pre>const int RED_LED_PIN = 9; const int GREEN_LED_PIN = 10; const int BLUE_LED_PIN = 11; int redIntensity = 0; int greenIntensity = 0; int blueIntensity = 0; const int DISPLAY_TIME = 100;  void setup() { }  void loop(){   for (greenIntensity = 0; greenIntensity &lt;= 255; greenIntensity+=5) {     redIntensity = 255-greenIntensity;     analogWrite(GREEN_LED_PIN, greenIntensity);     analogWrite(RED_LED_PIN, redIntensity);     delay(DISPLAY_TIME);   }   for (blueIntensity = 0; blueIntensity &lt;= 255; blueIntensity+=5) {     greenIntensity = 255-blueIntensity;     analogWrite(BLUE_LED_PIN, blueIntensity);     analogWrite(GREEN_LED_PIN, greenIntensity);     delay(DISPLAY_TIME);   }   for (redIntensity = 0; redIntensity &lt;= 255; redIntensity+=5) {     blueIntensity = 255-redIntensity;     analogWrite(RED_LED_PIN, redIntensity);     analogWrite(BLUE_LED_PIN, blueIntensity);     delay(DISPLAY_TIME);   } }</pre>

## Introduction to Arduino

### SparkFun Electronics Summer Semester

## Circuit 13

<p><b>Explanation:</b></p> <p>This circuit is actually two different circuits. One circuit for the flex sensor and another for the servo. See How the Circuits Work, Circuit 4 for an explanation of the servo circuit. The flex sensor circuit gets electricity from the 5V on the Arduino. The electricity passes through the flex sensor and sends a signal to Analog Pin # 0 on the Arduino. The value of this signal changes depending on the amount of bend in the flex sensor. This analog reading is then used in the code you load onto the Arduino and sets the position of the servo. The resistor and flex sensor create a voltage divider which is measured by Analog Pin # 0. Finally the electricity reaches ground, closing the circuit and allowing electricity to flow from power source to ground.</p>	<p><b>Schematic:</b></p> 
<p><b>Components:</b></p> <p>Arduino Digital Pin # 9: Power source, PWM output from Arduino board.</p> <p>Arduino Analog Pin # 0: Analog input to Arduino board.</p> <p>10K Ohm Resistor: A resistor resists the current flowing through the circuit.</p> <p>Flex Sensor: A resistor with a value that varies depending on the amount of bend in the sensor.</p> <p>Servo: Sets the position of the servo arm depending on the voltage of the signal received.</p> <p>+5V: Five Volt power source.</p> <p>Gnd: Ground</p>	<p><b>Code:</b></p> <pre>#include &lt;Servo.h&gt; Servo myservo;  int potpin = 0; int val;  void setup() {   Serial.begin(9600);   myservo.attach(9); }  void loop() {   val = analogRead(potpin);   Serial.println(val);   val = map(val, 50, 300, 0, 179);   myservo.write(val);   delay(15); }</pre>

This analog input is definitely very different from any other input we have looked at so far but the concept is the same. We treat the sensor as a resistor in a voltage divider to get a reading and then change our output depending on that reading.

## Introduction to Arduino

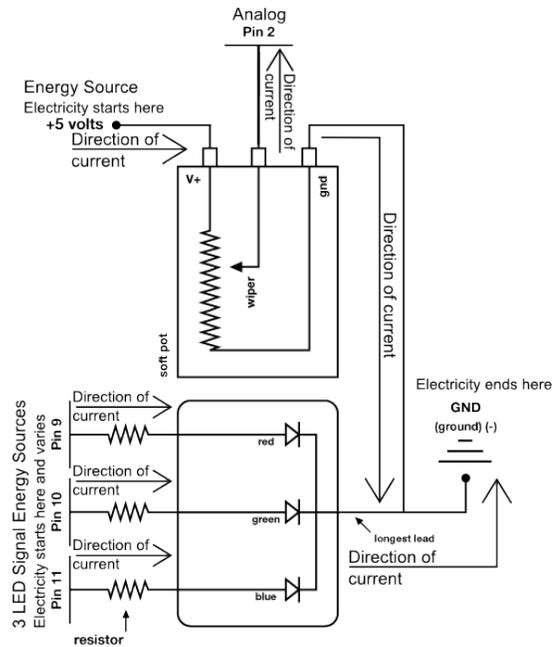
### SparkFun Electronics Summer Semester

## Circuit 14

### Explanation:

This circuit is actually two different circuits. One circuit for the soft pot and another for the RGB LED. See How the Circuits Work, Circuit 12 for an explanation of the RGB LED circuit. The soft pot circuit gets electricity from the 5V on the Arduino. The electricity passes through the soft pot and sends a signal out the com line of the soft pot to Analog Pin # 0 on the Arduino. The value of this signal changes depending on where the wiper (any type of contact) touches the soft pot. This analog reading is then used in the code you load onto the Arduino and sets the color of the RGB LED. Notice that yet again our sensor and the input pin form a voltage divider, only this time the voltage divider is completely inside the sensor. The wiper divides the resistor into two different portions with values that depend on the position of the wiper. Finally the electricity reaches ground, closing the circuit and allowing electricity to flow from power source to ground.

### Schematic:



### Components:

Arduino Digital Pins # 9, 10, 11: Power source, PWM output from Arduino board.

Arduino Analog Pin # 0: Analog input to Arduino board.

330 Ohm Resistor: A resistor resists the current flowing through the circuit. In the RGB LED circuit it reduces the current so the LED it is attached to does not burn out.

Flex Sensor: A resistor with a value that varies depending on the amount of bend in the sensor.

RGB LED: A grouping of three LEDs, Red, Green and Blue. Power goes in three different anodes (+, the short wires) and out one common cathode (-, the long wire). Lights up!

+5V: Five Volt power source.

Gnd: Ground

### Code:

```
const int RED_LED_PIN = 9;
const int GREEN_LED_PIN = 10;
const int BLUE_LED_PIN = 11;

void setup() {
  //No setup necessary but you still need it
}

void loop() {
  int sensorValue = analogRead(0);
  int redValue = constrain(map(sensorValue, 0, 512, 255, 0), 0, 255);
  int greenValue =
  constrain(map(sensorValue, 0, 512, 0, 255), 0, 255) - constrain(map(sensorValue, 512, 1023, 0, 255), 0, 255);
  int blueValue =
  constrain(map(sensorValue, 512, 1023, 0, 255), 0, 255);

  analogWrite(RED_LED_PIN, redValue);
  analogWrite(GREEN_LED_PIN, greenValue);
  analogWrite(BLUE_LED_PIN, blueValue);
}
```

## Introduction to Arduino

### SparkFun Electronics Summer Semester

### Powering Your Projects

When dealing with electronics, it is always a good idea to know how much power you need and how you're going to get it. If you want your project to be portable, or run separately from a computer, you'll need an alternate power source. Plus, not all Arduino projects can be run off 5V from the USB port. Fortunately there are a lot of options, one or more of which should suit your purposes perfectly.

### Understanding Battery Ratings

One popular way to get power to your project is through batteries. There are tons of different kinds of batteries (AA, AAA, C, D, Coin Cell, Lithium Polymer, etc.). In fact, there are too many to go over here, however, they all have a few things in common which can help you choose which ones to use. Each battery has a positive (+) and negative terminal (-) that you can think of as your power and ground. Batteries also have ratings in volts and milliamp hours (written mAh). Given this info along with how much current your circuit will draw, you can figure out how long a battery will last. For example, if I have a battery rated at 1.2v for 2500 mAh, and my circuit requires 100mA (milliamps) current, my battery will last around 17.5 hours. Wait, what? Why not 25 hours you say? Well, you shouldn't drain your battery completely, and other factors such as temperature and humidity can affect battery life, so typically the equation for determining battery life is:

$$\text{(Capacity rating of battery (in mAh)} \div \text{Current Consumption of Circuit)} \times 0.7$$

Note that we could still use our 2500 mAh battery in a 500mA circuit, but then our battery life would only be 3.5 hours. Make sense? There's a lot to understand about powering circuits, so don't worry if it's not all clicking. Just take an educated guess, be safe, use your multimeter, and make adjustments.

It is also worth mentioning that batteries are not the only potential source of power for your project. If your project will be outside or near a window, consider using solar power. There's plenty of good documentation online, but basically, solar cells have the same kinds of voltage and current ratings that any power source might have; the only difference is that the percentage you get from your solar panel depends on how much sunlight it's getting. Check out <http://www.solarbotics.com/> for some good products and documentation using solar power.

## Introduction to Arduino

### SparkFun Electronics Summer Semester

### Powering Your Projects

So, what if your circuit needs 12v, and all you have are a bunch of 1.5v batteries? Or what if you need your project to be powered for longer, but you don't want to give it too much power? This is where your knowledge of series and parallel may actually come in handy.

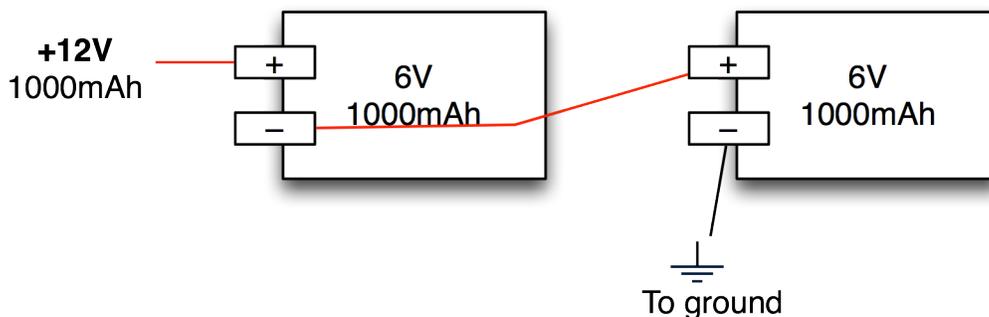
Here's the rule:

Connecting batteries **in series** increases the voltage but maintains the capacity (mAh) - this what you want to do if you need more power.

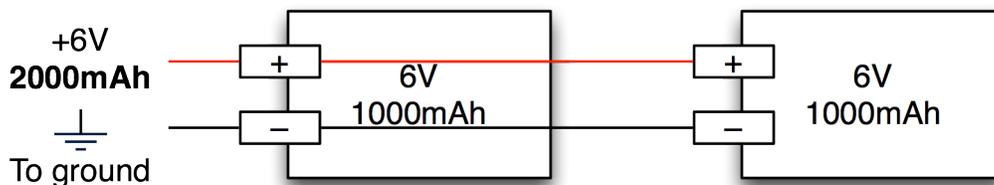
Connecting batteries **in parallel** maintains the voltage but increases the capacity. This is what you want to do if you need your power supply to last longer.

Here's how to hook them up:

#### Batteries in Series



#### Batteries in Parallel



As always, use caution. Batteries of the same kind (same voltage and capacitance) work best in these kinds of situations. Using different kinds of batteries may also work but it is not recommended, as the results are not as predictable.

## Introduction to Arduino

### SparkFun Electronics Summer Semester

# Transistors

## What is a transistor?

Transistors are semiconductors used to amplify or switch an electrical signal on and off.

## Why is a transistor useful?

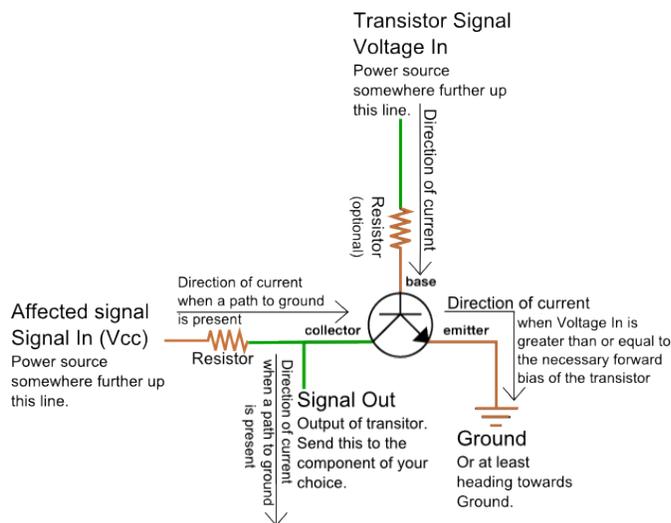
Often you will need more power to run a component than your Arduino can provide. A transistor allows you to control the higher power signal by breaking or closing a circuit to ground. Combining this higher power allows you to amplify the electrical signal in your circuit.

## What is in a transistor?

A transistor circuit has four parts; a signal power source (connects to transistor base), an affected power source (connects to transistor collector), voltage out (connects to transistor collector), and ground (connected to transistor emitter).

## How do you put together a transistor circuit?

It's really pretty easy. Here is a schematic and explanation detailing how:



The transistor voltage in signal is the signal that is used to control the transistor's base.

Signal in is the power source for the signal out which is controlled by the transistor's action.

Signal out is the output of the signal originating from signal in, it is controlled by the collector.

The amount of electrical current allowed through the transistor and out of the emitter to ground is what closes the entire circuit, allowing electrical current to flow through signal out.

## Ok, how is this transistor information used?

It depends on what you want to do with it really. There are two different purposes outlined above for the transistor, we will go over both.

If you wish to use the transistor as a switch the signal in and voltage in signal are connected to the same power source with a switch between them. When the switch is moved to the closed position an electrical signal is provided to the transistor base creating forward bias and allowing the electrical signal to travel from the signal in to the transistor's collector to the emitter and finally to ground. When the circuit is completed in this way the signal out is provided with an electrical current from signal in.

The signal amplifier use of the transistor works the same way only Signal In and Voltage In are not connected. This disconnection allows the user to send differing values to the base of the transistor. The closer the voltage in value is to the saturation voltage of the transistor the more electrical current that is allowed through the emitter to ground. By changing the amount of electrical current allowed through to ground you change the signal value of signal out.

## Introduction to Arduino

### SparkFun Electronics Summer Semester

# Voltage Dividers

## What is a voltage divider?

Voltage dividers are a way to produce a voltage that is a fraction of the original voltage.

## Why is a voltage divider useful?

A voltage divider is useful because you can take readings from a circuit that has a voltage beyond the limits of your input pins. By creating a voltage divider you can be sure that you are getting an accurate reading of voltage from a circuit. Voltage dividers are also used to provide an analog Reference signal.

## What is in a voltage divider?

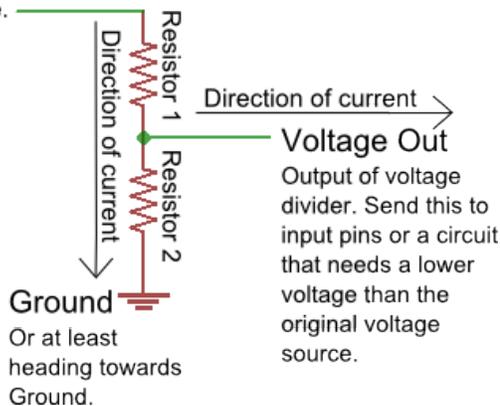
A voltage divider has three parts; two resistors and a way to read voltage between the resistors.

## How do you put together a voltage divider?

It's really pretty easy. Here is a schematic and explanation detailing how:

### Voltage In

Power source somewhere further up this line.



Often resistor # 1 is a resistor with a value that changes, possibly a sensor or a potentiometer.

Resistor # 2 has whatever value is needed to create the ratio the user decides is acceptable for the voltage divider output.

The Voltage In and Ground portions are just there to establish which way the electrical current is heading; there can be any number of circuits before and after the voltage divider.

Here is the equation that represents how a voltage divider works:

$$V_{out} = V_{in} \frac{R_2}{(R_1 + R_2)}$$

If both resistors have the same value then Voltage Out is equal to 1/2 Voltage In.

## Ok, how is this voltage divider information used?

It depends on what you want to do with it really. There are two different purposes outlined above for the voltage divider, we will go over both.

If you wish to use the voltage divider as a sensor-reading device first you need to know the maximum voltage allowed by the analog inputs you are using to read the signal. On an Arduino this is 5V. So, already we know the maximum value we need for Vout. The Vin is simply the amount of voltage already present on the circuit before it reaches the first resistor. You should be able to find the maximum voltage your sensor outputs by looking on the Datasheet; this is the maximum amount of voltage your sensor will let through given the voltage in of your circuit. Now we have exactly one variable left, the value of the second resistor. Solve for R2 and you will have all the components of your voltage divider figured out! We solve for R1's highest value because a smaller resistor will simply give us a smaller signal, which will be readable by our analog inputs.

Powering an analog Reference is exactly the same as reading a sensor except you have to calculate for the Voltage Out value you want to use as the analog Reference.

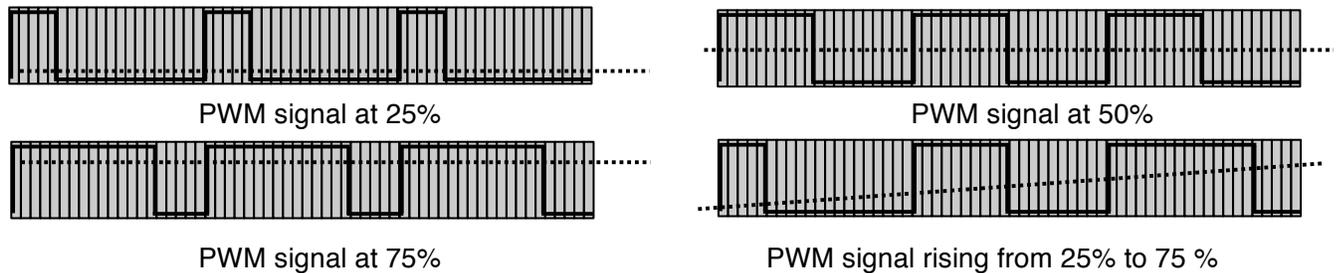
Given three of these values you can always solve for the missing value using a little algebra, making it pretty easy to put together your own voltage divider.

## Pulse Width Modulation

Computers and microprocessors only understand two things, ON and OFF. These are represented in a few different ways. There is ON and OFF, One and Zero, or HIGH and LOW. Ones and Zeros are used in the computer language Binary, HIGH and LOW are used with electricity, ON and OFF are plain old human speak.

But what if we want to turn something digital less than 100% ON? Then we use something called PWM, or Pulse Width Modulation. The way your Arduino microprocessor does this is by turning the electricity on a PWM pin ON and then OFF very quickly. The longer the electricity is ON the closer the PWM value is to 100%. This is very useful for controlling a bunch of stuff. For example: the brightness of a light bulb, volume of sound, or the speed of a motor.

For this reason some of the pins on your Arduino are labeled PWM or Pulse Width Modulation pins. This means you can send a bunch of ones and zeros real quick and the Arduino board will read these ones and zeros as an average somewhere between one and zero. The dotted line in the diagrams represents the average. See tables below.



Luckily a lot of the work has been done for you so you don't have to figure out the actual patterns of ones and zeros. All you have to do is pick a number between 0 and 255 and type the command `analogWrite`. The number zero means the pin is set fully off, the number 255 means the pin is set fully on, and all other numbers set the pin to values between ON (100% or 255) and OFF (0% or 0). You can use PWM on any pin labeled PWM and do not need to set the pin mode before sending an `analogWrite` command.

A microprocessor creates a PWM signal by using a built in clock. The microprocessor measures a certain amount of time (also called a window or a period) and turns the PWM pin ON (or HIGH) for the first part of this window and then OFF (or LOW) near the end of the window. The window is filled up with a different length ON (or HIGH) signal depending on the PWM value. If the PWM value is 50% then the PWM signal is ON (or HIGH) for half of the window. If the PWM value is 25% then the PWM signal is ON (or HIGH) for a quarter of the window. The only time the window will not have a LOW value is if the PWM signal is turned completely ON the whole time and therefore equal to 100% ON. The opposite is true as well, if the PWM signal is set to 0% or OFF, then there will not be any HIGH value at the beginning of the window.

To write a PWM value to one of the PWM enabled pins (3, 5, 6, 9, 10, 11) simply use the following code:

```
analogWrite(pin, value);
```

Where `pin` is one of the PWM enabled pin numbers and `value` is a value between 0 and 255.

## Introduction to Arduino

### SparkFun Electronics Summer Semester

# Basic Operators

Often when you are programming you will need to do simple (and sometimes not so simple) mathematical operations. The signs used to do this vary from very simple to confusing if you've never seen them before. Below is a table of definitions as well as some examples:

Arithmetic operators	Relational operators	Logical operators
+ (addition) - (subtraction) * (multiplication) / (division) % (modulus) = (assignment)	== (equality) != (inequality) > (greater-than) < (less-than) >= (greater-than-or-equal-to) <= (less-than-or-equal-to)	! (NOT) && (AND)    (OR)
Arithmetic operators are your standard mathematical signs (no example provided)	Relational operators are used to compare values and variables	Logical operators are used to join two or more conditional statements together

Pay attention to = and ==. = is used to assign variable values, == to compare values.

Relational operator example:	Logical operator example:
<pre>if (x!=7){ //loop body code here }</pre> <p>Compares x to the number 7, executes code inside body loop if the value of x <b>does not equal</b> 7</p>	<pre>if ((x==7)   (x==9)){ //loop body code here }</pre> <p>Compares x to the number 7 and 9, executes code inside body loop if the value of x equals 7 <b>or</b> 9</p>

### Comments

As you use code other people have written you will notice //, /\* and \*/ symbols. These are used to “comment” lines out so they do not affect the code. This way programmers can add comments to help you understand what the code does. Good code has comments that explain what each block of code (functions, classes, etc....) does but does not explain simpler portions of the code as this would be a waste of time. Commenting lines out is also a useful tool when you are writing code yourself. If you have a section of code you are working on, but isn't quite finished or doesn't work, you can comment it out so it does not affect the rest of your code when you compile or upload it.

//	/*	*/
This is used comment out a single line	This is used to start a section of commented lines	This is used to end or close a section of commented lines
//commented out line	/*comments start here	comments end here*/

## Introduction to Arduino

### SparkFun Electronics Summer Semester

# Serial Communication

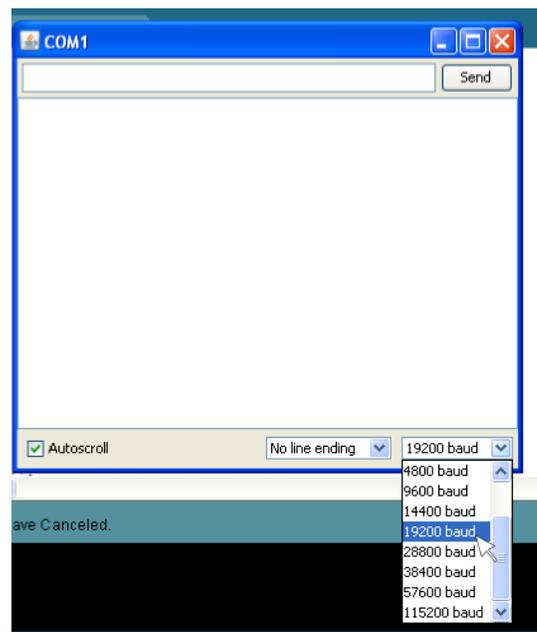
Serial is used to communicate between your computer and the Arduino as well as between Arduino boards and other devices. Serial uses a serial port (makes sense huh?) also known as UART, which stands for universal asynchronous receiver/transmitter to transmit and receive information. In this case the computer outputs Serial Communication via USB while the Arduino receives and transmits Serial using, you guessed it, the RX and TX pins. You use serial communication every time you upload code to your Arduino board. You will also use it to debug code and troubleshoot circuits. Basic serial communication is outlined in the following pages along with a simple activity to help you understand the concepts.

**Serial Monitor:** This is where you monitor your serial communication and set baud rate.

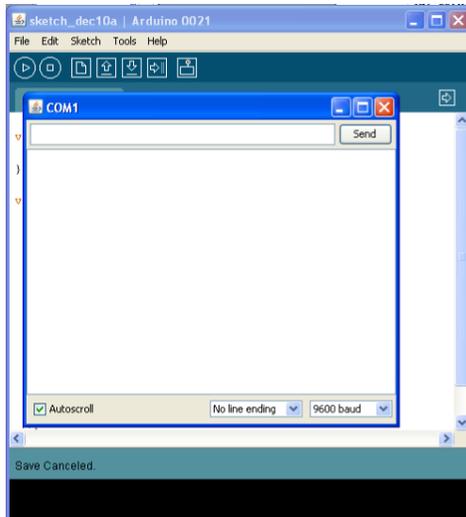
Activating the Serial Monitor:



Setting the Serial Monitor baud rate:



What the activated Serial Monitor looks like:



There are many different baud rates, (9600 is the standard for Arduino) the higher the baud rate the faster the machines are communicating.

In the examples above there is no Serial communication taking place yet. When you are running code that uses Serial any messages or information you tell Serial to display will show up in the window that opens when you activate the monitor.

### Things to remember about Serial from this page:

1. Serial is used to communicate, debug and troubleshoot.
2. Serial baud rate is the rate at which the machines communicate.

## Serial Communication

### Serial setup:

The first thing you need to know to use Serial with your Arduino code is Serial setup. To setup Serial you simply type the following line inside your `setup( )` function:

```
Serial.begin (9600);
```

This line establishes that you are using the Digital Pins # 0 and # 1 on the Arduino for Serial communication. This means that you will not be able to use these pins as Input or Output because you are dedicating them to Serial communication. The number 9600 is the baud rate, this is the rate at which the computer and the Arduino communicate. You can change the baud rate depending on your needs but you need to make sure that the baud rate in your Serial setup and the baud rate on your Serial Monitor are the same. If your baud rates do not match up the Serial Monitor will display what appears to be gibberish, but is actually the correct communication incorrectly translated.

### Using Serial for code debugging and circuit troubleshooting:

Once Serial is configured using the basic communication for debugging and troubleshooting is pretty easy. Anywhere in your sketch you wish the Arduino board to send a message type the line `Serial.println("communication here");`. This command will print whatever you type inside the quotation marks to the Serial Monitor followed by a return so that the next communication will print to the next line. If you wish to print something without the return use `Serial.print("communication here");`. To display the value of a variable using `println` simply remove the quotation marks and type the variable name inside the parenthesis. For example, type `Serial.println( i );` to display the value of the variable named `i`. This is useful in many different ways, if, for example, you wish to print some text followed by a variable or you want to display multiple variables before starting a new line in the Serial Monitor.

These lines are useful if you are trying to figure out what exactly your Arduino code is doing. Place a `println` command anywhere in the code, if the text in the `println` command shows up in your Serial Monitor you will know exactly when the Arduino reached that portion of code, if the text does not show up in the Serial Monitor you know that portion of code never executed and you need to rewrite.

To use Serial to troubleshoot a circuit use the `println` command just after reading an input or changing an output. This way you can print the value of a pin signal. For example, type `Serial.print("Analog pin 0 reads:");` and `Serial.println(analogRead(A0));` to display the signal on Analog Input Pin # 0. Replace the second portion with `Serial.println(digitalRead(10));` to display the signal on Digital Pin # 10.

### Things to remember about Serial from this page:

1. If Serial is displaying gibberish check the baud rates.
2. Use `Serial.print("communication here");` to display text.
3. Use `Serial.println("communication here");` to display text and start a new line.
4. Use `Serial.print(variableName);` to display the value stored in `variableName`.
5. Use `Serial.print(digitalRead(10));` to display the state of Digital Pin # 10.

## Serial Communication

### Using Serial for communication:

This is definitely beyond the scope of the S.I.K. but here are some basics for using Serial for device to device communication (other than your computer), not just debugging or troubleshooting. (The following paragraphs assume that you have Serial Communication hardware properly connected and powered on two different devices.)

First set up Serial as outlined on the previous page.

Use `Serial.println("Outgoing communication here");` to send information out on the transmit line.

When receiving communication the Serial commands get a little more complicated. First you need to tell the Arduino to listen for incoming communication. To do this you use the command `Serial.available()`; this command tells the computer how many bytes have been sent to the receive pin and are available for reading. The Serial receive buffer (computer speak for a temporary information storage space) can hold up to 128 bytes of information.

Once the Arduino knows that there is information available in the Serial receive buffer you can assign that information to a variable and then use the value of that variable to execute code. For example to assign the information in the Serial receive buffer to the variable `incomingByte` type the line; `incomingByte = Serial.read();` `Serial.read()` will only read the first available byte in the Serial receive buffer, so either use one byte communications or study up on parsing and string variable types. Below is an example of code that might be used to receive Serial communication at a baud rate of 9600.

```
int incomingByte = 0; //declare variable incomingByte, assign it value 0
void setup ( ) {
    Serial.begin(9600); //begin serial communication at baud rate 9600
}
void loop ( ) {
    //if there is information in the Serial receive buffer
    if (Serial.available() > 0){
        //assign the first byte in buffer to incomingByte
        incomingByte = Serial.read();
    }
    if (incomingByte == 'A'){           //if incomingByte is A
        //execute code inside these brackets if incomingByte is A
    }
    if (incomingByte == 'B'){           //if incomingByte is B
        //execute code inside these brackets if incomingByte is B
    }
}
```

### Additional things to note about Serial:

You cannot transmit and receive at the same time using Serial, you must do one or the other. You cannot hook more than two devices up to the same Serial line. In order to communicate between more than two devices you will need to use an Arduino library such as NewSoftSerial.