Arduino Sensor Beginners Guide

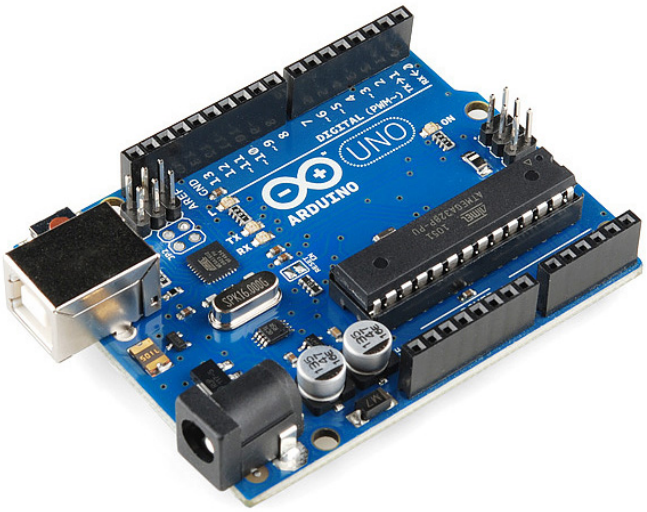
So you want to learn arduino. Good for you. Arduino is an easy to use, cheap, versatile and powerful tool that can be used to make some very effective sensors.

This guide is meant to give you the basics of getting started with Arduino, and provide you with some basic code that will work with many sensors.

The Materials

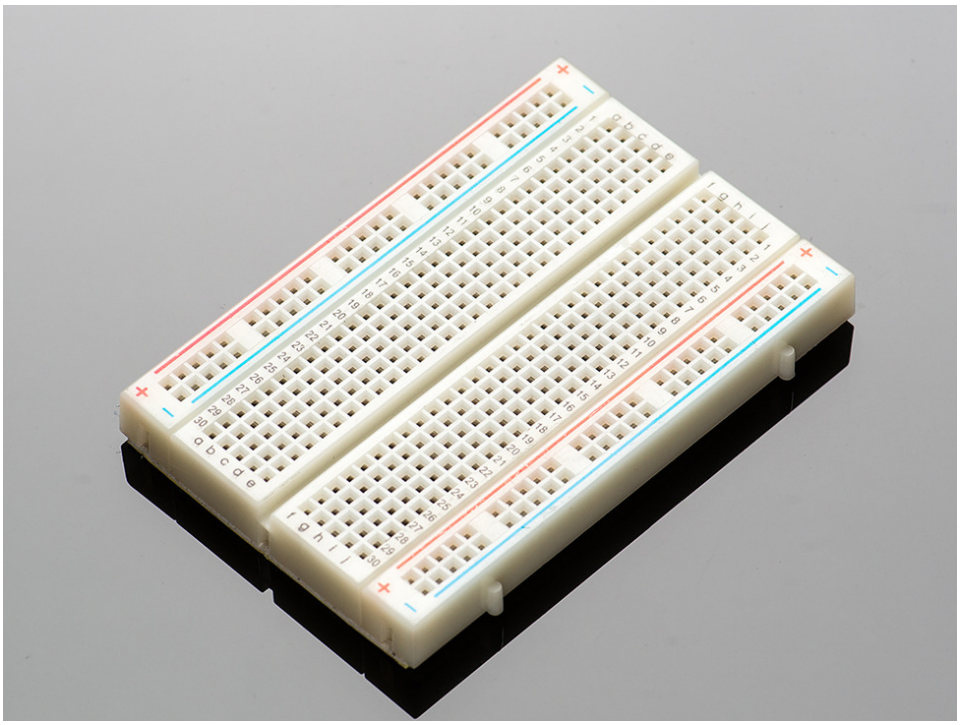
To begin, you will need

1. An ARDUINO:



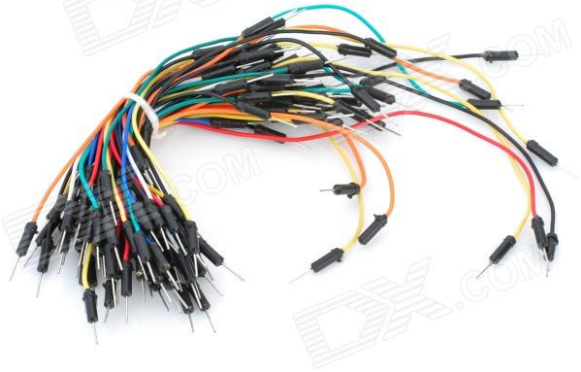
I personally use an ARDUINO UNO V2, but there are many different types available. For our purposes they should all work the same so if you already have one, great.

2. A breadboard: Breadboards are named after cutting boards that early circuit enthusiasts used to hold their sensors. A breadboard is used as a place to hold your sensors, resistors and wires, and also serves as an easy way to connect things together. Breadboards come in all shapes and sizes. I would recommend something like this.



This will cost you about 5 dollars online.

3. Wire:



Wires are essential to Arduino. They are what allow you to actually connect your Arduino to stuff. Most small, hobby wire will work. All you need to make sure is that it is small enough to easily fit into the pins on your Arduino and breadboard.

4. A Sensor:

Sensors come in all shapes and sizes. Most of the guides I have written use sensors similar to these, but don't feel restricted to these ones. Many of them work pretty much the same way.

Using Your Breadboard

Different breadboards are set up in different ways. All you need to worry about with this one are the middle two big pin columns (column a-e and column f-j).

Each big column has a bunch of rows (the exact number varies between boards). All of the pins within the same big column and row are connected together. This means that if we were to plug a 5v power source into pin a1, pins b1, c1, d1 and e1 will also receive power. No other pins on the board will be powered though.

If we wanted to make a basic circuit to say, power an LED we first will plug our power source (in this case the Arduino 5v port) into one of the rows.

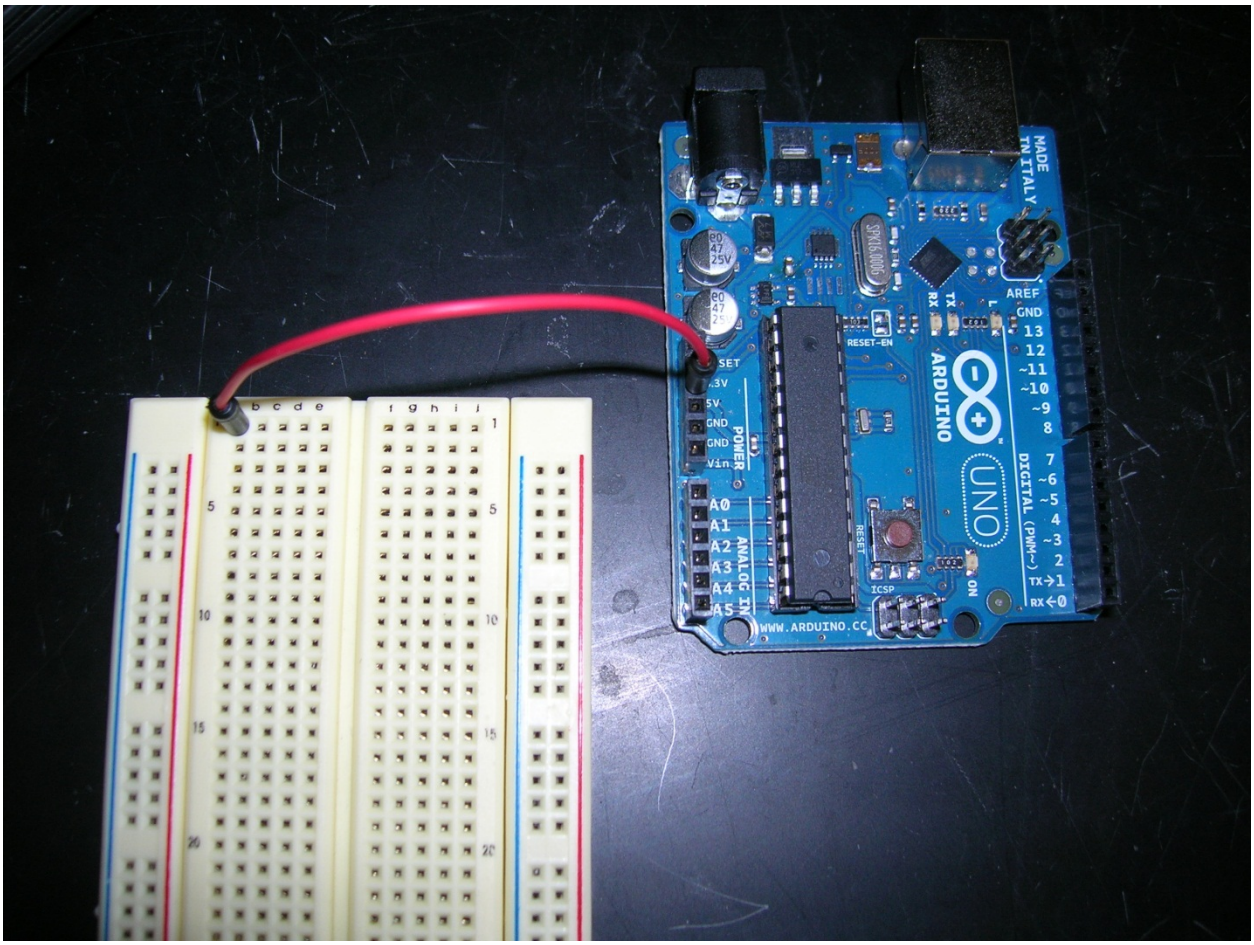
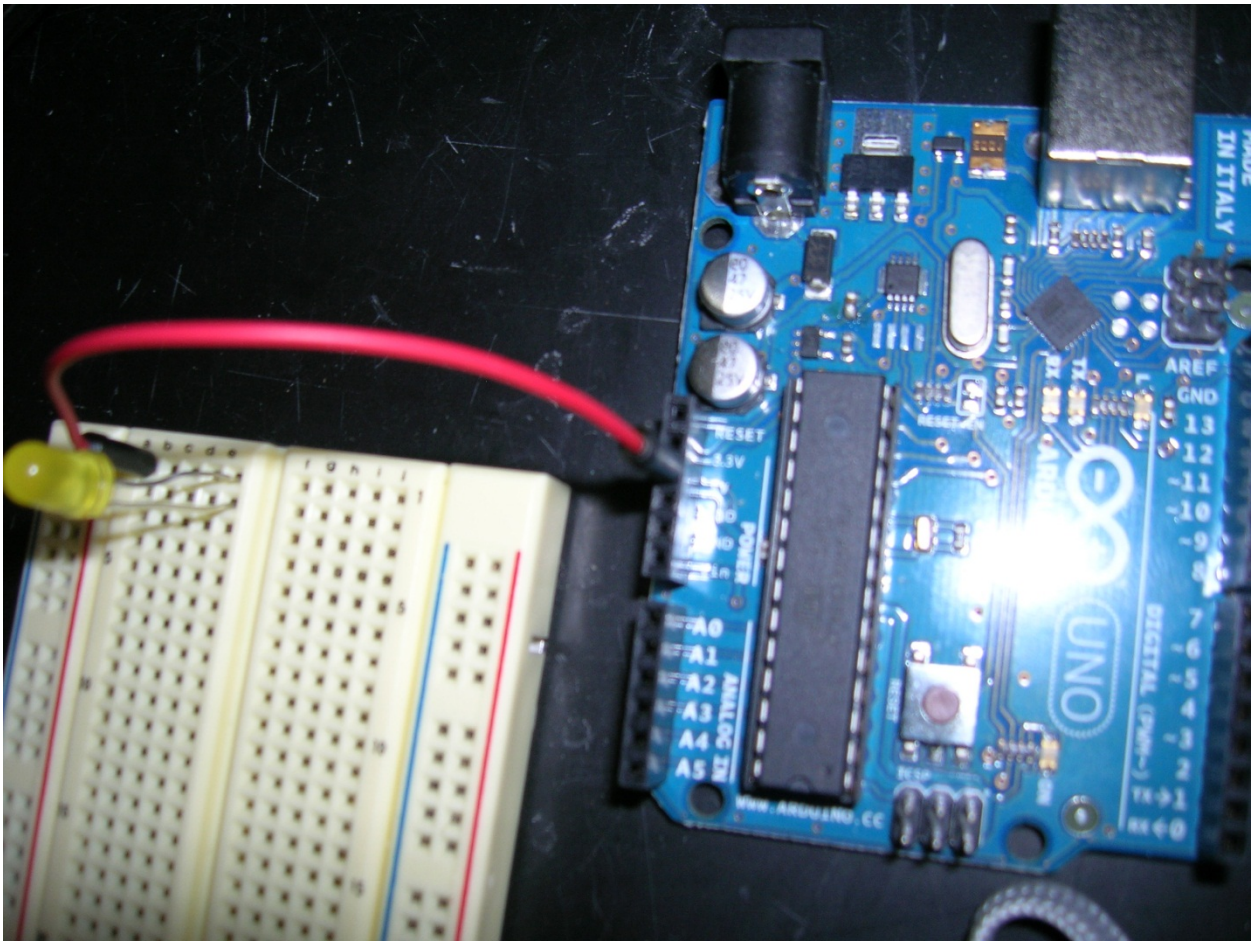


Figure 1: Pin 1A, 1B, 1C, 1D and 1E now have power. Nothing else does.

Now the row is powered. Next we'll plug our LED into the row.



Notice we don't have both prongs of the LED going into the powered row. The energy must flow through the led in order to power it. If both pins were in the powered row, the electricity would have nowhere to go and would not flow through the LED.

To give the electricity somewhere to go we connect half of it into an unpowered lane and connect that to the ground (GND) port on the Arduino.

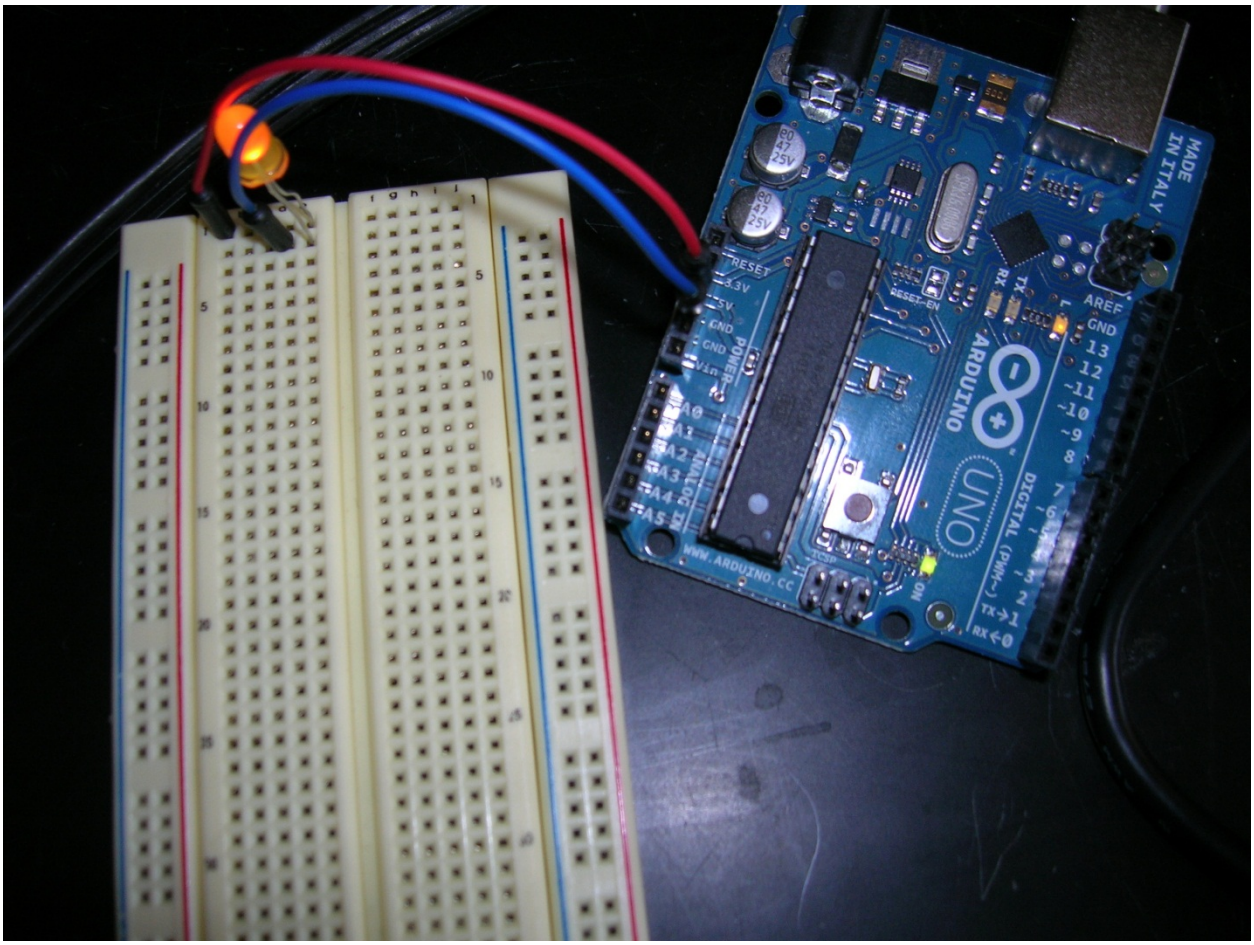


Figure 2: The blue wire is connected to GND on the arduino and 2D on the breadboard.

Now the electricity has a path to run along; from the Arduino, through the LED and into the ground. The circuit is complete and the LED lights up.

That's it! Breadboards are pretty simple and easy to use but using them properly is absolutely essential to getting your sensors working. If you use the breadboard wrong your sensor may not receive power, or you may short circuit it and damage your sensor.

Wiring and Programming a Sensor

First things first, you need to find out if your sensor is an analog sensor or a digital sensor. If your sensor measures a range of values it is probably an analog sensor. Examples of common analog sensors include light, temperature, humidity, sound etc. If the sensor only measures 2 states like presences/absence, on off, etc. it's probably a digital sensor. Sensors like these are

often things like reed switches, buttons, basic tilt sensors, sensors that respond to loud sounds, and so on. There are exceptions, but they are less common and will not be covered in this guide.

If your sensor is analog, keep reading. If it is digital, skip to the digital section below.

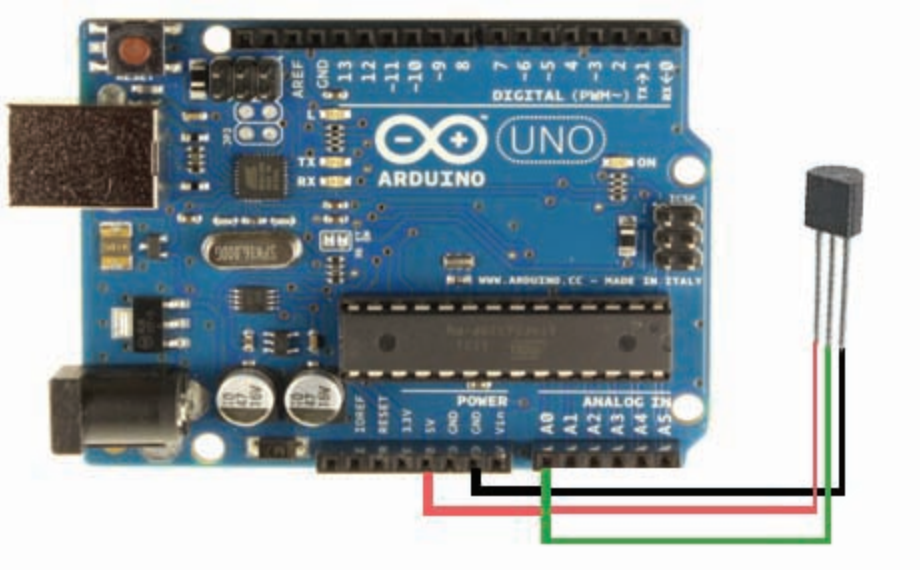
ANALOG SENSORS

Analog sensors vary widely in function but most operate under the same basic principle. They are generally resistors that will change their resistance in response to a stimulus. When the resistance changes, they consume a different amount of voltage. If we know how the voltage relates to the stimulus, we can easily use the voltage output to determine the value of whatever it is we are trying to measure.

Step 1: What does your sensor look like? It probably has 3 pins. If not, I'm sorry but this guide probably won't work for you. Try our linear hall magnetic module guide instead.

One pin is the ground, one pin is the 5v input, and one pin is the reading output. Power runs from the 5v to the ground, and data runs from the sensor through the output pin and into the Arduino. If you are using sensors like mine, they will be connected to a small circuit board and 2 of the pins are labeled. One is labeled S and the other is labeled -. S stands for sensor and is our data output. – means ground and is where we plug our ground into. The remaining pin (usually in the center) is the 5v input.

If your sensor does not have labeled pins, try something like this.

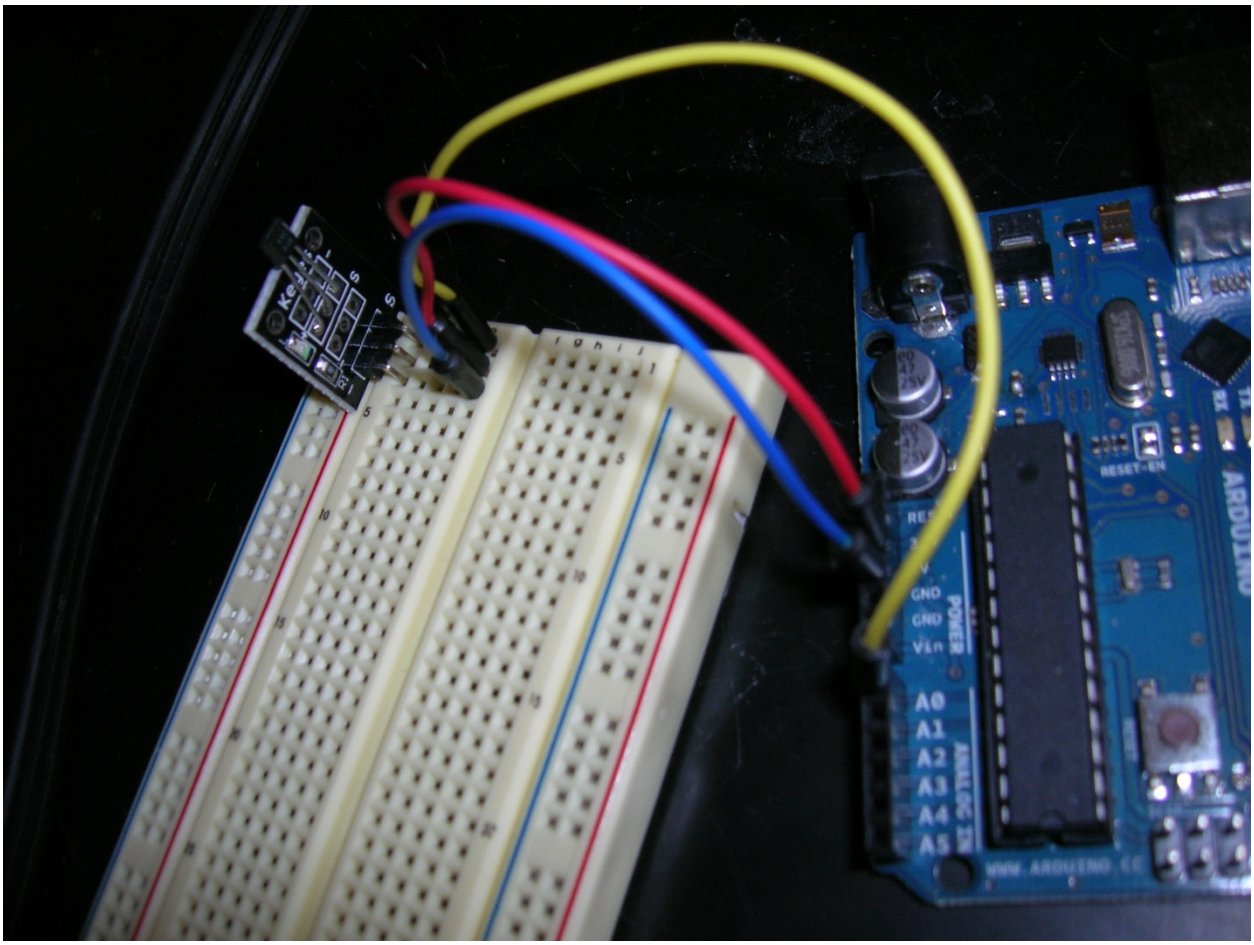


This will probably work but keep in mind the setup of sensors can be different. You will want to be careful when you first supply power to the circuit. If the sensor is plugged in wrong may heat up and break the sensor. I would suggest google imaging your sensor and seeing how other people set it up prior to plugging it in.

A good habit to get into is to double check your wiring before plugging the Arduino into the computer. A misplaced wire can easily damage or completely fry a sensor (believe me, I've fried more than a couple).

Step 2: Plug in your sensor. Plug the three pins of your sensor into three different rows on your breadboard. Connect a wire from the 5v on your Arduino to row containing the 5v pin on your sensor. Connect a wire from A0 on your breadboard to breadboard row containing the pin marked with an S. Connect a wire from GND on your breadboard to the row containing a pin with a – on it.

Your setup should look something like this.



Step 3: Coding

Coding can be a bit overwhelming at first. You are essentially trying to learn a new language, and may be frustrated that you can't get your code to work properly or do what you want. If you are just learning Arduino, I would recommend looking for code online instead of trying to teach it all to yourself. There is no shortage of helpful people online who post guides and code to projects they've done. As you gain experience you will become more familiar with the language and what the codes are actually doing. Until that point though, don't discourage yourself by trying to write something.

The following code is a very basic code that will work for most analog sensors. It's highly commented as well so it should be able to explain what it is doing pretty well.

int sensorPin = 0; //This piece tells the arduino which sensorpin we are working with, in this case it is analog pin zero.

//This piece of code also gives the pin a name, in this case it's name is sensorPin. Now, whenever we refer to sensorPin, the arduino will know that means analog port 0.

void setup () //This pretty much just says the code is starting. Don't worry about it too much.

{

Serial.begin(9600); //this opens communications between the computer and the arduino. The number indicates the speed of the communication.

//If for some reason your serial monitor just spits out garbage data (random characters) try changing this number to a different value. For instance, my computer does not work avobe 9600. There are several presets you can look up online.

}

void loop()//This sets up a loop of code that the arduino will run over and over again.

{

int reading = analogRead(sensorPin); //here we are telling the arduino that anytime we say "reading" we mean whatever value is being given to sensorPin by the sensor.

float voltage = reading * 5.0; //this tells the arduino to remember that voltage is our reading * 5

voltage /=1024.0; //the arduino is converting it's number for voltage to actual voltage by deviding it by 1024.

Serial.println(voltage); Serial.println ("volts");//this tells the arduino to write the value it receves from sensorPin to the serial monitor so we can see it.It also tells the arduino to write volts afterwards.

delay(1000); //tells the arduino to wait for a second so we don't get too much data too fast.

}

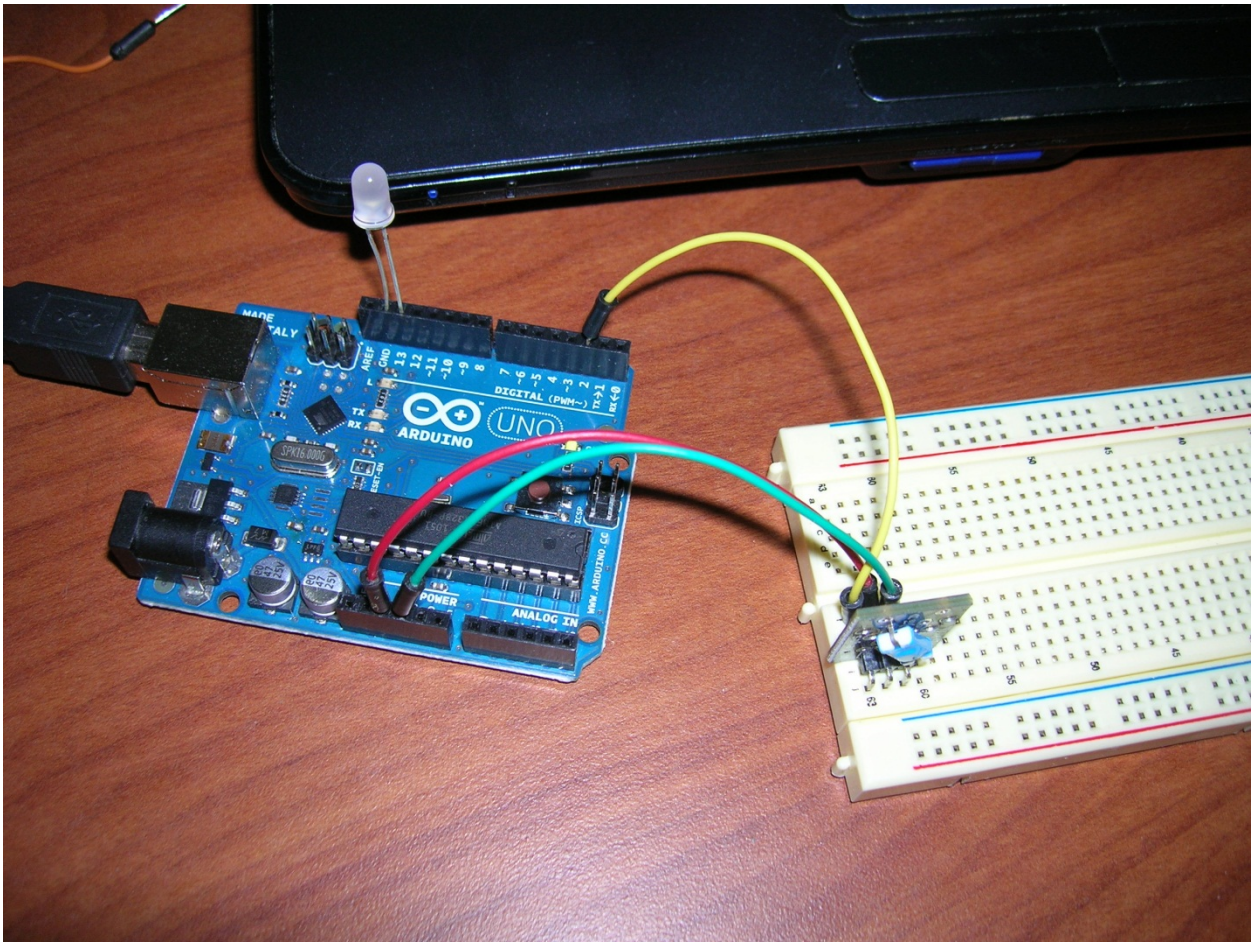
DIGITAL SENSORS

Digital sensors tend to be simpler in function than analog sensors. Analog sensors provide variable voltage that changes based on whatever they are measuring. Digital sensors just act as switches. They are either on or off and will only change state when a threshold signal is reached. There are exceptions to this, but these sensors tend to be more complicated and will not be covered here. If you do have one of these sensors, try our digital temperature sensor guide, or the Linear Hall magnetic module guide.

Step 1: What does your sensor look like? It probably has 3 pins. If not, I'm sorry but this guide probably won't work for you. Try our linear hall magnetic module guide instead.

One pin is the ground, one pin is the 5v input, and one pin is the reading output. Power runs from the 5v to the ground, and data runs from the sensor through the output pin and into the Arduino. If you are using sensors like mine, they will be connected to a small circuit board and 2 of the pins are labeled S and -. S stands for sensor and is our data output. – means ground and is where we plug our ground into. The remaining pin (usually in the center) is the 5v input.

Your setup should look something like this.



The – symbol on the sensor is connected to GND on the Arduino. The middle pin is connected to 5V on the Arduino. The S pin on the sensor is connected to digital port 2 on the Arduino digital ports 0 and 1 have different functions and are usually skipped over when making sensors. The LED is designed to turn on or off depending on what the digital sensor is sensing. The LED is not necessary as the Arduino has a built in LED that does the same thing, but I like having a brighter light so I threw it in. The long leg of the LED connects to digital pin 13, and the short leg connects to GND.

A good habit to get into is to double check your wiring before plugging the Arduino into the computer. A misplaced wire can easily damage or completely fry a sensor (believe me, I've fried more than a couple).

If your sensor looks nothing like mine, I'm sorry but I can't give you advice on the setup. Digital sensors come in many shapes and sizes and with different configurations so I can't cover them all here. My advice would be to google image your sensor and see if you can find how other people have set up your sensor.

Step 3: Coding

Coding can be a bit overwhelming at first. You are essentially trying to learn a new language, and may be frustrated that you can't get your code to work properly or do what you want. If you are just learning Arduino, I would recommend looking for code online instead of trying to teach it all to yourself. There is no shortage of helpful people online who post guides and code to projects they've done. As you gain experience you will become more familiar with the language and what the codes are actually doing. Until that point though, don't discourage yourself by trying to write something.

The following code is a very basic code that will work for most digital sensors. It's highly commented as well so it should be able to explain what it is doing pretty well.

```
const int sensorPin = 2;     // Here we are telling the arduino that anytime we refer to sensorPin, we mean digital pin 2.

const int ledPin =  13;      // here, we are telling the arduino that if we say ledPin, we are refering to the LED in pin 13. The intergrated LED is also connected to this pin.


int sensorState = 0;         // this is a variable to describe our sensor. We are telling the arduino that whatever state the sensor is in when the arduino turns on will be defined as state 0.

void setup() { // just says the code is starting

  pinMode(ledPin, OUTPUT); //tells the arduino that ledPin is where the information will be displayed.

  pinMode(sensorPin, INPUT); //tells the arduino that sensorPin is where the information is comming from.

}


void loop(){//the following code is contained in a loop, so the arduino will do it forever.

  // read the state of the pushbutton value:

  sensorState = digitalRead(sensorPin); //we are creating a variable here and defining it as whatever state sensorPin is in.
```

```
 if (sensorState == LOW) { //if sensorPin reads a value above zero (defined above)...
   digitalWrite(ledPin, HIGH);  //turn the led on
 }
 else {//otherwise
   digitalWrite(ledPin, LOW); //turn the LED off.
 }
}
```

Conclusions

That should just about do it. Hopefully now you will have a working sensor. At the very least I hope you have a somewhat better idea of how Arduino works. It is a little bit complicated at first, and it can be frustrating, but keep with it. It makes it all the more satisfying when you do finally figure it out. If you have questions that aren't answered in this guide (and let's be honest, you probably do) try looking around on google. There is no shortage of Arduino enthusiasts who are more than willing to help out.

If you have any more questions, feel free to e-mail me at bensegee@hotmail.com

I can't guarantee that I will be able to help, but I will do what I can.

Happy coding!