Let's Code Blacksburg's Arduino Cookbook
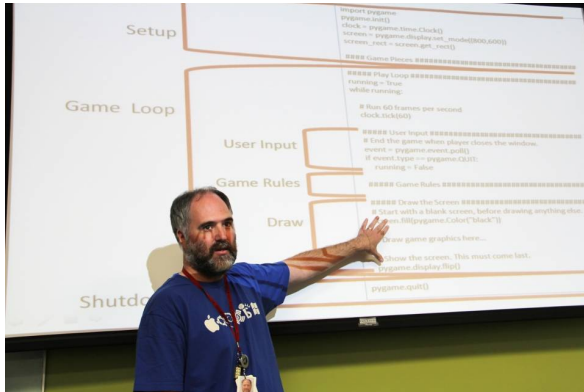
Version 2016-07-11
By Monta Elkins, Eddie Sheffield and Thomas Weeks
Let's Code Blacksburg 2016

How To Cook (use this Arduino cookbook)

<u>What Is This Cookbook?</u>

This Arduino circuits and programming instruction guide is organized into a "cookbook" style layout. The cookbook illustrates how to create and write various arduino based circuits and programs. These instructions are organized into "Recipes" or instruction guides that can be combined in different ways to come up with new creations. All the code from this cookbook is in our code repository at https://github.com/LetsCodeBlacksburg/arduino-recipes/

<u>How This All Works:</u>

For example, want to make a robot? Learn how to move a servo motor with our *Servo recipe*. Then learn how to read a an *Ultrasonic Eyes recipe*. Combine them to make a collision avoiding robot! Or use the *Potentiometer (knob) recipe* plus the *Sound/speaker recipe* and you've got a musical instrument! The only limit is your own imagination!

The circuit and programming recipes, just like this intro, each feature a **What**, **How** and **Fail** section – quickly telling you what you're doing, illustrating how to do it, and what to look for if something fails. Identifying failure is important as it is what enables you to "fail fast", learn and move on quickly to success. <u>Failure is good! Without it, learning is difficult and much less satisfying.</u>

<u>Failure:</u>

A bit more on failure – one needs to understand and embrace that ***Failure is a natural part of learning***, but before you can learn and move on from failure, you must:

> 1) ***Recognize*** failure has or is occurring
> 2) ***Step Back*** and determine (or guess at) the nature or root of the problem
> 3) ***Test Your Assumptions*** (of the problem) and correct assumptions when needed
> 3) ***Identify & Address the Problem***, work around it or create a new way of accomplishing the desired outcome.

Also understand that the failure sections are *not* definitive documents on every possible failure you can encounter for a given recipe. <u>No "failure guide" can easily contain every possible failure for a given circuit or technical process.</u> The failure section is more a guide to help <u>get your thinking cap back on straight</u> and think about the nature of the problem, and what quick fixes or work-arounds might get you back on track!

<u>Don't be afraid of failure!</u> Identify it, embrace it, learn from it and move on.
As Thomas Edison says,

> *"Negative results are just what I want. They're just as valuable to me as positive results. I can never find the thing that does the job best until I find the ones that don't."*
> — *Thomas Edison*

As one of our class instructors cries out to his students,

> *"Fail fast, fail cheap!"*
> — *Monta Elkins*

Get Cooking! With Let's Code Blacksburg Cookbook Recipes:

Need Some Ideas To Get Cooking? Check Out These Recipes !!!

This whole cookbook is designed like legos. You put together the various recipes to create thousands of cool, unusual inventions! For example, you can mix and match the recipes like this :

- Use the *Light Sensor recipe* to detect afternoon sunlight and close the blinds with a servo!
- The *People Motion Sensor recipe* you can detect movement, sound and alarm and then use the *Ethernet shield recipe* to send a TXT message or email!
- With just the *Light Sensor recipe* and a speaker you can make eerie sci-fi music!
- Combine multiple servos and potetiometers recipies to control a two axis robotic, popsicle stick arm (Advanced project. Ask for a special recipe handout if interested.)
- Use the *Ethernet shield recipe* to share real time house measurements to the Web or send via an SMS text message or email!

The only limits are your own imagination, available memory, and I/O pins! :)

* - Recommended or required for beginners
** - Really fun *and* easy for new beginners

Recipe Name	Description
*Installing Arduino Software & Drivers Recipe	Required on new PCs to talk to the Arduino.
*TEST: LED (light) Blink Recipe	Test compiles and uploads code to blink a built in LED.
OUTPUT: Serial Monitor Recipe	Outputs text back to the PC. Great for live troubleshooting.
*BUILD: Breadboard or Protoboard Recipe	Breadboards are your pallet for creating temporary circuits.
*INPUT: Potentiometer Recipe	A knob or "pot", a adjustable resistor for creating a variable voltage.
**BUILD: LED Chase Light Recipe	Wire up and blink a chase light circuit.
CONTROL: Servo Recipe	A servo is a PWM, digital motor that you can control the angular position of.
CONTROL: Robot DC Motor Control	Using the KeyesL298 H-Bridge motor controller w/DC motors.
INPUT: Ultrasonic "Eyes" Range Sensor	Ultrasonic sensor to measure distance to objects up to 24" out.
INPUT: Line Following IR Sensor	Using a three-element infrared line sensor.
**OUTPUT: Sound Generation with Arduino	Making tones, sounds and music with speaker.
**INPUT: Light Sensor	Detecting light intensity with a CdS photo-sensor.
**INPUT: People Motion Sensor	PIR motion sensor (senses motion up to 20ft away).
INPUT / OUTPUT: The LCD Display / Keypad shield	Displaying text and taking keypress inputs.
COMM: Ethernet Web & Email Communication	Using Wired Ethernet Networking for Web & Email.

The Authors:

Eddie Sheffield, Thomas "Tweeks" Weeks, and Monta Elkins*

* - the guy who came up with the idea of the arduino cookbook concept... convinced us all it would work.. and then got his friends to do all the heavy lifting.   Thanks Monta! ;)

Installing Arduino Software & Drivers Recipe   needed on a fresh PC to talk to the Arduino

What:

This is the process that installs two things; the IDE or programming environment that you use to program and upload code to the Arduino, and the USB/serial port drivers (if needed on Mac and Windows) to "talk" to the board over the USB interface.

> **NOTE:** If you can not get this working (test it using the Blink Recipe), then you will not be able to work with any other recipes in this cookbook.

How:

For Linux
    •RedHat:

```
# yum -y install arduino  #(reqs: uisp avr-libc avr-gcc-c++ rxtx avrdude)
```

    •Ubuntu:

```
$ sudo apt-get install arduino     #(reqs uisp avr-libc gcc-avr avrdude librxtx-java)
```

    •or for other installs or source based installs, go here:
        http://playground.arduino.cc/Learning/Linux

For Mac/OSX:
    •Download & Install Software from: http://arduino.cc/en/Guide/MacOSX#.UwGmXXWqYY0

For Windows:
    •Download & install the software from: http://arduino.cc/en/Guide/Windows#.UwGlyHWqYY0

Fail:

Linux T-Shooting:
    •check permissions of /dev/ttyUSB0 or /dev/ttyACM0 (user needs r/w or 777 access)
    •May have to open port permissions to:

```
# usermod -a -G uucp,dialout,lock $USER
```

   or may have to tempfix as root :

```
# chmod 777 /dev/ttyUSB0
```

Windows T-Shooting:
    •Make sure the special USB serial port drivers are installed
        http://arduino.cc/en/Guide/UnoDriversWindowsXP#.UwGbtHWqYY0
    •Check/fix COM port settings

Mac T-Shooting:
    •Make sure the special USB serial port drivers are installed
        http://arduino.cc/en/Guide/MacOSX#toc3
    •check device permissions (similar to Linux)

**TEST: LED (light) Blink Recipe**   test compiles and uploads code to blink a built in LED

What:

This process simply compiles and uploads code to the Arduino for execution and blinks a light when it succeeds. It's the easiest and fastest (fail fast) test method to verify you can talk to your arduino.

There is a small LED (the light) connected to pin 13 of the Arduino.   When that pin is 'high' (meaning +5 volts for this Arduino clone), the LED lights.

> **NOTE:** Arduino Software and driver should be installed. (see Installing Arduino Software and Drivers recipe).
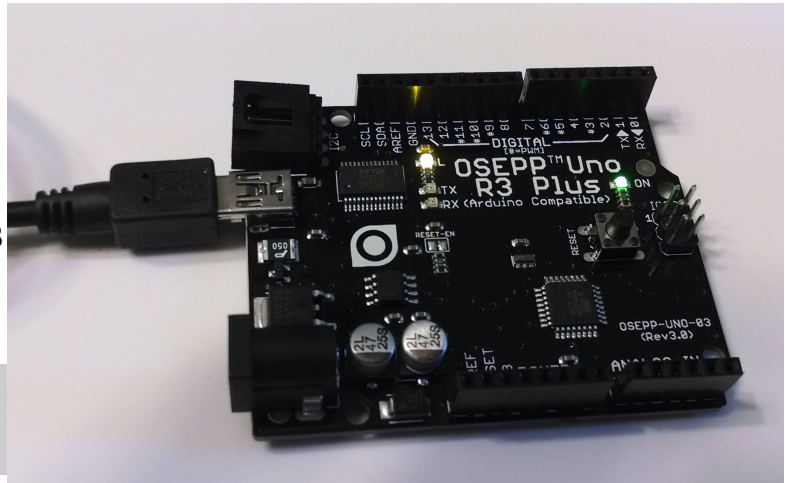
Arduino Clone with power and "blink" light

How:

Select appropriate port
"Tools / Serial port" (see Fail section if the Serial port menu is ghosted).

Select Arduino Uno.
"Tools / Board / Arduino Uno".

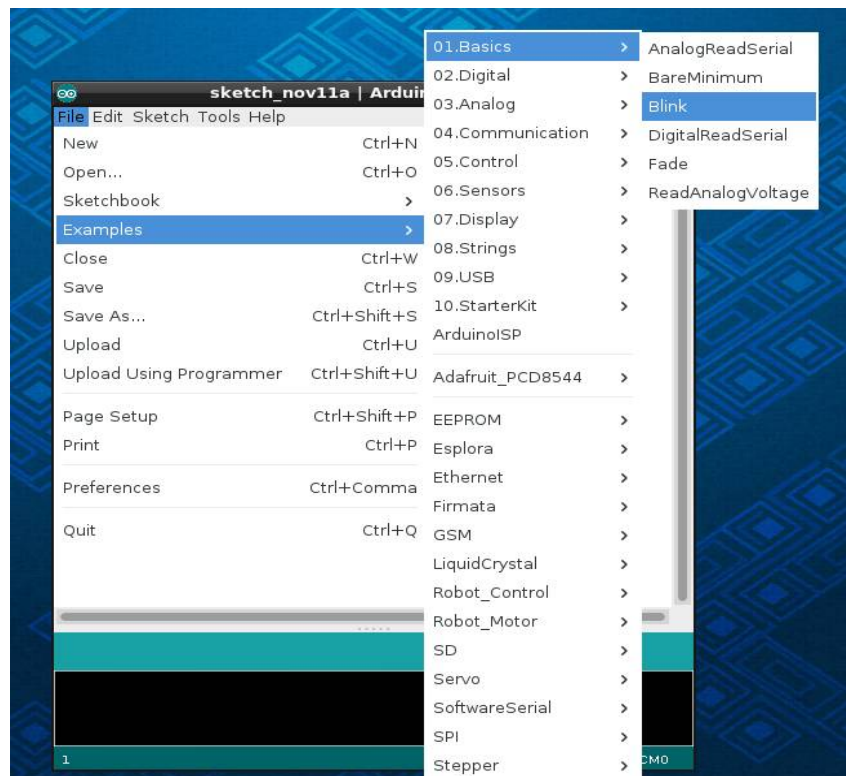Load blink onto the arduino, "File / Examples / Basics / Blink" (see right).

Click on the upload icon to compile and upload your program.

Several lights will blink during the upload, then your program runs.

Look for the steady light blinking here.

Change both lines that say **delay(1000);** to **delay(100);** and reupload the program.

Loading the "blink" program in the Arduino IDE

Look for the light to blink faster. This shows that the program changes you made are actually uploaded to the Arduino.

This blinking is done by the command **digitalWrite(led, HIGH);** which sends a "HIGH" 5volts to the pin# in the variable "led". When pin 13 is 'LOW' (meaning connected to ground) the LED (light) is off. This is done by the command **digitalWrite(led, LOW);**

NOTE: Anything following "//" on a line is considered comments and ignored.

Fail:

If you can not write to the arduino, get some error, or the serial port is ghosted:
Verify the correct serial port
Unplug Arduino and list serial ports,
then plug up Arduino and list serial ports again. An additional serial port should appear. That new serial port should be the Arduino port.
Try plugging Arduino into a different port USB port
See the fail section of the "Installing Arduino Software & Drivers" recipe (port/permissions/drivers)
Test with a different cable and Arduino board.
Try running the arduino program as root (gets around all permission errors. For testing only)

OUTPUT: Serial Monitor Recipe   outputs text back to the PC. Great for live troubleshooting

What:

The serial output on the arduino can be used to echo or print real time program data back to the PC over the serial USB port. This is very handy for troubleshooting or looking at run time values, states and problem code.

How:

Upload and execute the following sketch on the Arduino.

```
void setup() {
  // initialize serial communications at 9600 bps:
  Serial.begin(9600);
}

void loop() {

  Serial.println("Hello World");

  // wait 1 second before the next loop
  delay(1000);
}
```

Then open the Serial Monitor by selection "Tools / Serial Monitor" or clicking its icon on the far right . After which you should see the line "Hello World" appear repeatedly in the serial monitor window.
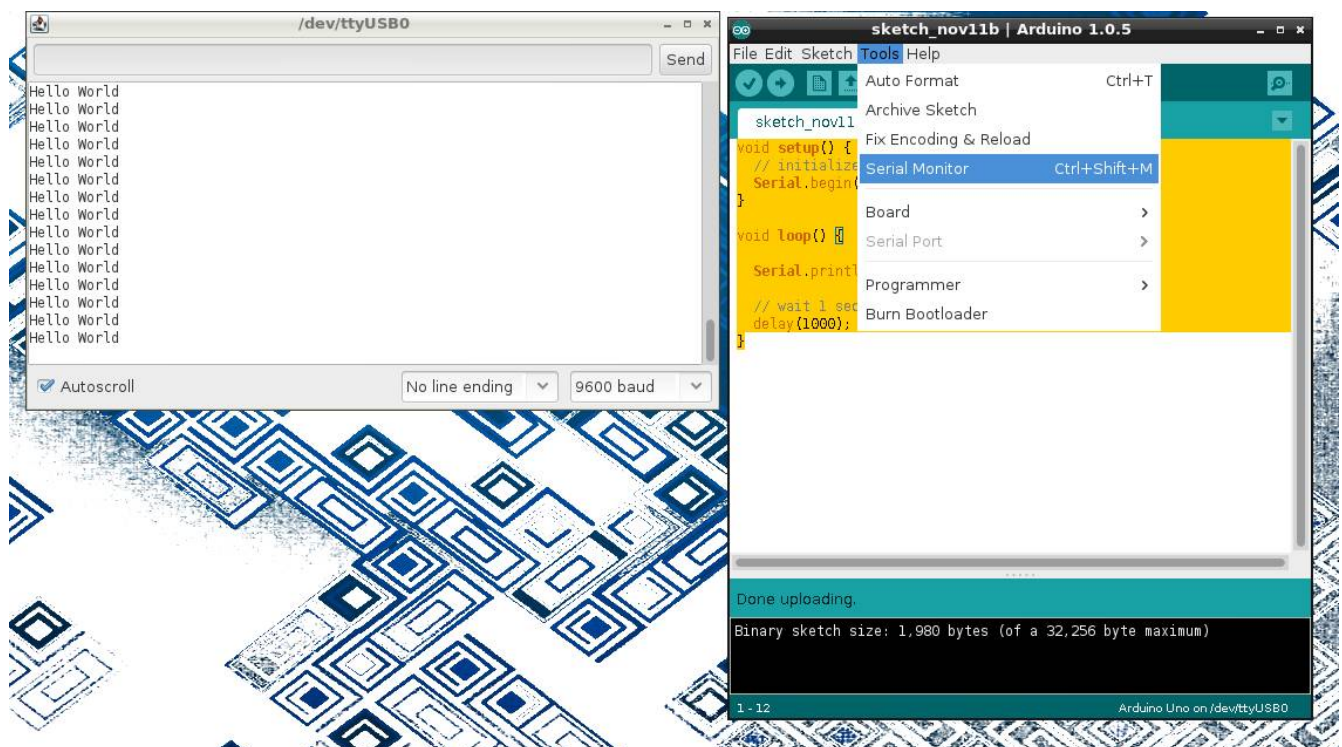
Serial.begin(9600); opens the serial port at on the Arduino and sets its speed to 9600 baud

The **Serial.println()** statement prints a line followed by a newline. When the Serial Monitor is opened it watched for characters appearing on the laptop's serial (or USB serial) port and prints them in the serial monitor window.

Fail:

1. Be sure that you have the **Serial.begin(9600);** defined in the **void setup () {** code block.
- Look for RX and TX lights to blink on Arduino rapidly during upload.
- If program is running successfully look for the TX light to blink once per second, showing that it is "trying" to transmitting data back to the laptop.
- Double check your Serial Monitor settings for both port and speed.

> **NOTE:** you have to restart the Serial Monitor after each upload of a new program, because the upload process uses the same serial port connection.

Breadboard or Protoboard Recipe breadboards are your pallet for creating temporary circuits
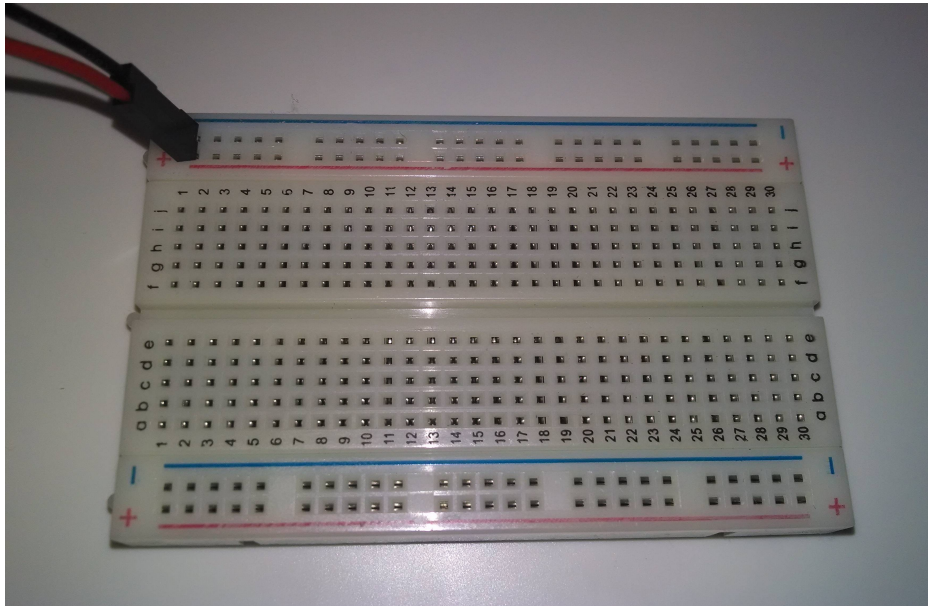
What:

How does a breadboard work?
Wire pins, pushed into the breadboard are connected together as shown in the schematic below. This allows the quick building and testing of circuits.

> **NOTE:** Unplug the Arduino from the laptop (and any other power supply while making and verifying connections).

Connection wire colors do not matter; but traditionally power (+) wires are red and ground (-) wires are blue (or black). The (+) red breadboard row and (-) blue breadboard row gives you a common hookup location on the breadboard for power (red) and ground (blue, GND). Using them also makes troubleshooting a little easier.

Average "breadboard" or "protoboard" (w/ horizontal +/- power rails)

Under the white plastic of the breadboard you see that the holes are connected. This is what makes working with a breadboard like legos for electronics.

WARNING: Never connect or short +5v power or (+) to GND or (-). This short will probably blow your laptop's USB port and damage other hardware. Not to mention upsetting your instructor and being mocked by your classmates – probably getting you a lame nick name like "smokey" or "shorty".
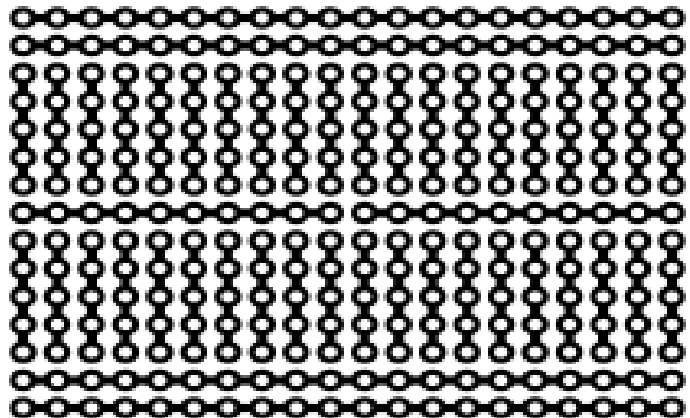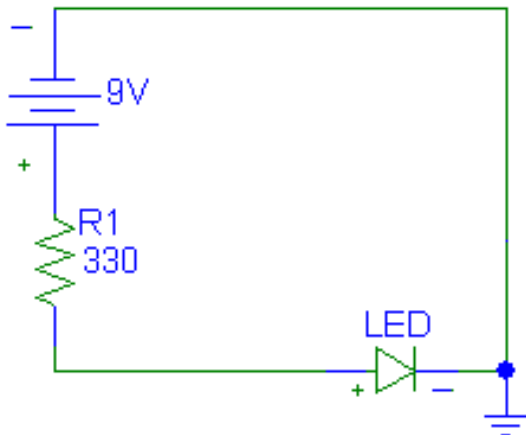

Protoboard schematic. -Wikipedia

How:

To build a circuit, you normally:
  1) start with a schematic diagram (left) which illustrates what is connected to what,
  2) orient your components correctly (note the polarity (+ and -) on the LED light)
  3) and connect them together on the breadboard:
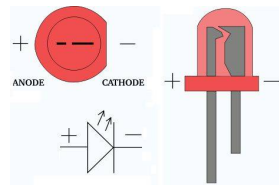
Schematic diagram   *LED polarity*   *Blank breadboard*
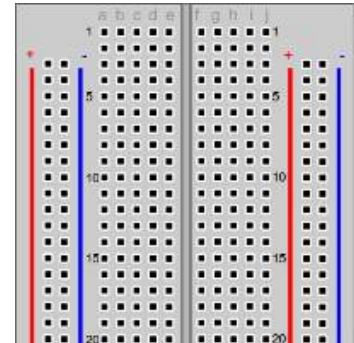
This is what the assembled circuit might look like:

-Batt to ground/-rial   to -LED,   +LED to Resistor   to +rail   to +Batt

Fail:

NEVER connect + to – directly. This will damage the breadboard, the wires, the battery and make your system inoperative.

INPUT: Potentiometer Recipe   a knob or "pot" is a adjustable resistor for creating a variable voltage

What:

The potentiometer is a just a variable resistor. In most implementations here you will see it used as a knob that will give you a variable voltage (e.g. 0 – 5 volts) to control things ho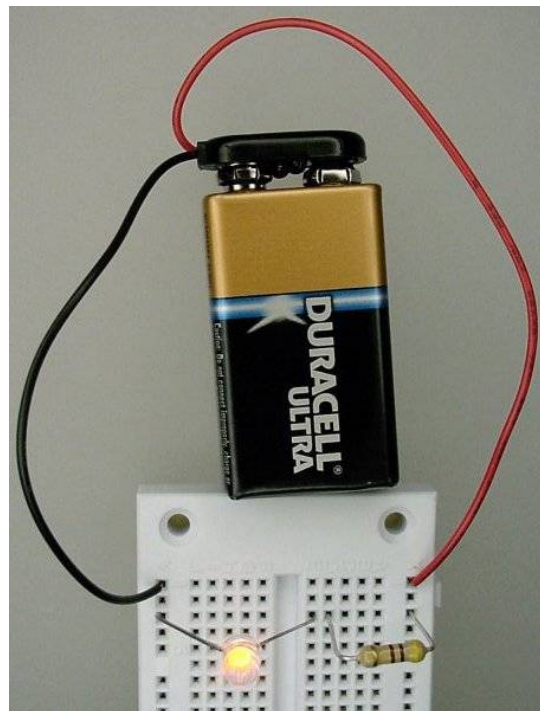oked to the arduino. The outer two pins, left(1) and right(3), get hooked to ground (GND) and +5 volts, and in this configuration the middle pin (2) will provide a 0 - 5 volt range that can in turn be applied to an arduino analog input such as A0. If operated like this, the arduino will read that value in (when instructed) and convert any analog voltage (at that moment) to a number between 0 - 1023 through a process called analog to digital conversion, much like an MP3 recorder does for audio.
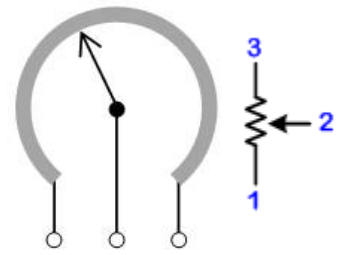
Mechanical and schematic diagrams for a potentiometer

How:

Firmly insert the potentiometer (also called a "pot") into the breadboard. Connect the leftmost side to ground (GND or -) and the rightmost side to power (+5v or +, or +3.3v if configured for 3.3v operation). The middle connector is the output of the pot in this case. Connect the middle wiper arm to an analog in pin on the Arduino. The A0 input is good.

Verify the circuit by reading and printing the pot value.

```
void setup() {
  // initialize serial communications at 9600 bps:
  Serial.begin(9600);
}

void loop() {
  int sensorValue0;
  sensorValue0 = analogRead(A0);  //read the pot input
  Serial.print ("Pot 0 value=");
  Serial.println(sensorValue0);

  // wait 1 second before the next loop
  delay(1000);
}
```

Turn the pot left and right, the printed value should go from (near) 0 to (near) 1023

Fail:

- Verify the pot is seated firmly in the protoboard
- Verify 5 volts across the pot's outter pins with a multi-meter
- Are you reading the input value into a variable?
- Are you printing the correct variable for testing?
        Discuss 'deadband' and "print on change"
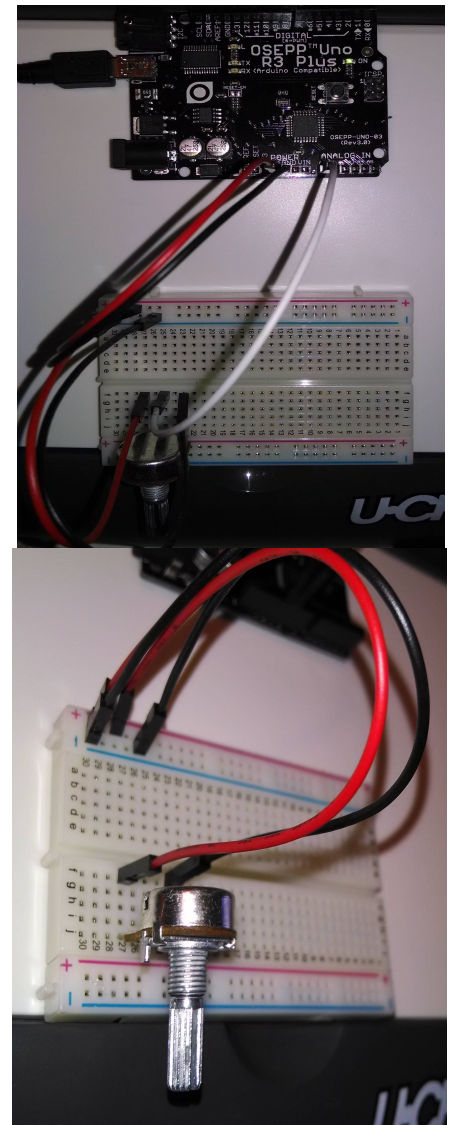
BUILD: LED Chase Light Recipe

What:

The goal of this recipe is to learn how to hook up multiple LEDs and resistors to the arduino in order that make them strobe back and forth to form a chase light. Add in an optional potentiometer and you can control things like LED chase speed or LED brightness.

You will need:
- a breadboard
- 6-7 LEDs
- a 330 ohm resistor
- 5 or 10k potentiometer (optional)

How:

Before hooking anything up, first note that the LED has a longer leg (positive) and a short leg (negative). Unlike a light bulb, LED lights have + and – polarity and need to be correctly connected.

> **WARNING:** If you get the polarity of an LED hooked up backwards it simply won't light. However, get the polarity right but *without a current limiting resistor* and you can blow it. Please don't blow our LEDs or we'll call you "smokey". :^)
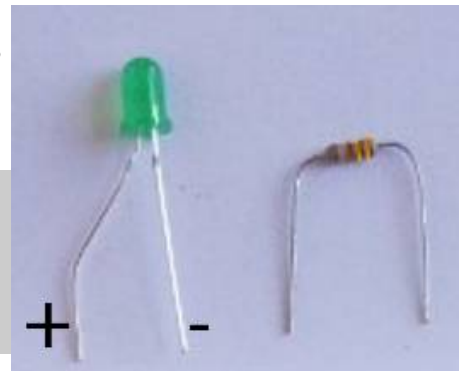
Connect Single LED + Resistor:
First, start off by hooking up just one LED and current limiting resistor and get that working on the arduino's digital output pin 12. To do this:
- Connect the GND on the Arduino to the 330 ohm resistor and then the resistor down to the breadboard's - or blue row. This is blue row is where you will connect the - side of the LED.
  > **NOTE:** Leave +5V disconnected from the breadboard for this recipe. We're using the + (red) row for something else in this circuit.
- Connect LED's positive (long) leg to digital pin 12
- Connect LED's negative (short) leg to - blue row of the breadboard
- Connect other side of the resistor to GND (blue row) on breadboard
- Load the "blink" program and change **digitalWrite(13);** to pin "12"
- Upload & run

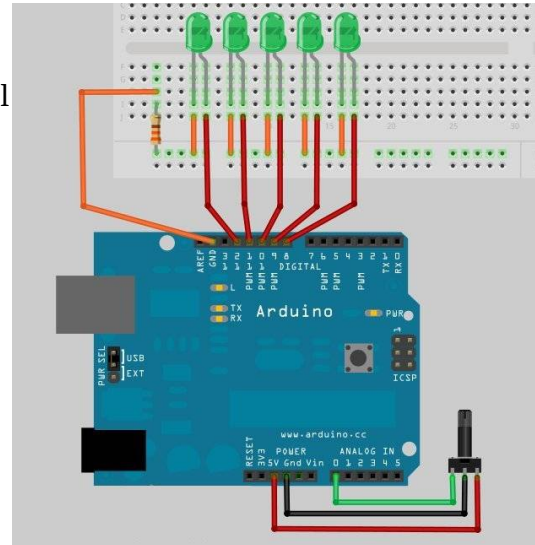Chase Lights:
After you have one LED up and running, hook up the remaining 4-5 LEDs the same way to pins 8, 9, 10 and 11, all back through the same resistor through the blue(-) power strip. See photo (right).
Hook up:

- LED positive (long) legs to digital pins 8, 9, 10, 11, 12
  (top "DIGITAL" input/output section)
- LED negative (short) legs to the resistor's common blue row
- Modify program **void setup()** section to configure LED pins 8-12 as outputs
- Compile and upload ⊕

TEST: Test to see if you can light up each LED with the digitalWrite() command

Use Potentiometer (knob) Value For Timing:

- Add potentiometer to breadboard to control variable chase light speed
- Wire pot leg pin 1 (left pin) to ground (GND or 0v) on arduino
- Wire pot leg pin 2 (middle) to A0 or "Analog0" (on "ANALOG IN" header (bottom right))
- Wire pot leg pin 3 (right pin) to +5v on arduino
- Use "analogRead(0);" function read or *sample* the pot value (0-5V maps to value 0 -1023)
- Replace blink's
        delay(1000);

    in milli-seconds with:
        delay(analogRead(0));

    to use the pot read 0-1023 value as the new delay value between LED flashes.

- Compile, upload and run code
- Twist knob to adjust chase light speed (delay)

Try This: You can either read the analogRead(0) just once at the beginning of your LED flashes (using it for each LED on/off cycle), or re-read the pot for each LED flash cycle. Try it both ways. Observe the difference.

Try This: See data from the arduino on your PC in real time with **Serial.print(analogRead(0));** to see your pot value in the serial console

Fail:

- LED is not lighting: Check the polarity or try new LED (another "smokey" may have used it ;)
- Pot does nothing: Make sure you have the pot's pins hooked up to GND and +5v correctly or that you're reading the correct analog input. If problems, then Serial.println() the value of the pot to verify it's working as expected.

CONTROL: Servo Recipe

a servo is a PWM, digitally driven motor that you can control the angular position of.

What:

Connect and control a servo motor (a digitally, position controlled motor) with the Arduino with a potentiometer (knob). You will read the value of the pot, and based on that value, change the position of the servo motor using one of the arduino's PWM (pulse width modulation) outputs.

How:

Use a 3 pin header to connect the servo to the protoboard.
- Connect the brown wire on the servo to ground.
- Connect the red wire on the servo to the Vin pin on the Arduino.
- Connect the orange wire to pin 9 on the Arduino.

WARNING: Placing a 330 ohm buffering resistor on the servo input line (orange) is a good idea to help protect the circuit in case of mis-wiring.   "Only you can prevent arduino fires Smokey."

Use the following code to test the servo.

```
#include <Servo.h>
// create a servo object
  Servo servo0;

void setup() {
  servo0.attach(9);  // servo is attached to pin 9
}

void loop() {

  servo0.write(60);  // tell servo to go to the 60 degree position
  delay(1000);        // wait 1 second
  servo0.write(120); // tell servo to go to the 120 degree position
  delay(1000);        // wait 1 second

}
```

Look for the preceding code to move the servo to the "60 degree" position, wait 1 second, then move the servo to the "120 degree" position. The actual movement degree may vary somewhat depending on the servo. The initial position of the plastic servo arm that presses onto the servo toothed shaft may be changed by gently pulling it up, off the servo, turning it and then pressing it back down, re-engaging the "teeth" in a different rotational position.
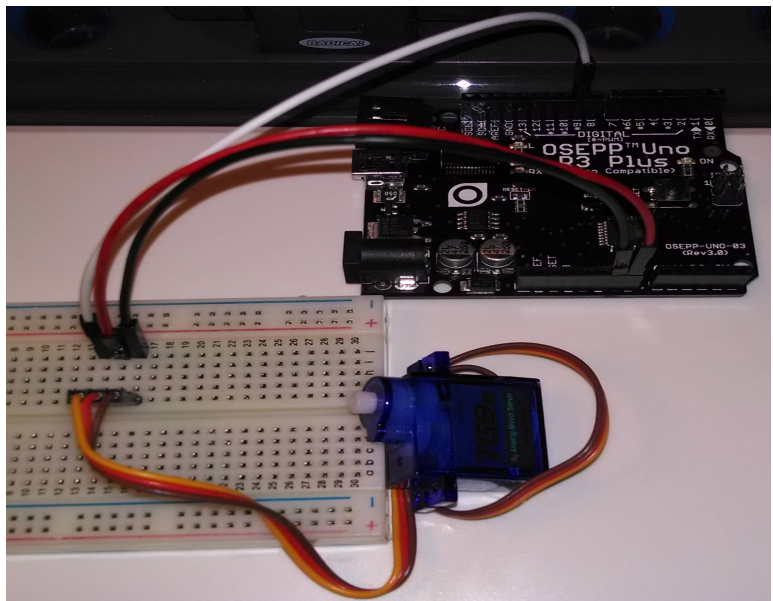
How:

The position of a servo is set by sending it a 1 – 2 millisecond pulse. A 1ms pulse represents approximately 0 degrees of servo rotation. A 1.5 ms pulse represents approximately 90 degrees. A 2 ms pulse represents a servo position of 180 degrees.

This pulse should be sent every 20 ms or so. The exact timing between pulses is not critical.

While we could easily write code to pulse the servo control the proper time (between 1 and 2 ms) every 20 ms, the Servo library used above takes care of that for us and can control multiple servos simultaneously.
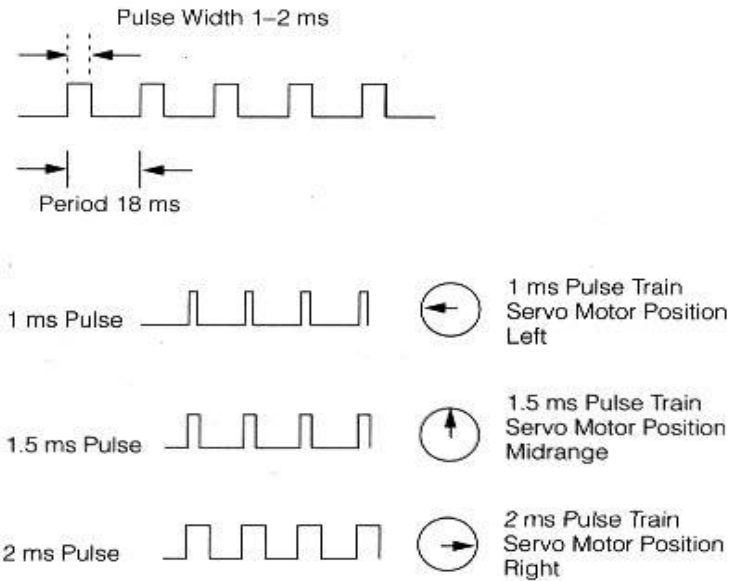
Illustration 1: Source: seattlerobotics.org

NOTE: Servo range may vary; not all servos have a full 180 degree range.

Fail:

Unplug servo power line an plug in back in. Listen for servo to make a small move if power is connected properly.

Servo's can consume more power than available from the Arduino and from laptop USB port. Try connecting the external battery pack for additional servo power.

Check that the values printed to the Serial Monitor make see what's going on as the pot is moved.

Make sure the brown and red servo wires go to to GND and +5v respectively (call instructor if unsure).

Make sure the orange wire is connected to the correct PWM pin on the Arduino, especially if there is more than one servo connected. Change to servo control pin defined in the software if necessary.

If the electrical connections are suspect, try replacing the 3 pin headers with 3 jumper wires

CONTROL: Robot DC Motor Control

Using the KeyesL298 H-Bridge motor controller w/DC motors.

What:

Connect the left and right motors to the motor controller. Then to adjust the speed of each DC motor, you program the arduino to provide opposite HIGH/LOW direction signals into the motor controller's INput1/INput2 direction signals, then together with an EnableA (for motor-A for example) the A motor will begin spinning in one direction. Invert the IN1/IN2 signals to LOW/HIGH + ENA(HIGH), and it spins in the opposite direction. To control the motor speed, use a ~PWM output pin on the arduino and send it a PWM signal between 0-255 (0 off, 255 full speed) using **analogWrite(100)** (for example) to get a medium-slow spin.

How:

If you're using a two motor robot, (see photo) hook up motor-A to the L298's top motor terminals and the left motor to the motor-B terminals.

M.Board	Function	Sensor Shield
ENA	motor-a enable	D6
IN1	motor-a +	D7
IN2	motor-a -	D5
IN3	motor-b +	D4
IN4	motor-b -	D2
ENB	motor-b enable	D3
GND	Ground	GND
+5V	Logic Supply	VCC

Jumper	On/Off
CSB	ON
CSA	ON
UR1(pullup)	OFF
UR2(pullup)	OFF
UR3(pullup)	OFF
UR4(pullup)	OFF
5V_EN	OFF
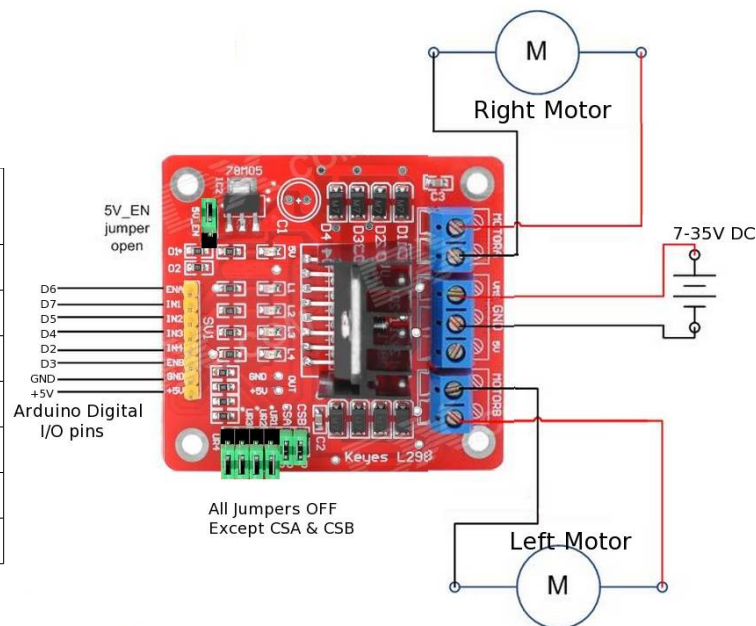
The polarity of the motors (motor-a +, motor-a -) doesn't even really matter as the opposing motor will probably be turned around (180 degrees), so for your application you simply change the HIGH/LOW signals you send the IN1 and IN2 board inputs and that toggles the motor direction. Here's the example code for driving the motors:

```
#include <Arduino.h>

//  Let's Code Blacksburg
//  Motor Drive test #1 code
//       (version .1 -ME )

// Motor controller signals and the arduino pin assignments
const int ENB=3;    // motor-b enable   PD3
const int IN4=2;    // motor-b –      PD2
const int IN3=4;    // motor-b +      PD4

const int IN2=5;    // motor-a –      PD5
const int IN1=7;    // motor-a +      PD7
```

```
    const int ENA=6;       // motor-a enable PD6

    void setup()
    {
    // lcbb  motor CONTROL test code.
      // Motor A  setup
      pinMode (ENA,OUTPUT); //motor A enable
      pinMode (IN2,OUTPUT); //motor A wire 1 polarity
      pinMode (IN1,OUTPUT); //motor A wire 2 polarity

      // Motor B setup
      pinMode (ENB,OUTPUT); //motor B enable
      pinMode (IN4,OUTPUT); //motor B wire 1 polarity
      pinMode (IN3,OUTPUT); //motor B wire 2 polarity

      pinMode (13,OUTPUT);   // LED for testing

      // Set polarity for motor A
      digitalWrite (IN4,LOW);
      digitalWrite (IN3,HIGH);

      // Set polarity for motor B
      digitalWrite (IN2,LOW);
      digitalWrite (IN1,HIGH);

    }

    void loop()
    {
      int motorSpeed=255;   // Any PWM range from 0-255

      digitalWrite(13,HIGH); // Turn on LED
      analogWrite(ENA,255); // set Motor A speed 100%
      analogWrite(ENB,255); // set Motor B speed 100%

      delay(3000);

      digitalWrite(13,LOW);   // Turn off LED
      analogWrite(ENA,0); // set Motor A speed 0%
      analogWrite(ENB,0); // set Motor B speed 0%

      delay(3000);
    }
```

If both of your motors are not turning on and off, you either have coding/typo problem or a hardware problem and need to see the **Fail:** section. If your motors are both turning on and off correctly, then try commenting out the last two **analogWrite** lines like this:

```
    //analogWrite(ENA,0); // set Motor A speed 0%
    //analogWrite(ENB,0); // set Motor B speed 0%
```

so your motors will be always on. Next flip your bot over, plug in the batteries to the arduino board and see if your bot goes in a straight line.

Q: Does your bot go in a solid straight line (with the code above)? A:_____

Not only can you vary the speed of each motor, but each motor has a unique, minimum usable speed (PWM) value where it begins to turn – under which it's really not usable. In fact, it's best to put the motors under slight load (touch them with your fingers or let it push the bot) and you'll see that minimum speed is even higher under load than at no load.

Flip your bot on it's back so you can watch the wheels and repalce your **loop()** code block with the code below, and using the serial console (w/ the **Serial.print(motorspeed)** below), see what your minimum, loaded, motor speed is (the minimum PWM speed at which both wheels can move at roughly the same speed):

```
... (setup and other code) ...

void loop()
{
  int motorSpeed=0;          // starting speed
  int minSpeed=motorSpeed;   // minimum speed -\
  int maxSpeed=255;          // maximum, adjust -these to find what' best

  while (motorSpeed < maxSpeed) {
      motorRight(motorSpeed);
      motorLeft(motorSpeed);
      Serial.println(motorSpeed); // Print speed over serial
      motorSpeed+=2;       // increment the motor speed by 2
      delay (100);
  }

  while (motorSpeed > minSpeed ) {
      motorRight(motorSpeed);
      motorLeft(motorSpeed);
      motorSpeed-=2;     // decrement the motor speed by 2
      Serial.println(motorSpeed); // Print speed over serial
      delay (100);
  }
}

void motorRight(int speed)
{
  // Control the speed of Right motor-
  analogWrite(ENA,speed);  // Send a PWM speed control to Right motorn
  // might have to swap with PD3 if your robot is wired differently than
mine
}

void motorLeft(int speed)
{
  //Control the speed of motorB
  analogWrite(ENB,speed);  // Send a PWM speed control to Left Motor
}
```

Looking at the serial console's output...

Q: What was your minimum usable motor speed (for both motors under load?) A:_____

Now set your `minSpeed` for that value, re-upload and see roughly where your bot travels in a straight line and make that value your new minSpeed and decrease your max speed to 2 higher than minSpeed and see if you can keep your bot running straight.

Q: What was your best straight line speed? A:_____

Fail:

One or both motors are not spinning at all.
- **Code:** Typo in the ENA, ENB or IN1/2/3/4 pin assignments
- **Hardware:** Double check your wiring (motor board to Arduino/Shield)
- **Hardware:** Wiring of the motors to the motor board, the power or the wires going to the motors
   (have an instructor use a meter to check you motor resistance, motor wire/solder)
- **Hardware:** Your batteries are low (they should each measure between 3.7 – 4.2v)
   (have an instructor use a meter to check your voltage levels)

INPUT: Ultrasonic "Eyes" Range Sensor   Using an ultrasonic ping sensor to detect object distance.

What:

The ultrasonic ping sensor is simply a digital module that allows you to send out a "ping" in the form of a microsecond sound burst, listen for the return echo and given the approximate speed of sound in air (using the delay between the ping and the echo back) calculate the distance to the object. Some ping sensors have a single ping/echo pin that you both send on and listen on (output + input), while others (such as the one we're using here) have one pin dedicated to the ping (called the "trigger") and one pin dedicated to

How a ping sensor detects distance.

receiving the echo (called "echo"). There is no analog signal processing that needs to be done as this is all handled by the module, so we're just dealing with a nice clean, processed digital I/O signals and microsecond timing. However, the microsecond timing is very critical, and some cheaper sensors can not resolve distances much over a foot out.

How:

Hook up the ping sensor to any two digital I/O pins free on your arduino. Here we're using pins 12 (trigger) and 13 (echo). Be sure the Vcc (+5V) and GND are correctly wired before applying power.

No load up the build in example code from Examples / Sensors / Ping. This example code is made for the three pin Parallax version (with just one pin for trigger & echo), however you can easily modify it to work as seen below:
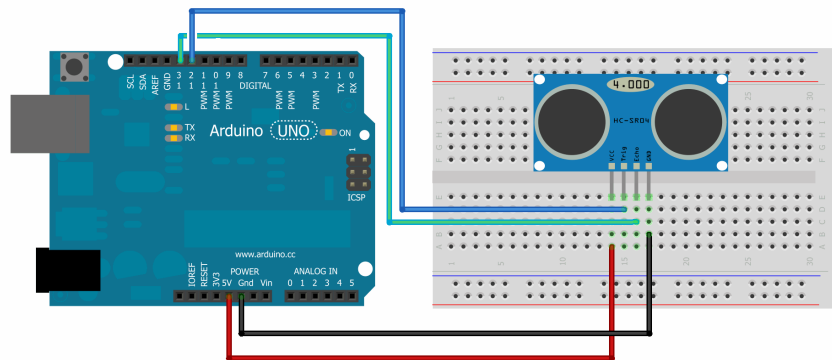
NOTE: If you have the US-100 version of this module, remove the jumper from the rear to make it use pulses instead of serial data.

```
#include <Arduino.h>

const int triggerPin = 12;
const int echoPin = 13;

void setup() {
  Serial.begin(9600);            // For outputting distance data
  pinMode(echoPin, INPUT);       // Make echoPin an INPUT
  pinMode(triggerPin, OUTPUT);   // Make the triggerPin an OUTPUT
}

void loop()
{
  // establish variables for duration of the ping,
  // and the distance result in inches:
  long duration, inches;

  // The PING))) is triggered by a HIGH pulse of 2 or more microseconds.
```

```
    // Give a short LOW pulse beforehand to ensure a clean HIGH pulse:
    pinMode(triggerPin, OUTPUT);
    digitalWrite(triggerPin, LOW);
    delayMicroseconds(2);
    digitalWrite(triggerPin, HIGH);
    delayMicroseconds(5);
    digitalWrite(triggerPin, LOW);

    // A HIGH
    // pulse whose duration is the time (in microseconds) from the sending
    // of the ping to the reception of its echo off of an object.
    duration = pulseIn(echoPin, HIGH);

    // convert the return echo time into a distance
    inches = duration / 74 / 2;

    Serial.print(inches);        // print it out to the serial port.
    Serial.println("in, ");

    delay(100);
}
```

Git code: **https://github.com/LetsCodeBlacksburg/arduino-robotics/tree/master/LCBB_ping_trigger_echo_sensor_simple**

After compiling and running this, then click on the serial monitor icon 🔍, you should get a stream of distances like this:

```
10in,
11in,
7in,
6in,
4in,
3in,
2in,
1in,
1in,
1in,
1in,
2in,
```

Think about at what distances you want to do things like (if doing a robot) slow your robot, stop your robot, or change directions.

Fail:

No distance readings can result from:
- bad wiring of power or trigger / echo pins
- not defining triggerPin as output and echoPin as input (or using them correctly)
- code typos

INPUT: Line Following IR Sensor

(Using a three-element infrared line sensor.)

What:

Line following robots need a way of detecting light vs dark lines on the floor. This three element digital sensor (with potentiometer threshold setting) is a popular low cost way of creating line following robots.

Three of these make a great line follower

How:

Each line follower sensor has a Voltage pin (V), and Ground pin (G) and a Signal pin (S).   One way to connect a three IR line follower circuits is by using the often unused analog inputs A0, A1 and A2 for the three signal lines from the sensors. The sensor's G & V pins can be connected to any ground and Vcc voltage sources. If using the Sensor Shield, these pins are usually free on any unused, three pin digital I/O connectors. The point here is to use the G & V pins on unused digital I/O connectors, but <u>route the S (signal) lines over to an analog inputs, not the unused digital I/O S pin</u>.

> **NOTE: The code below uses the serial monitor to watch all three modules. The serial montior is a great way of watching the state of a lot of things at the same time like this.**

Here's the example code for testing all three sensors while monitoring them over the serial monitor :

The wiring of the three element IR sensor

```
#include <Arduino.h>

//  Let's Code Blacksburg
//  IR sensor test code
//       (version .2 -tweeks )
//  This code reads the three IR line follower sensors
//  (Left, Middle and Right) on analog pins A0, A1
//   and A2,  and outputs L for black line or _ for
//   white surface (no line) on the serial monitor.

int sensorLeft=0;
int sensorMiddle=0;
int sensorRight=0;

void setup()
{
    Serial.begin(9600);  // for outputing t-shooting codes
}

void loop()
{
  // Sensor readings give you a 1 if it sees the black line
  // and a 0 if it sees white.

  sensorLeft=digitalRead(A0);    // Left uses analog A0
  sensorMiddle=digitalRead(A1);  // Middle uses analog A1
```

```
      sensorRight=digitalRead(A2);    // Right uses analog A2

      if (sensorLeft==1) {
        Serial.print(" L ");
      }
      else {
        Serial.print(" _ ");
      }

      if (sensorMiddle==1) {
        Serial.print(" L ");
      }
      else {
        Serial.print(" _ ");
      }

      if (sensorRight==1) {
        Serial.println(" L ");
      }
      else {
        Serial.println(" _ ");
      }

    }
```

The latest version of this code can be found here:
https://github.com/LetsCodeBlacksburg/arduino-
robotics/blob/master/LCBB_bot_tri_ir_line_sensor_simple/LCBB_bot_tri_ir_line_sensor_simple.ino

To test this, make a "testing card" out of a 3x5 card and black electrical tape.   Place the bot on it's back
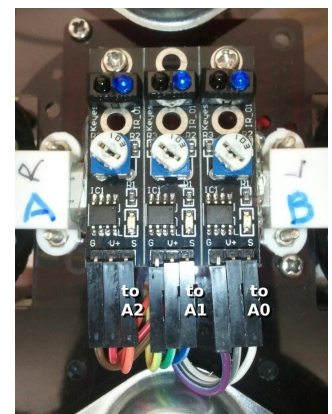and move the tape-card back and forth in front of the sensors.

Here's the example output of the serial monitor from this test code as you move the tape-card across the
sensors, along with some comments I added indicating what you should tell your robot (motors) to do:

```
 _  _  L     Go right
 _  L  L     Go right a little
 _  L  L     Go right a little
 _  L  L     Go right a little
 _
 _  L  _     Go straight
 _  L  _     Go straight
 L  L  _     Go left a little
 L  _  _     Go left
 L  _  _     Go left
 _  _  _     Keep going the way you were before losing the line
```

Get it? See how this

Fail:

Common failure modes of these sensors are:
  • **Hardware:** Hooking to the wrong pins (power or signal lines)
  • **Hardware:** The underside of the board is shorting out on a metal surface (very bad)
  • **Software:** Typos using analog reads instead of `digitalRead(A0)` (for example)

CONTROL: Potentiometer Control of a Servo's Position Recipe (read a pot, move a servo)

What:

A potentiometer is just a variable resistor in the form of a knob. In order to use a potentiometer to control a servo:
- Follow the the potentiometer recipe, connecting the pot output to the Arduino pin A0 (analog 0).
- Follow the Servo recipe, connecting the servo input wire to Pin 9 of the Arduino.
- Use the program below

How:

Here is a sample of how it could be wired up.

After testing the Pot and servo separately use the following code to test Servo control by a potentiometer.

Turning the pot left and right should cause a corresponding rotation in the servo.

The servo may move "faster" or "slower" than the pot depending on the map in the code below.

(We are using the values 30 to 150 for the degree settings, because some servo's have difficulties at the extremes of their movement.)
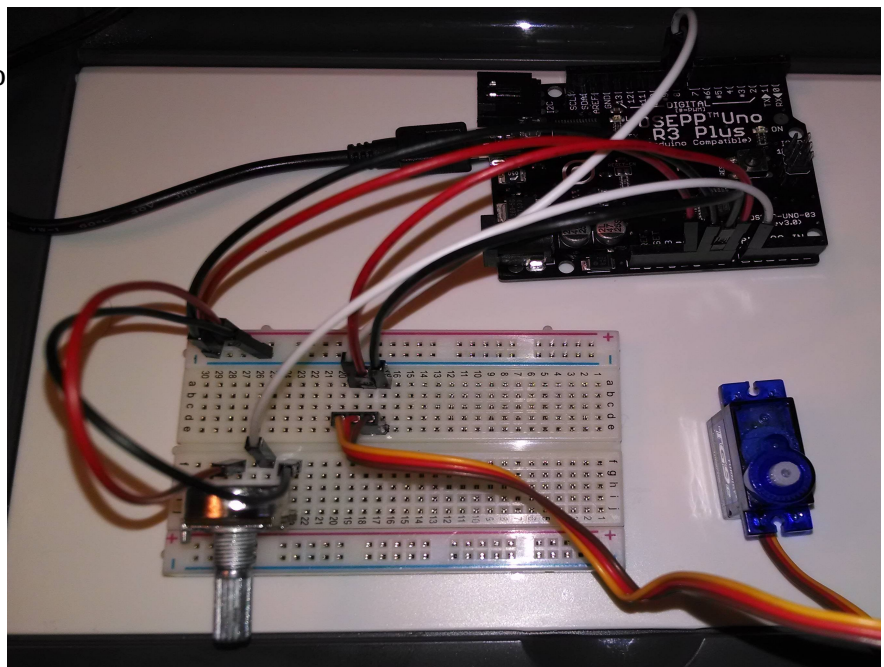

```
#include <Servo.h>  // servo library
// create a servo object
  Servo servo0;

void setup() {
  // initialize serial communications at 9600 bps:
  Serial.begin(9600);
  servo0.attach(9);  // servo is attached to pin 9
}

void loop() {
  int servo0Setting=90;  // for the servo position later

  int sensorValue0;      // Read Pot value

  sensorValue0 = analogRead(A0);
  Serial.print ("Pot 0 value=");
  Serial.println(sensorValue0);
```

```
    // Map the pot reading to the servo degree setting
    //    we'll use 30 to 150 degrees for the servo

    servo0Setting=map(sensorValue0,0,1023,30,150);
    Serial.print ("Servo 0 setting");
    Serial.println (servo0Setting);


    // Set Servo position
    servo0.write(servo0Setting);  // tell servo to go to the designated
position


}
```

Try This: You can change the direction of the servo to potentiometer mapping by swapping the pot input values in the map statement.
e.g. change this:
> **servo0Setting=map(sensorValue0, 0, 1023, 30, 150);**
> to this:
> **servo0Setting=map(sensorValue0, 1023, 0, 30, 150);**

Of course different pins for the pot and servo may be used, but the test program will have to be modified to match.

Fail:

OUTPUT: Sound Generation with Arduino (making tones, sounds and music with speaker)

What:

Sounds are generated by feeding a changing voltage into a speaker. How often the voltage changes per second is the frequency of the sound. A "pure" sound would be a signal that changes smoothly over time – like a sine wave.
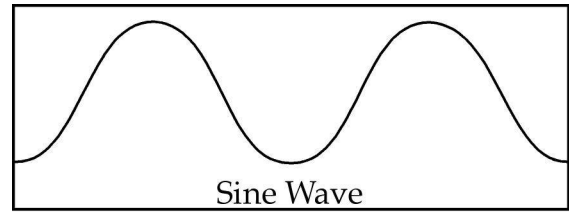
Sine waves can be hard to generate digitally. But for making basic tones, a square waves are much easier to generate, require no real additional hardware and often works for most uses.

Sine Wave

SQUARE WAVE
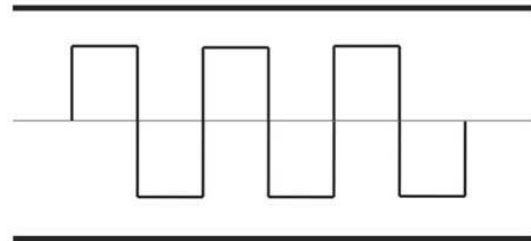
How:

Parts Needed:
- Speaker (The larger one in the kit. Not the 5v buzzer.)
- Red and black wires.

Assembly:

- Connect the black wire from the black side of the speaker connector to the ground hole on the Arduino.
- Connect the red wire from the red side of the speaker connector to DIGITAL pin 5.

> **NOTE:** The wires are a tight fit into the speaker connector. It may push the speaker wire out of the connector. So try to hold the speaker wire and connector while pushing in the other wires.

The Code:

```
int soundOut = 5;       // Sound output pin
int del = 2;            // This determines how long the signal stays
                        // HIGH or LOW in milliseconds

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

void loop()
{
  digitalWrite(soundOut, HIGH);
  delay(del);
  digitalWrite(soundOut, LOW);
  delay(del);
}
```

Pretty simple, right? The delay is in milliseconds. So each pass through the loop is a complete "cycle" and bit more than 4 milliseconds total. There is some overhead to executing the various statements so that is why it comes out more than 4, but only by a little and it's close enough for us. That makes the frequency about 250 Hertz. That's close to a "B" an octave below middle "C" on a piano. Here's a chart for reference:

So let's modify it a bit to change tones back and forth – kind of like a European siren.

```
int soundOut = 5; // Sound output pin
int del = 1;      // Delay or duration of HIGH and LOW pulses
                  // Longer delay times = lower frequency
                  // So shorter delay 1 = high tone, longer delay 2 = low tone

int count = 0;    // Counter to control how long to play a high or low tone

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

void loop()
{
  digitalWrite(soundOut, HIGH);
   delay(del);

  digitalWrite(soundOut, LOW);
   delay(del);

  // Since del is the square wave's pulse time (inverse of frequency)
  // and count is a total amount of time, add del to count so we
  // always wait the same amount of time
   count += del;

  // If the desired amount of time as been reached, reset the
  // count and toggle the frequency (1 to 2)
  if(count > 500)   // if past tone count (try 100, 25, 10, 4, 3, 2, 1 !)
  {
    count = 0;      // reset the duration counter
    if(del == 1)    // a == compares two things
    {
      del = 2;      // toggles the tone
    } else {
      del = 1;      // a single = sets a variable
    }
  }
}
```

Well, that's getting complicated! And if we want our Arduino to do anything else at the same time, it's going to get pretty nuts!

But remember the PWM info from the motors & servos? That's a square wave too! So lets try that.

```
int soundOut = 5;        // Sound output pin

void setup()
{
  pinMode(soundOut, OUTPUT);

  // Doing this here shows that we keep generating sound
  // Even after setup is complete and the loop can do
  // other things
  analogWrite(soundOut, 128);
}

void loop()
{
}
```

That's handy! See what happens if you change the analogWrite value.

But there's a limitation here. The PWM signal the Arduino generates is 490Hz. So you can't change the frequency.

So here's a better trick...

```
int soundOut = 5;        // Sound output pin

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

void loop()
{
  tone(soundOut, 440);    // Surprise! Arduino can play sounds directly!
}
```

Wow, now that's handy! It's similar to the PWM technique in that the sound keeps playing by itself (try moving the "tone" command to the setup instead of loop). In fact, internally it uses the same hardware. But instead of a single frequency that varies the pulse width, you tell it the frequency and it's always a 50-50 high/low division. You can also add a third value for milliseconds (1000ms = 1 second) so it will only play for a given length of time. Pretty slick! Here's a tone chart.

Note	1	2	3	4	5	6	7	8
B	62	123	247	494	988	1976	3951	
A sharp/B flat	58	117	233	466	932	1865	3729	
A	55	110	220	440	880	1760	3520	
G sharp/A flat	52	104	208	415	831	1661	3322	
G	49	98	196	392	784	1568	3136	
F sharp/G flat	46	92	185	370	740	1480	2960	
F	44	87	175	349	698	1397	2794	
E	41	82	165	330	659	1319	2637	
D sharp/E flat	39	78	156	311	622	1245	2489	4978
D	37	73	147	294	587	1175	2349	4698
C sharp/D flat	35	69	139	277	554	1109	2217	4434
C	33	65	131	262	523	1047	2093	4186

Try This: Write twinkle twinkle little star! (try tone duration of 400ms and delay between notes of 500ms) and the chart above (starting in key of C).

So why even bother with other sound techniques? All technology has pros and cons. Inthis case, there are some limitations with the "tone" command. For one thing, it can only play one tone at a time. So even if you have multiple speakers hooked up, you can't sent sound to all of them at once. Another problem is that since it uses some of the same hardware inside the Arduino, you can't use PWM on pins 3 or 11 at the same time you're generating tones. If these aren't a problem, and the sound quality is good enough, use tone.

If you're interested in these advanced techniques, check out "Advanced Arduino Sound Synthesis" in "Make:" magazine volume 35, page 80.

INPUT: Light Sensor (detecting light intensity with a CdS photo-sensor)

What:

A light sensing cell or photo-resistor or Cadmium Sulfie (CdS) cell turns light into a variable resistance. Usually dark = high resistance and high light = low resistance. In simple terms it is a bit like a potentiometer or volume knob for light.

Just like a variable pot resistor, it can be used as an input to change programming values for time delays, LED light intensity, or even motor movement or sound!
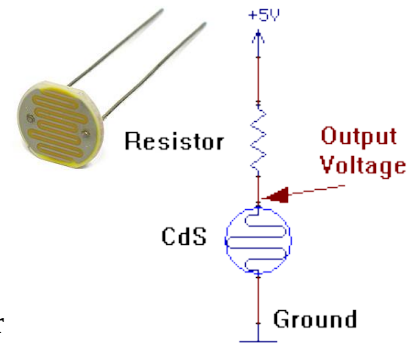
How:

This circuit is pretty straigh forward. It is a light sensor and a 10k ohm resistor wired in series (between +5V and GND on the breadboard), and you take the voltage measurement from where the sensor and resisor meet on the breadboard.

This will give you a roughly 0.5v (value in the arduino with a light shining on it) to 3.5v (value in the arduino in ambeint light) range read in from the arduino.
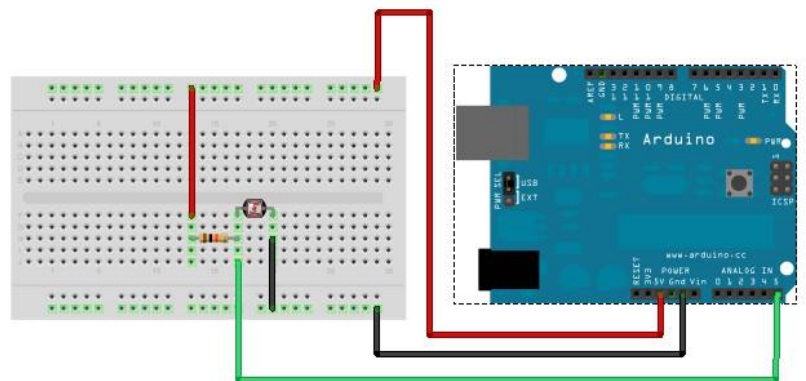
Here's the code:

Wiring diagram for the 10k resistor and light sensor.
Source: http://wiki.xinchejian.com/wiki/Introduction_to_Arduino

```
// With a 1k resistor in series with this light sensor, you should
// get around 900 range in the dark, around 600 in ambient, and
// around 2-200 range with an LED shining on it.
// This is for a CdS photo resistor cell with approximate attributes
// of:    DARK ~ 100k      AMBIENT ~ 2k      LIT ~ 0.7k

int ls = A5;               //analog pin to which LS is connected
int ls_value = 0;          //variable to store LS values,
                           //we'll set this to zero first.

void setup()
{
  Serial.begin(9600);    //Start the serial monitor
  pinMode(ls, INPUT);    //Tell arduino that this pin is for input
}

void loop()
{
  ls_value = analogRead(ls);     //reads the light sensor values
  Serial.println(ls_value);      //prints the LS values to serial monitor
  delay(50);                     //slow it down a bit
}
```

Try This: Ever hear of a Thermin? It's a musical instrument you can play by holding your hand over it! You can make one by combining this light detector with the last program from the sound recipe. You'll need to add their setup int variable and setup sections to your own, and then use the tone line below in your loop:

```
tone(soundOut, ls_value, 10);  //You just made a light Theremin
```

Tip: For a smoother sound, take out your delay(50); line!

Q: Do you like the sounds you can create with light?
Look over the recipe for "Ultrasonic Eyes" and see if you can make music with that sensor too!
Do you think it will sound any different?

Fail:

There's not a lot ot go wrong with this setup, however commong problems may include:
- not having the power—to--resistor--to--lightsensor--to-GND hooked up correctly
- not tapping your A5 input reading off where the light sensor and resistor meet on the breadboard.
- Not having all your variables from the two programs both integrated into a the three unified int, setup and loop sections.

INPUT: People Motion Sensor   (PIR motion sensor for detecting motion up to 20ft away)

What:
The "PIR" in PIR motion sensor stands for Passive Infra Red and motion sensor modules that passively detects body heat (or change in the temperature profile) in a visible space for up to 20ft away. Its operation is very simple and cheap to employ, and is commonly used in burglar alarms and outdoor motion activated lights.
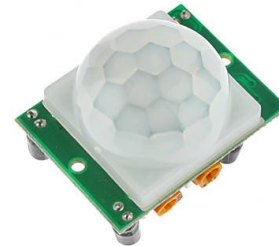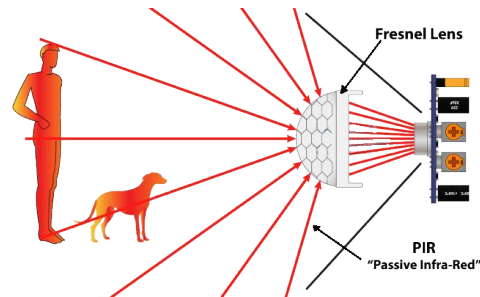
How:
The PIR sensor is very simple to operate. It has only three pins. One power (+5V) pin, one ground (GND) pin and one active high OUT signal pin*. Once the module powers up and averages out its field of view (this takes between 15-30 seconds), then the OUT pin will go HIGH when there is motion and low a short time after the motion subsides.

> *NOTE: This sensor runs on 5V, but sends out its OUTput signal at 3.3volts, which means that the most reliable setting for using it with an arduino is achieved by putting your arduino into 3.3v digital mode (usually a small 5v/3.3v switch on the arduino).
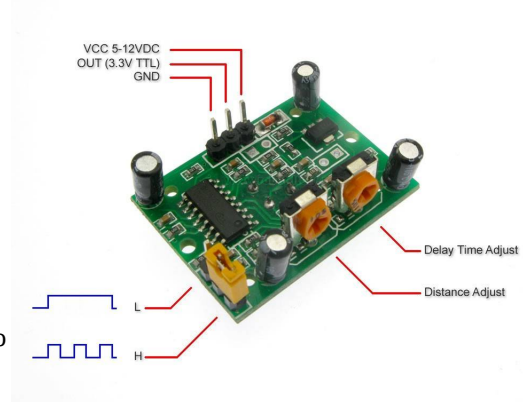
The L H mode jumper controls whether constant movement creates a constant HIGH condition (set to L) , or multiple HIGH/LOW pulses for as long as it senses motion (H mode). In most cases the L-mode is desired (HIGH for as long as there in movement). The delay-time and distance adjust (sensitivity) potentiometers should not be adjusted without speaking to an instructor first).

Connection is very strait forward, but we will connect it in this lab to the arduino's digital I/O pin #2.

The arduino requires no special software library to use this PIR module, but simply reads the OUT signal pin – HIGH for motion, LOW for no motion.. In fact, this sensor does not even need an arduino, but could be connected directly to a battery and power relay for controlling lights, space heaters, alarms, etc. However the arduino is convenient for giving such projects a little more intelligence and flexibility.
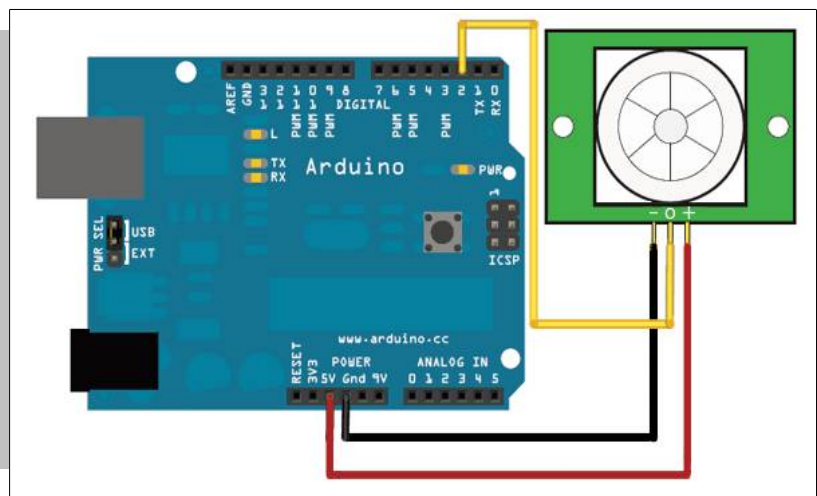
Here's the code for simply turning on and off a light with your PIR motion sensor:

```
////////////////////////////
//VARIABLES
int calibrationTime = 20; //the time we give the sensor to calibrate (20-30sec)
int pirPin = 2;      //the digital pin connected to the PIR sensor's output
int ledPin = 13;     // the built in LED pin

////////////////////////////
void setup(){
  Serial.begin(9600);
  pinMode(pirPin, INPUT);        // Set up the various I/O pins
  pinMode(ledPin, OUTPUT);
  digitalWrite(pirPin, LOW);

  //give the sensor some time to calibrate (monitor state on the serial port)
  Serial.print("calibrating sensor ");
    for(int i = 0; i < calibrationTime; i++){
      Serial.print(".");
      delay(1000);
      }
    Serial.println(" done");
    Serial.println("SENSOR ACTIVE");
    delay(50);
  }

////////////////////////////
void loop(){

  if(digitalRead(pirPin) == HIGH){       // If the PIR is HIGH
      digitalWrite(ledPin, HIGH);        // then turn on the LED pin and
      Serial.println("HIGH (motion)");   // print "HIGH (motion)" on serial port
  }
  else {
      digitalWrite(ledPin, LOW);         // else, make the LED pin low and
      Serial.println("LOW (no motion)"); // print "LOW (no motion)" on serial port
  }
  delay(250);                            // 250mS delay
}
```

What else can you control or output using a motion sensor?

Fail:
The most common problems with this circuit is getting your pins hooked up wrong. BE SURE TO
HAVE A TA CHECK YOUR WIRING BEFORE POWERING UP YOUR CIRCUIT!
-Check your wiring
-Make sure you program in a wait-time for your sensor to initialize
-Be sure your arduino is set to 3.3v mode and that you are sending 5V to power the sensor

INPUT / OUTPUT: The LCD / Keypad shield   (Displaying text and taking keypress inputs)

What:

By itself the Arduino has limited capability for interacting with humans. The LCD / Keypad shield provides a simple way to display information and allow direct user input through four direction and one select button.

> *NOTE: Shields are plug in arduino modules that are super easy to use, however their pins are easy to bend, so **please be careful and don't force the shield or be rough with it!***

How:

Hardware setup for shields is extremely easy – just power off/unplug the arduino, then carefully line up the shield pins and plug in the shield! The only gotcha is to be sure and align the pins on the shield correctly.

> ***Lining up and Connecting/Disconnecting a Shield:*** *There are fewer pins on the shield than there are holes in some Arduinos. To line up the pins correctly, hold the Arduino with the USB connection to your left and the shield with the buttons below the display in your right hand. Line up the farthest right shield pins with the farthest right Arduino holes. Carefully make sure all the remaining pins are aligned with the corresponding holes. Watch out for slightly bent pins! (call an instructor over if you see bent pins) Slowly squeeze the two boards together, being careful NOT to press on the display. To disconnect a shield, slowly rock it back and forth (end to end) while gently pulling it away from the arduino until it separates. Never just pry it off or you'll bend delicate pins!*

Whenever you use any shield, it's important to understand what pins the shield actually use, both for being able to communicate with it and for knowing any conflicts which could occur if you want to stack shields. Below is a list of the pins used by the LCD / Keypad shield.

Pin	Function
Analog 0	Button (select, up, right, down, and left)
Digital 4	DB4
Digital 5	DB5
Digital 6	DB6
Digital 7	DB7
Digital 8	RS – Register Select (Data or Signal Display Selection)
Digital 9	Enable
Digital 10	Backlight Control

You might be tempted to stack the LCD shield on top of the Ethernet shield. However if you look at the pinouts you'll find that the Ethernet shield also uses pins 4 and 10. So you wouldn't be able to stack and use them at the same time. Even though they are located on the same board, the buttons and the LCD are logically separate and are handled separately. We'll start with the buttons.

<u>Reading the buttons:</u>

The buttons on the shield use a rather clever design. If you've connected buttons to the Arduino in the past, you probably connected them to digital inputs, maybe with a pull-up resistor. But that limits you to one button per pin. However this shield uses a single ANALOG pin for reading all the buttons. How is this possible?

The buttons are wired to a series of resistors so that when you press a button, a different voltage is presented to the analog 0 pin, and thus a different reading when you call analogRead for each button.

Enter the following code:

```
void setup() {
  // initialize serial communication at 9600 bits per second:
  Serial.begin(9600);
}

// the loop routine runs over and over again forever:
void loop() {
  // read the input on analog pin 0:
  int buttonValue = analogRead(A0);
  // print out the value you read:
  Serial.println(buttonValue);
  delay(1);        // delay in between reads for stability
}
```

This code reads the analog pin and writes the result to the serial port. So after uploading this sketch, open the serial monitor in the IDE and watch what happens.

Q: Press the buttons. What happens with the different buttons?_____
_____

TA Sign-Off: _____

Did you press the right-most button? It may be labeled RST or RESET. This button is different from the others. This button is hard-wired as a RESET button. Pressing it will restart the Arduino, much like if you unplugged the power and then plugged it back it. You can't read or use it like the others.

Let's expand on the original button code a bit with a helpful function to decode those buttons into something a bit friendlier.

```
#define btnRIGHT  0
#define btnUP     1
#define btnDOWN   2
#define btnLEFT   3
#define btnSELECT 4
#define btnNONE   5

// read the buttons
int read_LCD_buttons()
{
 int adc_key_in = analogRead(0);       // read the value from the sensor
 // These values should work with all current keypads.
 // If you get unexpected results check the readings you get with you shield

 // We make this the 1st option for speed reasons
 // since it will be the most likely result
 if (adc_key_in > 1000) return btnNONE;
 if (adc_key_in < 50)   return btnRIGHT;
 if (adc_key_in < 250)  return btnUP;
 if (adc_key_in < 450)  return btnDOWN;
 if (adc_key_in < 650)  return btnLEFT;
 if (adc_key_in < 1000)  return btnSELECT;
 return btnNONE;  // when all others fail, return this...
}

// the setup routine runs once when you press reset:
void setup() {
  // initialize serial communication at 9600 bits per second:
  Serial.begin(9600);
}

// the loop routine runs over and over again forever:
void loop() {
  // read the input on analog pin 0:
  int sensorValue = read_LCD_buttons();
  // print out the value you read:
  Serial.println(sensorValue);
  delay(1);         // delay in between reads for stability
}
```

Enter, upload, and run the code as before. Now in your serial monitor window you should see '5' if nothing is pressed and other values between '0' and '4' when pressing a button.

Why not code in the exact values you saw in the first program? That might not be reliable or portable. While all the shields use the same common design and values, there is still some variation between parts. Each value should be within 20% of it's value, but unless something is REALLY wrong the values will be within a predictable range, so by mapping those values to a ranges we can make our program tolerant of differences between boards or temperature effects.
So that's all there is to the buttons! So let's move to the more exciting bit – the LCD!

<u>Writing to the LCD:</u>

Now, you **could** attempt to program the LCD directly. It uses a standard HD44780 LCD controller in 4-bit mode if you want to look up how to do it. But unless you have some very special requirements you'll want to stick with the standard LiquidCrystal library that comes with the Arduino IDE. It makes using the LCD a breeze!

The full reference for the library can be found at http://arduino.cc/en/Reference/LiquidCrystal

The LiquidCrystal library is actually fairly generic. It will work with any HD44780 based display – you just have to tell it how it's connected. Using the shield the connection is fixed, but you still have to tell the library those connections – it can't tell automatically what you have plugged in.

Here's a simple program setting up and displaying a message and the time since the Arduino started up.

```
#include <LiquidCrystal.h>

// Create the LiquidCrystal object and define the pins used on the LCD
shield
// The parameters are: rs, rw, enable, d4, d5, d6, d7
LiquidCrystal lcd(8, 9, 4, 5, 6, 7);

void setup()
{
 // Start the library, telling it we have a 16 column x 2 row display
 lcd.begin(16, 2);
 // Put the cursor on the first column of the first row
 lcd.setCursor(0, 0);
 lcd.print("Hello, world!");
 // First column of second row
 lcd.setCursor(0, 1);
 lcd.print("Seconds: ");
}

void loop()
{
 lcd.setCursor(9,1);      // move cursor to second line "1" and 9 spaces over
 lcd.print(millis()/1000);    // display seconds elapsed since power-up
}
```

Pretty straightforward, right? There are a couple of "gotchas" to keep in mind:

1. Notice that, like most everything else in programming, the numbering of the rows and columns start with 0, not 1.
2. When you print something, it only prints EXACTLY what you tell it. If there was something else on the row before and your new text is shorter, it will only overwrite as much as the new text leaving behind a portion of the old. So you may need to pad out your strings with spaces to clear out any old output.

There are a lot more capabilities in the library – scrolling, blinking, and even creating special characters. Look at the reference linked above for more information.

So for now, let's put all this together, reading buttons and displaying the result.

```
//Sample using LiquidCrystal library
#include <LiquidCrystal.h>

/******************************************************
This program will test the LCD panel and the buttons
Mark Bramwell, July 2010
http://www.dfrobot.com/wiki/index.php?
title=Arduino_LCD_KeyPad_Shield_(SKU:_DFR0009)
******************************************************/

// select the pins used on the LCD panel
LiquidCrystal lcd(8, 9, 4, 5, 6, 7);

// define some values used by the panel and buttons
int lcd_key     = 0;
int adc_key_in  = 0;
#define btnRIGHT  0
#define btnUP     1
#define btnDOWN   2
#define btnLEFT   3
#define btnSELECT 4
#define btnNONE   5

// read the buttons
int read_LCD_buttons()
{
 adc_key_in = analogRead(0);      // read the value from the sensor
 if (adc_key_in > 1000) return btnNONE; // We make this the 1st option for
speed reasons since it will be the most likely result
 if (adc_key_in < 50)    return btnRIGHT;
 if (adc_key_in < 250)   return btnUP;
 if (adc_key_in < 450)   return btnDOWN;
 if (adc_key_in < 650)   return btnLEFT;
 if (adc_key_in < 1000)   return btnSELECT;

 return btnNONE;  // when all others fail, return this...
}


void setup()
{
 lcd.begin(16, 2);
 lcd.setCursor(0,0);
 lcd.print("Push the buttons");
}

void loop()
{
 lcd.setCursor(9,1);              // move cursor to second line "1" and 9
spaces over
 lcd.print(millis()/1000);       // display seconds elapsed since power-up

 lcd.setCursor(0,1);              // move to the begining of the second line
 lcd_key = read_LCD_buttons();  // read the buttons
```

```
   switch (lcd_key)                   // depending on which button was pushed, we
perform an action
 {
   case btnRIGHT:
     {
       lcd.print("RIGHT ");
       break;
     }
   case btnLEFT:
     {
       lcd.print("LEFT   ");
       break;
     }
   case btnUP:
     {
       lcd.print("UP     ");
       break;
     }
   case btnDOWN:
     {
       lcd.print("DOWN   ");
       break;
     }
   case btnSELECT:
     {
       lcd.print("SELECT");
       break;
     }
   case btnNONE:
     {
       lcd.print("NONE   ");
       break;
     }
 }

 }
```

This code is from the DF Robot wiki page, combining reading the buttons with displaying some text in response.

Stretch Goal #1: Can you come up with a simple game using the buttons and the display? Perhaps start by moving a character around the screen in response to button presses!

Things to think about:
  – When moving your character around, what has to happen in the character's old position?
  – What will you do when you hit the end of a row?
  – What will you do when you run off the top or bottom of the screen?

Fail:

COMM: Ethernet Web & Email Communications (Using Ethernet networking for Web & Email)

What:
The Ethernet shield gives your arduino projects a wired connection to a local network or the full Internet. The board (once connected) can either take a static or fetch a dynamic IP address (from a DHCP server or home router) and function as either a client (e.g. "browser") or a server (e.g. web server, etc) — or even be used to send out emails, SMS/Text messages or even tweets!

The built in Wiz5100 networking chip does a very good job at offloading much of the complexities of running networking connections on the Arduino, but the required libraries do take up quite a bit of memory. So if you will be fine for doing very basic web, email or other basic networking communications, but anything more complex than than (dynamic web content, serving video or lots of images) and you'll want to start looking for a full computer platform with more horsepower, memory and networking support such as the RaspberryPi, Beaglebone or other embedded Linux Operating System.

How:
This module is wired to require pins 13, 12, 11, 10 for Ethernet usage, and pin 4 for SD card use (see Micro-SD card recipe).

This shield uses the Ethernet Library located here:
http://arduino.cc/en/Reference/Ethernet

Connecting and Disconnecting a Shield:
With your arduino unplugged, and on a static free surface, you will first want to carefully connect your Ethernet shield to your arduino, lining up the pins on the end opposite the shiny Ethernet and USB ports (see right, bottom). If your boards have the six ICSP pins, that is a good way of lining everything up (see photo on right) .

Once the shield is mounted, you can connect the Arduino/Laptop USB cable, and then the Ethernet cable. (photo at top of page).

To remove a shield, DO NOT just "pull it off"; you will bend our pins! Instead, gently half-way pry one side lose, then half way pry the other. Then gently lift off the shield and place it in a protective container.
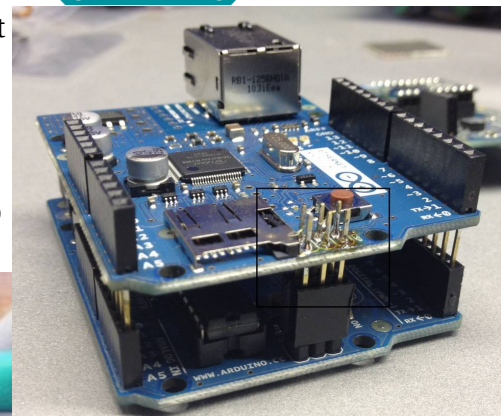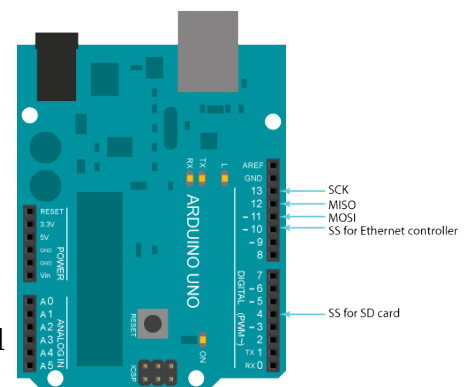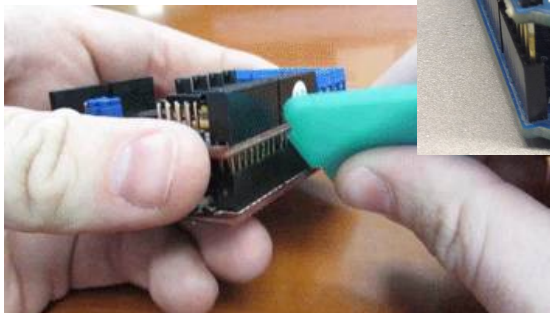
Getting an IP Address:

The code below can be typed in, or loaded by using the example code Examples / Ethernet / DhcpAddressPrinter :

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

// Change at least the last hex number below to any two digit value so that
// you have a unique address on your class network. Otherwise, there will
// be MAC address conflicts and you and your neighbors will have problems.
byte mac[] = {
  0x00, 0xAA, 0xBB, 0xCC, 0xDE, 0x02 };

// Initialize the Ethernet server library
// with using DHCP for your IP (see serial monitor) and port you wish
// to use (port 80 is default for HTTP):
EthernetServer server(80);

void setup() {
 // Open serial communications and wait for port to open:
  Serial.begin(9600);
  while (!Serial) {
   ; // wait for serial port to connect. Needed for Leonardo only

    // start the Ethernet connection:
    if (Ethernet.begin(mac) == 0) {
      Serial.println("Failed to configure Ethernet using DHCP");
      // no point in carrying on, so do nothing forevermore:
      for(;;)
       ;
    }
   }

  // start the Ethernet connection and the server:
  Ethernet.begin(mac);
  server.begin();
  Serial.print("Arduino server is at ");
  Serial.println(Ethernet.localIP());
}

// Main execution loop
void loop() {
  Serial.print(".");     // Serial monitor heartbeat
  delay(1000);           // Simple 1 second do nothing
}
```

Quick Connectivity Test:
After compiling and uploading this code, click on
the serial monitor icon ![icon], and copy or write
down your IP in the serial console window (a four
part number like 10.4.2.189).

```
/dev/ttyUSB0                                    _ □ ✕
                                          [Send]
server is at 10.4.20.189




☑ Autoscroll          No line ending ▾   9600 baud ▾
```

If you are able to list your IP (above), then hook
your laptop up to the same wired network, and try
pinging your arduino like this:

```
ping -c 3 10.4.20.189        # Pings arduino three times (on Win use -n)
PING 10.4.20.171 (10.4.20.171) 56(84) bytes of data.
64 bytes from 10.4.20.171: icmp_seq=1 ttl=128 time=0.235 ms
64 bytes from 10.4.20.171: icmp_seq=2 ttl=128 time=0.252 ms
64 bytes from 10.4.20.171: icmp_seq=3 ttl=128 time=0.221 ms

--- 10.4.20.171 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2000ms
rtt min/avg/max/mdev = 0.221/0.236/0.252/0.012 ms
```

Answer Questions and Have TA Sign Off

Q: Did your ping test work? Yes/No: **TA Sign-Off:**

Q: If so, what is your IP address? A: **TA Sign-Off:**

If that worked, then your Arduino is officially on the network and you can now continue to to set it up
as a little web server, email or twitter client. If problems, see the **Fail** section below.

Making a Mini Web Server:

Once you have the ping/connectivity test above working, you're ready to start serving web content or real time input sensor data. First, we need to replace the entire main loop to 1) detect client connections, 2) write out the HTTP web headers, 3) print some basic web content (test text). See below:

```
// Main execution loop
void loop() {
  // listen for incoming clients
  EthernetClient client = server.available();
  // If a client has connected, do this
  if (client) {
    Serial.println("new client");
    // an http request ends with a blank line
    boolean currentLineIsBlank = true;
    while (client.connected()) {
      if (client.available()) {
        char c = client.read();
        Serial.write(c);
        // if you've gotten to the end of the line (received a newline
        // character) and the line is blank, the http request has ended,
        // so you can send a reply
        if (c == '\n' && currentLineIsBlank) {
          // send a standard http response header
          client.println("HTTP/1.1 200 OK");
          client.println("Content-Type: text/html");
          client.println("Connection: close");  // the connection will be closed after gets response
          client.println("Refresh: 5");  // refresh the page automatically every 5 sec
          client.println();
          client.println("<!DOCTYPE HTML>");
          client.println("<html>");                 // Opens HTML content
          client.println("<pre>");                  // Preformatted text block
          client.println("TEST... SUCCESS!!!");     // Actually printed to browser screen
          client.println("</pre>");
          client.println("</html>");                // closes out html web page
          break;
        }
        if (c == '\n') {
          // you're starting a new line
          currentLineIsBlank = true;
        }
        else if (c != '\r') {
          // you've gotten a character on the current line
          currentLineIsBlank = false;
        }
      }
    }
    // give the web browser time to receive the data
    delay(10);
    // close the connection:
    client.stop();
    Serial.println("client disconnected");
  }
}
```

Congratulations! If you have **TEST... SUCCESS!!!** on your browser screen, then your Arduino is now serving static web content.

Show Instructor or TA and Get Sign Off

Q: Did you get your arduino to display the test web page? Yes/No:_____   Inst:_____ _

If you had problems in this section, check out some tips in the **Fail:** section at the end of this recipe.

Displaying Analog Readings:

After you get the basic web content bring served from your arduino, next try serving out some analog voltages values from the six analog inputs 0 – 5. Find the "TEST... SUCCESS!!!" line in the HTML content code area of your current code and replace the whole line with this code below

```
client.println("================================");
client.println("=== Tweeks Arduino Webserver ===");
client.println("================================");
// output the value of each analog input pin
for (int analogChannel = 0; analogChannel < 6; analogChannel++) {
  int sensorReading = analogRead(analogChannel);
  client.print("analog input ");
  client.print(analogChannel);
  client.print(" is ");
  // Adjust the for loop, adjust "n" of  x<sensorReading/n to
  // scale the 1023 bar graph (play with it)
  int x=0;
  for (x=0 ; x<sensorReading/15 ; x=x+1) {
    client.print("=");    // builds your ASCII TXT bar graph
  }
  client.print(sensorReading);  // prints the actual value
  client.print("<br />");
}
client.println("</pre>");
client.println("</html>");       // closes out html web page
```

If you got it working, your web browser should be displaying real time graphs of your six analog input readings like this:

```
================================
=== Tweeks Arduino Webserver ===
================================
analog input 0 is ====================310
analog input 1 is =====================328
analog input 2 is ====================313
analog input 3 is =======================346
analog input 4 is ====================301
analog input 5 is ====================314
```

Display Real World Values on The Web:

Want to read a real value? Look at the "**What**" sections of the **INPUT: Potentiometer Recipe** to hook up a potentiometer (knob) to Analog input A0. Look at the "What" and "How" sections of **INPUT: Light Sensor** to measure the amount of ambient light in the room.

Q: Did you get your pot or light sensors to display on the web? One / Both   Inst: _____

> **Stretch Goal #1:** Can you make your pot (knob) value print out to be "loud" "medium" or "soft", or your light sensor value print out to be "lights on" or "lights off"?

> **Stretch Goal #2:** Want cooler looking graphs and text windows? Check out the HTML codes you can use in place of the "=" signs here (hint, use the &code; format to client.print() : http://www.theasciicode.com.ar/extended-ascii-code/bottom-half-block-ascii-code-220.html

Email is still being tested, but here's the code you can start playing with if email is your desire:

To use it for the Nov 20th 2014 workshop, see the guest instructor Steve Swenson for a guest SMTP username, password for relaying (or sending through) email out of his SMTP accounts. For this November workshop, also:

- Change line 13 to smtp.emailsrvr.com, keep the port at 587.
- Change line 73 to add your base64 encoded email address, so it should look like:
  ```
  "AUTH login –insert-your-base64-username-here–"
  ```
- Change line 76 to contain your base64 encoded password
- Change line 79 and 89 to the from address (-username-@rackeddit.net)
- Change line 82 and 90 to the TO address, e.g. me@gmail.com
- Everyone must chose a unique MAC address on line 107
- Suspect the function on line 38 isn't used. You can cut it to save some bytes.
- If you need to base64 encode anything else, you can use this fiddle:
  http://jsfiddle.net/pb4usf47/

Here's the code:
(for line numbers, reference the github code here:
https://gist.github.com/ctigeek/a6ac72253ddd588e9b35#file-emailtest-ino

```
// Arduino script for sending an email by @ctigeek.

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

EthernetClient  client;
signed long next;

int connectToServer() {
   Serial.println(F("Connecting to server..."));

   if (!client.connect("smtp.server.com",587))
     return 0;

   Serial.println(F("connected to mail server."));
   return 1;
}

int waitForReply() {
  next = millis() + 5000;
  while(client.available()==0)
  {
    if (next – millis() < 0)
      return 0;
  }
  int size;
  while((size = client.available()) > 0)
  {
    uint8_t* msg = (uint8_t*)malloc(size);
    size = client.read(msg,size);
    Serial.write(msg,size);
    free(msg);
```

```
  }
  return 1;
}

int sendAndWaitForReply(char* txtToSend) {
    next = millis() + 5000;
    client.println(txtToSend);
    Serial.println(txtToSend);
    if (!waitForReply())
        return 0;

    return 1;
}

void justSend(const __FlashStringHelper* txtToSend) {
    client.println(txtToSend);
    Serial.println(txtToSend);
}

int sendAndWaitForReply(const __FlashStringHelper* txtToSend) {
    next = millis() + 5000;
    client.println(txtToSend);
    Serial.println(txtToSend);
    if (!waitForReply())
        return 0;

    return 1;
}

int sendEmail() {
  if (!connectToServer())
    return 0;

  if (!waitForReply())
    return 0;

  if (!sendAndWaitForReply(F("EHLO Arduino")))
    return 0;

  if (!sendAndWaitForReply(F("AUTH login "))) //enter base64 encoded email
address here
    return 0;

  if (!sendAndWaitForReply(F(""))) //enter base64 encoded password here
    return 0;

  if (!sendAndWaitForReply(F("MAIL FROM:<your email address here>")))
    return 0;

  if (!sendAndWaitForReply(F("RCPT TO:<recipients email address here.>")))
    return 0;

  if (!sendAndWaitForReply(F("DATA")))
    return 0;

  justSend(F("MIME-Version: 1.0"));
  justSend(F("From: sender@domain.com"));
```

```
      justSend(F("To: recipient@domain.com"));
      justSend(F("Subject: Arduino!"));
      justSend(F("Content-type: text/plain; charset=us-ascii"));
      justSend(F("")); //this separates the headers from the body.

      justSend(F("Arduinos are so small!"));

      if (!sendAndWaitForReply(F("."))) //this means EOF to the smtp server.
        return 0;
      if (!sendAndWaitForReply(F("QUIT")))
        return 0;

      return 1;
    }

    void setup() {
      Serial.begin(9600);
      uint8_t mac[6] = {0x00,0x01,0x02,0x03,0x04,0x05};
      Ethernet.begin(mac);

      Serial.print(F("localIP: "));
      Serial.println(Ethernet.localIP());

      sendEmail();
    }

    void loop() {

    }
```

Fail:
For the Web portions of this recipe:
- If Failed the "Quick Connectivity Test", then:
  - Check that ethernet cable is connected and you have a link light on the arduino and the switch
  - Check that the lights on your ethernet board are properly lit (shield is properly hooked up).
  - Did you GET an IP address printed to the serial console?
  - Is your laptop hooked to the same wired network?
  - Check for error messages during the build (indicating typos). CODE IS CAPS SENSITIVE!
  - Look at how your neighbor got it working or ask the TA for help

- If Failed the "Making a Mini Web Server" "TEST.. SUCCCESS" (but passed the ping test):
  - PEBKAC... Check for typos.
- If Failed the "Displaying Analog Readings" section, then:
  - PEBKAC... Check for typos

- If you can't find typos, but it isn't working, copy/paste the example code from github, or the student thumb drive (if the instructor has code locally).

Future Recipes & Modules to Add

Arduino Coding/Theory
       programming sections
       reference commands
       coding best practices
IR Remote Control (w/dedicated thin remote)
Bluetooth Remote Control (w/phones)

Future Eletronics Lessons:
Using a multi-meter
Light and external LEDs
       20ma limits
       resistor
       pin modes
       dim light means port wasn't set to output
       pwm
Drive a motor / transistor
       20ma limits
       biasing a transistor
       pin modes
       pwm
Resistors and Ohms law recipe
soldering skills

far future
GPS
9 DOF GYRO
USB hids devices