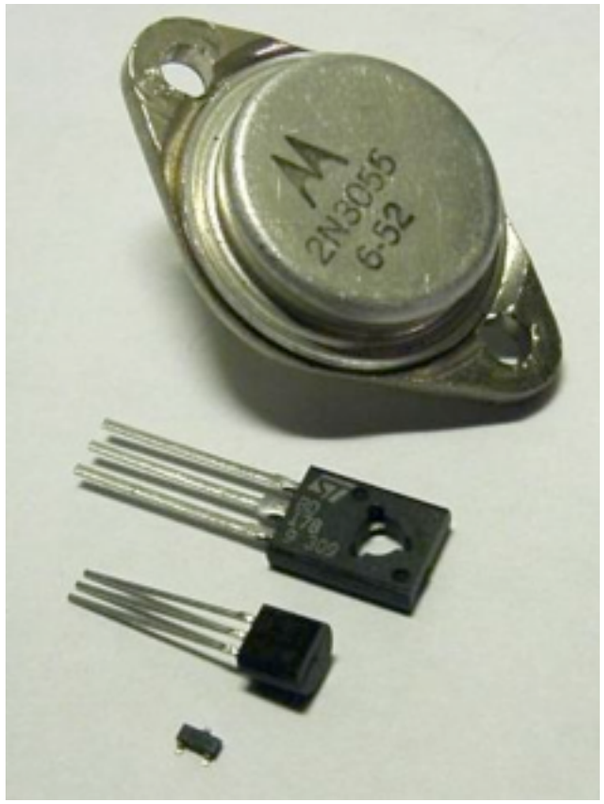
James Turrell

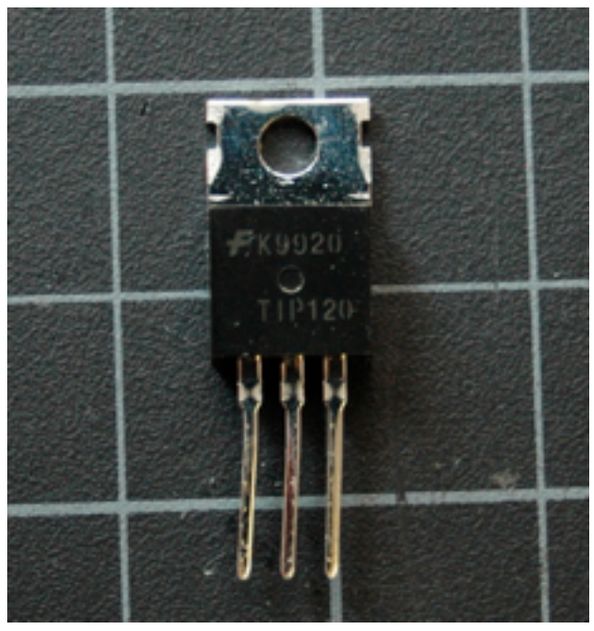
Passive & Active Components

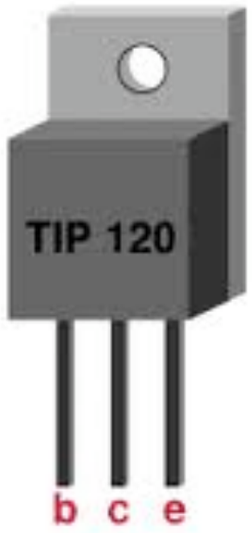


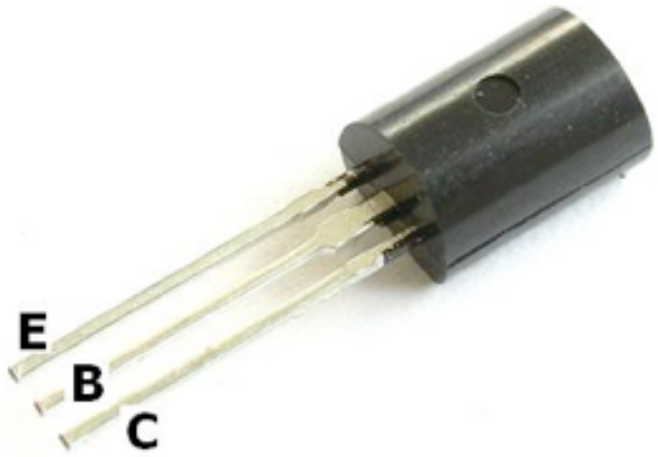
Transistors: are semiconductor devices with three leads instead of two and they allow a small current to control a much larger current.

By adding a second junction to a diode, you end up with a three-layer transistor device called either NPN (not-positive, positive and not-positive layers) or PNP which has two not positive layers of silicon sandwiching a positive layer of silicon.

I call this a silicon sandwich, with NPN having a p-type silicon layer sandwiched between two n-type silicon layers, and vice versa for the PNP. In the NPN transistor the middle layer acts as a gate or faucet, to allow a current to pass, when voltages are applied to the middle layer. This means that transistors can be used as digital logic switches and as amplifiers.







Transistors Continued

NPN transistor basics:

• To start the flow of current from collector to emitter, apply a relatively positive voltage to the base.
• In the schematic symbol, the arrow points from base to emitter and shows the direction of positive current.
• The base must be at least 0.6 volts “more positive” than the emitter, to start the flow.
• The collector must be “more positive” than the emitter.

All-transistor basics

• Never apply a power supply directly across a transistor.
You can burn it out with too much current.

• Protect a transistor with a resistor, in the same way you would protect an LED.

• Avoid reversing the connection of a transistor between positive and negative voltages.
• Sometimes an NPN transistor is more convenient in a circuit; sometimes a PNP happens to fit more easily. They both function as switches and amplifiers, the only difference being that you apply a relatively positive voltage to the base of an NPN transistor, and a relatively negative voltage to the base of a PNP transistor.

• PNP transistors are used relatively seldom, mainly because they were more difficult to manufacture in the early days of semiconductors. People got into the habit of designing circuits around NPN transistors.

• Remember that bipolar transistors amplify current, not voltage. A small fluctuation of current through the base enables a large change in current between emitter and collector.

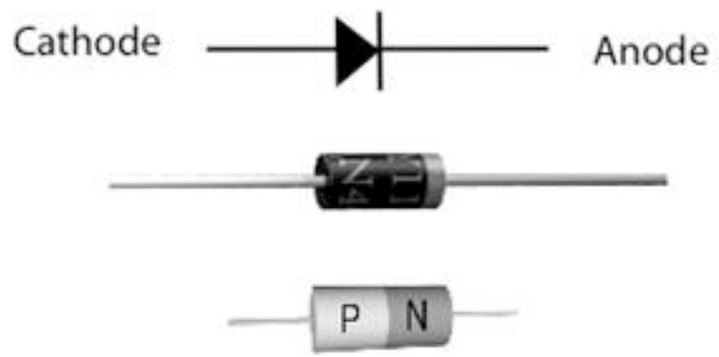
Active Components

<u>Diode:</u> The diode can be thought of as a one-way gate, for electrons. It allows electron current to flow in one direction and not the other (zener diodes are an exception to this).

Diodes are often used in voltage conditioning and helping to change AC voltages into DC voltages by removing one direction of current flow from an AC signal. This is called rectification of voltages. They can also be used to drop voltages down to lower voltage levels and they can be used in motor applications where reverse voltage spikes can potentially damage sensitive parts of your circuit.

A diode is an electronic semiconductor that uses p-type and n-type silicon junctions. That is, the p-type silicon is near the n-type silicon inside a cylindrical package below.

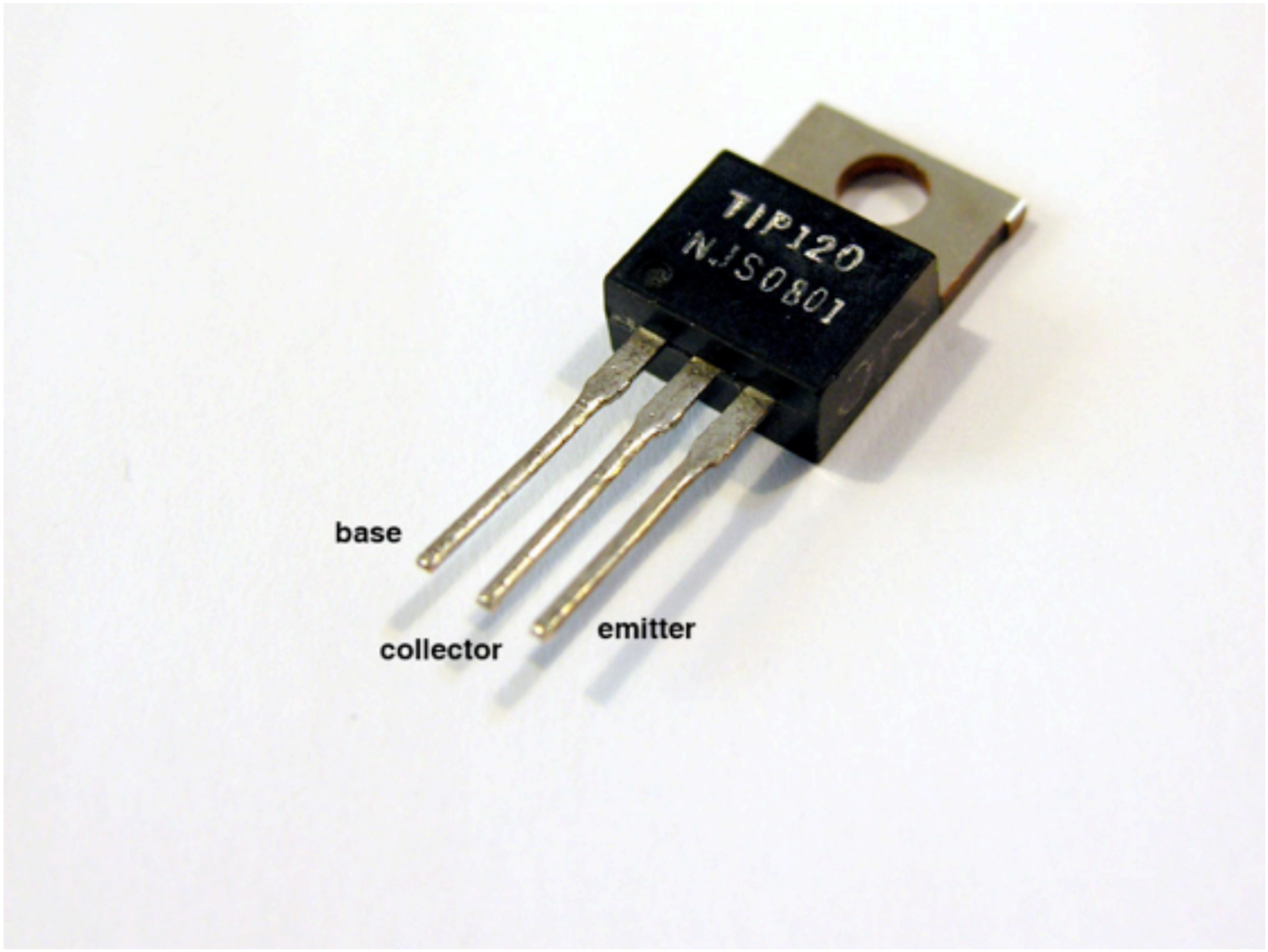
Diodes are often used in voltage conditioning and helping to change AC voltages into DC voltages by removing one direction of current flow from an AC signal. This is called rectification of voltages. They can also be used to drop voltages down to lower voltage levels and they can be used in motor applications where reverse voltage spikes can potentially damage sensitive parts of your circuit.



Building a Transistor Circuit

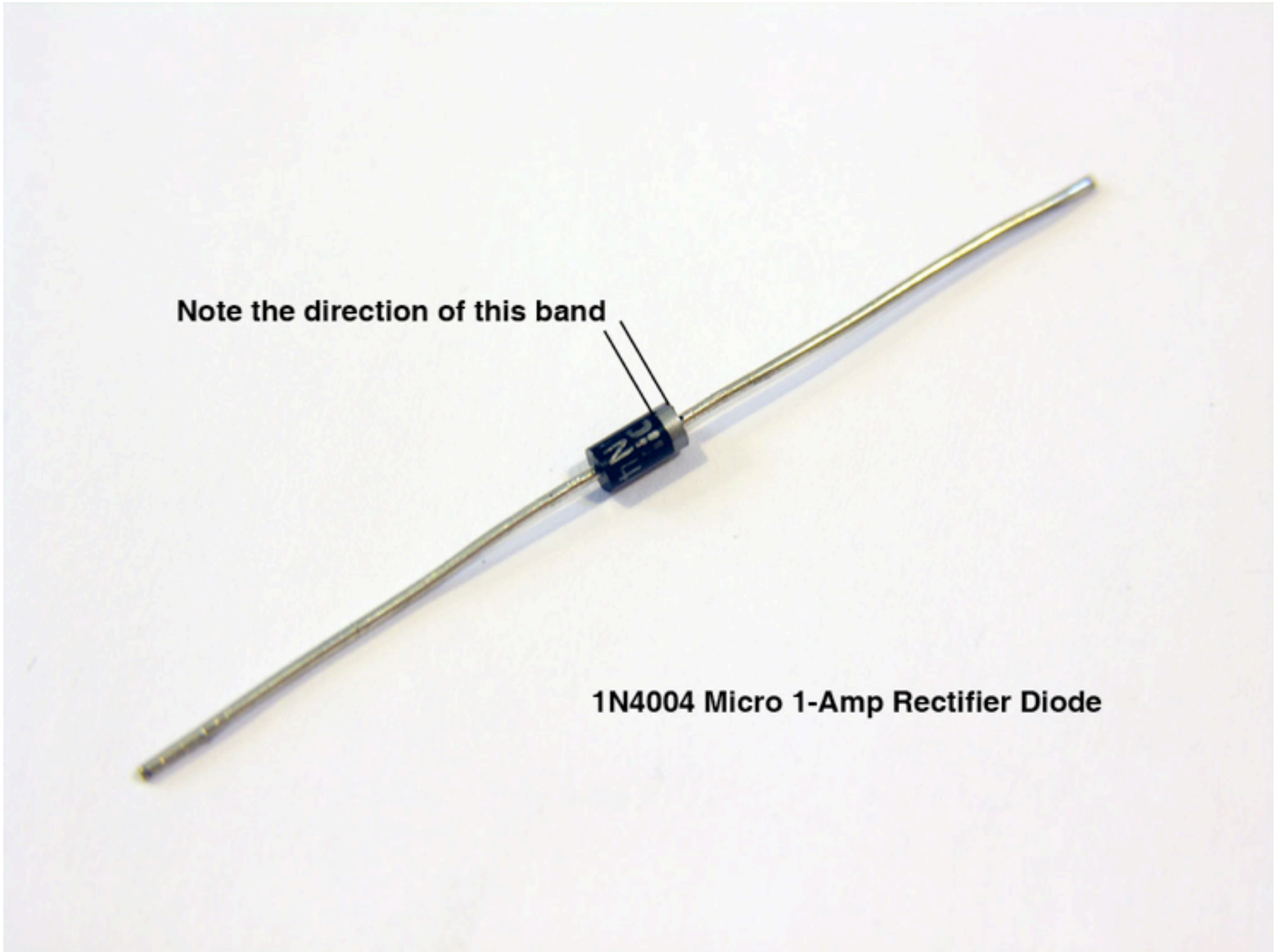
The pins on your Arduino can only be used to power devices that require a small amount of current (e.g. an LED). To power bigger loads (such as larger DC motors) we need to use an additional power supply other than your USB cable. For this exercise we will use the small DC motor that you were given in your kits. This motor can happily run off the Arduino power supply - but many motors cannot because they either draw too much current or they need a voltage higher than 5 volts. We will use this motor to learn what is needed for more powerful motors.

Inside a DC motor is a coil mounted on a shaft between two magnets. As current is applied, the coil is attracted to one magnet and repelled by the other resulting in a spinning motion. However, this also works in reverse - when you move a coil in a magnetic field it can induce current in the coil. This is called 'back' or 'kickback' voltage and it can constantly reset or even damage your Arduino. You can prevent this by putting a 'snubber' or 'kickback' diode in your circuit any time you are using an inductive load (DC motor, solenoid, etc.)



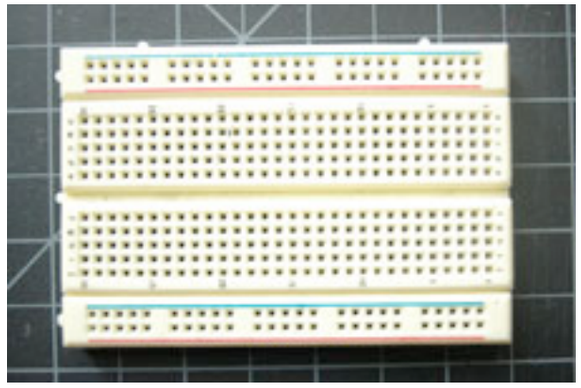
The pins on your Arduino can only be used to power devices that require a small amount of current (e.g. an LED). To power bigger loads (such as larger DC motors) we need to use an additional power supply other than your USB cable. For this exercise we will use the small DC motor that you were given on week 2. This motor can happily run off the Arduino power supply - but many motors cannot because they either draw too much current or they need a voltage higher than 5 volts. We will use this motor to learn what is needed for more powerful motors.

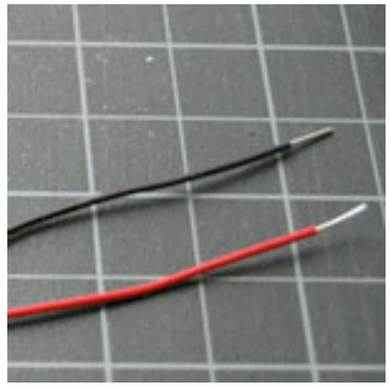
Inside a DC motor is a coil mounted on a shaft between two magnets. As current is applied, the coil is attracted to one magnet and repelled by the other resulting in a spinning motion. However, this also works in reverse - when you move a coil in a magnetic field it can induce current in the coil. This is called 'back' or 'kickback' voltage and it can constantly reset or even damage your Arduino. You can prevent this by putting a 'snubber' or 'kickback' diode in your circuit any time you are using an inductive load (DC motor, solenoid, etc.)

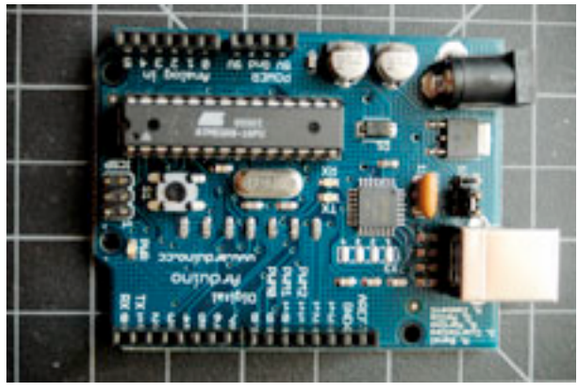


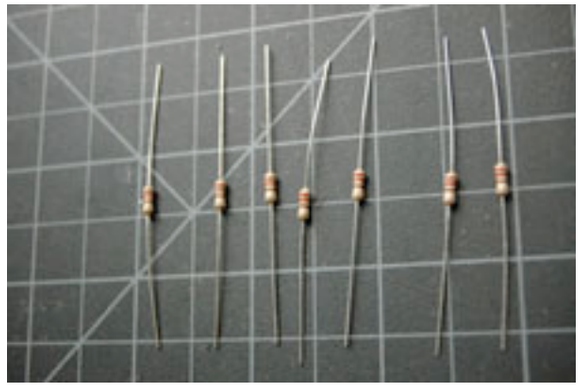
Once a motor starts turning its inertia keeps it spinning and it can generate 'back voltage'. A kickback diode acts like a one way valve and routes that voltage back into the motor so it can’t damage the rest of the circuit. Diodes are like LED’s in that they have polarity and will only allow electricity to flow in one direction. We will be using a **1N4004 Micro 1 Amp Rectifier Diode**. This diode can handle currents up to 1 amp. Electricity will flow toward the silver band on one end of the diode.

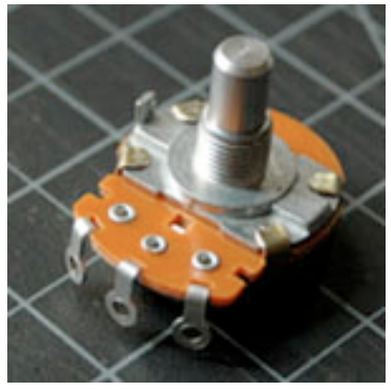
What you Need

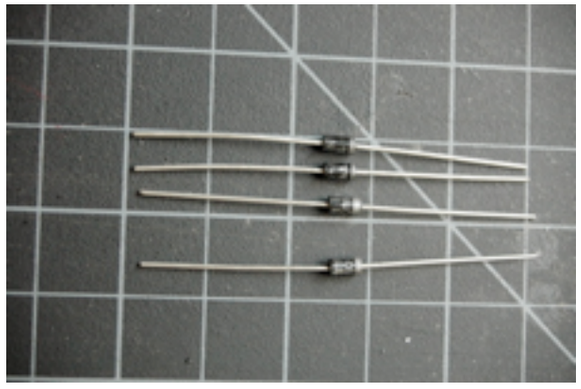






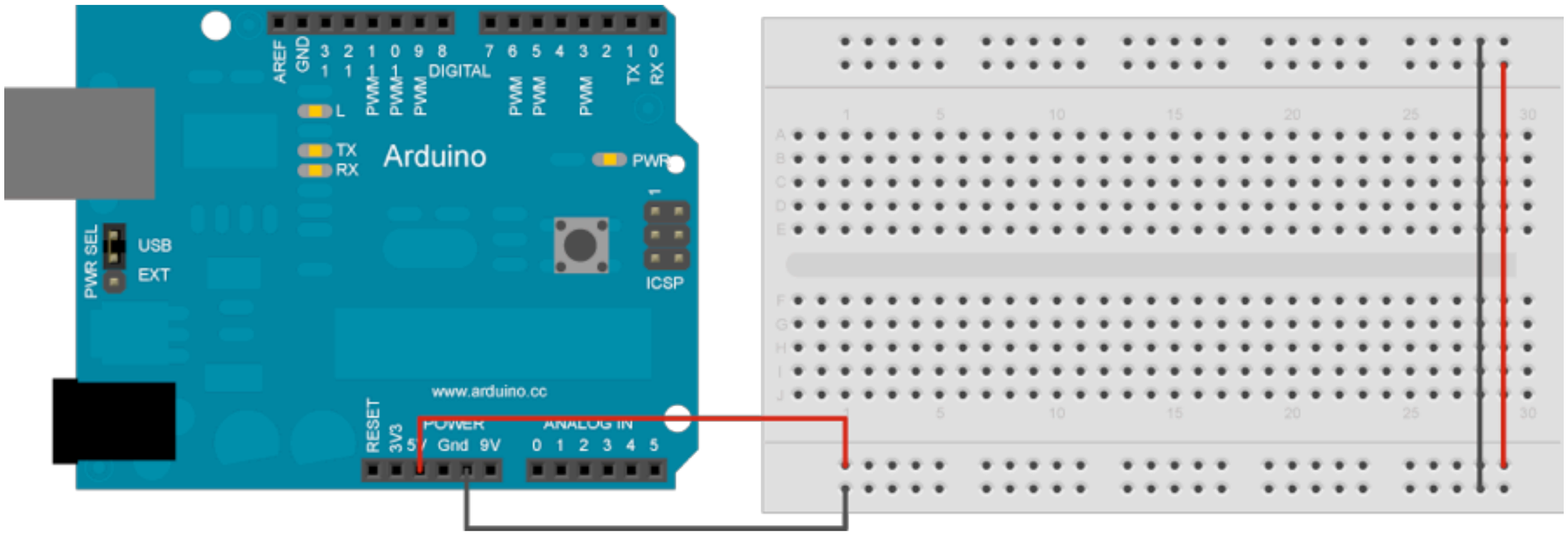






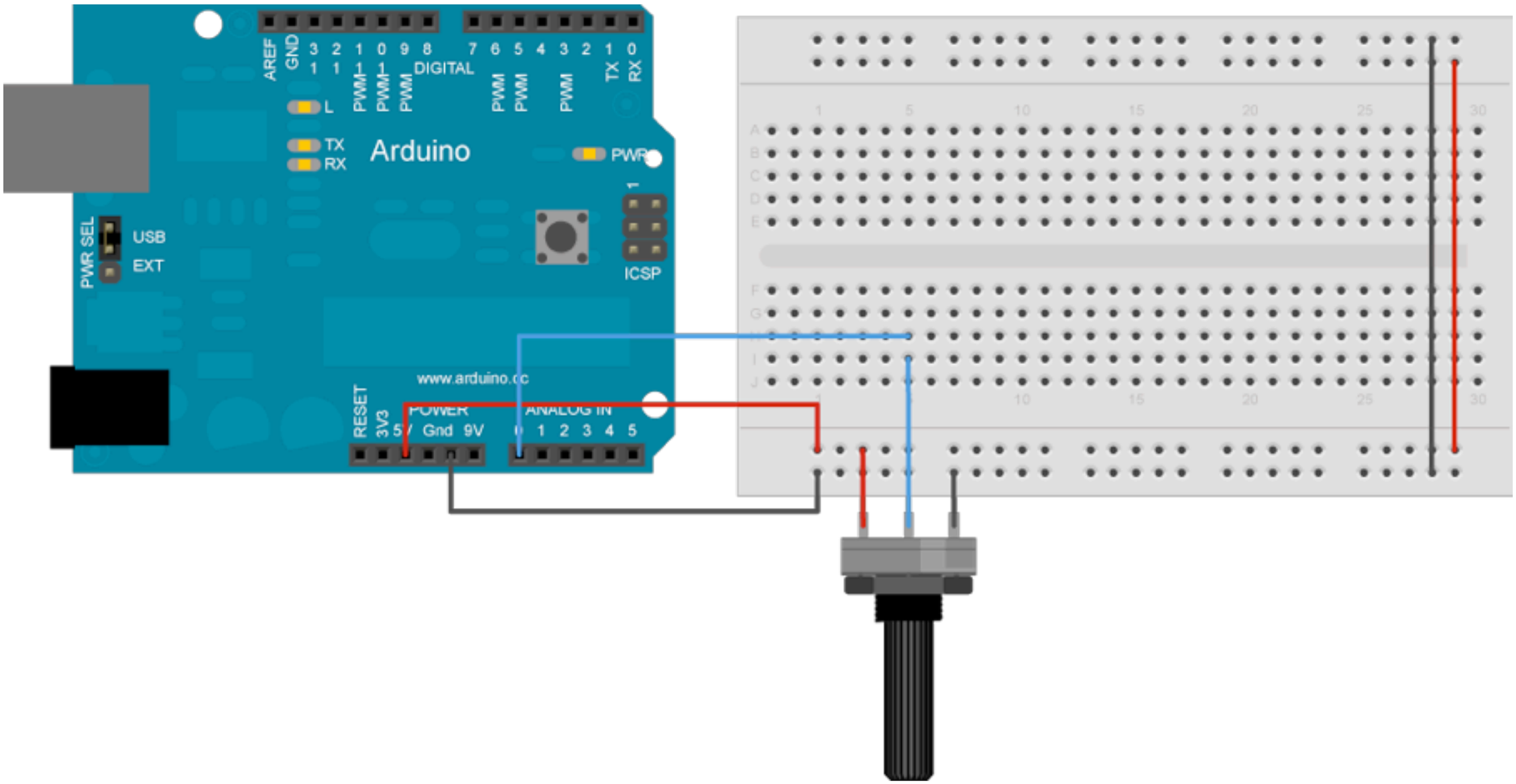
Prepare the breadboard

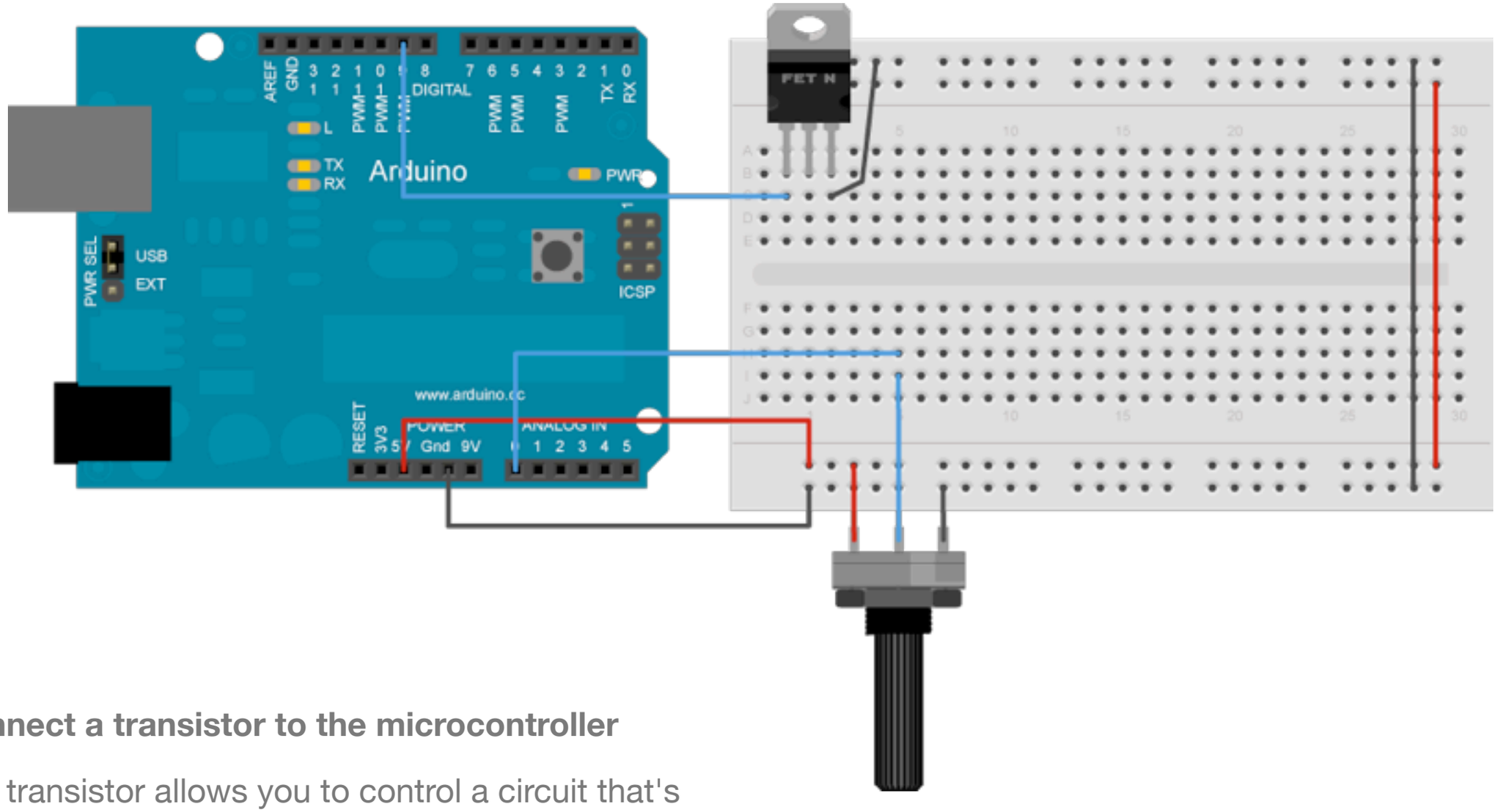
Connect power and ground on the breadboard to power and ground from the microcontroller. On the Arduino module, use the 5V and any of the ground connections:



Add a potentiometer

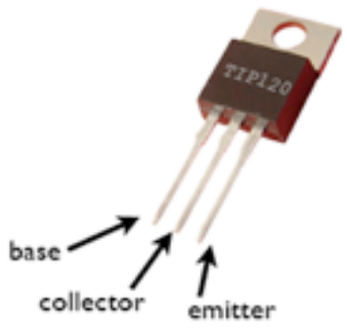
Connect a potentiometer to analog in pin 0 of the module. You'll use this later to control the output, whether it's a motor or a light.



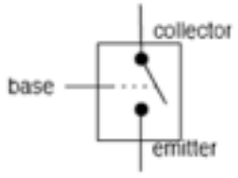


Connect a transistor to the microcontroller

The transistor allows you to control a circuit that's carrying higher current and voltage from the microcontroller. It acts as an electronic switch. The one you're using for this lab is an NPN-type transistor called a TIP120. It's designed for switching high-current loads. It has three connections, the base, the collector, and the emitter. The base is connected to the microcontroller's output. The high-current load (i.e. the motor or light) is attached to its power source, and then to the collector of the transistor. The emitter of the transistor is connected to ground.



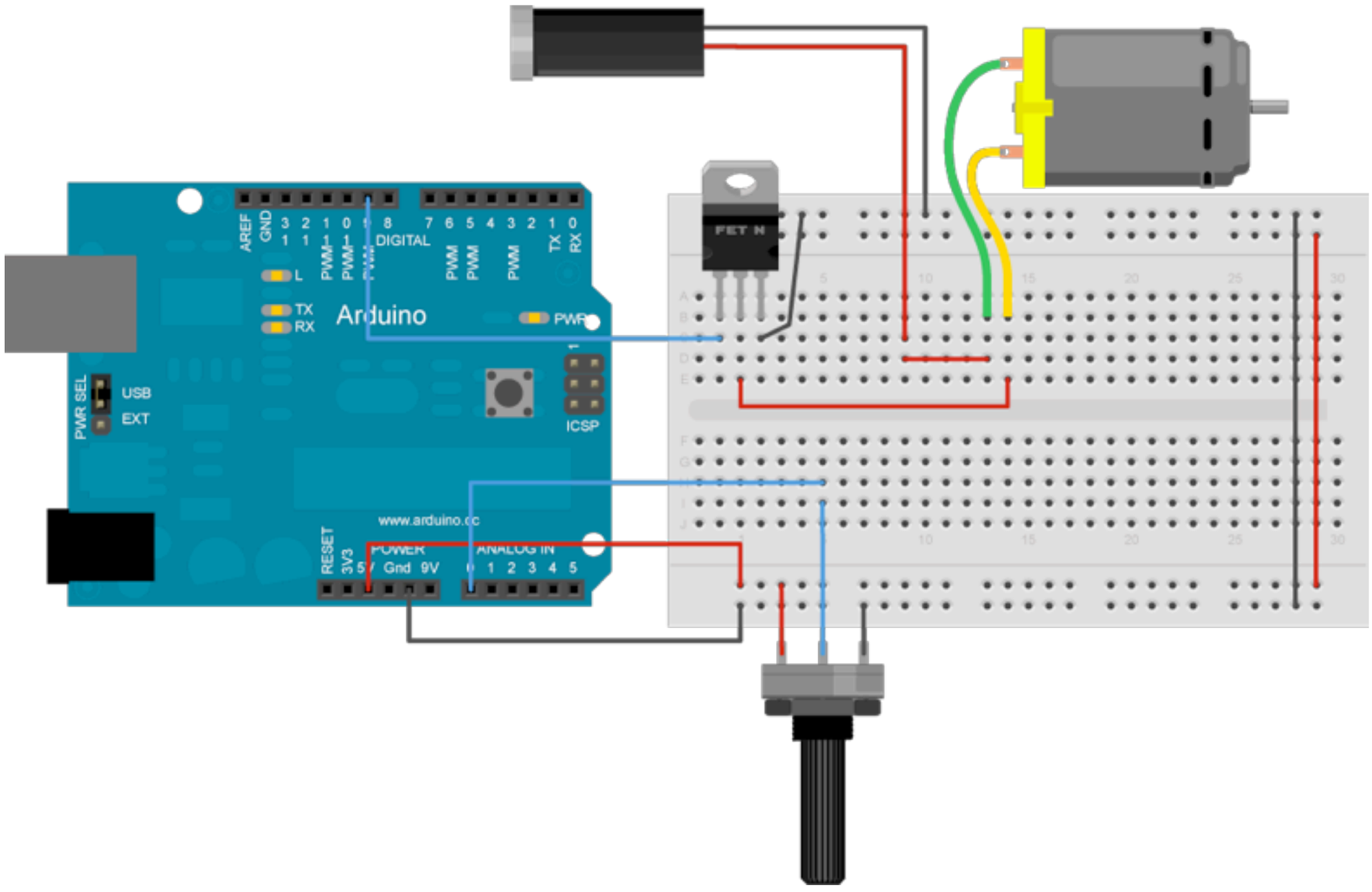
schematic symbol



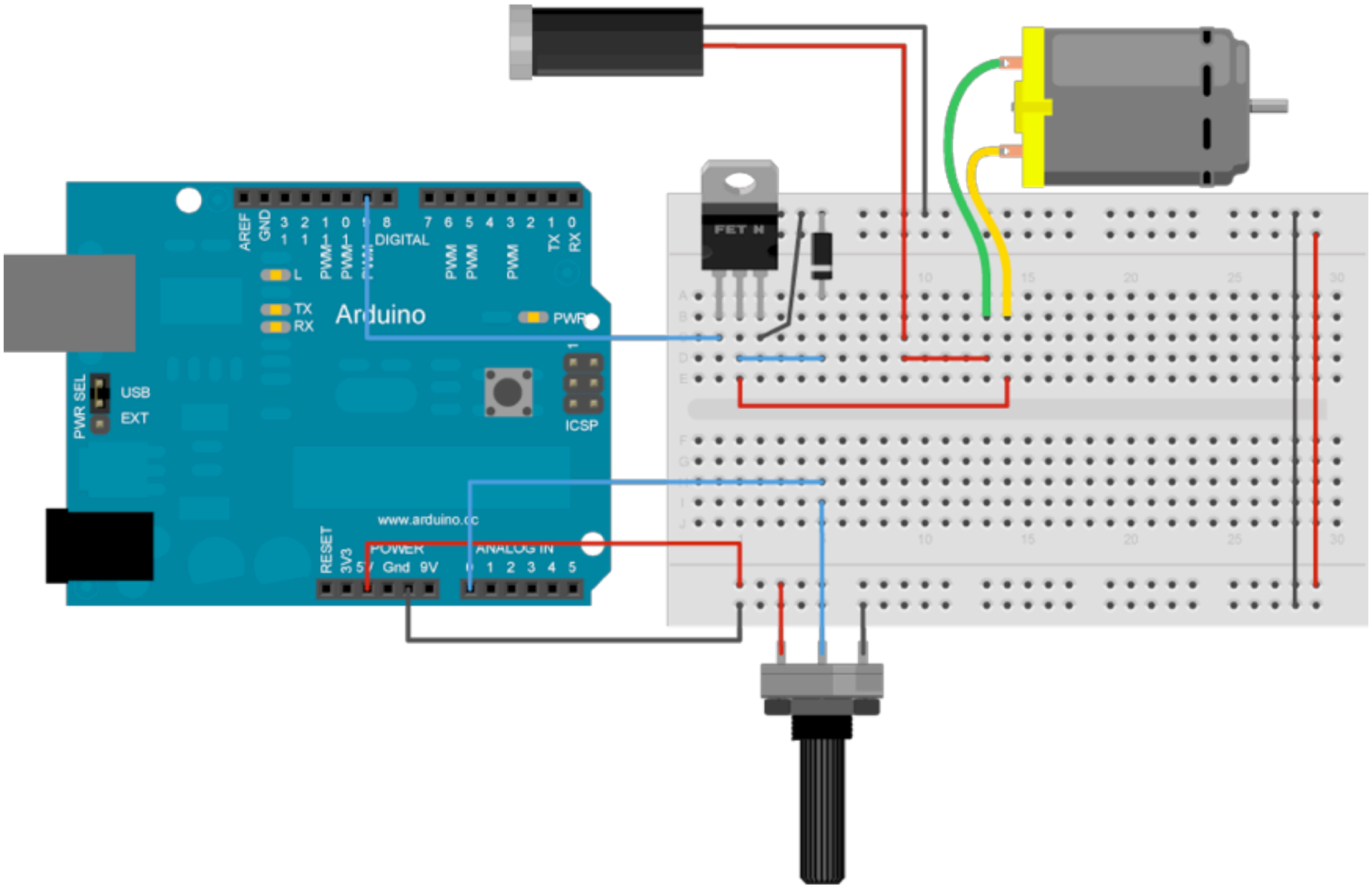
how it kind of works

Connect a motor and power supply

Attach a DC motor to the collector of the transistor. Most motors will require more amperage than the microcontroller can supply, so you will need to add a separate power supply as well. If your motor runs on around 9V, you could use a 9V battery. A 5V motor might run on 4 AA batteries. a 12V battery may need a 12V wall wart, or a 12V battery. The ground of the motor power supply should connect to the ground of the microcontroller, on the breadboard.



FInally, add diode in parallel with the collector and emitter of the transistor, pointing away from ground. The diode to protects the transistor from back voltage generated when the motor shuts off, or if the motor is turned in the reverse direction.



Download Code TIP120 Basic Motor

Breaking Down the Code

```
const int potPin = 0;              // Analog in 0 connected to the potentiometer
const int transistorPin = 9;       // connected to the base of the transistor
int potValue = 0;                  // value returned from the potentiometer
```

The **const** keyword stands for constant. It is a variable *qualifier* that modifies the behavior of the variable, making a variable "*read-only*". This means that the variable can be used just as any other variable of its type, but its value cannot be changed. You will get a compiler error if you try to assign a value to a **const** variable.

Constants defined with the *const* keyword obey the rules of *variable scoping* that govern other variables. This, and the pitfalls of using*#define*, makes the *const* keyword a superior method for defining constants and is preferred over using *#define*.

Breaking Down the Code

```
void setup() {
  // set  the transistor pin as output:
  pinMode(transistorPin, OUTPUT);
}
```

Above we named Pin 9 as the pin connected to the transistor, now we are making that pin an output because current will flow out of this pin to our motor

Breaking Down the Code

```
void loop() {
  // read the potentiometer, convert values to 0 - 255:
  potValue = analogRead(potPin) / 4;
  // use that to control the transistor:
  analogWrite(9, potValue);
}
```

potValue was named in the beginning of the code. We will take analog readings from the potentiometer (potValue) and divide the readings by four to get values between 0-255 rather than 0-1023.

We will use the data from the potentiometer to control the motor's speed

CHALLENGE

-Replace Motor With Solenoid
-Make the Solenoid move via code without using the potentiometer
-Create a Class Solenoid Concert

SWITCH CASE

Examples > Control Structures

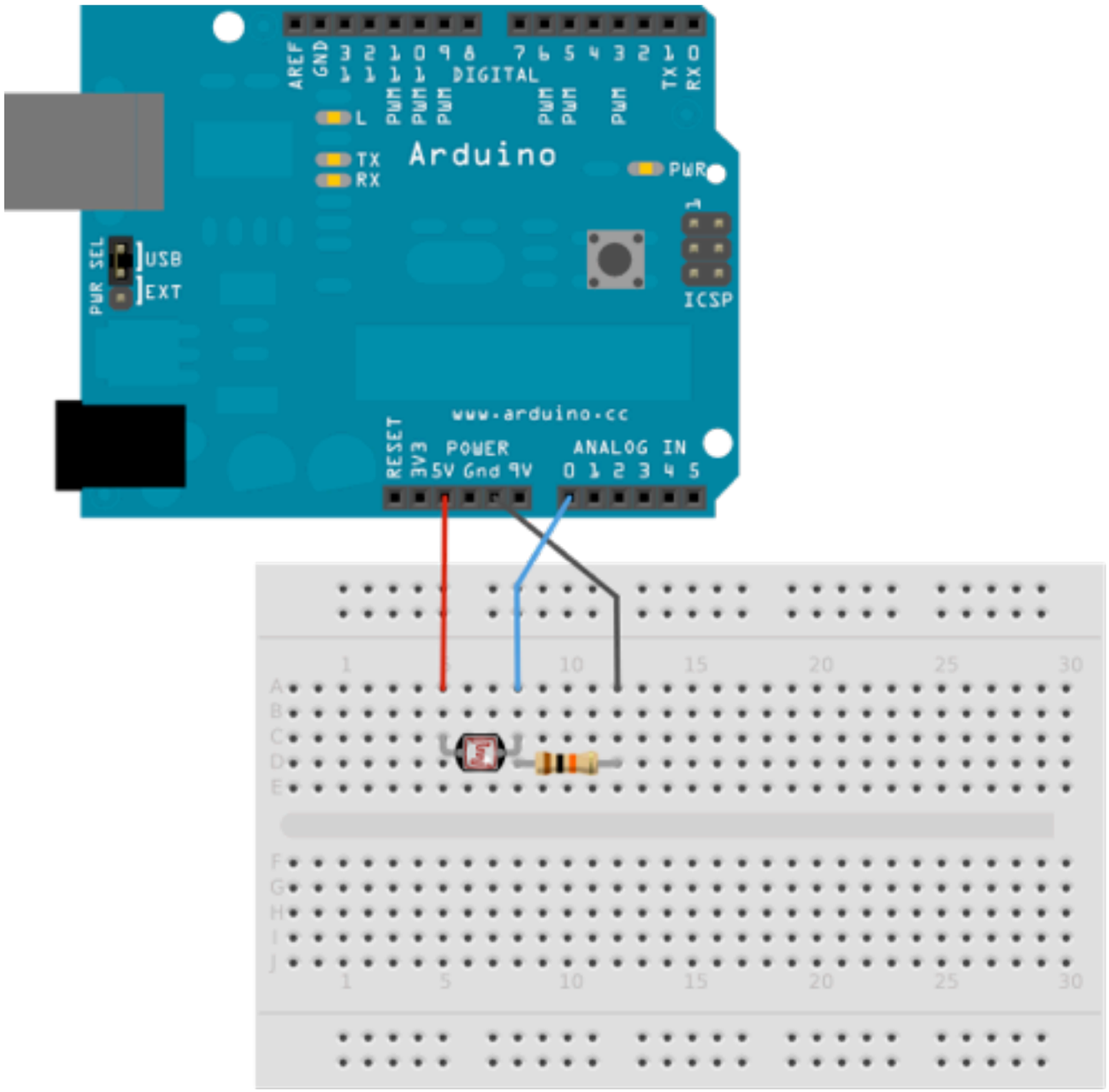
Switch (case) Statement, used with sensor input

An if statement allows you to choose between two discrete options, TRUE or FALSE. When there are more than two options, you can use multiple if statements, or you can use the **switch** statement. Switch allows you to choose between several discrete options. This tutorial shows you how to use it to switch between four desired states of a photo resistor: really dark, dim, medium, and bright.

This program first reads the photoresistor. Then it uses the `map()` function to map its output to one of four values: 0, 1, 2, or 3. Finally, it uses the `switch()` statement to print one of four messages back to the computer depending on which of the four values is returned.

Hardware Required

- Arduino Board
- (1) photocell, or analog sensor
- (1) 10k ohm resistors
- breadboard
- hook-up wire



Circuit

The photoresistor is connected to analog in pin 0 using a voltage divider circuit. A 10Kilohm resistor makes up the other side of the voltage divider, running from analog in 0 to ground. The `analogRead()` function returns a range of about 0 to 600 from this circuit in a reasonably lit indoor space.

Code

```
// these constants won't change:
const int sensorMin = 0;      // sensor minimum, discovered through
experiment
const int sensorMax = 600;    // sensor maximum, discovered through
experiment

void setup() {
  // initialize serial communication:
  Serial.begin(9600);
}

void loop() {
  // read the sensor:
  int sensorReading = analogRead(A0);
  // map the sensor range to a range of four options:
  int range = map(sensorReading, sensorMin, sensorMax, 0, 3);

  // do something different depending on the
  // range value:
  switch (range) {
  case 0:    // your hand is on the sensor
    Serial.println("dark");
    break;
  case 1:    // your hand is close to the sensor
    Serial.println("dim");
    break;
  case 2:    // your hand is a few inches from the sensor
    Serial.println("medium");
    break;
  case 3:    // your hand is nowhere near the sensor
    Serial.println("bright");
    break;
  }

}
```

If Statement (Conditional Statement)

"Examples > Control Structures"

The **if()** statement is the most basic of all programming control structures. It allows you to make something happen or not depending on whether a given condition is true or not. It looks like this:

```
if (someCondition) {
   // do stuff if the condition is true
}
```

There is a common variation called if-else that looks like this:

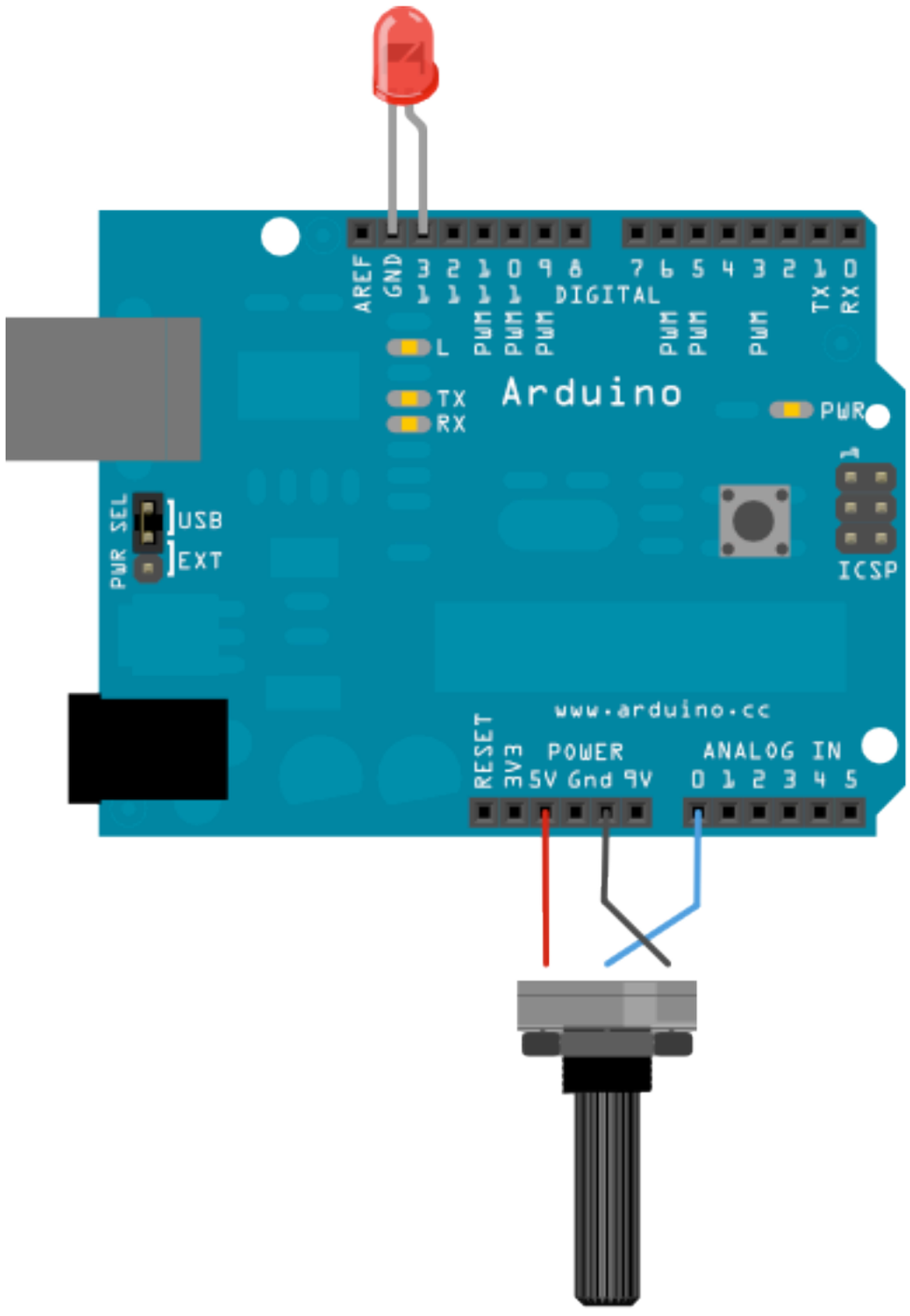
```
if (someCondition) {
   // do stuff if the condition is true
} else {
   // do stuff if the condition is false
}
```

If Statement (Conditional Statement)

There's also the else-if, where you can check a second condition if the first is false:

```
if (someCondition) {
   // do stuff if the condition is true
} else if (anotherCondition) {
   // do stuff only if the first condition is false
   // and the second condition is true
}
```

You'll use if statements all the time. The example below turns on an LED on pin 13 (the built-in LED on many Arduino boards) if the value read on an analog input goes above a certain threshold.



Hardware Required

- Arduino Board
- (1) Potentiometer or variable resistor
- (1) 220 ohm resistor
- (1) LED
- hook-up wire

Code

In the code below, a variable called `analogValue` is used to store the data collected from a potentiometer connected to the Arduino on `analogPin 0`. This data is then compared to a threshold value. If the analog value is found to be above the set threshold the LED connected to digital pin 13 is turned on. If analogValue is found to be < threshold, the LED remains off.

```
// These constants won't change:
const int analogPin = A0;     // pin that the sensor is attached to
const int ledPin = 13;        // pin that the LED is attached to
const int threshold = 400;    // an arbitrary threshold level that's in the
range of the analog input

void setup() {
  // initialize the LED pin as an output:
  pinMode(ledPin, OUTPUT);
  // initialize serial communications:
  Serial.begin(9600);
}

void loop() {
  // read the value of the potentiometer:
  int analogValue = analogRead(analogPin);

  // if the analog value is high enough, turn on the LED:
  if (analogValue > threshold) {
    digitalWrite(ledPin, HIGH);
  }
  else {
    digitalWrite(ledPin,LOW);
  }

  // print the analog value:
  Serial.println(analogValue);

}
```

For Loop

For Loop

For Loop (aka The Knight Rider)

Often you want to iterate over a series of pins and do something to each one. For instance, this example blinks 6 LEDs attached the Arduino by using a **for()** loop to cycle back and forth through digital pins 2-7. The LEDS are turned on and off, in sequence, by using both the digitalWrite() and delay() functions .

We also call this example "Knight Rider" in memory of a TV-series from the 80's where David Hasselhoff had an AI machine named KITT driving his Pontiac. The car had been augmented with plenty of LEDs in all possible sizes performing flashy effects. In particular, it had a display that scanned back and forth across a line, as shown in this exciting fight between KITT and KARR. This example duplicates the KITT display.

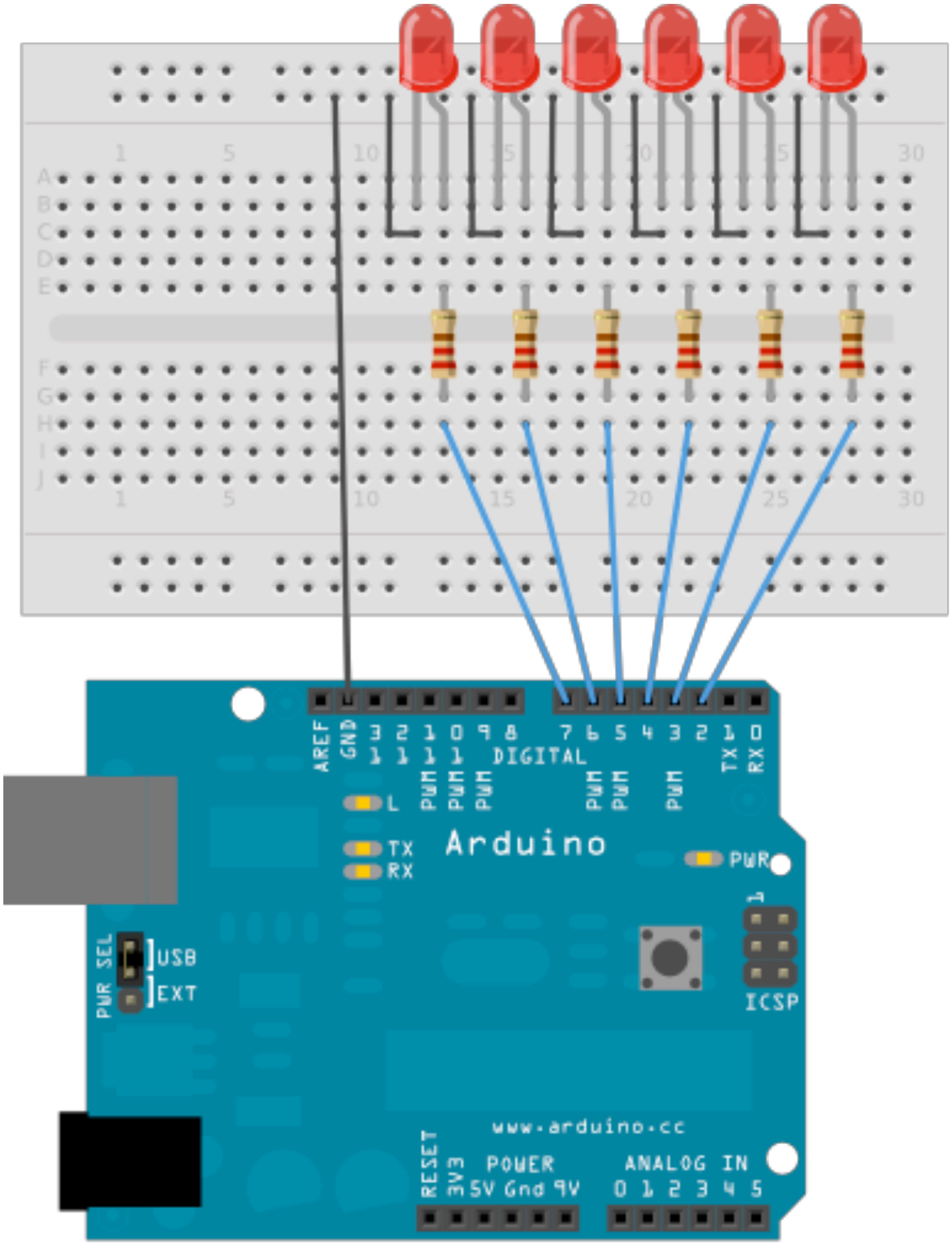
Hardware Required

- Arduino Board
- (6) 220 ohm resistors
- (6) LEDs
- hook-up wire
- breadboard

Circuit

Connect six LEDS, with 220 ohm resistors in series, to digital pins 2-7 on your Arduino.

For Loop


1 5 10 15 20 25 30
A
B
C
D
E
F
G
H
I
J
AREF GND 13 12 11 10 9 8 7 6 5 4 3 2 1 0
DIGITAL
PWM
TX RX
L
TX
RX
Arduino
PWR
PWR SEL
USB
EXT
ICSP
www.arduino.cc
RESET 3V3 POWER 5V Gnd Vin
ANALOG IN 0 1 2 3 4 5

For Loop

Code

The code below begins by utilizing a for() loop to assign digital pins 2-7 as outputs for the 6 LEDs used.

In the main loop of the code, two for() loops are used to loop incrementally, stepping through the LEDs, one by one, from pin 2 to pin seven. Once pin 7 is lit, the process reverses, stepping back down through each LED.

```
int timer = 100;           // The higher the number, the slower the timing.

void setup() {
  // use a for loop to initialize each pin as an output:
  for (int thisPin = 2; thisPin < 8; thisPin++)  {
    pinMode(thisPin, OUTPUT);
  }
}
```

++ means Increment or decrement a variable. In this cae thisPin++ means start at pin 2 and go up to 8

pin 2 ++ pin 3 ++ pin 4 ++ pin 5 ++ pin 6 ++ pin 7++ pin 8 STOP because thisPin is less than 9

Arrays

```
void loop() {
  // loop from the lowest pin to the highest:
  for (int thisPin = 2; thisPin < 8; thisPin++) {
    // turn the pin on:
    digitalWrite(thisPin, HIGH);
    delay(timer);
    // turn the pin off:
    digitalWrite(thisPin, LOW);
}
```

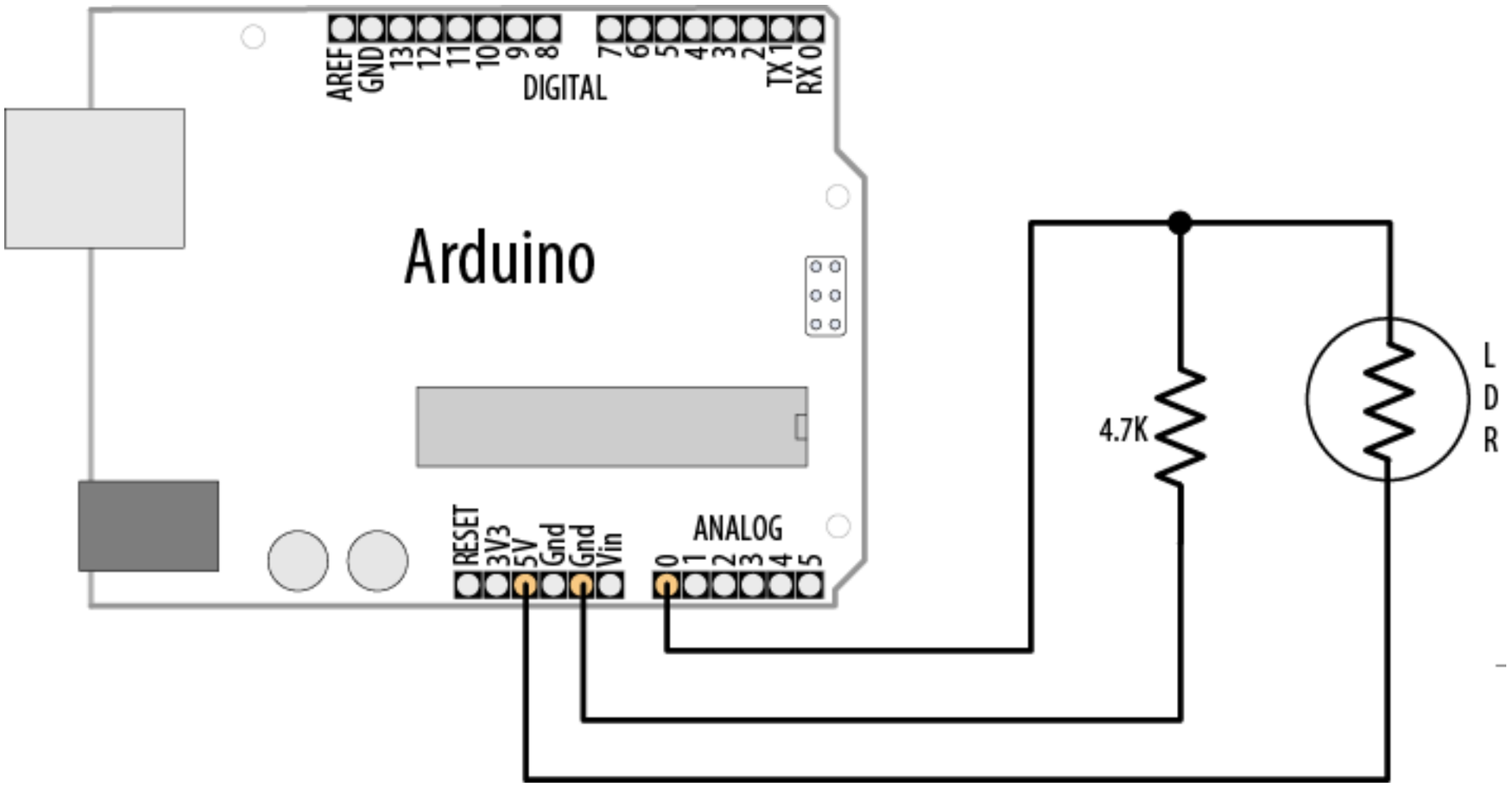
Arrays

```
// loop from the highest pin to the lowest:
for (int thisPin = 7; thisPin >= 2; thisPin--) {
  // turn the pin on:
  digitalWrite(thisPin, HIGH);
  delay(timer);
  // turn the pin off:
  digitalWrite(thisPin, LOW);
  }
}
```

Analog Input + Control

The sketch is based on the LED blinking code from the previous recipe, but instead of using a fixed delay, the rate is determined by a light-sensitive sensor called a light dependent resistor or LDR.If you want to view the value of the rate variable on your computer, you can print this to the Arduino Serial Monitor as shown in the revised loop code that follows. The sketch will display the blink rate in the Serial Monitor. You open the Serial Monitor window in the Arduino IDE.

Open LDR_Code_Example



Example from the Arduino Cookbook

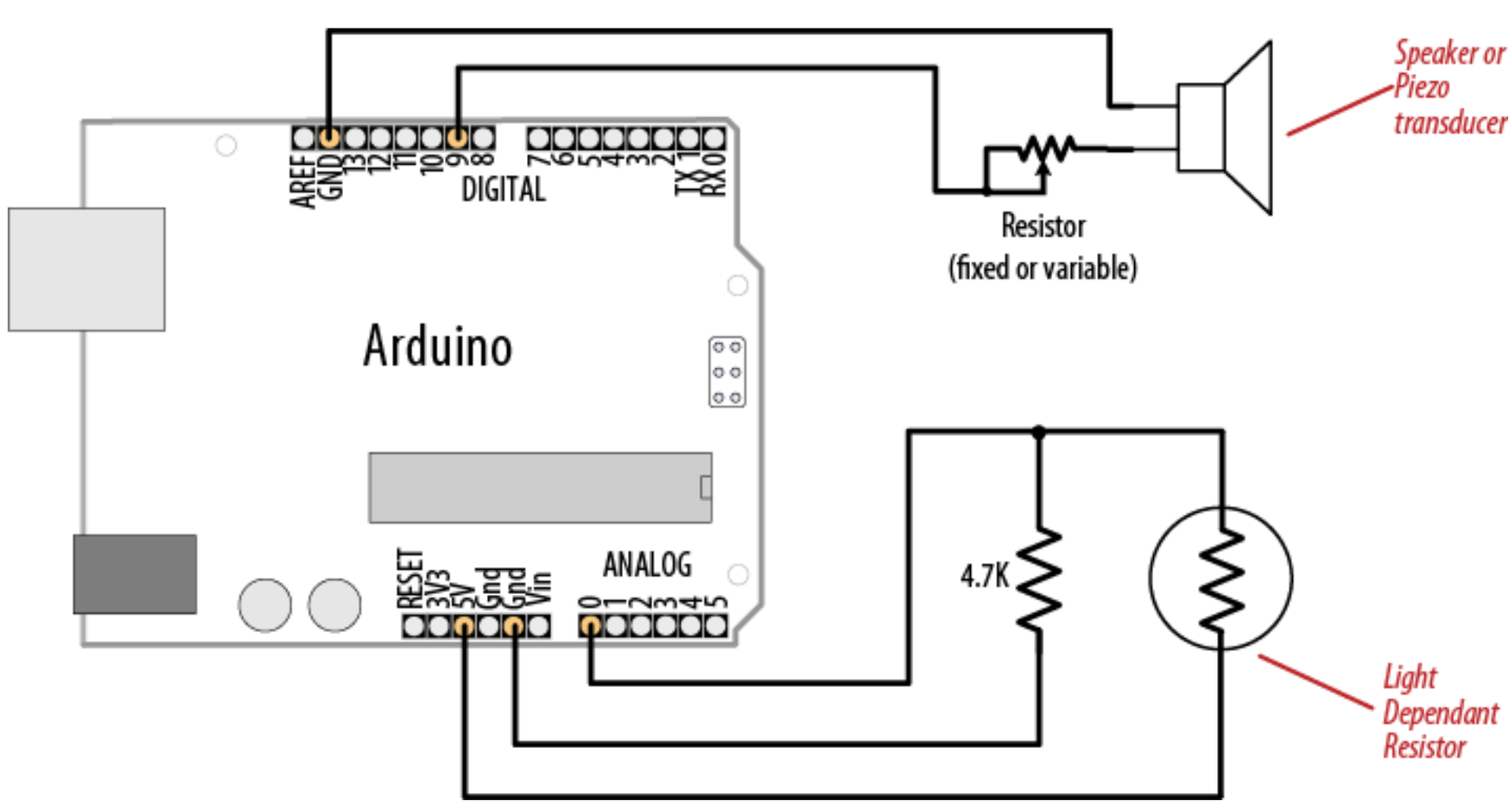
Analog Input + Control

Lets try this with sound!

A Few things will need changed in the circuit and code-

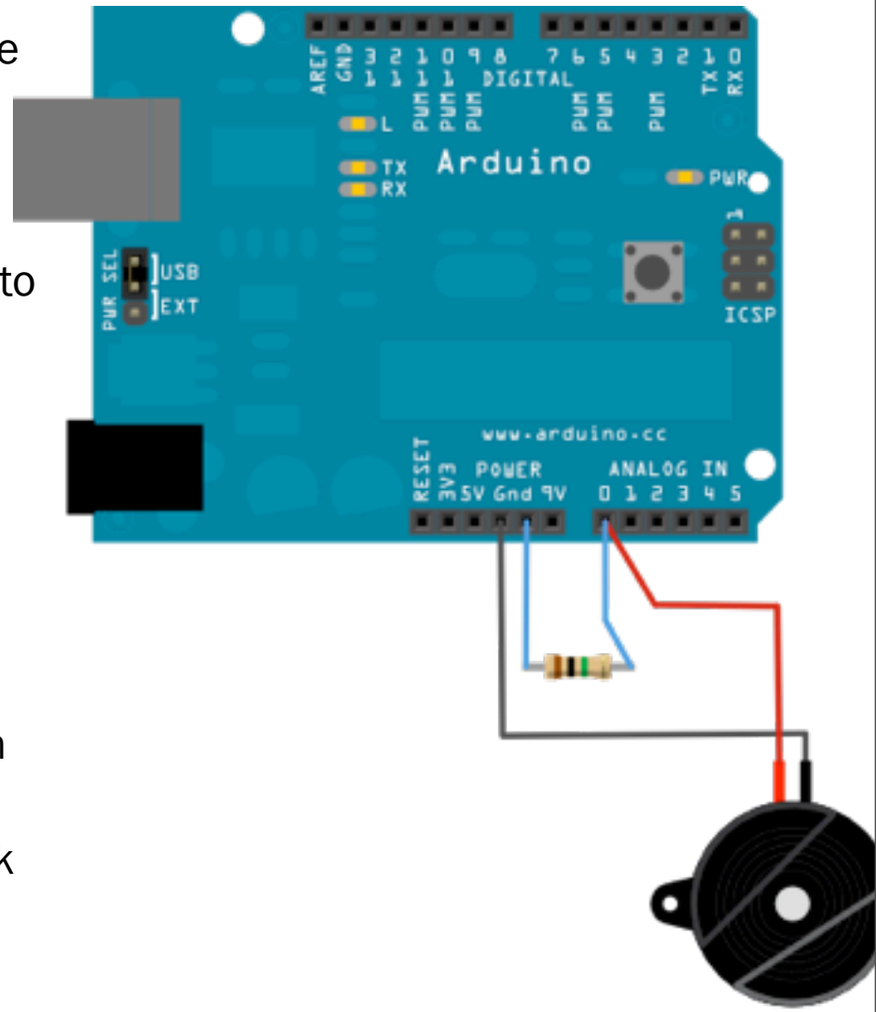
You will need to increase the on/off rate on the pin to a frequency in the audio spectrum. This is achieved, as shown in the following code, by dividing the rate by 100 in the line after the map function.

Open LDR_WithSound



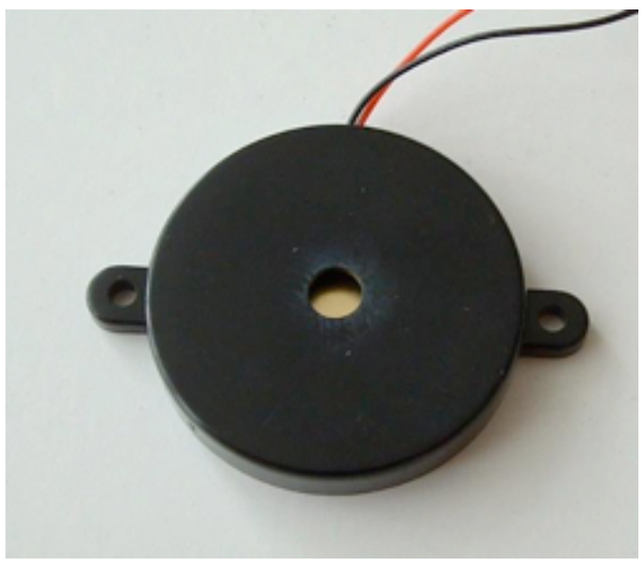
Knock Sensor Circuit

Here we use a Piezo element to detect sound, what will allow us to use it as a knock sensor. We are taking advantage of the processors capability to read analog signals through its ADC - analog to digital converter. These converters read a voltage value and transform it into a value encoded digitally. In the case of the Arduino boards, we transform the voltage into a value in the range 0..1024. 0 represents 0volts, while 1024 represents 5volts at the input of one of the six analog pins.

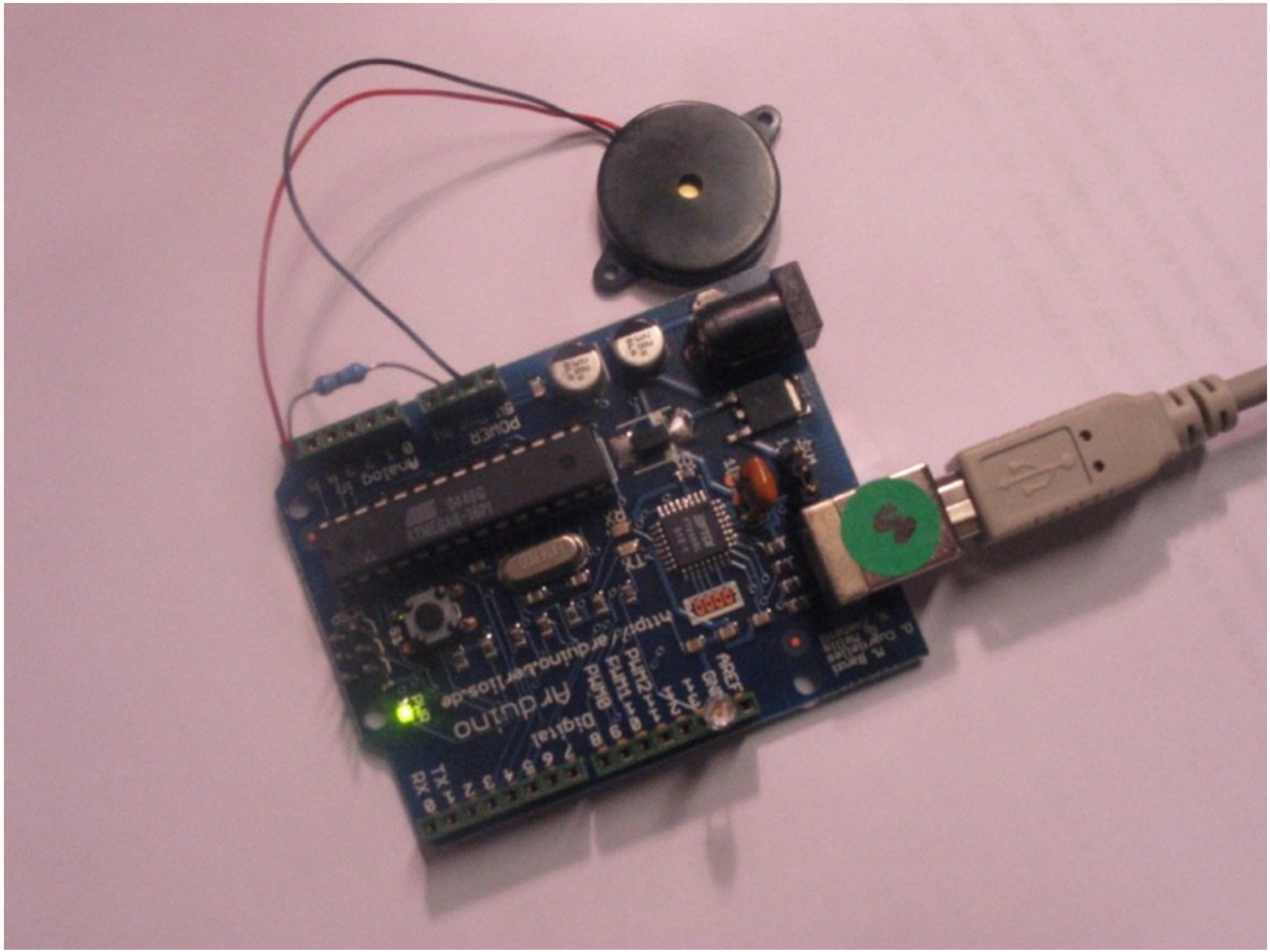


A Piezo is nothing but an electronic device that can both be used to play tones and to detect tones. In our example we are plugging the Piezo on the analog input pin number 0, that supports the functionality of reading a value between 0 and 5volts, and not just a plain HIGH or LOW.

The other thing to remember is that Piezos have polarity, commercial devices are usually having a red and a black wires indicating how to plug it to the board. We connect the black one to ground and the red one to the input. We also have to connect a resistor in the range of the Megaohms in parallel to the Piezo element; in the example we have plugged it directly in the female connectors. Sometimes it is possible to acquire Piezo elements without a plastic housing, then they will just look like a metallic disc and are easier to use as input sensors.



Open Knock Sensor Code



Arduino --> File --> Examples -->Sensors --> Knock

Code Review

Void Setup

The setup() function is called when a sketch starts. Use it to initialize variables, pin modes, start using libraries, etc. The setup function will only run once, after each powerup or reset of the Arduino board.

Void Loop

After creating a setup() function, which initializes and sets the initial values, the loop() function does precisely what its name suggests, and loops consecutively, allowing your program to change and respond. Use it to actively control the Arduino board.

pinMode

Configures the specified pin to behave either as an input or an output.

digitalWrite

Write a HIGH or a LOW value to a digital pin.

If the pin has been configured as an OUTPUT with pinMode(), its voltage will be set to the corresponding value: 5V (or 3.3V on 3.3V boards) for HIGH, 0V (ground) for LOW.

If the pin is configured as an INPUT, writing a HIGH value with digitalWrite() will enable an internal 20K pullup resistor (see the tutorial on digital pins). Writing LOW will disable the pullup. The pullup resistor is enough to light an LED dimly, so if LEDs appear to work, but very dimly, this is a likely cause. The remedy is to set the pin to an output with the pinMode() function.

digitalRead

Reads the value from a specified digital pin, either HIGH or LOW.

Code Review

analogRead

Reads the value from the specified analog pin. The Arduino board contains a 6 channel (8 channels on the Mini and Nano, 16 on the Mega), 10-bit analog to digital converter. This means that it will map input voltages between 0 and 5 volts into integer values between 0 and 1023. This yields a resolution between readings of: 5 volts / 1024 units or, .0049 volts (4.9 mV) per unit. The input range and resolution can be changed using analogReference().

It takes about 100 microseconds (0.0001 s) to read an analog input, so the maximum reading rate is about 10,000 times a second.

analogWrite

Writes an analog value (PWM wave) to a pin. Can be used to light a LED at varying brightnesses or drive a motor at various speeds. After a call to **analogWrite()**, the pin will generate a steady square wave of the specified duty cycle until the next call to **analogWrite()** (or a call to **digitalRead()** or **digitalWrite()** on the same pin). The frequency of the PWM signal is approximately 490 Hz.

On most Arduino boards (those with the ATmega168 or ATmega328), this function works on pins 3, 5, 6, 9, 10, and 11. On the Arduino Mega, it works on pins 2 through 13. Older Arduino boards with an ATmega8 only support analogWrite() on pins 9, 10, and 11. You do not need to call pinMode() to set the pin as an output before calling analogWrite().

delay

Pauses the program for the amount of time (in miliseconds) specified as parameter. (There are 1000 milliseconds in a second.)

millis

Returns the number of milliseconds since the Arduino board began running the current program. This number will overflow (go back to zero), after approximately 50 days.

const

The **const** keyword stands for constant. It is a variable *qualifier* that modifies the behavior of the variable, making a variable "*read-only*". This means that the variable can be used just as any other variable of its type, but its value cannot be changed. You will get a compiler error if you try to assign a value to a **const** variable.

Constants defined with the *const* keyword obey the rules of *variable scoping* that govern other variables. This, and the pitfalls of using*#define*, makes the *const* keyword a superior method for defining constants and is preferred over using*#define*.

Code Review

++ Increment of a variable

-- Decrement a variable

&&

True only if both operands are true, e.g.

```
if (digitalRead(2) == HIGH  && digitalRead(3) == HIGH) { // read two switches
  // ...
}
```

> greater than

< less than

; Used to end a statement.

{} An opening curly brace "{" must always be followed by a closing curly brace "}". Functions are located inside of these brackets.

= Stores the value to the right of the equal sign in the variable to the left of the equal sign

== equal to

if

`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
}
```

The program tests to see if someVariable is greater than 50. If it is, the program takes a particular action. Put another way, if the statement in parentheses is true, the statements inside the brackets are run. If not, the program skips over the code.

The brackets may be omitted after an *if* statement. If this is done, the next line (defined by the semicolon) becomes the only conditional statement.

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.