# 03

LED

220 OHM RESISTOR


TEMPERATURE SENSOR

INGREDIENTS

# LOVE-O-METER

TURN THE ARDUINO INTO A LOVE MACHINE. USING AN ANALOG INPUT, YOU'RE GOING TO REGISTER JUST HOW HOT YOU REALLY ARE!

*Discover: analog Input, using the serial monitor*

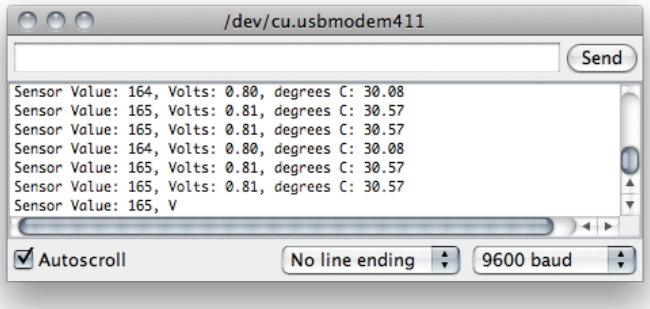
Time: **45 MINUTES**
Level: ■ □ □ □ □

Builds on projects: **1, 2**

*While switches and buttons are great, there's a lot more to the physical world than on and off. Even though the Arduino is a digital tool, it's possible for it to get information from analog sensors to measure things like temperature or light. To do this, you'll take advantage of the Arduino's built-in Analog-to-Digital Converter (ADC). Analog in pins A0-A5 can report back a value between 0-1023, which maps to a range from 0 volts to 5 volts.*


You'll be using a ***temperature sensor*** to measure how warm your skin is. This component outputs a changing voltage depending on the temperature it senses. It has three pins: one that connects to ground, another that connects to power, and a third that outputs a variable voltage to your Arduino. In the sketch for this project, you'll read the sensor's output and use it to turn LEDs on and off, indicating how warm you are. There are several different models of temperature sensor. This model, the TMP36, is convenient because it outputs a voltage that changes directly proportional to the temperature in degrees Celsius.

The Arduino IDE comes with a tool called the ***serial monitor*** that enables you to report back results from the microcontroller. Using the serial monitor, you can get information about the status of sensors, and get an idea about what is happening in your circuit and code as it runs.




Serial monitor

**Fig. 1**

## BUILD THE CIRCUIT

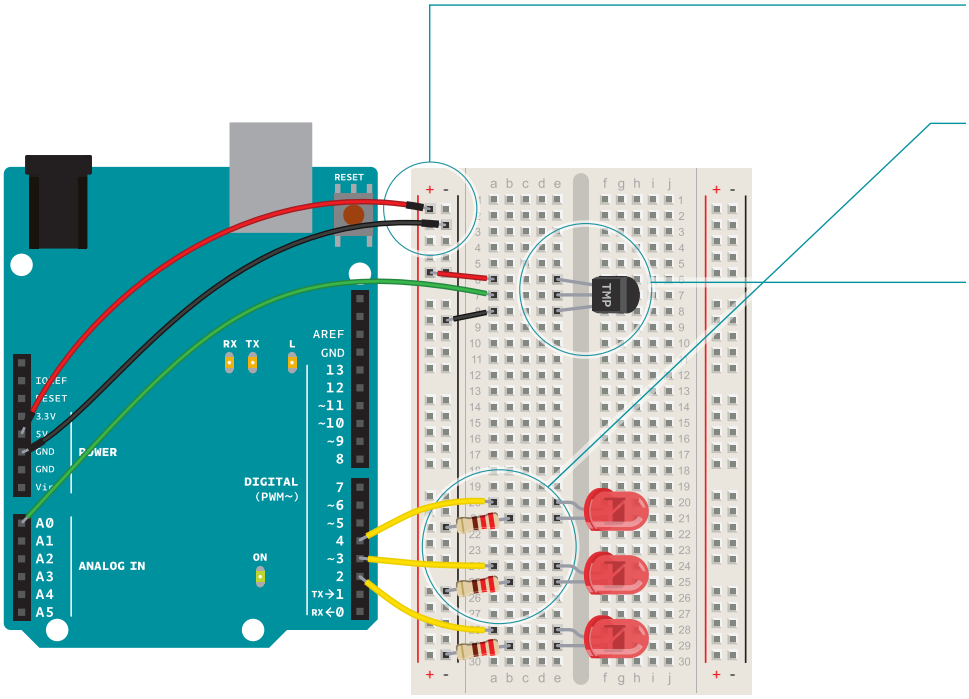



Fig. 2

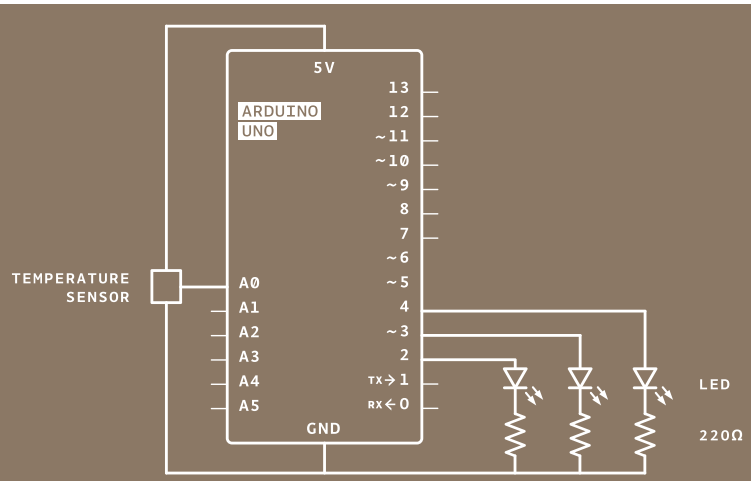



Fig. 3

In this project, you need to check the ambient temperature of the room before proceeding. You're checking things manually right now, but this can also be accomplished through calibration. It's possible to use a button to set the baseline temperature, or to have the Arduino take a sample before starting the `loop()` and use that as the reference point. Project 6 gets into details about this, or you can look at the Calibration example that comes bundled with the Arduino software:

***arduino.cc/calibration***

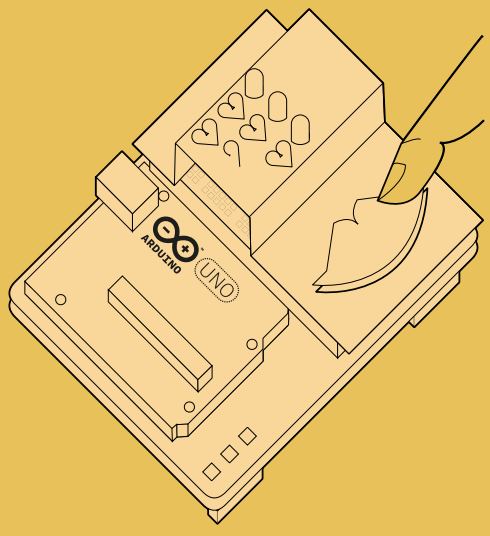
1. Just as you've been doing in the earlier projects, wire up your breadboard so you have power and ground.

2. Attach the cathode (short leg) of each of the LEDs you're using to ground through a 220-ohm resistor. Connect the anodes of the LEDs to pins 2 through 4. These will be the indicators for the project.

3. Place the TMP36 on the breadboard with the rounded part facing away from the Arduino (the order of the pins is important!) as shown in Fig. 2. Connect the left pin of the flat facing side to power, and the right pin to ground. Connect the center pin to pin A0 on your Arduino. This is analog input pin 0.

Create an interface for your sensor for people interact with. A paper cutout in the shape of a hand is a good indicator. If you're feeling lucky, create a set of lips for someone to kiss, see how well that lights things up! You might also want to label the LEDs to give them some meaning. Maybe one LED means you're a cold fish, two LEDs means you're warm and friendly, and three LEDs means you're too hot to handle!

1. Cut out a piece of paper that will fit over the breadboard. Draw a set of lips where the sensor will be, and cut some circles for the LEDs to pass through.




2. Place the cutout over the breadboard so that the lips cover the sensor and the LEDs fit into the holes. Press the lips to see how hot you are!

## THE CODE

A pair of useful constants

*Constants* are similar to variables in that they allow you to uniquely name things in the program, but unlike variables they cannot change. Name the analog input for easy reference, and create another named constant to hold the baseline temperature. For every 2 degrees above this baseline, an LED will turn on.
You've already seen the int datatype, used here to identify which pin the sensor is on. The temperature is being stored as a *float*, or floating-point number. This type of number has a decimal point, and is used for numbers that can be expressed as fractions.

Initialize the serial port to the desired speed

In the setup you're going to use a new command, `Serial.begin()`. This opens up a connection between the Arduino and the computer, so you can see the values from the analog input on your computer screen.
The argument `9600` is the speed at which the Arduino will communicate, 9600 bits per second. You will use the Arduino IDE's serial monitor to view the information you choose to send from your microcontroller. When you open the IDE's serial monitor verify that the baud rate is 9600.

Initialize the digital pin directions and turn off

Next up is a `for()` loop to set some pins as outputs. These are the pins that you attached LEDs to earlier. Instead of giving them unique names and typing out the `pinMode()` function for each one, you can use a `for()` loop to go through them all quickly. This is a handy trick if you have a large number of similar things you wish to iterate through in a program. Tell the `for()` loop to run through pins 2 to 4 sequentially.

Read the temperature sensor

In the `loop()`, you'll use a local variable named `sensorVal` to store the reading from your sensor. To get the value from the sensor, you call `analogRead()` that takes one argument: what pin it should take a voltage reading on. The value, which is between 0 and 1023, is a representation of the voltage on the pin.

Send the temperature sensor values to the computer

The function `Serial.print()` sends information from the Arduino to a connected computer. You can see this information in your serial monitor. If you give `Serial.print()` an argument in quotation marks, it will print out the text you typed. If you give it a variable as an argument, it will print out the value of that variable.

```
1 const int sensorPin = A0;
2 const float baselineTemp = 20.0;
```

```
3 void setup(){
4   Serial.begin(9600); // open a serial port
```

```
5   for(int pinNumber = 2; pinNumber<5; pinNumber++){
6     pinMode(pinNumber,OUTPUT);
7     digitalWrite(pinNumber, LOW);
8   }
9 }
```

**for() loop tutorial**
*arduino.cc/for*

```
10 void loop(){
11   int sensorVal = analogRead(sensorPin);
```

```
12   Serial.print("Sensor Value: ");
13   Serial.print(sensorVal);
```

### Convert sensor reading to voltage

With a little math, it's possible to figure out what the real voltage on the pin is. The voltage will be a value between 0 and 5 volts, and it will have a fractional part (for example, it might be 2.5 volts), so you'll need to store it inside a `float`. Create a variable named voltage to hold this number. Divide `sensorVal` by 1024.0 and multiply by 5.0. The new number represents the voltage on the pin.

Just like with the sensor value, you'll print this out to the serial monitor.

### Convert the voltage to temperature and send the value to the computer

If you examine the sensor's *datasheet*, there is information about the range of the output voltage. Datasheets are like manuals for electronic components. They are written by engineers, for other engineers. The datasheet for this sensor explains that every 10 millivolts of change from the sensor is equivalent to a temperature change of 1 degree Celsius. It also indicates that the sensor can read temperatures below 0 degrees. Because of this, you'll need to create an offset for values below freezing (0 degrees). If you take the voltage, subtract 0.5, and multiply by 100, you get the accurate temperature in degrees Celsius. Store this new number in a floating point variable called temperature.

Now that you have the real temperature, print that out to the serial monitor too. Since the temperature variable is the last thing you're going to be printing out in this loop, you're going to use a slightly different command: `Serial.println()`. This command will create a new line in the serial monitor after it sends the value. This helps make things easier to read in when they are being printed out.

### Turn off LEDs for a low temperature

With the real temperature, you can set up an `if()...else` statement to light the LEDs. Using the baseline temperature as a starting point, you'll turn on one LED on for every 2 degrees of temperature increase above that baseline. You're going to be looking for a range of values as you move through the temperature scale.

```
14   // convert the ADC reading to voltage
15   float voltage = (sensorVal/1024.0) * 5.0;
```

```
16   Serial.print(", Volts: ");
17   Serial.print(voltage);
```

```
18   Serial.print(", degrees C: ");
19   // convert the voltage to temperature in degrees
20   float temperature = (voltage - .5) * 100;
21   Serial.println(temperature);
```

```
22   if(temperature < baselineTemp){
23     digitalWrite(2, LOW);
24     digitalWrite(3, LOW);
25     digitalWrite(4, LOW);
```

**Starter Kit datasheets**
*arduino.cc/kitdatasheets*

Turn on one LED for a low temperature

The && operator means "`and`", in a logical sense. You can check for multiple conditions: "if the temperature is 2 degrees greater than the baseline, and it is less than 4 degrees above the baseline."

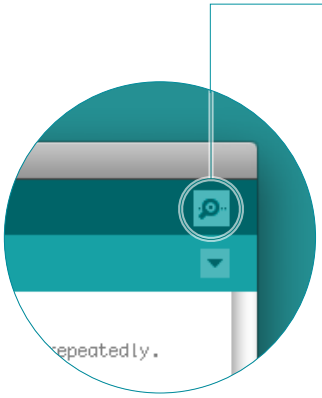
Turn on two LEDs for a medium temperature

If the temperature is between two and four degrees above the baseline, this block of code turns on the LED on pin 3 as well.

Turn on three LEDs for a high temperature

The Analog-to-Digital Converter can only read so fast, so you should put a small delay at the very end of your `loop()`. If you read from it too frequently, your values will appear erratic.

## USE IT




With the code uploaded to the Arduino, click the serial monitor icon. You should see a stream of values coming out, formatted like this : `Sensor: 200, Volts: .70, degrees C: 17`

Try putting your fingers around the sensor while it is plugged into the breadboard and see what happens to the values in the serial monitor. Make a note of what the temperature is when the sensor is left in the open air.

Close the serial monitor and change the baselineTemp constant in your program to the value you observed the temperature to be. Upload your code again, and try holding the sensor in your fingers. As the temperature rises, you should see the LEDs turn on one by one. Congratulations, hot stuff!

```
  }else if(temperature >= baselineTemp+2 &&
    temperature < baselineTemp+4){
    digitalWrite(2, HIGH);
    digitalWrite(3, LOW);
    digitalWrite(4, LOW);

  }else if(temperature >= baselineTemp+4 &&
    temperature < baselineTemp+6){
    digitalWrite(2, HIGH);
    digitalWrite(3, HIGH);
    digitalWrite(4, LOW);

  }else if(temperature >= baselineTemp+6){
    digitalWrite(2, HIGH);
    digitalWrite(3, HIGH);
    digitalWrite(4, HIGH);

  }
  delay(1);
}
```

Create an interface for two people to test their compatibility with each other. You get to decide what compatibility means, and how you'll sense it. Perhaps they have to hold hands and generate heat? Maybe they have to hug? What do you think?

*Expanding the types of inputs you can read, you've used analogRead() and the serial monitor to track changes inside your Arduino. Now it's possible to read a large number of analog sensors and inputs.*