

# Introduction to Arduino

## Data Bootcamp


**Instructor: MakeltHackin**

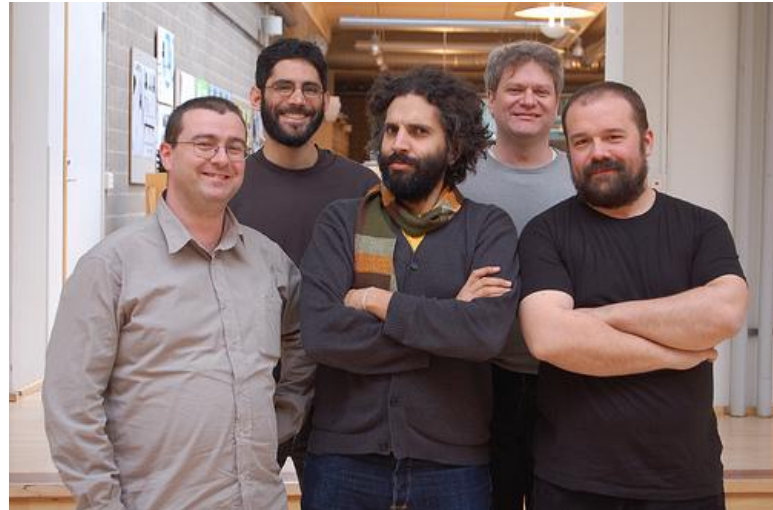
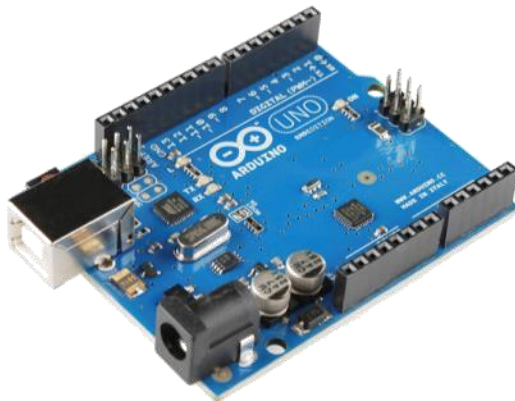


# Overview

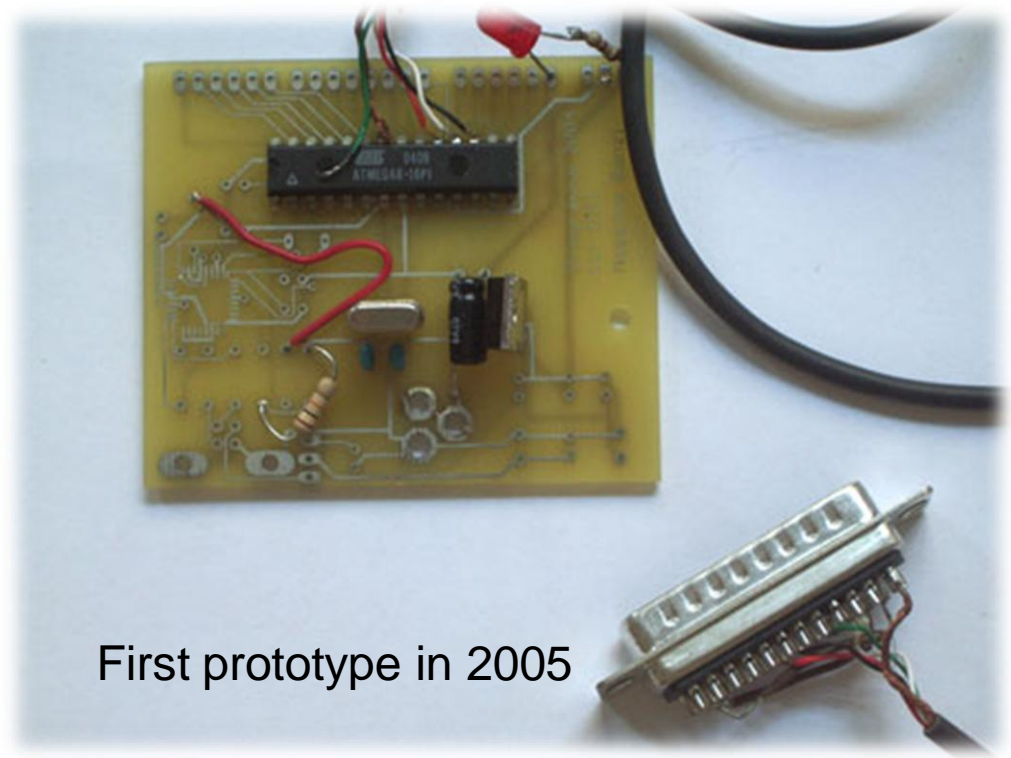
- Who is Arduino and What Does He Do?
- Arduino and Space Apps
- Parts, Tools, and Equipment
- Integrated Development Environment
- Example Sketches (aka code/programs/apps)
- Fun with LEDs!

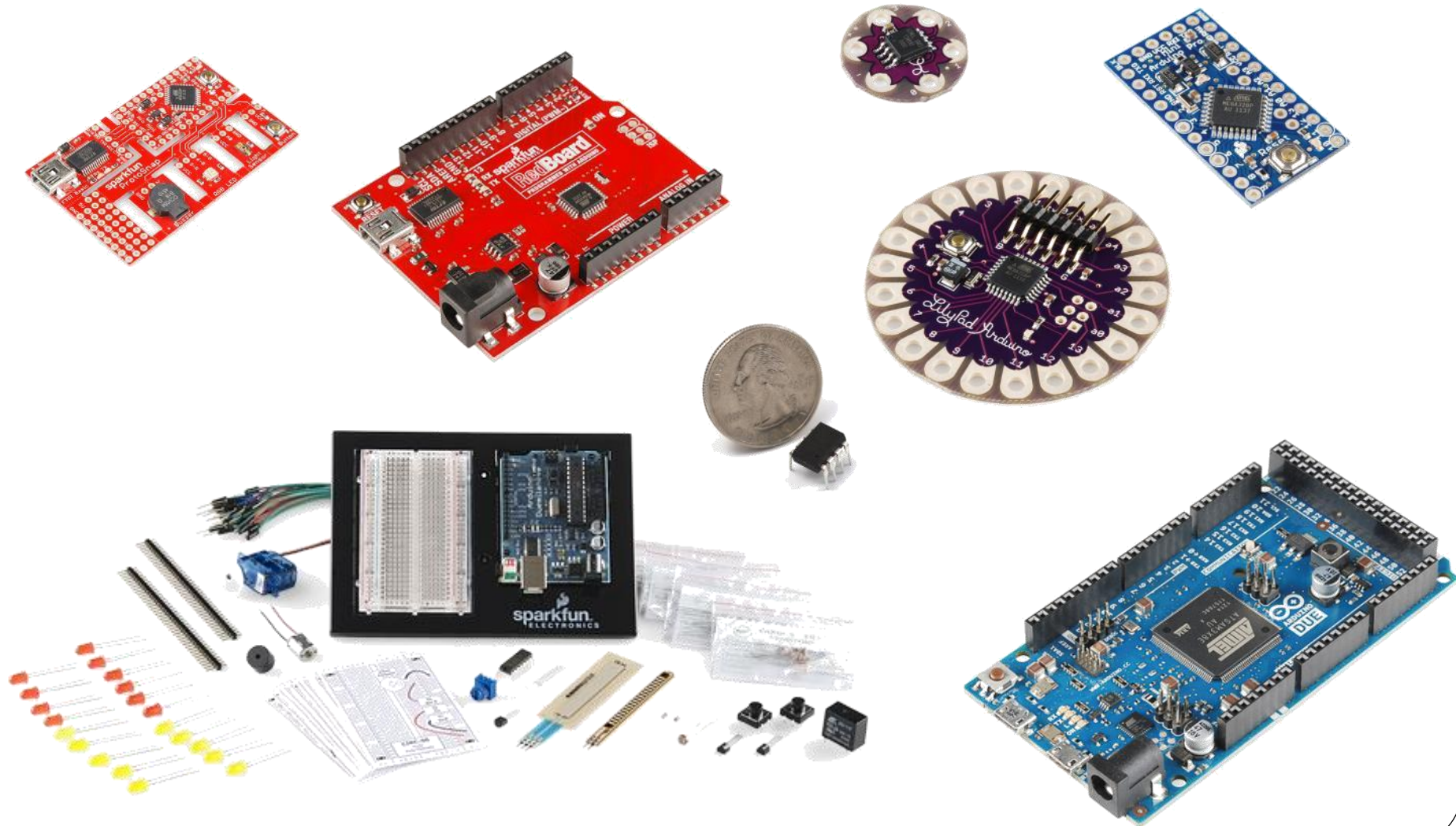
# Who is Arduino and What Does He Do?

- “Strong Friend” Created in Ivrea, Italy in 2005 by Massimo Banzi & David Cuartielles
- Open Source Hardware
-  Processor
- Coding is accessible & transferrable (C++, Processing, java)

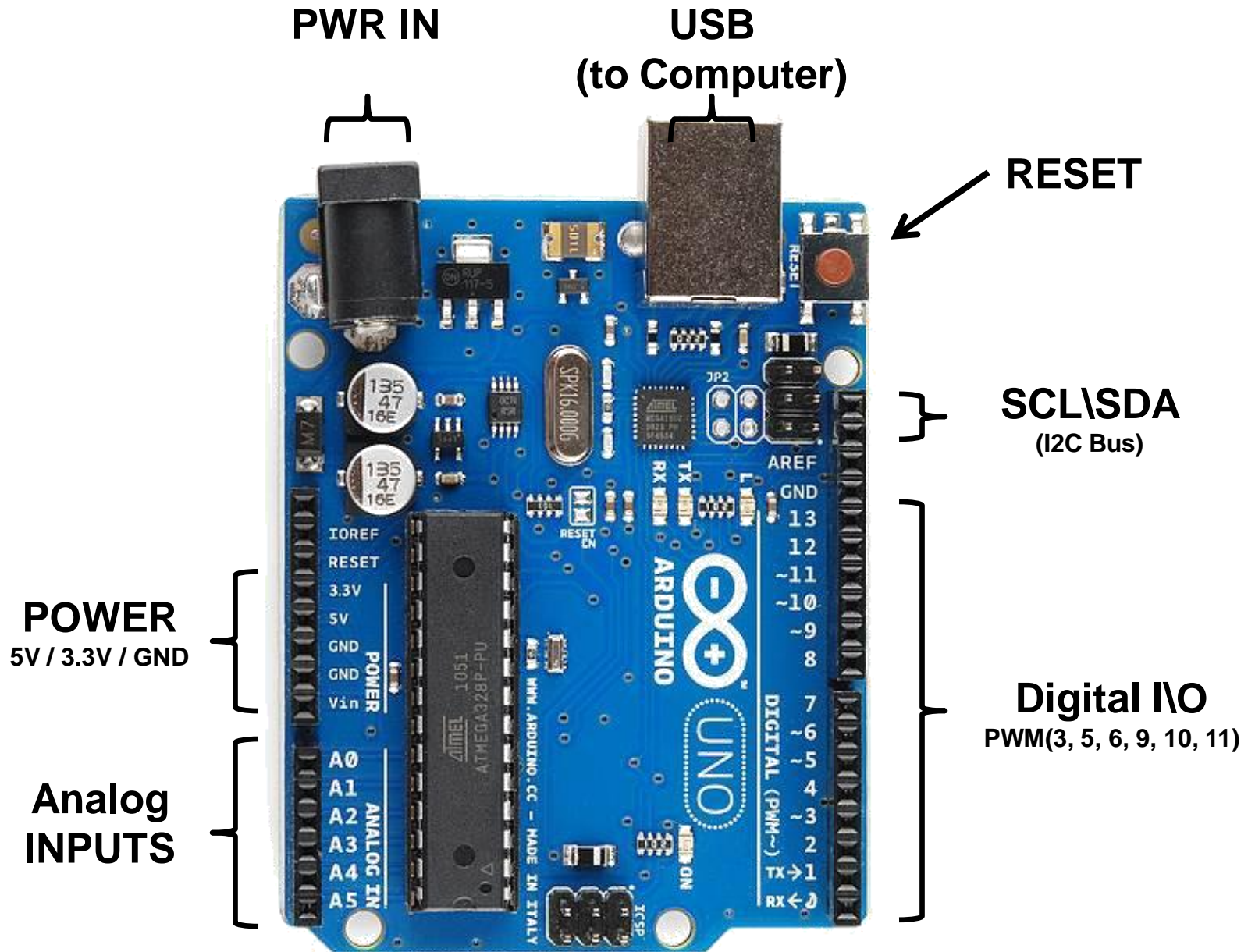


- Go-to gear for artists, hobbyists, students, and anyone with a gadgetry dream.
- Rose out of another formidable challenge: how to teach students to create electronics, fast.





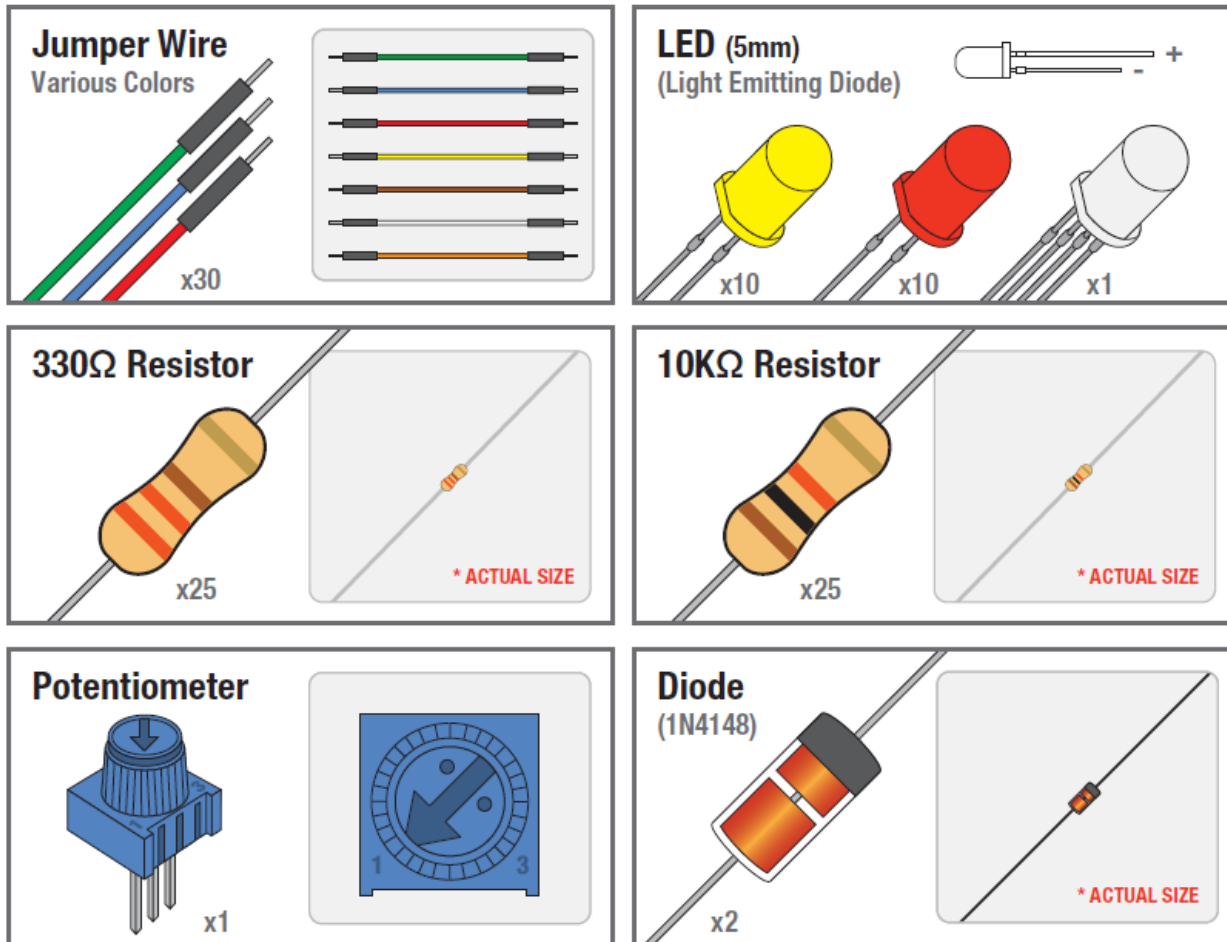




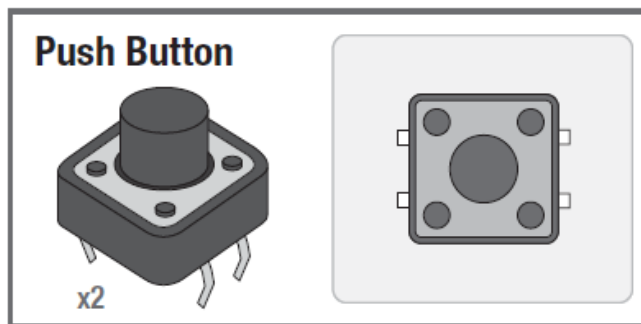
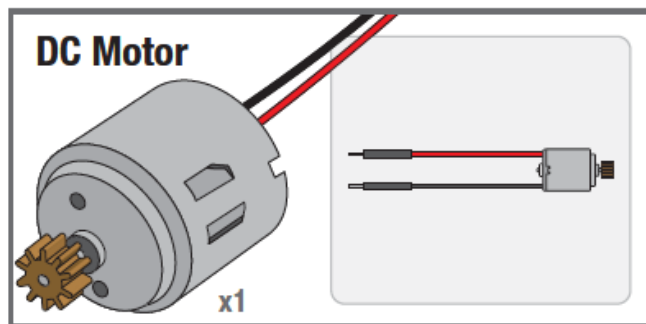
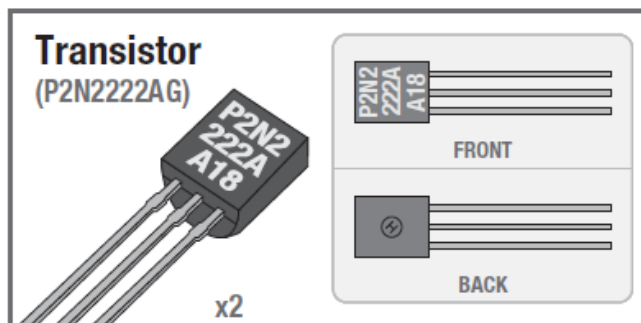
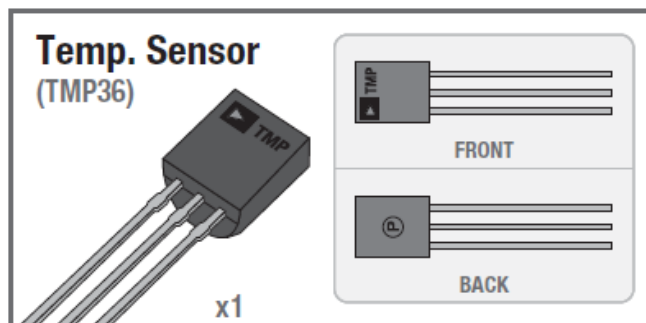
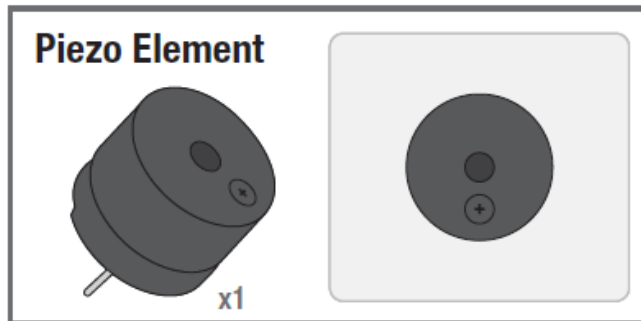
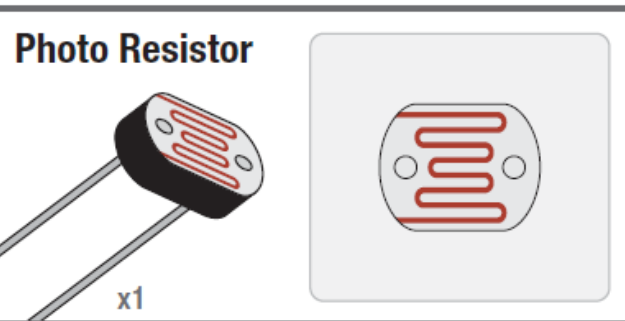
# Why Arduino?

- You'll become a better programmer
- You'll learn a lot about electronics
- Software + Hardware is cooler than Software

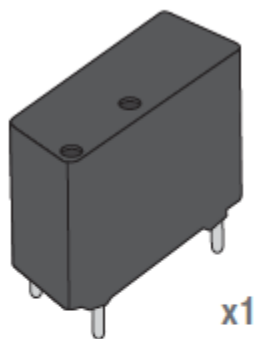
# Parts, Tools, Equipment





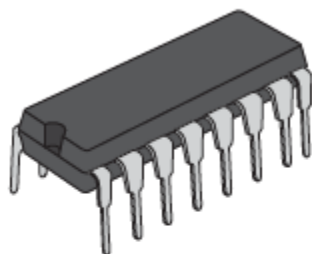


# Relay



x1

# Integrated Circuit (IC)



x1

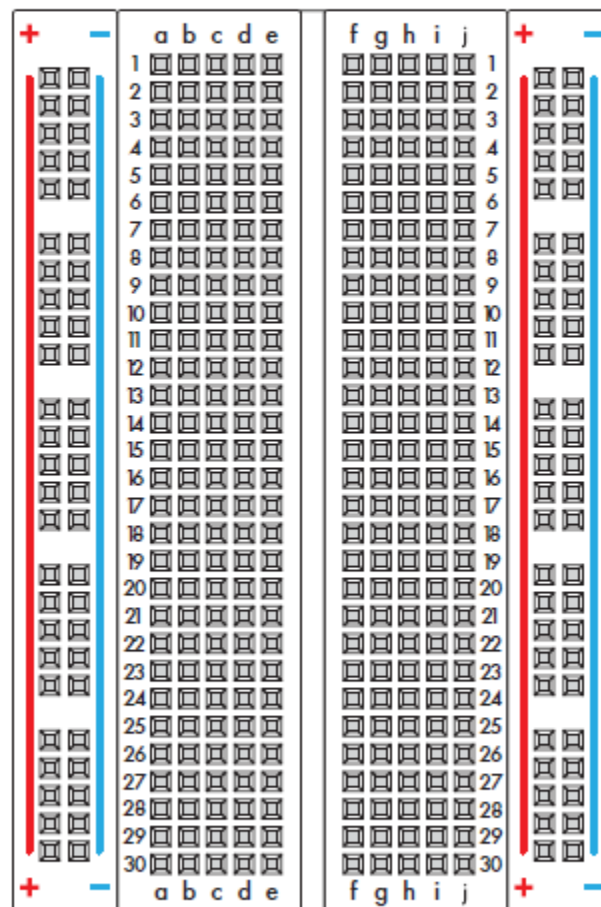
# LCD



x1

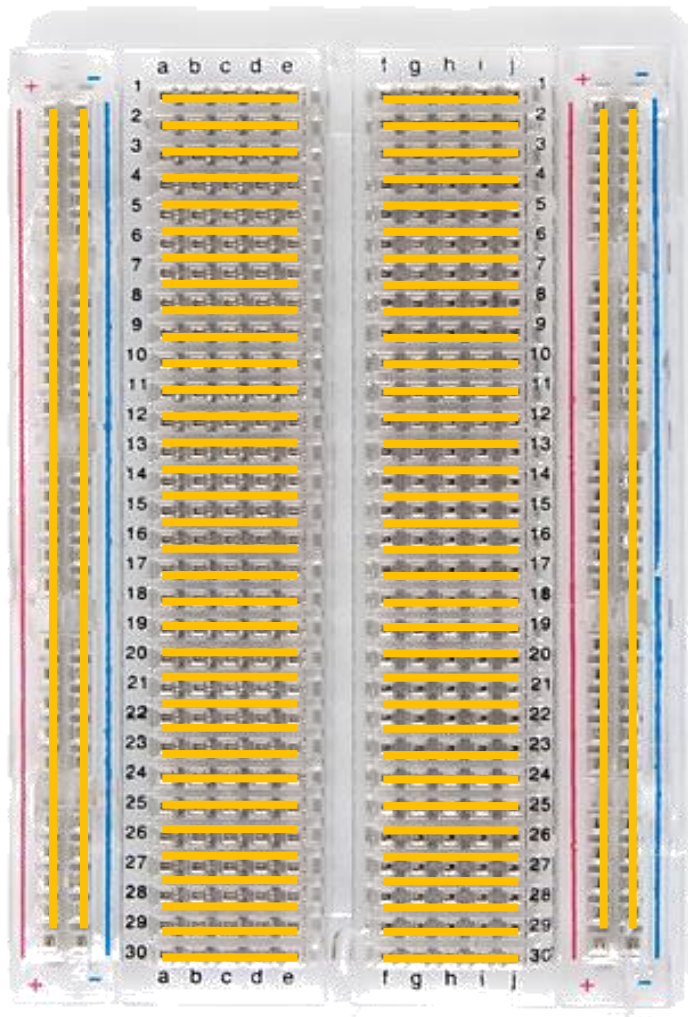
## Breadboard

Standard Solderless (Color may vary)



x1

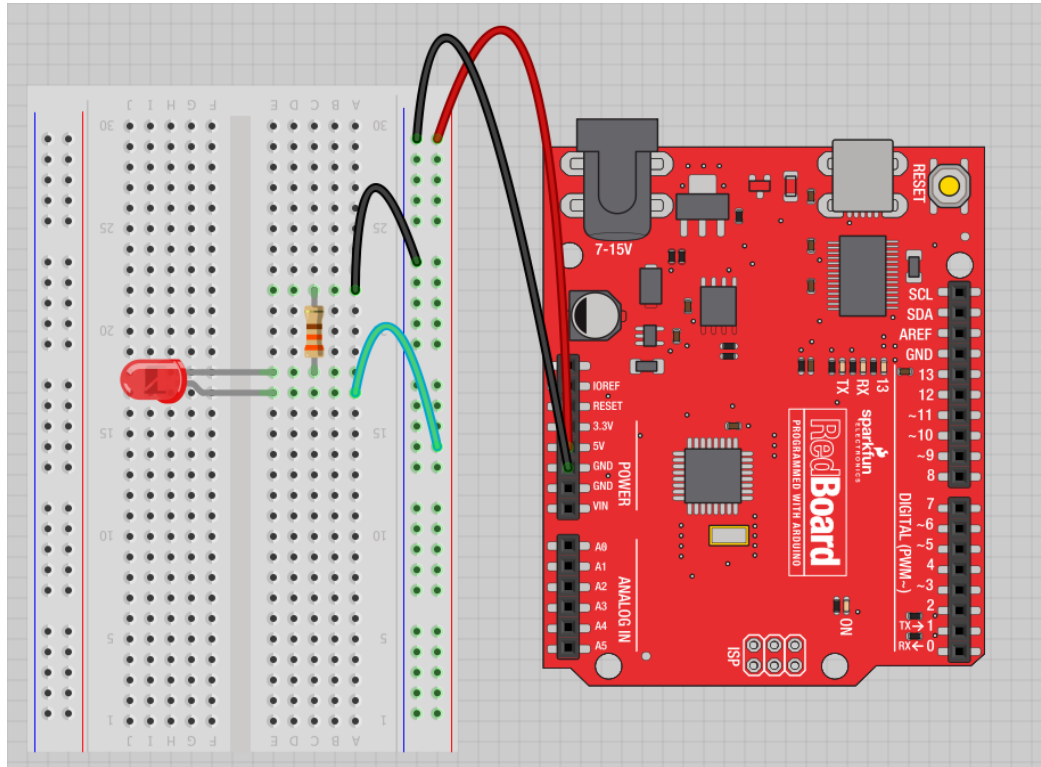
# Solderless Breadboard



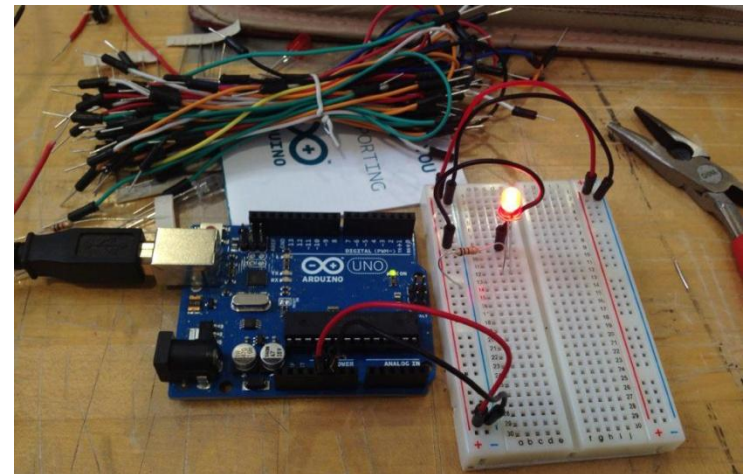
One of the most useful tools in an engineer or Maker's toolkit. The three most important things:

- A breadboard is easier than soldering
- A lot of those little holes are connected, which ones?
- Sometimes breadboards break
- Each row (horiz.) of 5 holes are connected.
- Vertical columns – called power bus are connected vertically

# Fritzing View of Breadboard Circuit



- What happens when you break the circuit?
- What if you wanted to add more than one



# Concepts: INPUT vs. OUTPUT

Referenced from the perspective of the microcontroller (electrical board).

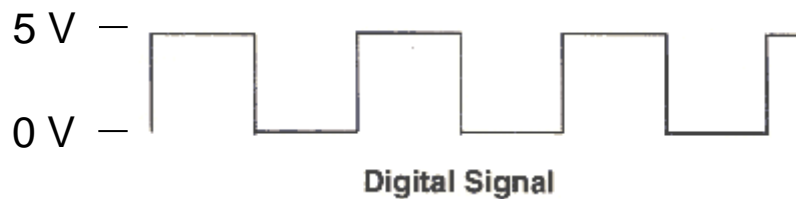
**Inputs** is a signal / information going into the board.

**Output** is any signal exiting the board.

<p><u>Examples</u>: Buttons Switches, Light Sensors, Flex Sensors, Humidity Sensors, Temperature Sensors...</p>	<p><u>Examples</u>: LEDs, DC motor, servo motor, a piezo buzzer, relay, an RGB LED</p>
---	--

# Concepts: Analog vs. Digital

- Microcontrollers are **digital** devices – ON or OFF. Also called – discrete.
- **analog** signals are anything that can be a full range of values. What are some examples? More on this later...





# Open up Arduino



## Hints:

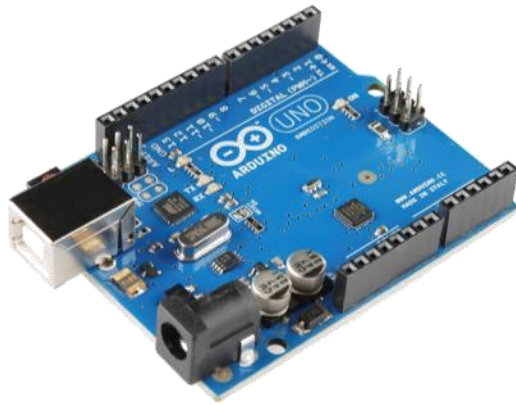
### **For PC Users →**

1. Let the installer copy and move the files to the appropriate locations, or
2. Create a folder under C:\Program Files (x86) called Arduino. Move the entire Arduino program folder here.

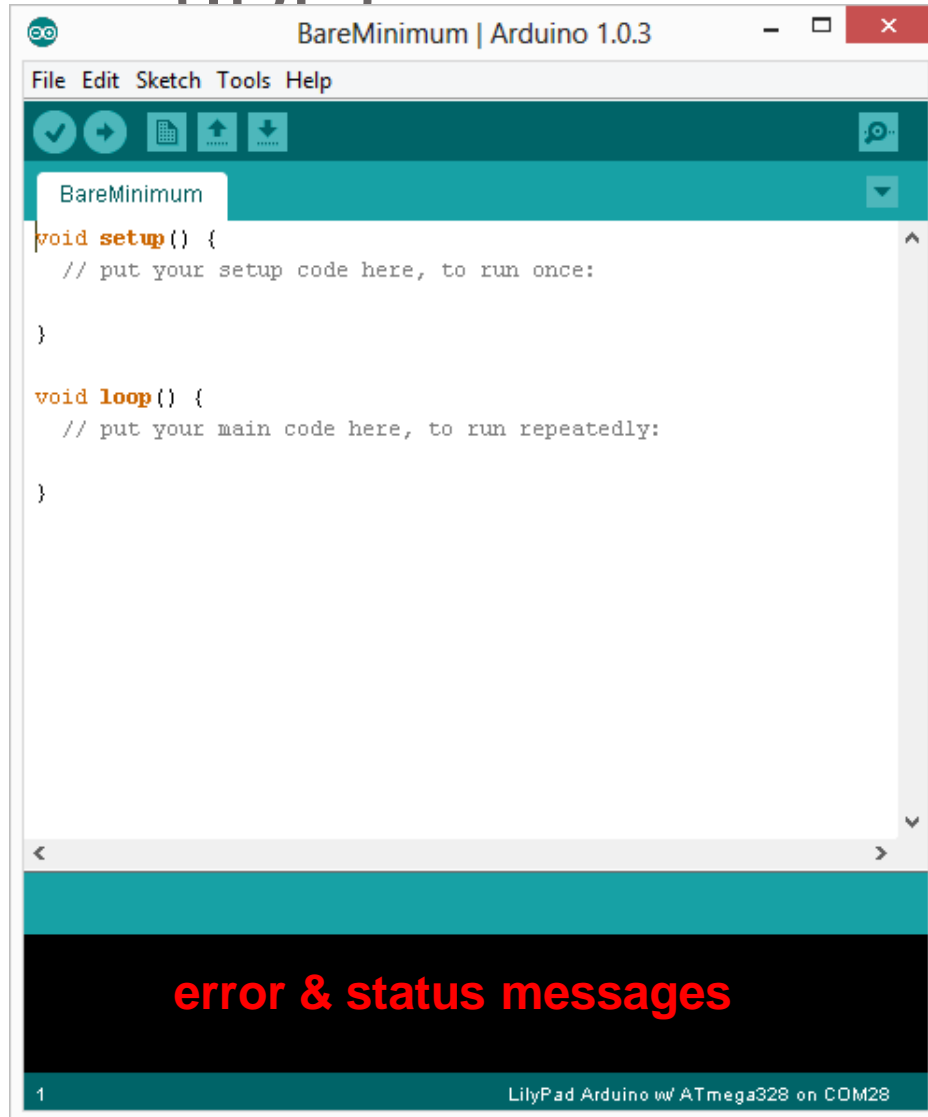
### **For Mac Users →**

1. Move the Arduino executable to the dock for ease of access.
2. Resist the temptation to run these from your desktop.

# Go ahead and plug your board in!



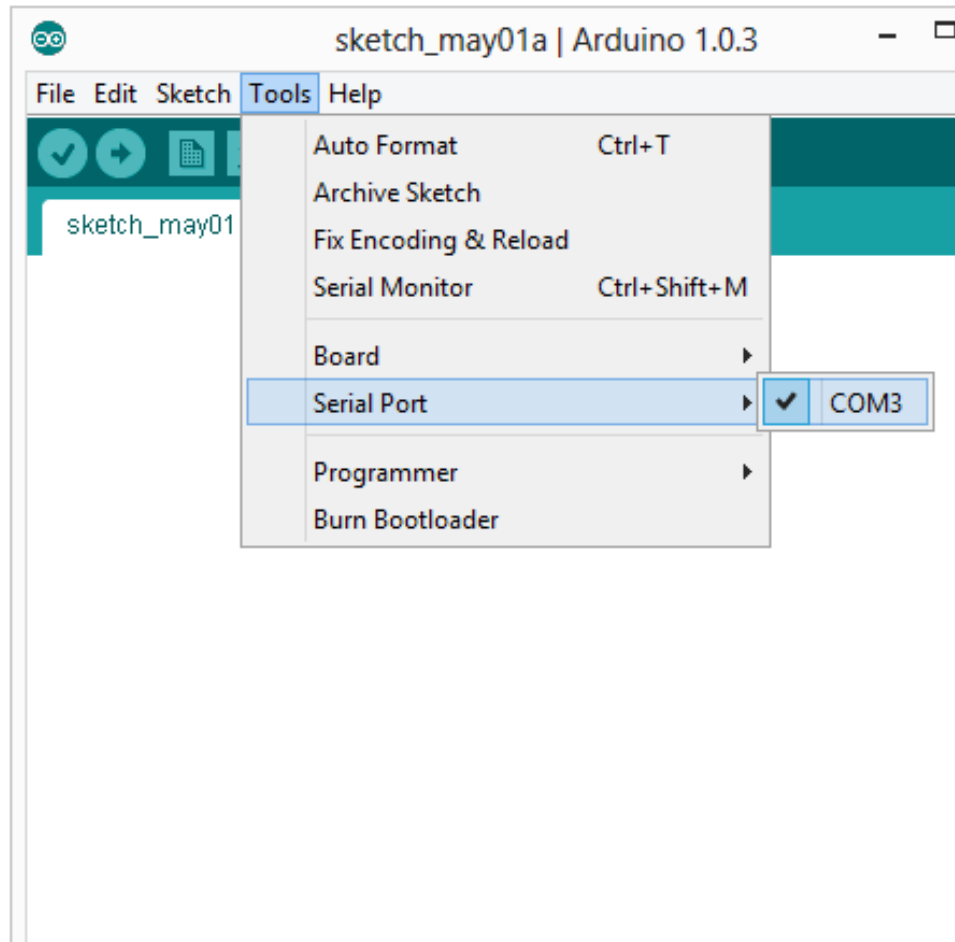
# Integrated Development Environment (IDE)



Two required functions / methods / routines:

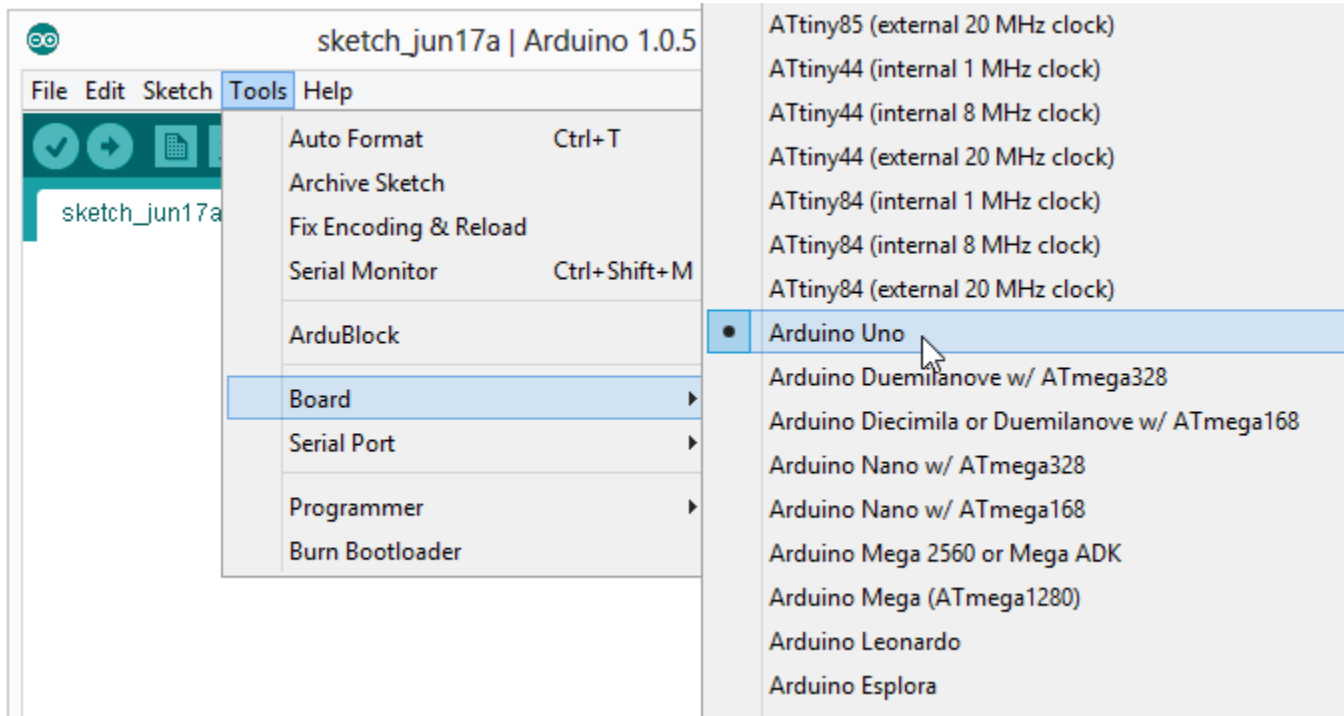
```
void setup()  
{  
  
    // runs once  
  
}
```

```
void loop()  
{  
  
    // repeats  
  
}
```



- Your computer communicates to the Arduino microcontroller via a serial port → through a USB-Serial adapter.
- Check to make sure that the drivers are properly installed.

# Settings: Tools → Board



- Next, double-check that the proper board is selected under the Tools→Board menu.

# Let's get to coding...

## Project #1 – Blink

“Hello World” of Physical Computing

*Pseudo-code – how should this work?*

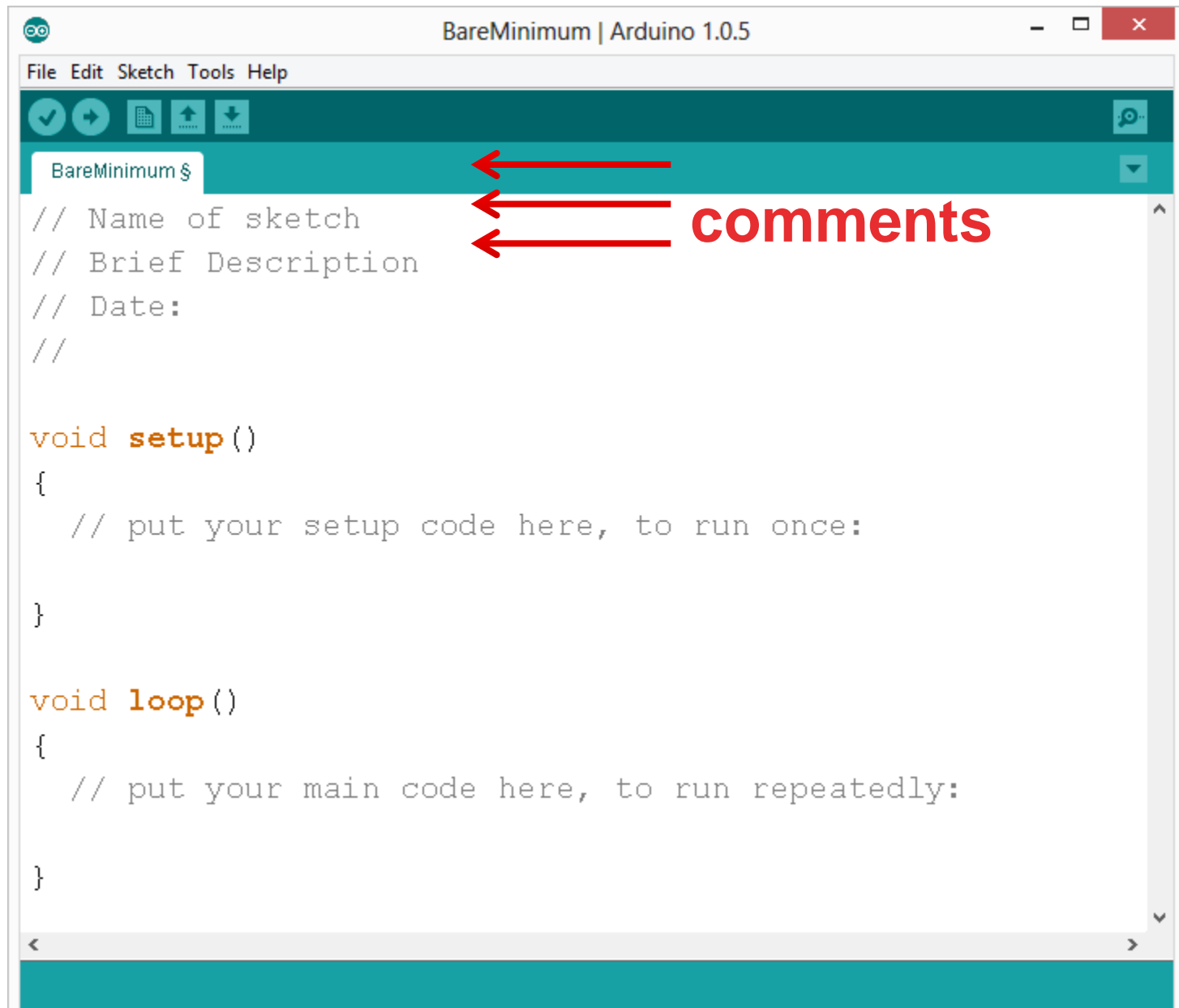




# Comments, Comments, Comments

- Comments are for you – the programmer and your friends...or anyone else human that might read your code.

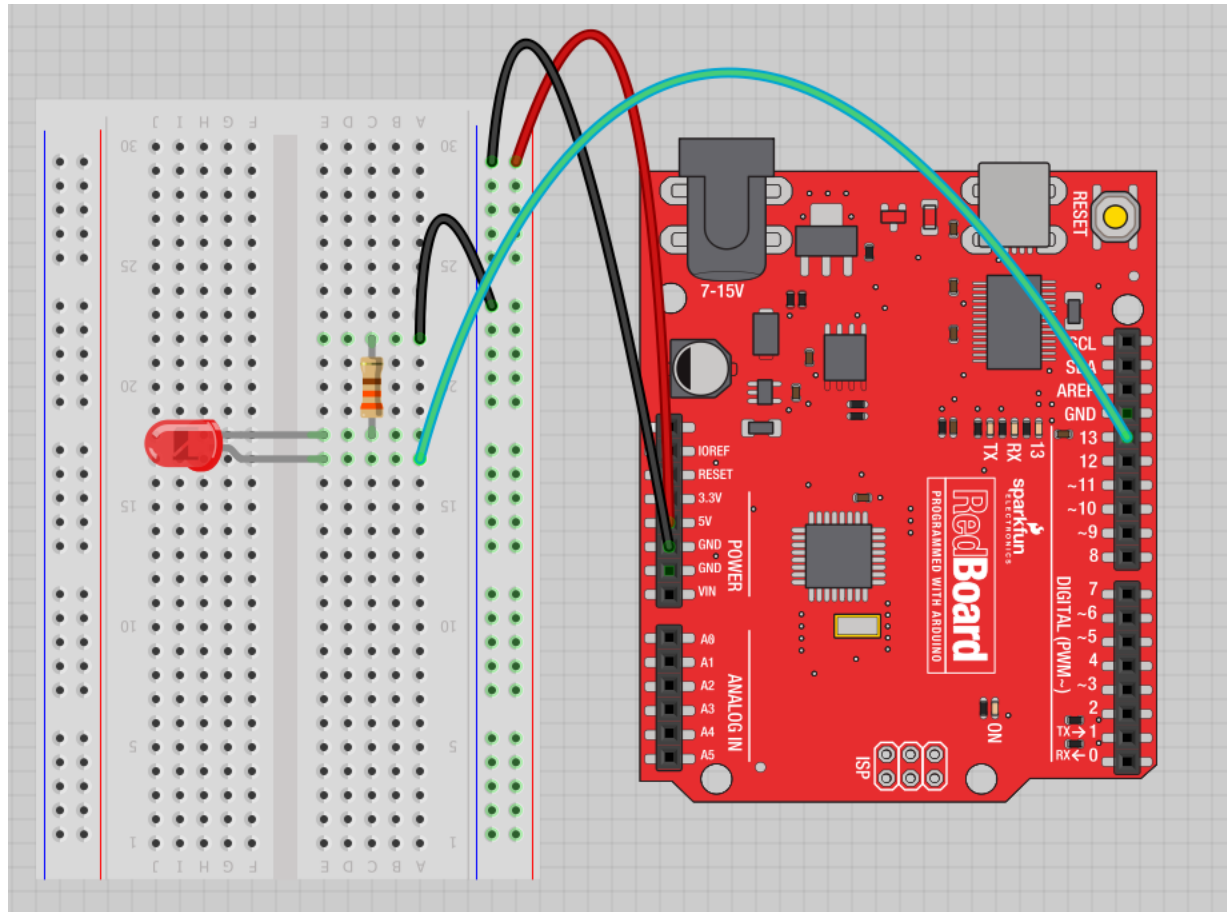
```
// this is for single line comments
// it's good to put a description at the
  top and before anything 'tricky'
/* this is for multi-line comments
   Like this...
   And this....
*/
```



# Three commands to know...

- **pinMode**(pin, INPUT/OUTPUT);
- ex: **pinMode**(13, OUTPUT);
  
- **digitalWrite**(pin, HIGH/LOW);
- ex: **digitalWrite**(13, HIGH);
  
- **delay**(time\_ms);
- ex: **delay**(2500); // delay of 2.5 sec.
  
- **// NOTE: -> commands are CASE-sensitive**

# Project #1: Wiring Diagram



Move the green wire from the power bus to pin 13 (or any other Digital I/O pin on the Arduino board).

Image created in Fritzing

# A few simple challenges

## Let's make LED#13 blink!

- **Challenge 1a** – blink with a 200 ms second interval.
- **Challenge 1b** – blink to mimic a heartbeat
- **Challenge 1c** – find the fastest blink that the human eye can still detect...
  - 1 ms delay? 10 ms delay? 20 ms delay? 30 ms delay???

# Try adding other LEDs

**Try the following extra projects at home:**

- **Can you blink two, three, or four LEDs?  
(Hint: Each LED will need it's own resistor)**
- **Generate your own morse code flashing**
- **How about a Knight Rider, Disco, or Police Light?**



# Programming Concepts: Variables

ProtosnapProMiniExample2 \$

```
// Comments go here
// Written by:  Joesephine Jones
// Date:  April 12, 2013

int sensorValue;
int ledPin;

void setup()
{
    // put your setup code here, to run once:
    int setupVariable;

}

void loop()
{
    // put your main code here, to run repeatedly:
    int loopScopeVariable
}
```

## Variable Scope

- *Global*

- ---

- *Function-level*

# Programming Concepts: Variable Types

- **Variable Types:**



8 bits

byte  
char



16 bits

int  
unsigned int



32 bits

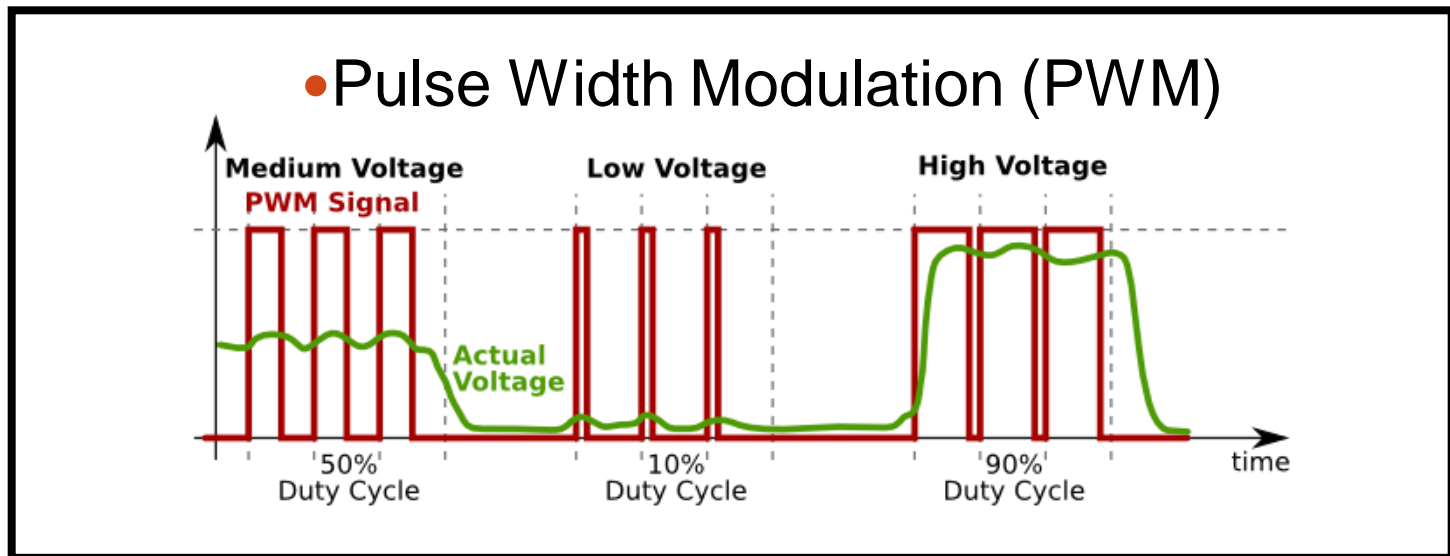
long  
unsigned long  
float

# Fading in and Fading Out (Analog or Digital?)

- A few pins on the Arduino allow for us to modify the output to mimic an analog signal.
- This is done by a technique called:  
Pulse Width Modulation (PWM)

# Concepts: Analog vs. Digital

- To create an analog signal, the microcontroller uses a technique called PWM. By varying the duty cycle, we can mimic an “average” analog voltage.



# Project #2 – Fading

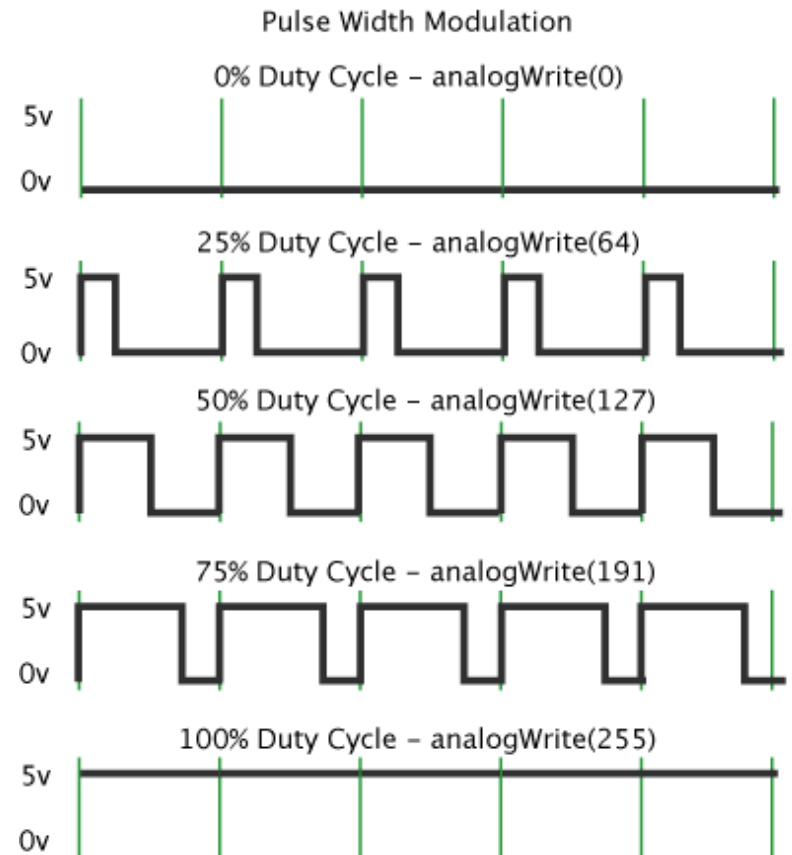
## Introducing a new command...

```
analogWrite(pin, val);
```

**pin** – refers to the OUTPUT pin  
(limited to pins 3, 5, 6, 9, 10, 11.)  
– denoted by a ~ symbol

**val** – 8 bit value (0 – 255).

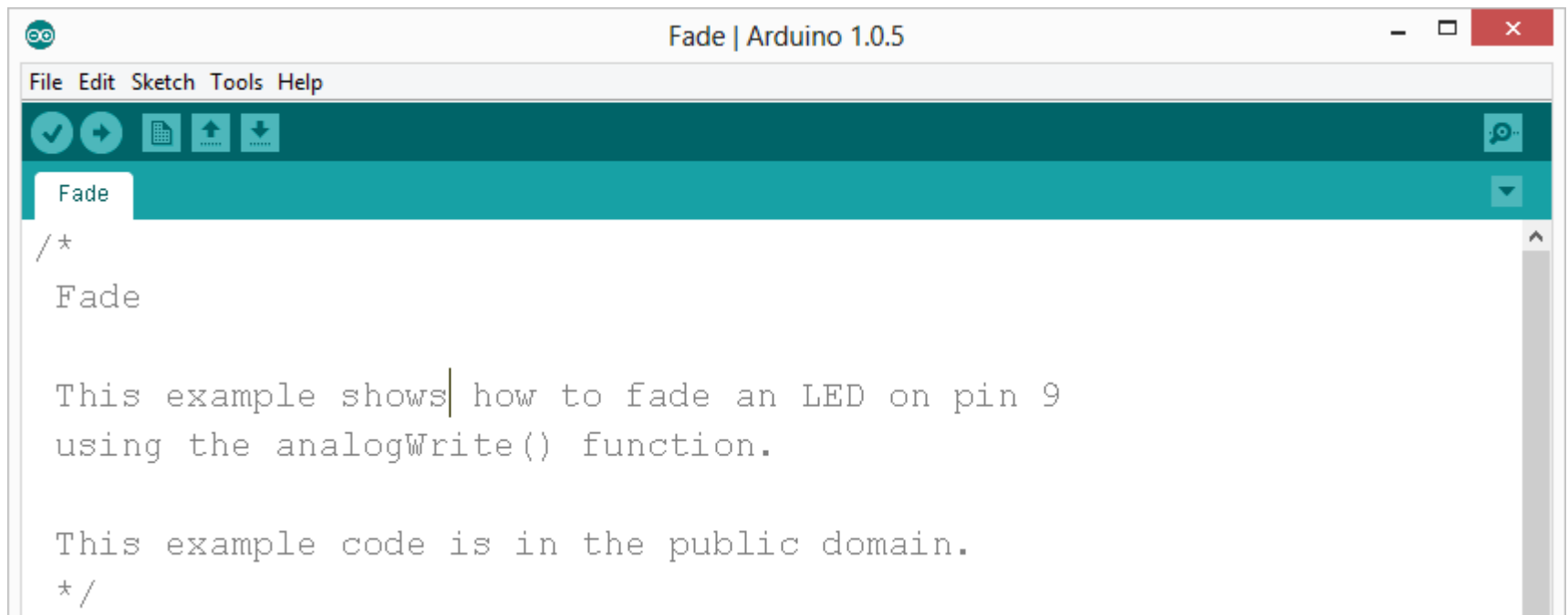
0 => 0V    |    255 => 5V



# Move one of your LED pins over to Pin 9


In Arduino, open up:

File → Examples → 01.Basics → Fade





# Fade - Code Review



```
/*  
Fade  
  
This example shows how to fade an LED on pin 9  
using the analogWrite() function.  
  
This example code is in the public domain.  
*/  
  
int led = 9;           // the pin that the LED is attached to  
int brightness = 0;    // how bright the LED is  
int fadeAmount = 5;    // how many points to fade the LED by
```

# Fade - Code Review

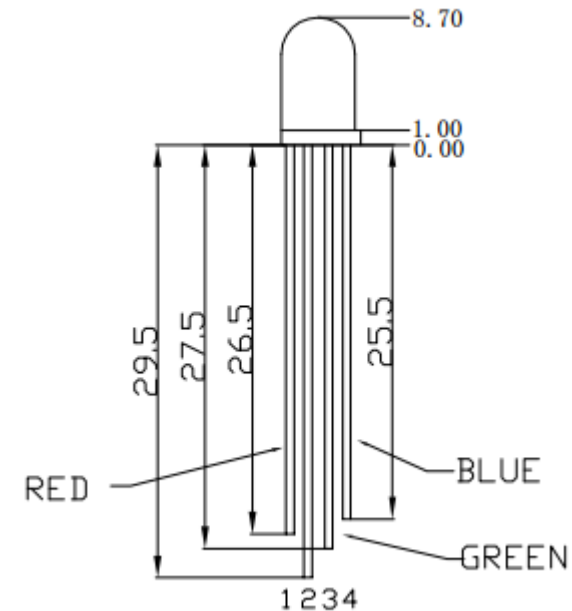
```
void setup() {  
    // declare pin 9 to be an output:  
    pinMode(led, OUTPUT);  
}  
  
// the loop routine runs over and over again forever:  
void loop() {  
    // set the brightness of pin 9:  
    analogWrite(led, brightness);  
  
    // change the brightness for next time through the loop:  
    brightness = brightness + fadeAmount;  
  
    // reverse the direction of the fading at the ends of the fade:  
    if (brightness == 0 || brightness == 255) {  
        fadeAmount = -fadeAmount ;  
    }  
    // wait for 30 milliseconds to see the dimming effect  
    delay(30);  
}
```

# Project# 2 -- Fading

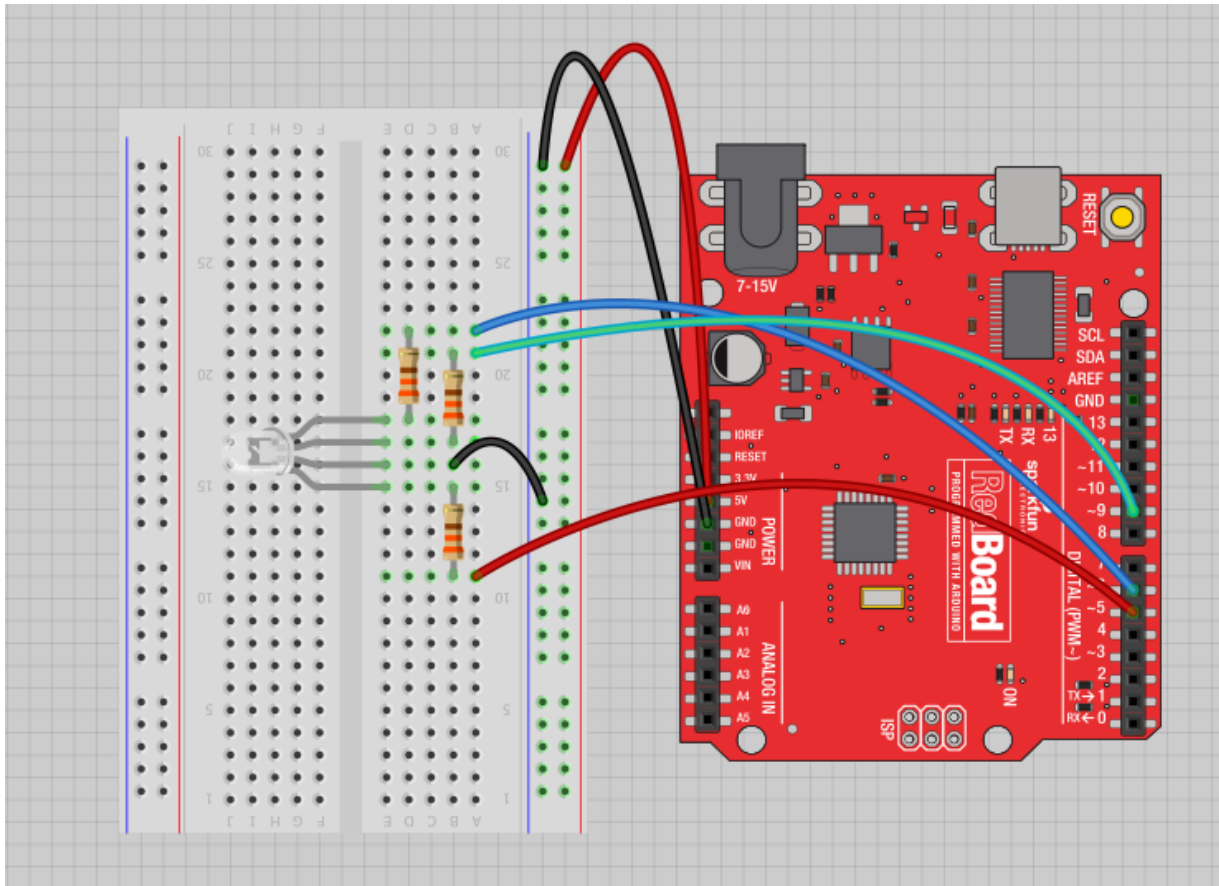
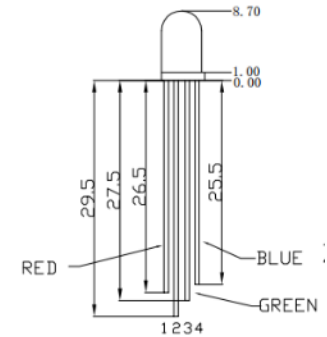
- **Challenge 2a** – Change the rate of the fading in and out. There are at least two different ways to do this – can you figure them out?
- **Challenge 2b** – Use 2 (or more) LEDs – so that one fades in as the other one fades out.
- **Challenge 2c** – Use a red and blue LEDs to create a “police light” effect.



- In the SIK, this is a standard – Common Cathode LED
- This means the negative side of the LED is all tied to Ground.



# Project 3 – RGB LED



- Note: The longest leg of the RGB LED is the Common Cathode. This goes to GND.

Use pins 5, 6, & 9

# How many unique colors can you create?

- $\# \text{ of unique colors} = 256 \cdot 256 \cdot 256$   
 $= 16,777,216 \text{ colors!}$



Use Colorpicker.com or experiment on your own.

Pick out a few colors that you want to try re-creating for a lamp or lighting display...

Play around with this with the `analogWrite()` command.

# RGB LED Color Mixing – Challenge 3a

```
• int redPin = 5;
• int greenPin = 6;
• int bluePin = 9;

• void setup()
• {
•     pinMode(redPin, OUTPUT);
•     pinMode(greenPin, OUTPUT);
•     pinMode(bluePin, OUTPUT);
• }
```

# RGB LED Color Mixing – Challenge 3a

```
• void loop()  
• {  
•     analogWrite(redPin, 255);  
•     analogWrite (greenPin, 255);  
•     analogWrite (bluePin, 255);  
• }
```



# Project# 3 – RGB LEDs

- **Challenge 3a** – Hookup RGB LED to pins 5,6,9 and the longest pin on the LED in series with a 220 ohm resistor then connected to the GND rail on the breadboard.
- Upload the example code, find out which pins are red, green, blue, and change values to brightness as desired.

# Coding Basics

Exploring code and communication

# Overview

- Variables, Statements, Loops, and Functions
- Serial Communication
- Nerd Texting
- Piezo Buzzers

# Programming Concepts: Variables

ProtosnapProMiniExample2 \$

```
// Comments go here
// Written by:  Joesephine Jones
// Date:  April 12, 2013

int sensorValue;
int ledPin;

void setup()
{
    // put your setup code here, to run once:
    int setupVariable;

}

void loop()
{
    // put your main code here, to run repeatedly:
    int loopScopeVariable
}
```

## Variable Scope

- *Global*

- ---

- *Function-level*

# Scope

```
int gPWMval;  // any function will see this variable

void setup()
{
    // ...
}

void loop()
{
    int i;      // "i" is only "visible" inside of "loop"
    float f;    // "f" is only "visible" inside of "loop"
    // ...

    for (int j = 0; j <100; j++){
        // variable j can only be accessed inside the for-loop brackets
    }
}
```

We set it equal to the function  
`digitalRead(pushButton)`

We declare a  
variable as an  
integer.

The function `digitalRead()` will return  
the value 1 or 0, depending on whether  
the button is being pressed or not  
being pressed.

```
int buttonState = digitalRead(pushButton);
```

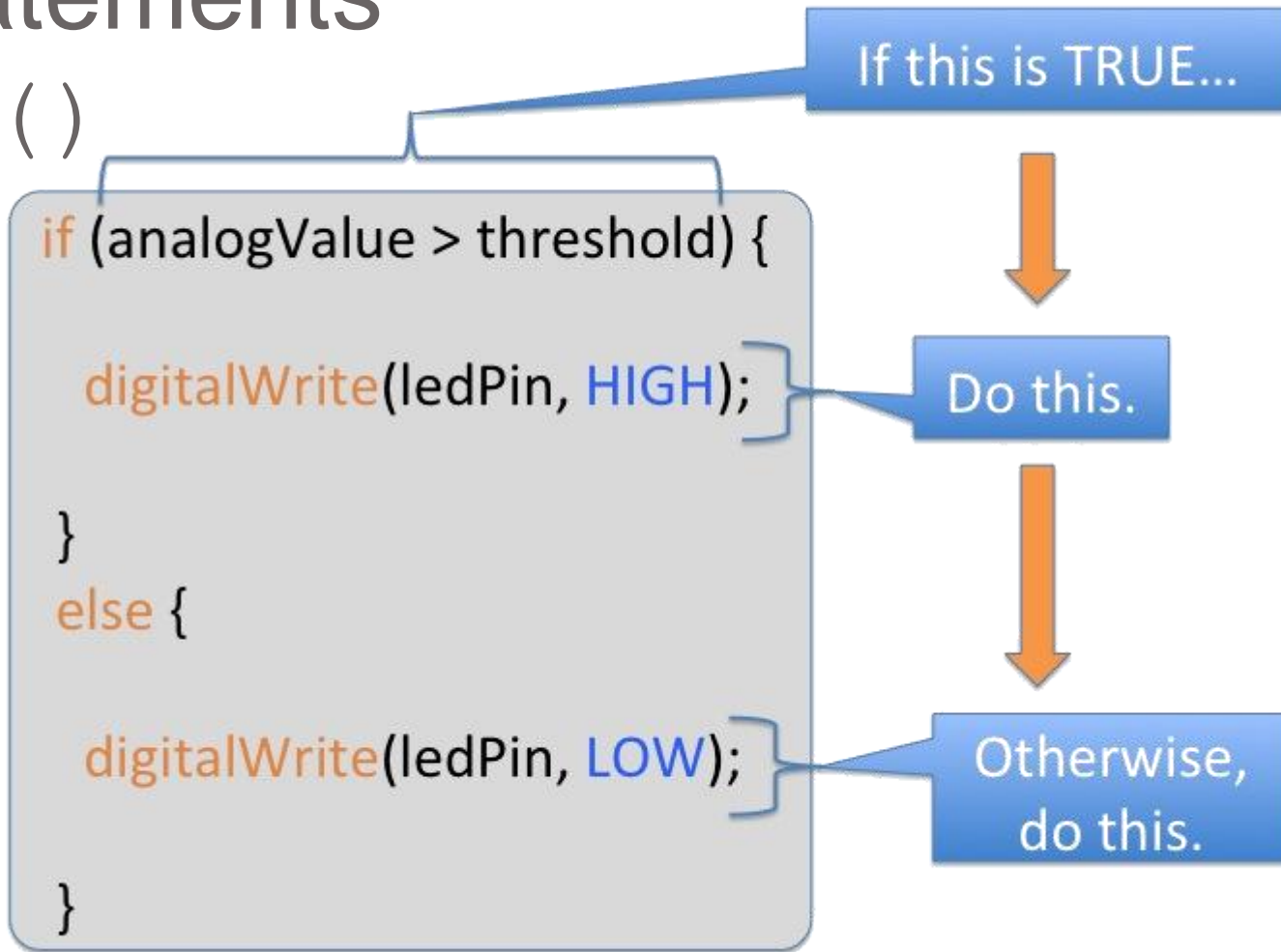
We name it  
`buttonState`

Recall that the `pushButton`  
variable stores the number 2

The value 1 or 0 will be saved in  
the variable `buttonState`.

# Programming: Conditional Statements

`if ( )`







# Boolean Operators

<Boolean>	Description
( ) == ( )	is equal?
( ) != ( )	is not equal?
( ) > ( )	greater than
( ) >= ( )	greater than or equal
( ) < ( )	less than
( ) <= ( )	less than or equal

```
if (pinFiveInput < 500)
{
    // do Thing A
}
else if (pinFiveInput >= 1000)
{
    // do Thing B
}
else
{
    // do Thing C
}
```

# For Loops

- The for statement is used to repeat a block of statements enclosed in curly braces. An increment counter is usually used to increment and terminate the loop. The for statement is useful for any repetitive operation, and is often used in combination with arrays to operate on collections of data/pins.

- There are three parts to the for loop header:

```
for (initialization; condition; increment) {  
    //statement(s);  
}
```

parenthesis

declare variable (optional)

initialize

test

increment or decrement

```
for(int x = 0; x < 100; x++){  
    println(x); // prints 0 to 99  
}
```

The diagram illustrates the four components of a for loop: 'declare variable (optional)' points to 'int x', 'initialize' points to '= 0', 'test' points to '< 100', and 'increment or decrement' points to 'x++'. A yellow arrow labeled 'parenthesis' spans the entire for loop header.

```
// Dim an LED using a PWM pin  
int PWMpin = 10; // LED in series with 470 ohm resistor on pin 10  
  
void setup()  
{  
    // no setup needed  
}  
  
void loop()  
{  
    for (int i=0; i <= 255; i++){  
        analogWrite(PWMpin, i);  
        delay(10);  
    }  
}
```

# While Loops

- while loops will loop continuously, and infinitely, until the expression inside the parenthesis, () becomes false. Something must change the tested variable, or the while loop will never exit. This could be in your code, such as an incremented variable, or an external condition, such as testing a sensor.

```
var = 0;
while(var < 200){
    // do something repetitive 200 times
    var++;
}
```

# Functions

- Segmenting code into functions allows a programmer to create modular pieces of code that perform a defined task and then return to the area of code from which the function was "called". The typical case for creating a function is when one needs to perform the same action multiple times in a program.

```
var FunctionName(param1, param2){  
    //do some stuff  
    return result;  
}
```

# Functions

- Functions help the programmer stay organized. Often this helps to conceptualize the program.
- Functions codify one action in one place so that the function only has to be thought out and debugged once.
- This also reduces chances for errors in modification, if the code needs to be changed.
- Functions make the whole sketch smaller and more compact because sections of code are reused many times.
- They make it easier to reuse code in other programs by making it more modular, and as a nice side effect, using functions also often makes the code more readable.

# Functions

## Anatomy of a C function

```
void setup(){  
  Serial.begin(9600);  
}
```

```
void loop() {  
  int i = 2;  
  int j = 3;  
  int k;
```

```
  k = myMultiplyFunction(i, j); // k now contains 6  
  Serial.println(k);  
  delay(500);  
}
```

```
int myMultiplyFunction(int x, int y){  
  int result;  
  result = x * y;  
  return result;  
}
```

Datatype of data returned,  
any C datatype.

"void" if nothing is returned.

Parameters passed to  
function, any C datatype.

Function name

int myMultiplyFunction(int x, int y){

int result;

Return statement,  
datatype matches  
declaration.

result = x \* y;

return result;

Curly braces required.

}



# Project# 4 – Fading using For Loop

- **Challenge 4a** – use a for loop instead to fade up and down the led

There is more than one way to code this

# Serial Communication

- Serial communication on pins TX/RX uses TTL logic levels (5V or 3.3V depending on the board). Don't connect these pins directly to an RS232 serial port; they operate at +/- 12V and can damage your Arduino board.
- Serial is used for communication between the Arduino board and a computer or other devices. All Arduino boards have at least one serial port (also known as a UART or USART): Serial. It communicates on digital pins 0 (RX) and 1 (TX) as well as with the computer via USB. Thus, if you use these functions, you cannot also use pins 0 and 1 for digital input or output.

# Serial Communication

- You can use the Arduino environment's built-in serial monitor to communicate with an Arduino board. Click the serial monitor button in the toolbar and select the same baud rate used in the call to `begin()`.

# Serial Communication

- Open the Basics > Digital Read Serial example
- Connect a jumper wire to Pin 2. After starting the program, open the serial monitor and switch the other side of the jumper wire from the GND port to the 5V port.
- What do you notice when switching back and forth?

# Software Serial

- Allows for more serial ports

```
#include <SoftwareSerial.h>

SoftwareSerial mySerial(10, 11); // RX, TX

void setup() {
  // Open serial communications and wait for port to open:
  Serial.begin(57600);
  while (!Serial) {
    ; // wait for serial port to connect. Needed for native USB port only
  }

  Serial.println("Goodnight moon!");

  // set the data rate for the SoftwareSerial port
  mySerial.begin(4800);
  mySerial.println("Hello, world?");
}

void loop() { // run over and over
  if (mySerial.available()) {
    Serial.write(mySerial.read());
  }
  if (Serial.available()) {
    mySerial.write(Serial.read());
  }
}
```

# Project #5 – Nerd Texting

- Use Serial Communication to talk with your friend
- Open the SoftwareSerial > SoftwareSerialExample program
- Change Baud rate to 9600 for both Serial and Software Serial
- Connect jumper wires to pins 10/11 (reverse for friend)
- In Serial Monitor, choose 9600 Baud and Newline
- Type messages in the input box and hit send

# Piezo Buzzer and tone()

- Generates a square wave of the specified frequency (and 50% duty cycle) on a pin. A duration can be specified, otherwise the wave continues until a call to noTone(). The pin can be connected to a piezo buzzer or other speaker to play tones.
- Only one tone can be generated at a time. If a tone is already playing on a different pin, the call to tone() will have no effect. If the tone is playing on the same pin, the call will set its frequency.
- Use of the tone() function will interfere with PWM output on pins 3 and 11 (on boards other than the Mega).

# Project #6 – Piezo Tone

- Challenge 6a – Hook up piezo buzzer to pin 8. Make sure the longer lead is the pin going to 8. The shorter one should go to GND. Open the challenge 6a example and play around with it.
- Explore the different tone examples in the “digital” library examples



# Sensors

how robots sense the environment

# Overview

- What are sensors and why do we need them
- Tilt Sensor
- Light Sensor
- Ultrasonic Distance (Ping) Sensor

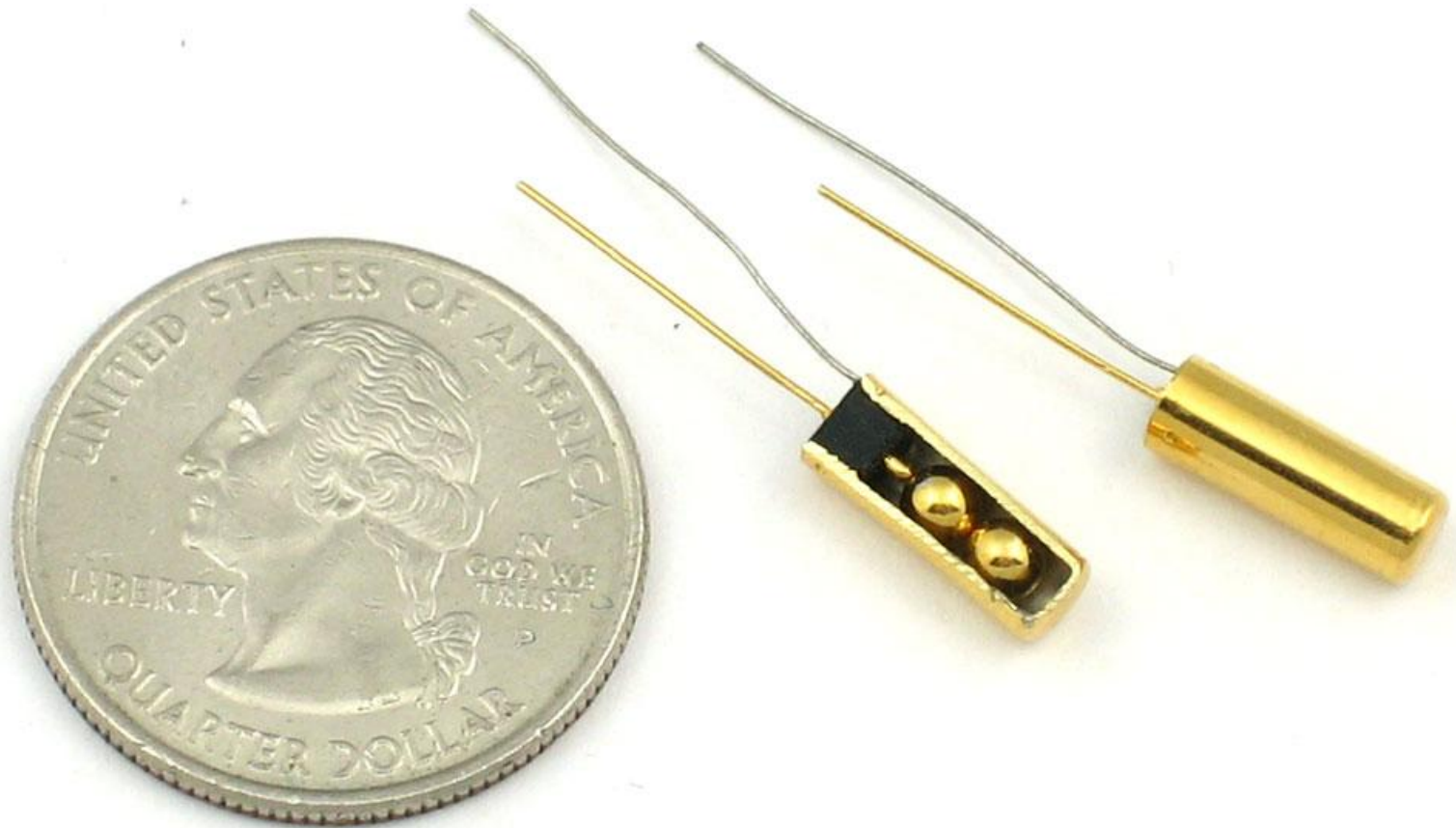
# Sensors



- A sensor is a device that detects and responds to some type of input from the physical environment. The specific input could be light, heat, motion, moisture, pressure, or any one of a great number of other environmental phenomena.



# Tilt Sensor

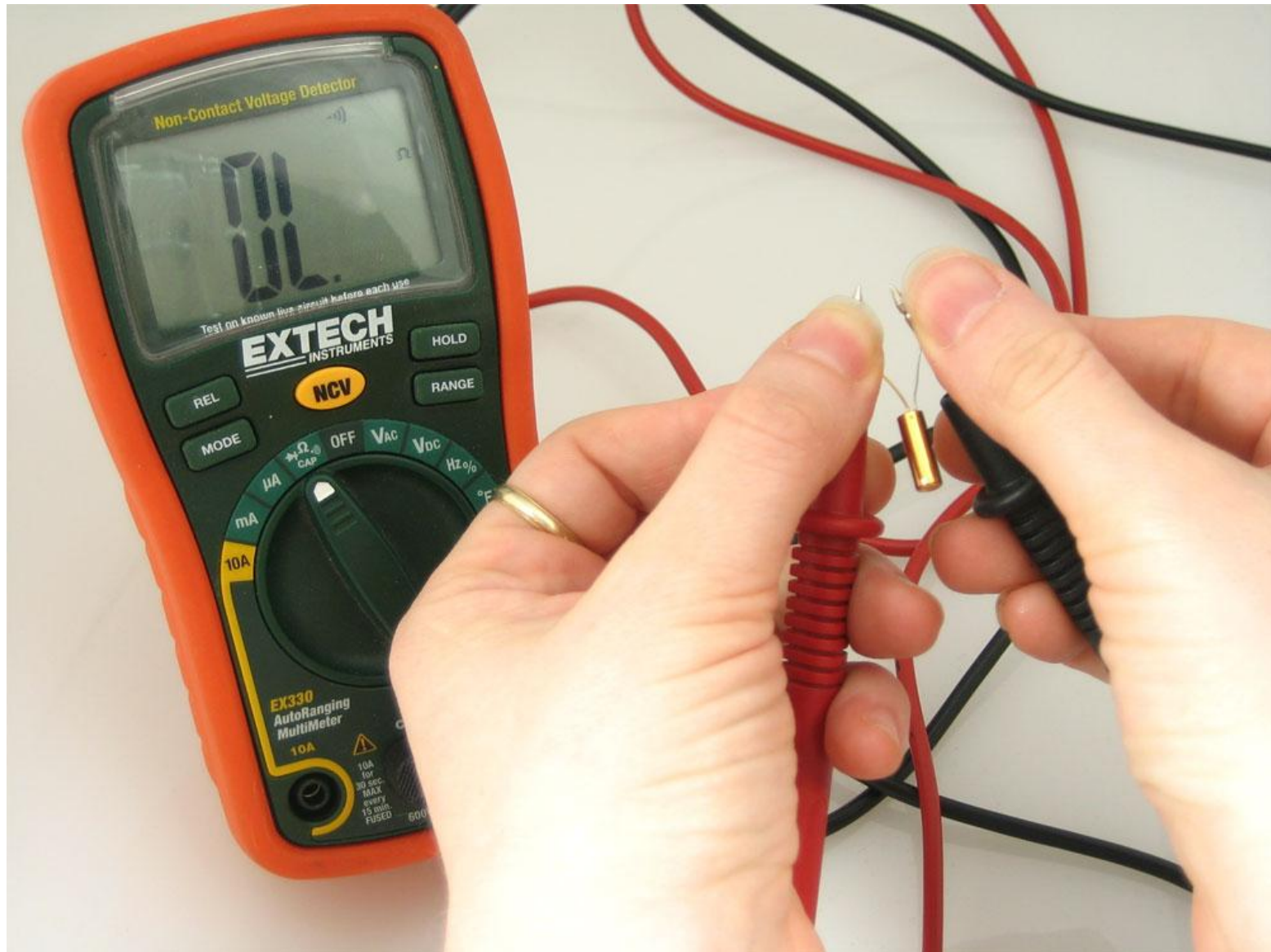




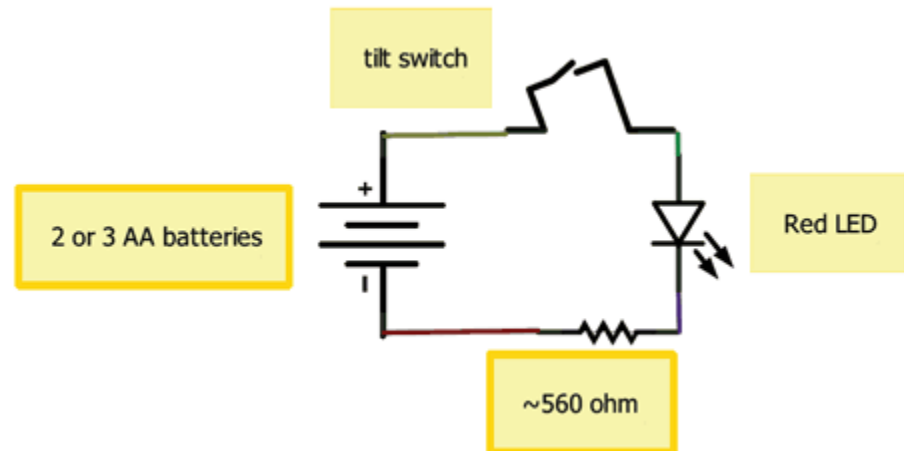
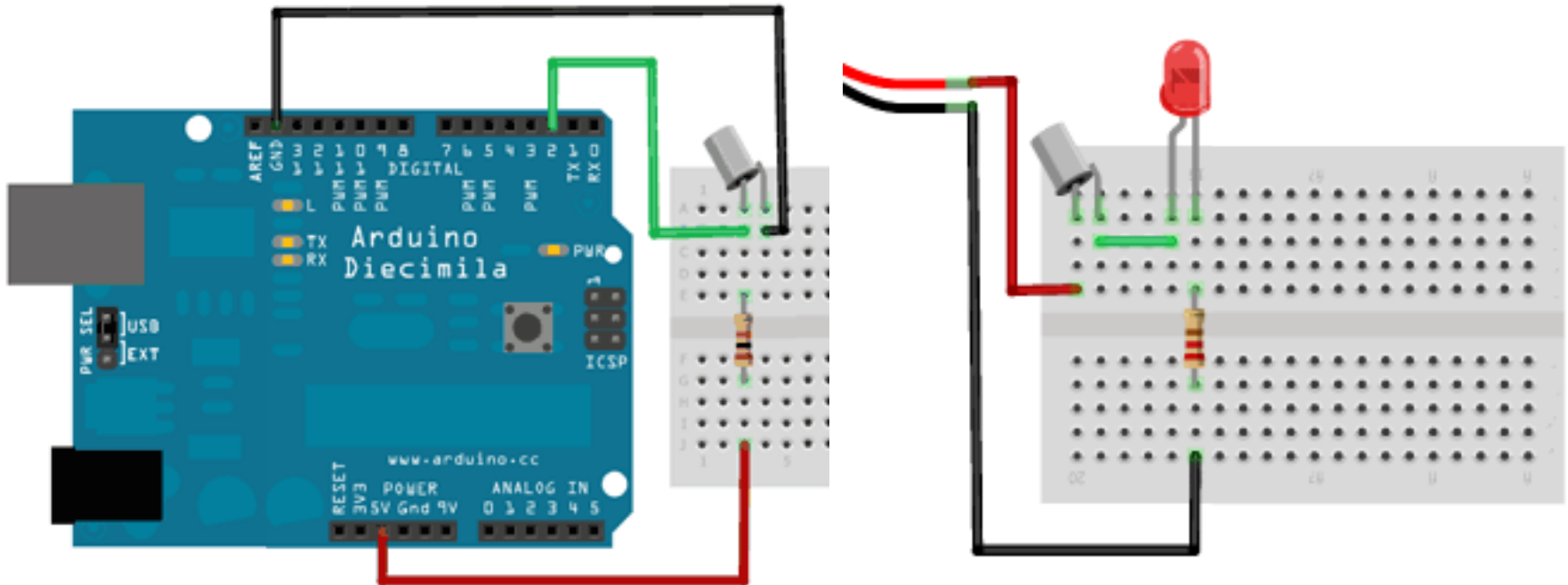
# Tilt Sensor

- Tilt sensors allow you to detect orientation or inclination. They are small, inexpensive, low-power and easy-to-use. If used properly, they will not wear out. Their simplicity makes them popular for toys, gadgets and appliances. Sometimes they are referred to as "mercury switches", "tilt switches" or "rolling ball sensors" for obvious reasons.
- They are usually made by a cavity of some sort (cylindrical is popular, although not always) and a conductive free mass inside, such as a blob of mercury or rolling ball. One end of the cavity has two conductive elements (poles). When the sensor is oriented so that that end is downwards, the mass rolls onto the poles and shorts them, acting as a switch throw.

# Testing a Tilt Sensor



# Tilt Sensor – TiltSwitchLED Code

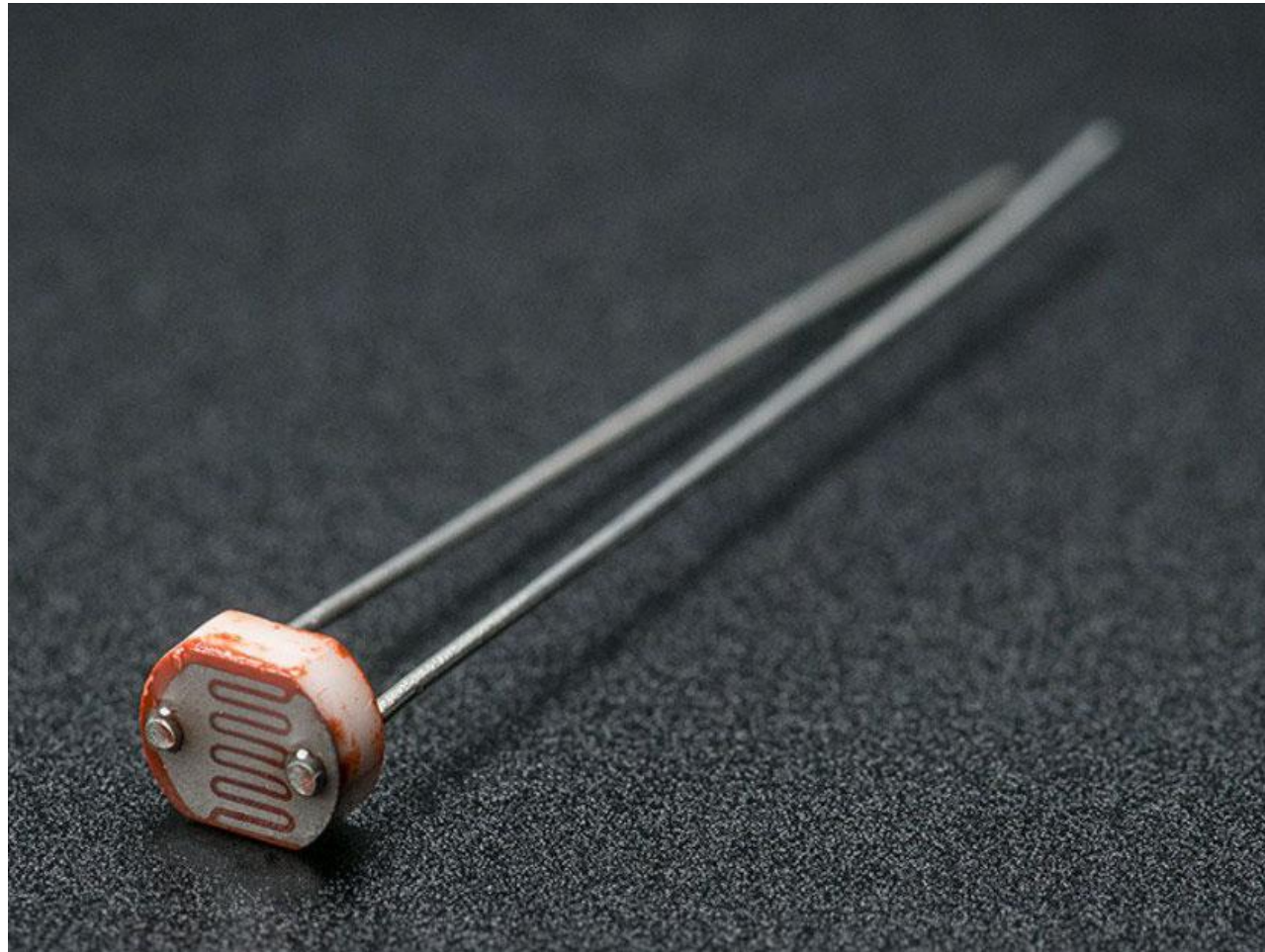




# Project #7 – Tilt Switch and LED

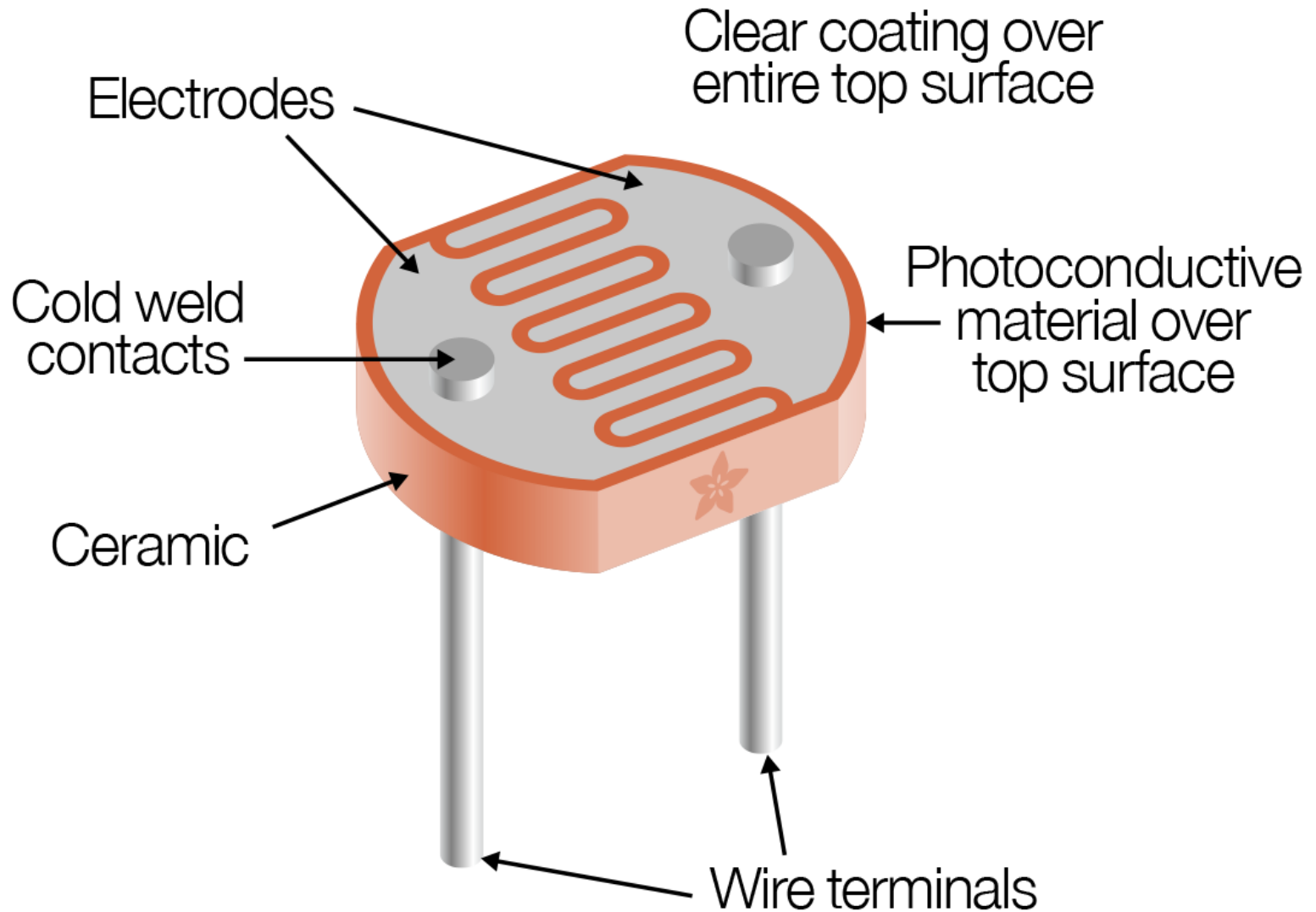
- Challenge 7a – Use a 10 kOhm resistor to create a “pull-up resistor.” Run a jumper wire from pin 2 to tilt switch, and place the resistor in the circuit going to the 5V rail. Connect another jumper wire from the other Tilt Switch pin to the GND rail on the breadboard.
- Open the example, upload code, and play with the circuit. LED should turn off when tipped upside down.

# Light Sensor / Photocell

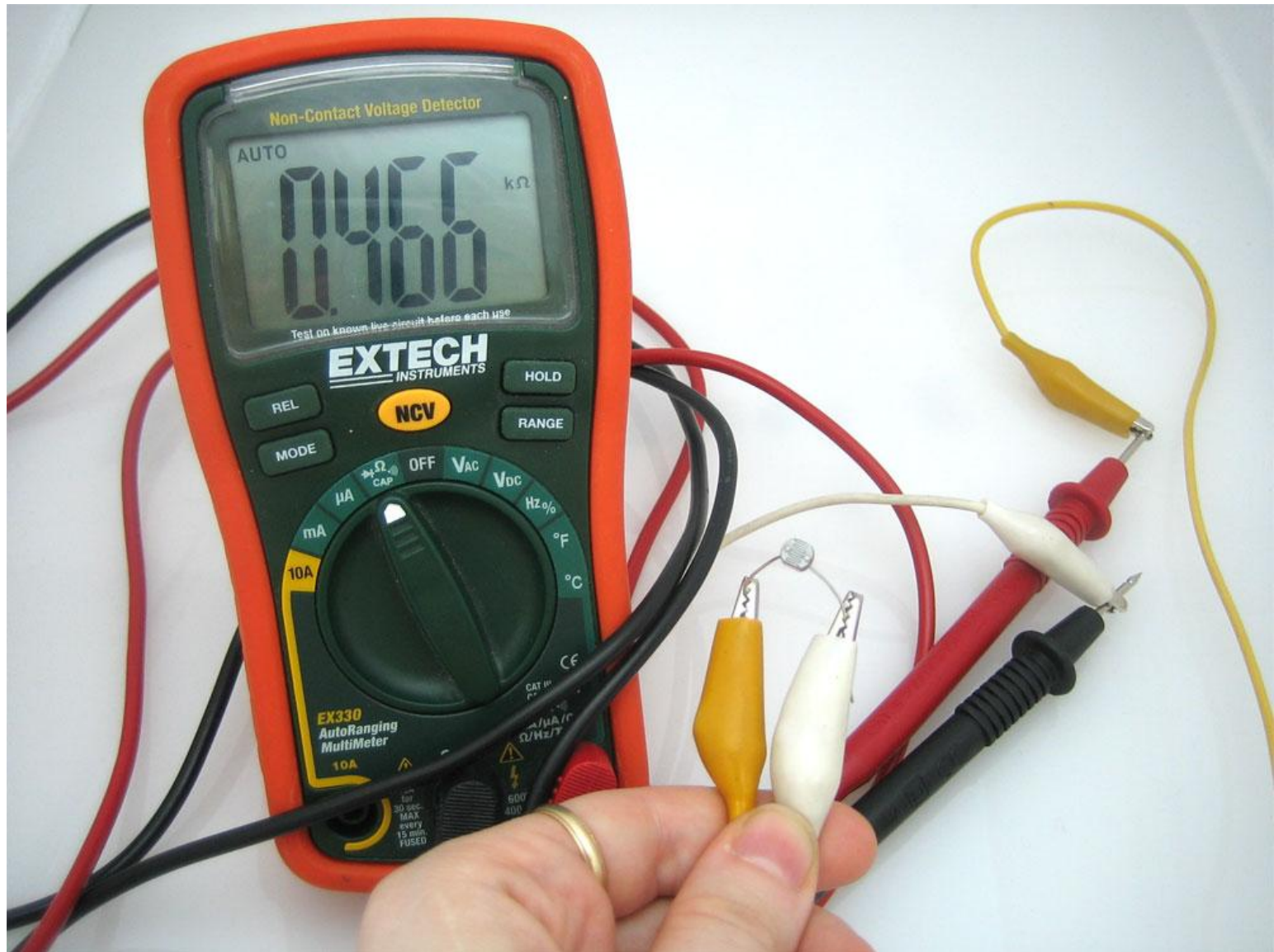


- Photocells are sensors that allow you to detect light. They are small, inexpensive, low-power, easy to use and don't wear out. For that reason they often appear in toys, gadgets and appliances. They are often referred to as CdS cells (they are made of Cadmium-Sulfide), light-dependent resistors (LDR), and photoresistors.
- Photocells are basically a resistor that changes its resistive value (in ohms  $\Omega$ ) depending on how much light is shining onto the squiggly face. They are very low cost, easy to get in many sizes and specifications, but are very inaccurate. Each photocell sensor will act a little differently than the other, even if they are from the same batch. The variations can be really large, 50% or higher! For this reason, they shouldn't be used to try to determine precise light levels in lux or millicandela. Instead, you can expect to only be able to determine basic light changes.

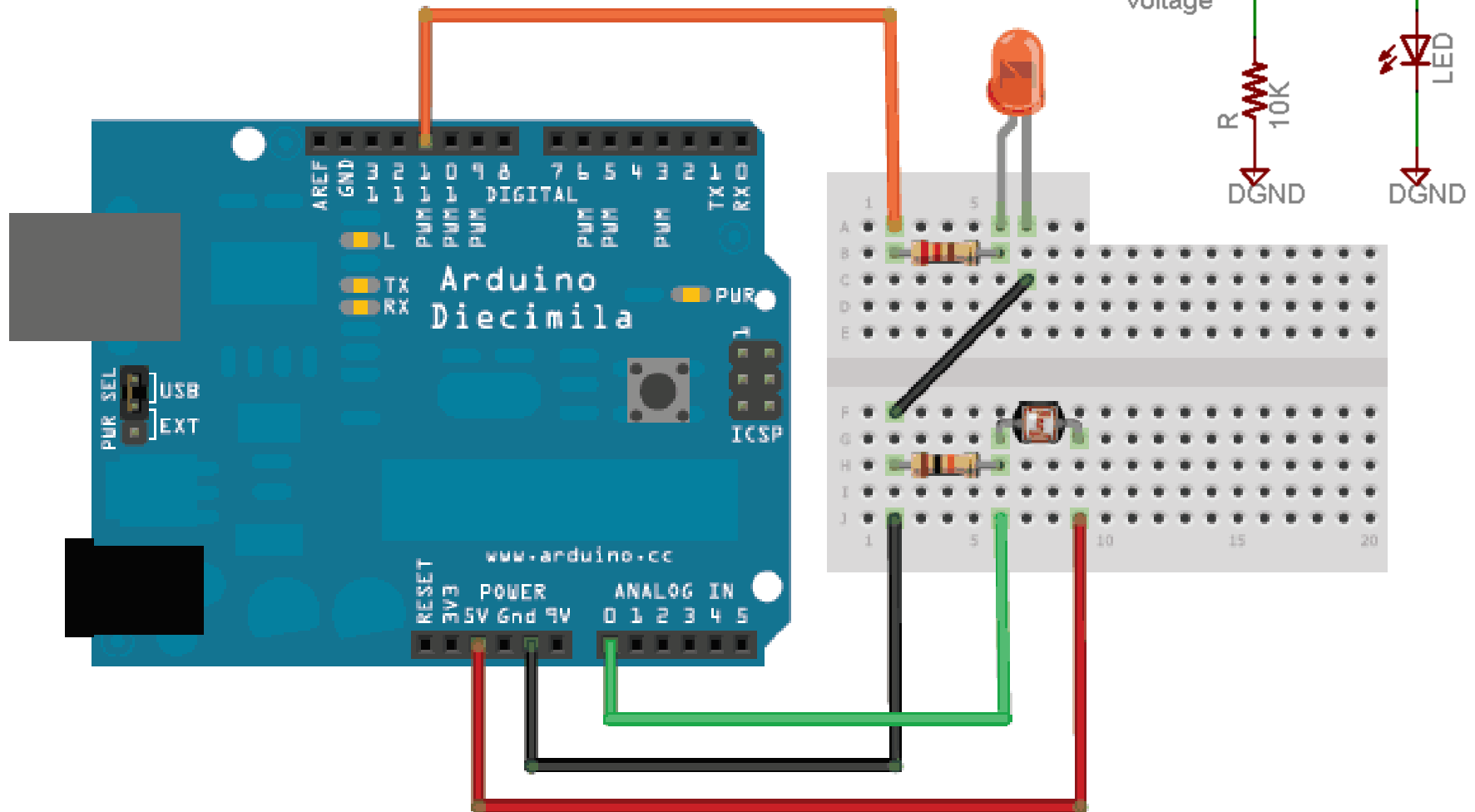
# Light Sensor



# Testing a Light Sensor



# LightSensor Code



# Project #8 – Light Sensor and LED

- Challenge 8a – Use a 10 kOhm resistor to create a “pull-down resistor.” Run a jumper wire from pin A0 to light sensor, and place the resistor in the circuit going to the GND rail. Connect another jumper wire from the other light sensor pin to the 5V rail on the breadboard. Connect positive lead of LED to pin 11.
- Open the example, upload code, open Serial Monitor, and play with the circuit. Try adjusting the ambient light around the sensor. Cover it with your hand, use your phone’s light, etc.



# Ultrasonic Distance Sensor

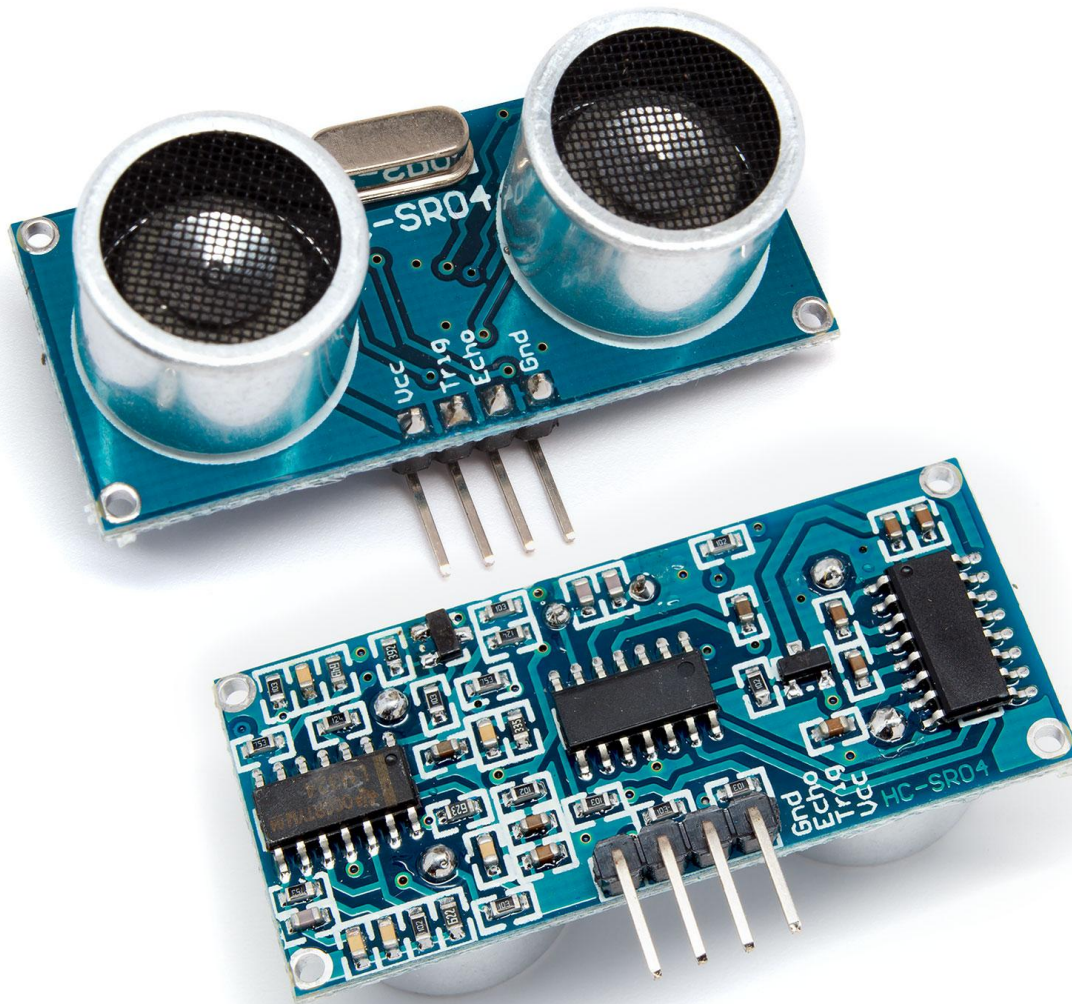




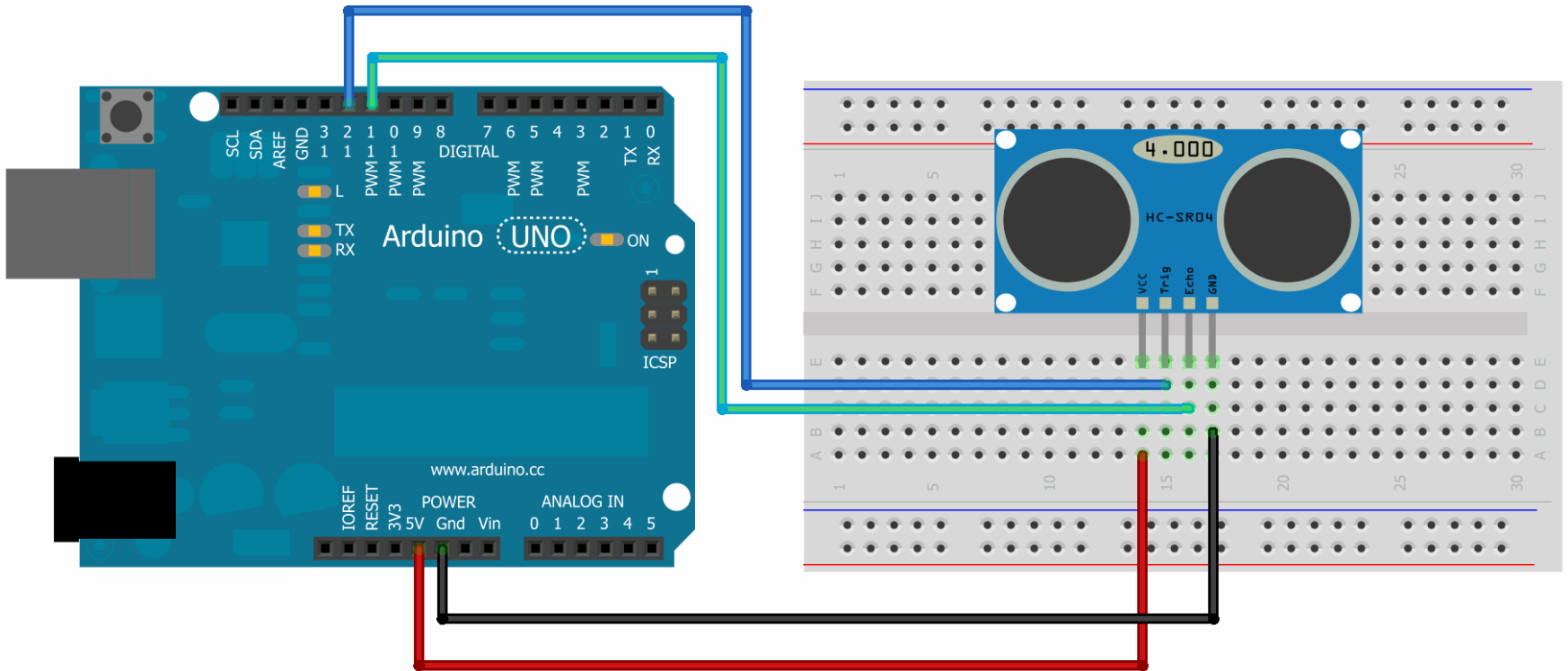
# Ultrasonic Distance Sensor

- The HC-SR04 is an ultrasonic ranging sensor. This economical sensor provides 2cm to 400cm of non-contact measurement functionality with a ranging accuracy that can reach up to 3mm. Each HC-SR04 module includes an ultrasonic transmitter, a receiver and a control circuit.
- There are four pins on the HC-SR04:  
VCC (Power), Trig (Trigger), Echo (Receive), and GND (Ground).

# Ultrasonic Distance Sensor



# PingSensor Code



# Project #9 – Ping Sensor

- Challenge 9a – Connect VCC pin to 5V rail, GND pin to GND rail, trigger pin to Arduino pin 12, echo pin to Arduino pin 11.
- Open the example, upload code, open Serial Monitor, and play with the sensor.
- Challenge 9b – Hook up piezo buzzer to pin 8. Open 9b example and play around with it.

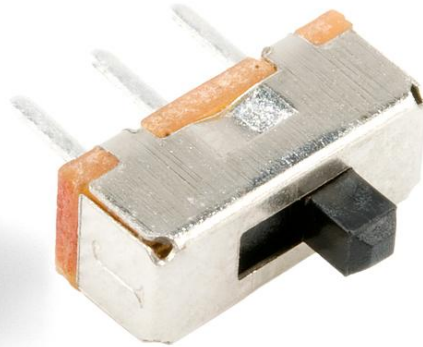
# **Buttons and Switches**

## turning wires into keyboards

# Overview

- Slide Switch
- Momentary Buttons

# Buttons and Switches



- Buttons and Switches are digital inputs that are very useful in the Arduino world. They can be used to turn things on, off, or send commands. They can also be thought of as a digital sensor with two possible states.
- One of the most elementary and easy-to-overlook circuit component is the switch.
- Switches don't require any fancy equations to evaluate. All they do is select between an open circuit and a short circuit. Simple. But how could we live without buttons and switches!? What good is a blinky circuit with no user input? Or a deadly robot with no kill switch? What would our world be without with big red buttons you should never, ever press.



- A switch controls the open-ness or closed-ness of an electric circuit. They allow control over current flow in a circuit (without having to actually get in there and manually cut or splice the wires). Switches are critical components in any circuit which requires user interaction or control.
- A switch can only exist in one of two states: open or closed. In the **off** state, a switch looks like an open gap in the circuit. This, in effect, looks like an **open circuit**, preventing current from flowing.
- In the **on** state, a switch acts just like a piece of perfectly-conducting wire. A short. This **closes the circuit**, turning the system “on” and allowing current to flow unimpeded through the rest of the system

- Switch actuation can come from pushing, sliding, rocking, rotating, throwing, pulling, key-turning, heating, magnetizing, kicking, snapping, licking,...any physical interaction which can cause the mechanical linkages inside the switch to come into, or go out of, contact.
- All switches fall into one of two distinct categories: momentary or maintained.
- **Maintained** switches – like the light switches on your wall – stay in one state until actuated into a new one, and then remain in that state until acted upon once again. These switches might also be called **toggle** or **ON/OFF** switches.
- **Momentary** switches only remain active as long as they're actuated. If they're not being actuated, they remain in their "off" state. You've probably got a momentary switch (or 50) right in front of you...keys on a keyboard!
- Semantic alert! Most of the switches we refer to as "buttons" fall in the momentary category. Activating a button usually means pressing down on it in some manner, which just *feels* like a momentary control. There are such things as a maintained button, but for this tutorial when we slip and talk about "buttons", think "momentary push-down switch"

# Slide Switches



# Slide Switches

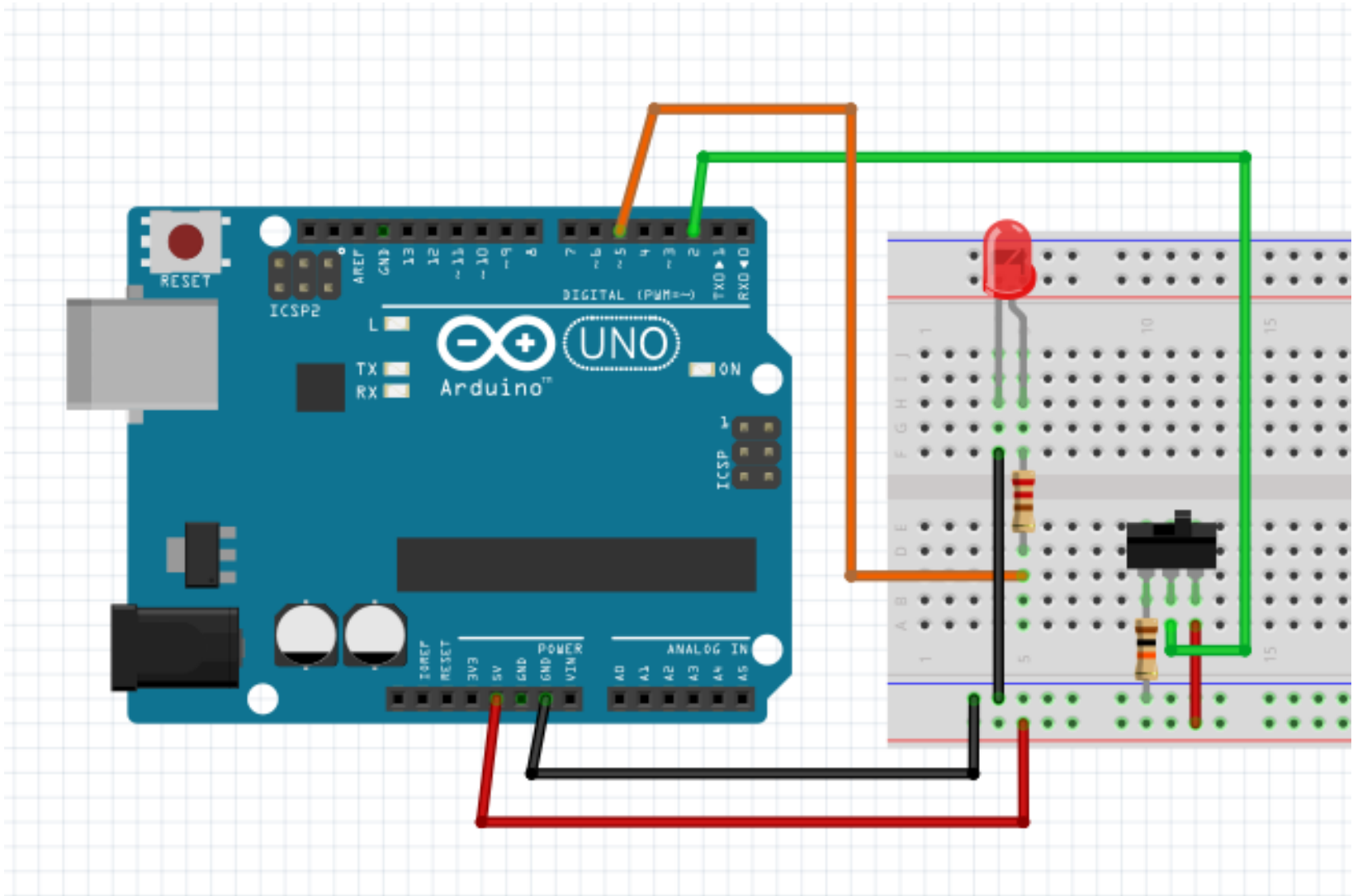
- Need a really basic, no-frills ON/OFF or selector switch? Slide switches might be for you! These switches have a tiny little nub which protrudes from the switch, and it slides across the body into one of two (or more) positions.
- You'll usually find slide switches in SPDT or DPDT configurations. The common terminal is usually in the middle, and the two select positions are on the outside.



# Testing a Slide Switch



# Slide Switch –SwitchLED Code



# Project #10 – Switch and LED

- Challenge 10a – Connect one end pin of switch to 10 kOhm resistor and connect those to GND rail. Connect other end of switch with jumper wire to 5V rail. Run jumper wire from middle pin on switch to Arduino pin 2. Hook up LED to Arduino pin 5.
- Open the example, upload code, and play with the switch.

# Momentary Switches AKA Buttons

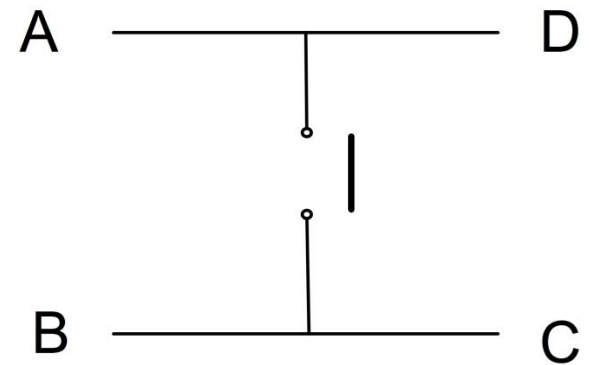
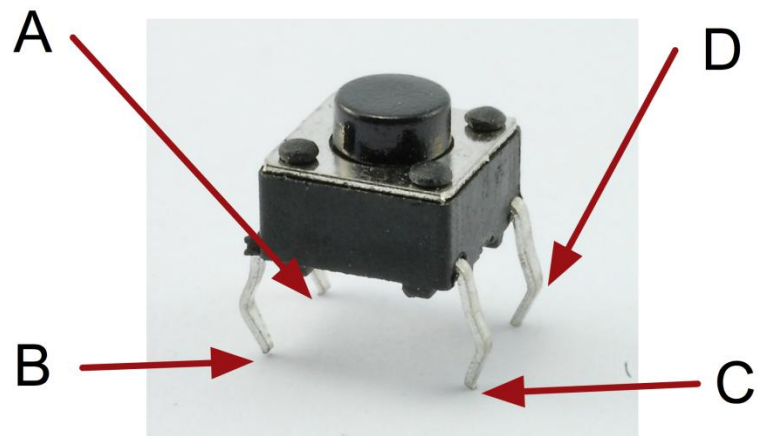




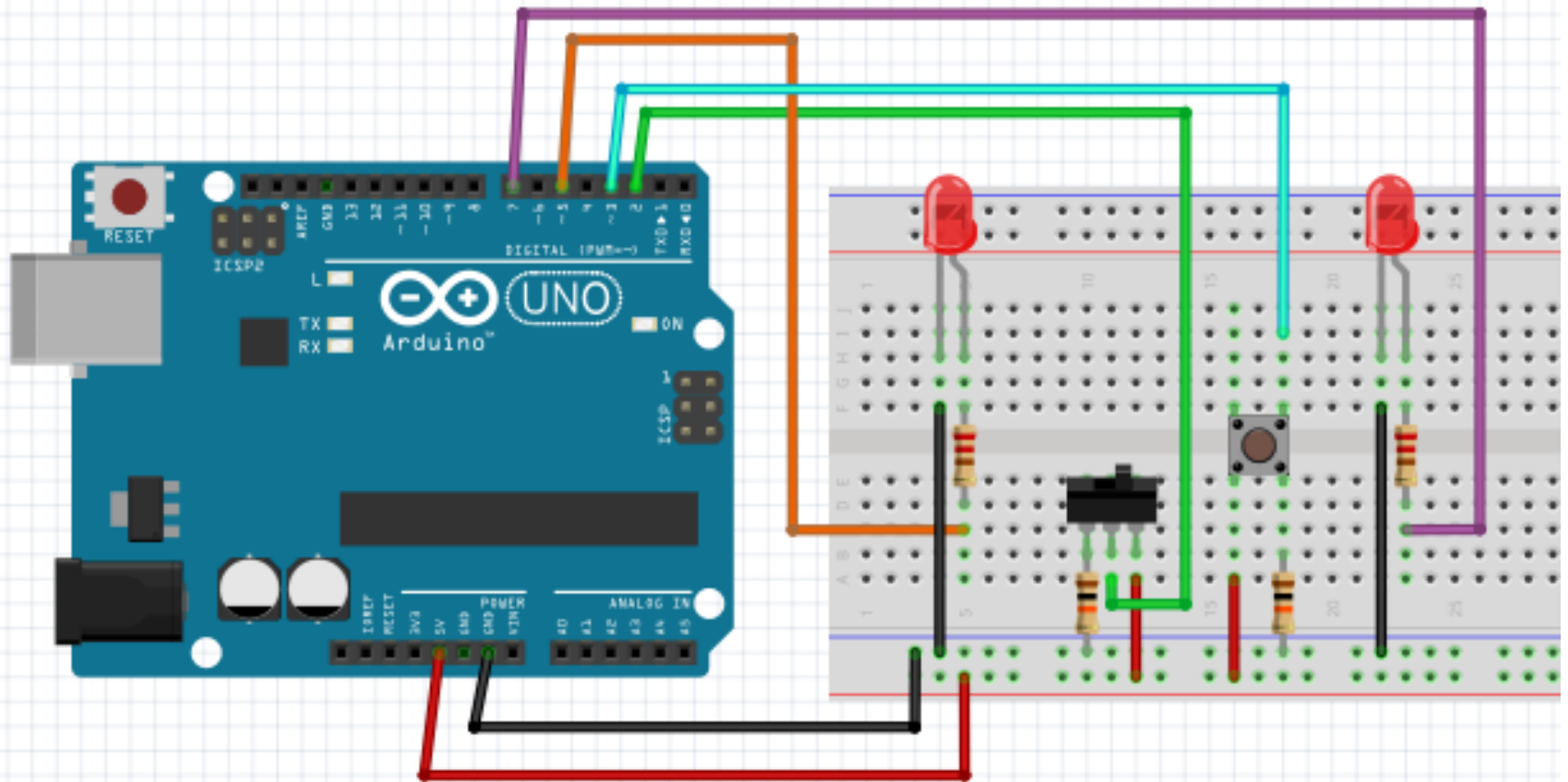
# Momentary Switches AKA Buttons

- Momentary switches are switches which only remain in their on state as long as they're being actuated (pressed, held, magnetized, etc.). Most often momentary switches are best used for intermittent user-input cases; stuff like reset or keypad buttons.
- Push-button switches are the classic momentary switch. Typically these switches have a really nice, tactile, “clicky” feedback when you press them. They come in all sorts of flavors: big, small, colorful, illuminated (when an LED shines up through the button). They might be terminated as through-hole, surface-mount, or even panel-mount.

# Momentary Switches AKA Buttons



# Buttons – ButtonLED Code



# Project #11 – Button and LED

- Challenge 11a – Connect pin C of button to 10 kOhm resistor and connect it to GND rail. Connect pin B on button with jumper wire to 5V rail. Run jumper wire from pin D on button to Arduino pin 3. Hook up an LED to Arduino pin 7.
- Open the example, upload code, and play with the button.
- Challenge 11b – Combine Switch (10a) and Button (11a) codes to have a program to use a button to turn LEDs using both a switch and a button.

# International Morse Code

1. The length of a dot is one unit.
2. A dash is three units.
3. The space between parts of the same letter is one unit.
4. The space between letters is three units.
5. The space between words is seven units.

A ● ■  
B ■ ● ● ●  
C ■ ● ■ ●  
D ■ ● ●  
E ●  
F ● ● ■ ●  
G ■ ■ ●  
H ● ● ● ●  
I ● ●  
J ● ■ ■ ■  
K ■ ● ■  
L ● ■ ● ●  
M ■ ■  
N ■ ●  
O ■ ■ ■  
P ● ■ ■ ●  
Q ■ ■ ● ■  
R ● ■ ●  
S ● ● ●  
T ■

U ● ● ■  
V ● ● ● ■  
W ● ■ ■  
X ■ ● ● ■  
Y ■ ● ■ ■  
Z ■ ■ ● ●

1 ● ■ ■ ■ ■  
2 ● ● ■ ■ ■  
3 ● ● ● ■ ■  
4 ● ● ● ● ■  
5 ● ● ● ● ●  
6 ■ ● ● ● ●  
7 ■ ■ ● ● ●  
8 ■ ■ ■ ● ●  
9 ■ ■ ■ ■ ●  
0 ■ ■ ■ ■ ■

# Project #12 – Morse Code with Button and Buzzer

- Challenge 12a – Connect pin C of button to 10 kOhm resistor and connect it to GND rail. Connect pin B on button with jumper wire to 5V rail. Run jumper wire from pin D on button to Arduino pin 3. Hook up an LED to Arduino pin 7. Hook up piezo buzzer to pin 8.
- Open the example, upload code, and play with the button.

# Servos

I like to move it, move it

# Overview

- The Sweeping Servo



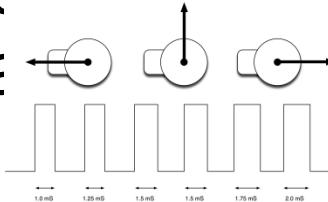
# The Sweeping Servo

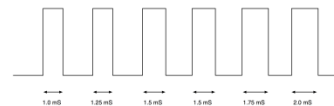


# The Sweeping Servo

- Servo motors are great devices that can turn to a specified position.
- Usually, they have a servo arm that can turn 180 degrees. Using the Arduino, we can tell a servo to go to a specified position and it will go there. As simple as that!
- Servo motors were first used in the Remote Control (RC) world, usually to control the steering of RC cars or the flaps on a RC plane. With time, they found their uses in robotics, automation, and of course, the Arduino world.
- Here we will see how to connect a servo motor and then how to turn it to different positions.

# The Sweeping Servo

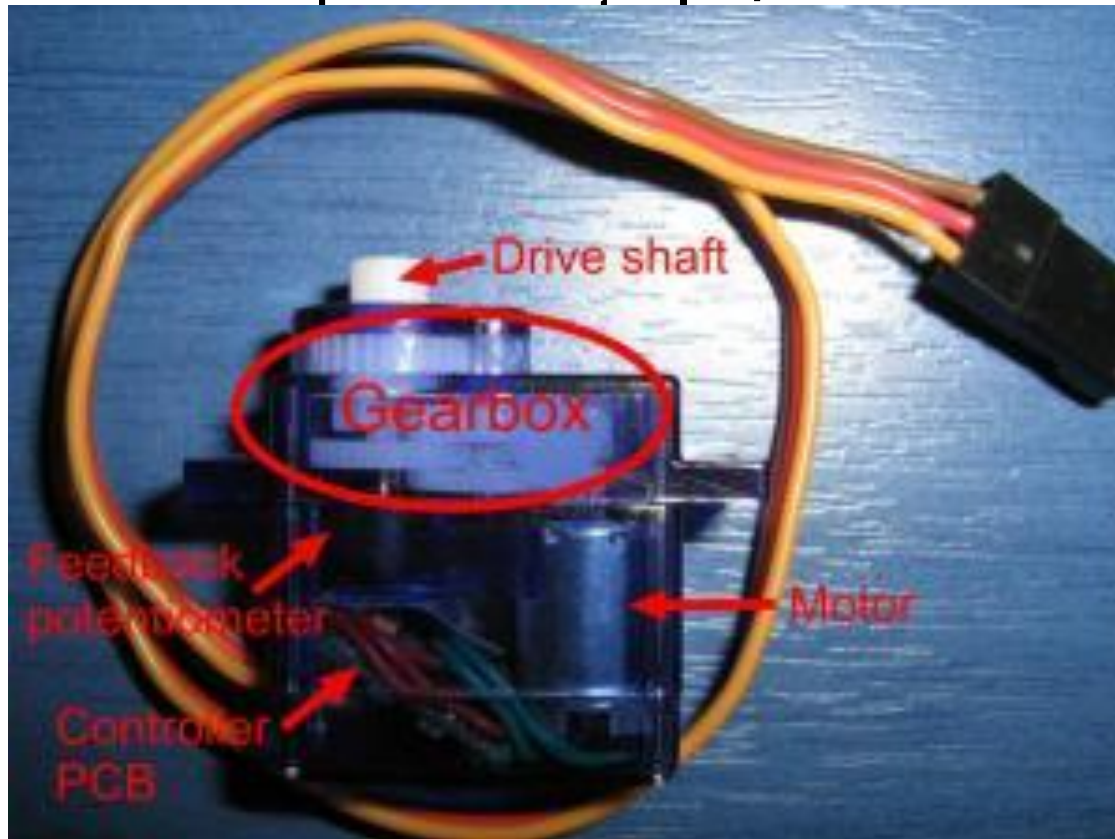
The position of the servo motor is set by the length of a pulse. The servo expects to receive a pulse roughly every 20 milliseconds. If that pulse is high for 1 millisecond, then the servo angle will be zero, if it is 1.5 milliseconds, then it will be at its centre position and if it is 2 milliseconds it will be at its end point. rees.



The end points of the servo can vary and many servos only turn through about 170 degrees. You can also buy 'continuous' servos that can rotate through the full 360 degrees.

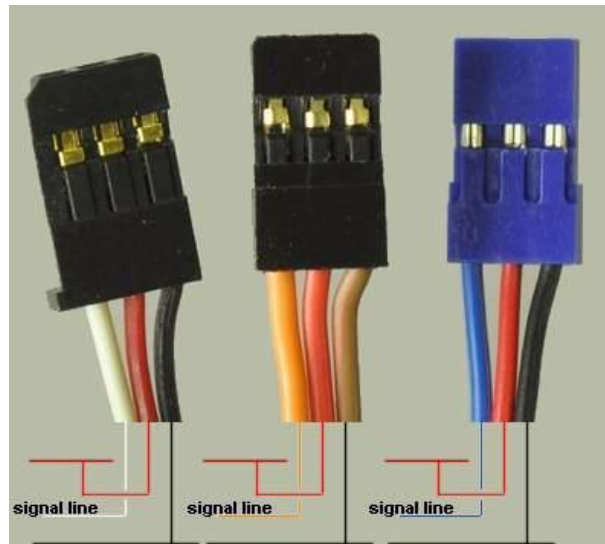
# The Sweeping Servo

- A servo motor has everything built in: a motor, a feedback circuit, and most important, a motor driver. It just needs one power line, one ground,



# The Sweeping Servo

- The servo motor has a female connector with three pins.  
The darkest is usually the ground.  
The power cable in all standards should be red  
The remaining line is the signal line, connected to a digital PWM pin on the Arduino.



# Project #13 – Servo and Serial

- Challenge 13a – Use the following steps to connect a servo motor to the Arduino using jumper wires:

Brown wire – Connect to GND on the Arduino.

Red wire - Connect to 5V on the Arduino.

Yellow wire - Connect to pin 11 on the Arduino.

- Open the example, upload code, and play with the servo using the serial monitor.
- Type in a number from 1 to 180 in the input box, hit enter and WATCH MAGIC HAPPEN!!!!

# Project #13 – Servo Sweep

- Challenge 13b – Use the following steps to connect a servo motor to the Arduino using jumper wires:

Brown wire – Connect to GND on the Arduino.

Red wire - Connect to 5V on the Arduino.

Yellow wire - Connect to pin 11 on the Arduino.

- Open the example, upload code, and WATCH MORE MAGIC HAPPEN!!!!

**The End**  
**Good luck!!!!**