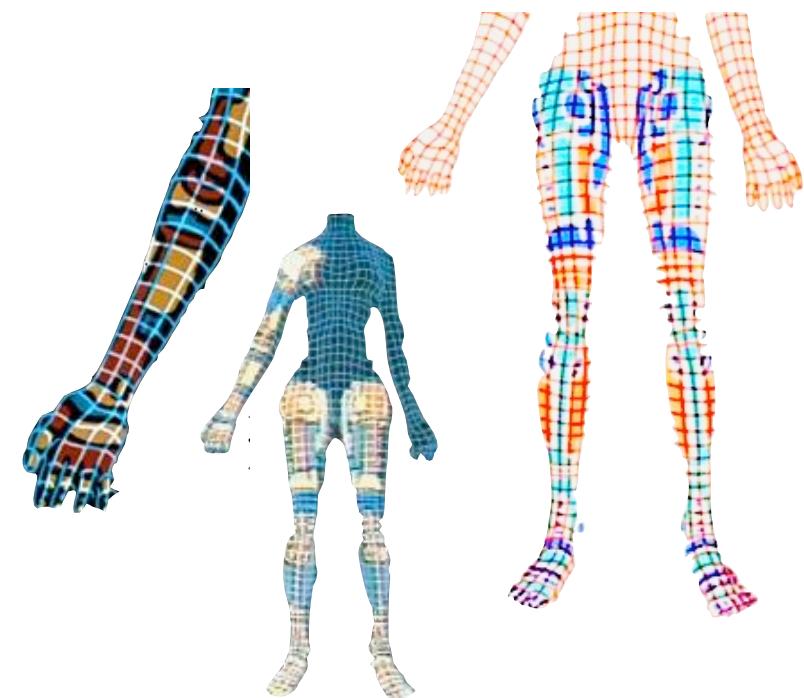


Bionic Arduino

Introduction to Microcontrollers with Arduino

Class 2



What's for Today

- Random Behavior
- RGB LEDs
- Color mixing
- Analog input with variable resistors
- Potentiometers & photocells
- Basic serial input & output
- Playing sound with piezo buzzers

This is a lot of stuff, let's see how far we get.

Recap: Blinky LED

Make sure things still work

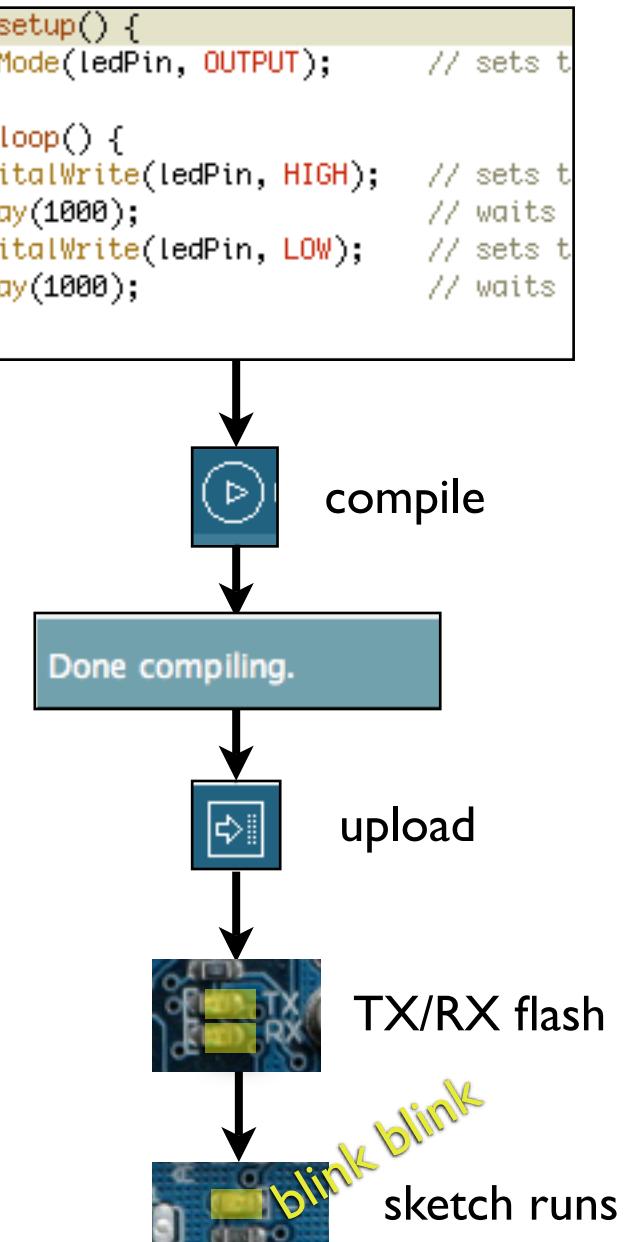
```
int ledPin = 13; // LED connected to digital pin 13

void setup()
{
  pinMode(ledPin, OUTPUT); // sets the digital pin as output
}

void loop()
{
  digitalWrite(ledPin, HIGH); // sets the LED on
  delay(1000); // waits for a second
  digitalWrite(ledPin, LOW); // sets the LED off
  delay(1000); // waits for a second
}
```

```
void setup() {
  pinMode(ledPin, OUTPUT); // sets the digital pin as output
}
void loop() {
  digitalWrite(ledPin, HIGH); // sets the LED on
  delay(1000); // waits for a second
  digitalWrite(ledPin, LOW); // sets the LED off
  delay(1000); // waits for a second
}
```

Load “File/Sketchbook/Examples/Digital/Blink”



Change the “delay()” values to change blink rate

Known Good Configuration

Rule #1 of experimenting:

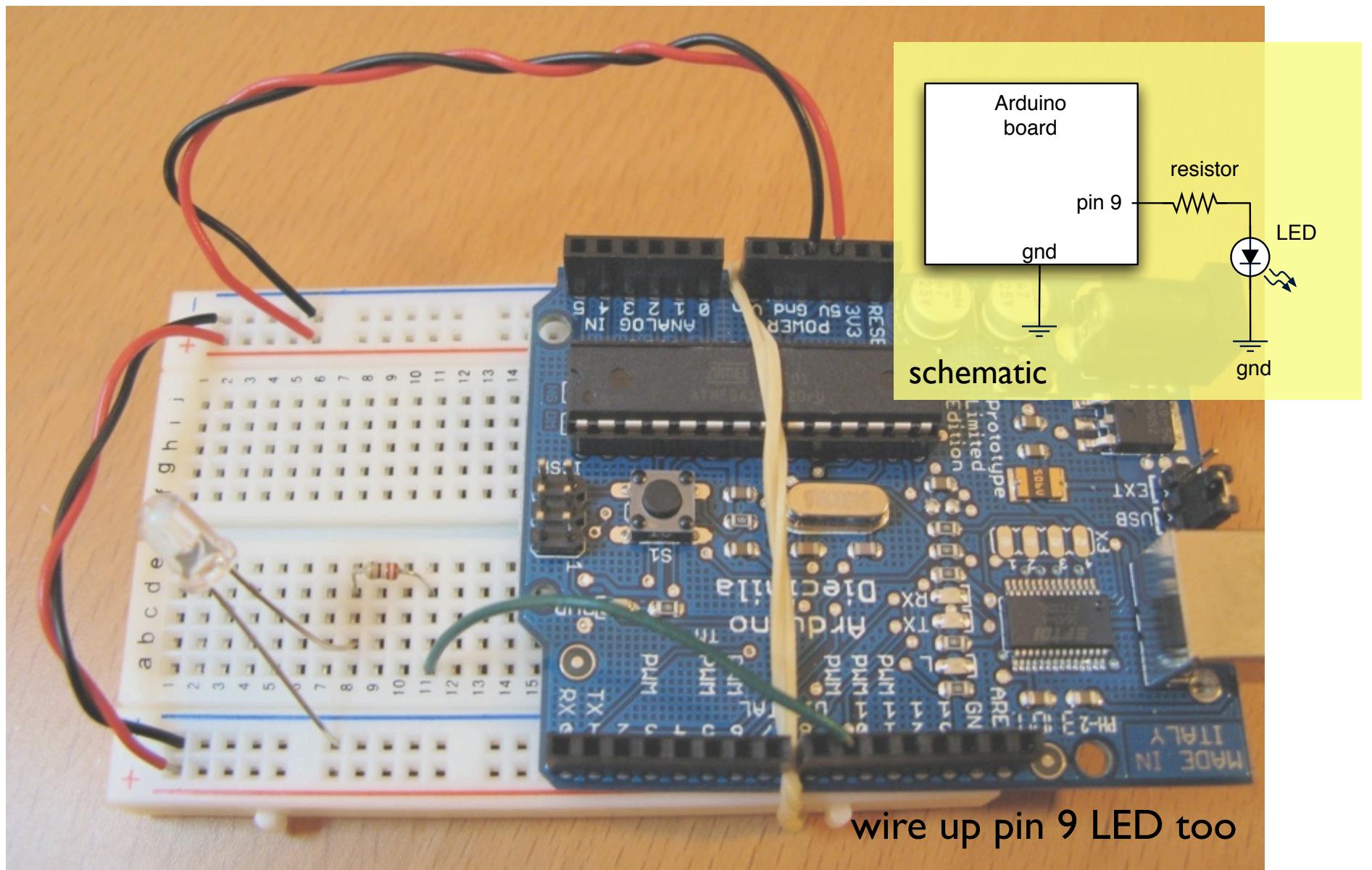
Before trying anything new,

Get back to a known working state

So spend a few minutes & get “Blink” working again

Get your entire edit->compile->upload->run working
Even if it becomes so second nature to you that you feel you shouldn't need to, do it anyway.
Especially when mysterious problems arise, revert to a known state

Getting the Board Set Up

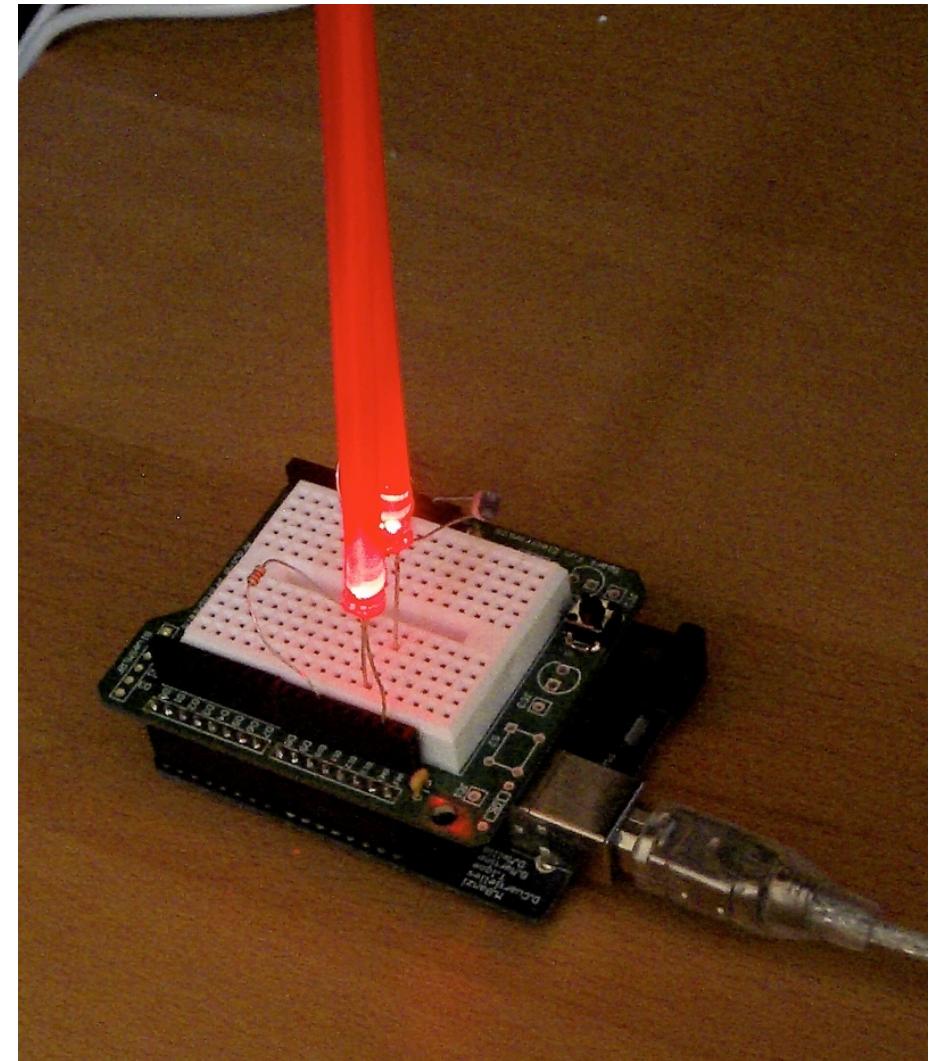


Questions / Review

Any questions, comments, or problems?

Aside: LED Light Tubes

Snug-fit straws on
the end of your
LEDs to make
them glow more
visibly



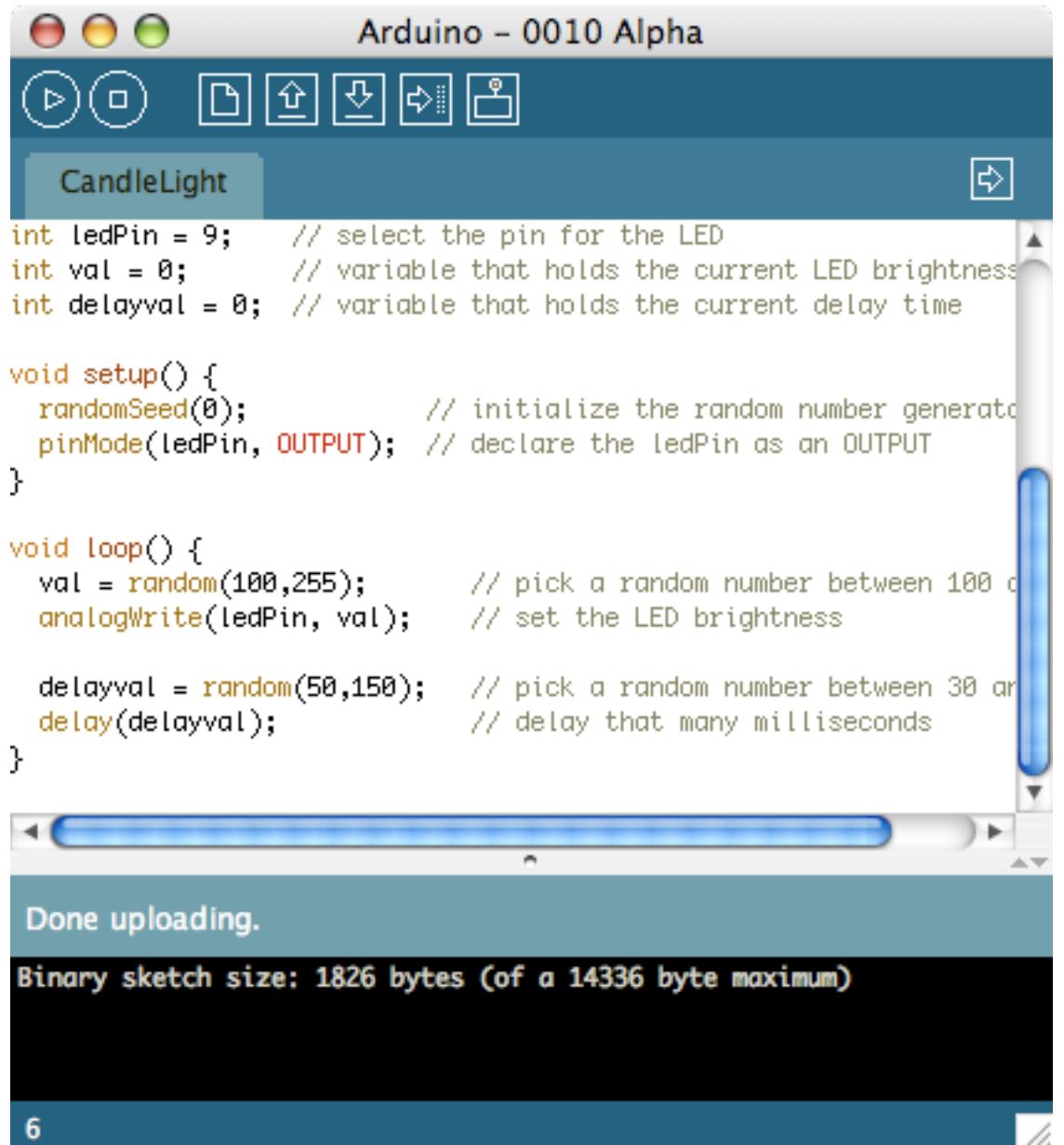
I have a box of multi-colored straws for whatever color LED you like

Random Behavior

“CandleLight”

Uses simple
pseudo random
number generator
to mimic flame

Use `random(min,max)`
to pick a number between
min & max.



The screenshot shows the Arduino IDE interface with the title bar "Arduino - 0010 Alpha". Below the title bar is a toolbar with various icons. The main area displays a sketch named "CandleLight". The code is as follows:

```
int ledPin = 9; // select the pin for the LED
int val = 0; // variable that holds the current LED brightness
int delayval = 0; // variable that holds the current delay time

void setup() {
    randomSeed(0); // initialize the random number generator
    pinMode(ledPin, OUTPUT); // declare the ledPin as an OUTPUT
}

void loop() {
    val = random(100,255); // pick a random number between 100 and 255
    analogWrite(ledPin, val); // set the LED brightness

    delayval = random(50,150); // pick a random number between 50 and 150
    delay(delayval); // delay that many milliseconds
}
```

At the bottom of the IDE window, a message says "Done uploading." and "Binary sketch size: 1826 bytes (of a 14336 byte maximum)".

This sketch is in the handout.

Can also use random numbers to make random decisions.

Note: not truly random, but good enough for most purposes.

Analog Input

To computers, analog is chunky

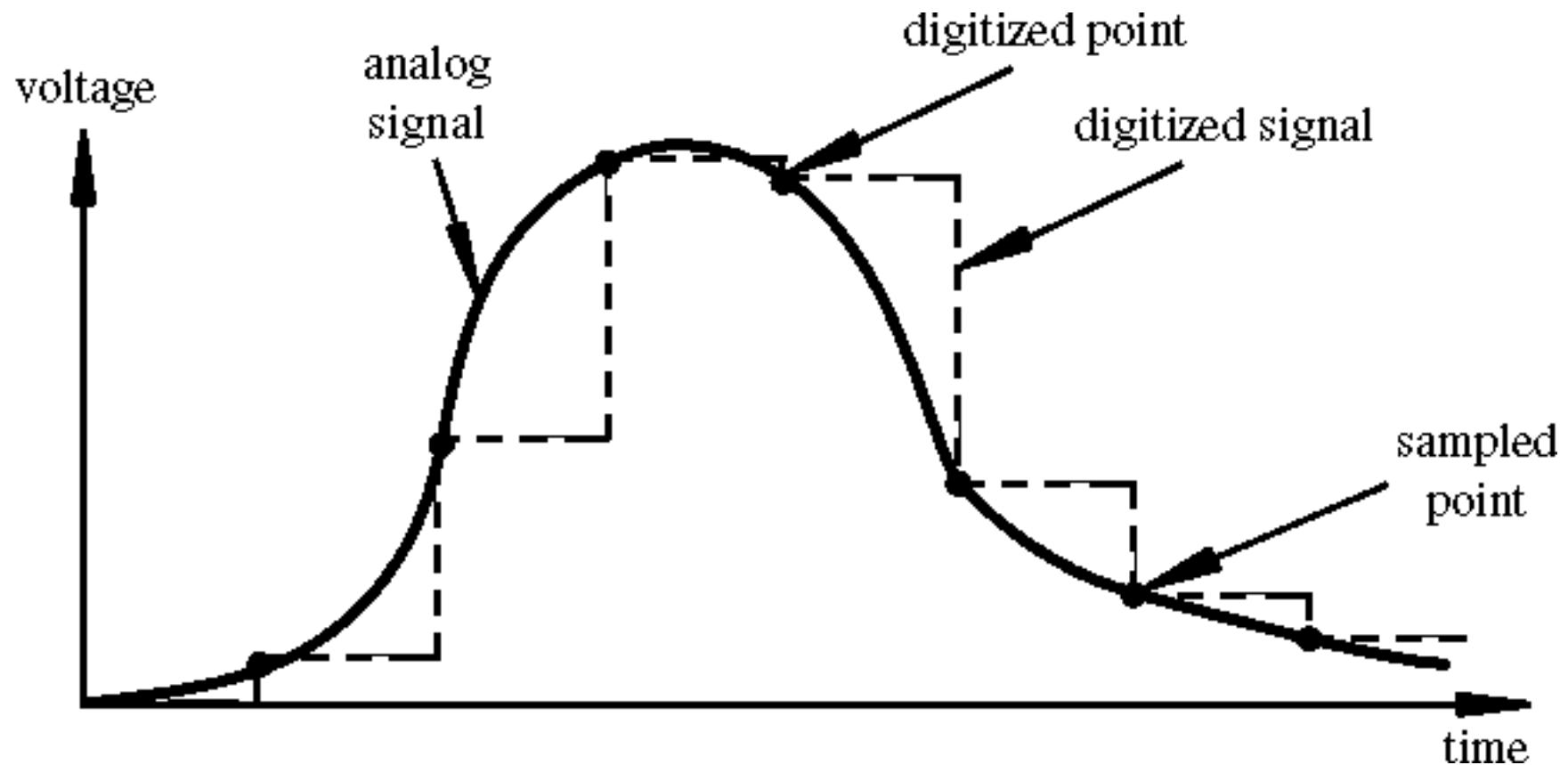
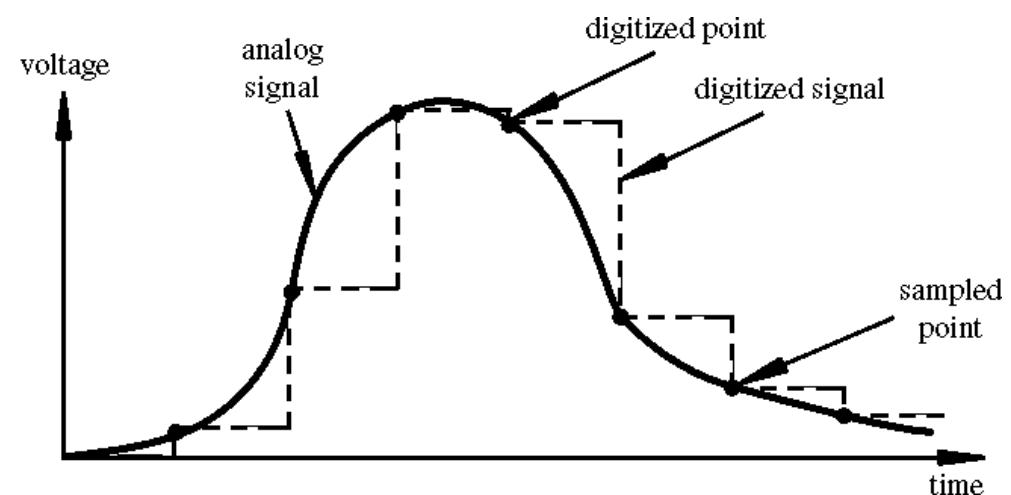


image from: <http://www.engr.colostate.edu/~dga/me307/lectures.html>

Analog Input

- Many states, not just two (HIGH/LOW)
- Number of states (or values, or “bins”) is *resolution*
- Common computer resolutions:
 - 8-bit = 256 values
 - 16-bit = 65,536 values
 - 32-bit = 4,294,967,296 values



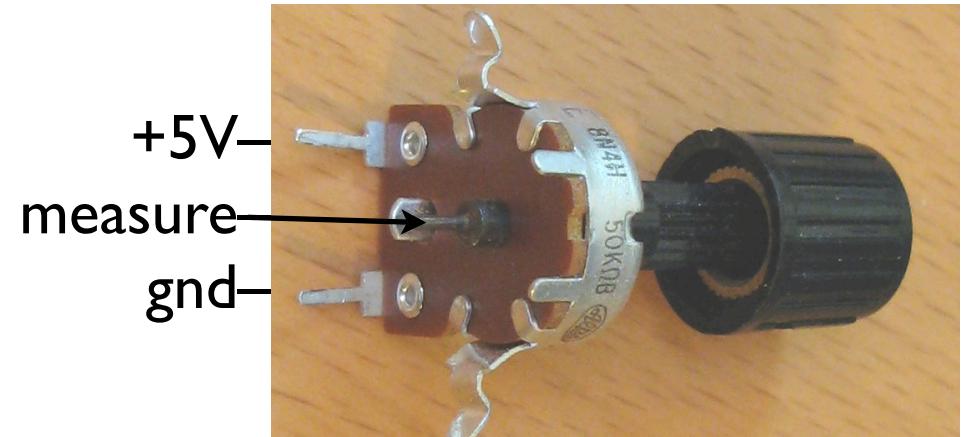
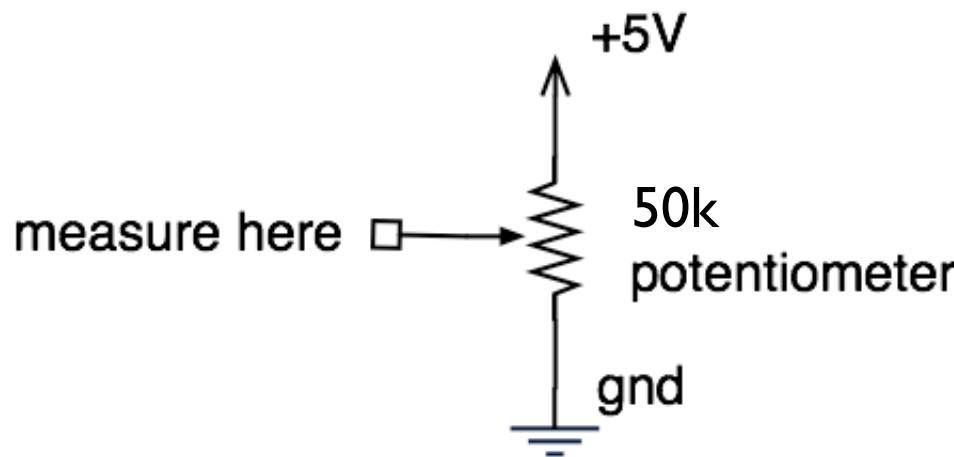
Analog Input

- Arduino (ATmega168) has six ADC inputs
- (ADC = Analog to Digital Converter)
- Reads voltage between 0 to 5 volts
- Resolution is 10-bit (1024 values)
- In other words, $5/1024 = 4.8 \text{ mV}$ smallest voltage change you can measure

Analog Input

Sure sure, but how to make a varying voltage?

With a *potentiometer*. Or just *pot*.



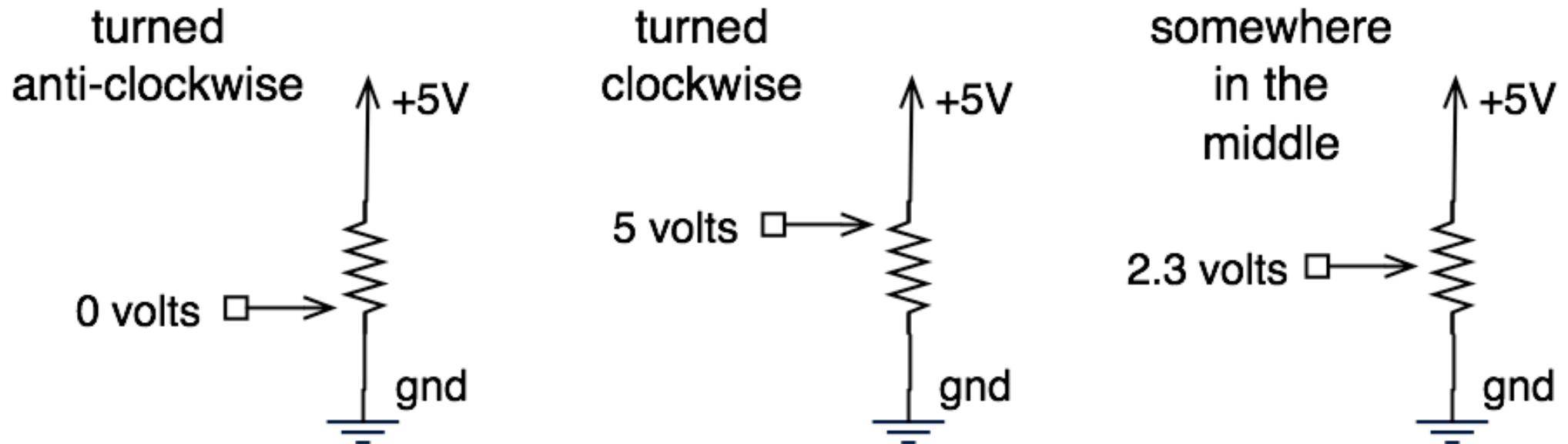
The pot you have



pots also look like this

Potentiometers

Moving the knob is like moving where the arrow taps the voltage on the resistor



When a resistor goes across a voltage difference, like +5V to Gnd, the voltage measured at any point along a resistor's length is proportional to the distance from one side.

If you take apart a pot, there's a little wiper just like in the schematic symbol. But I might have the directions reversed (clockwise vs. anti-clockwise).

What good are pots?

- Anytime you need a ranged input
 - (we're used to knobs)
- Measure rotational position
 - steering wheel, robotic joint, etc.
- But more importantly for us, potentiometers are a good example of a *resistive sensor*

There are many kinds of resistive sensors

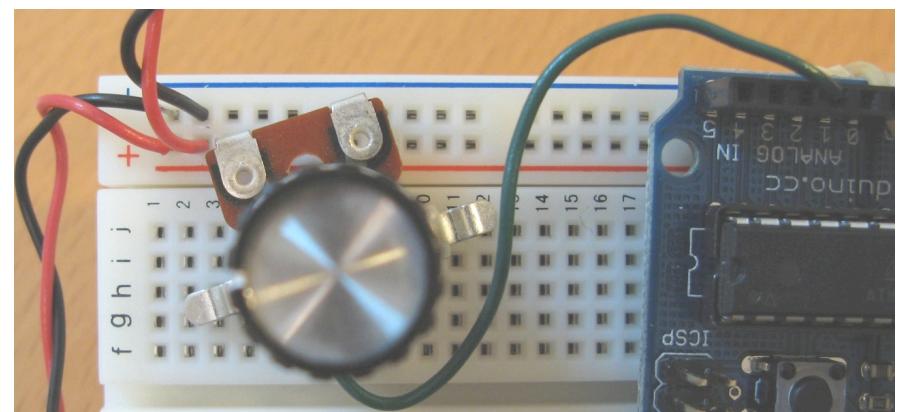
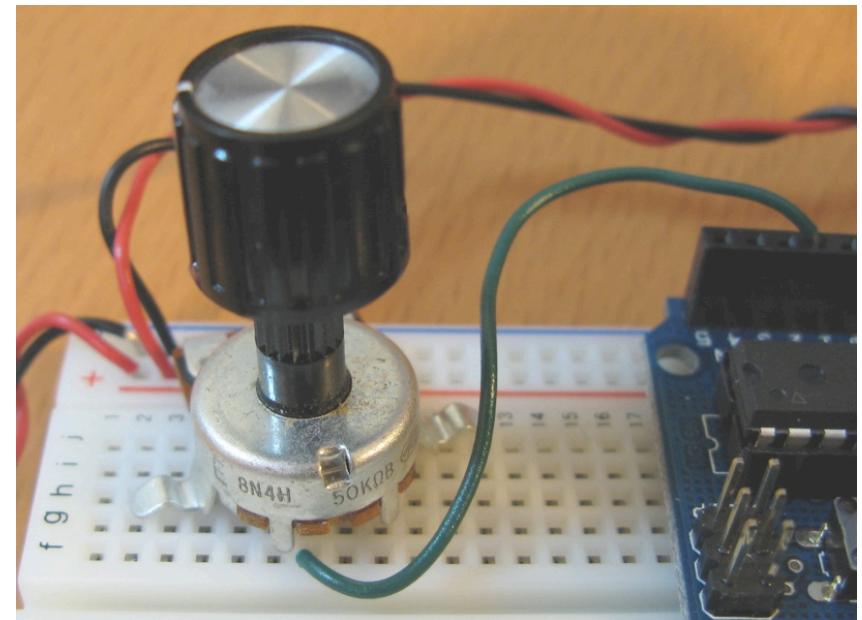
Arduino Analog Input

Plug pot directly into breadboard

Two “legs” plug into +5V & Gnd
(red + & blue -) buses

Middle “post” plugs into a row
(row 7 here)

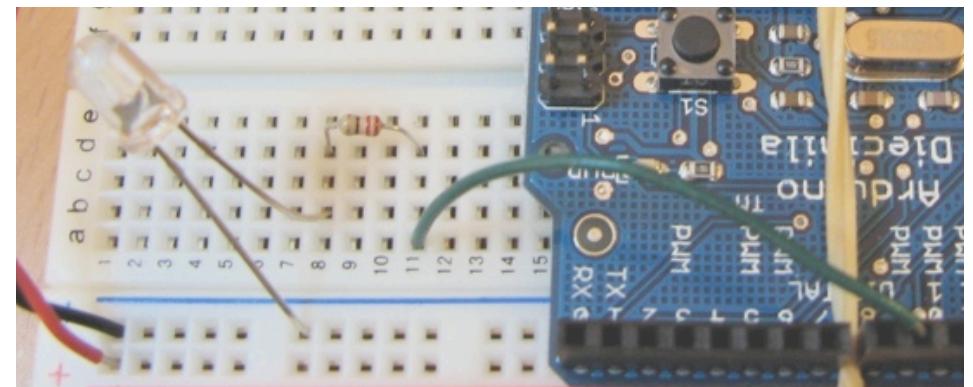
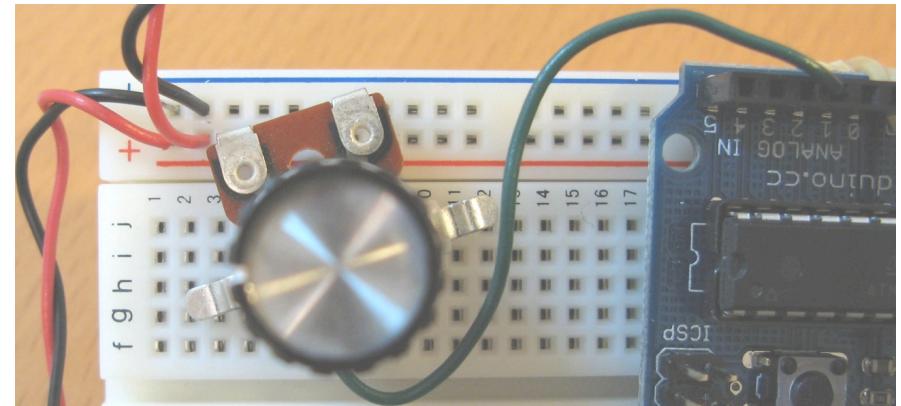
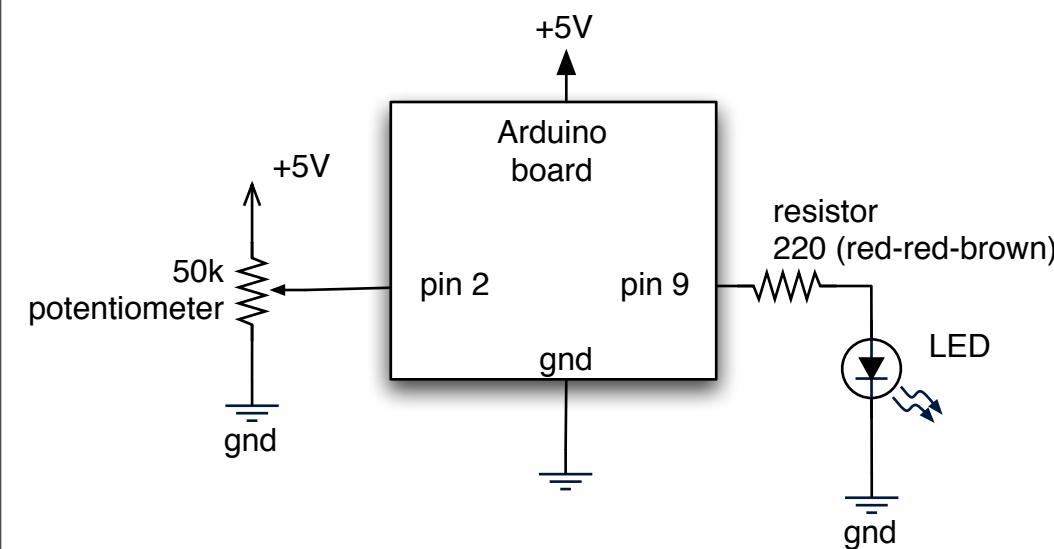
Run a wire from that row to
Analog In 2



Why are we using Analog In 2? Because it's in the middle. There's no reason, any of the 6 analog inputs would work the same.

Pot & LED Circuit

This is what your board should have on it now



In schematics, inputs are usually on the left, outputs on the right
Also, more positive voltages are on the top, more negative on the bottom

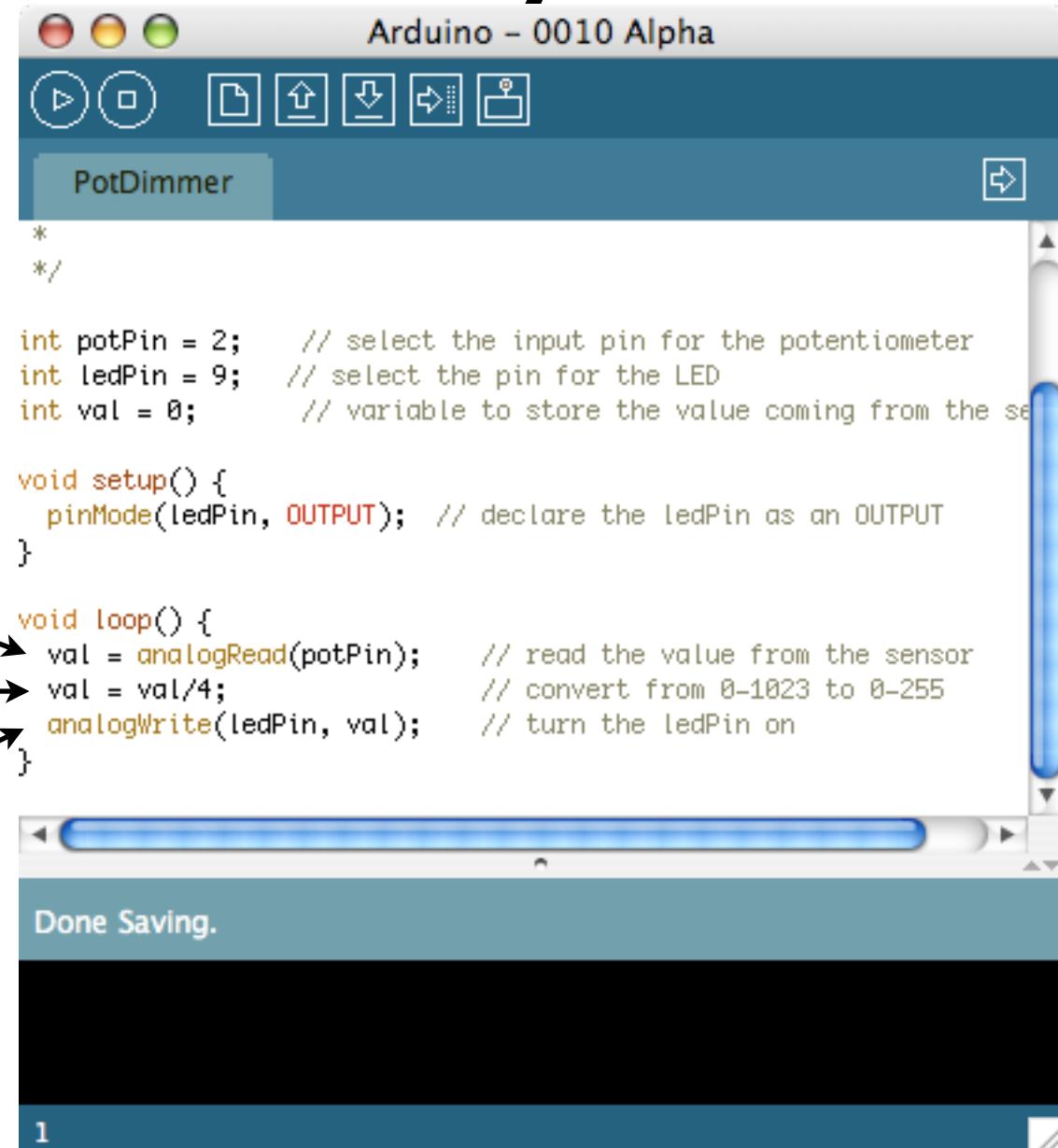
Varying Brightness by Hand

“PotDimmer”

Turn the knob to
change LED
brightness

input
process the
input data
output

Most all embedded
systems have a
 $\text{input} \rightarrow \text{process} \rightarrow \text{output}$
loop



The screenshot shows the Arduino IDE interface with the title bar "Arduino - 0010 Alpha". Below the title bar are standard IDE icons: play, stop, upload, download, and others. The main workspace contains the following code:

```
int potPin = 2;      // select the input pin for the potentiometer
int ledPin = 9;      // select the pin for the LED
int val = 0;          // variable to store the value coming from the sensor

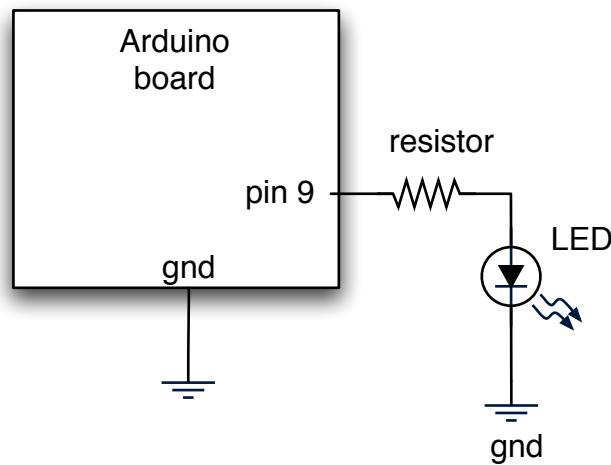
void setup() {
    pinMode(ledPin, OUTPUT); // declare the ledPin as an OUTPUT
}

void loop() {
    val = analogRead(potPin); // read the value from the sensor
    val = val/4;              // convert from 0-1023 to 0-255
    analogWrite(ledPin, val); // turn the ledPin on
}
```

Below the code, a message box displays "Done Saving." and the number "1" at the bottom.

Sketch available in handout

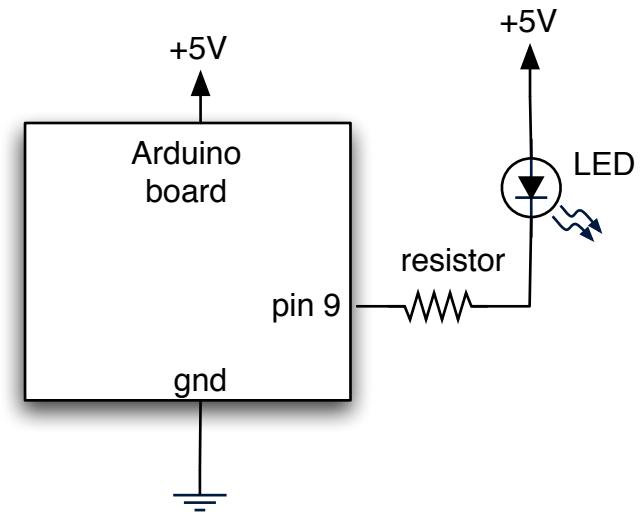
Two Ways to Hook up LEDs



To turn ON: `digitalWrite(9,HIGH)`

To turn OFF: `digitalWrite(9,LOW)`

To set brightness: `analogWrite(9,val)`



To turn ON: `digitalWrite(9,LOW)`

To turn OFF: `digitalWrite(9,HIGH)`

To set brightness: `analogWrite(9,255-val)`

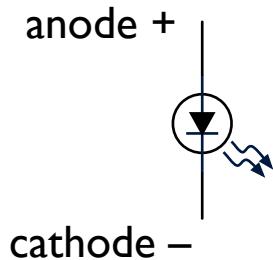
We've been using the one on the left because it makes more sense.

But you'll see the method on the right as well.

The reason for this is that some circuits can switch to Gnd better than they can switch to +5V.

RGB LEDs

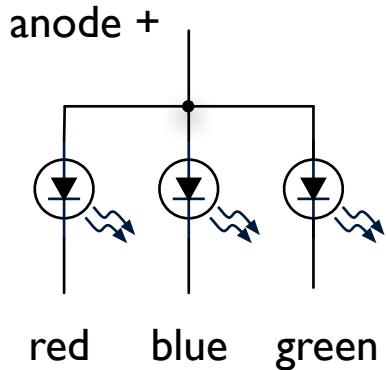
Normal LED



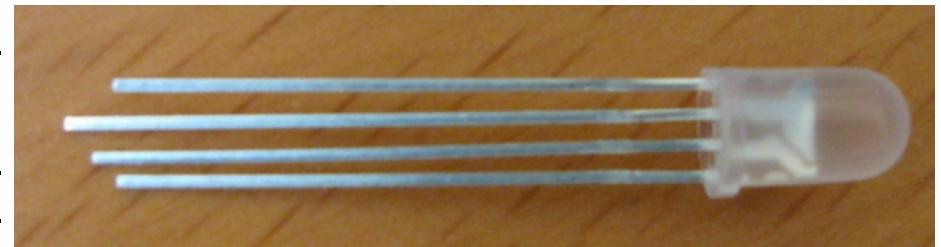
anode +
cathode -



RGB LED



red cathode -
anode +
blue cathode -
green cathode -



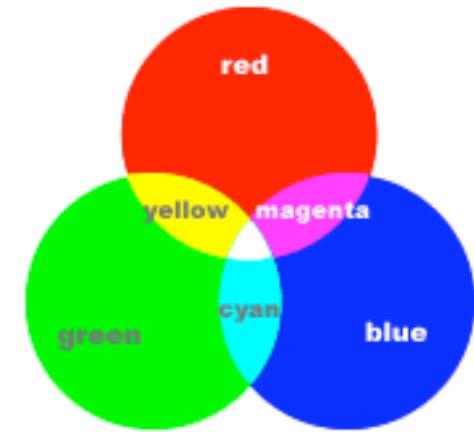
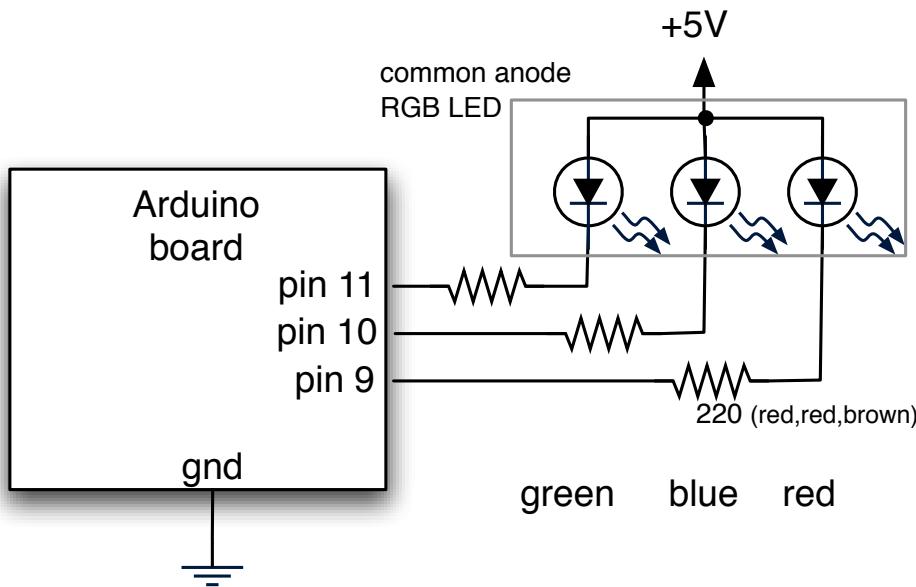
actually 3 LEDs in one package

RGB LED, aka “tri-color LED”

Common-anode RGB LEDs are much more available than common-cathode.
This is why we’re changing around the logic.

Color Mixing

With just 3 LEDs you can make any* color



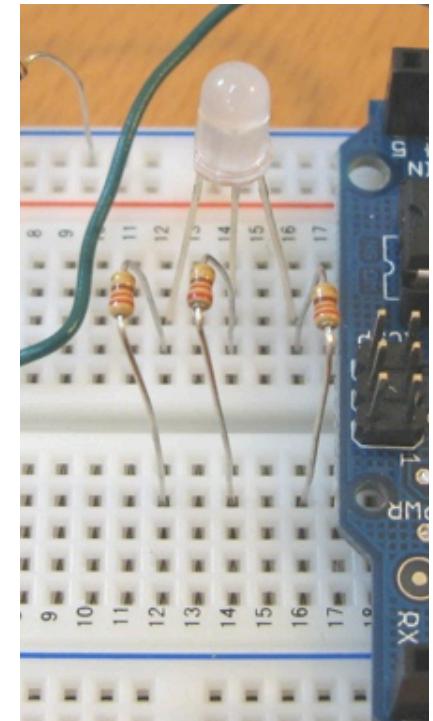
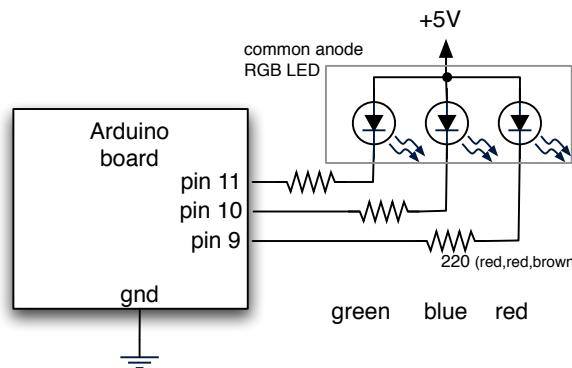
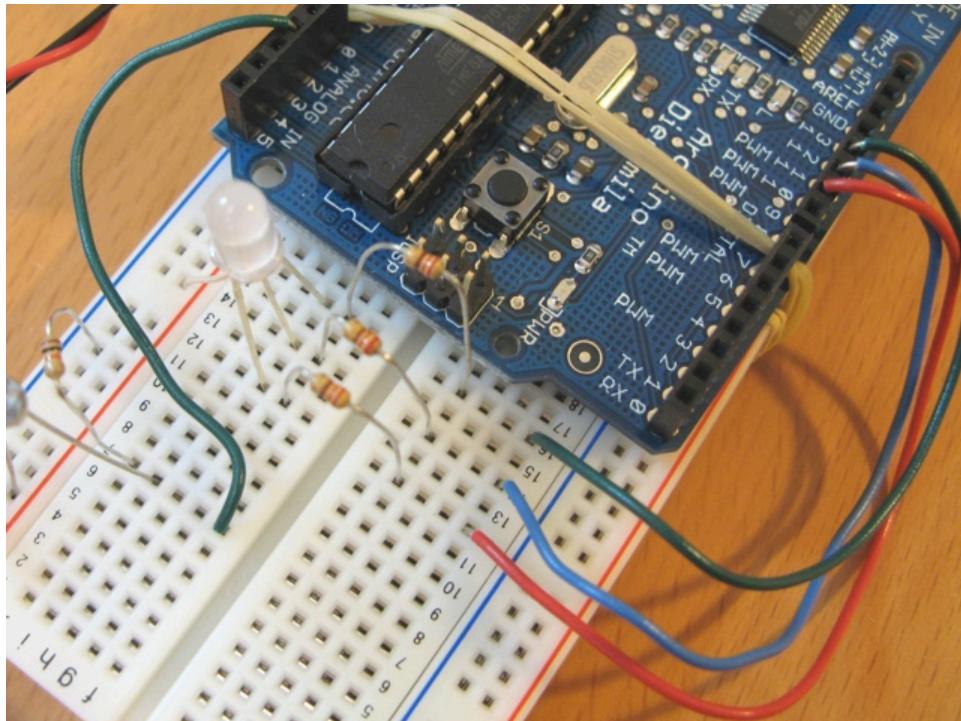
With RGB you can
make any color
(except black)

Mixing light is the additive color model
(paint is subtractive color, and can give you brown)

*besides the additive/subtractive color different, it's hard to get the mix to be just right for a variety of annoying reasons:

- the physics of LEDs mean that different color LEDs put out different amounts of light
- our eyes respond non-linearly across the spectrum, i.e. we're more sensitive to green than red
- the lenses in most RGB LEDs don't focus each color to the same spot

Laying out RGB LED Circuit



slightly bend the longest lead and plug it into the +5v (red) bus
plug remaining leads into rows (12,14,&16 here)
connect 220 (red-red-brown) resistors across middle to matching rows
run wires from resistors to pins 9,10,11 of Arduino, can color-code if you want

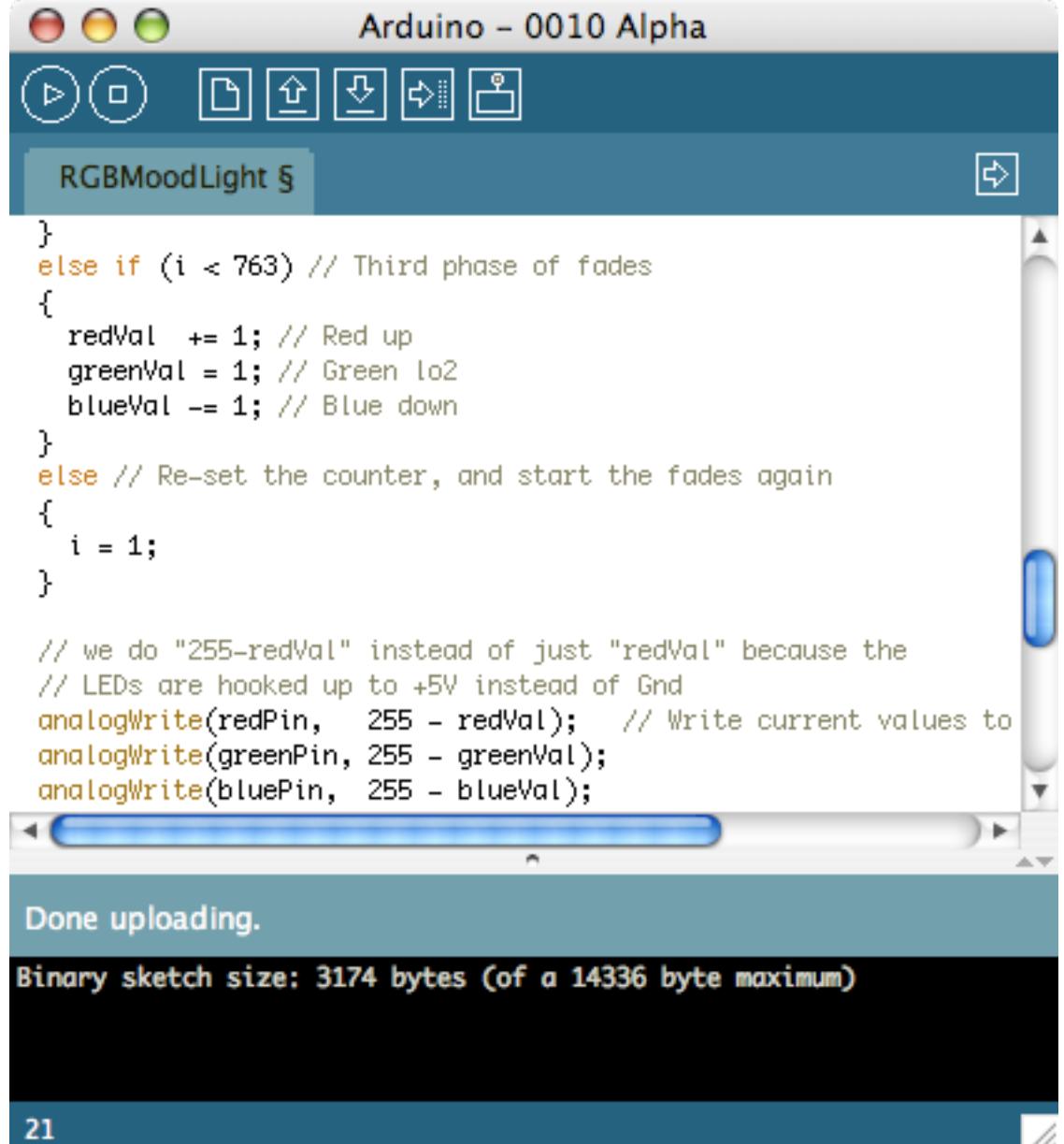
Ignore the green wire in the pictures, that's another circuit.
Keep the pot from last circuit if you can.

RGB Color Fading

“RGBMoodLight”

Slow color fading
and mixing

Also outputs the current
color values to the serial port



The screenshot shows the Arduino IDE interface. The title bar says "Arduino - 0010 Alpha". Below it are standard toolbar icons. The central area shows the code for the "RGBMoodLight" sketch. The code uses a loop to fade colors, with comments explaining the logic for three phases of fading (red up, green up, blue down) and a section to reset the counter and start again. It also includes code to write the current values to three analog pins (redPin, greenPin, bluePin). A progress bar at the bottom indicates the upload is complete. The status message "Done uploading." is displayed, along with the binary sketch size information: "Binary sketch size: 3174 bytes (of a 14336 byte maximum)".

```
        }
    else if (i < 763) // Third phase of fades
    {
        redVal += 1; // Red up
        greenVal = 1; // Green lo2
        blueVal -= 1; // Blue down
    }
    else // Re-set the counter, and start the fades again
    {
        i = 1;
    }

    // we do "255-redVal" instead of just "redVal" because the
    // LEDs are hooked up to +5V instead of Gnd
    analogWrite(redPin, 255 - redVal); // Write current values to
    analogWrite(greenPin, 255 - greenVal);
    analogWrite(bluePin, 255 - blueVal);

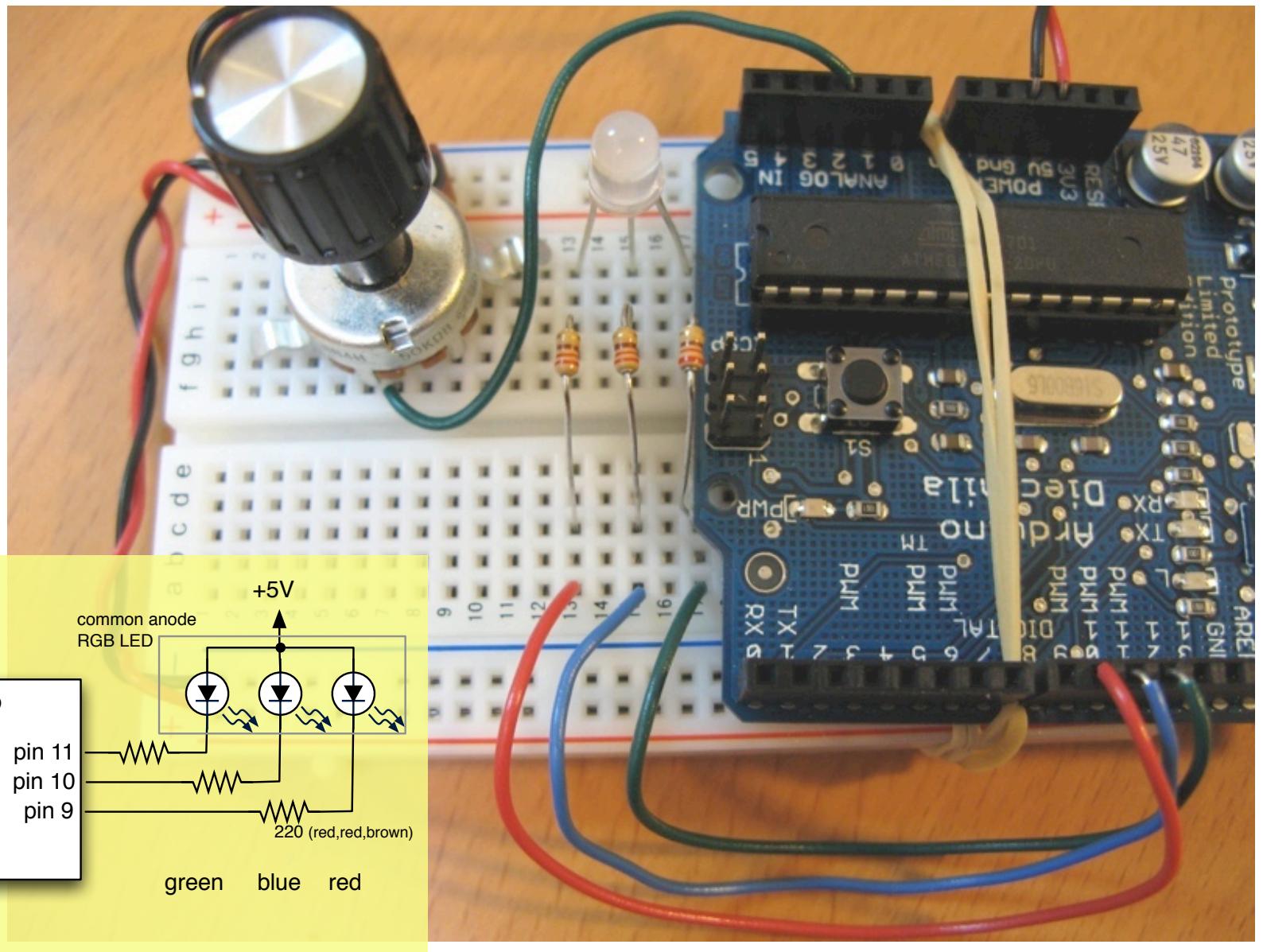
```

21

This sketch is located in the handout.
We'll get to the serial port stuff in a minute.

It just ramps up and down the red,green,& blue color values and writes them with analogWrite()
from <http://www.arduino.cc/en/Tutorial/DimmingLEDs>

Pot-controlled RGB

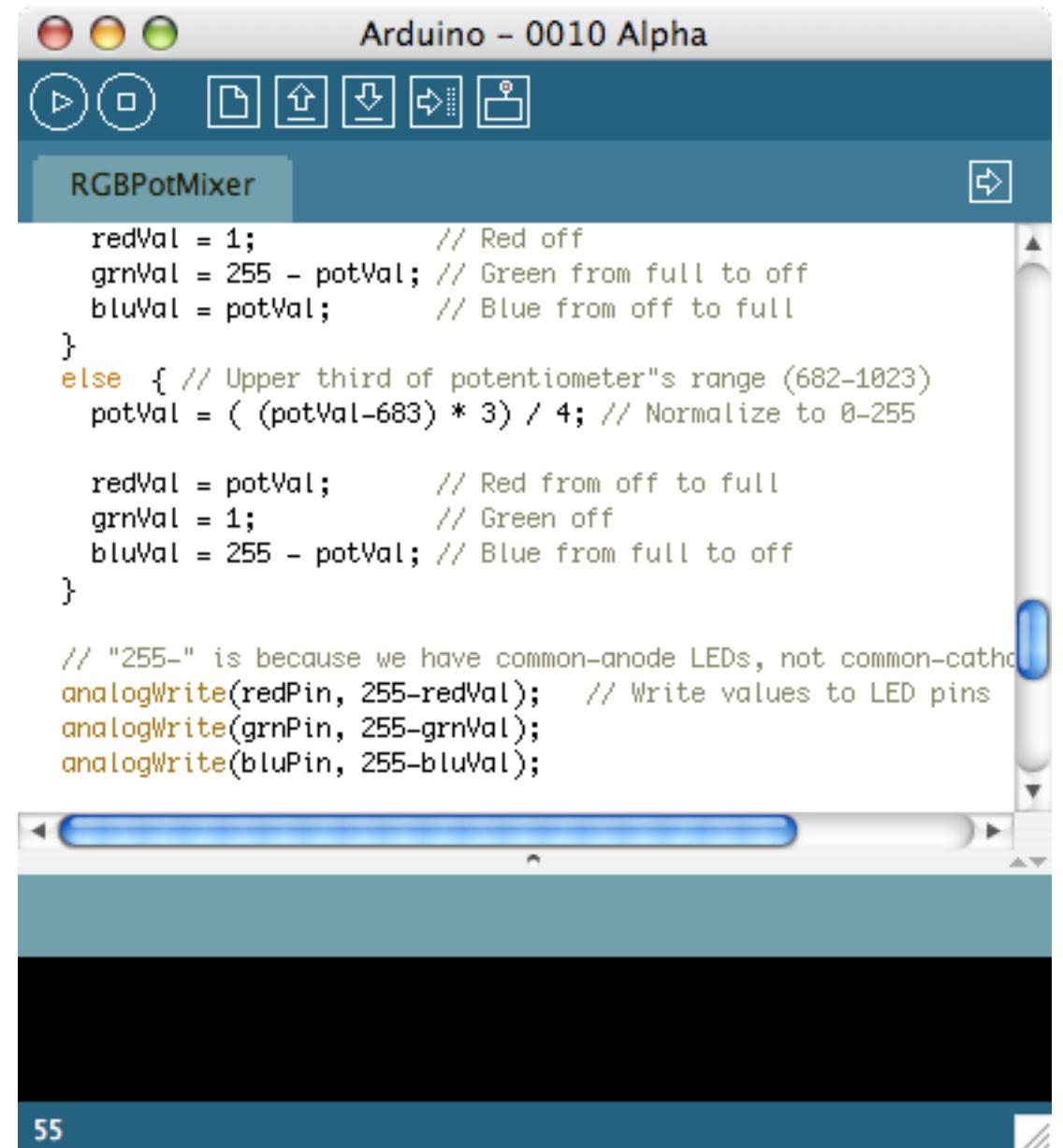


Pot-controlled RGB

“RGBPotMixer”

Use the pot from
before to control
the color mix

The code turns the single ranged
input value into “sectors” where
each sector is a color



The screenshot shows the Arduino IDE interface with the title bar "Arduino - 0010 Alpha". Below the title bar are standard IDE icons: play, stop, upload, download, and others. The main window displays the code for "RGBPotMixer". The code uses a potentiometer value (potVal) to determine the color output based on different ranges. It includes comments explaining the logic for turning off each color and normalizing the input. At the bottom of the code, it writes the values to three analog pins (redPin, grnPin, bluPin). The bottom portion of the IDE shows a preview window displaying a black screen.

```
redVal = 1;           // Red off
grnVal = 255 - potVal; // Green from full to off
bluVal = potVal;      // Blue from off to full
}
else { // Upper third of potentiometer's range (682-1023)
  potVal = ( (potVal-683) * 3 ) / 4; // Normalize to 0-255

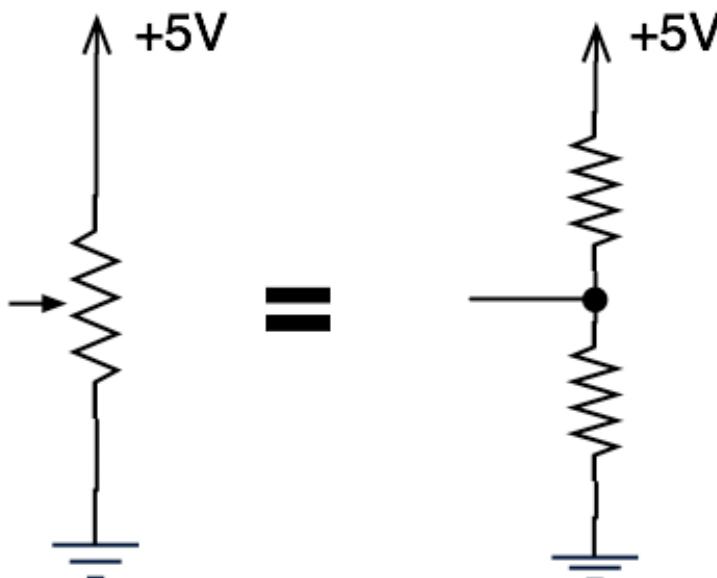
  redVal = potVal;      // Red from off to full
  grnVal = 1;           // Green off
  bluVal = 255 - potVal; // Blue from full to off
}

// "255-" is because we have common-anode LEDs, not common-cathode
analogWrite(redPin, 255-redVal); // Write values to LED pins
analogWrite(grnPin, 255-grnVal);
analogWrite(bluPin, 255-bluVal);
```

Also see “RGBPotMixer2” for a variation.
How would you change it to adjust brightness?

Sensing the Dark

- Pots are example of a *voltage divider*
- Voltage divider splits a voltage in two
- Same as two resistors, but you can vary them



Sensing the Dark: Photocells

- aka. photoresistor, light-dependent resistor
- A *variable* resistor
- Brighter light == lower resistance
- Photocells you have range approx. 0-10k-1M

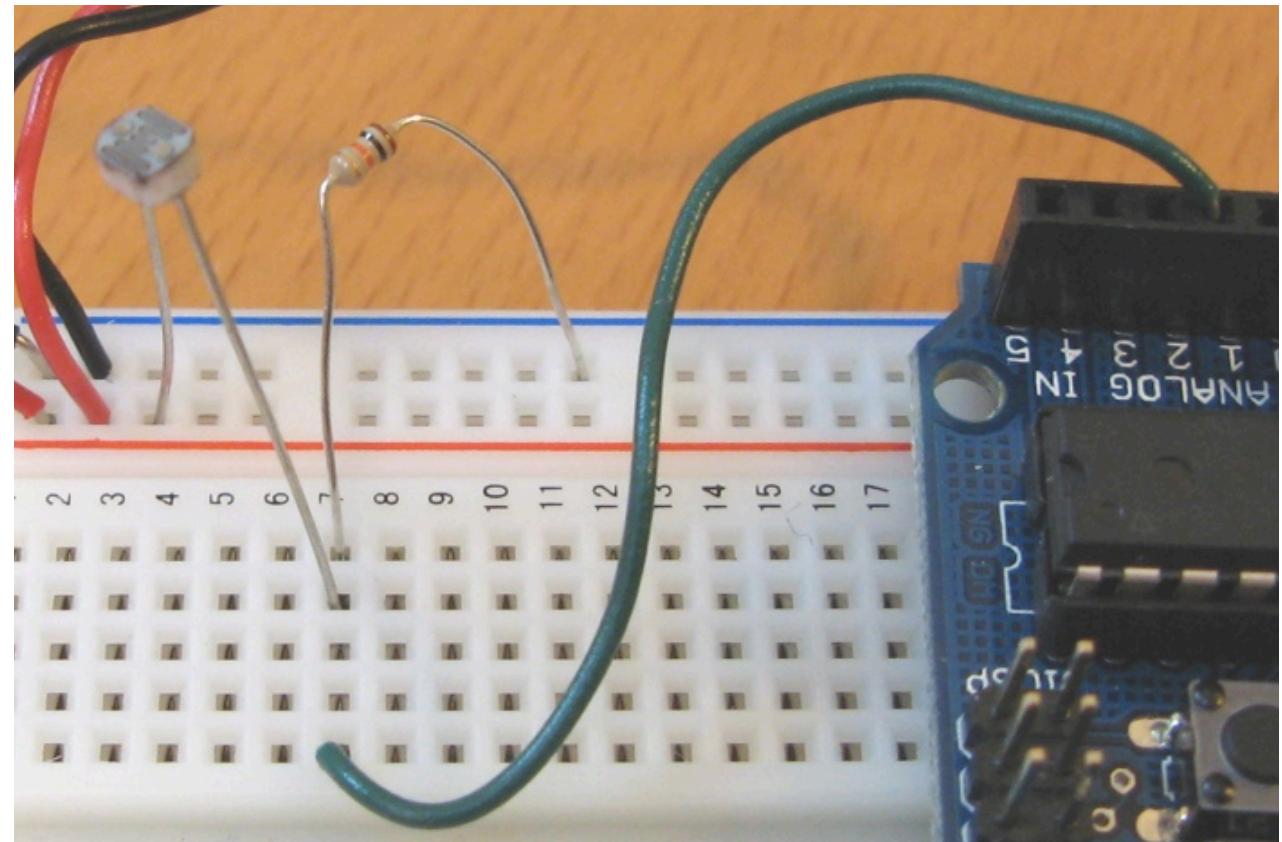
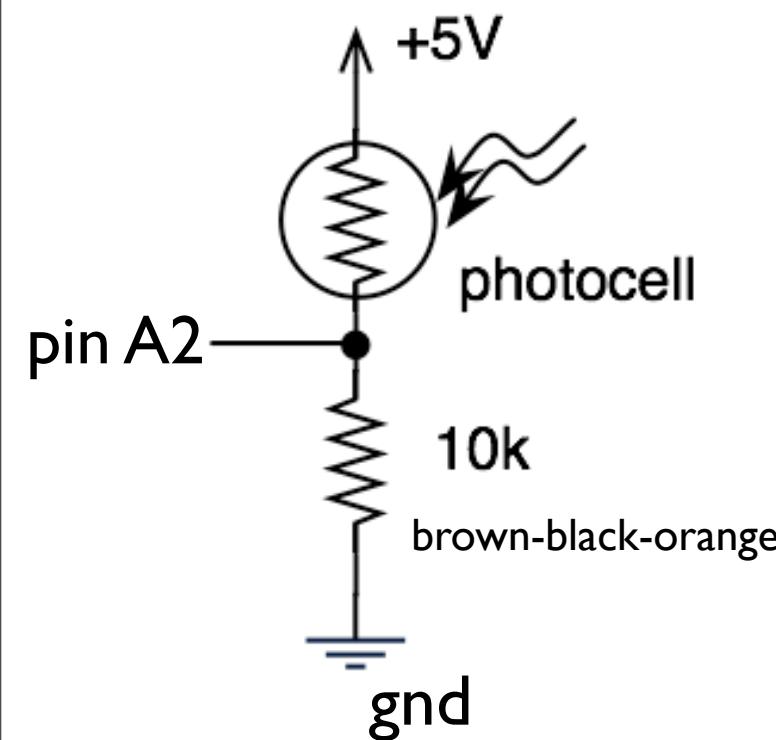


schematic symbol



Pretty cheap too. Can get a grab bag of 100 misc from Jameco for \$20

Photocell Circuit

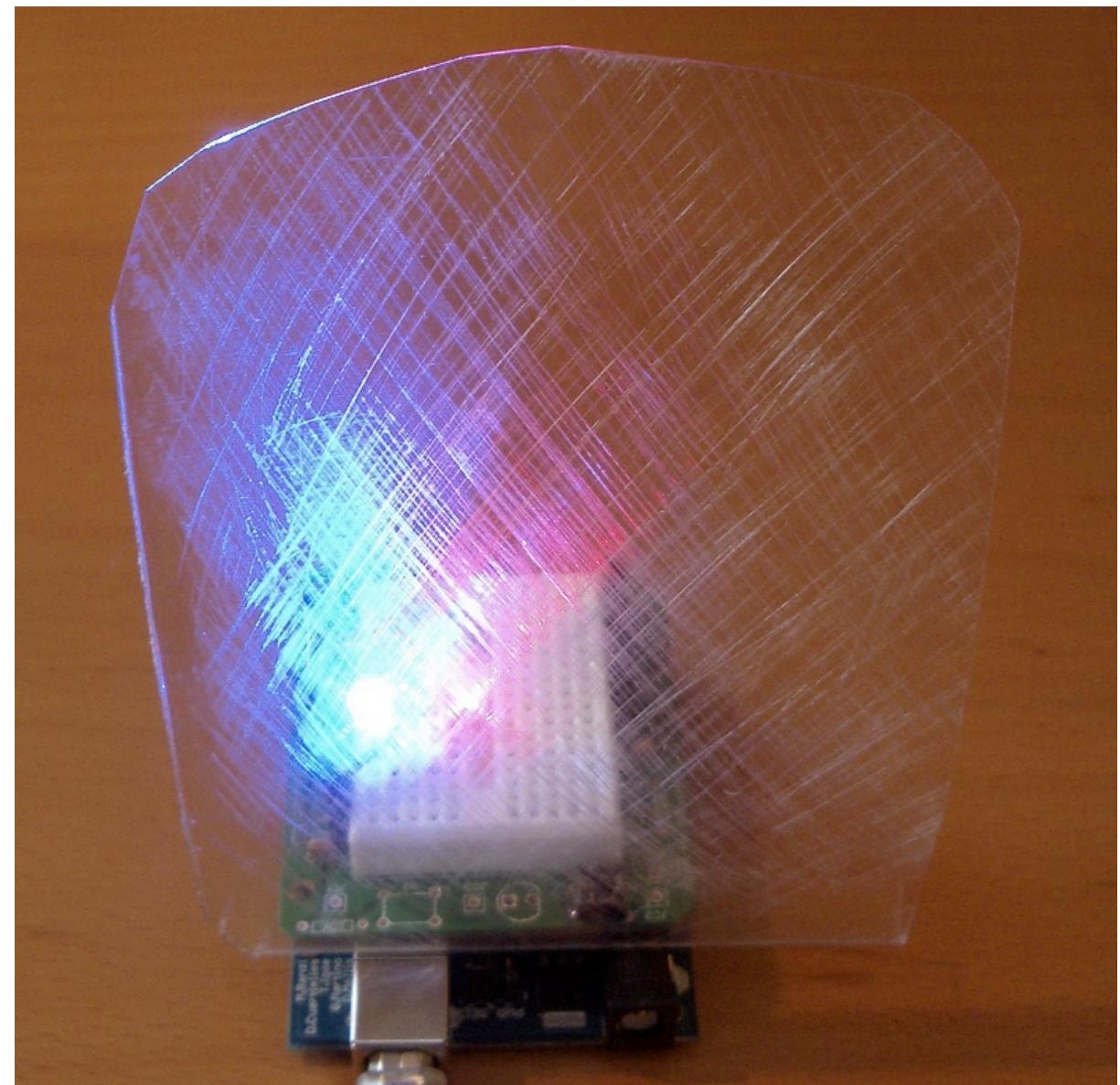


Try it with RGBPotMixer from before

Looks a lot like the pot circuit, doesn't it?

Mood Light

Diffuser made from
piece of plastic
scratched with
sandpaper



Also, can use plastic wrap scrunched up to make an interesting diffuser.

Resistive sensors

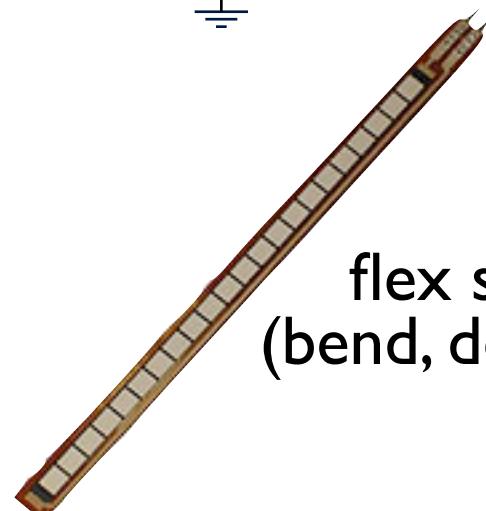
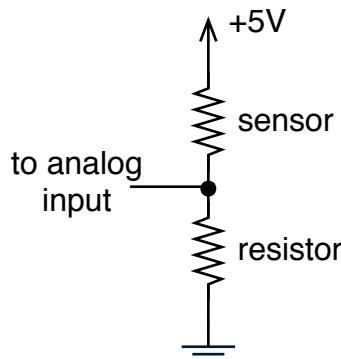


thermistor
(temperature)

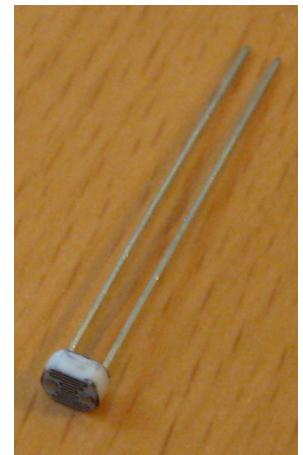


force sensors
(pressure)

circuit is the same
for all these



flex sensor
(bend, deflection)



photocell
(light)

also air pressure
and others

Thermistor image from: <http://www.facstaff.bucknell.edu/mastascu/elessonsHTML/Sensors/TempR.html>

Also see: <http://www.ladyada.net/make/midisense/makesensor.html>

Communicating with Others

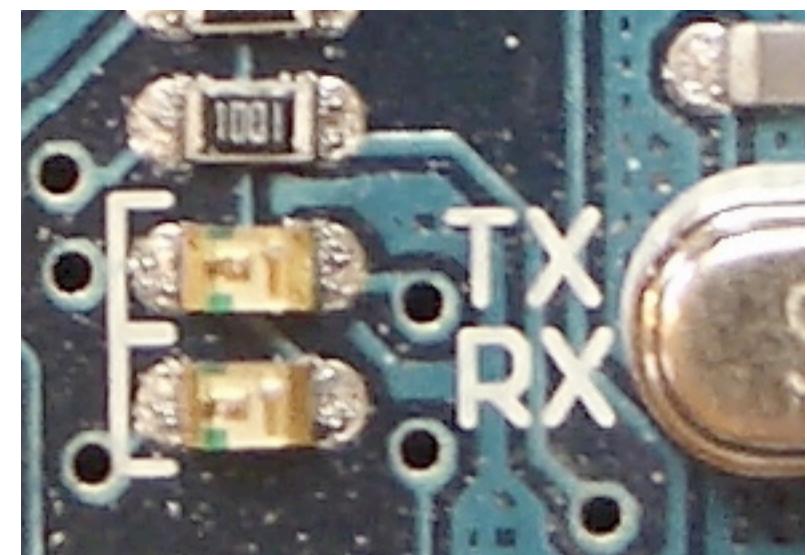
- Arduino can use same USB cable for programming and to talk with computers
- Talking to other devices uses the “Serial” commands
 - `Serial.begin()` – prepare to use serial
 - `Serial.print()` – send data to computer
 - `Serial.read()` – read data from computer

Can talk to not just computers.

Most things more complex than simple sensors/actuators speak serial.

Watch the TX/RX LEDS

- TX – sending to PC
- RX – receiving from PC
- Used when programming or communicating



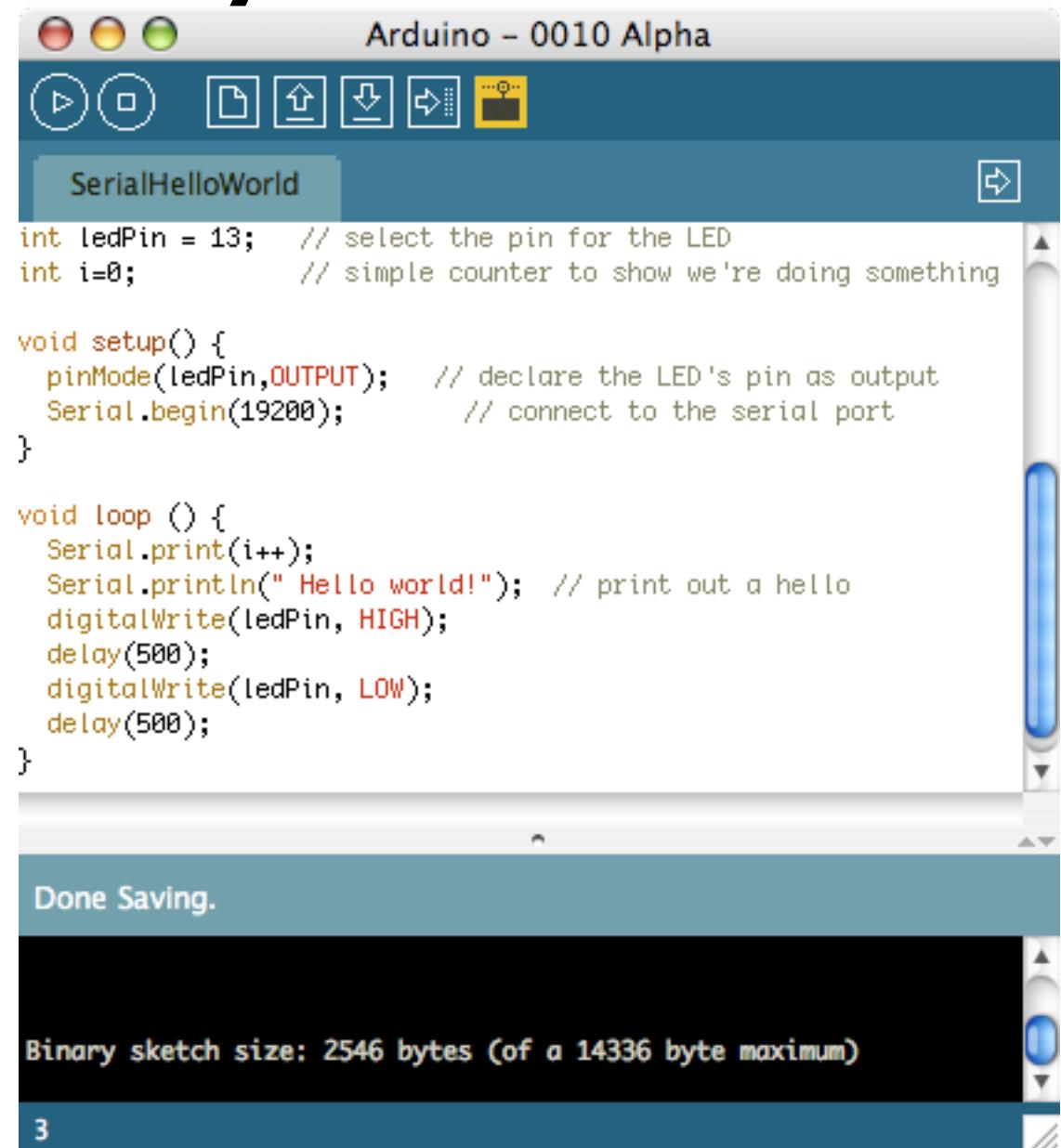
Arduino Says “Hi”

“SerialHelloWorld”

Sends “Hello world!”
to your computer

Click on “Serial
Monitor” button to
see output

Watch TX LED compared
to pin 13 LED



The screenshot shows the Arduino IDE interface with the title bar "Arduino - 0010 Alpha". Below the title bar is a toolbar with various icons. The main area displays the code for "SerialHelloWorld". The code initializes pin 13 as an output, sets up the serial port at 19200 baud, and then enters a loop where it prints the current value of a counter (i) followed by "Hello world!", toggles the LED connected to pin 13, and delays for 500 milliseconds. A message "Done Saving." is visible in the status bar at the bottom.

```
int ledPin = 13; // select the pin for the LED
int i=0; // simple counter to show we're doing something

void setup() {
  pinMode(ledPin,OUTPUT); // declare the LED's pin as output
  Serial.begin(19200); // connect to the serial port
}

void loop () {
  Serial.print(i++);
  Serial.println(" Hello world!"); // print out a hello
  digitalWrite(ledPin, HIGH);
  delay(500);
  digitalWrite(ledPin, LOW);
  delay(500);
}
```

Done Saving.

Binary sketch size: 2546 bytes (of a 14336 byte maximum)

3

This sketch is located in the handout, but it's pretty short.
Use on-board pin 13 LED, no need to wire anything up.

Telling Arduino What To Do

“SerialReadBasic”

You type “H”, LED blinks

In “Serial Monitor”,
type “H”, press Send

Serial.available() tells
you if data present to read

The screenshot shows the Arduino IDE interface. At the top, it says "Arduino - 0010 Alpha". Below the title bar are standard OS X window controls (red, yellow, green buttons) and a toolbar with icons for play, stop, upload, and others. The main area displays the code for "SerialReadBasic":

```
pinMode(ledPin,OUTPUT); // declare the LED's pin as output
Serial.begin(19200); // connect to the serial port
}

void loop () {
// Serial.available() is a way to see if there's serial data
// without pausing your code
if( Serial.available() ) {
    val = Serial.read(); // read the serial port
    if( val == 'H' ) { // if it's an 'H', blink the light
        digitalWrite(ledPin, HIGH);
        delay(1000);
        digitalWrite(ledPin, LOW);
    }
}
}

19200 baud H Send
```

At the bottom of the IDE, the status bar shows the number "7".

The "Serial Monitor" window below the code editor shows the baud rate set to "19200 baud" and the character "H" entered in the text input field. A "Send" button is to the right of the input field.

This sketch is in the handout

Always check Serial.available() or if Serial.read() != -1 to determine if there's actual data to read.

Can modify it to print “hello world” after it receives something, but before it checks for ‘H’. This way you can verify it’s actually receiving something.

Arduino Communications

is just serial communications

- Psst, Arduino doesn't really do USB
- It really is “serial”, like old RS-232 serial
- All microcontrollers can do serial
- Not many can do USB
- Serial is easy, USB is hard



serial terminal from the olde days

Serial Communications

- “Serial” because data is broken down into bits, each sent one after the other down a single wire.
 - The single ASCII character ‘B’ is sent as:

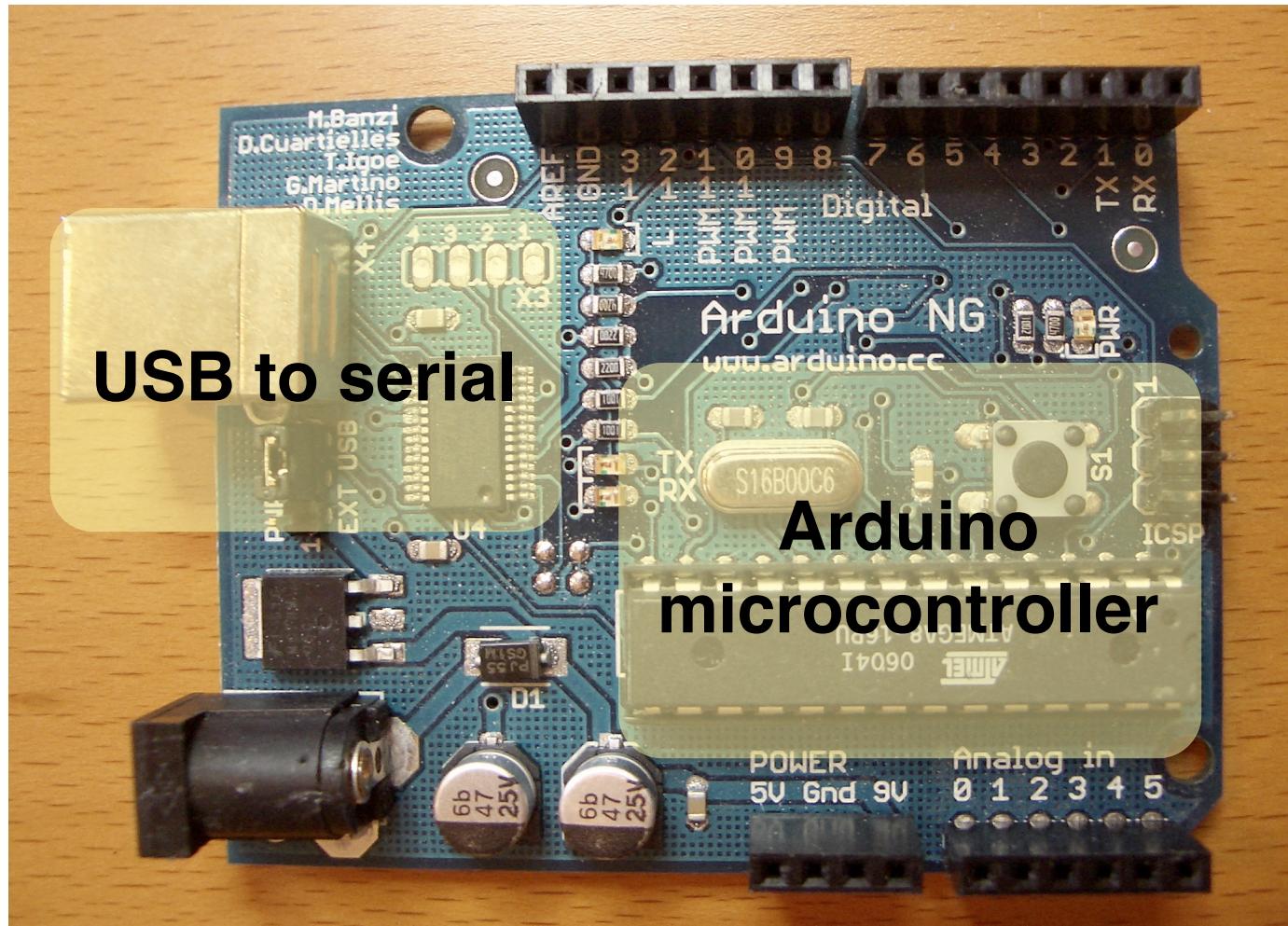
'B' = 0 1 0 0 0 0 1 0
 = L H L L L L H L
 = 
 HIGH
 LOW

- Toggle a pin to send data, just like blinking an LED
 - You could implement sending serial data with `digitalWrite()` and `delay()`
 - A single data wire needed to send data. One other to receive.

Note, a single data wire. You still need a ground wire.

Arduino & USB-to-serial

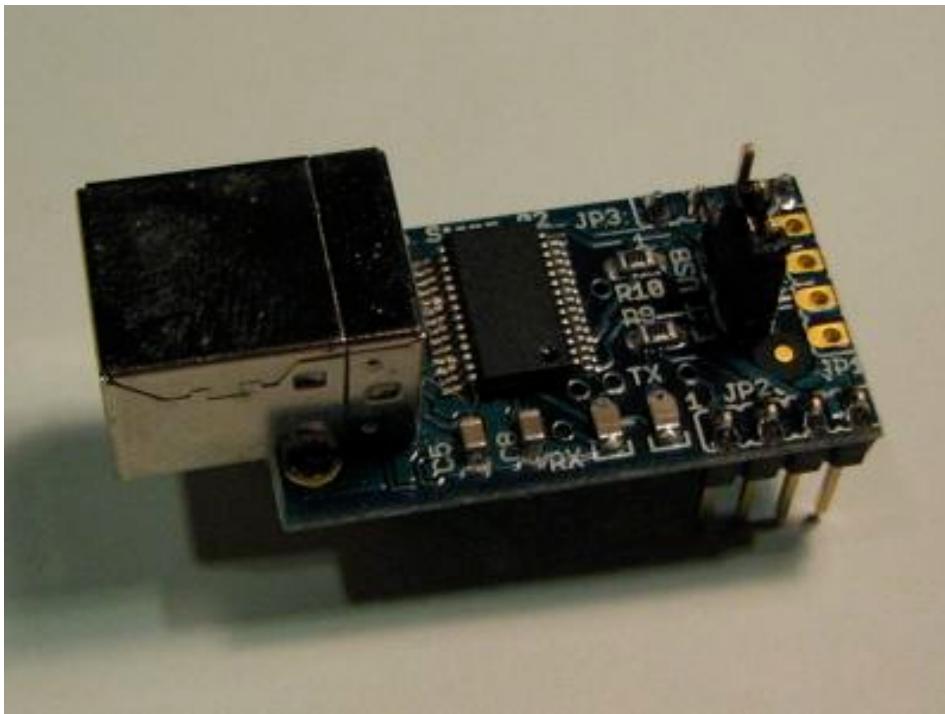
Arduino board is really two circuits



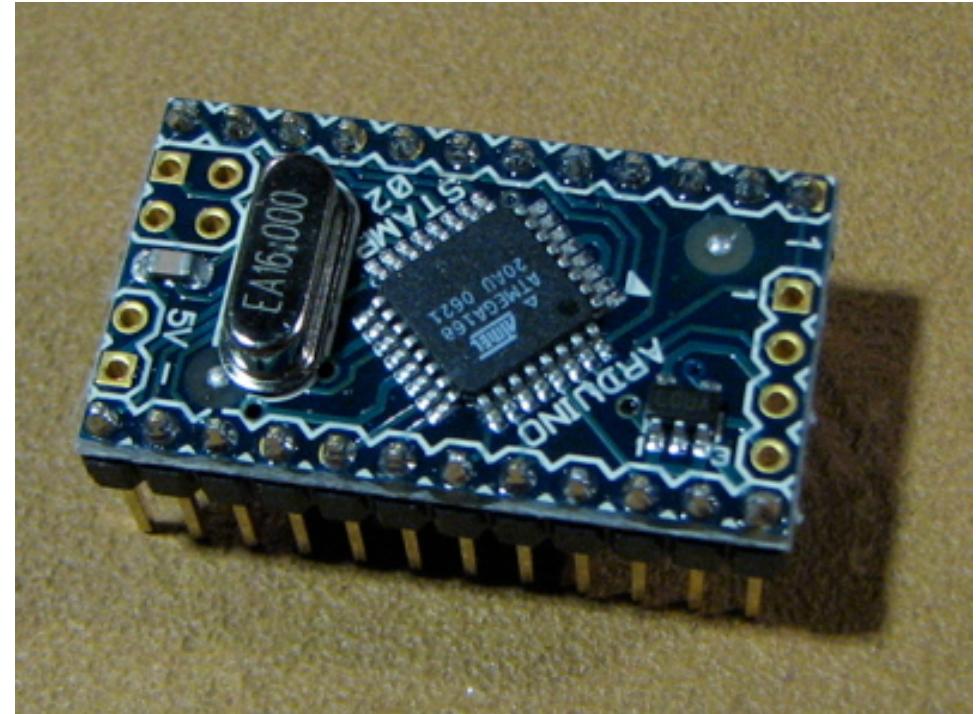
Original Arduino boards were RS-232 serial, not USB.

Arduino Mini

Arduino Mini separates the two circuits



Arduino Mini USB adapter

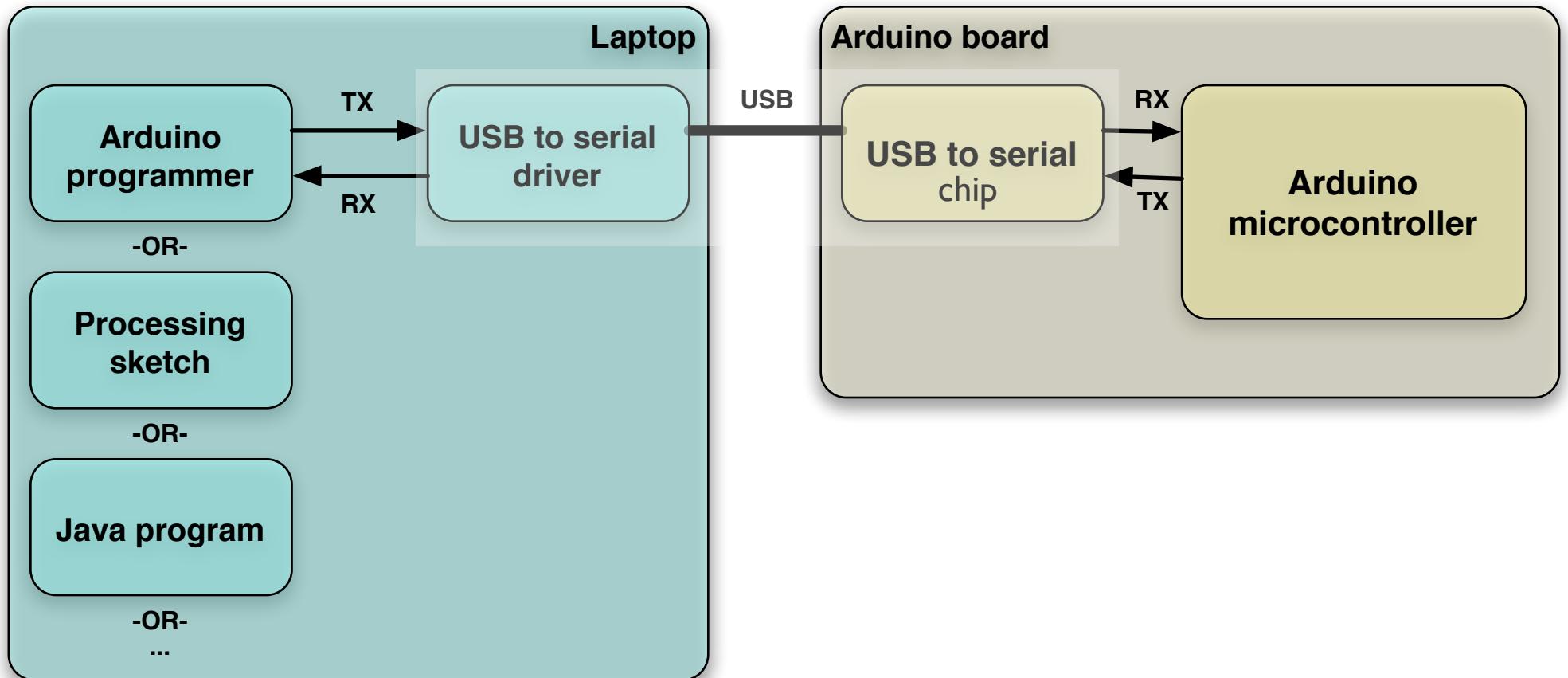


Arduino Mini

aka. "Arduino Stamp"

If you don't talk with a computer, the USB-to-serial functionality is superfluous.

Arduino to Computer



USB is totally optional for Arduino
But it makes things easier

Original Arduino boards were RS-232 serial, not USB.

All programs that talk to Arduino (even the Arduino IDE) think that they're talking via a serial port.

Arduino & USB

- Since Arduino is all about serial
- And not USB,
- Interfacing to things like USB flash drives, USB hard disks, USB webcams, etc. is *not* possible

Also, USB is a host/peripheral protocol. Being a USB “host” means needing a lot of processing power and software, not something for a tiny 8kB microcontroller.
It can be a peripheral. In fact, there is an open project called “AVR-USB” that allows AVR chips like used in Arduino to be proper USB peripherals. See: <http://www.obdev.at/products/avrusb/>

Controlling the Computer

- Can send sensor data from Arduino to computer with `Serial.print()`
- There are many different variations to suite your needs:

```
int val = 123;  
Serial.print(val);      // sends 3 ASCII chars "123"  
Serial.print(val,DEC); // same as above  
Serial.print(val,HEX); // sends 2 ASCII chars "7B"  
Serial.print(val,BIN); // sends 8 ASCII chars "01111011"  
Serial.print(val,BYTE); // sends 1 byte, the verbatim value
```

Controlling the Computer

You write one program on Arduino, one on the computer

In Arduino: read sensor, send data as byte

```
void loop() {
    val = analogRead(analogInput);      // read the value on analog input
    Serial.print(val/4,BYTE);          // print a byte value out
    delay(50);                      // wait a bit to not overload the port
}
```

In Processing: read the byte, do something with it

```
import processing.serial.*;

Serial myPort; // The serial port

void setup() {
    String portname = "/dev/tty.usbserial-A3000Xv0";
    myPort = new Serial(this, myPort, 9600);
}

void draw() {
    while (myPort.available() > 0) {
        int inByte = myPort.read();
        println(inByte);
    }
}
```

But writing Processing programs is for later

Controlling the Computer

- Receiving program on the computer can be in any language that knows about serial ports
- C/C++, Perl, PHP, Java, Max/MSP, Python, Visual Basic, etc.
- Pick your favorite one, write some code for Arduino to control

If interested, I can give details on just about every language above.

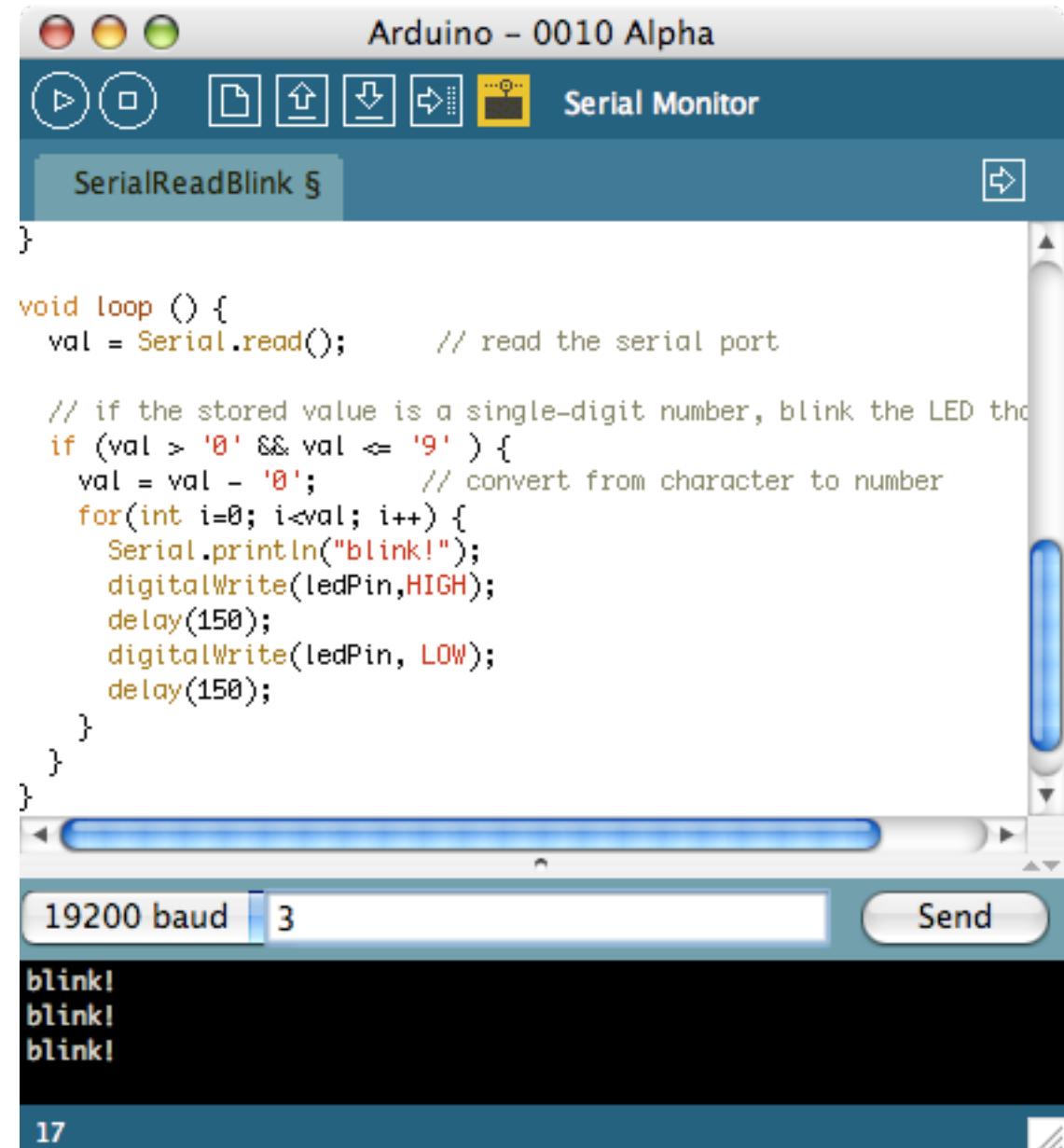
Controlling Arduino, Again

“SerialReadBlink”

Type a number 1-9
and LED blinks that
many times

Converts typed ASCII value
into usable number

Most control issues are
data conversion issues



The screenshot shows the Arduino IDE interface. The title bar reads "Arduino - 0010 Alpha". The menu bar includes File, Edit, Tools, Sketch, Help, and a language selection dropdown. Below the menu is a toolbar with icons for Open, Save, Print, and others. The main area displays the code for "SerialReadBlink.ino". The code reads a character from the serial port, converts it to a number, and then blinks an LED that many times. The Serial Monitor window at the bottom shows the baud rate set to 19200 and the text "blink!" repeated three times, corresponding to the number 3 entered in the input field.

```
void loop () {
    val = Serial.read();      // read the serial port

    // if the stored value is a single-digit number, blink the LED that many times
    if (val > '0' && val <= '9' ) {
        val = val - '0';      // convert from character to number
        for(int i=0; i<val; i++) {
            Serial.println("blink!");
            digitalWrite(ledPin,HIGH);
            delay(150);
            digitalWrite(ledPin, LOW);
            delay(150);
        }
    }
}
```

19200 baud 3 Send

blink!
blink!
blink!

This sketch is also in the handout

Serial-controlled RGB

“SerialRGBLED”

Send color commands
to Arduino

e.g. “r200”, “g50”, “b0”

Sketch parses what you
type, changes LEDs

The screenshot shows the Arduino IDE's Serial Monitor window. The title bar says "Arduino - 0010 Alpha". The menu bar includes icons for File, Open, Save, Upload, Download, Tools, and Serial Monitor. The "SerialRGBLED" tab is selected. The code area contains the following sketch:

```
void loop () {
    //read the serial port and create a string out of what you read
    readSerialString(serInString);

    colorCode = serInString[0];
    if( colorCode == 'r' || colorCode == 'g' || colorCode == 'b' ) {
        colorVal = atoi(serInString+1);
        Serial.print("setting color ");
        Serial.print(colorCode);
        Serial.print(" to ");
        Serial.print(colorVal);
        Serial.println();
        serInString[0] = 0;                                // indicates we've used the first character
        if(colorCode == 'r')
            analogWrite(redPin, 255-colorVal);
        else if(colorCode == 'g')
            analogWrite(greenPin, 255-colorVal);
    }
}
```

The Serial Monitor window shows the baud rate set to 19200 and the message "b255" entered. The output window displays:

```
19200 baud b255
enter color command (e.g. 'r43') :
setting color r to 255
```

This sketch is in the handout.

Color command is two parts: colorCode and colorValue

colorCode is a character, ‘r’, ‘g’, or ‘b’.

colorValue is a number between 0-255.

Sketch shows rudimentary character string processing in Arduino.

This is still one of the hardest tasks, unfortunately.

Reading Serial Strings

- The function “Serial.available()” makes reading strings easier
- Can use it to read all available serial data from computer
- The “readSerialString()” function at right takes a character string and sticks available serial data into it

```
//read a string from the serial and store it in an array
//you must supply the array variable
void readSerialString (char *strArray) {
    int i = 0;
    if(!Serial.available()) {
        return;
    }
    while (Serial.available()) {
        strArray[i] = Serial.read();
        i++;
    }
    strArray[i] = 0; // indicate end of read string
}
```

Pay no attention to the pointer symbol (“*”)
Must be careful about calling readSerialString() too often or you'll read partial strings

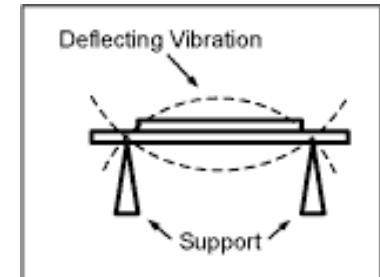
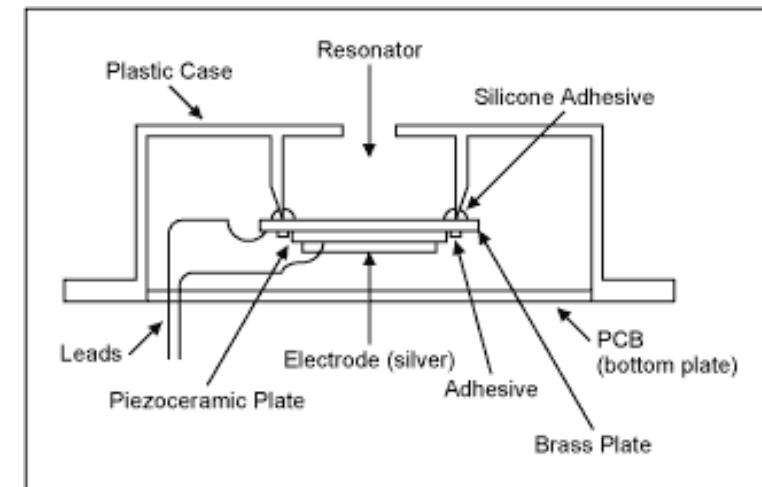
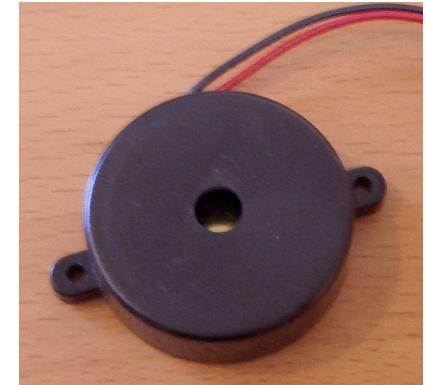
Piezoelectrics

- Big word – *piezein* is greek for “squeeze”
- Some crystals, when squeezed, make a spark
- Turns out the process goes the other way too
- Spark a quartz crystal, and it flexes
- Piezo buzzers use this to make sound
(flex something back and forth, it moves air)

Piezo buzzers don't have quartz crystals, but instead a kind of ceramic that also exhibits piezoelectric properties.
I pronounce it “pie-zoh”. Or sometimes “pee-ay-zoh”.

Piezo Buzzers

- Two wires, red & black.
Polarity matters: black=ground
- Apply an oscillating voltage to make a noise
- The buzzer case supports the piezo element and has resonant cavity for sound



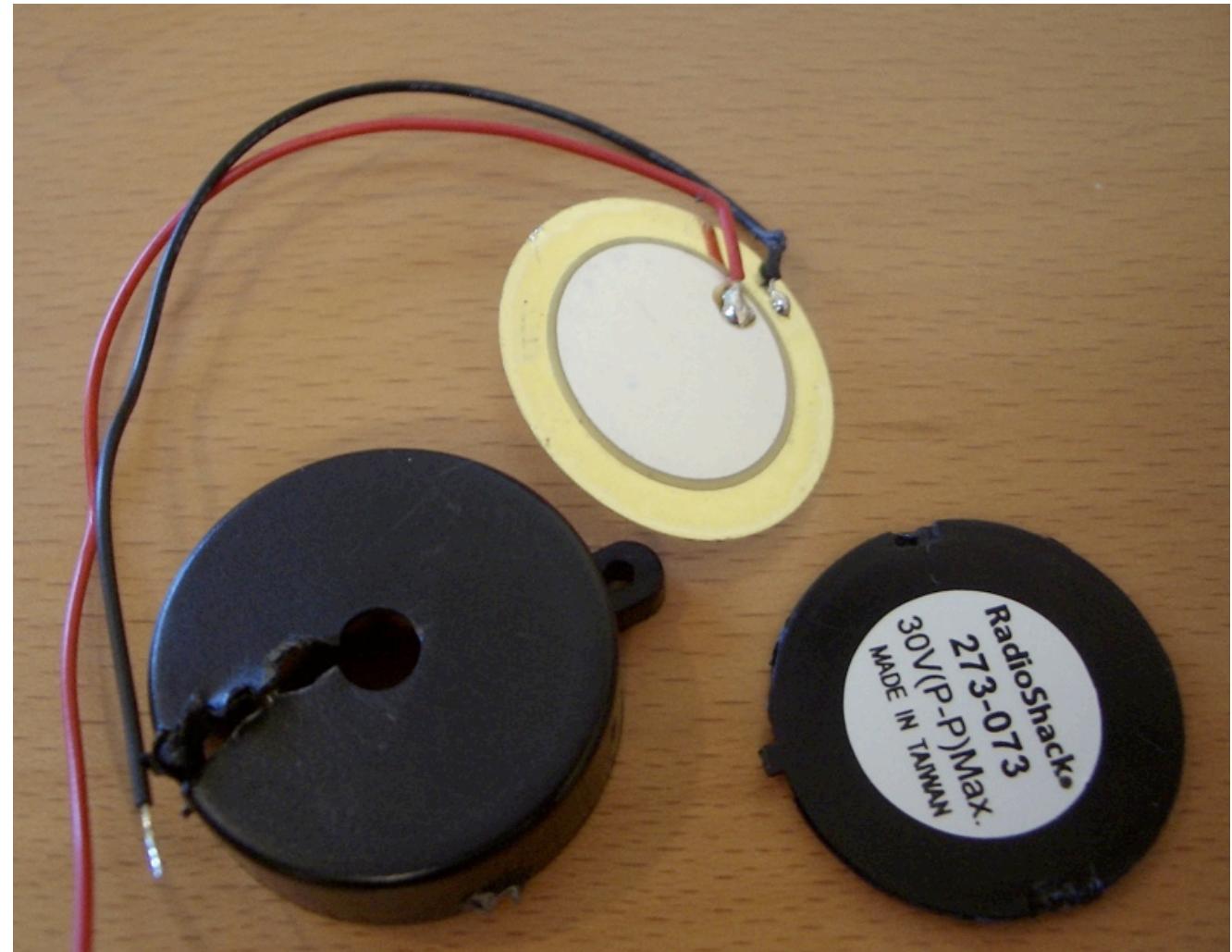
Oscillating voltage alternately squeezes and releases the piezo element.
Must apply fluctuating voltage, a steady HIGH or LOW won't work.

What's in a Piezo Buzzer?

You can get at the piezo element pretty easily.

Be careful not to crack the white disc that is the actual piezo

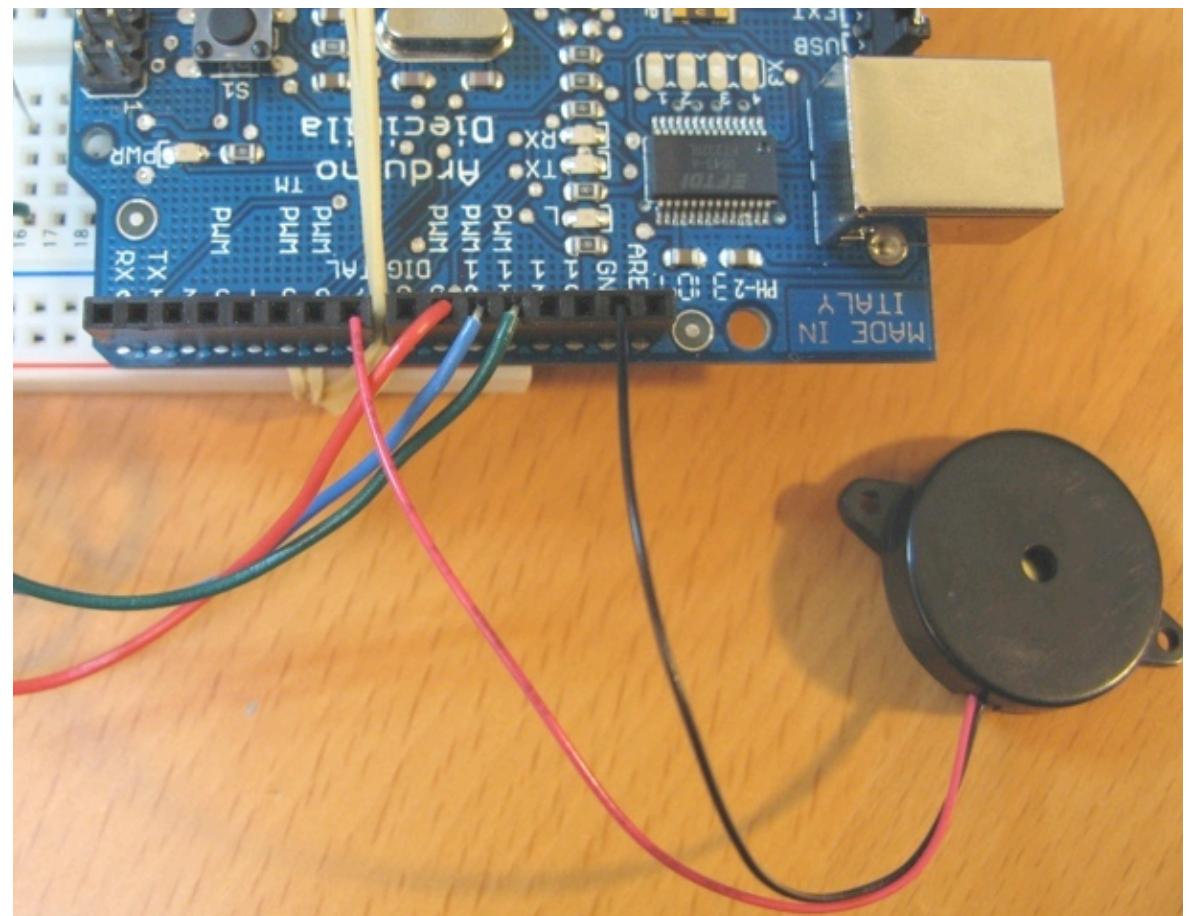
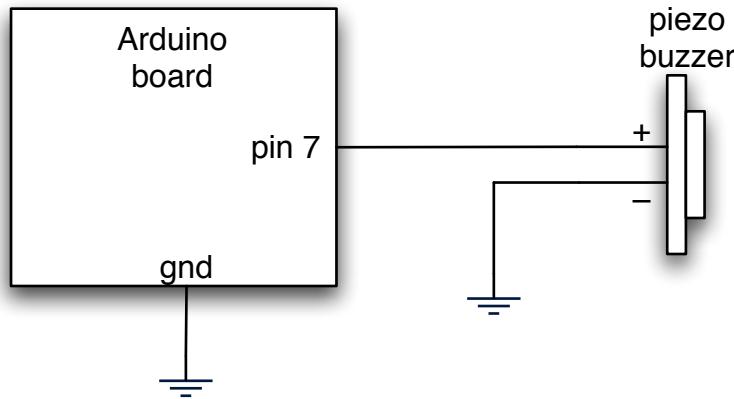
Only take it out of its case to use it as a sensor



another \$1.99 I won't be getting back from Radio Shack

Of course, you usually destroy the enclosure to get at the element.
And it's the enclosure that has the proper support and resonant cavity to make a loud sound

Piezo Buzzer



Piezo leads are very thin. The breadboard holes grab them better than the header sockets, which is why the jumper leads are used.
Or you can jam a jumper wire in the holes to hold in the piezo leads.

Play a Melody

“SoundSerial”

Play the piezo beeper
with the Serial Monitor

Type multiple letters
from “cdefgabC” to
make melodies

The screenshot shows the Arduino IDE's Serial Monitor window titled "Arduino - 0010 Alpha". The sketch name "SoundSerial" is selected. The code implements a melody player using the Serial port. It reads a byte from the serial port, prints it, and then looks for a note name in an array. If found, it plays the note for 50 cycles. The notes are mapped to tones [0-7] and play for different durations. The baud rate is set to 19200. The serial monitor displays the message "ready" followed by the transmitted melody sequence: "aaffbbegbdfabcddefgdefab".

```
void loop() {
    digitalWrite(speakerPin, LOW);
    serByte = Serial.read();
    if (serByte != -1) {
        Serial.print(serByte, BYTE);
        ledState = !ledState; // flip the LED state
        digitalWrite(ledPin, ledState); // write to LED
    }
    for (count=0;count<=8;count++) { // look for the note
        if (names[count] == serByte) { // ahh, found it
            for( int i=0; i<50; i++ ) { // play it for 50 cycles
                digitalWrite(speakerPin, HIGH);
                delayMicroseconds(tones[count]);
                digitalWrite(speakerPin, LOW);
                delayMicroseconds(tones[count]);
            }
        }
    }
}
```

19200 baud

ready

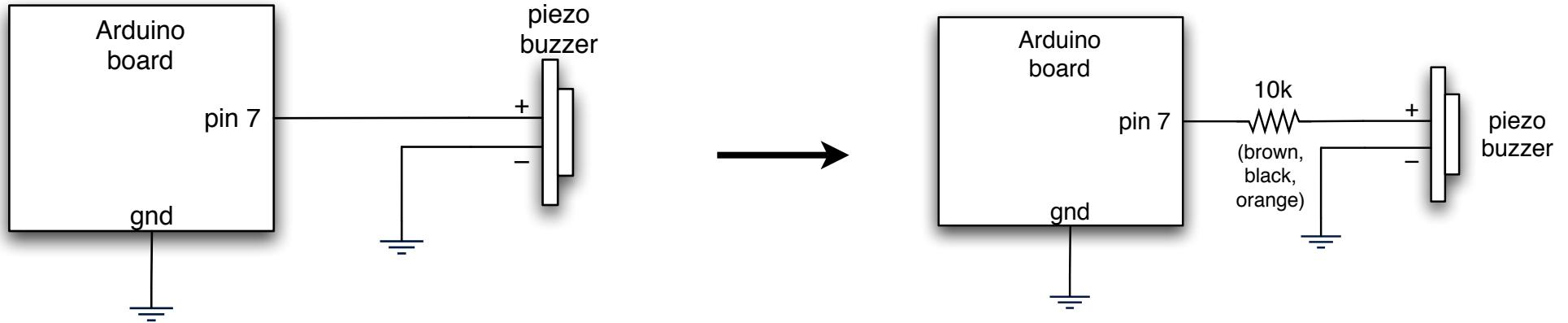
aaffbbegbdfabcddefgdefab

31

This sketch is in the handout,
Notice the problem with this sketch?
Different notes play for different amounts of time.
50 cycles of low C isn't the same amount of time as 50 cycles of high B

Making it Quieter

Easiest way: add a resistor



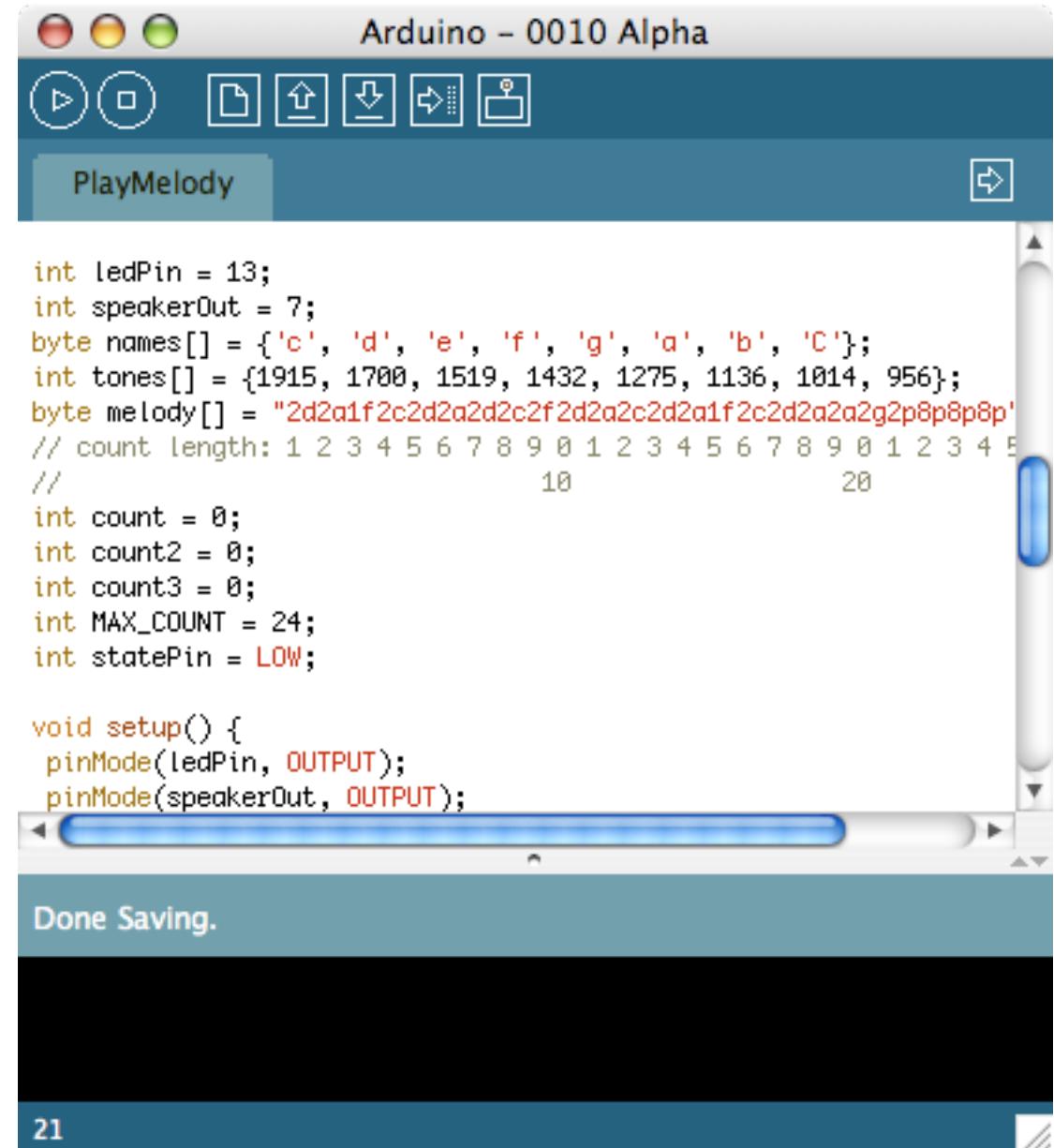
Like most things in electronics, if you want less of something, add a resistor.
A better value would probably be 1k, but we don't have that on hand.
This may not seem important now, but wait for the next project.

Play a Stored Melody

“PlayMelody”

Plays a melody stored
in the Arduino

Could be battery-powered, play
melody on button trigger, control
playback speed with photocell, etc.



The screenshot shows the Arduino IDE interface with the title bar "Arduino - 0010 Alpha". Below the title bar is a toolbar with various icons. The main area displays the code for the "PlayMelody" sketch. The code defines variables for pins, note names, tones, and a melody string. It includes a setup function that initializes the pins as outputs. A message "Done Saving." is displayed in the status bar at the bottom. The status bar also shows the number "21" and a small icon.

```
int ledPin = 13;
int speakerOut = 7;
byte names[] = {'c', 'd', 'e', 'f', 'g', 'a', 'b', 'C'};
int tones[] = {1915, 1700, 1519, 1432, 1275, 1136, 1014, 956};
byte melody[] = "2d2a1f2c2d2a2d2c2f2d2a2c2d2a1f2c2d2a2g2p8p8p8p";
// count length: 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
//                                         10                                20
int count = 0;
int count2 = 0;
int count3 = 0;
int MAX_COUNT = 24;
int statePin = LOW;

void setup() {
  pinMode(ledPin, OUTPUT);
  pinMode(speakerOut, OUTPUT);
```

Melody definition is sort of like the old cell ringtone style
Melody playing logic is a little hard to follow, since it is timing critical.

Make a Theremin

“ooo-weee-ooooo”

The original spooky
sound machine

Works by measuring your
body's electric field

No touching needed!

We'll use light in lieu of RF



Leon Theremin

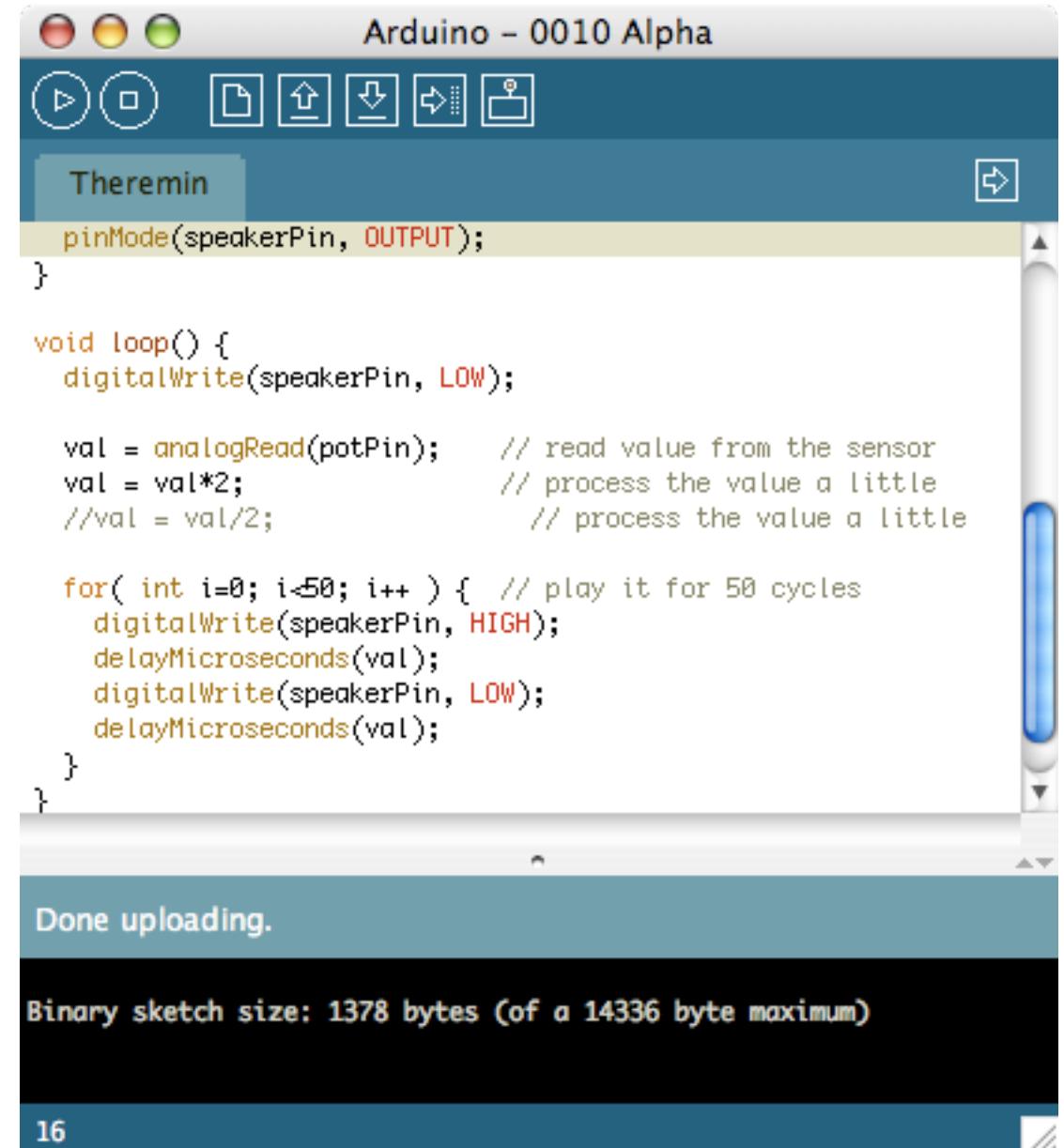
As heard on Star Trek, Beach Boys, horror movies, Mars Attacks!, and bad New Age songs.
Works sorta like those touch switches, but no touching here.
That is, your body becomes a variable capacitor.

Light Theremin

“Theremin”

Move hand over
photocell to
change pitch

Play with val processing & cycles count
to alter sensitivity, pitch and timbre



The screenshot shows the Arduino IDE interface with the title bar "Arduino - 0010 Alpha". Below the title bar are standard file menu icons. The main area displays a code editor with a sketch titled "Theremin". The code uses analog input to control digital output, specifically a speaker pin. It includes a setup section for setting the speaker pin as an output and a loop section that reads the value from a photocell, processes it, and then alternates the speaker pin between HIGH and LOW states 50 times per cycle. A message "Done uploading." is visible in the status bar at the bottom, along with the text "Binary sketch size: 1378 bytes (of a 14336 byte maximum)".

```
pinMode(speakerPin, OUTPUT);
}

void loop() {
    digitalWrite(speakerPin, LOW);

    val = analogRead(potPin);      // read value from the sensor
    val = val*2;                  // process the value a little
    //val = val/2;                // process the value a little

    for( int i=0; i<50; i++ ) {   // play it for 50 cycles
        digitalWrite(speakerPin, HIGH);
        delayMicroseconds(val);
        digitalWrite(speakerPin, LOW);
        delayMicroseconds(val);
    }
}

Done uploading.

Binary sketch size: 1378 bytes (of a 14336 byte maximum)
```

16

Okay so maybe it sounds more like a bad video game than a spooky movie
The glitchy sound is cause because of the time it takes to read the sensor
There are ways around such stuff, but requires more complex programming using timers & interrupts
The sound can get annoying quick

Other Serial Devices



to Wi-Fi

to Ethernet



to graphic LCD

to 8-servo controller

Lantronix Wi-Port and Lantronix Xport <http://lantronix.com/>

Seetron Serial Graphic display and Mini SSC <http://www.seetron.com/slcds.htm> <http://www.seetron.com/ssc.htm>

Serial Examples



to Roomba

You've already seen this. :)
<http://hackingroomba.com/>

Going Further

- Piezo buzzers
 - Can hook up multiple buzzers for polyphonic sound
 - Can play waves other than just square waves using PWM techniques
 - Can also be used as input devices (we'll cover that later)

Going Further

- Serial communications
 - Not just for computer-to-Arduino communications
 - Many other devices speak serial
 - Older keyboards & mice speak are serial (good for sensors!)
 - Interface boards (graphic LCDs, servo drivers, RFID readers, Ethernet, Wi-Fi)

Going Further

- RGB LEDS
 - You can pretty easily replicate the Ambient Orb (\$150) functionality
 - Make a status display for your computer
 - Computer-controlled accent lighting (a wash of color against the walls)



Ambient Orb doesn't connect to computer though. Uses the pager network.
Ambient Devices: <http://www.ambientdevices.com/>

END Class 2

<http://todbot.com/blog/bionicarduino/>

Tod E. Kurt

tod@todbot.com

Feel free to email me if you have any questions.