# ARDUINO:

## Intro to Microcontrollers - Day 2

The Community

http://www.arduino.cc/

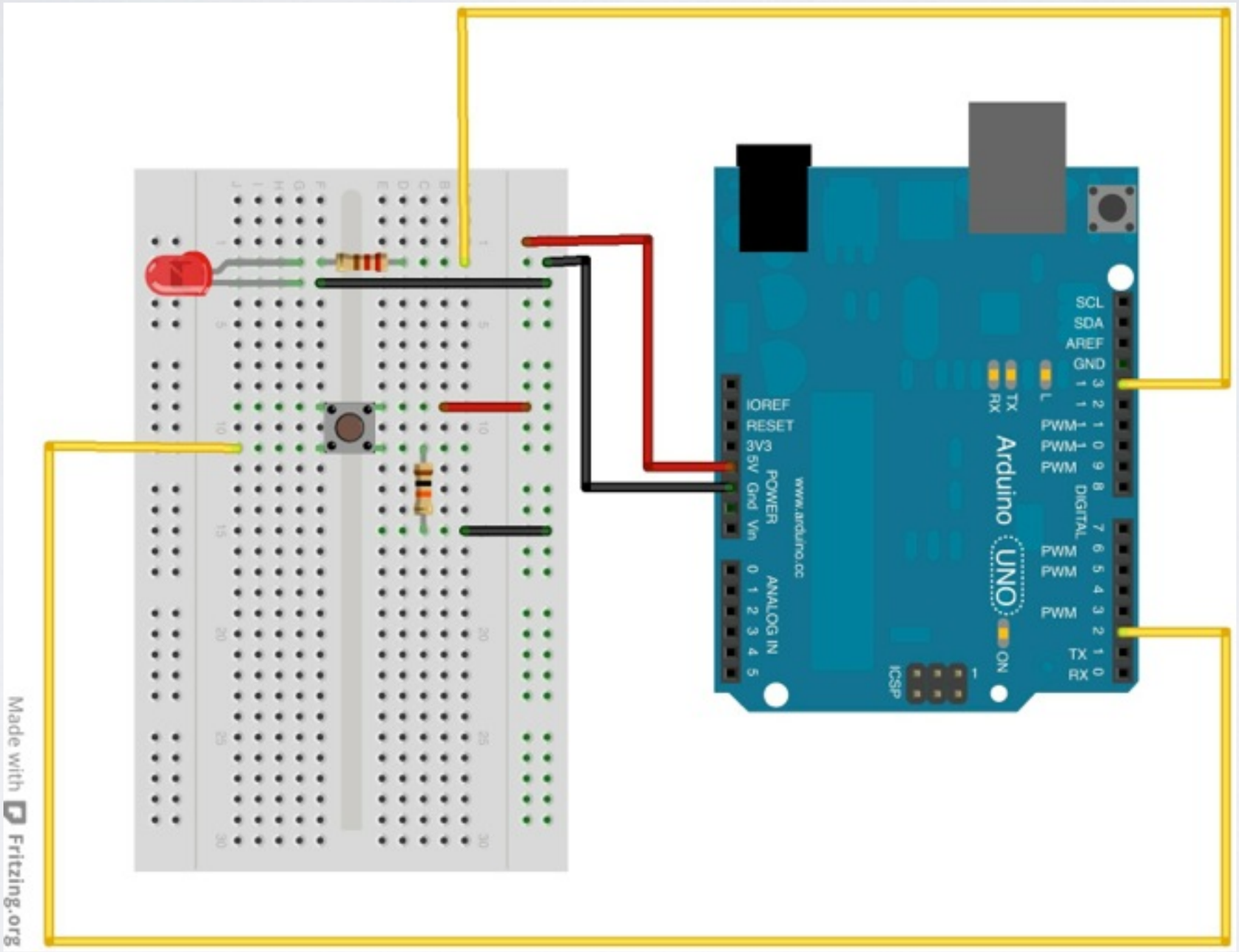
http://www.adafruit.com/

https://www.sparkfun.com

http://fritzing.org/home/

Way More….

# Digital Input



# DIGITAL INPUT (BUTTON)

Parts for this project

- Solderless Breadboard

- 7 x Flexible Wire Jumpers

- 1 x LEDs (any color)

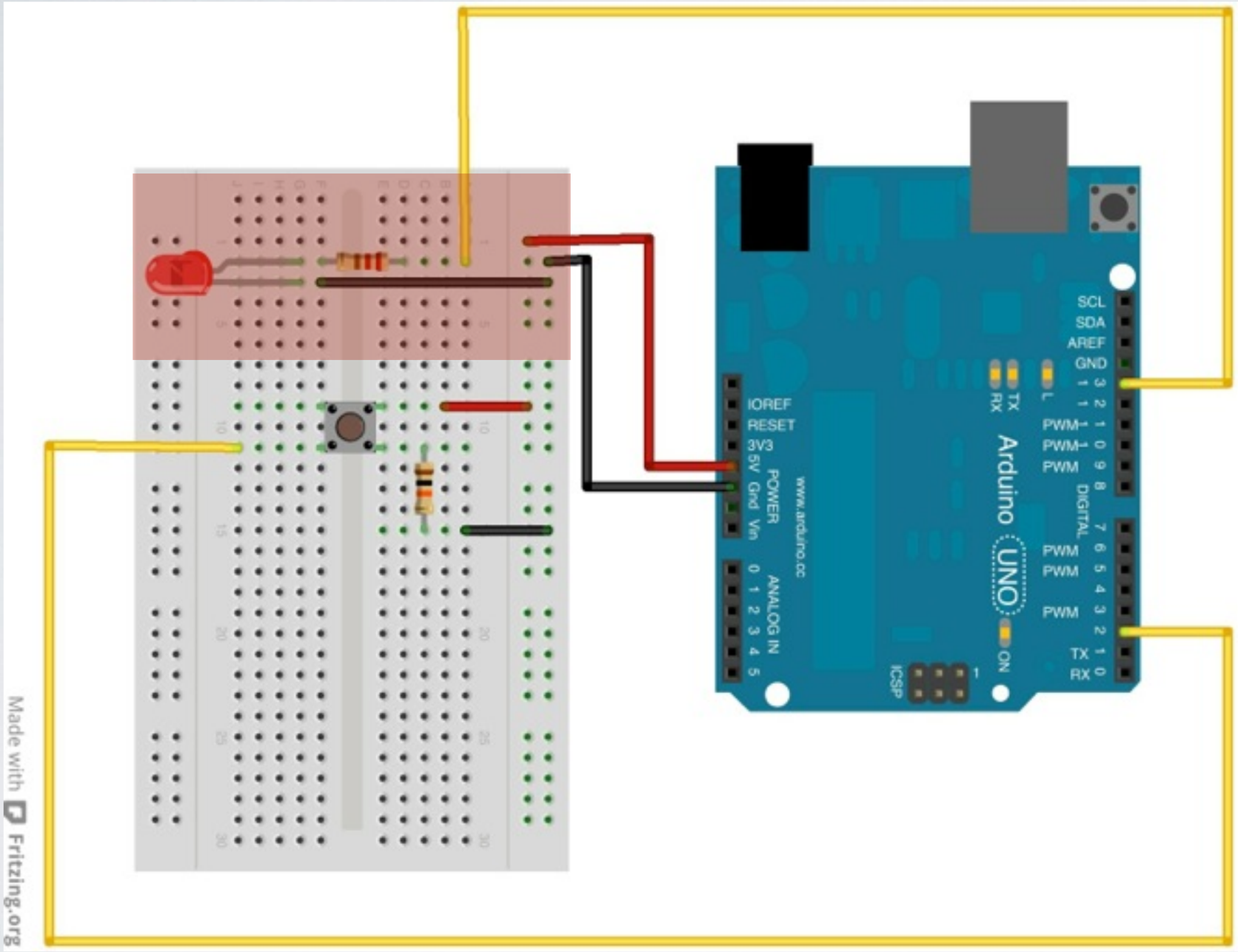
- 2 x 220 Ohm Resistors

- 1 x Tactile Pushbutton

- Arduino Duo board

- USB Cable

# DIGITAL INPUT (BUTTON)

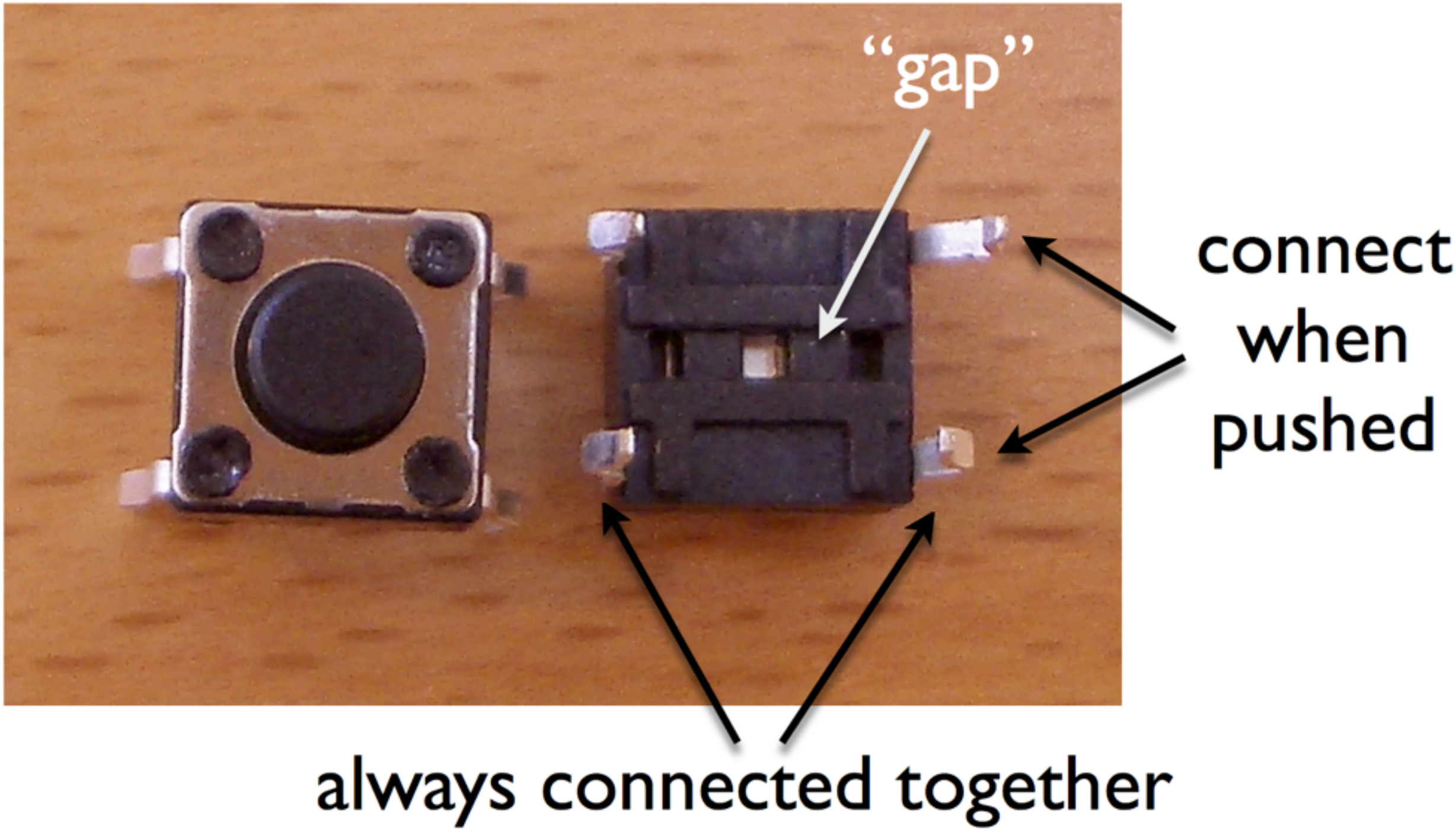



Connect an LED, cathode to ground pin and anode in upper row.

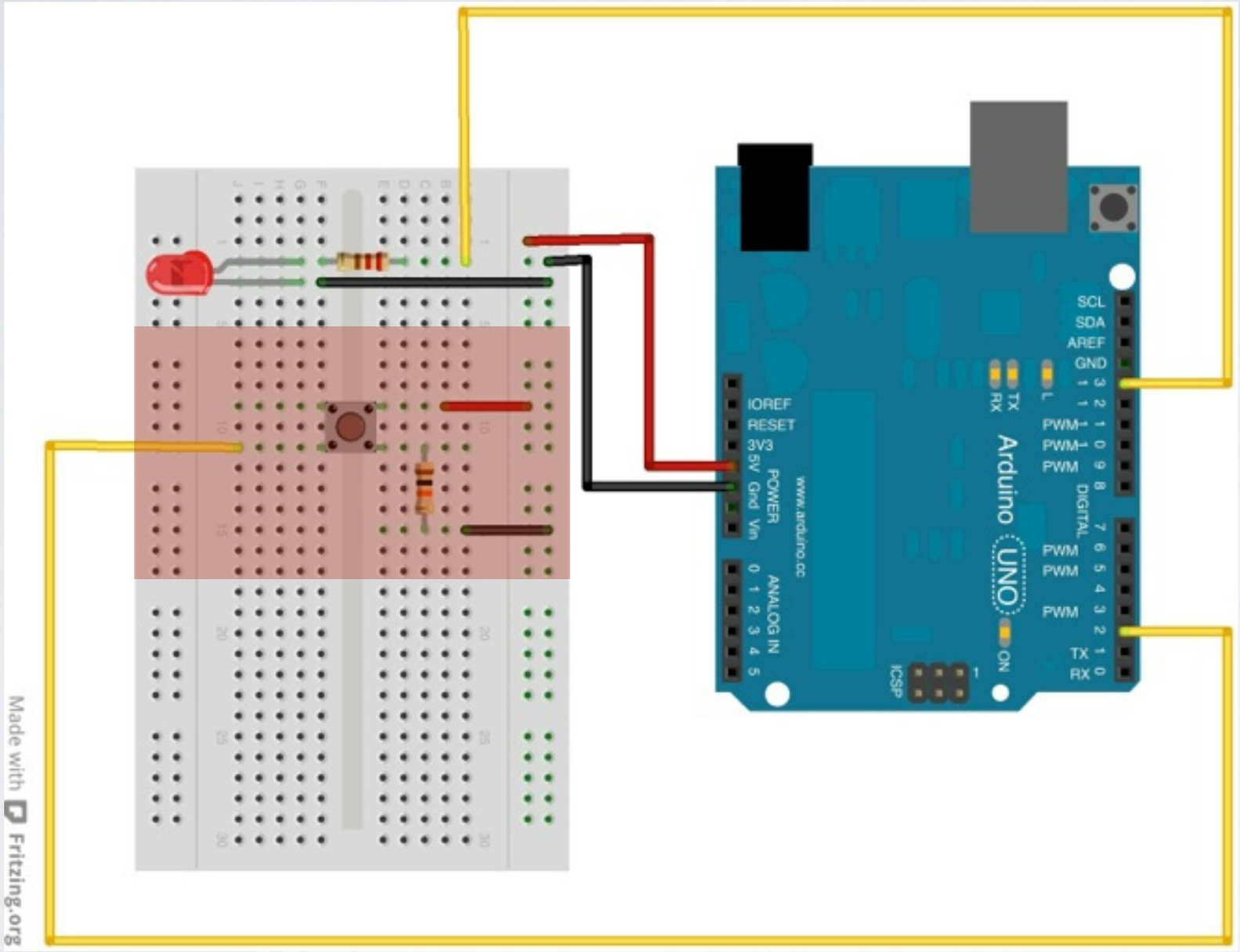
Place a resistor in series with the anode(long lead of LED)

Connect the resistor to Pin 13

# DIGITAL INPUT (BUTTON)



# DIGITAL INPUT (BUTTON)




Connect 5V and GND to the side strips

Add a pushbutton across the center vertical row.

Connect the top pin to 5V.

Connect lower side to ground in series with a resistor.

Connect the opposite lower pin of the switch to pin 2

# DIGITAL INPUT (BUTTON)

```
// constants won't change. They're used here to set pin numbers:
const int buttonPin = 2;     // the number of the pushbutton pin
const int ledPin =  13;      // the number of the LED pin

// variables will change:
int buttonState = 0;         // variable for reading the pushbutton status

void setup() {
  // initialize the LED pin as an output:
  pinMode(ledPin, OUTPUT);
  // initialize the pushbutton pin as an input:
  pinMode(buttonPin, INPUT);
}
```

```
void loop(){
  // read the state of the pushbutton value:
  buttonState = digitalRead(buttonPin);

  // check if the pushbutton is pressed.
  // if it is, the buttonState is HIGH:
  if (buttonState == HIGH) {
    // turn LED on:
    digitalWrite(ledPin, HIGH);
  }
  else {
    // turn LED off:
    digitalWrite(ledPin, LOW);
  }
}
```

# DIGITAL INPUT (BUTTON)

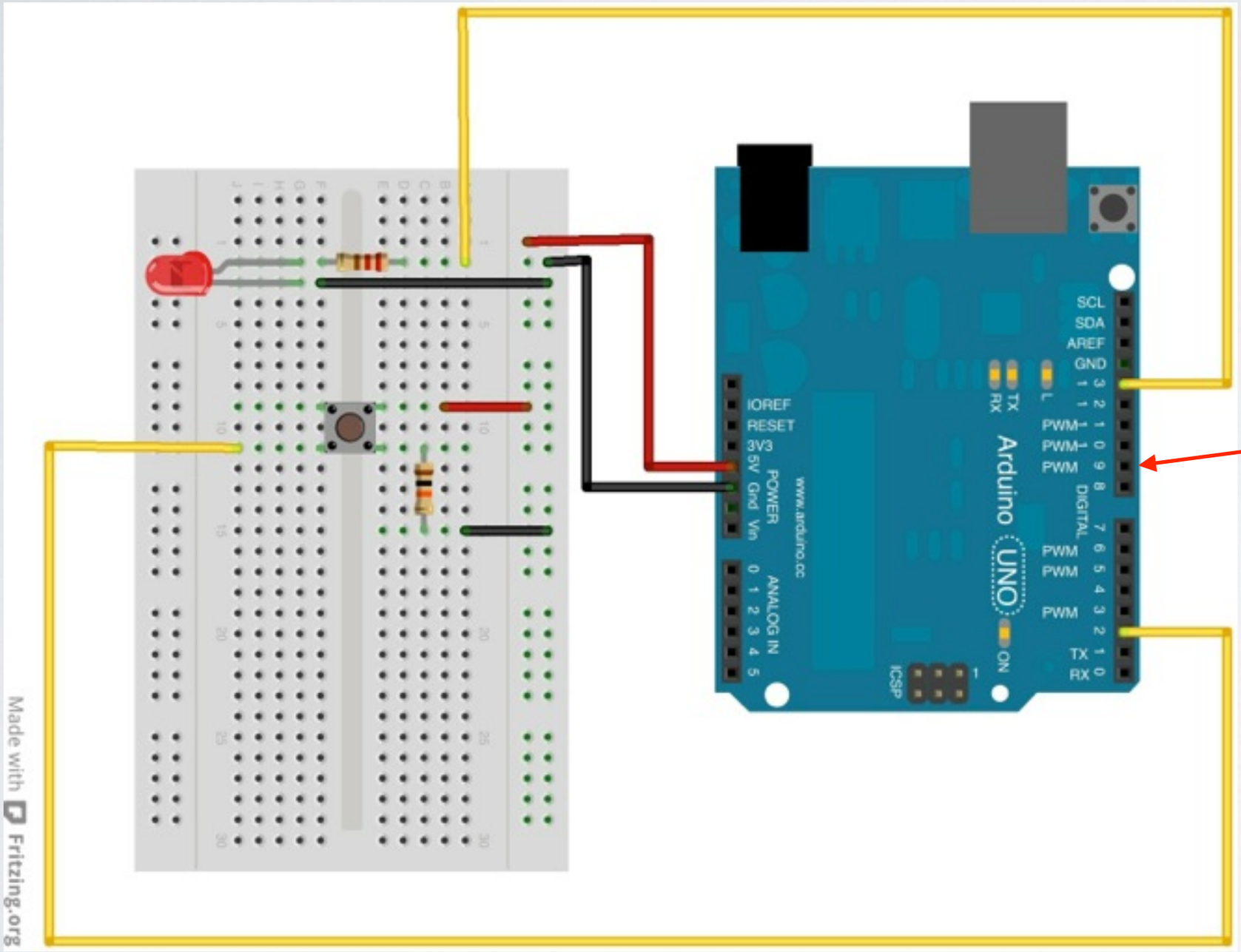
**digitalRead()**

**Syntax:**
**digitalRead(pin)**

**Parameters**
**pin: the number of the digital pin you want to read (int)**

**Returns**
**HIGH or LOW**

# VARIATION 1




Keep Same Circuit

*Except, Move the LED pin from 13 to pin 9

# FADE OR BLINK

## Upload Fade_Or_Blink from the code folder.

```
/*
 * FadeOrBlink
 *
 *
 */

int ledPin = 9;                // choose the pin for the LED
int inputPin = 2;              // choose the input pin (for a pushbutton)
int val = 0;                   // variable for reading the pin status
int fadeval = 0;

void setup() {
  pinMode(ledPin, OUTPUT);     // declare LED as output
  pinMode(inputPin, INPUT);    // declare pushbutton as input
}

void loop(){
  val = digitalRead(inputPin);  // read input value
  if (val == HIGH) {            // pushed button means do blinking
    digitalWrite(ledPin, LOW);  // turn LED OFF
    delay(50);
    digitalWrite(ledPin, HIGH); // turn LED ON
    delay(50);
  }
  else {  // else button isn't pressed so do fading
    for(fadeval = 0 ; fadeval <= 255; fadeval+=5) { // fade in (from min to max)
      analogWrite(ledPin, fadeval);          // sets the value (range from 0-255)
      delay(10);
    }
    for(fadeval = 255; fadeval >=0; fadeval-=5) { // fade out (from max to min)
      analogWrite(ledPin, fadeval);
      delay(10);
    }
  }
}
```

# FADE OR BLINK

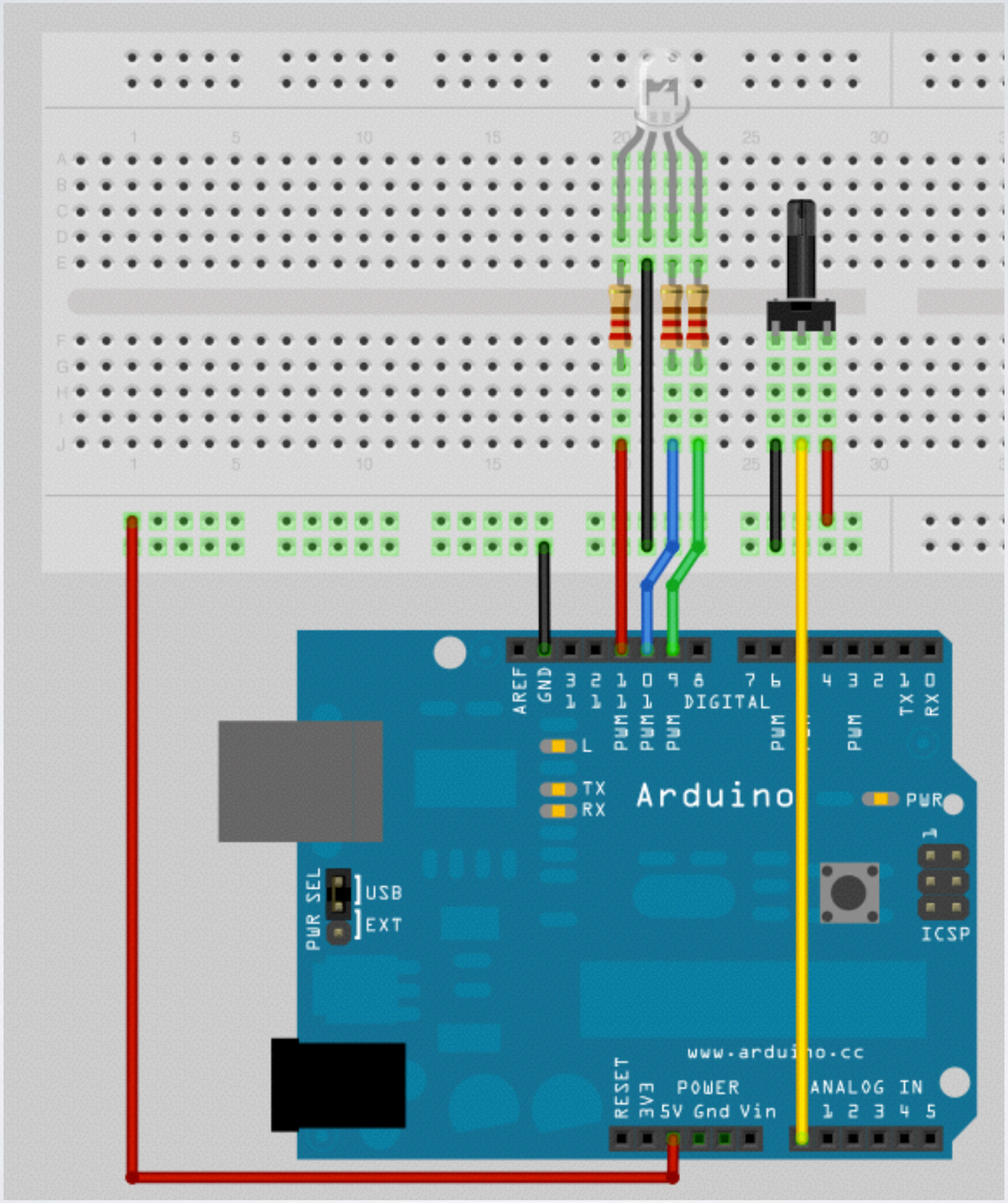
**analogWrite();**

**Syntax:**
**analogWrite(pin, value)**

**Parameters:**
**pin: the pin to write to.**

**value: the duty cycle: between 0 (always off) and 255 (always on).**

# ANALOG-IN ANALOG-OUT



# ANALOG-IN ANALOG-OUT

Parts for this project

- Solderless Breadboard

- 8 x Flexible Wire Jumpers

- 1 x RGB LED

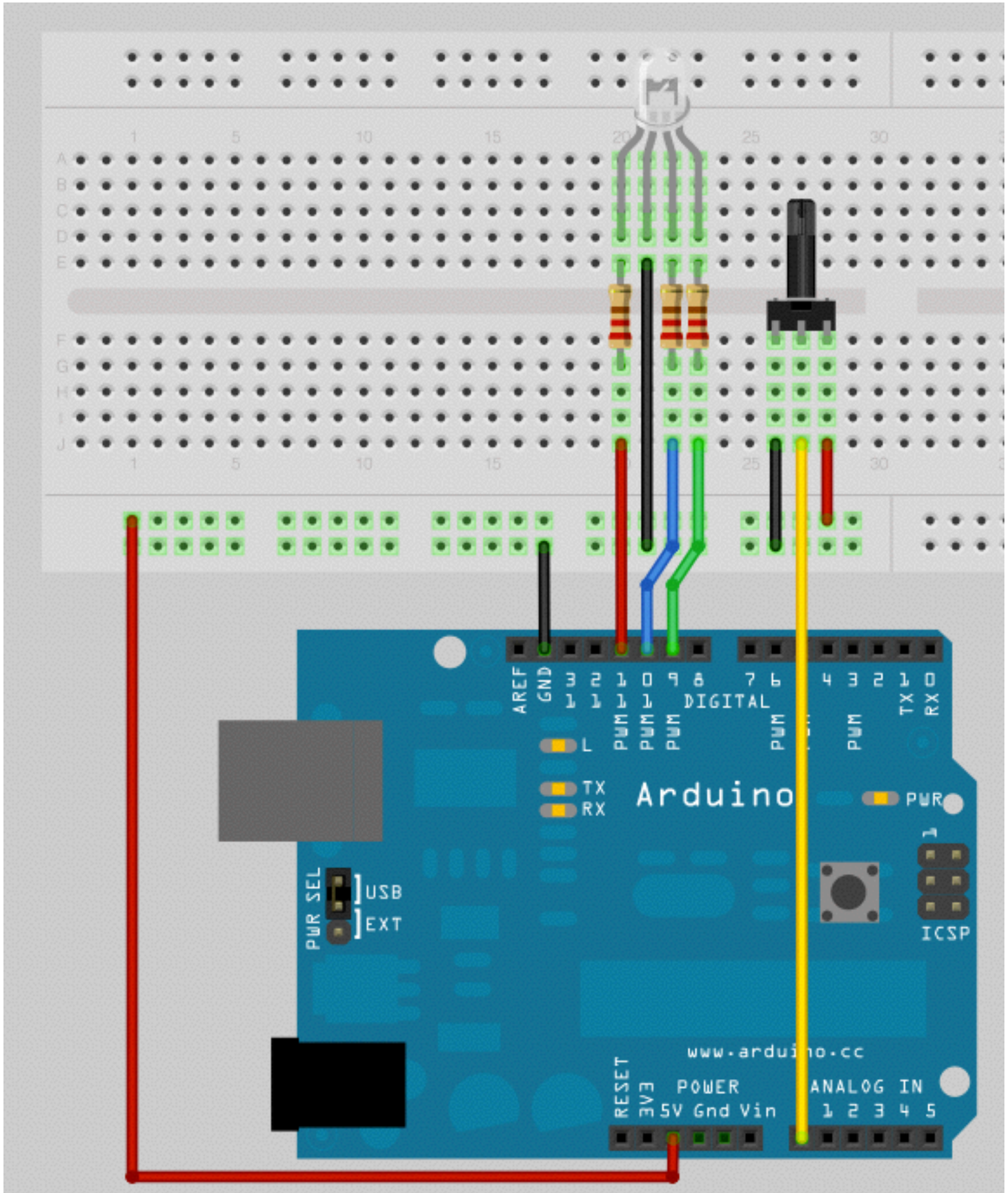
- 3 x 220 Ohm Resistors

- 1 x Blue Potentiometer with white knob

- Arduino Duo board

- USB Cable

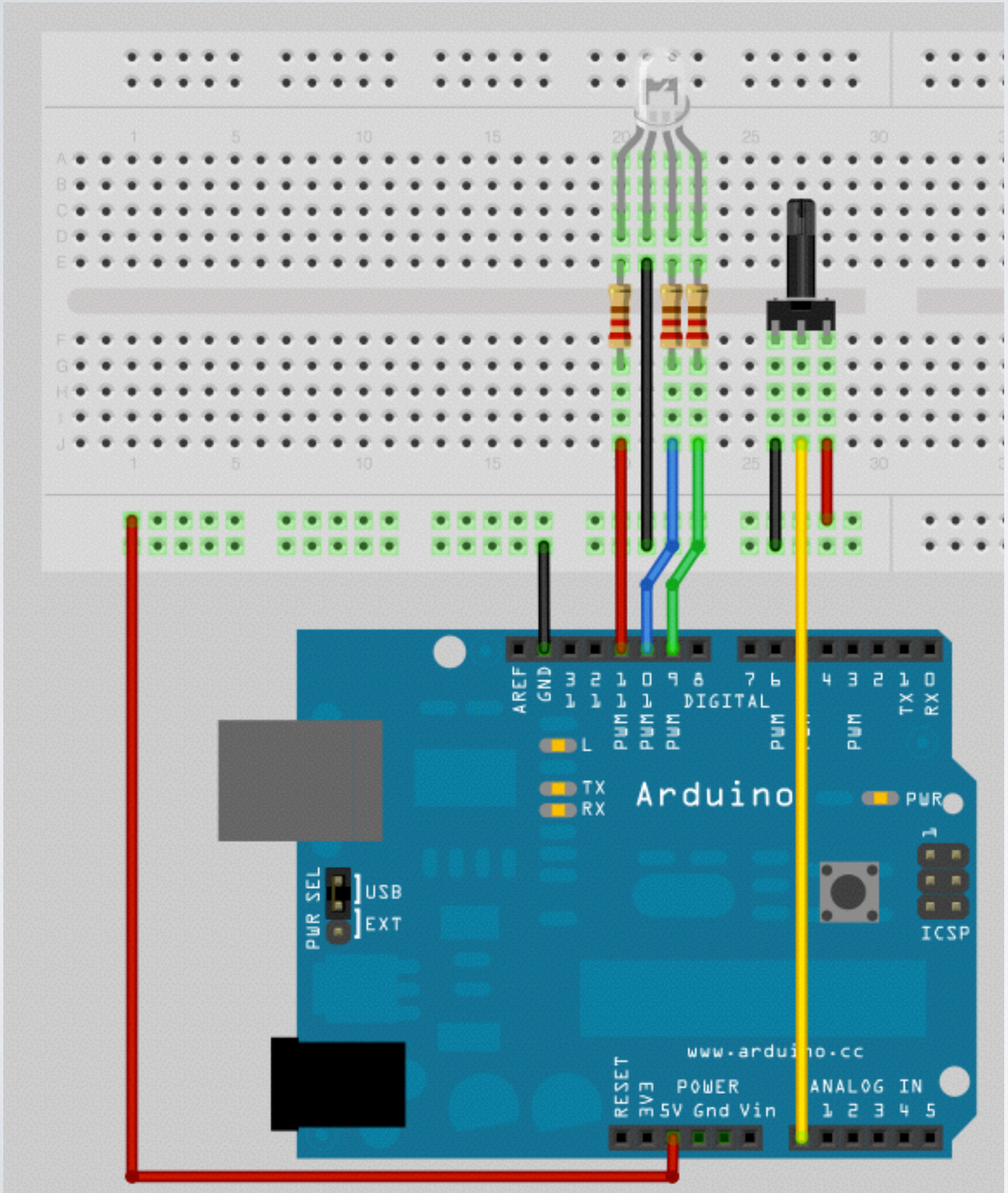
# ANALOG-IN ANALOG-OUT




**Connect 5V and GND to the side strips**

**Connect the short leads of the LED to pins 9,10,11 in series with a resistor.**

# ANALOG-IN ANALOG-OUT




**Connect the Potentiometer across 3 row**

**One side goes to ground.**

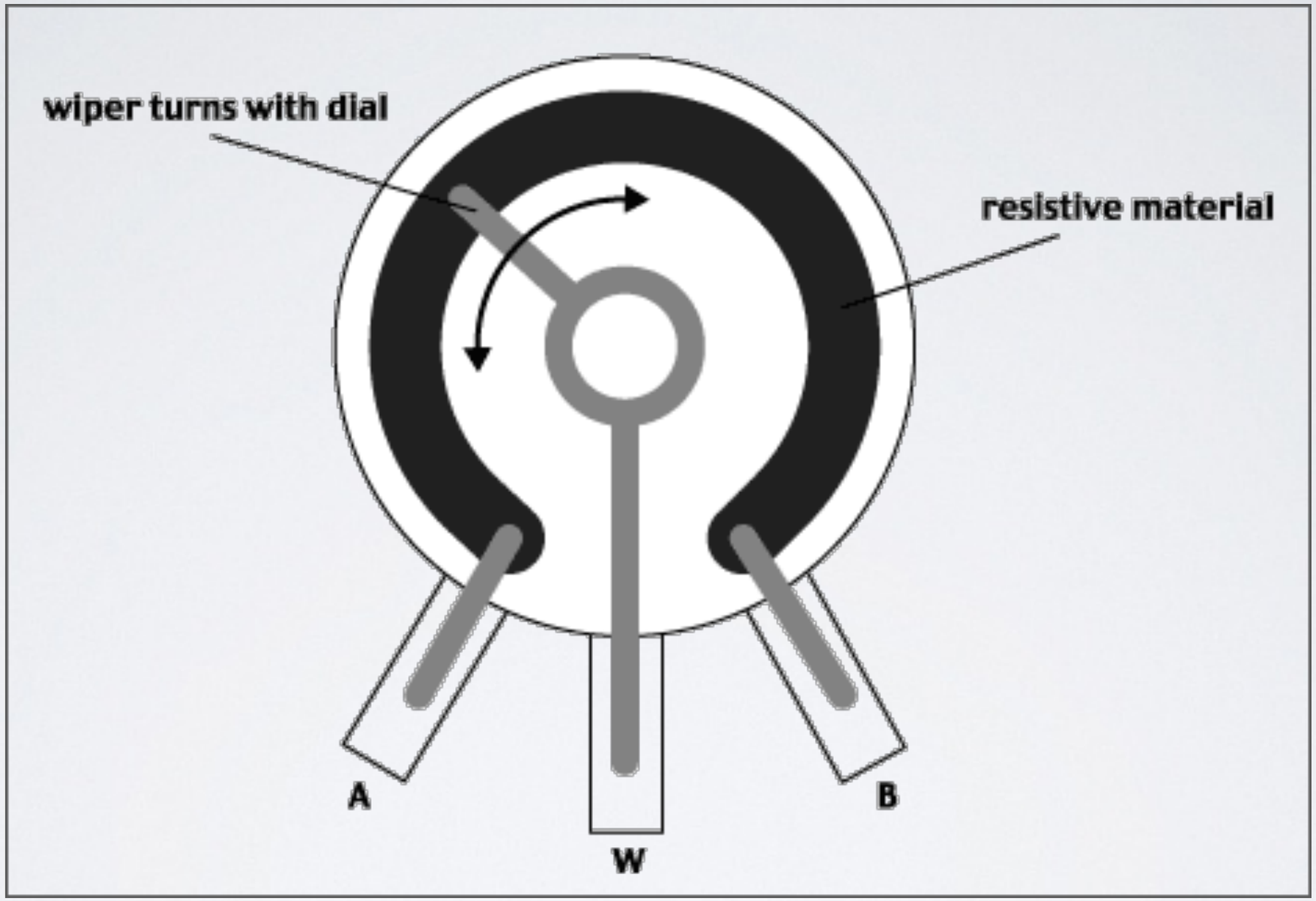
**The other goes to +5V**

**The Center Pin Connects to analog in 0 (Marked A0)**

# Upload RBG_Pot_Fader.ino

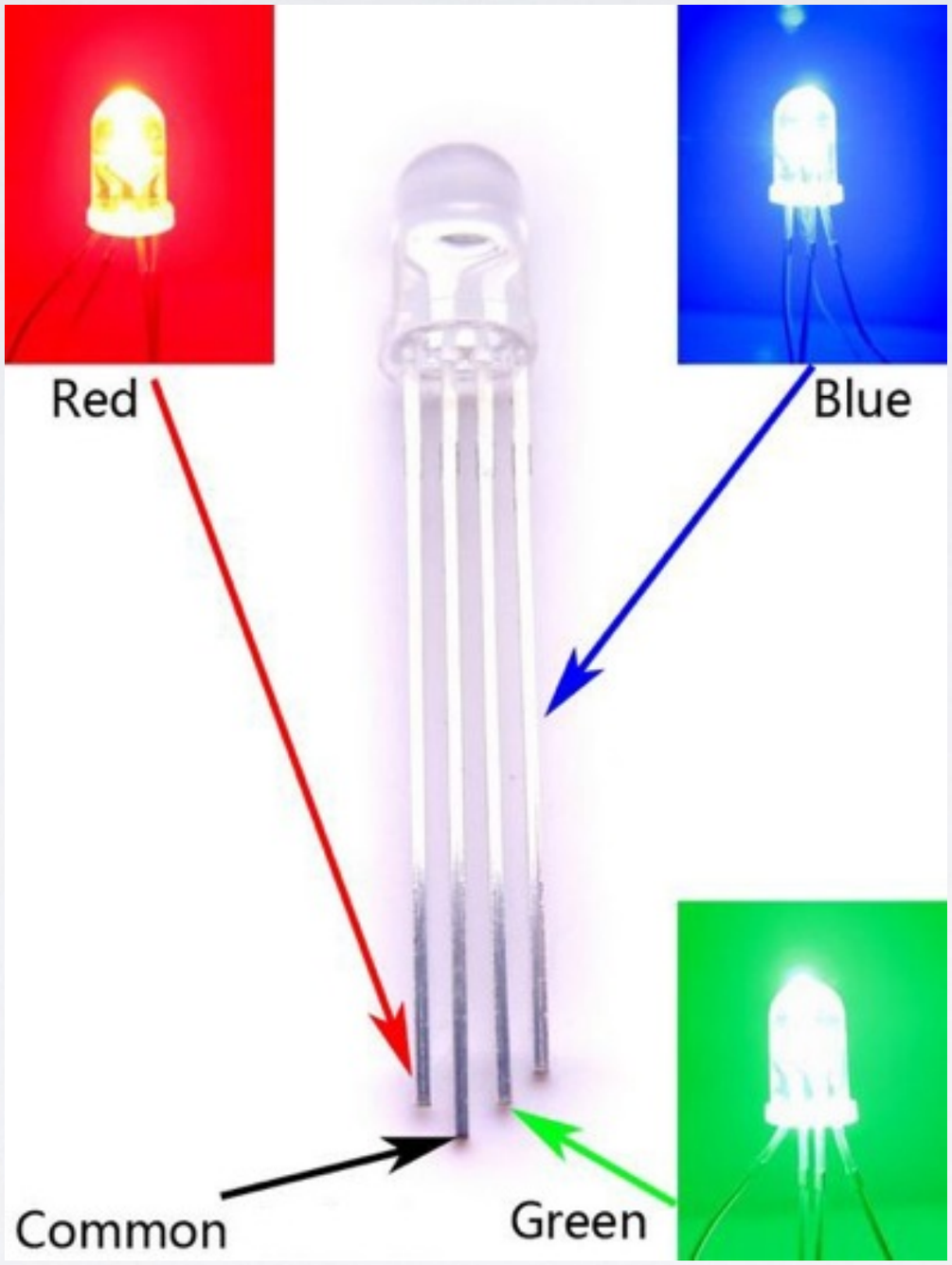
# ANALOG-IN ANALOG-OUT

## Potentiometer



# ANALOG-IN ANALOG-OUT

## RBG LED

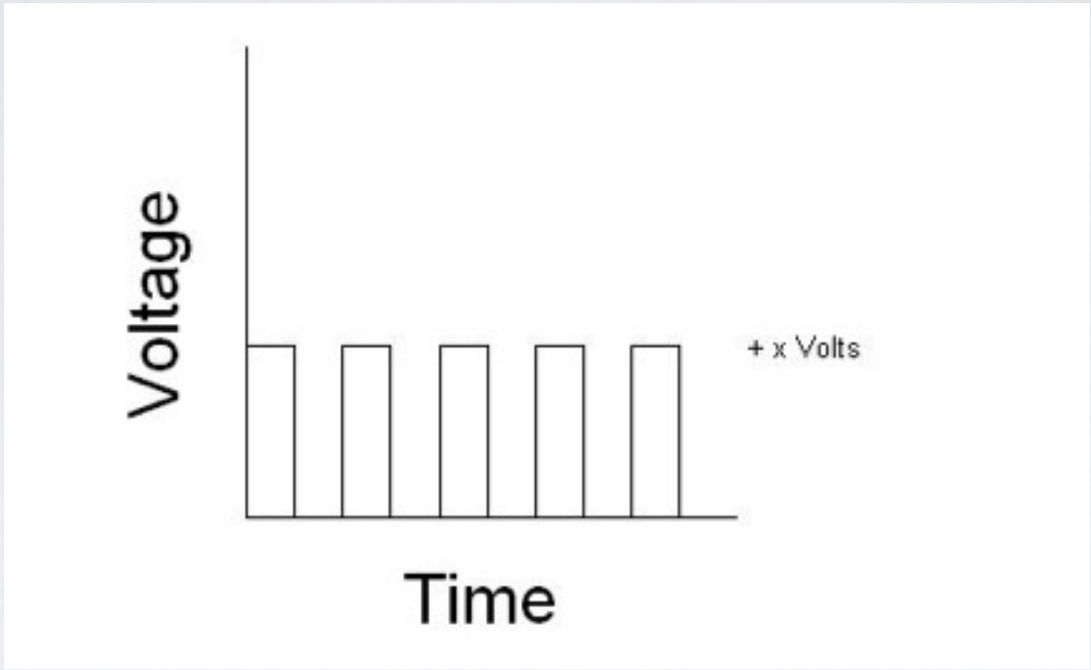


# PULSE WIDTH MODULATION

**-PWM stands for Pulse Width Modulation.**

**-Rather than limiting or controlling the voltage or current going through your LED, the voltage is fixed. When the full voltage is sent through the LED, it will emit the maximum of light it can (basically the same as connecting directly to a fully charged battery)**

# PULSE WIDTH MODULATION




As you can see, we have now changed our constant voltage input for a square wave. In this graphic, our voltage is on half the time and off half the time.

The technical term here would be that we have a 50% duty cycle (50% of the time on.) With such an input on our LED, the amount of light given off appears diminished by 50%.

The reason I say it appears, is because it gives off its maximum amount of light, but only 50% of the time, so our eyes perceive only half the amount of light (more on this on the next slide).

# RGB COLOR MIXING

**Color mixing**

Now that you have red, green and blue light, you can start having fun with color mixing. Color mixing is the neat ability that our eyes have to combine different light colors and create a new color

A additive (light) color mixing diagram

According to this diagram, if we have both red and blue light mixed together we should get a violet light.

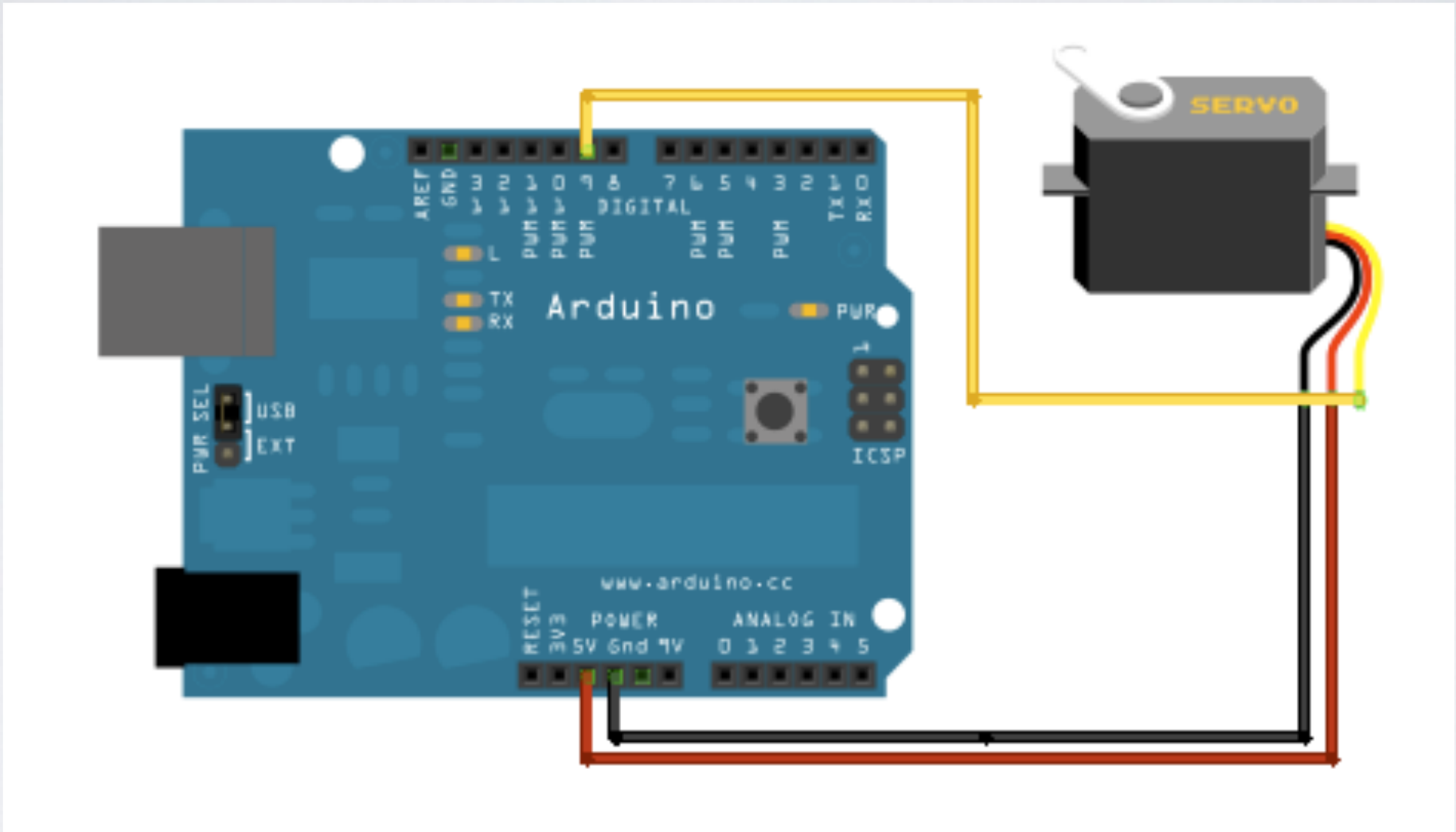
# RGB COLOR MIXING

**RainbowRGB**

**SimpleRGB**

**RandomRGB Fader**

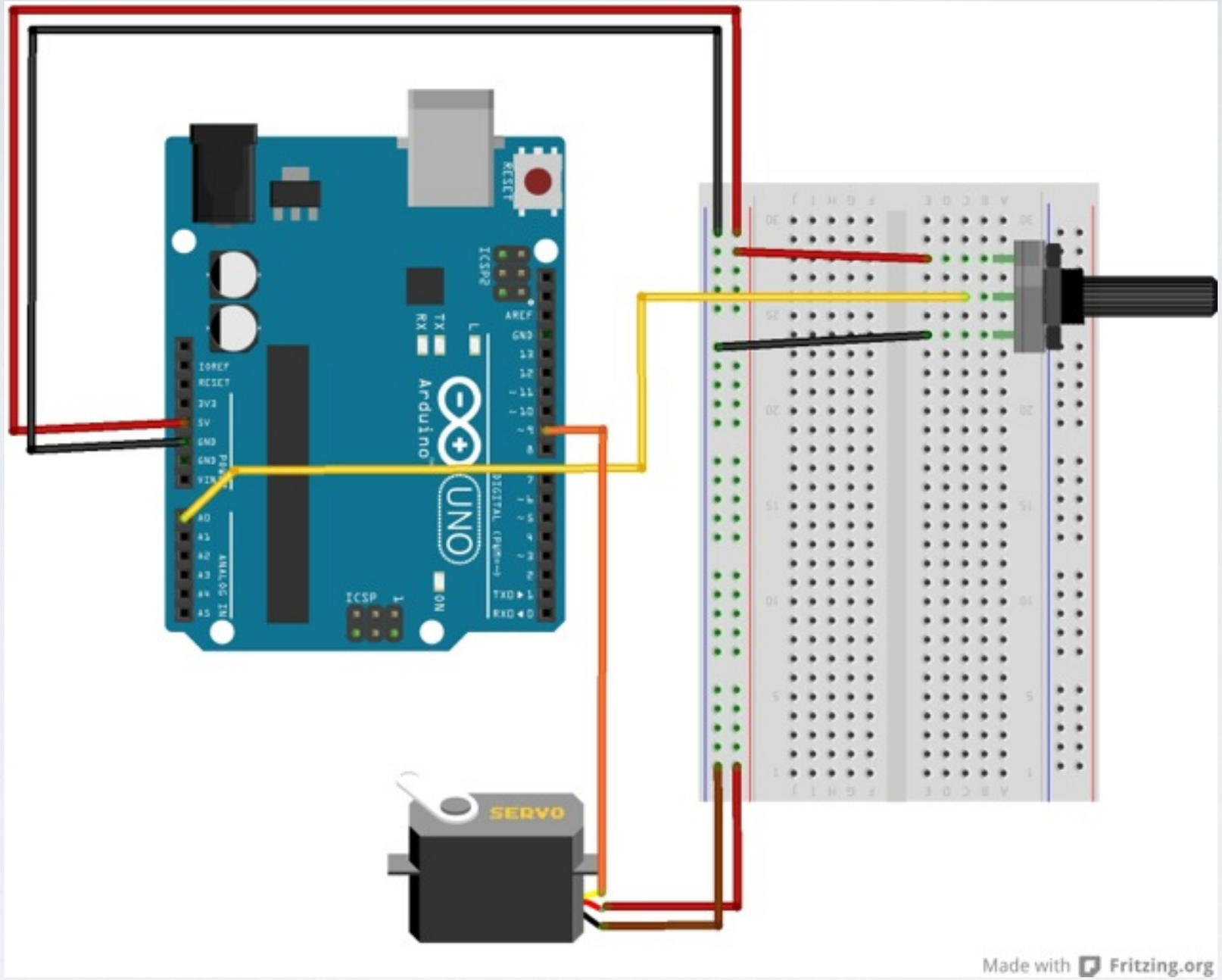
# SERVO


SERVO
AREF
GND
DIGITAL
PWM
TX
RX
L
Arduino
PWR
ICSP
PWR SEL
USB
EXT
www.arduino.cc
RESET
3V3
POWER
5V
Gnd
Vin
ANALOG IN
0 1 2 3 4 5

# SERVO
# SERVO_SWEEP.INO

```
#include <Servo.h>

Servo myservo;  // create servo object to control a servo
                // a maximum of eight servo objects can be created

int pos = 0;    // variable to store the servo position

void setup()
{
  myservo.attach(9);  // attaches the servo on pin 9 to the servo object
}

void loop()
{
  for(pos = 0; pos < 180; pos += 1)  // goes from 0 degrees to 180 degrees
  {                                  // in steps of 1 degree
    myservo.write(pos);              // tell servo to go to position in variable 'pos'
    delay(15);                       // waits 15ms for the servo to reach the position
  }
  for(pos = 180; pos>=1; pos-=1)     // goes from 180 degrees to 0 degrees
  {
    myservo.write(pos);              // tell servo to go to position in variable 'pos'
    delay(15);                       // waits 15ms for the servo to reach the position
  }
}
```

# SERVO WITH KNOB



# SERVO
# SERVO_KNOB.INO

```
// Controlling a servo position using a potentiometer (variable resistor)
// by Michal Rinott <http://people.interaction-ivrea.it/m.rinott>

#include <Servo.h>

Servo myservo;  // create servo object to control a servo

int potpin = 0;  // analog pin used to connect the potentiometer
int val;    // variable to read the value from the analog pin

void setup()
{
  myservo.attach(9);  // attaches the servo on pin 9 to the servo object
}

void loop()
{
  val = analogRead(potpin);            // reads the value of the potentiometer (value between 0 and 1023)
  val = map(val, 0, 1023, 0, 179);     // scale it to use it with the servo (value between 0 and 180)
  myservo.write(val);                  // sets the servo position according to the scaled value
  delay(15);                           // waits for the servo to get there
}
```

# SERIAL COMMUNICATIONS

```
/*
 * Hello World!
 *
 * This is the Hello World! for Arduino.
 * It shows how to send data to the computer
 */


void setup()                    // run once, when the sketch starts
{
  Serial.begin(9600);           // set up Serial library at 9600 bps

  Serial.println("Hello world!");  // prints hello with ending line break
}

void loop()                     // run over and over again
{
                              // do nothing!
}
```

# SERIAL COMMUNICATIONS

```
int a = 5;
int b = 10;
int c = 20;

void setup()                    // run once, when the sketch starts
{
  Serial.begin(9600);           // set up Serial library at 9600 bps

  Serial.println("Here is some math: ");

  Serial.print("a = ");
  Serial.println(a);
  Serial.print("b = ");
  Serial.println(b);
  Serial.print("c = ");
  Serial.println(c);

  Serial.print("a + b = ");       // add
  Serial.println(a + b);

  Serial.print("a * c = ");       // multiply
  Serial.println(a * c);

  Serial.print("c / b = ");       // divide
  Serial.println(c / b);

  Serial.print("b - c = ");       // subtract
  Serial.println(b - c);
}

void loop()                     // we need this to be here even though its empty
{
}
```

# SERIAL COMMUNICATIONS

*Place an LED between Pin 13 and GND

```
/*
 * Serial Read Basic
 * -----------------
 * Blink the pin 13 LED if an 'H' is received over the serial port
 *
 *
 */

int ledPin = 13;   // select the pin for the LED
int val = 0;       // variable to store the data from the serial port

void setup() {
  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);
    }
  }

}
```

# CONTROL STRUCTURES
## IF Statement

**`if`**, which is used in conjunction with a comparison operator, tests whether a certain condition has been reached, such as an input being above a certain number. The format for an if test is:

```
if (someVariable > 50)
{
  // do something here
}
```

# CONTROL STRUCTURES
## IF Statement

```
if (x > 120) digitalWrite(LEDpin, HIGH);

if (x > 120)
digitalWrite(LEDpin, HIGH);

if (x > 120){ digitalWrite(LEDpin, HIGH); }

if (x > 120){
  digitalWrite(LEDpin1, HIGH);
  digitalWrite(LEDpin2, HIGH);
}                                    // all are correct
```

# CONTROL STRUCTURES
## IF / ELSE

**if/else** allows greater control over the flow of code than the basic if statement, by allowing multiple tests to be grouped together. For example, an analog input could be tested and one action taken if the input was less than 500, and another action taken if the input was 500 or greater. The code would look like this:

```
if (pinFiveInput < 500)
{
  // action A
}
else
{
  // action B
}
```

# CONTROL STRUCTURES
## IF / ELSE

**if/elseif** Each test will proceed to the next one until a true test is encountered. When a true test is found, its associated block of code is run, and the program then skips to the line following the entire if/else construction. If no test proves to be true, the default else block is executed, if one is present, and sets the default behavior.

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

# CONTROL STRUCTURES
## IF Statement Comparison Operators:

x == y (x is equal to y)

x != y (x is not equal to y)

x <  y (x is less than y)

x >  y (x is greater than y)

x <= y (x is less than or equal to y)

x >= y (x is greater than or equal to y)

# CONTROL STRUCTURES
## Switch Case

**else** Like if statements, switch...case controls the flow of programs by allowing programmers to specify different code that should be executed in various conditions. In particular, a switch statement compares the value of a variable to the values specified in case statements. When a case statement is found whose value matches that of the variable, the code in that case statement is run.

The break keyword exits the switch statement, and is typically used at the end of each case. Without a break statement, the switch statement will continue executing the following expressions ("falling-through") until a break, or the end of the switch statement is reached.

```
switch (var) {
    case 1:
      //do something when var equals 1
      break;
    case 2:
      //do something when var equals 2
      break;
    default:
      // if nothing else matches, do the default
      // default is optional
  }
}
```

# CONTROL STRUCTURES
## 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.

```
while(expression){
  // statement(s)
}
```

**EXAMPLE**

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

# CONTROL STRUCTURES
## do - while

The **do** loop works in the same manner as the **while** loop, with the exception that the condition is tested at the end of the loop, so the **do** loop will *always* run at least once.

```
do
{
    // statement block
} while (test condition);
```

**EXAMPLE**

```
Example
do
{
  delay(50);              // wait for sensors to
stabilize
  x = readSensors();   // check the sensors

} while (x < 100);
```

# CONTROL STRUCTURES
## break

break is used to exit from a do, for, or while loop, bypassing the normal loop condition. It is also used to exit from a switch statement.

```
Example
for (x = 0; x < 255; x ++)
{
    digitalWrite(PWMpin, x);
    sens = analogRead(sensorPin);
    if (sens > threshold){      // bail out on sensor
detect
        x = 0;
        break;
    }
    delay(50);
}
```

# CONTROL STRUCTURES
## break

break is used to exit from a do, for, or while loop, bypassing the normal loop condition. It is also used to exit from a switch statement.

```
Example
for (x = 0; x < 255; x ++)
{
    digitalWrite(PWMpin, x);
    sens = analogRead(sensorPin);
    if (sens > threshold){      // bail out on sensor
detect
        x = 0;
        break;
    }
    delay(50);
}
```

# CONTROL STRUCTURES
## continue

The continue statement skips the rest of the current iteration of a loop (do, for, or while). It continues by checking the conditional expression of the loop, and proceeding with any subsequent iterations.

```
for (x = 0; x < 255; x ++)
{
    if (x > 40 && x < 120){          // create jump in
values
      continue;
    }

    digitalWrite(PWMpin, x);
    delay(50);
}
```

# CONTROL STRUCTURES
## return

Terminate a function and return a value from a function to the calling function, if desired

**Example**

```
int checkSensor(){
    if (analogRead(0) > 400) {
        return 1;
    else{
        return 0;
    }
}
```