كلية الهندسة
قسم هندسة الروبوت والأنظمة الذكية

جَامعة المَنارة
MANARA UNIVERSITY

تطبيقات ميكاترونيك1

محاضرة 2

Microcontrollers
Arduino

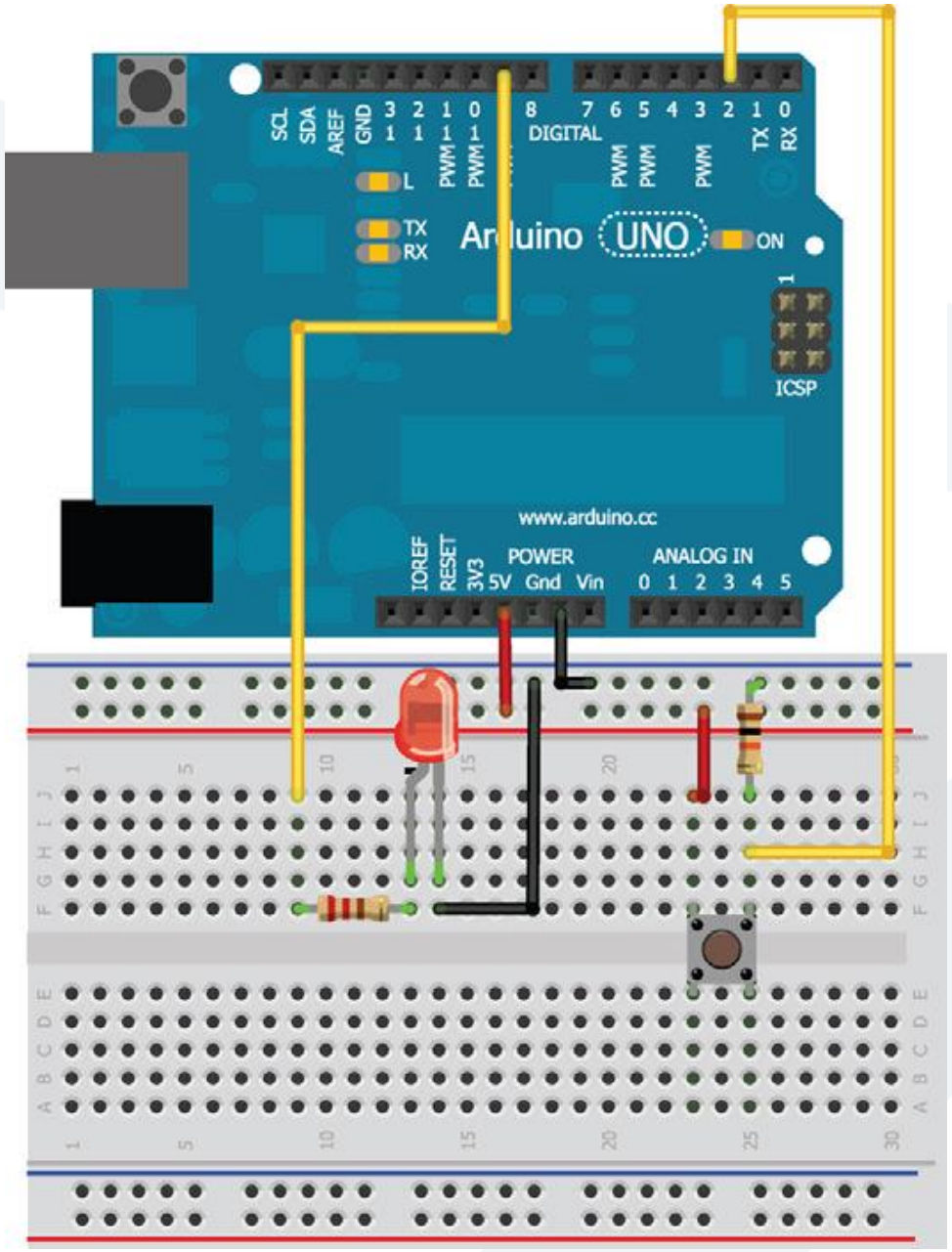
د. فادي متوج
د.عيسى الغنام

2024-2023

Reading Digital Inputs

After we have successfully generated both digital and analog outputs. The next step is to read digital inputs, such as switches and buttons, so that we can interact with our project in real time.

Reading Digital Inputs with Pulldown Resistors



Pullup and Pulldown resistor

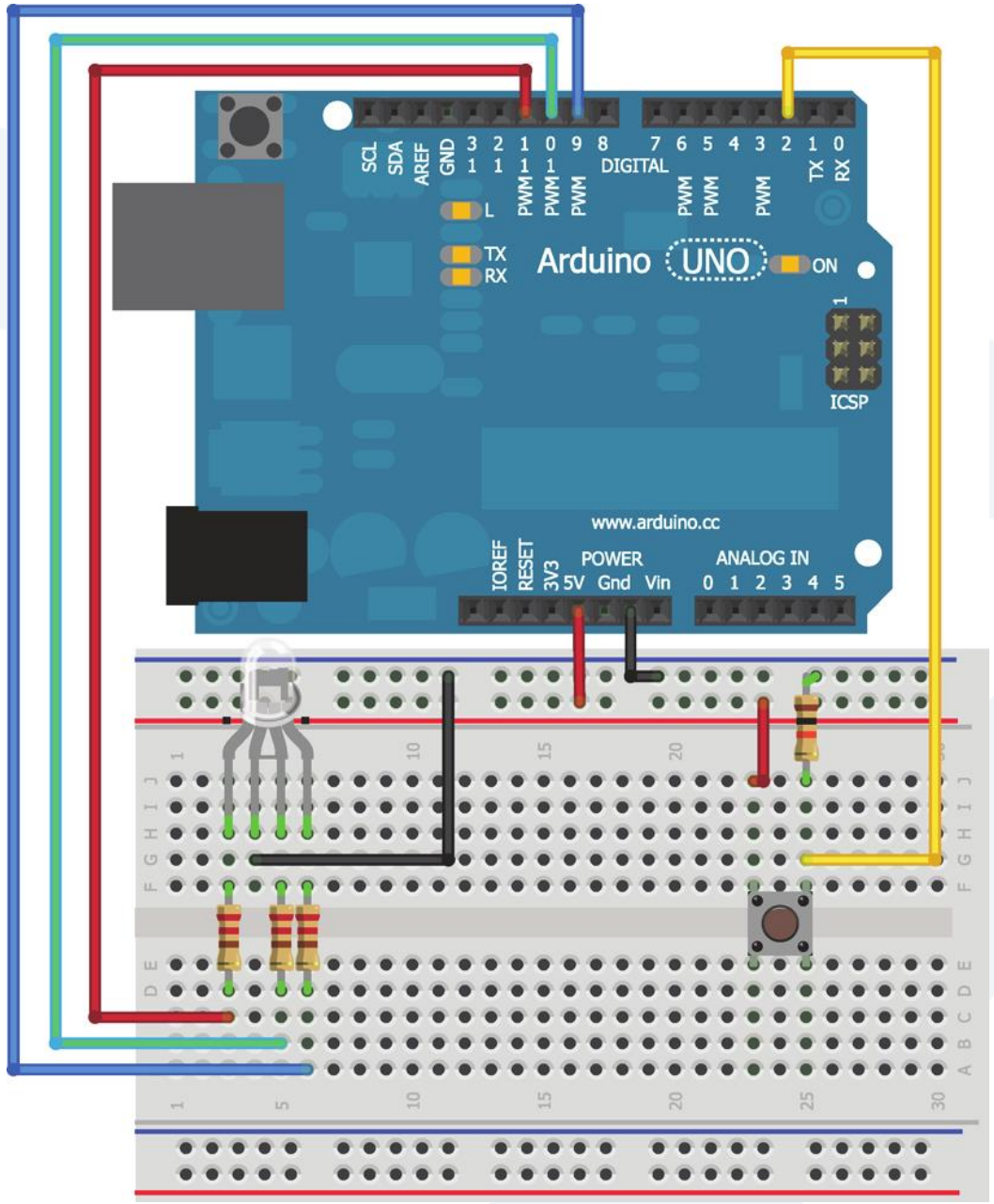
- Nearly all digital inputs use a pullup or pulldown resistor to set the "default state" of the input pin.
- Imagine the circuit without the 10kΩ resistor.
  - When the button is pressed ? In this scenario, the pin would obviously read a high value.
  - When the button is not being pressed? In that scenario, the input pin we are reading is essentially connected to nothing—the input pin is said to be "**floating**". And because the pin is not physically connected to 0V or 5V, reading it could cause unexpected results as electrical noise causes its value to fluctuate between high and low.
- To avoid this, the pulldown resistor is installed.

- This example uses a pulldown resistor, but we can also use a pullup resistor by connecting the resistor to 5V instead of ground and by connecting the other side of the button to ground.
- In this setup, the input pin reads a high-logic value when the button is unpressed and a low-logic value when the button is being pressed.
- Pulldown and pullup resistors are important because they ensure that the button does not create a short circuit between 5V and ground when pressed and that the input pin is never left in a floating state.

```
const int LED=9;          //The LED is connected to pin 9
const int BUTTON=2;   //The Button is connected to pin 2
void setup()
{
pinMode (LED, OUTPUT);       //Set the LED pin as an output
pinMode (BUTTON, INPUT);   //Set button as input (not required
}
)
void loop()
{
if (digitalRead(BUTTON) == LOW)
{
digitalWrite(LED, LOW);
}
else
{
digitalWrite(LED, HIGH); } }
```

Building a Controllable RGB LED Nightlight

- It's possible to mix colors with an RGB LED by changing the brightness of each color.
- In this scenario, we use a common cathode LED. That means that the LED has four leads. One of them is a cathode pin that is shared among all three diodes, while the other three pins connect to the anodes of each diode color.
- We wire that LED up to three PWM pins through current-limiting resistors on the Arduino.


SCL
SDA
AREF
GND
DIGITAL
L
TX
RX
Arduino
UNO
ON
ICSP
www.arduino.cc
IOREF
RESET
3V3
POWER
5V
Gnd
Vin
ANALOG IN
0 1 2 3 4 5

- In the following program, we have defined seven total color states, plus one off state for the LED.
- Using the `analogWrite()` function, we can choose our own color-mixing combinations.
- An LED state counter is incremented each time the button is pressed, and it is reset back to zero when we cycle through all the options.

```
const int BLED=9;      //Blue LED on Pin 9
const int GLED=10;    //Green LED on Pin 10
const int RLED=11;     //Red LED on Pin 11
const int BUTTON=2;  //The Button is connected to pin 2
int ledMode = 0;       //Cycle between LED states
void setup()
{
pinMode (BLED, OUTPUT);     //Set Blue LED as Output
pinMode (GLED, OUTPUT);     //Set Green LED as Output
pinMode (RLED, OUTPUT);     //Set Red LED as Output
pinMode (BUTTON, INPUT);   //Set button as input (not required)
}
```

```
/* LED Mode Selection
* Pass a number for the LED state and set it accordingly.
*/
void setMode(int mode) {
//RED
if (mode == 1) {
digitalWrite(RLED, HIGH);
digitalWrite(GLED, LOW);
digitalWrite(BLED, LOW);
}
//GREEN
else if (mode == 2) {
digitalWrite(RLED, LOW);
digitalWrite(GLED, HIGH);
digitalWrite(BLED, LOW); }
//BLUE
else if (mode == 3) {
digitalWrite(RLED, LOW);
digitalWrite(GLED, LOW);
digitalWrite(BLED, HIGH);}
```

```
//PURPLE (RED+BLUE)
if (mode == 4) {
analogWrite(RLED, 127);
analogWrite(GLED, 0);
analogWrite(BLED, 127); }
//TEAL (BLUE+GREEN)
else if (mode == 5) {
analogWrite(RLED, 0);
analogWrite(GLED, 127);
analogWrite(BLED, 127); }
//ORANGE (GREEN+RED)
else if (mode == 6) {
analogWrite(RLED, 127);
analogWrite(GLED, 127);
analogWrite(BLED, 0); }
//WHITE (GREEN+RED+BLUE)
else if (mode == 7) {
analogWrite(RLED, 85);
analogWrite(GLED, 85);
analogWrite(BLED, 85); }
```

```
//OFF (mode = 0)
else{
digitalWrite(RLED, LOW);
digitalWrite(GLED, LOW);
digitalWrite(BLED, LOW); }
}
void loop()
{
if (digitalRead(BUTTON) == HIGH)  //if it was pressed...
{
ledMode++;    //increment the LED value
}
// if you've cycled through the different options,
// reset the counter to 0
if (ledMode == 8) ledMode = 0;
setMode(ledMode);  //change the LED state
}
```

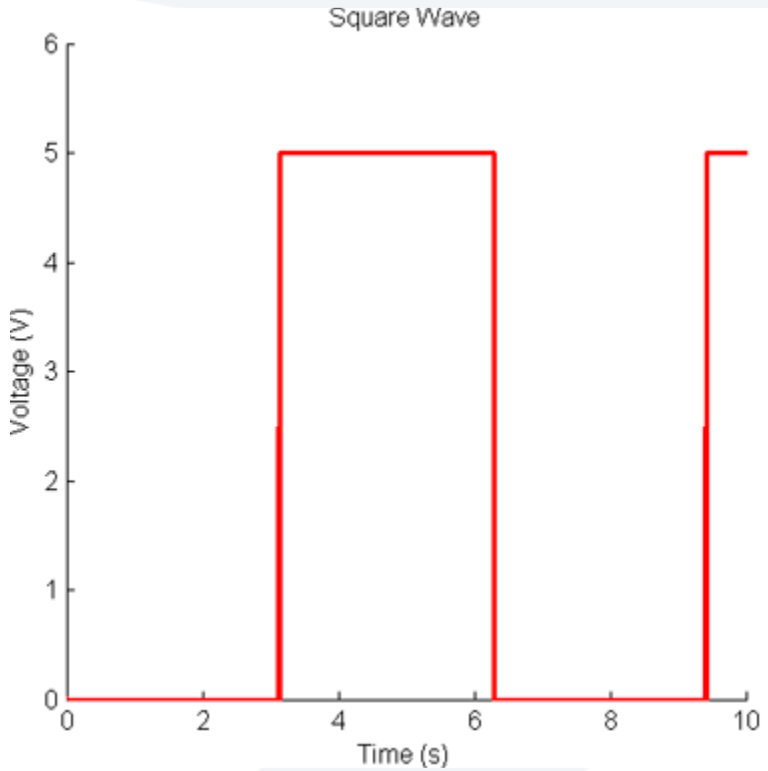
Reading Analog Sensors

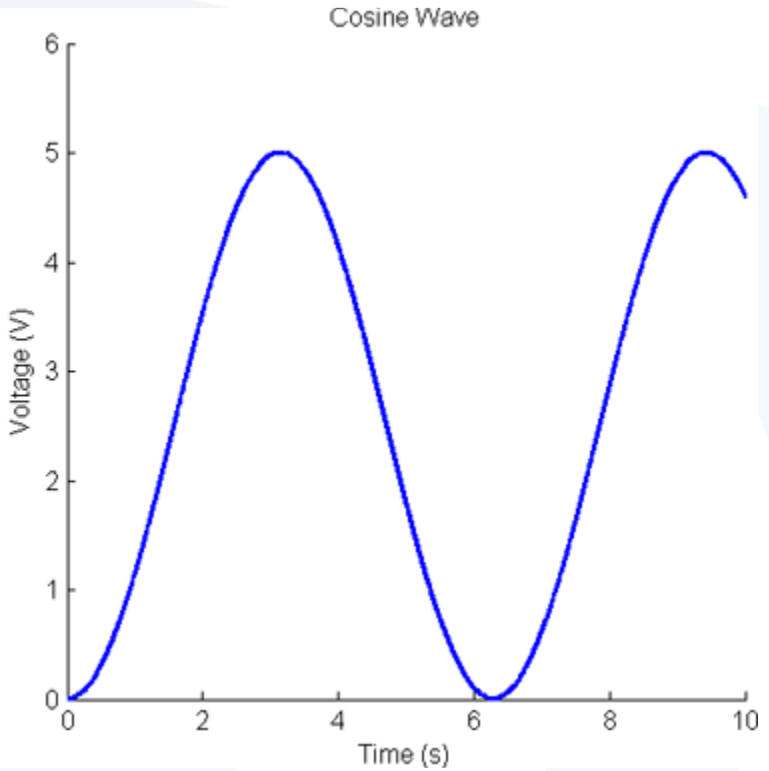
- The world around us is analog. Even though we might hear that the world is "going digital," the majority of observable features in our environment will always be analog in nature.
- The world can assume an infinite number of states, whether we are considering the color of sunlight, the temperature of the ocean, or the concentration of contaminants in the air.
- We will focus on developing techniques for discretizing these infinite possibilities into digital values that can be analyzed with a microcontroller system like the Arduino

Understanding Analog and Digital Signals

- If we want our devices to interface with the world, they will inevitably be interfacing with analog data.
- Digital information (what our computer or the Arduino processes) is a series of binary (or digital) data. Each bit has only has one of two values.
- The world around us, however, rarely expresses information in only two ways. Sunlight, wind speeds, and cars passing or people walking around cannot be classified as binary data.
- Sunlight is not on or off; its brightness varies over the course of a day. Similarly, wind does not just have two states; it gusts at different speeds all the time.

Comparing Analog and Digital Signals





- Analog signals are those that cannot be discretely classified; they vary within a range, theoretically taking on an infinite number of possible values within that range.
- A computer system could never feasibly measure an infinite number of decimal places for an analog value because memory and computer power must be finite values.
- If that's the case, how can we interface our Arduino with the "real world"?
- The answer is analog-to-digital converters (ADC), which can convert analog values into digital representations with a finite amount of precision and speed.

Converting an Analog Signal to a Digital One

Suppose that we want to measure the brightness of our room:

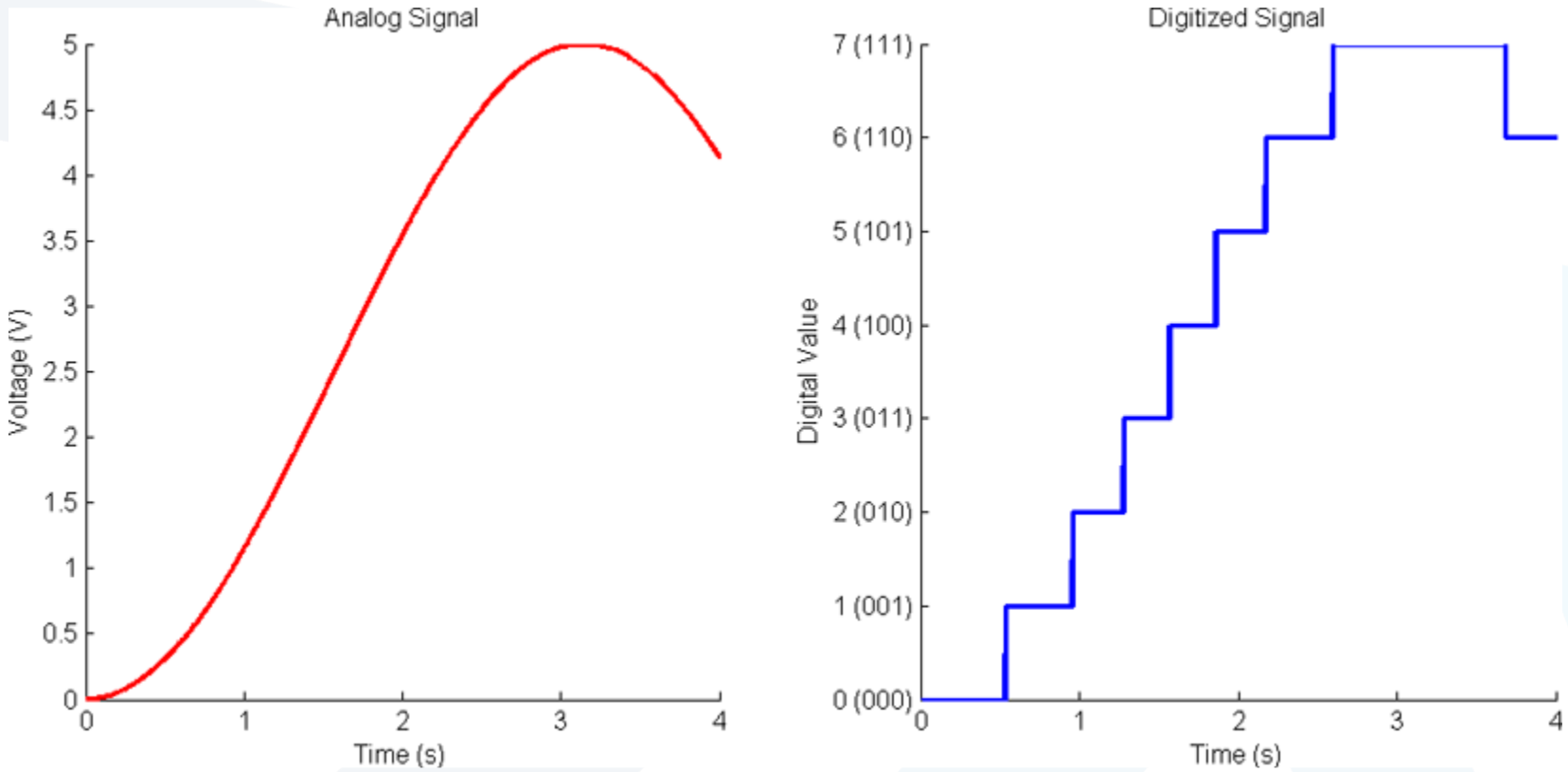
- Good light sensor could produce a varying output voltage that changes with the brightness of the room.
- When it is pitch black, the device would output 0V, and when it's completely saturated by light, it would output 5V, with values in between corresponding to the varying amount of light.

That's all well and good, but how do we go about reading those values with an Arduino to figure out how bright the room is?

- We can use the Arduino's analog-to-digital converter (ADC) pins to convert analog voltage values into number representations that we can work with.

- The accuracy of an ADC is determined by the resolution.
- In the case of the Arduino Uno, there is a **10-bit ADC** for doing your analog conversions.
- "10-bit" means that the ADC can subdivide (or quantize) an analog signal into 210 different values.
- If we do the math, we will find that $2^{10} = 1024$; hence, the Arduino can assign a value from **0** to **1023** for any analog value that we give it.
- The reference voltage determines the max voltage that we are expecting, and, therefore, the value that will be mapped to 1023.
- So, with a 5V reference voltage, putting 0V on an ADC pin returns a value of 0, 2.5V returns a value of 512 (half of 1023), and 5V returns a value of 1023.

جَامعة
المَنارة
MANARA UNIVERSITY



- In the case of the Arduino Uno, there are 1024 steps rather than the 8 shown here.

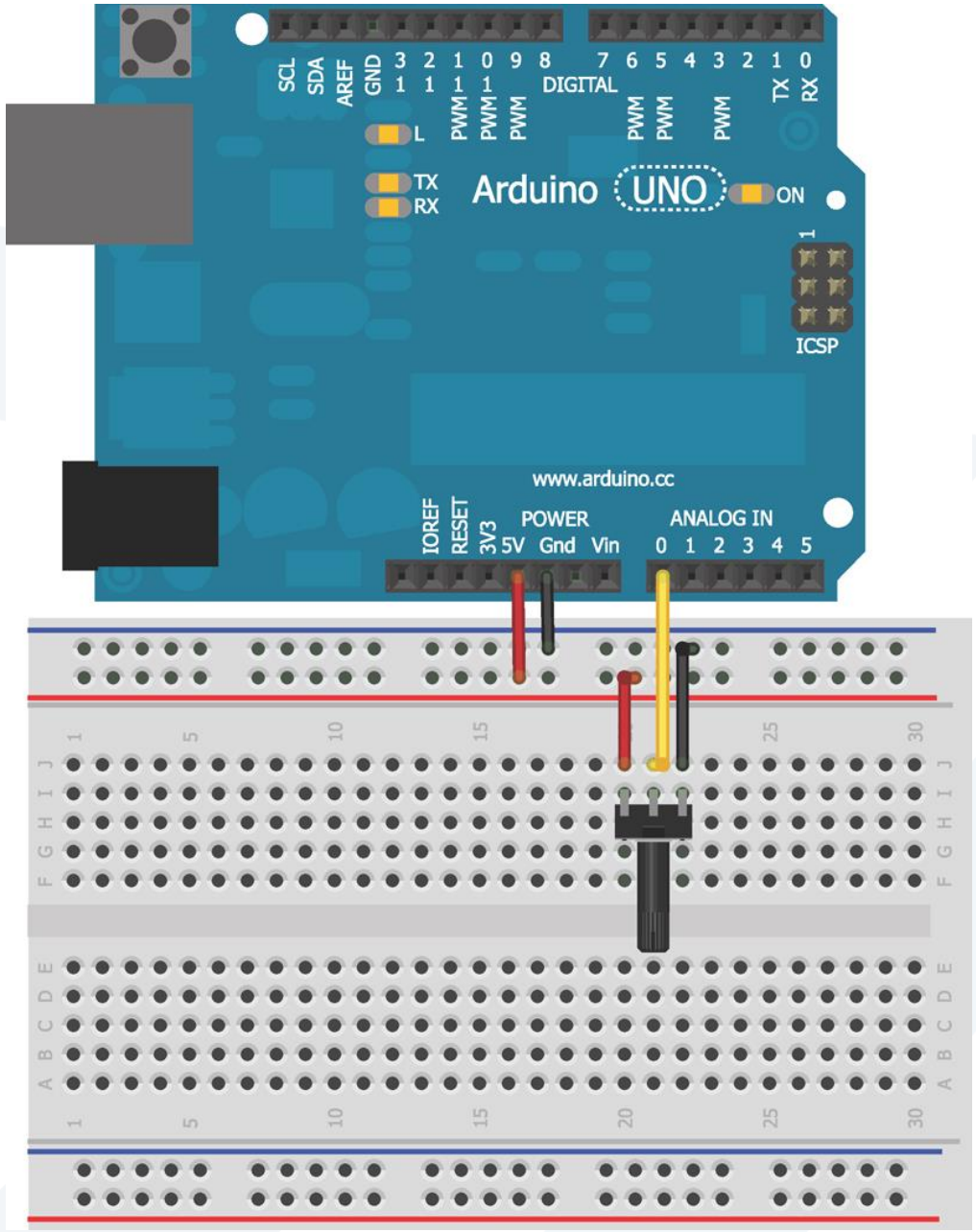
جَامعة
المَنارة
MANARA UNIVERSITY

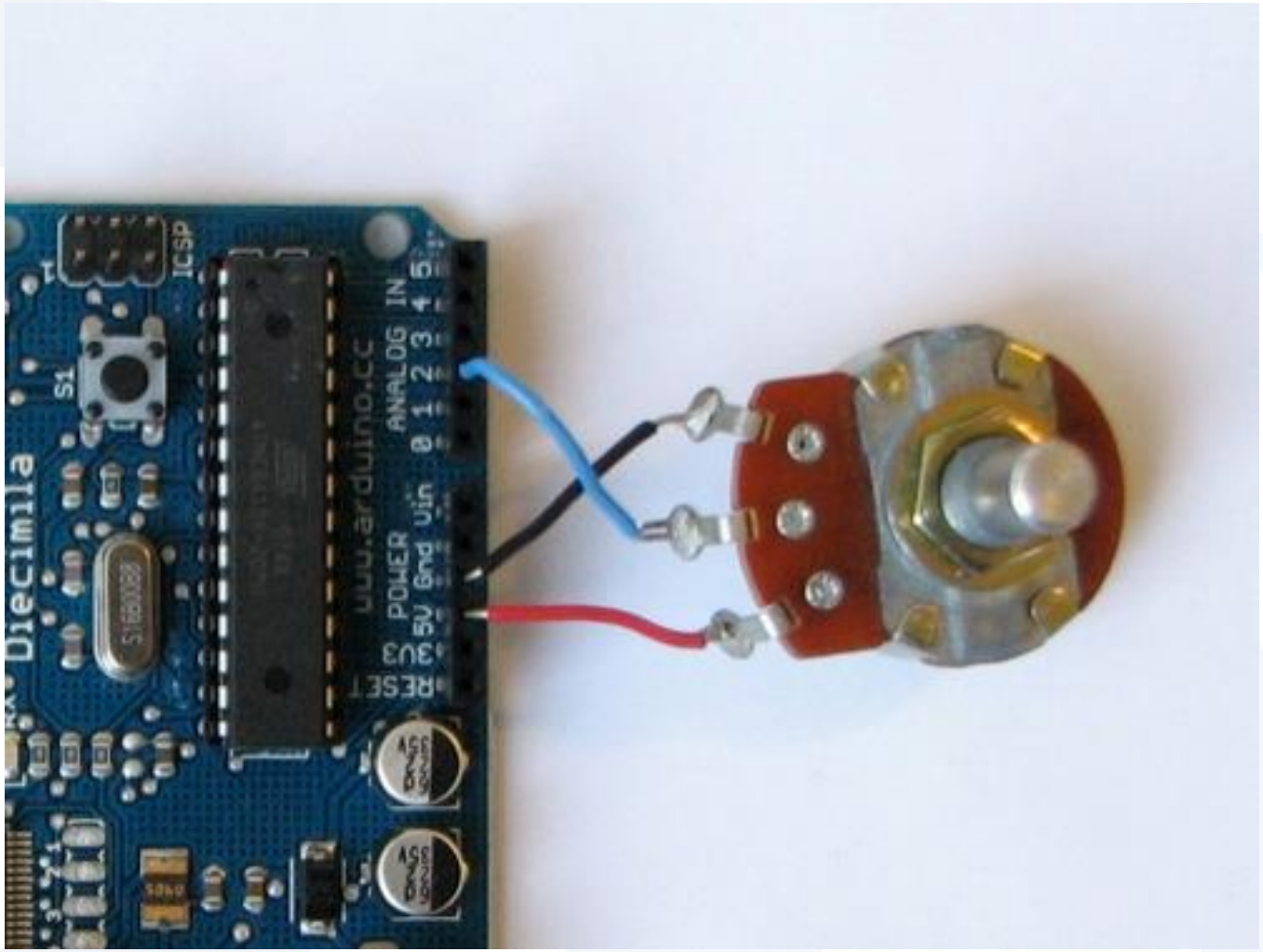
Reading Analog Sensors with the Arduino:

analogRead()

Reading a Potentiometer

- Potentiometers are variable voltage dividers that look like knobs.
- They come in lots of sizes and shapes, but they all have three pins.
- We connect one of the outer pins to ground, and the other to the 5V.
- Potentiometers are symmetrical, so it doesn't matter which side you connect the 5V and ground to.
- We connect the middle pin to analog input 0 on our Arduino.


SCL
SDA
AREF
GND
DIGITAL
L
TX
RX
Arduino
UNO
ON
ICSP
www.arduino.cc
IOREF
RESET
3V3
POWER
5V
Gnd
Vin
ANALOG IN
0 1 2 3 4 5

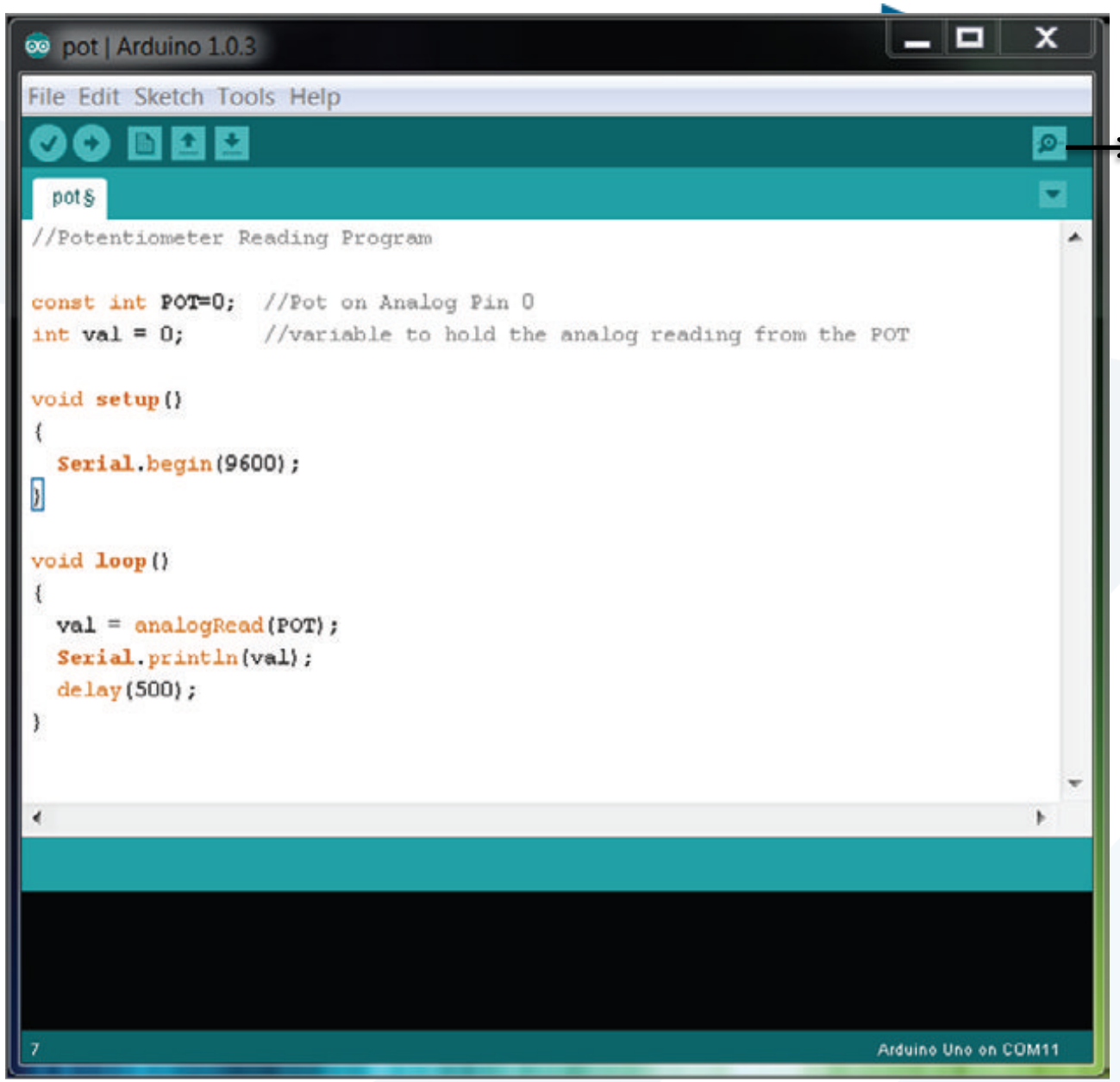

ICSP
S1
Diecimila
www.arduino.cc
ANALOG IN
POWER
Gnd
Vin
5V
3V3
RESET

- As we turn the potentiometer, we vary the voltage that we are feeding into analog input 0 between 0V and 5V.
- We will use the Arduino's **serial communication** functionality to print out the potentiometer's ADC value on our computer as it changes.
- We will 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.

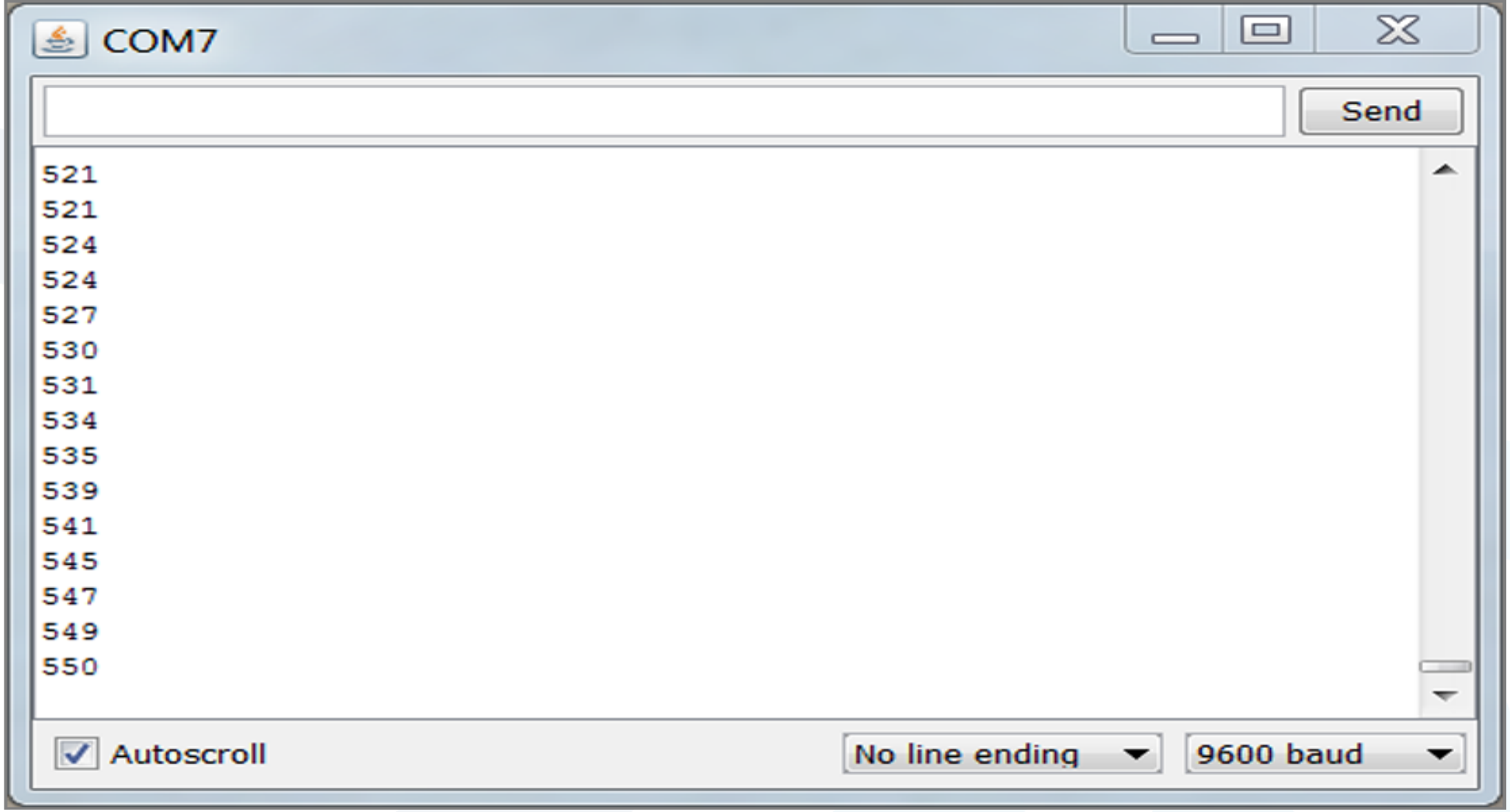
```
//Potentiometer Reading Program
const int POT=0;   //Pot on analog pin 0
int val = 0; //variable to hold the analog reading from the POT
void setup()
{
Serial.begin(9600);
}
void loop()
{
val = analogRead(POT);
Serial.println(val);
delay(500);
}
```

- The serial interface to the computer must be started in the setup.
- `Serial.begin()` takes one argument that specifies the communication speed, or baud rate.
- The baud rate specifies the number of bits being transferred per second.
- Faster baud rates enable us to transmit more data in less time, but can also introduce transmission errors in some communication systems.

- In each iteration through the loop, the **val** variable is set to the present value that the ADC reports from analog pin 0.
- The `analogRead()` command requires the number of the ADC pin to be passed to it. In this case, it's 0 because that's what we hooked the potentiometer up to.
- we can also pass A0, though the `analogRead()` function , or we can pass 0 as shorthand.
- After the value has been read (a number between 0 and 1023), `Serial.println()` prints that value over serial to the computer's serial terminal, followed by a "newline" that advances the cursor to the next line.
- The loop then delays for half a second (so that the numbers don't scroll by faster than we can read them), and the process repeats.
- After loading this onto our Arduino, we will notice that the **TX LED** on our Arduino is blinking every 500ms
- This LED indicates that our Arduino is transmitting data via the USB connection to the serial terminal on our computer.



Serial monitor



- If we are getting funky characters, we must make sure that we have the baud rate set correctly.
- Because we set it to 9600 in the code, we need to set it to 9600 in this window as well.

Using Analog Sensors

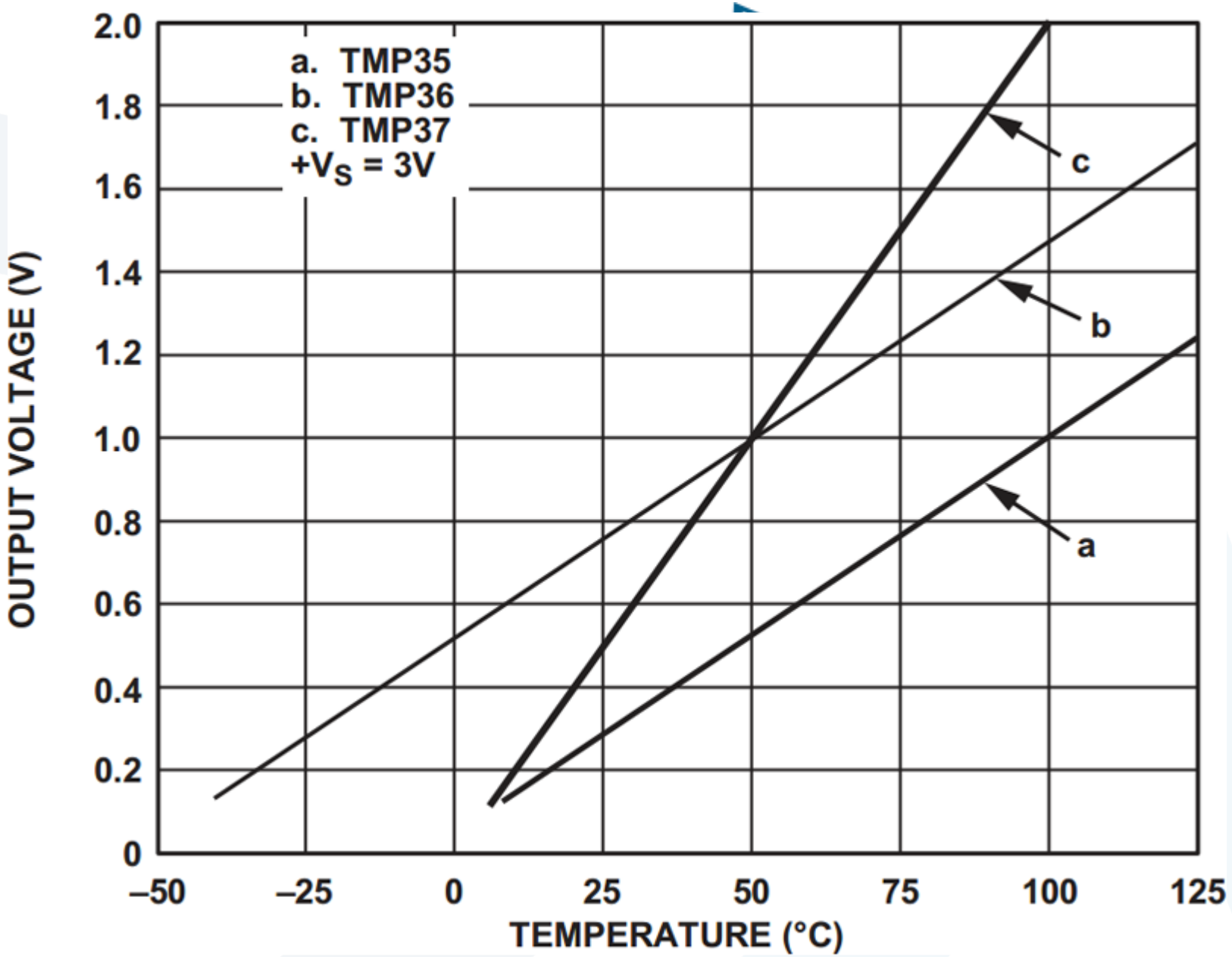
- All kinds of sensors generate analog output values corresponding to "real-world" action. Examples of such include the following:
  - Accelerometers that detect tilting (many smartphones and tablets now have these)
  - Magnetometers that detect magnetic fields (for making digital compasses)
  - Infrared sensors that detect distance to an object
  - Temperature sensors that can tell you about the operating environment of your project
- **Many of these sensors are designed to operate in a manner similar to the potentiometer :** we provide them with a power (VCC) and ground (GND) connection, and they output an analog voltage between VCC and GND on the third pin that we hook up to our Arduino's ADC.

TMP36 Temperature Sensor

- The TMP36 temperature sensor easily correlates temperature readings in Celsius with voltage output levels.
- Since every **10mV** corresponds to **1C**, we can easily create a linear correlation to convert from the voltage we measure back to the absolute temperature of the ambient environment:

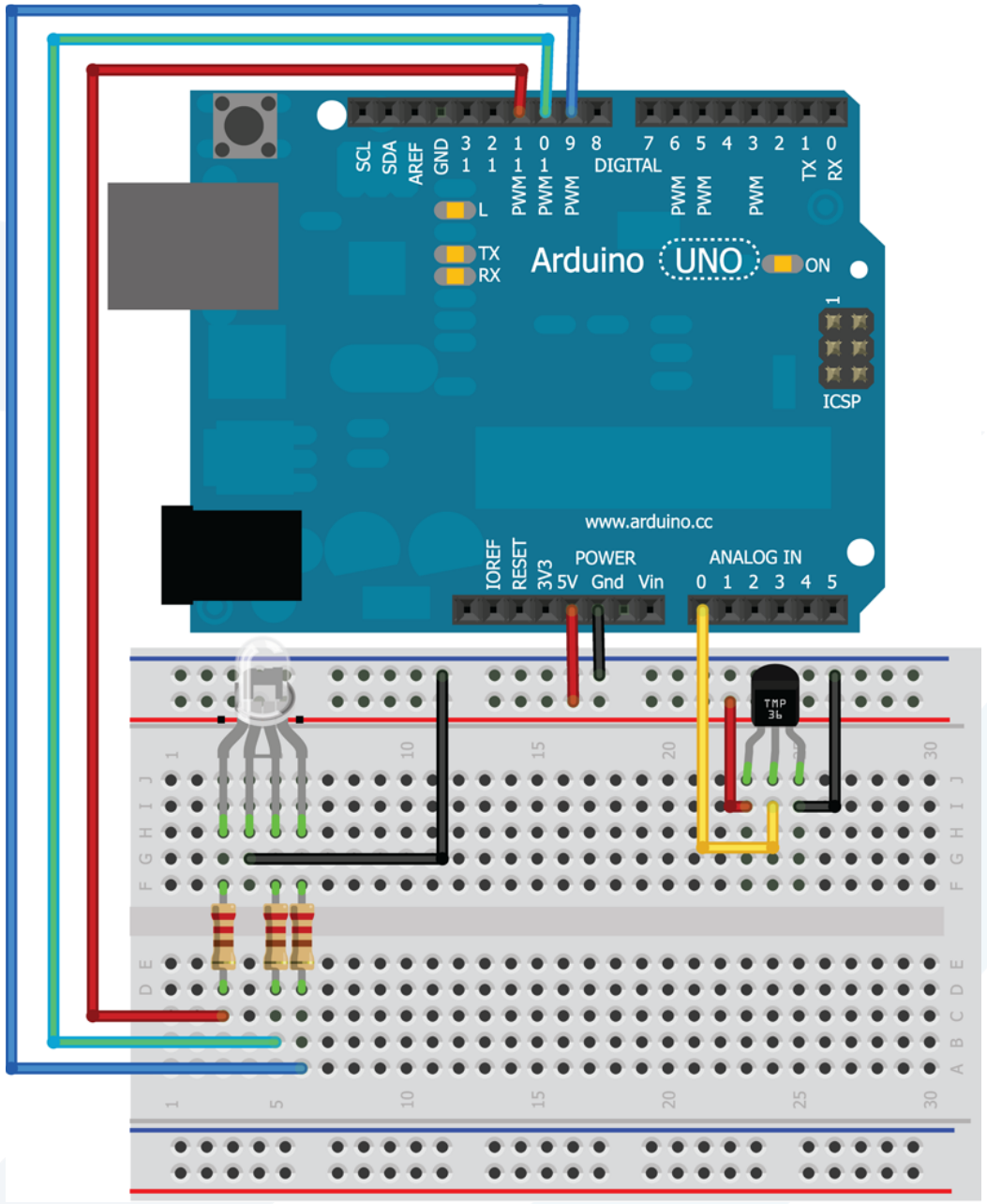
  C = [(Vout in mV) – 500]/10.

- The offset of –500 is for dealing with temperatures below 0C.


a. TMP35
b. TMP36
c. TMP37
+VS = 3V
OUTPUT VOLTAGE (V)
2.0
1.8
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0
–50
–25
0
25
50
75
100
125
TEMPERATURE (°C)
a
b
c

Working with Analog Sensors to Sense Temperature

- This simple example uses the **TMP36** temperature sensor.
- We will make a simple temperature alert system. The light will glow green when the temperature is within an acceptable range, will turn red when it gets too hot, and will turn blue when it gets too cold.
- The following steps are basically the same for any analog sensor we might want to use.


SCL
SDA
AREF
GND
DIGITAL
TX
RX
L
Arduino
UNO
ON
ICSP
www.arduino.cc
IOREF
RESET
3V3
5V
Gnd
Vin
POWER
ANALOG IN
TMP
36

- First , we need to ascertain what values we want to use as our cutoffs.
- From the datasheet, the following equation can be used to convert between the temperature (C) and the voltage (mV):

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

- Plugging in the value of 700mV, we can confirm that it equates to a temperature of 20C.
- Using this same logic, we can determine that 22C is a digital value of 147 and 18C is a digital value of 139. Those values will serve as the cutoffs that will change the color of the LED to indicate that it is too hot or too cold.

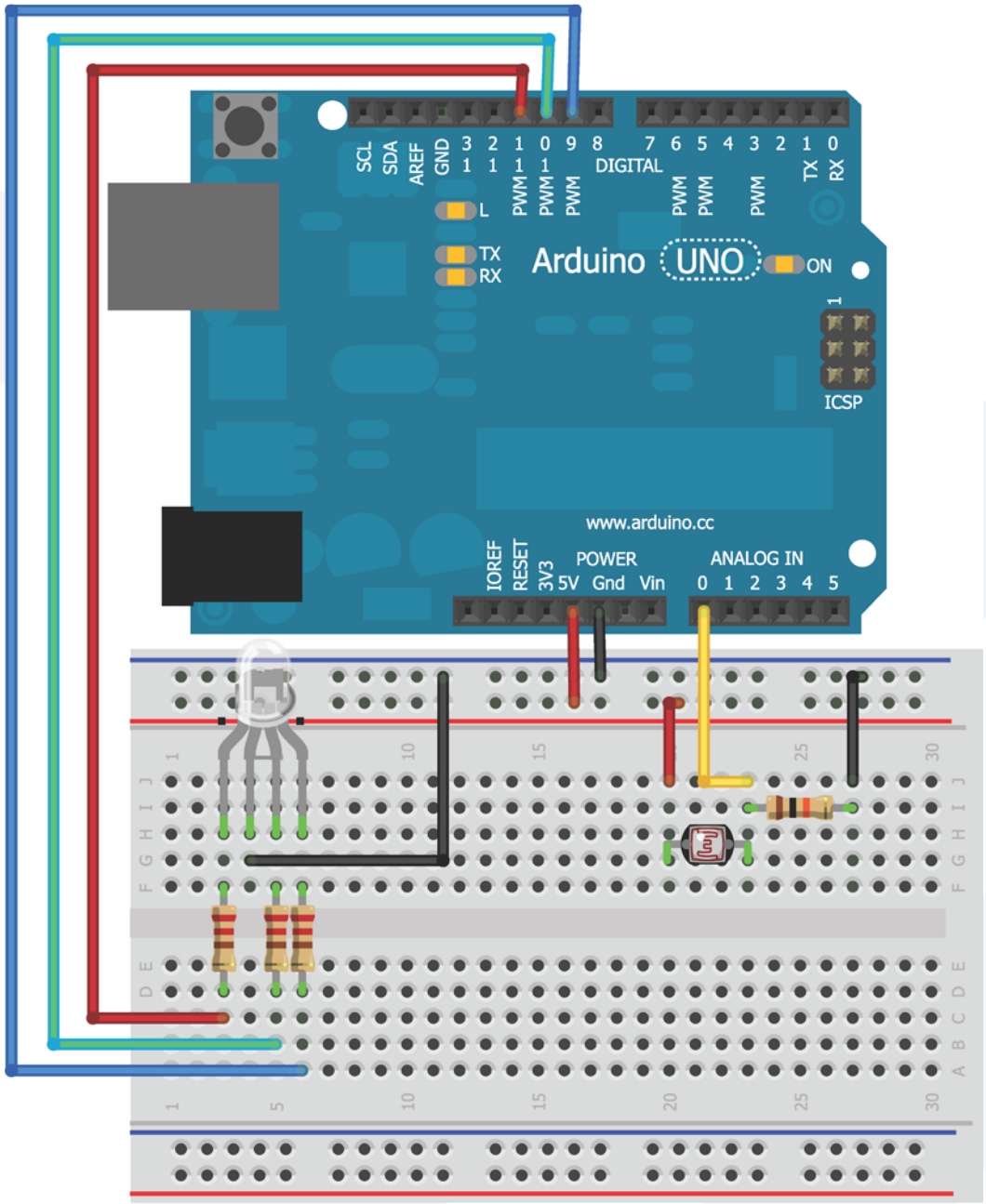
```
//Temperature Alert!
const int BLED=9; //Blue LED on pin 9
const int GLED=10; //Green LED on pin 10
const int RLED=11; //Red LED on pin 11
const int TEMP=0; //Temp Sensor is on pin A0
const int LOWER_BOUND=139; //Lower Threshold
const int UPPER_BOUND=147; //Upper Threshold
int val = 0; //Variable to hold analog reading
void setup()
{
pinMode (BLED, OUTPUT); //Set Blue LED as Output
pinMode (GLED, OUTPUT); //Set Green LED as Output
pinMode (RLED, OUTPUT); //Set Red LED as Output
}
```

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

Photoresistors

- Photoresistors change resistance depending on the amount of light that hits them.
- For a 200kΩ photoresistor.
  - When in complete darkness, its resistance is about 200kΩ.
  - When saturated with light, the resistance drops nearly to zero.

- We can use the data from our photoresistor to make a more intelligent nightlight.
- The nightlight should get brighter as the room gets darker, and vice versa.
- We use the serial monitor sketch, pick the values that represent when our room is at full brightness or complete darkness.
- For example, we 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, the resistor value you are using, and the value of your photoresistor.


SCL
SDA
AREF
GND
DIGITAL
PWM
TX
RX
L
Arduino
UNO
ON
ICSP
www.arduino.cc
IOREF
RESET
3V3
5V
Gnd
Vin
POWER
ANALOG IN

Using Analog Inputs to Control Analog Outputs

- We can use the analogWrite()command to set the brightness of an LED.
- However, it is an 8-bit value; that is, it accepts values between 0 and 255 only, whereas the ADC is returning values as high as 1023.
- The Arduino programming language has two functions that are useful for mapping between two sets of values: the map() and constrain()functions.

map()function

- The map() function looks like this:

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

- value is the information we are starting with. In our case, that's the most recent reading from the analog input.
- fromLow and fromHigh are the input boundaries. These are values we found to correspond to the minimum and maximum brightness in our room. For example, they were 200 and 900.
- toLow and toHigh are the values we want to map them to.
- Because analogWrite() expects value between 0 and 255, we use those values.
- However, we want a darker room to map to a brighter LED. Therefore, when the input from the ADC is a low value, we want the output to the LED to be a high value, and vice versa.

- 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 our fromLow and fromHigh values are 200 and 900, respectively, and our 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.
- 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. Obviously, we don't want that because we can't pass a value greater than 255 to the analogWrite()function.
- We can deal with this by using the constrain() function.
- The constrain() function looks like this: output = constrain(value, min, max)
- If we pass the output from the map function into the constrain function, we 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 Nightlight
const int RLED=9;      //Red LED on pin 9 (PWM)
const int LIGHT=0;     //Light Sensor on analog pin 0
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(RLED, OUTPUT);      //Set 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(RLED, val);         //Control the LED
}
```