

*Heaven's Light Is Our Guide*

Rajshahi University of Engineering & Technology  
Department of Computer Science & Engineering

CSE 3105  
Computer Interfacing & Embedded System

Interfacing with Analog Sensors

Md. Nasif Osman Khansur  
Lecturer  
Dept. of CSE, RUET

# References

1. Exploring Arduino 2e by Jeremy Blum [Chapter 3]

# PIN Functions

Functionality	Arduino Uno	STM32 Blue Pill
Digital Pins (Input/Output)	0 - 13 A0 - A5	PA0 - PA15 PB0 - PB15 PC13 - PC15
Analog Pins (Input only)	A0 - A5	PA0 - PA7 PB0 - PB1
PWM Pins	3, 5, 6, 9, 10, 11	PA0 - PA3 PA6 - PA10 PB0 - PB1 PB6 - PB9

# Introduction

Consider the projects you completed in the preceding chapter. You used a switch to control an LED. A switch is a digital input—it has only two possible states: on or off, high or low, 1 or 0, and so on.

However, the world around you is analog.

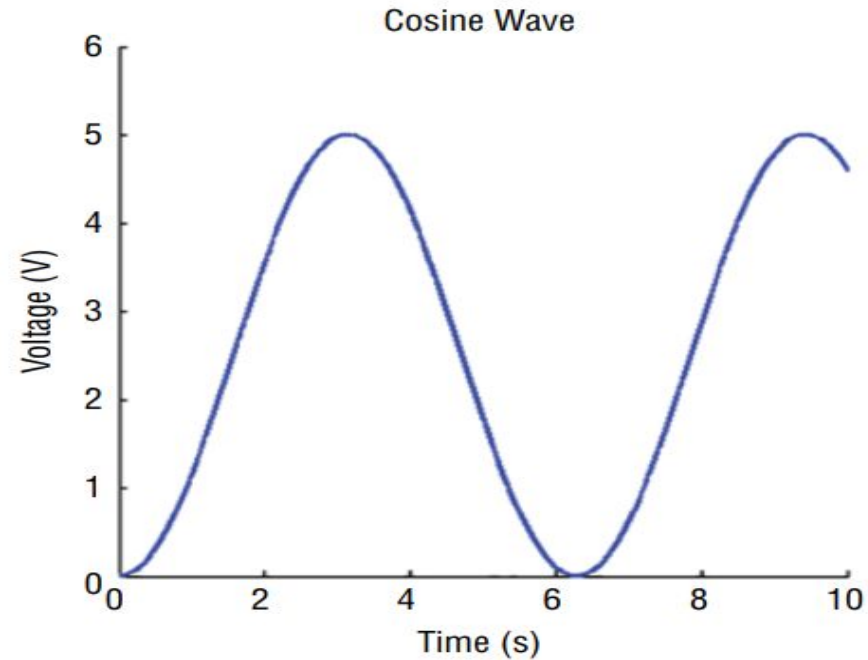
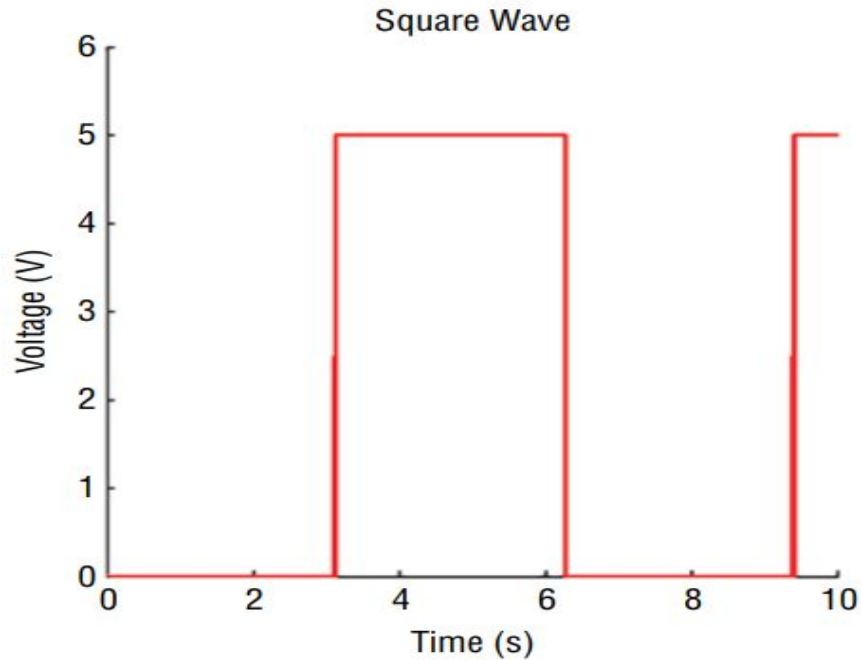
- Sunlight is not on or off; its brightness varies over the course of a day.
- Wind does not just have two states; it gusts at different speeds and directions all the time.

# Objectives

You will be able

- To learn differences between analog and digital signal
- To convert between the two signals
- To develop techniques on using analog sensors

# Analog and Digital Signals



An **analog** signal is a voltage, current, or physical quantity that continuously and infinitely varies in accordance with some time-varying parameter

A **digital** signal is a signal that represents data as a sequence of discrete values; at any given time it can only take on one of a finite number of values.

# Converting Analog Signal to Digital

Analog signals are signals that have a continuous sequence with continuous values. These types of signals can come from sound, light, temperature and motion.

Digital signals are represented by a sequence of discrete values where the signal is broken down into sequences that depend on the time series or sampling rate

The accuracy of an ADC is determined by its resolution. The resolution of the ADC is the number of bits it uses to digitize the input samples.

The higher the resolution, the more steps that are available for representing each value.

ADCs follow a sequence when converting analog signals to digital. They first sample the signal, then quantify it to determine the resolution of the signal, and finally set binary values and send it to the system to read the digital signal. [\[Link\]](#)

# Converting Analog Signal to Digital

Sampling frequency: samples per second (how many samples or data points it takes within a second)

One important equation on the sample rate is:

$$f_s = 1/T$$

Where,

$f_s$  = Sample Rate/Frequency

$T$  = Period of the sample or the time it takes before sampling again

❖ What happens when the sampling rate is considerably slower?

Aliasing means that when a digital image/signal is reconstructed, it differs greatly from the original image/signal caused from sampling.

If the sampling rate is slow and the frequency of the signal is high, the ADC will not be able to reconstruct the original analog signal which will cause the system to read incorrect data



# Converting Analog Signal to Digital

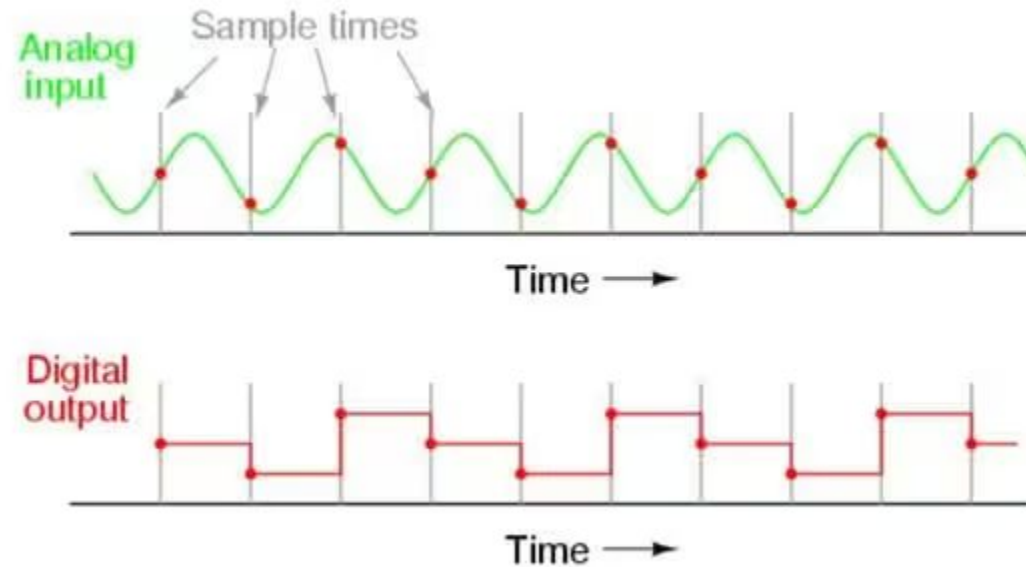


Figure : An example of how aliasing happens.

# Converting Analog Signal to Digital

One rule of thumb when figuring out if aliasing will happen is using **Nyquist Theorem**. According to the theorem, *the sampling rate/frequency needs to be at least twice as much as the highest frequency in the signal to recreate the original analog signal*. The following equation is used to find the Nyquist frequency:

$$f_{\text{Nyquist}} = 2f_{\text{Max}}$$

Where,

$f_{\text{Nyquist}}$  = Nyquist frequency

$f_{\text{Max}}$  = The max frequency that appears in the signal

For example, if the signal that you input into the digital system has a max frequency of 100 kHz, then the sampling rate on your ADC needs to be equal or greater than 200 kS/s. This will allow for a successful reconstruction of the original signal.

# Converting Analog Signal to Digital

• Sample ADC Resolution Formula:

$$\text{Step Size} = V_{\text{Ref}}/N$$

Where,

Step Size = The resolution of each level in terms of voltage  
 $V_{\text{Ref}}$  = The voltage reference (range of voltages)

$N$  = Total level size of ADC

To find  $N$  size, use this equation:

$$N = 2^n$$

Where,

$n$  = Bit Size

# Converting Analog Signal to Digital

Consider what a simpler, 3-bit ADC would do.

# Converting Analog Signal to Digital

Consider what a simpler, 3-bit ADC would do.

Because  $2^3 = 8$ , there are a total of eight logic levels, from 0 to 7.

Therefore, any analog value that is passed to a 3-bit ADC will have to be assigned a value from 0 to 7.

# Converting Analog Signal to Digital

Consider what a simpler, 3-bit ADC would do.

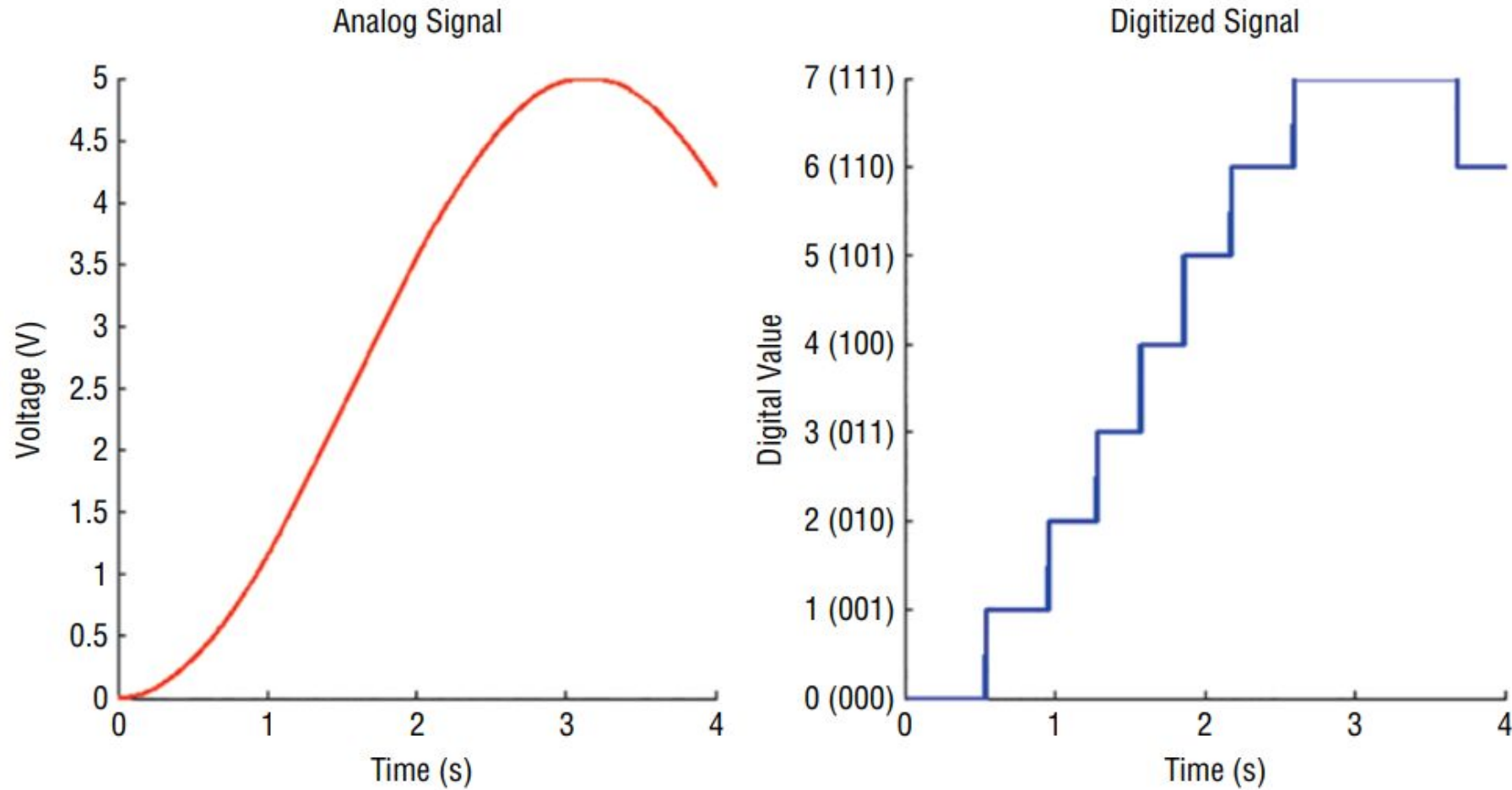
Because  $2^3 = 8$ , there are a total of eight logic levels, from 0 to 7.

Therefore, any analog value that is passed to a 3-bit ADC will have to be assigned a value from 0 to 7.

Arduino Uno has 10-bit ADC.

STM32 Blue Pill has 12-bit ADC.

# Converting Analog Signal to Digital



# Converting Analog Signal to Digital

$$\frac{ADCResolution}{AREF} = \frac{ADCReading}{AnalogInput}$$



# Converting Analog Signal to Digital

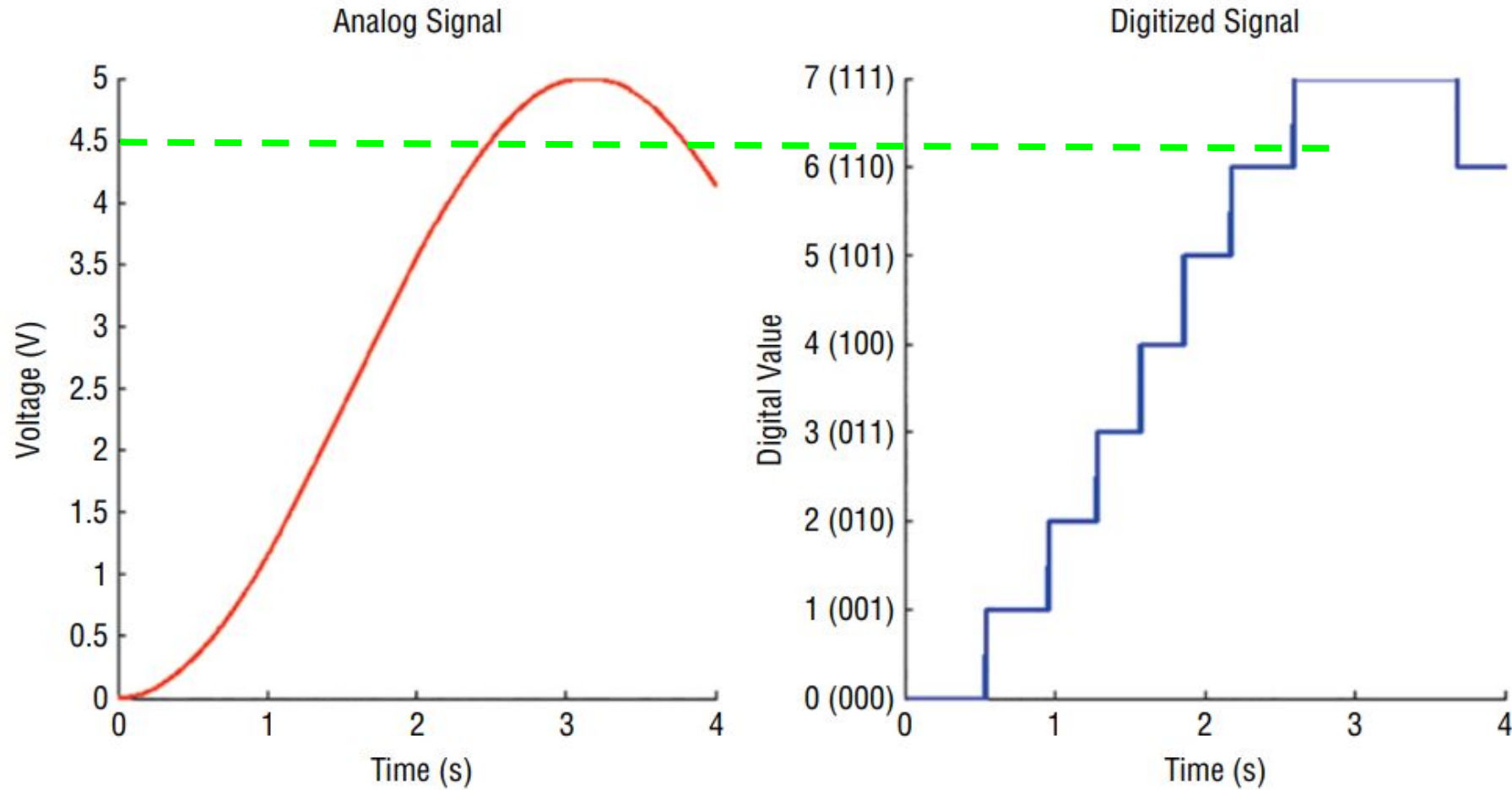
$$\frac{ADCResolution}{AREF} = \frac{ADCReading}{AnalogInput}$$

$$\frac{8}{5} = \frac{x}{4}$$

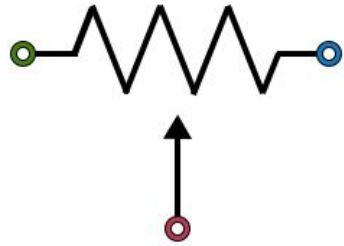
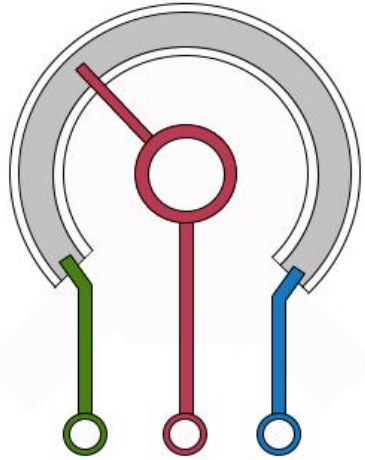
$$x \approx 6$$

$$x = 110$$

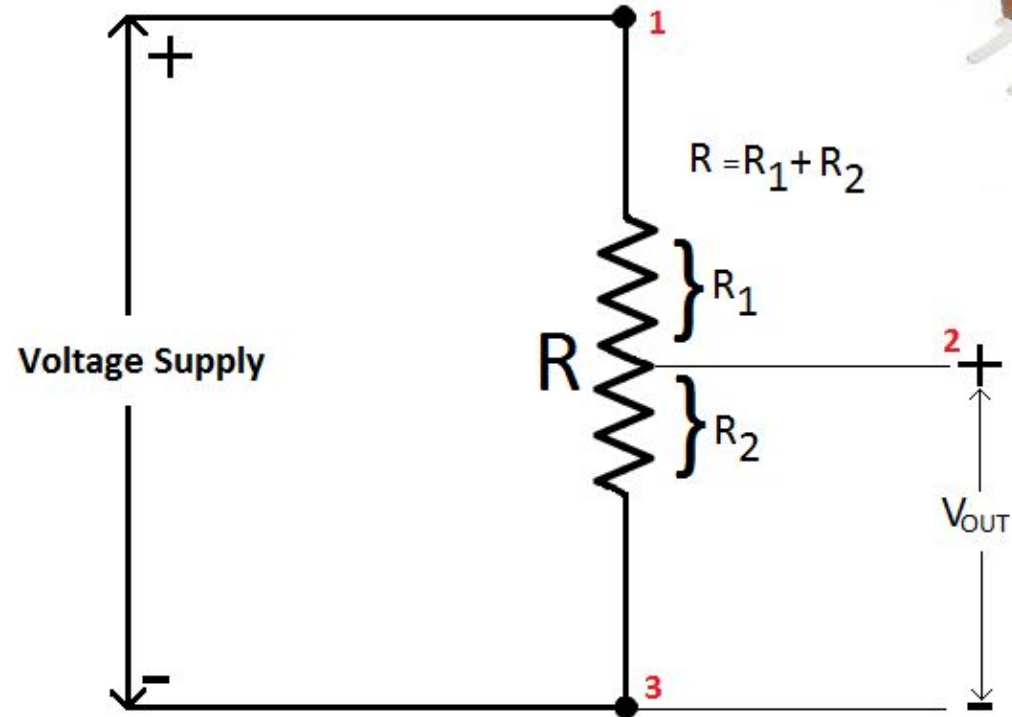
# Converting Analog Signal to Digital



# Reading a Potentiometer



A potentiometer (also pot or electronic pot) is a variable resistor in which a wiper sweeps from one end of the resistive element to the other, resulting in resistance that is proportional to the wiper's position.

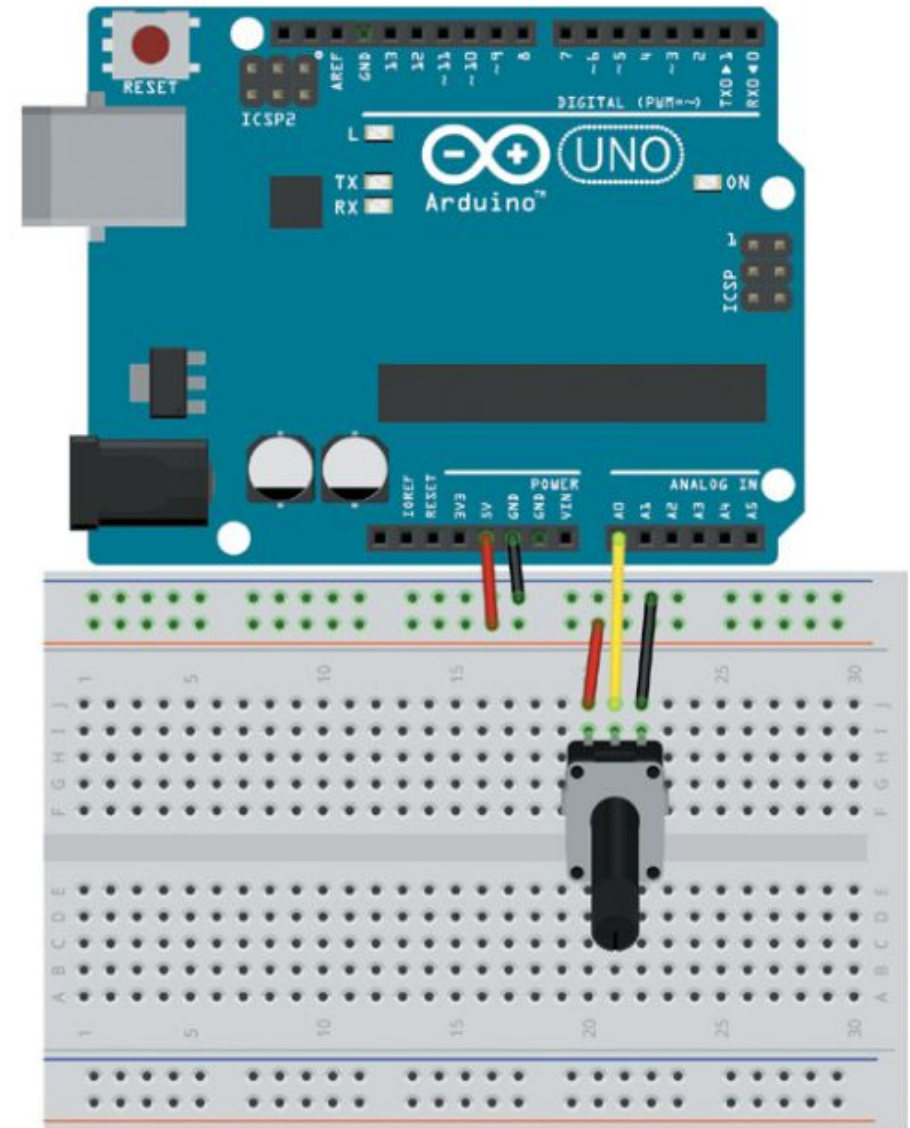


[\[Link\]](#)

[\[Link\]](#)

# Reading a Potentiometer

Potentiometers are symmetrical, so it doesn't matter which side you connect the 5V rail and ground to. You connect the middle pin to analog input 0 on your Arduino.

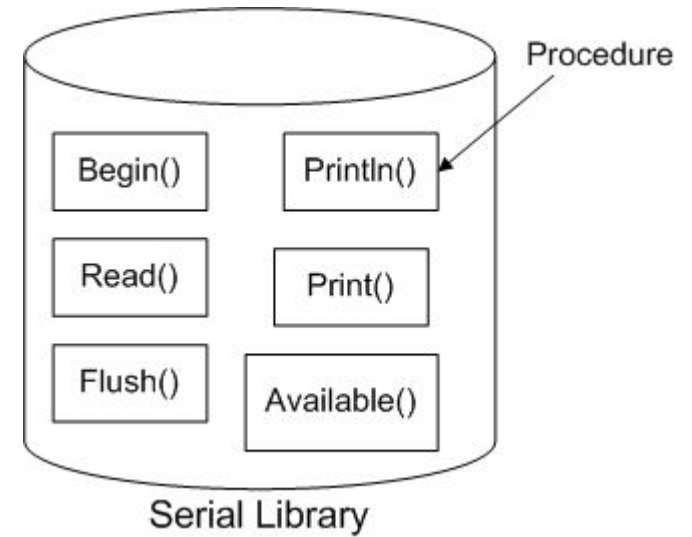
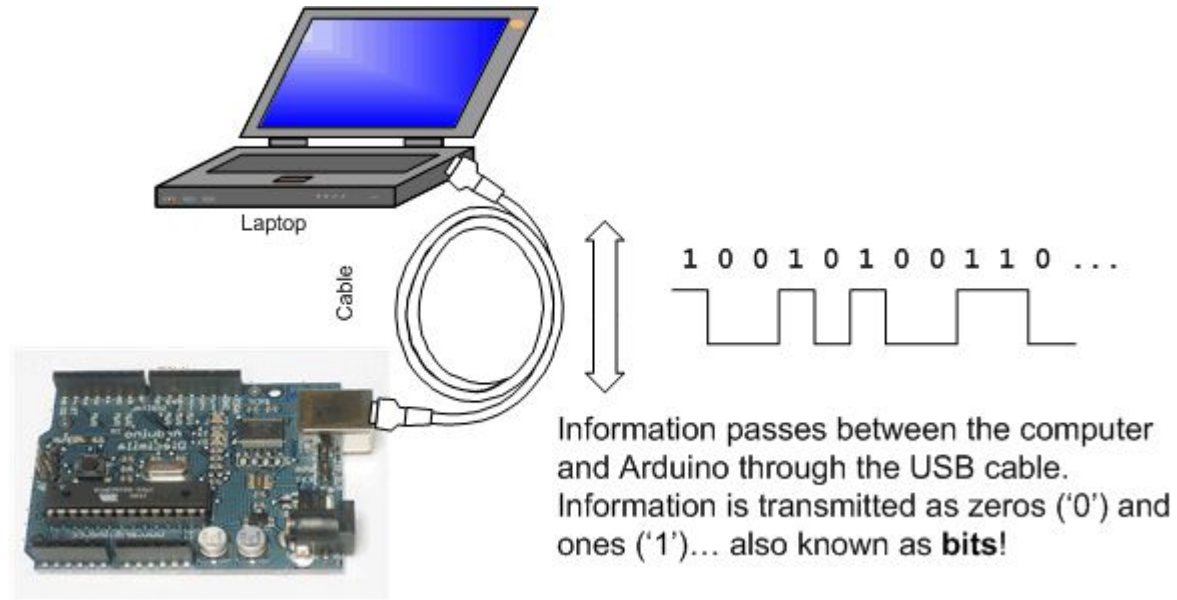


# Reading a Potentiometer

As you turn the potentiometer, you're varying the voltage that you are feeding into analog input between 0V and 5V. You can confirm this with a multimeter in voltage measurement mode by hooking it up as shown and reading the display as you turn the potentiometer's knob. The red (positive) probe should be connected to the middle pin, and the black (negative) probe should be connected to whichever side is connected to ground.



# Serial Library



[\[Link\]](#)

# Reading a Potentiometer

Before you use the potentiometer to control another piece of hardware, use the Arduino's serial communication functionality to print out the potentiometer's ADC value on your computer as it changes.

Use the `analogRead()` function to read the value of the analog pin connected to the Arduino, and the `Serial.println()` function to print it to the Arduino IDE serial monitor.

# Reading a Potentiometer

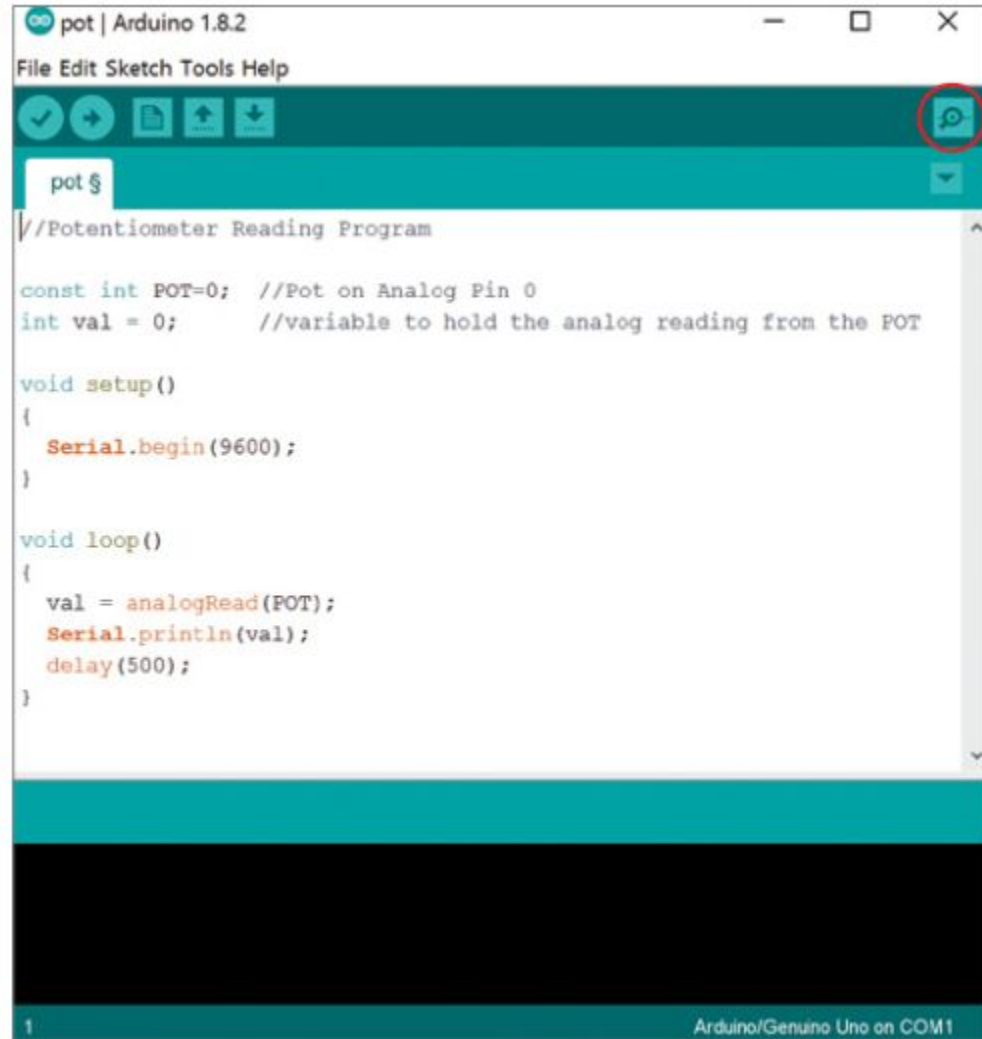
```
// Potentiometer Reading Program
const int POT=A0; // Pot on Analog Pin A0
int val = 0; // Variable to hold the analog reading from the POT

void setup() {
    pinMode (POT, INPUT);
    Serial.begin(9600);
}
void loop() {
    val = analogRead(POT);
    Serial.println(val);
    delay(500);
}
```

The baud rate specifies the number of bits being transferred per second. Faster baud rates enable you to transmit more data in less time, but can also introduce transmission errors in some communication systems. A common value is 9600 baud, which is what you will use throughout most of this book.



# Reading a Potentiometer



```
pot | Arduino 1.8.2
File Edit Sketch Tools Help

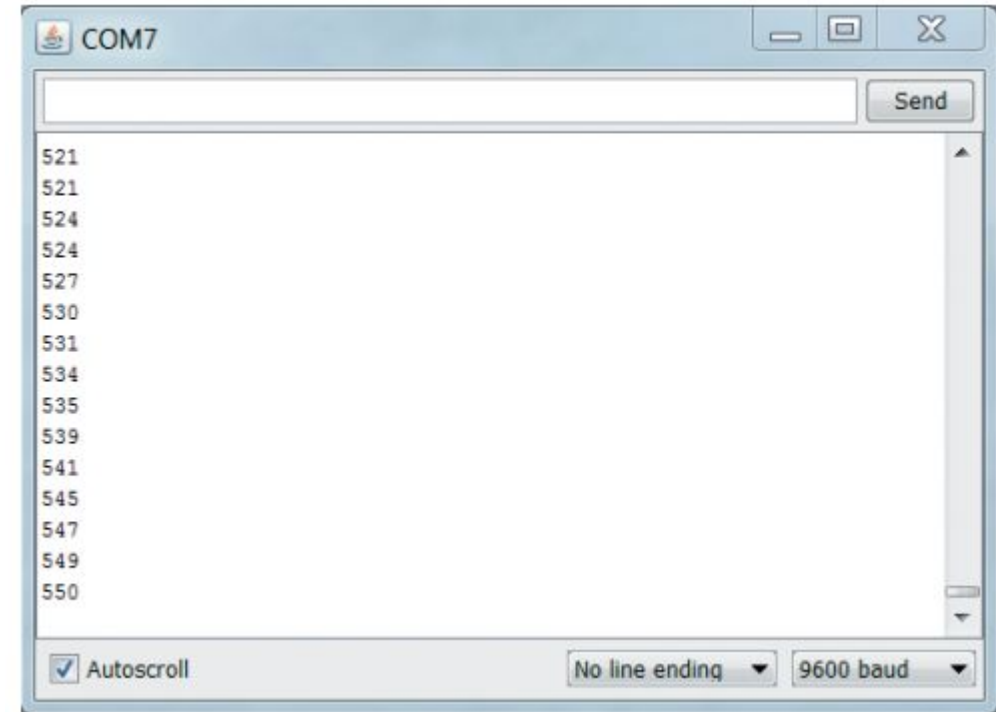
pot $
//Potentiometer Reading Program

const int POT=0; //Pot on Analog Pin 0
int val = 0;      //variable to hold the analog reading from the POT

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

void loop()
{
  val = analogRead(POT);
  Serial.println(val);
  delay(500);
}
```

1 Arduino/Genuino Uno on COM1



```
COM7
521
521
524
524
527
530
531
534
535
539
541
545
547
549
550

[Send]

[Autoscroll] [No line ending] [9600 baud]
```

Because you set it to 9600 in the code, you need to set it to 9600 in this window

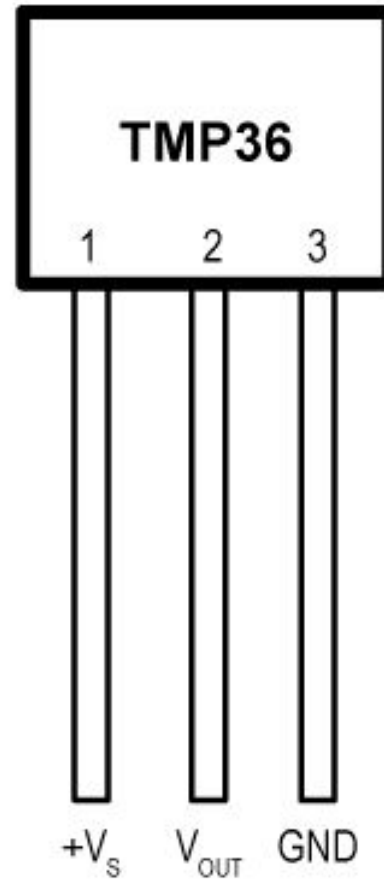
# Using Analog Sensors

Although potentiometers generate an analog voltage value on a pin, they aren't really sensors in the traditional meaning.

All kinds of sensors generate analog output values corresponding to “real-world” actions.

- **Accelerometers** that detect tilting.
- **Magnetometers** that detect magnetic fields.
- **Infrared sensors** that detect distance to an object
- **Temperature sensors** that can tell you about the operating environment of your project.

# TMP36 Temperature Sensor

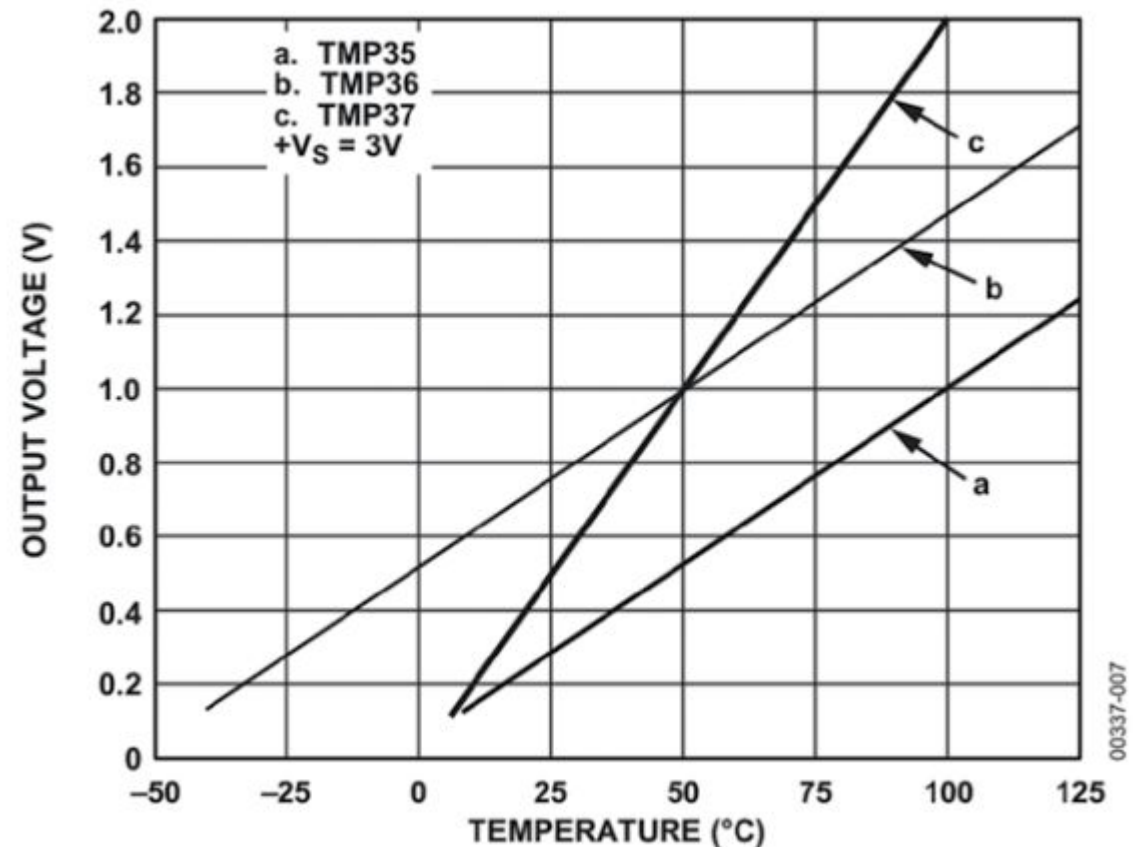


[\[Link\]](#)

# TMP36 Temperature Sensor

The TMP36 temperature sensor easily correlates temperature readings in Celsius with voltage output levels.

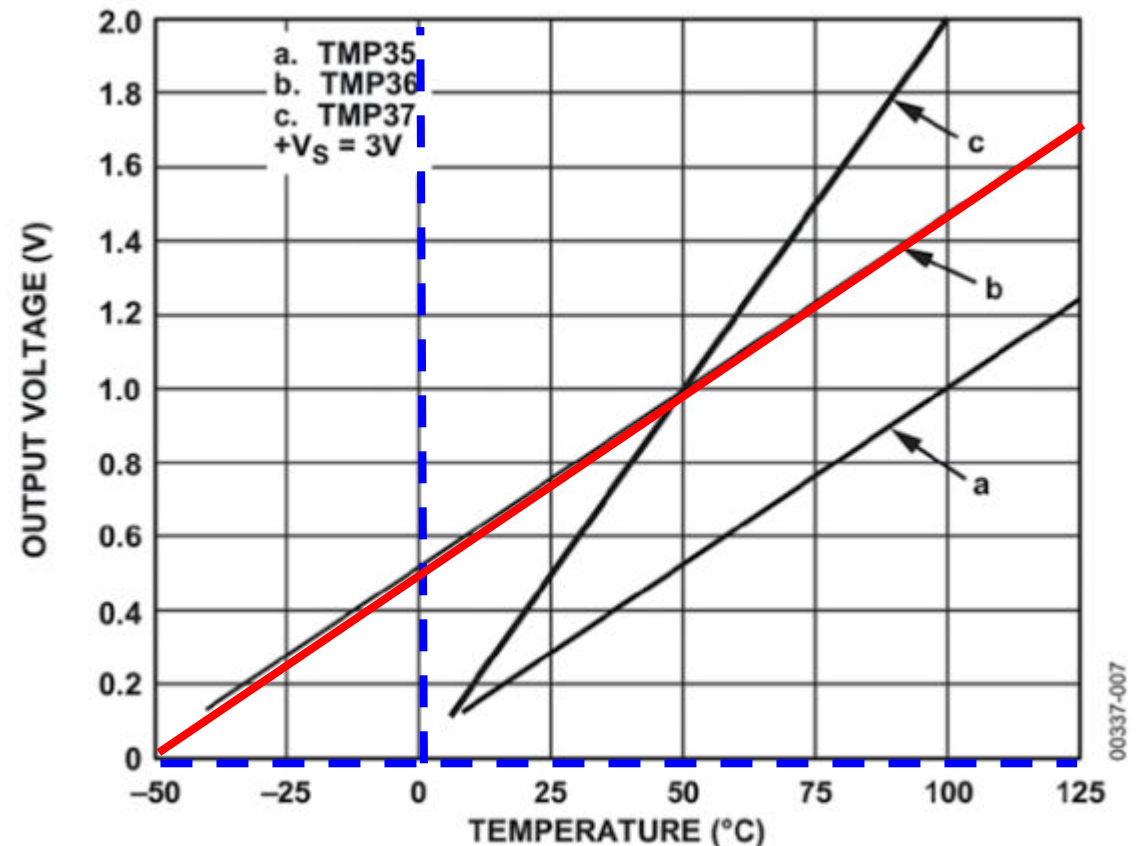
- Temperature range  
-50°C to 125°C
- Voltage output range  
0V to 1.75V



# TMP36 Temperature Sensor

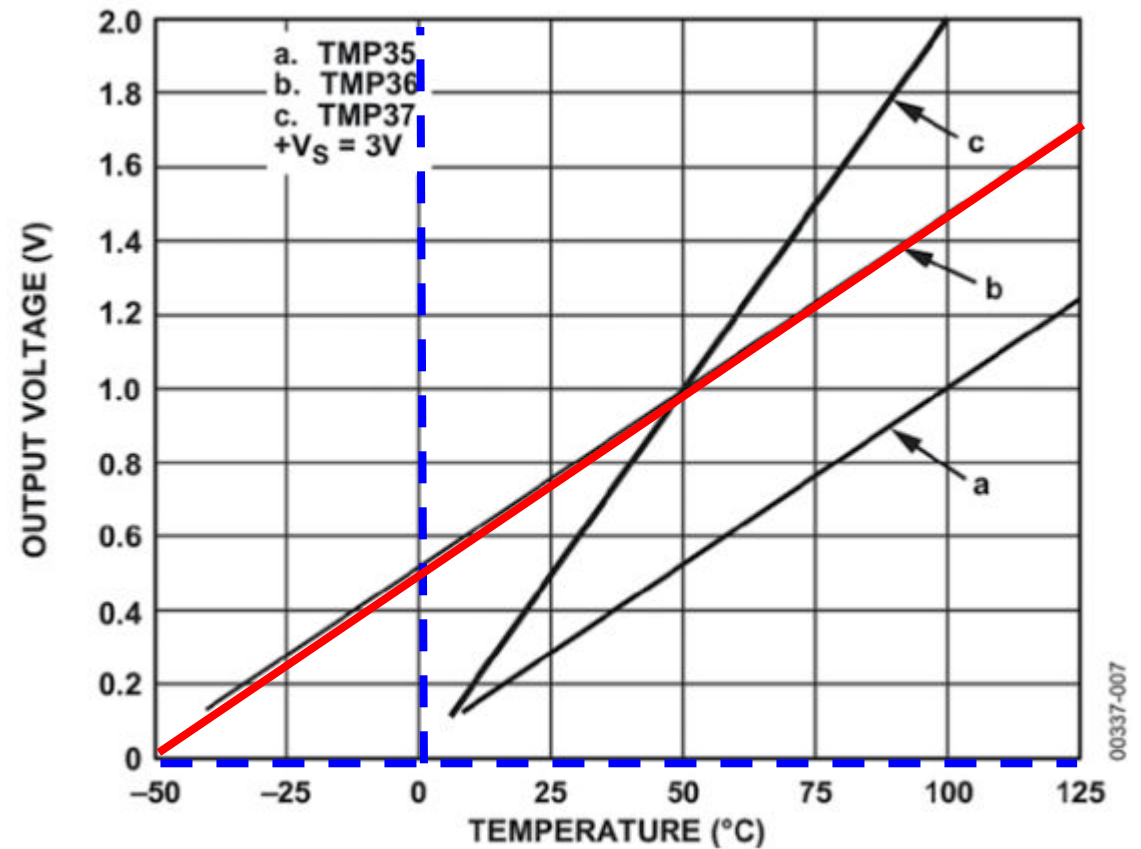
The TMP36 temperature sensor easily correlates temperature readings in Celsius with voltage output levels.

- Temperature range  
-50°C to 125°C
- Voltage output range  
0V to 1.75V



# TMP36 Temperature Sensor

- Temperature range  
-50°C to 125°C
- Voltage output range  
0V to 1.75V



# TMP36 Temperature Sensor

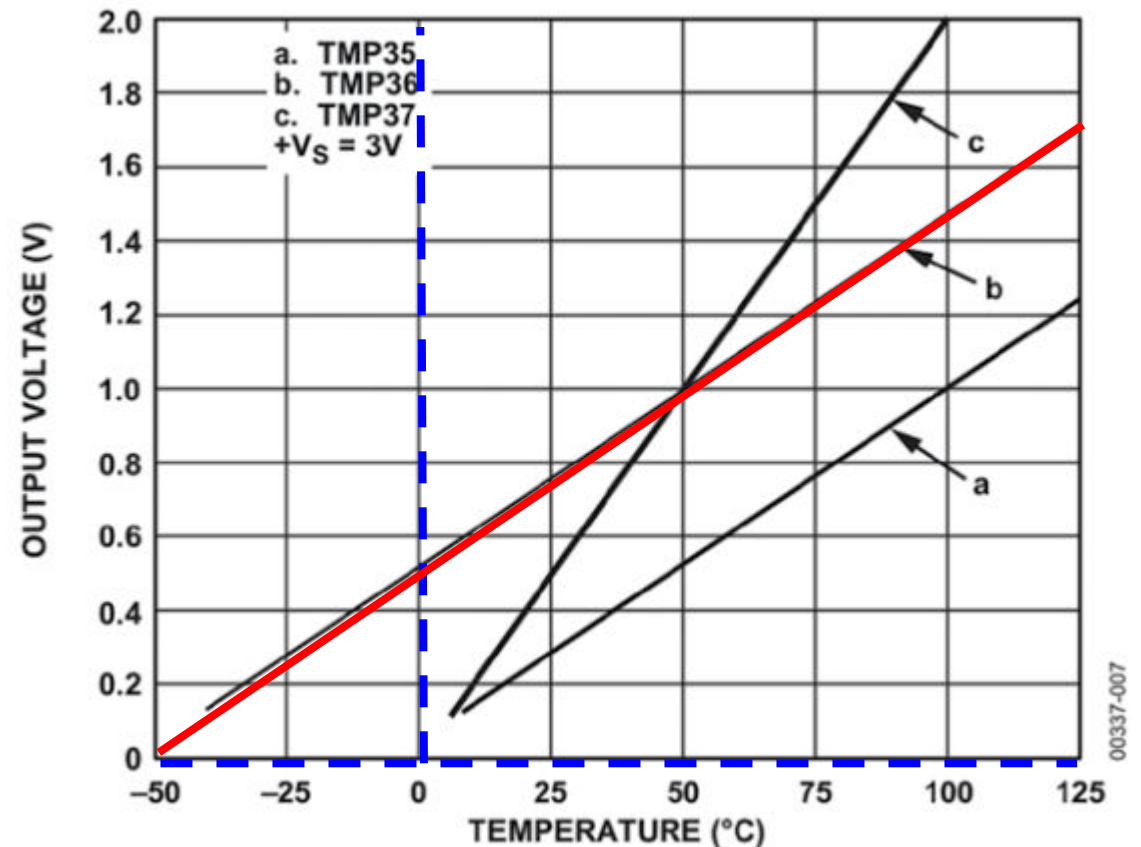
- Temperature range  
-50°C to 125°C
- Voltage output range  
0V to 1.75V

$$V = mT + c$$

$$V = 0.01T + 0.5$$

$$100V = T + 50$$

$$T = 100V - 50$$



# TMP36 Temperature Sensor

$$T = 100V - 50$$

1mV change per 1°C change. Need to convert V to mV.

$$T = 100V - 50$$

$$T = 100(\text{mV}/1000) - 50$$

$$T = 0.1\text{mV} - 50$$

$$10T = \text{mV} - 500$$

$$\text{Temperature}(\text{°C}) \times 10 = \text{voltage}(\text{mV}) - 500$$



# TMP36 Temperature Sensor

Suppose the ADC output is 143. What is the room temperature?  
Assume that the ADC is 10 bits. Reference voltage is 5v.

# TMP36 Temperature Sensor

Suppose the ADC output is 143. What is the room temperature?  
Assume that the ADC is 10 bits. Reference voltage is 5v.

$$\frac{ADCResolution}{AREF} = \frac{ADCReading}{AnalogInput}$$

# TMP36 Temperature Sensor

Suppose the ADC output is 143. What is the room temperature?  
Assume that the ADC is 10 bits. Reference voltage is 5v.

$$\begin{aligned}1023 / 5 &= 143 / x \\ \Rightarrow x &= (143 * 5) / 1023 \\ \Rightarrow x &= 0.7 \text{ V} \\ \Rightarrow x &= 700 \text{ mV}\end{aligned}$$

$$\frac{ADCResolution}{AREF} = \frac{ADCReading}{AnalogInput}$$

# TMP36 Temperature Sensor

Suppose the ADC output is 143. What is the room temperature?  
Assume that the ADC is 10 bits. Reference voltage is 5v.

$$\begin{aligned}1023 / 5 &= 143 / x \\ \Rightarrow x &= (143 * 5) / 1023 \\ \Rightarrow x &= 0.7 \text{ V} \\ \Rightarrow x &= 700 \text{ mV}\end{aligned}$$

$$\begin{aligned}10T &= \text{mV} - 500 \\ \Rightarrow 10T &= 700 - 500 \\ \Rightarrow 10T &= 200 \\ \Rightarrow T &= 20\end{aligned}$$

$$\frac{ADCResolution}{AREF} = \frac{ADCReading}{AnalogInput}$$

# TMP36 Temperature Sensor

Suppose the ADC output is 143. What is the room temperature?  
Assume that the ADC is 10 bits. Reference voltage is 5v.

$$\begin{aligned}1023 / 5 &= 143 / x \\ \Rightarrow x &= (143 * 5) / 1023 \\ \Rightarrow x &= 0.7 \text{ V} \\ \Rightarrow x &= 700 \text{ mV}\end{aligned}$$

$$\begin{aligned}10T &= \text{mV} - 500 \\ \Rightarrow 10T &= 700 - 500 \\ \Rightarrow 10T &= 200 \\ \Rightarrow T &= 20\end{aligned}$$

$$\frac{ADCResolution}{AREF} = \frac{ADCReading}{AnalogInput}$$

Values will be different  
for Blue Pill as its ADC  
is 12 bits.

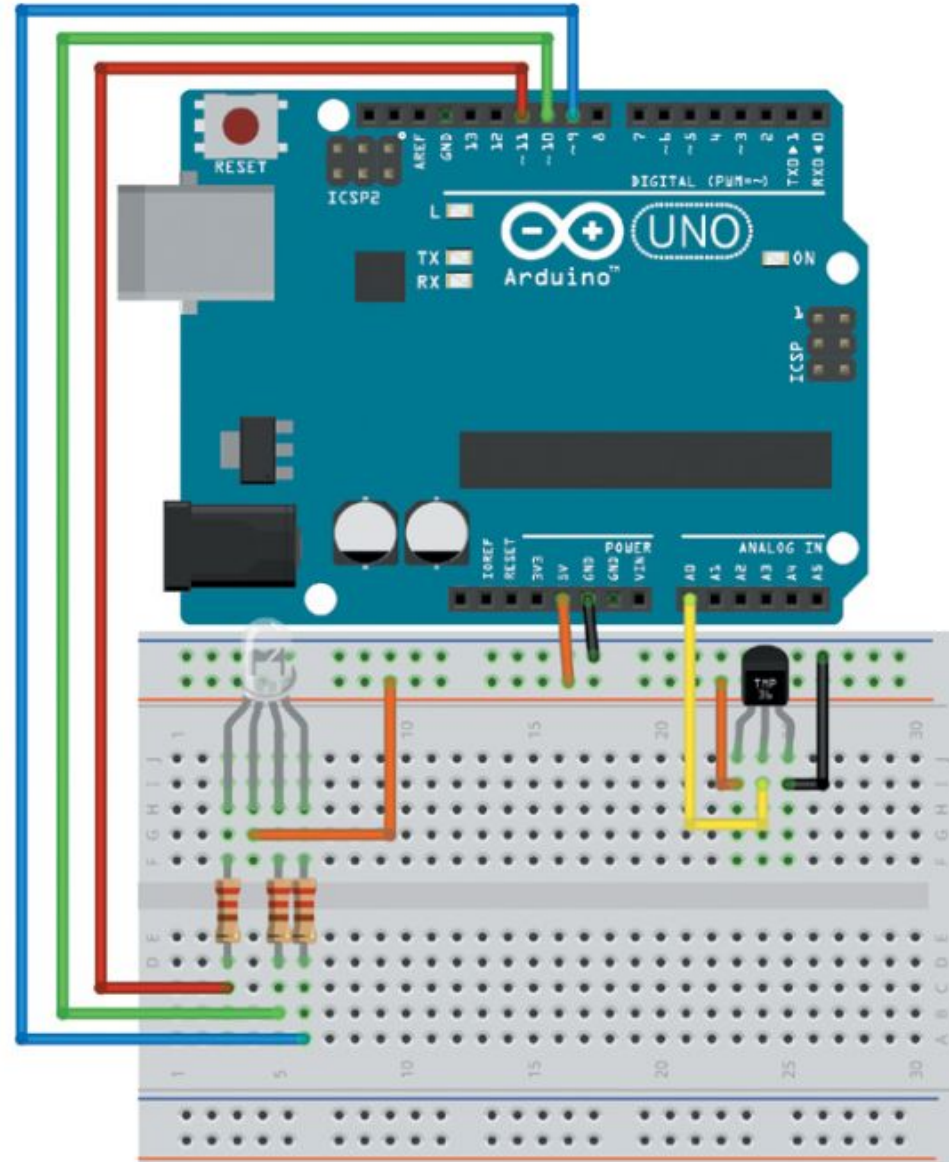
# TMP36 Temperature Sensor

For our program, we will be considering 3 temperatures.

- Normal: 20°C -> 143
- Too Cold: 18°C -> 139
- Too Hot: 22°C -> 147

These values will serve as the cutoffs that will change the color of the LED to indicate that it is too hot or too cold.

# TMP36 Temperature Sensor



# TMP36 Temperature Sensor

```
// Temperature Alert!
const int BLED=B0; // Blue LED Cathode on Pin B0
const int GLED=B1; // Green LED Cathode on Pin B1
const int RLED=B2; // Red LED Cathode on Pin B2
const int TEMP=A0; // Temp Sensor is on pin A0
const int LOWER_BOUND=139; // Lower Threshold
const int UPPER_BOUND=147; // Upper Threshold
int val = 0; // Variable to hold analog reading

void setup() {
    pinMode (TEMP, INPUT);
    pinMode (BLED, OUTPUT); // Set Blue LED as Output
    pinMode (GLED, OUTPUT); // Set Green LED as Output
    pinMode (RLED, OUTPUT); // Set Red LED as Output
}
```



# TMP36 Temperature Sensor

```
void loop() {  
    val = analogRead(TEMP);  
    if (val < LOWER_BOUND) { // LED is Blue  
        digitalWrite(RLED, HIGH);  
        digitalWrite(GLED, HIGH);  
        digitalWrite(BLED, LOW);  
    }  
    else if (val > UPPER_BOUND) { // LED is Red  
        digitalWrite(RLED, LOW);  
        digitalWrite(GLED, HIGH);  
        digitalWrite(BLED, HIGH);  
    }  
    else { // LED is Green  
        digitalWrite(RLED, HIGH);  
        digitalWrite(GLED, LOW);  
        digitalWrite(BLED, HIGH);  
    }  
}
```

# Using Variable Resistors to Make Your Own Analog Sensors

Thanks to physics, tons of devices change resistance as a result of physical action. For example

- Some conductive inks change resistance when squished or flexed (**force sensors and flex sensors**)
- Some semiconductors change resistance when struck by light (**photoresistors**)
- Some polymers change resistance when heated or cooled (**thermistors**)

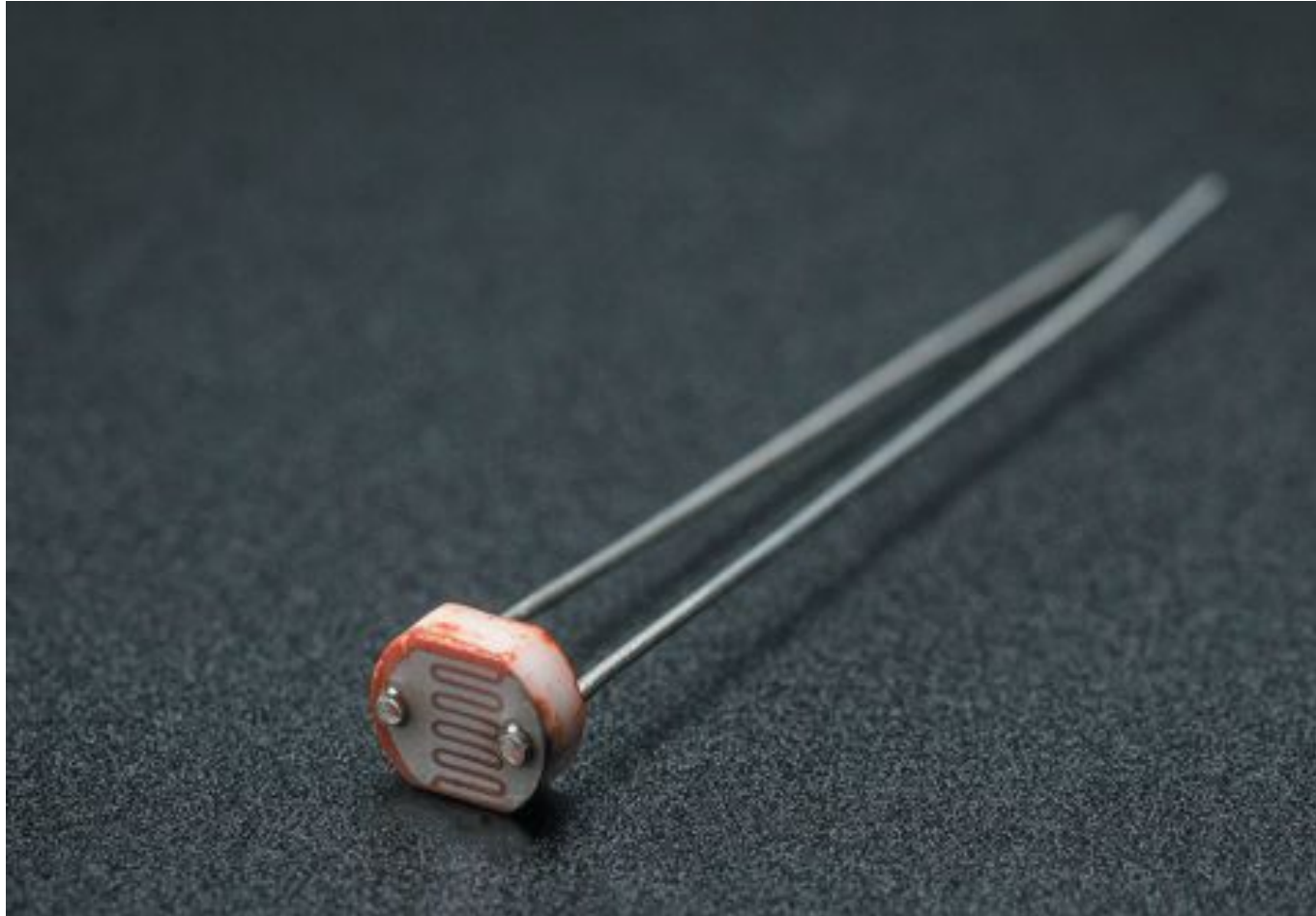
Because these sensors are changing resistance and not voltage, you need to create a voltage divider circuit so that you can measure their resistance change.

# Using Variable Resistors to Make Your Own Analog Sensors



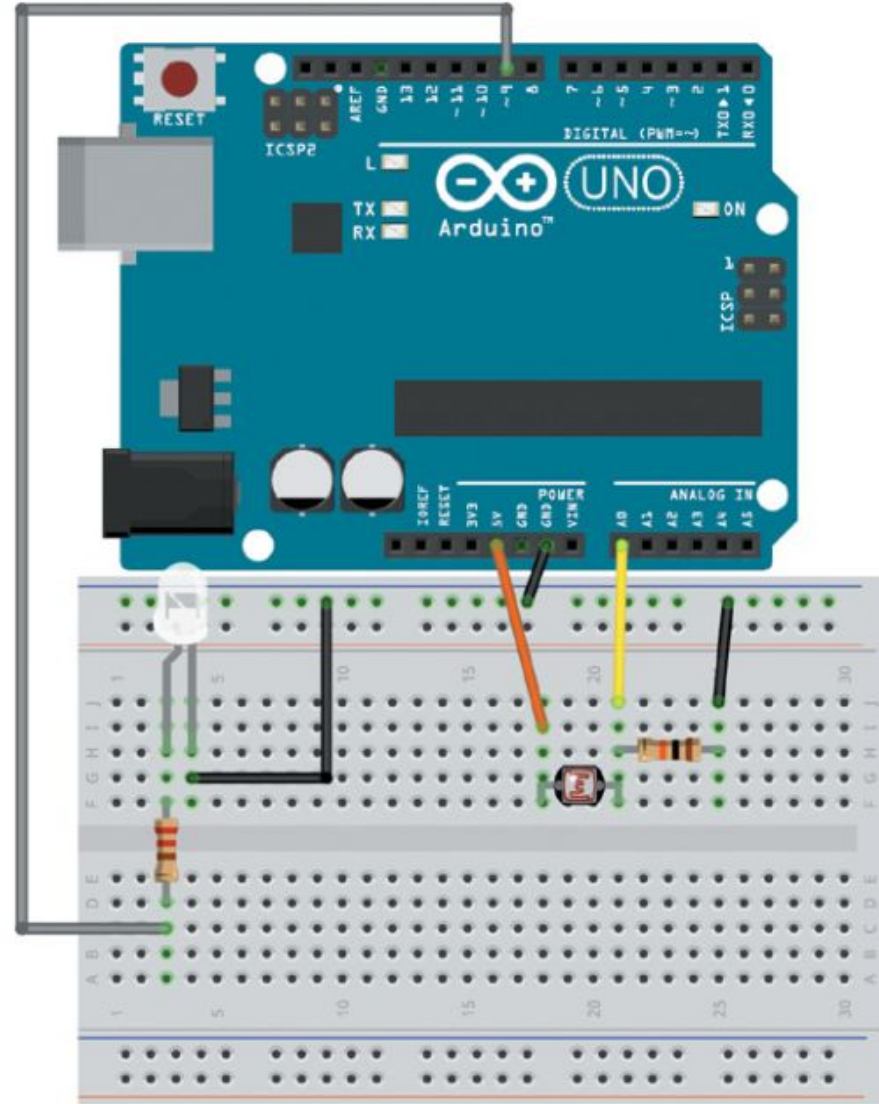
$$V_{R1} = V_{in} \left( \frac{R1}{R1 + R2} \right)$$

# Using Variable Resistors to Make Your Own Analog Sensors



In my case, I found that a dark room has a value of around 200 and a completely bright room has a value around 900. These values will vary for you based upon your lighting conditions.

# Using Analog Inputs to Control Analog Outputs



# Using Analog Inputs to Control Analog Outputs

Arduino programming language has two functions that are useful for mapping between two sets of values: the `map()` and `constrain()` functions. The `map()` function looks like this:

```
output = map(value, fromLow, fromHigh, toLow, toHigh)
```

# Using Analog Inputs to Control Analog Outputs

```
output = map(value, fromLow, fromHigh, toLow, toHigh)
```

**value** is the information you are starting with. In your case, that's the most recent reading from the analog input.

**fromLow** and **fromHigh** are the input boundaries. These are values you found to correspond to the minimum and maximum brightness in your room. In my case, they were 200 and 900

**toLow** and **toHigh** are the values you want to map the brightness values to. Because `analogWrite()` expects a value between 0 and 255, you use those values. However, you want a darker room to map to a brighter LED. Therefore, when the input from the ADC is a low value, you want the output to the LED's PWM pin to be a high value, and vice versa.

# Using Analog Inputs to Control Analog Outputs

Conveniently, the map function can handle this automatically; simply swap the high and low values so that the low value is 255 and the high value is 0.

The map() function creates a linear mapping. For example, if your fromLow and fromHigh values are 200 and 900, respectively, and your toLow and toHigh values are 255 and 0, respectively, 550 maps to 127 because 550 is halfway between 200 and 900 and 127 is halfway between 255 and 0.



# Using Analog Inputs to Control Analog Outputs

Importantly, however, the `map()` function does not constrain these values.

So, if the photoresistor does measure a value below 200, it is mapped to a value above 255 (because you are inverting the mapping). Obviously, you don't want that because you can't pass a value greater than 255 to the `analogWrite()` function. You can deal with this by using the `constrain()` function.

```
output = constrain(value, min, max)
```

If you pass the output from the `map` function into the `constrain` function, you can set the `min` to 0 and the `max` to 255, ensuring that any numbers above or below those values are constrained to either 0 or 255.

# Automatic Night Light

```
// Automatic Night Light
const int WLED=PA0; // White LED Anode on pin PA0 (PWM)
const int LIGHT=PA1; // Light Sensor on Analog Pin PA1
const int MIN_LIGHT=200; // Minimum Expected light value
const int MAX_LIGHT=900; // Maximum Expected Light value
int val = 0; // Variable to hold the analog reading

void setup() {
    pinMode(WLED, OUTPUT); // Set White LED pin as output
}

void loop() {
    val = analogRead(LIGHT); // Read the light sensor
    val = map(val, MIN_LIGHT, MAX_LIGHT, 255, 0); // Map the light reading
    val = constrain(val, 0, 255); // Constrain light value
    analogWrite(WLED, val); // Control the White LED
}
```



Thank You!