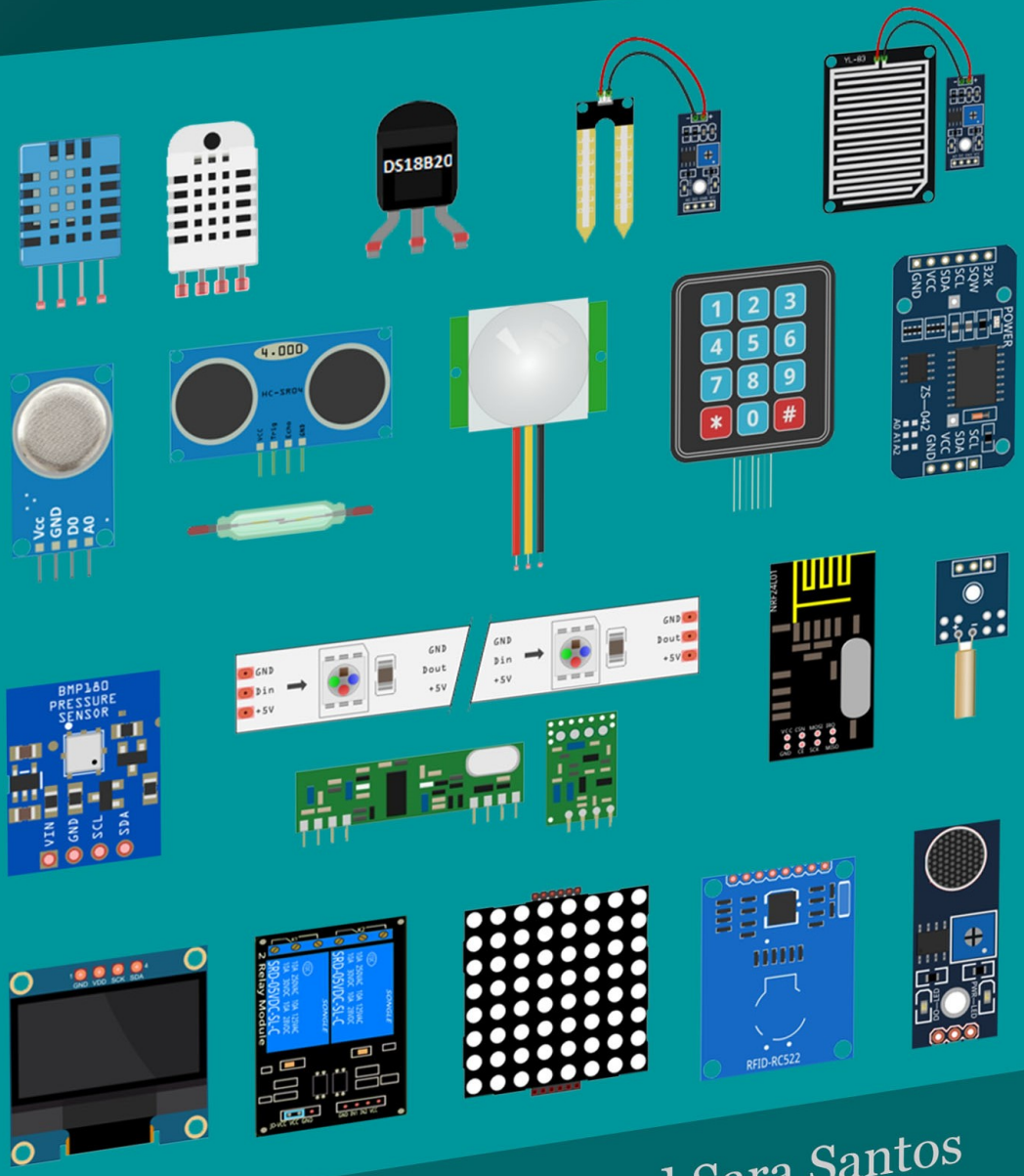
ULTIMATE GUIDE FOR ARDUINO SENSORS/MODULES



Written By Rui Santos and Sara Santos

ULTIMATE GUIDE FOR ARDUINO SENSORS/MODULES

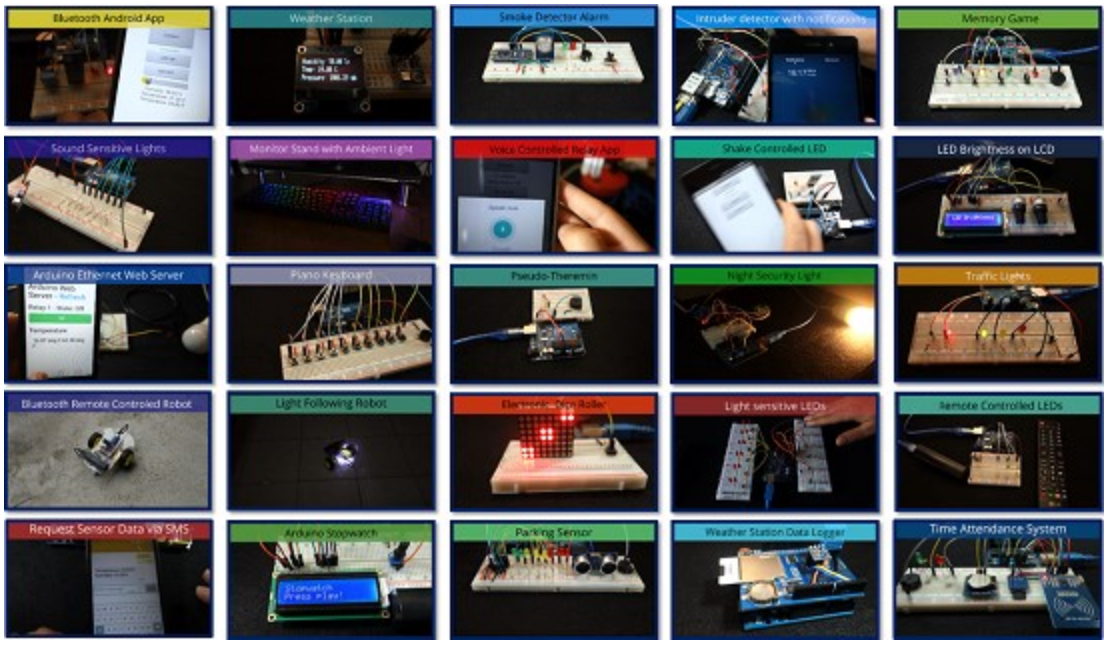
Download other RNT products

Random Nerd Tutorials is an online resource with electronics projects, tutorials and reviews. Creating and posting new projects takes a lot of time. At this moment, Random Nerd Tutorials has nearly 200 free blog posts with complete tutorials using open-source hardware that anyone can read, remix and apply to their own projects: http://randomnerdtutorials.com

To keep free tutorials coming, there's also paid content or as I like to call "Premium Content". To support Random Nerd Tutorials you can download Premium content here. Check all our products below.

Arduino Step-by-step Projects Course

This is a step-by-step course to get you building cool Arduino projects even with no prior experience! This Arduino course is a compilation of 25 projects divided into 5 Modules that you can build by following clear step-by-step instructions with schematics and downloadable code. Click the figure below to check out the course.



Home Automation Using ESP8266 (3rd Edition)

Home Automation Using ESP8266 (3rd Edition) is my step-by-step guide designed to help you get started with this amazing WiFi module called ESP8266. This eBook contains all the information you need to get up to speed quickly and start your own venture with the ESP8266 applied to Home Automation! Read full product description.

Build a Home Automation System for $100

Learn **Raspberry Pi**, **ESP8266**, **Arduino** and **Node-RED**.

This is a premium step-by-step course to get you building a real world home automation system using open-source hardware and software. Read full product description.

Android Apps for Arduino with MIT App Inventor 2

Android Apps for Arduino with MIT App Inventor 2 is a practical course in which you're going to build 8 Android applications to interact with the Arduino.

Read full product description.

Find best price

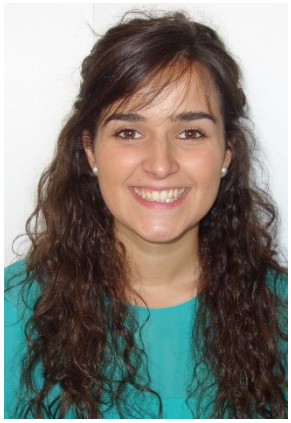
Maker Advisor is part of the Random Nerd Tutorials website. We find the best deals, flash sales and coupons for tools and gear that makers, hobbyists and DIYers like. We share daily deals, write unbiased reviews and compare tools. Visit Maker Advisor to check the latest deals.

MAKER ADVISOR

About the Authors

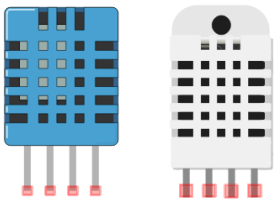
Hi there! Thank you for reading the "Ultimate Guide for Arduino Sensors and Modules".

> *"I'm Rui Santos, I have a master's degree in Electrical and Computer Engineering. I have more than four years of experience teaching electronics and programming with the Random Nerd Tutorials blog. I'm also founder of RNTLab.com and author of BeagleBone For Dummies. Most of my projects and tutorials are related with Arduino, Home Automation, ESP8266, and Raspberry Pi."*



> *"I'm Sara Santos, I have a master's degree in Bioengineering and I've been working with Rui at Random Nerd Tutorials since 2015. My main tasks at Random Nerd tutorials include: recording and editing video, photo shooting, content creation, making projects, proofreading, etc… I also write and build courses with Rui."*

Table of Contents

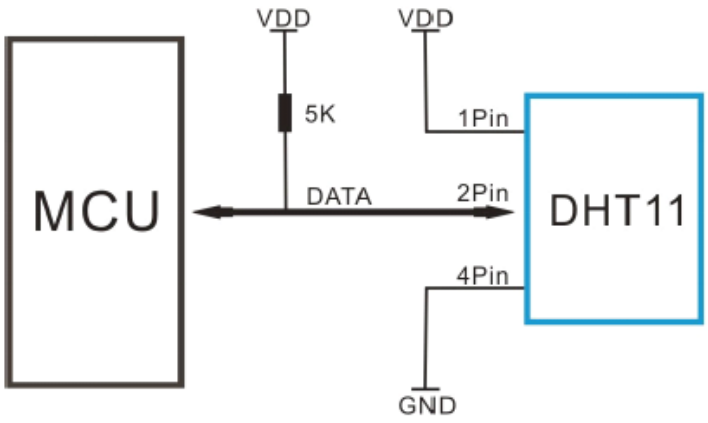


DHT11/DHT22 Temperature and Humidity Sensor

These DHTXX sensors are very popular among the Arduino Tinkerers. The DHT sensors are inexpensive sensors for measuring temperature and humidity.

These sensors contain a chip that does analog to digital conversion and spits out a digital signal with the temperature and humidity.

These signals are easy to read with any microcontroller (MCU).



Specifications DHT11 vs DHT22

There are two versions of the DHT sensor:

DHT11

- Range: 20-90%
- Absolute accuracy: ±5%
- Repeatability: ±1%
- Long term stability: ±1% per year
- Price: $1 to $5

DHT22

- Range: 0-100%
- Absolute accuracy: ±2%
- Repeatability: ±1%
- Long term stability: ±0.5% per year
- Price: $4 to $10

As you can see from the specs above, the DHT22 is a bit more accurate.

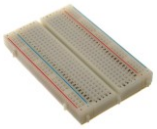
Where to buy?

Click the links below to compare the sensor at different stores and find the best price:

- Click here to see DHT11 on Maker Advisor
- Click here to see DHT22 on Maker Advisor

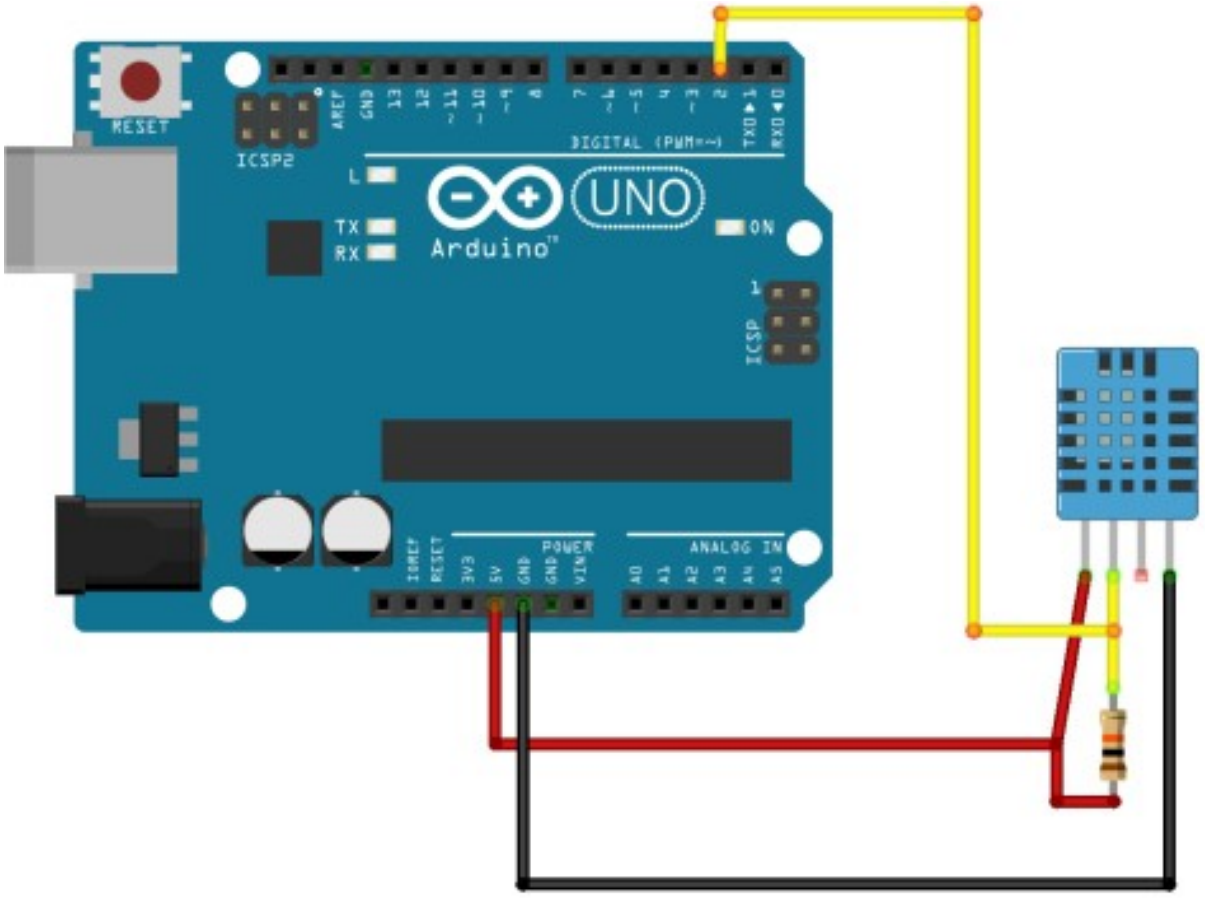
Arduino with DHT11 Sensor

For this example you need the following components:

Figure	Name	Check Price
	Arduino UNO	Find best price on Maker Advisor
	1x DHT11	Find best price on Maker Advisor
	Breadboard	Find best price on Maker Advisor
	10KΩ Resistor (or 4.7KΩ)	Find best price on Maker Advisor
	Jumper Wires	Find best price on Maker Advisor

Schematic

Here's how to connect the DHT11 to an Arduino:



Pin	Wiring to Arduino UNO
1st pin - VCC	5V
2nd pin - Data OUT	Digital pin 2
3rd pin	Don't connect
4th pin - GND	GND

Source code

Below you can find the code you need for this project. But first you need to install the **DHT** library.

1. Download the DHT11 library here
2. Unzip the **DHT** library
3. Rename the extracted folder to **DHT** and remove the "-". Otherwise your Arduino IDE won't recognize your library
4. Install the DHT11 in your Arduino IDE: go to **Sketch ▸ Include Library ▸ Add . ZIP library** and select the library you've just downloaded
5. Restart your Arduino IDE
6. Go to **File ▸ Examples ▸ DHT sensor library ▸ DHTtester**
7. Upload the code

```
// Example testing sketch for various DHT humidity/temperature sensors
// Written by ladyada, public domain

#include "DHT.h"

#define DHTPIN 2     // what digital pin we're connected to

// Uncomment whatever type you're using!
#define DHTTYPE DHT11   // DHT 11
//#define DHTTYPE DHT22   // DHT 22  (AM2302), AM2321
//#define DHTTYPE DHT21   // DHT 21 (AM2301)

// Connect pin 1 (on the left) of the sensor to +5V
// NOTE: If using a board with 3.3V logic like an Arduino Due connect pin 1
// to 3.3V instead of 5V!
// Connect pin 2 of the sensor to whatever your DHTPIN is
// Connect pin 4 (on the right) of the sensor to GROUND
// Connect a 10K resistor from pin 2 (data) to pin 1 (power) of the sensor

// Initialize DHT sensor.
// Note that older versions of this library took an optional third parameter
to
// tweak the timings for faster processors.  This parameter is no longer
needed
// as the current DHT reading algorithm adjusts itself to work on faster
procs.
```

```
DHT dht(DHTPIN, DHTTYPE);

void setup() {
  Serial.begin(9600);
  Serial.println("DHTxx test!");

  dht.begin();
}

void loop() {
  // Wait a few seconds between measurements.
  delay(2000);

  // Reading temperature or humidity takes about 250 milliseconds!
  // Sensor readings may also be up to 2 seconds 'old' (its a very slow
sensor)
  float h = dht.readHumidity();
  // Read temperature as Celsius (the default)
  float t = dht.readTemperature();
  // Read temperature as Fahrenheit (isFahrenheit = true)
  float f = dht.readTemperature(true);

  // Check if any reads failed and exit early (to try again).
  if (isnan(h) || isnan(t) || isnan(f)) {
    Serial.println("Failed to read from DHT sensor!");
    return;
  }

  // Compute heat index in Fahrenheit (the default)
  float hif = dht.computeHeatIndex(f, h);
  // Compute heat index in Celsius (isFahreheit = false)
  float hic = dht.computeHeatIndex(t, h, false);

  Serial.print("Humidity: ");
  Serial.print(h);
  Serial.print(" %\t");
  Serial.print("Temperature: ");
  Serial.print(t);
  Serial.print(" *C ");
  Serial.print(f);
  Serial.print(" *F\t");
```

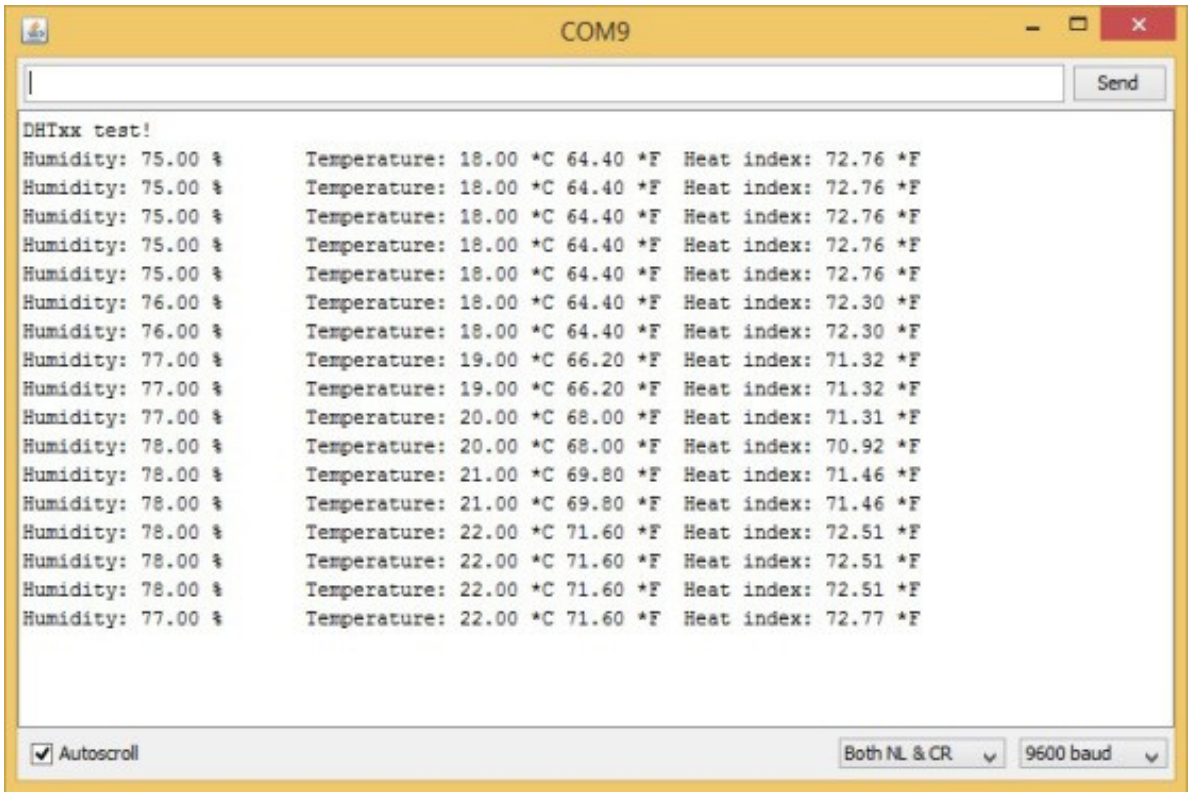
```
  Serial.print("Heat index: ");
  Serial.print(hic);
  Serial.print(" *C ");
  Serial.print(hif);
  Serial.println(" *F");
}
```

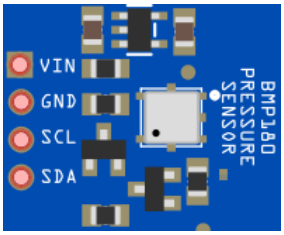
{;} SOURCE CODE

https://github.com/RuiSantosdotme/Random-Nerd-Tutorials/blob/master/Projects/DHTtester.ino

Demonstration

In this project the Arduino is measuring the temperature and humidity. Those two measurements are being displayed on the serial monitor. Here's what you should see in your Arduino IDE serial monitor.





BMP180 Barometric Sensor

The BMP180 barometric sensor (model GY-68) is the one in the following figure (front and back view). It is a very small module with 1mm x 1.1mm (0.039in x 0.043in).

Front view

Back view

It measures the absolute pressure of the air around it. It has a measuring range from 300hPa to 1100hPa with an accuracy down to 0.02hPa. It can also measure altitude and temperature.

The BMP180 barometric sensor communicates via I2C interface. This means that it communicates with the Arduino using just 2 pins.

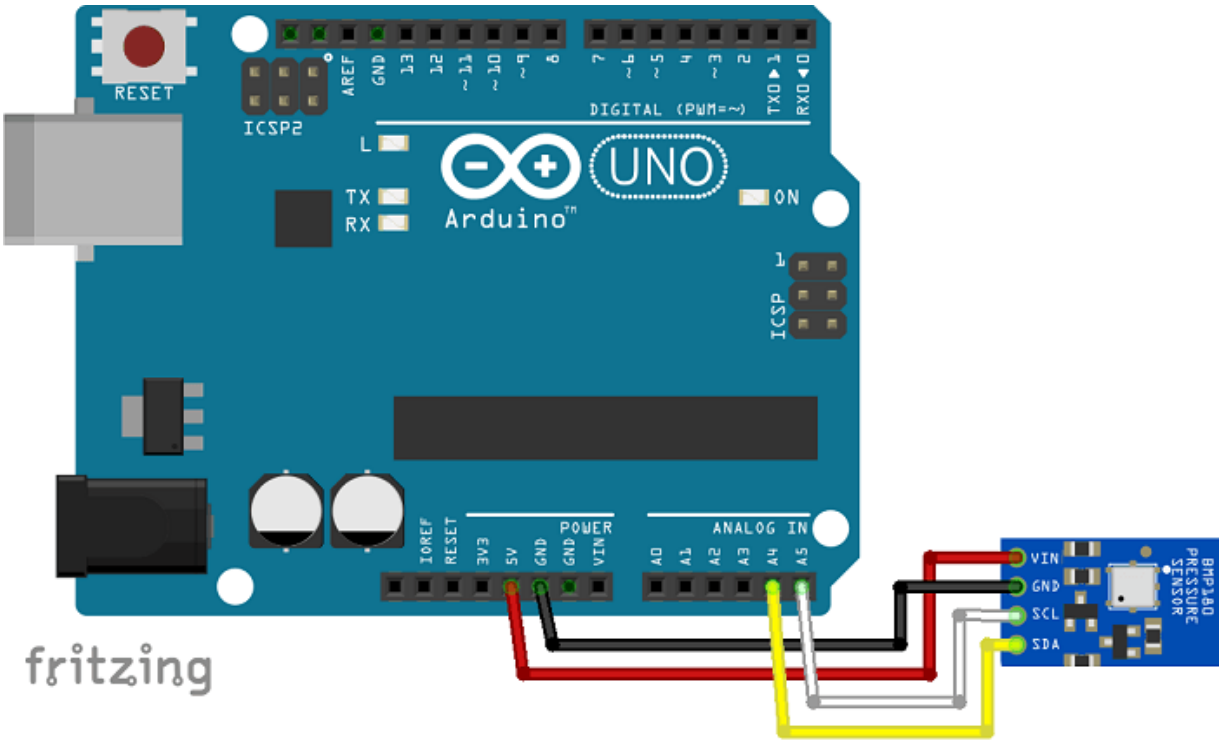
Where to buy?

Click the link below to compare the sensor at different stores and find the best price:

- BMP180 barometric sensor

Schematic

Wire your sensor to the Arduino as shown in the schematic diagram below.



Wiring the sensor to the Arduino UNO is pretty straightforward:

Pin	Wiring to Arduino Uno
Vin	5V
GND	GND
SCL	A5
SDA	A4

Code

To control the BMP180 barometric sensor, you need to install the **SFE_BMP180** Library.

Installing the SFE_BMP180 Library

1. Click here to download the SFE_BMP180 library. You should have a .zip folder in your Downloads folder

2. Unzip the .zip folder and you should get BMP180_Breakout_Arduino_Library-master folder
3. Rename your folder from ~~BMP180_Breakout_Arduino_Library-master~~ to BMP180_Breakout_Arduino_Library
4. Move the BMP180_Breakout_Arduino_Library folder to your Arduino IDE installation libraries folder
5. Finally, re-open your Arduino IDE

Uploading the code

Go to **File ▸ Examples ▸SparkfunBMP180 ▸ SFE_BMP180_example**.This example is very well commented and explained on how the sensor reads the pressure, temperature and computes the altitude. Don't upload the code now, you need to set the altitude first.

```
/* SFE_BMP180 library example sketch

This sketch shows how to use the SFE_BMP180 library to read the
Bosch BMP180 barometric pressure sensor.
https://www.sparkfun.com/products/11824

Like most pressure sensors, the BMP180 measures absolute pressure.
This is the actual ambient pressure seen by the device, which will
vary with both altitude and weather.

Before taking a pressure reading you must take a temparture reading.
This is done with startTemperature() and getTemperature().
The result is in degrees C.

Once you have a temperature reading, you can take a pressure reading.
This is done with startPressure() and getPressure().
The result is in millibar (mb) aka hectopascals (hPa).

If you'll be monitoring weather patterns, you will probably want to
remove the effects of altitude. This will produce readings that can
be compared to the published pressure readings from other locations.
To do this, use the sealevel() function. You will need to provide
the known altitude at which the pressure was measured.

If you want to measure altitude, you will need to know the pressure
at a baseline altitude. This can be average sealevel pressure, or
```

```
a previous pressure reading at your altitude, in which case
subsequent altitude readings will be + or - the initial baseline.
This is done with the altitude() function.

Hardware connections:

- (GND) to GND
+ (VDD) to 3.3V

(WARNING: do not connect + to 5V or the sensor will be damaged!)

You will also need to connect the I2C pins (SCL and SDA) to your
Arduino. The pins are different on different Arduinos:

Any Arduino pins labeled:  SDA  SCL
Uno, Redboard, Pro:        A4   A5
Mega2560, Due:             20   21
Leonardo:                   2    3

Leave the IO (VDDIO) pin unconnected. This pin is for connecting
the BMP180 to systems with lower logic levels such as 1.8V

Have fun! -Your friends at SparkFun.

The SFE_BMP180 library uses floating-point equations developed by the
Weather Station Data Logger project: http://wmrx00.sourceforge.net/

Our example code uses the "beerware" license. You can do anything
you like with this code. No really, anything. If you find it useful,
buy me a beer someday.

V10 Mike Grusin, SparkFun Electronics 10/24/2013
V1.1.2 Updates for Arduino 1.6.4 5/2015
*/

// Your sketch must #include this library, and the Wire library.
// (Wire is a standard library included with Arduino.):

#include <SFE_BMP180.h>
#include <Wire.h>
```

```
// You will need to create an SFE_BMP180 object, here called "pressure":

SFE_BMP180 pressure;

#define ALTITUDE 1655.0 // Altitude of SparkFun's HQ in Boulder, CO. in
meters

void setup()
{
  Serial.begin(9600);
  Serial.println("REBOOT");

  // Initialize the sensor (it is important to get calibration values stored
on the device).

  if (pressure.begin())
    Serial.println("BMP180 init success");
  else
  {
    // Oops, something went wrong, this is usually a connection problem,
    // see the comments at the top of this sketch for the proper connections.

    Serial.println("BMP180 init fail\n\n");
    while(1); // Pause forever.
  }
}

void loop()
{
  char status;
  double T,P,p0,a;

  // Loop here getting pressure readings every 10 seconds.

  // If you want sea-level-compensated pressure, as used in weather reports,
  // you will need to know the altitude at which your measurements are taken.
  // We're using a constant called ALTITUDE in this sketch:

  Serial.println();
  Serial.print("provided altitude: ");
  Serial.print(ALTITUDE,0);
```

```
  Serial.print(" meters, ");
  Serial.print(ALTITUDE*3.28084,0);
  Serial.println(" feet");

  // If you want to measure altitude, and not pressure, you will instead need
  // to provide a known baseline pressure. This is shown at the end of the
sketch.

  // You must first get a temperature measurement to perform a pressure
reading.

  // Start a temperature measurement:
  // If request is successful, the number of ms to wait is returned.
  // If request is unsuccessful, 0 is returned.

  status = pressure.startTemperature();
  if (status != 0)
  {
    // Wait for the measurement to complete:
    delay(status);

    // Retrieve the completed temperature measurement:
    // Note that the measurement is stored in the variable T.
    // Function returns 1 if successful, 0 if failure.

    status = pressure.getTemperature(T);
    if (status != 0)
    {
      // Print out the measurement:
      Serial.print("temperature: ");
      Serial.print(T,2);
      Serial.print(" deg C, ");
      Serial.print((9.0/5.0)*T+32.0,2);
      Serial.println(" deg F");

      // Start a pressure measurement:
      // The parameter is the oversampling setting, from 0 to 3 (highest res,
longest wait).
      // If request is successful, the number of ms to wait is returned.
      // If request is unsuccessful, 0 is returned.
```

```
      status = pressure.startPressure(3);
      if (status != 0)
      {
        // Wait for the measurement to complete:
        delay(status);

        // Retrieve the completed pressure measurement:
        // Note that the measurement is stored in the variable P.
        // Note also that the function requires the previous temperature
measurement (T).
        // (If temperature is stable, you can do one temperature measurement
for a number of pressure measurements.)
        // Function returns 1 if successful, 0 if failure.

        status = pressure.getPressure(P,T);
        if (status != 0)
        {
          // Print out the measurement:
          Serial.print("absolute pressure: ");
          Serial.print(P,2);
          Serial.print(" mb, ");
          Serial.print(P*0.0295333727,2);
          Serial.println(" inHg");

          // The pressure sensor returns abolute pressure, which varies with
altitude.
          // To remove the effects of altitude, use the sealevel function and
your current altitude.
          // This number is commonly used in weather reports.
          // Parameters: P = absolute pressure in mb, ALTITUDE = current
altitude in m.
          // Result: p0 = sea-level compensated pressure in mb

          p0 = pressure.sealevel(P,ALTITUDE); // we're at 1655 meters
(Boulder, CO)
          Serial.print("relative (sea-level) pressure: ");
          Serial.print(p0,2);
          Serial.print(" mb, ");
          Serial.print(p0*0.0295333727,2);
          Serial.println(" inHg");
```

```
          // On the other hand, if you want to determine your altitude from
the pressure reading,
          // use the altitude function along with a baseline pressure (sea-
level or other).
          // Parameters: P = absolute pressure in mb, p0 = baseline pressure
in mb.
          // Result: a = altitude in m.

          a = pressure.altitude(P,p0);
          Serial.print("computed altitude: ");
          Serial.print(a,0);
          Serial.print(" meters, ");
          Serial.print(a*3.28084,0);
          Serial.println(" feet");
        }
        else Serial.println("error retrieving pressure measurement\n");
      }
      else Serial.println("error starting pressure measurement\n");
    }
    else Serial.println("error retrieving temperature measurement\n");
  }
  else Serial.println("error starting temperature measurement\n");
  delay(5000);  // Pause for 5 seconds.
}
```

SOURCE CODE

https://github.com/RuiSantosdotme/Random-Nerd-Tutorials/blob/master/Projects/SFE_BMP180_example.ino

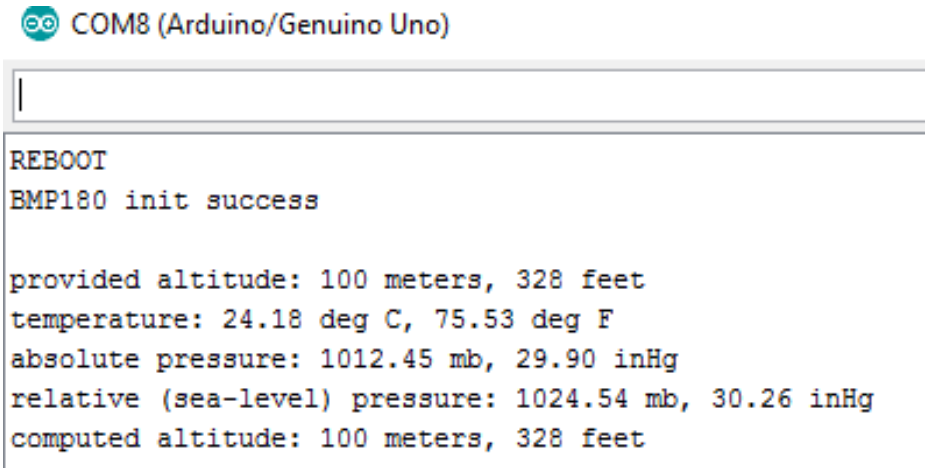
Set the altitude

Before uploading the code, you need to set up your current altitude. Go to elevationmap.net, insert your address and check your altitude's location. Set your altitude in the code. The place where you should write your altitude is commented.

```
#define ALTITUDE 1655.0 // Altitude of SparkFun's HQ in Boulder, CO. in meters
```

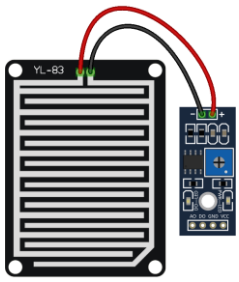
Demonstration

After uploading the code, open your serial monitor at a baud rate of 9600. You'll see your sensor readings.



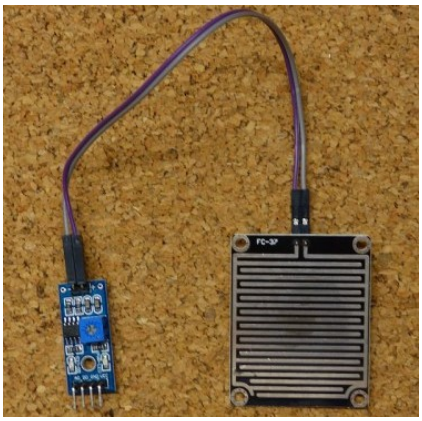
Wrapping up

The BMP180 is an interesting sensor to be used in your own weather station. Because the pressure changes with the altitude, this sensor can also compute the altitude.

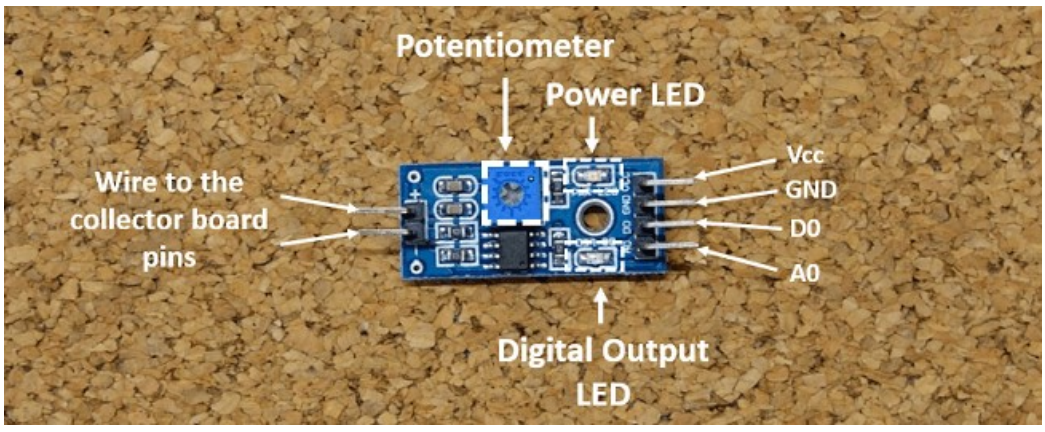


FC-37 or YL-83 Rain Sensor

The rain sensor is used to detect water and it can detect beyond what a humidity sensor can. The FC-37 rain sensor (or other versions like YL-83) is set up by two pieces: the electronic board (at the left) and the collector board (at the right) that collects the water drops, as you can see in the following figure:



The rain sensor has a built-in potentiometer for sensitivity adjustment of the digital output (D0). It also has a power LED that lights up when the sensor is turned on and a digital output LED.

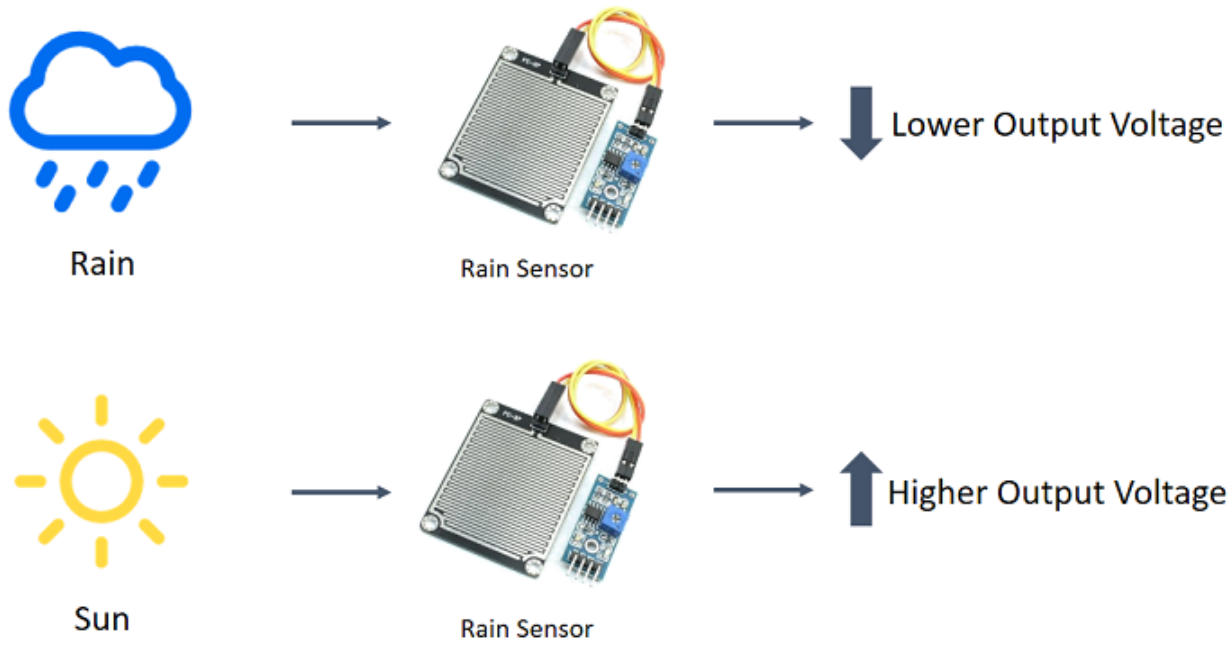


How does it work?

In simple terms, the resistance of the collector board varies accordingly to the amount of water on its surface.

When the board is:

- **Wet:** the resistance increases, and the output voltage decreases
- **Dry:** the resistance is lower, and the output voltage is higher



Where to buy?

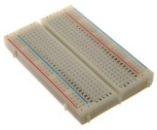
Click the link below to compare the sensor at different stores and find the best price:

- YL-83 rain sensor

Example: Rain Sensor with Arduino

This is a simple example to show you how you can use the rain sensor in your projects with Arduino. You will read the analog sensor values and print them in the Arduino IDE serial monitor.

For this example, you'll need the following parts:

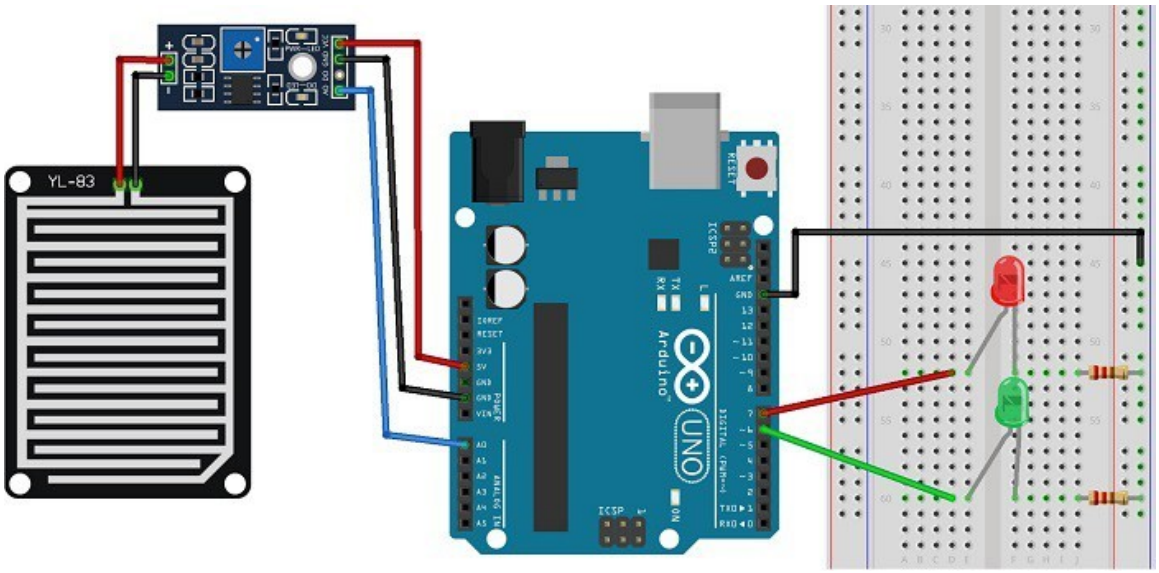
Figure	Name	Check Price
	Arduino UNO	Find best price on Maker Advisor
	YL-83 Rain Sensor	Find best price on Maker Advisor
	Breadboard	Find best price on Maker Advisor
	2× 220Ω resistor	Find best price on Maker Advisor
	2× LEDs	Find best price on Maker Advisor
	Jumper Wires	Find best price on Maker Advisor

Pin Wiring

Sensor Pin	Wiring to Arduino Uno
A0	Any analog pin (A5 in this example)
D0	Digital pins
GND	GND
VCC	5V

Schematic

Follow the next schematic diagram to complete the project:



Code

Upload the following sketch to your Arduino board (feel free to adjust the variable **`thresholdValue`** with a different threshold value):

```
/*

 All the resources for this project:
 http://randomnerdtutorials.com/

*/

int rainPin = A0;
int greenLED = 6;
int redLED = 7;
// you can adjust the threshold value
int thresholdValue = 500;

void setup(){
  pinMode(rainPin, INPUT);
  pinMode(greenLED, OUTPUT);
  pinMode(redLED, OUTPUT);
```

```
  digitalWrite(greenLED, LOW);
  digitalWrite(redLED, LOW);
  Serial.begin(9600);
}

void loop() {
  // read the input on analog pin 0:
  int sensorValue = analogRead(rainPin);
  Serial.print(sensorValue);
  if(sensorValue < thresholdValue){
    Serial.println(" - It's wet");
    digitalWrite(greenLED, LOW);
    digitalWrite(redLED, HIGH);
  }
  else {
    Serial.println(" - It's dry");
    digitalWrite(greenLED, HIGH);
    digitalWrite(redLED, LOW);
  }
  delay(500);
}
```

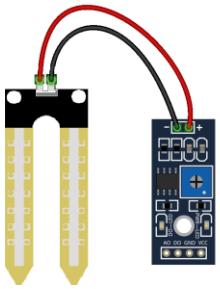
{;} SOURCE CODE

https://github.com/RuiSantosdotme/Random-Nerd-Tutorials/blob/master/Projects/rain_sensor_arduino.ino

Open the Arduino IDE serial monitor to see the results. Then, add drops of water to the collector board and see how that changes the readings. When the value goes below a certain threshold, a red LED will light up, and when the value goes above a certain threshold, a green LED will light up.

Wrapping up

If you want to know when it's raining, you need to set up your rain sensor with the Arduino outside. Be aware that you should protect your Arduino and your circuit from water. A waterproof project box can be pretty handy in this situation.

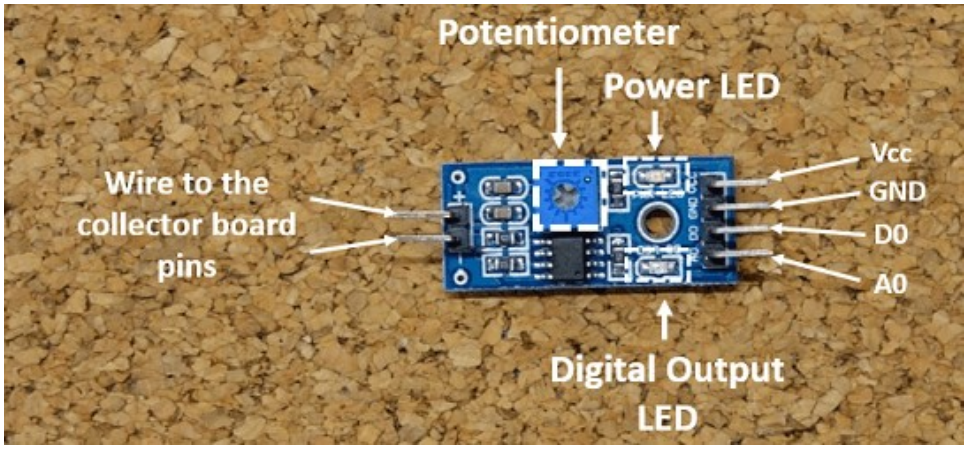


YL-69 or HL-69 Soil Moisture Sensor

The soil moisture sensor or the hygrometer is usually used to detect the humidity of the soil. So, it is perfect to build an automatic watering system or to monitor the soil moisture of your plants.

The sensor is set up by two pieces: the electronic board (at the right), and the probe with two pads, that detects the water content (at the left).

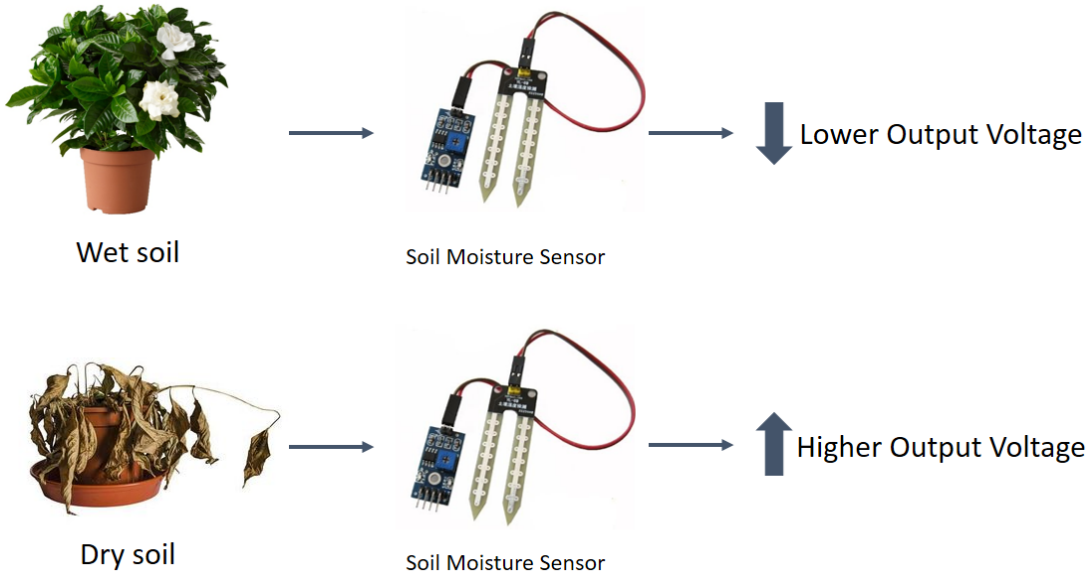
The sensor has a built-in potentiometer for sensitivity adjustment of the digital output (D0), a power LED and a digital output LED, as you can see in the following figure.



How does it work?

The voltage that the sensor outputs changes accordingly to the water content in the soil. When the soil is:

- **Wet**: the output voltage decreases
- **Dry**: the output voltage increases



The output can be a **digital signa**l (D0) LOW or HIGH, depending on the water content. If the soil humidity exceeds a certain predefined threshold value, the modules outputs LOW, otherwise it outputs HIGH. The threshold value for the digital signal can be adjusted using the potentiometer. The output can also be an **analog signal** and so you'll get a value between 0 and 1023.

Where to buy?

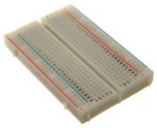
Click the link below to compare the sensor at different stores and find the best price:

- HL-69 soil moisture sensor

Soil Moisture Sensor with the Arduino

In this example, you'll read the analog sensor output values using the Arduino and print those readings in the Arduino IDE serial monitor.

For this example, you'll need the following components:

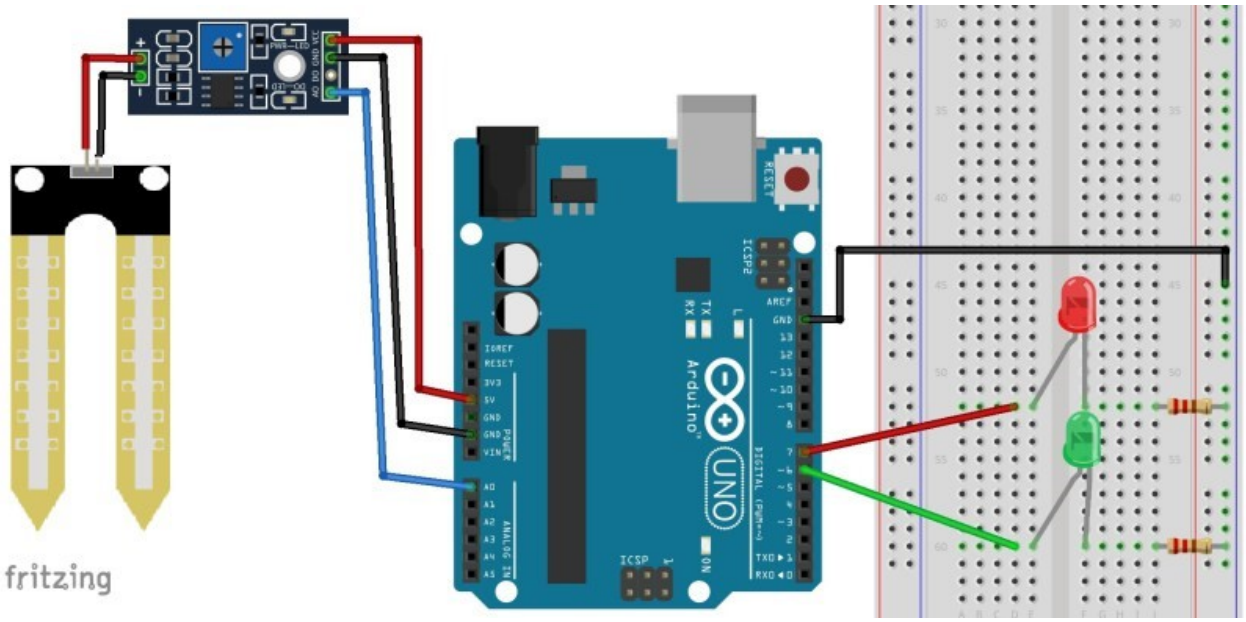
Figure	Name	Check Price
	Arduino UNO	Find best price on Maker Advisor
	HL-69 Rain Sensor	Find best price on Maker Advisor
	Breadboard	Find best price on Maker Advisor
	2× 220Ω Resistor	Find best price on Maker Advisor
	2× LEDs	Find best price on Maker Advisor
	Jumper Wires	Find best price on Maker Advisor

Pin Wiring

Sensor Pin	Wiring to Arduino Uno
A0	Any analog pin (A5 in this example)
D0	Digital pins
GND	GND
VCC	5V

Schematic

To complete the project, follow the next schematic diagram:



Code

Upload the following sketch to your Arduino board:

```
/* All the resources for this project:
 http://randomnerdtutorials.com/
 */

int rainPin = A0;
int greenLED = 6;
int redLED = 7;
// you can adjust the threshold value
int thresholdValue = 800;

void setup(){
  pinMode(rainPin, INPUT);
  pinMode(greenLED, OUTPUT);
  pinMode(redLED, OUTPUT);
  digitalWrite(greenLED, LOW);
  digitalWrite(redLED, LOW);
  Serial.begin(9600);
}
```

```
void loop() {
  // read the input on analog pin 0:
  int sensorValue = analogRead(rainPin);
  Serial.print(sensorValue);
  if(sensorValue < thresholdValue){
    Serial.println(" - Doesn't need watering");
    digitalWrite(redLED, LOW);
    digitalWrite(greenLED, HIGH);
  }
  else {
    Serial.println(" - Time to water your plant");
    digitalWrite(redLED, HIGH);
    digitalWrite(greenLED, LOW);
  }
  delay(500);
}
```

{;} SOURCE CODE

https://github.com/RuiSantosdotme/Random-Nerd-Tutorials/blob/master/Projects/soil_moisture_sensor_arduino.ino

Open the Arduino IDE serial monitor to see the values. Then, try your sensor in a wet and in a dry soil and see what happens.

When the analog value goes above a certain threshold, a red LED will turn on (indicates that the plant needs watering), and when the value goes below a certain threshold, a green LED will turn on (indicates that the plant is ok).

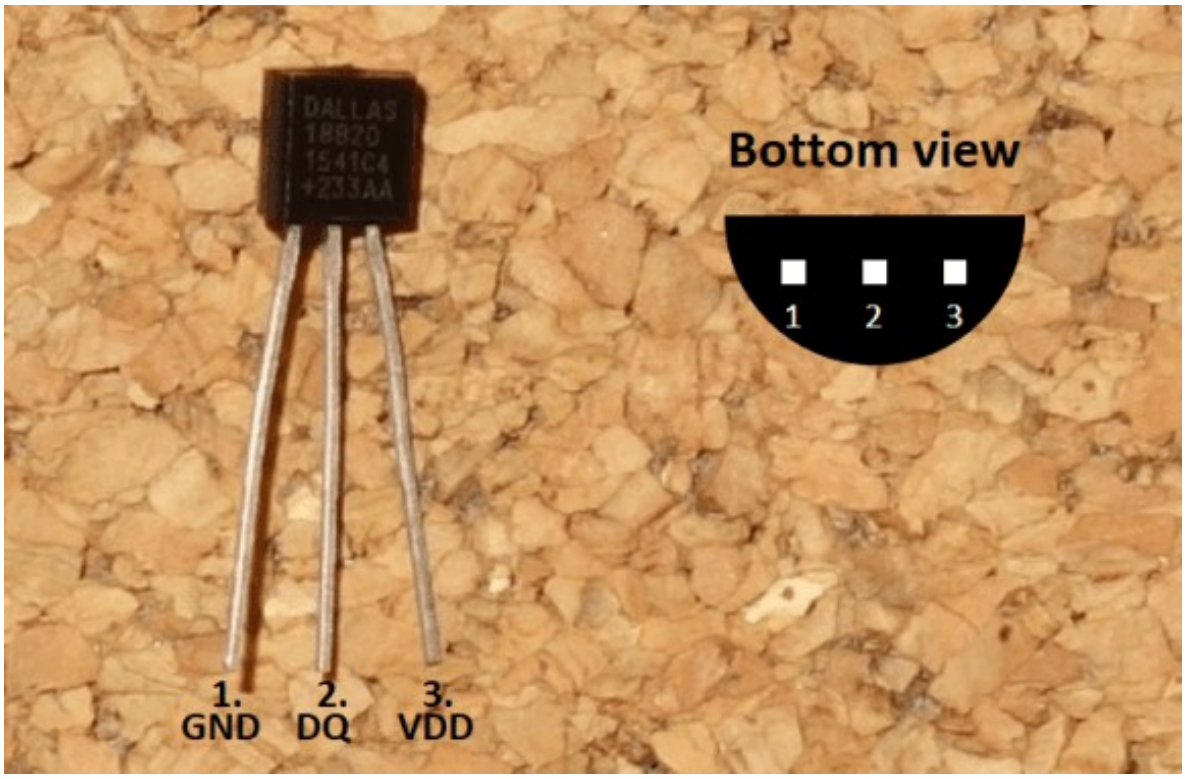
Wrapping up

The moisture sensor allows to monitor the water content in the soil. This is useful if you want to build an automatic watering system. You can also use it to just monitor your plants soil moisture.

DS18B20 Temperature Sensor

The DS18B20 temperature sensor is a 1-wire digital temperature sensor. This means that you can read the temperature with a very simple circuit setup. It communicates on common bus, which means that you can connect several devices and read their values using just one digital pin of the Arduino.

The sensor has just three pins as you can see in the following figure:



The DS18B20 is also available in waterproof version:

Features

Here's some main features of the DS18B20 temperature sensor:

- Communicates over 1-wire bus communication
- Operating range temperature: -55ºC to 125ºC
- Accuracy +/-0.5 ºC (between the range -10ºC to 85ºC)

Where to buy?

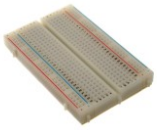
Click the link below to compare the sensor at different stores and find the best price:

- DS18B20 digital temperature sensor

DS18B20 temperature sensor with the Arduino

In this example, you'll read the temperature using the DS18B20 sensor and the Arduino. The readings will be displayed on the Arduino Serial Monitor.

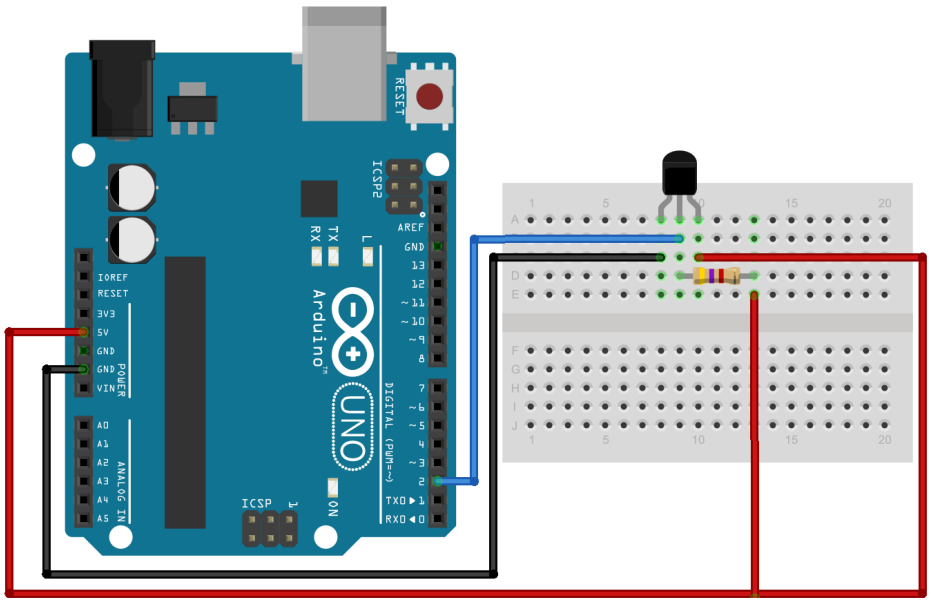
For this example, you'll need the following parts:

Figure	Name	Check Price
	Arduino UNO	Find best price on Maker Advisor
	DS18B20 temperature sensor	Find best price on Maker Advisor
	Breadboard	Find best price on Maker Advisor
	4.7kΩ Resistor	Find best price on Maker Advisor
	Jumper Wires	Find best price on Maker Advisor

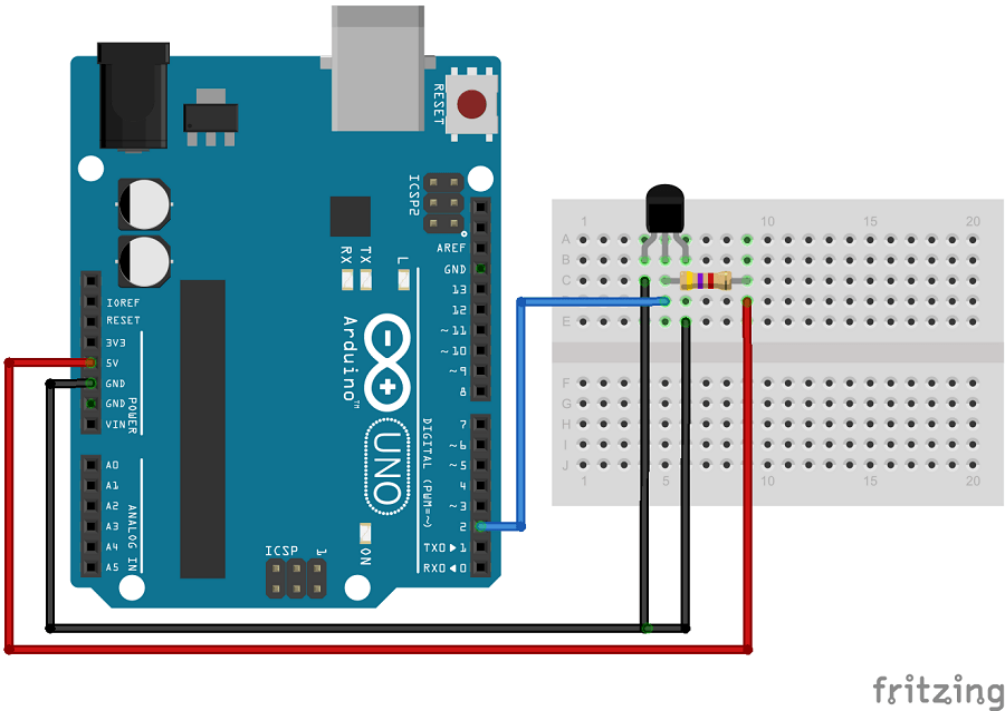
Schematic

The sensor can operate in two different modes:

- **Normal mode**: 3-wire connection is needed. Here's the schematic you need to follow:



- **Parasite mode**: only 2 wires required, the data and ground. The sensor derives its power from the data line. In this case, here's the schematic you need to follow:



You can read the temperature of more than one sensor at the same time using just one digital Arduino pin. For that, you just need to connect together all the DQ pins to any digital Arduino pin.

Code

You'll need to install the OneWire Library and DallasTemperature Library.

Installing the OneWire Library

1. Click here to download the OneWire library. You should have a .zip folder in your Downloads
2. Unzip the .zip folder and you should get OneWire-master folder
3. Rename your folder from ~~OneWire-master~~ to OneWire
4. Move the OneWire folder to your Arduino IDE installation libraries folder
5. Finally, re-open your Arduino IDE

Installing the DallasTemperature Library

1. Click here to download the DallasTemperature library.You should have a .zip folder in your Downloads

2. Unzip the .zip folder and you should get Arduino-Temperature-Control-Library-master folder
3. Rename your folder ~~from Arduino-Temperature-Control-Library-master~~ to DallasTemperature
4. Move the DallasTemperature folder to your Arduino IDE installation libraries folder
5. Finally, re-open your Arduino IDE

After installing the needed libraries, upload the following code to your Arduino board.

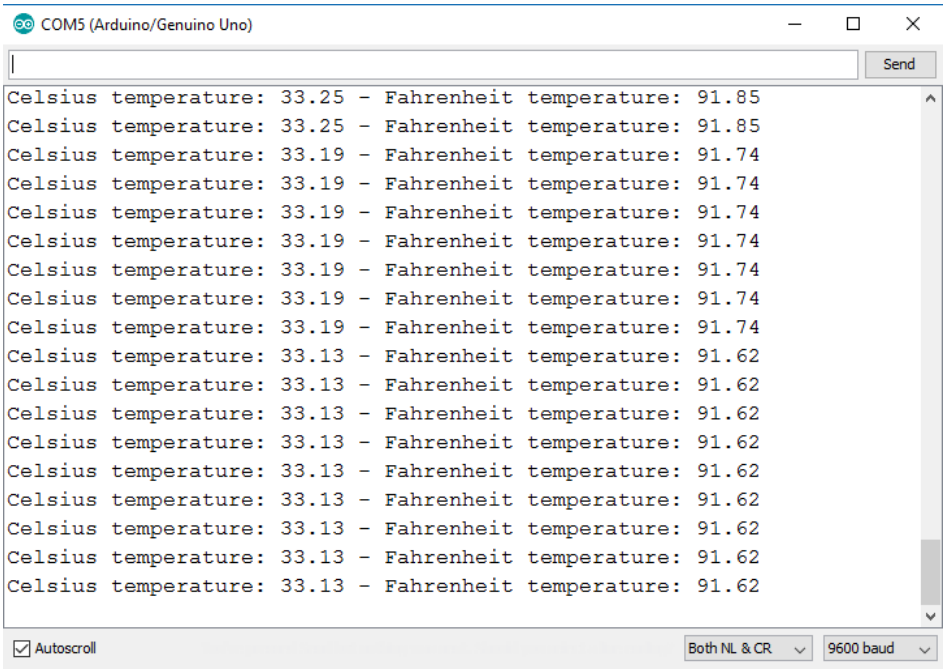
```
/*  Rui Santos
  Complete project details at http://randomnerdtutorials.com
  Based on the Dallas Temperature Library example
*/

#include <OneWire.h>
#include <DallasTemperature.h>

// Data wire is connected to the Arduino digital pin 2
#define ONE_WIRE_BUS 2

// Setup a oneWire instance to communicate with any OneWire devices
OneWire oneWire(ONE_WIRE_BUS);

// Pass our oneWire reference to Dallas Temperature sensor
DallasTemperature sensors(&oneWire);

void setup(void)
{
  // Start serial communication for debugging purposes
  Serial.begin(9600);
  // Start up the library
  sensors.begin();
}

void loop(void){
  // Call sensors.requestTemperatures() to issue a global temperature and
Requests to all devices on the bus
  sensors.requestTemperatures();

  Serial.print("Celsius temperature: ");
```

```
  // Why "byIndex"? You can have more than one IC on the same bus. 0 refers
to the first IC on the wire
  Serial.print(sensors.getTempCByIndex(0));
  Serial.print(" - Fahrenheit temperature: ");
  Serial.println(sensors.getTempFByIndex(0));
  delay(1000);
}
```

{;} SOURCE CODE

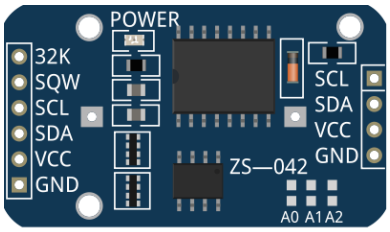
https://github.com/RuiSantosdotme/Random-Nerd-Tutorials/blob/master/Projects/DS18B20_Temperature_Sensor_Arduino.ino

Finally, you should open the Arduino IDE serial monitor at a 9600 baud rate and you'll see the temperature displayed in both Celsius and Fahrenheit:



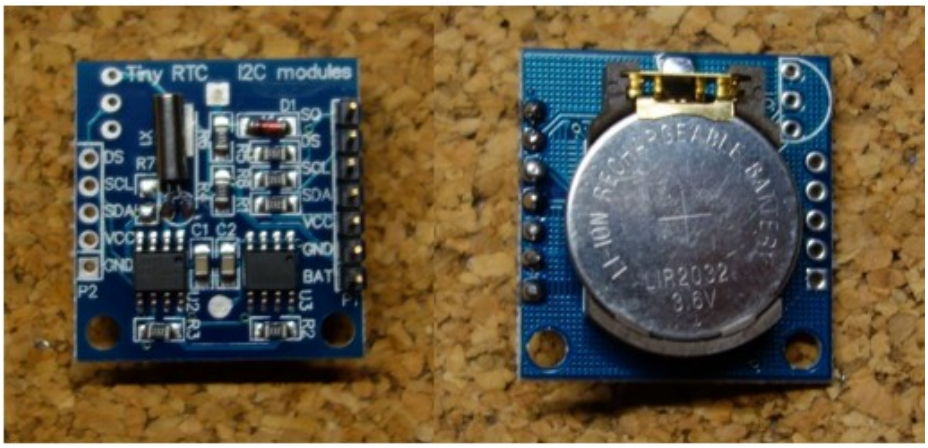
Wrapping up

The DS18B20 temperature sensor is a very useful one. It allows you to read the temperature both in Celsius and Fahrenheit via One Wire communication. This means you can have readings from several sensors without multiplying the connections to the Arduino.



DS1307 or DS3231 Real Time Clock (RTC)

The real time clock module is the one in the figure below (front and back view).



When you first use this module, you need to solder some header pins.

As you can see in the picture above, the module has a backup battery installed. This allows the module to retain the time, even when it's not being powered up by the Arduino. This way, every time you turn on and off your module, the time doesn't reset.

This module uses I2C communication. This means that it communicates with the Arduino using just 2 pins.

Where to buy?

Click the link below to compare the module at different stores and find the best price:

- DS1307 Real Time Clock Module

Pin wiring

Module Pin	Wiring to Arduino Uno
SCL	A5
SDA	A4
VCC	5V
GND	GND

If you're using other Arduino board rather than the Uno, check out what are their SCL and SDA pins.

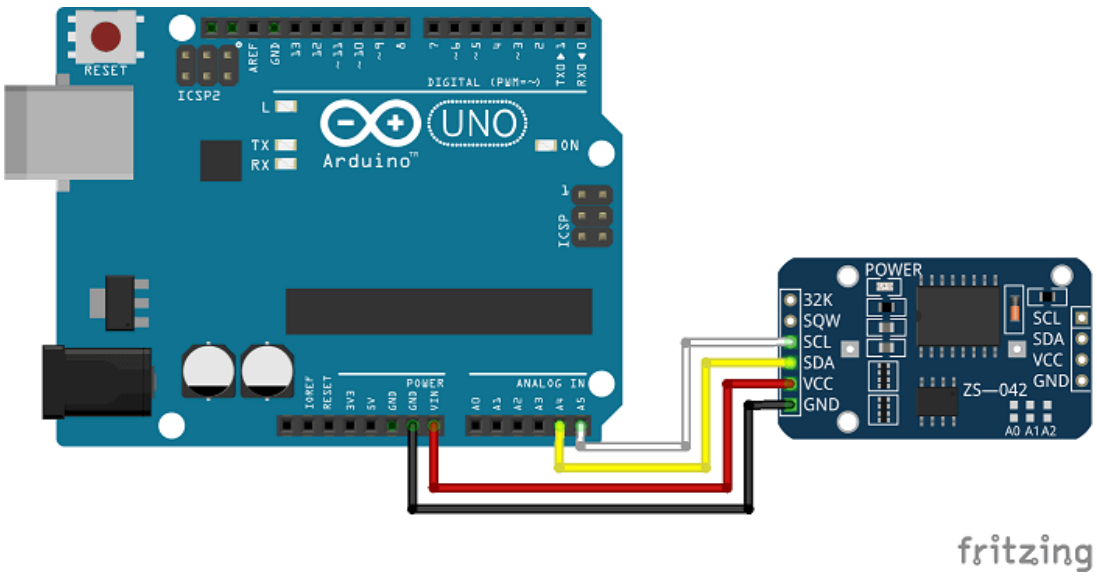
- **Nano**: SDA (A4); SCL(A5)
- **MEGA**: SDA (20); SCL(21)
- **Leonardo**: SDA (20); SCL(21)

Displaying date and time in the serial monitor

In this example you'll display the data and time in the serial monitor.

Schematics

Connect your Real Time Clock module to your Arduino as in the schematics below.



Code

Working with the RTC requires two important steps:

- **setting the current time**, so that the RTC knows what time is it
- **retaining the time**, so that the RTC always gives the correct time, even when it is turned off

Set the current time in the Real Time Clock

For setting the current time you need to change the code provided. Set your current time in the function `setDS3231time()`

```
// set the initial time here:
// DS3231 seconds, minutes, hours, day, date, month, year
setDS3231time(33,53,16,1,25,9,16);
```

The parameters for the function are highlighted in red: seconds, minutes, hours, day of the week, date, month and year (in this order). Sunday is the day 1 of the week and Saturday is 7. Don't forget to uncomment that line of code.

After setting the current time, you can upload the provided code with the required modifications.

The code provided was written by John Boxall from tronixstuff. You can read his tutorial here.

```
// Written by John Boxall from http://tronixstuff.com

#include "Wire.h"
#define DS3231_I2C_ADDRESS 0x68
// Convert normal decimal numbers to binary coded decimal
byte decToBcd(byte val){
  return( (val/10*16) + (val%10) );
}
// Convert binary coded decimal to normal decimal numbers
byte bcdToDec(byte val){
  return( (val/16*10) + (val%16) );
}
void setup(){
  Wire.begin();
  Serial.begin(9600);
```

```
  // set the initial time here:
  // DS3231 seconds, minutes, hours, day, date, month, year
  setDS3231time(30,42,16,5,13,10,16);
}
void setDS3231time(byte second, byte minute, byte hour, byte dayOfWeek, byte
dayOfMonth, byte month, byte year){
  // sets time and date data to DS3231
  Wire.beginTransmission(DS3231_I2C_ADDRESS);
  Wire.write(0); // set next input to start at the seconds register
  Wire.write(decToBcd(second)); // set seconds
  Wire.write(decToBcd(minute)); // set minutes
  Wire.write(decToBcd(hour)); // set hours
  Wire.write(decToBcd(dayOfWeek)); // set day of week (1=Sunday, 7=Saturday)
  Wire.write(decToBcd(dayOfMonth)); // set date (1 to 31)
  Wire.write(decToBcd(month)); // set month
  Wire.write(decToBcd(year)); // set year (0 to 99)
  Wire.endTransmission();
}
void readDS3231time(byte *second,
byte *minute,
byte *hour,
byte *dayOfWeek,
byte *dayOfMonth,
byte *month,
byte *year){
  Wire.beginTransmission(DS3231_I2C_ADDRESS);
  Wire.write(0); // set DS3231 register pointer to 00h
  Wire.endTransmission();
  Wire.requestFrom(DS3231_I2C_ADDRESS, 7);
  // request seven bytes of data from DS3231 starting from register 00h
  *second = bcdToDec(Wire.read() & 0x7f);
  *minute = bcdToDec(Wire.read());
  *hour = bcdToDec(Wire.read() & 0x3f);
  *dayOfWeek = bcdToDec(Wire.read());
  *dayOfMonth = bcdToDec(Wire.read());
  *month = bcdToDec(Wire.read());
  *year = bcdToDec(Wire.read());
}
void displayTime(){
  byte second, minute, hour, dayOfWeek, dayOfMonth, month, year;
  // retrieve data from DS3231
```

```
readDS3231time(&second, &minute, &hour, &dayOfWeek, &dayOfMonth, &month,
&year);
// send it to the serial monitor
Serial.print(hour, DEC);
// convert the byte variable to a decimal number when displayed
Serial.print(":");
if (minute<10){
  Serial.print("0");
}
Serial.print(minute, DEC);
Serial.print(":");
if (second<10){
  Serial.print("0");
}
Serial.print(second, DEC);
Serial.print(" ");
Serial.print(dayOfMonth, DEC);
Serial.print("/");
Serial.print(month, DEC);
Serial.print("/");
Serial.print(year, DEC);
Serial.print(" Day of week: ");
switch(dayOfWeek){
case 1:
  Serial.println("Sunday");
  break;
case 2:
  Serial.println("Monday");
  break;
case 3:
  Serial.println("Tuesday");
  break;
case 4:
  Serial.println("Wednesday");
  break;
case 5:
  Serial.println("Thursday");
  break;
case 6:
  Serial.println("Friday");
  break;
```

```
  case 7:
    Serial.println("Saturday");
    break;
  }
}
void loop(){
  displayTime(); // display the real-time clock data on the Serial Monitor,
  delay(1000); // every second
}
```

SOURCE CODE

https://github.com/RuiSantosdotme/Random-Nerd-Tutorials/blob/master/Projects/rtc_example.ino

Retain the time in the Real Time Clock

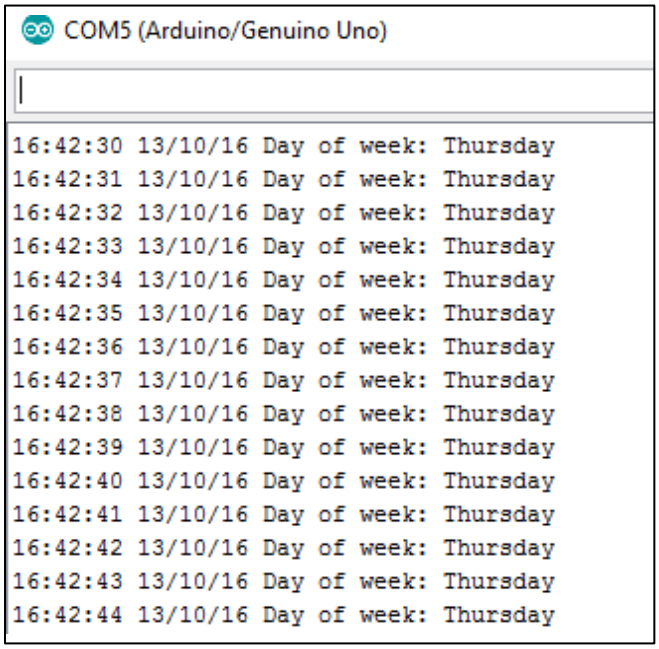
If you don't want to reset the time every time the RTC is turned off, you should do the following: after setting up the time, you should comment the function that sets the time and upload the code again.

```
// set the initial time here:
// DS3231 seconds, minutes, hours, day, date, month, year
//setDS3231time(33,53,16,1,25,9,16);
```

This is a very important step to set up the time in your RTC. If you don't do this, every time your RTC resets, it will display the time that you've set up previously instead of the current time.

Demonstration

Open the serial monitor at a baud rate of 9600 and you'll see the results. Here's the Serial Monitor displaying the current date and time.



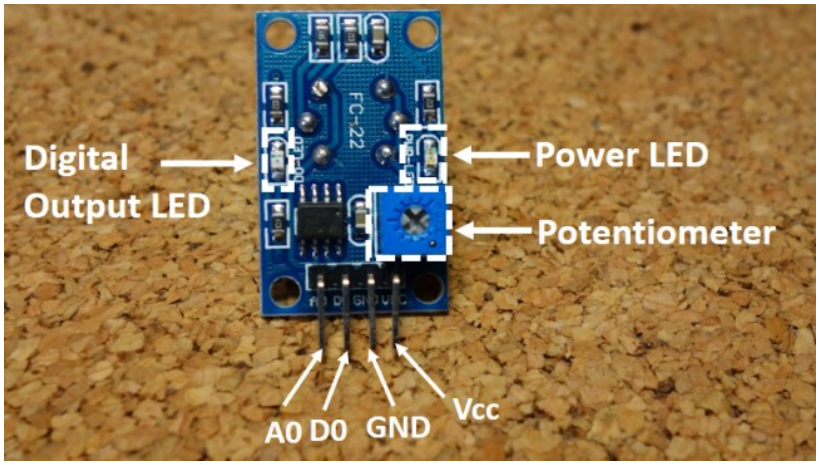
Wrapping up

The RTC module is really useful and you can use it as a clock, timer, etc. A very interesting project to do with the RTC is an alarm clock with and OLED display, for example.

MQ-2 Gas/Smoke Sensor

This guide shows how to build a smoke detector that beeps when it detects flammable gas or smoke. The MQ-2 smoke sensor is shown in the following figure:

The MQ-2 smoke sensor is sensitive to smoke and to the following flammable gases: LPG, butane, propane, methane, alcohol and hydrogen. The resistance across the sensor is different depending on the type of the gas. The smoke sensor has a built-in potentiometer that allows you to adjust the sensor digital output (D0) threshold. This threshold sets the value above which the digital pin will output a HIGH signal.

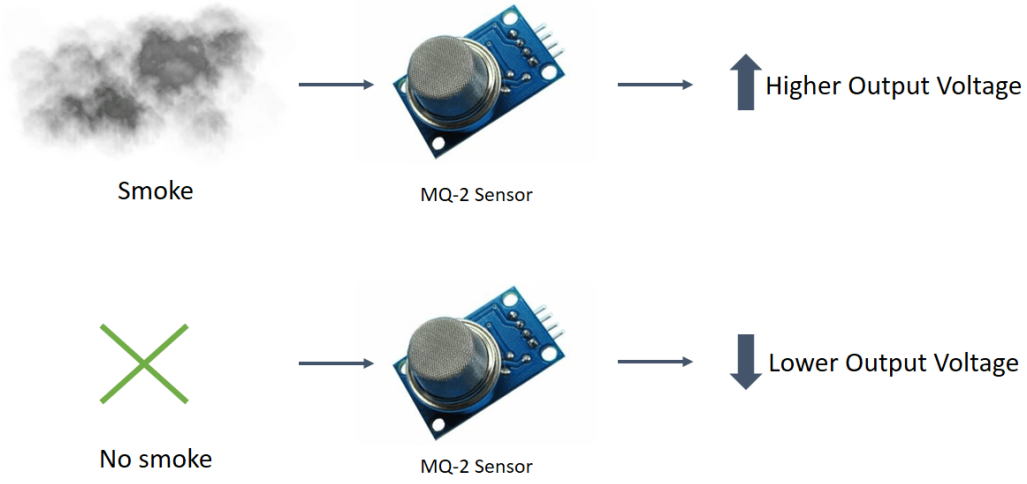


How does it work?

The voltage that the sensor outputs changes accordingly to the smoke/gas level that exists in the atmosphere. The sensor outputs a voltage that is proportional to the concentration of smoke/gas.

In other words, the relationship between voltage and gas concentration is the following:

- The greater the gas concentration, the greater the output voltage
- The lower the gas concentration, the lower the output voltage



The output can be an analog signal (A0) that can be read with an analog input of the Arduino or a digital output (D0) that can be read with a digital input of the Arduino.

Where to buy?

Click the link below to compare the sensor at different stores and find the best price:

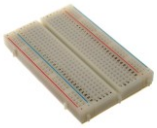
- MQ-2 Gas and Smoke Sensor

Gas Sensor with Arduino

In this example, you will read the sensor analog output voltage. When the smoke reaches a certain level, it will make sound a buzzer and a red LED will turn on.

When the output voltage is below that level, a green LED will be on.

For this example, you'll need the following parts:

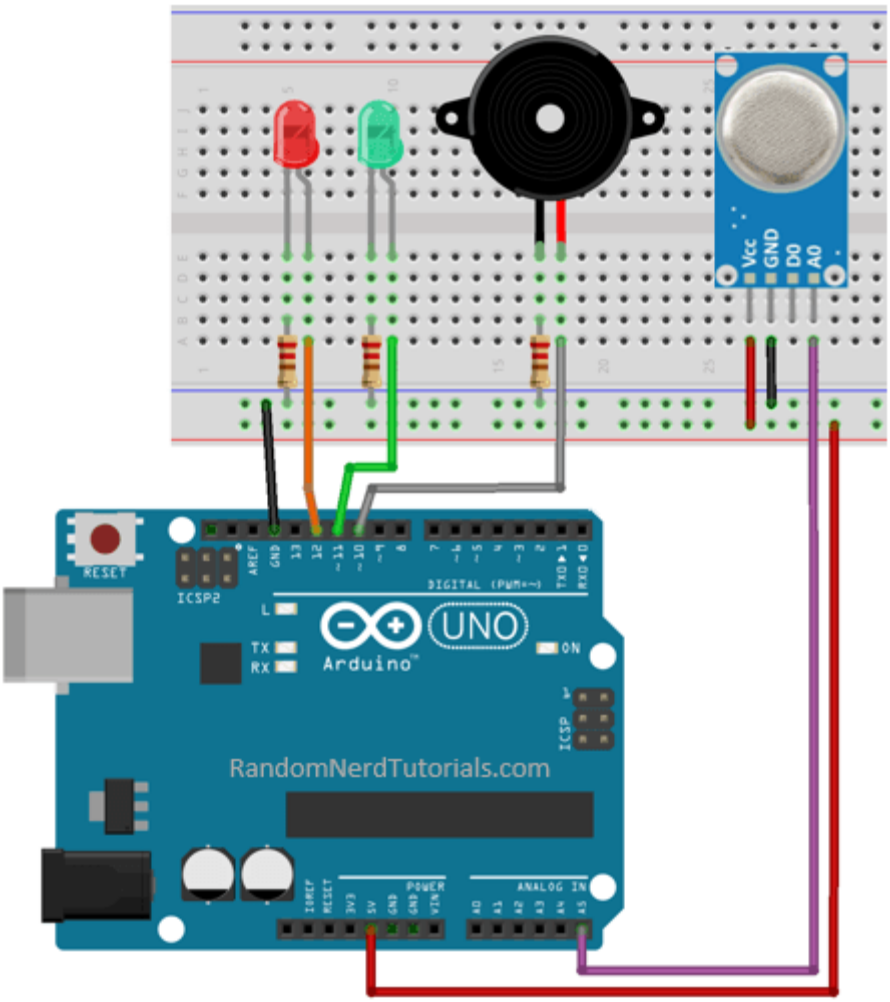
Figure	Name	Check Price
	Arduino UNO	Find best price on Maker Advisor
	MQ-2 gas/smoke sensor	Find best price on Maker Advisor
	Breadboard	Find best price on Maker Advisor
	2 x LEDs	Find best price on Maker Advisor
	3 x 220kΩ Resistor	Find best price on Maker Advisor
	Piezo buzzer	Find best price on Maker Advisor
	Jumper Wires	Find best price on Maker Advisor

Pin wiring

Sensor Pin	Wiring to Arduino Uno
A0	Analog pins
D0	Digital pins
GND	GND
VCC	5V

Schematic

Follow the next schematic diagram to complete the project:



Code

Upload the following sketch to your Arduino board (feel free to adjust the variable `sensorThres` with a different threshold value):

```
/*******
 All the resources for this project:
 http://randomnerdtutorials.com/
*******/

int redLed = 12;
int greenLed = 11;
int buzzer = 10;
```

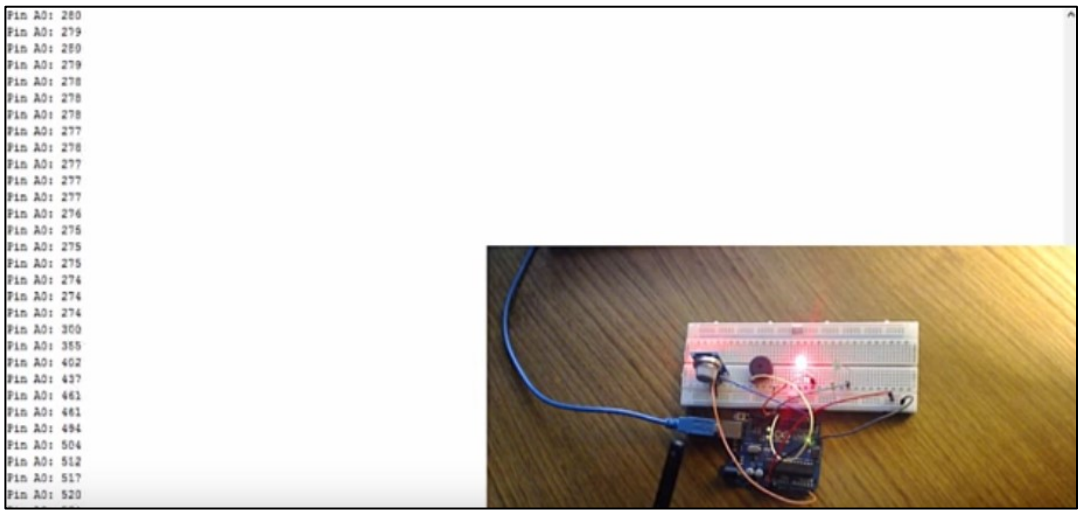
```
int smokeA0 = A5;
// Your threshold value
int sensorThres = 400;

void setup() {
  pinMode(redLed, OUTPUT);
  pinMode(greenLed, OUTPUT);
  pinMode(buzzer, OUTPUT);
  pinMode(smokeA0, INPUT);
  Serial.begin(9600);
}

void loop() {
  int analogSensor = analogRead(smokeA0);

  Serial.print("Pin A0: ");
  Serial.println(analogSensor);
  // Checks if it has reached the threshold value
  if (analogSensor > sensorThres)
  {
    digitalWrite(redLed, HIGH);
    digitalWrite(greenLed, LOW);
    tone(buzzer, 1000, 200);
  }
  else
  {
    digitalWrite(redLed, LOW);
    digitalWrite(greenLed, HIGH);
    noTone(buzzer);
  }
  delay(100);
}
```

{;} SOURCE CODE

https://github.com/RuiSantosdotme/Random-Nerd-Tutorials/blob/master/Projects/smoke_detector.ino

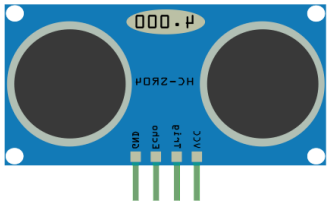
Demonstration

When you press a lighter next to the sensor, the red LED lights up and the buzzer beeps. You can also see at the serial monitor, the values changing and surpassing the threshold value.



Wrapping up

The MQ-2 gas sensor allows you to detect a wide variety of gases in the environment. It is very useful to build a smoke detector at home.



HC-SR04 Ultrasonic Sensor

The HC-SR04 ultrasonic sensor uses sonar to determine distance to an object like bats do. It offers excellent non-contact range detection with high accuracy and stable readings in an easy-to-use package. From 2 cm to 400 cm or 1" to 13 feet. It comes complete with ultrasonic transmitter and receiver module.

The HC-SR04 ultrasonic sensor is shown in the following figure:

Features

- Power Supply :+5V DC
- Quiescent Current : <2mA
- Working Current: 15mA
- Effectual Angle: <15°
- Ranging Distance : 2cm – 400 cm/1" – 13ft
- Resolution : 0.3 cm
- Measuring Angle: 30 degree
- Trigger Input Pulse width: 10uS
- Dimensions: 45mm x 20mm x 15mm

Where to buy?

Click the link below to compare the sensor at different stores and find the best price:

- HC-SR04 Ultrasonic Sensor

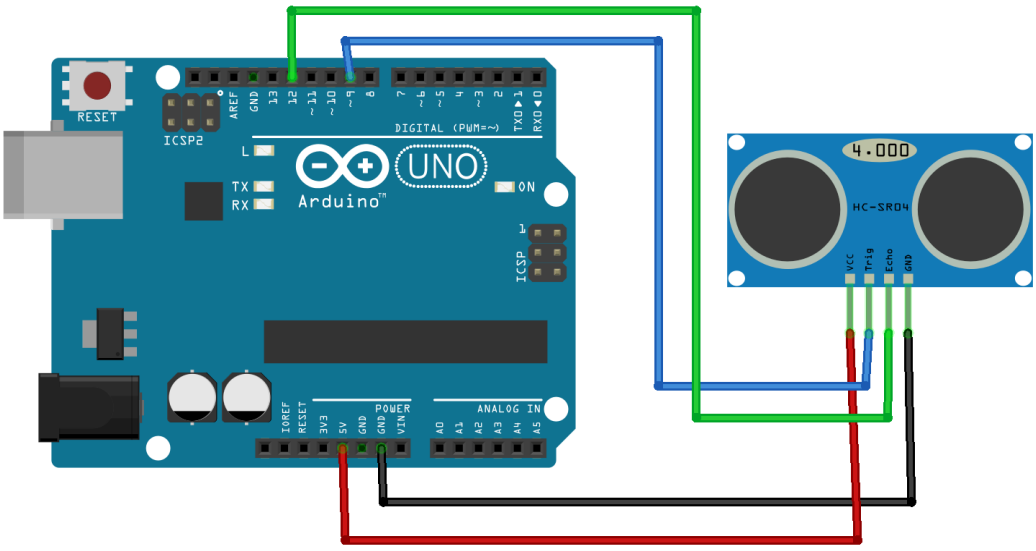
Arduino with HC – SR04 Sensor

In this project the ultrasonic sensor reads and writes the distance in the serial monitor.

Note: There's an Arduino library called NewPing that can make your life easier when using this sensor.

Schematic

Wire your sensor by following the schematic diagram below.



Sensor Pin	Wiring to Arduino Uno
VCC	5V
Trig	Trigger (INPUT) D9
Echo	Echo (OUTPUT) D12
GND	GND

Source code

Upload the following code to your Arduino IDE.

```
/*
 * created by Rui Santos, http://randomnerdtutorials.com
 *
 * Complete Guide for Ultrasonic Sensor HC-SR04
 *
    Ultrasonic sensor Pins:
        VCC: +5VDC
        Trig : Trigger (INPUT) - Pin11
        Echo: Echo (OUTPUT) - Pin 12
        GND: GND
 */

int trigPin = 11;
int echoPin = 12;
long duration, cm, inches;

void setup() {
  //Serial Port begin
  Serial.begin (9600);
  //Define inputs and outputs
  pinMode(trigPin, OUTPUT);
  pinMode(echoPin, INPUT);
}

void loop()
{
   // The sensor is triggered by a HIGH pulse of 10 or more microseconds.
  // Give a short LOW pulse beforehand to ensure a clean HIGH pulse:
  digitalWrite(trigPin, LOW);
  delayMicroseconds(5);
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);

  // Read the signal from the sensor: a HIGH pulse whose
  // duration is the time (in microseconds) from the sending
  // of the ping to the reception of its echo off of an object.
```

```
  pinMode(echoPin, INPUT);
  duration = pulseIn(echoPin, HIGH);

  // convert the time into a distance
  cm = (duration/2) / 29.1;
  inches = (duration/2) / 74;

  Serial.print(inches);
  Serial.print("in, ");
  Serial.print(cm);
  Serial.print("cm");
  Serial.println();

  delay(250);
}
```

{;} SOURCE CODE

https://github.com/RuiSantosdotme/Random-Nerd-Tutorials/blob/master/Projects/Ultrasonic_Sensor_HC-SR04.c

Source code with NewPing

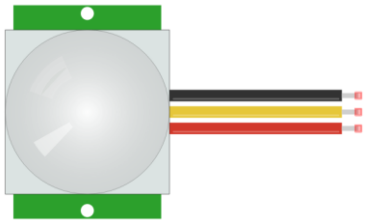
Below there is an example using the NewPing library. Download the library here.

```
/*
 * Posted on http://randomnerdtutorials.com
 * created by http://playground.arduino.cc/Code/NewPing
*/

#include <NewPing.h>

#define TRIGGER_PIN 12
#define ECHO_PIN 11
#define MAX_DISTANCE 200

NewPing sonar(TRIGGER_PIN, ECHO_PIN, MAX_DISTANCE); // NewPing setup of pins
and maximum distance.
```

```
void setup() {
   Serial.begin(9600);
}
void loop() {
   delay(50);
   unsigned int uS = sonar.ping_cm();
   Serial.print(uS);
   Serial.println("cm");
}
```

SOURCE CODE

https://github.com/RuiSantosdotme/Random-Nerd-Tutorials/blob/master/Projects/Ultrasonic_Sensor_HC-SR04_with_NewPing.c



PIR Motion Sensor

The PIR motion sensor is ideal to detect movement. PIR stand for "Passive Infrared". Basically, the PIR motion sensor measures infrared light from objects in its field of view. So, it can detect motion based on changes in infrared light in the environment. It is ideal to detect if a human has moved in or out of the sensor range.

The sensor in the figure above has two built-in potentiometers to adjust the delay time (the potentiometer at the left) and the sensitivity (the potentiometer at the right).

Where to buy?

Click the link below to compare the sensor at different stores and find the best price:

- PIR Motion Sensor (HC-SR501)

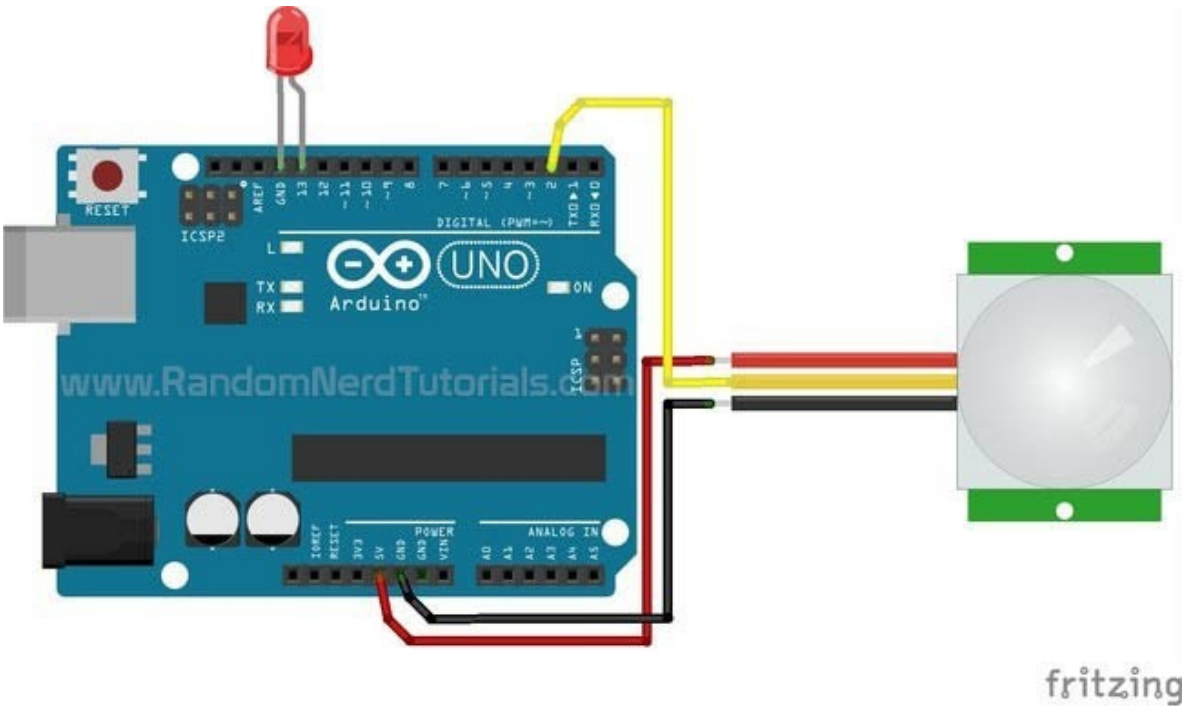
Detecting motion with the PIR motion sensor

In this example an LED will light up when movement is detected. For this example you need the following parts:

Figure	Name	Check Price
	Arduino UNO	Find best price on Maker Advisor
	PIR motion sensor	Find best price on Maker Advisor
	2 x LEDs	Find best price on Maker Advisor
	Jumper Wires	Find best price on Maker Advisor

Schematic

Assemble all the parts by following the schematic below.



Pin wiring

Sensor Pin	Wiring to Arduino Uno
GND	GND
OUT	Arduino digital pin
5V	5V

Code

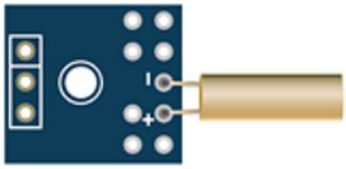
Upload the following code.

```
/*
    Arduino with PIR motion sensor
    For complete project details, visit:
http://RandomNerdTutorials.com/pirsensor
    Modified by Rui Santos based on PIR sensor by Limor Fried
*/

int led = 13;                // the pin that the LED is atteched to
int sensor = 2;              // the pin that the sensor is atteched to
int state = LOW;             // by default, no motion detected
int val = 0;                 // variable to store the sensor status (value)

void setup() {
  pinMode(led, OUTPUT);      // initalize LED as an output
  pinMode(sensor, INPUT);    // initialize sensor as an input
  Serial.begin(9600);        // initialize serial
}

void loop(){
  val = digitalRead(sensor);   // read sensor value
  if (val == HIGH) {           // check if the sensor is HIGH
    digitalWrite(led, HIGH);   // turn LED ON
    delay(100);                // delay 100 milliseconds

    if (state == LOW) {
      Serial.println("Motion detected!");
      state = HIGH;       // update variable state to HIGH
```

```
    }
  }
  else {
      digitalWrite(led, LOW); // turn LED OFF
      delay(200);             // delay 200 milliseconds

      if (state == HIGH){
        Serial.println("Motion stopped!");
        state = LOW;       // update variable state to LOW
    }
  }
}
```

SOURCE CODE

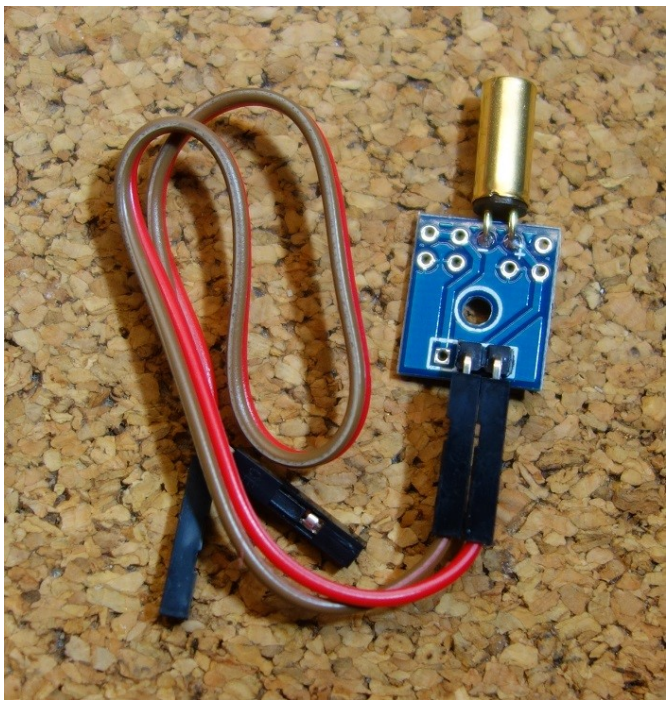
https://github.com/RuiSantosdotme/Random-Nerd-Tutorials/blob/master/Projects/Arduino_with_PIR_motion_sensor.ino



Tilt Sensor

The tilt sensor is many times referred to as inclinometer, tilt switch or rolling ball sensor. Using a tilt sensor is a simple way to detect orientation or inclination.

The tilt sensor module is the one in the following figure.

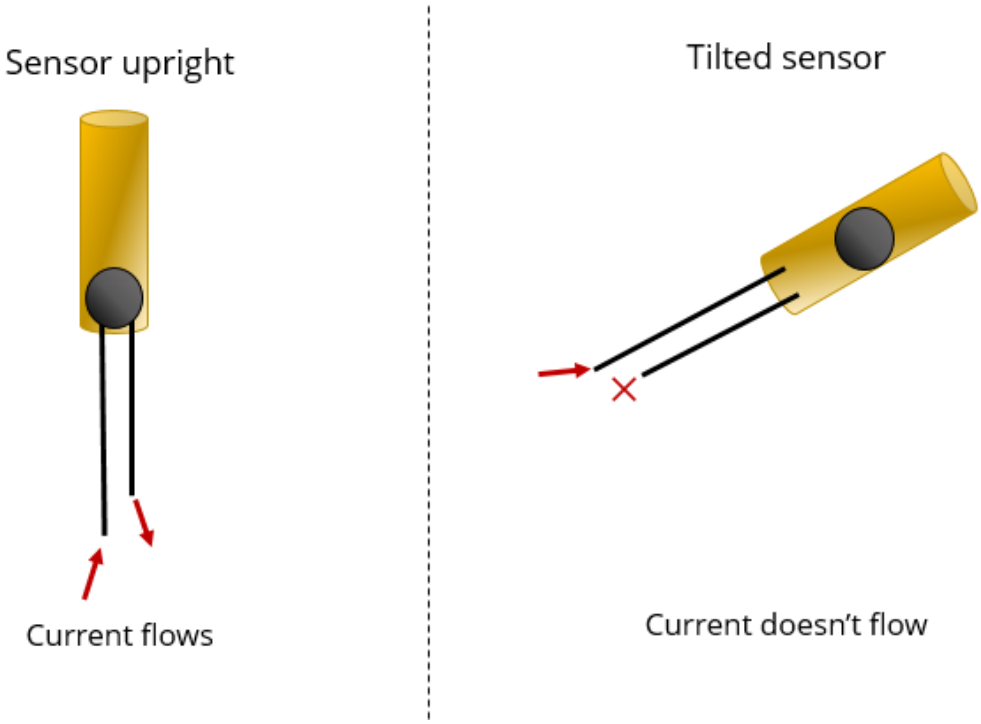


The tilt sensor allows to detect orientation or inclination. It detects if the sensor is completely upright or if it is tilted.

This makes it very useful to be used, for example, in toys, robots and other appliances whose working methodology depends on inclination.

How does it work?

The tilt sensor is cylindrical and contains a free conductive rolling ball inside with two conductive elements (poles) beneath.



Here's how it works:

- When the sensor is completely upright, the ball falls to the bottom of the sensor and connects the poles, allowing the current to flow.
- When the sensor is tilted, the ball doesn't touch the poles, the circuit is open, and the current doesn't flow.

This way, the tilt sensor acts like a switch that is turned on or off depending on its inclination. So, it will give digital information to the Arduino, either an HIGH or a LOW signal.

Where to buy?

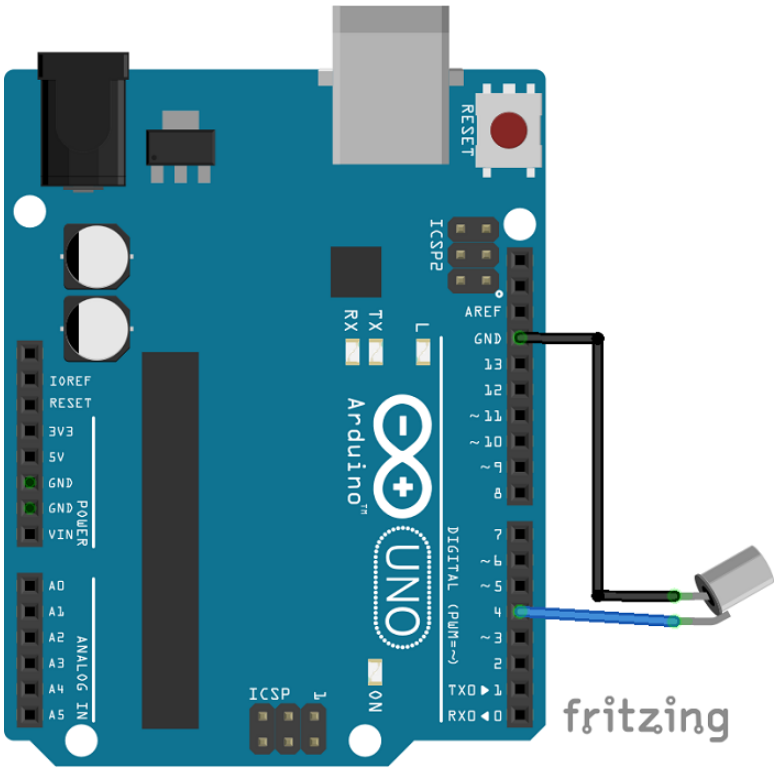
Click the link below to compare the sensor at different stores and find the best price:

- Tilt Sensor

Pin wiring

Wiring the tilt sensor to you Arduino is pretty straightforward. You just need to connect one pin to the Arduino digital pins and the GND to the GND.

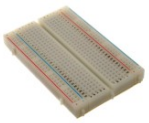
If you connect the sensor like so, you need to activate the Arduino internal pull-up resistor for the digital pin to which your sensor is connected to. Otherwise, you should use a 10kOhm pull up resistor in your circuit.



Example: Tilt sensitive LED

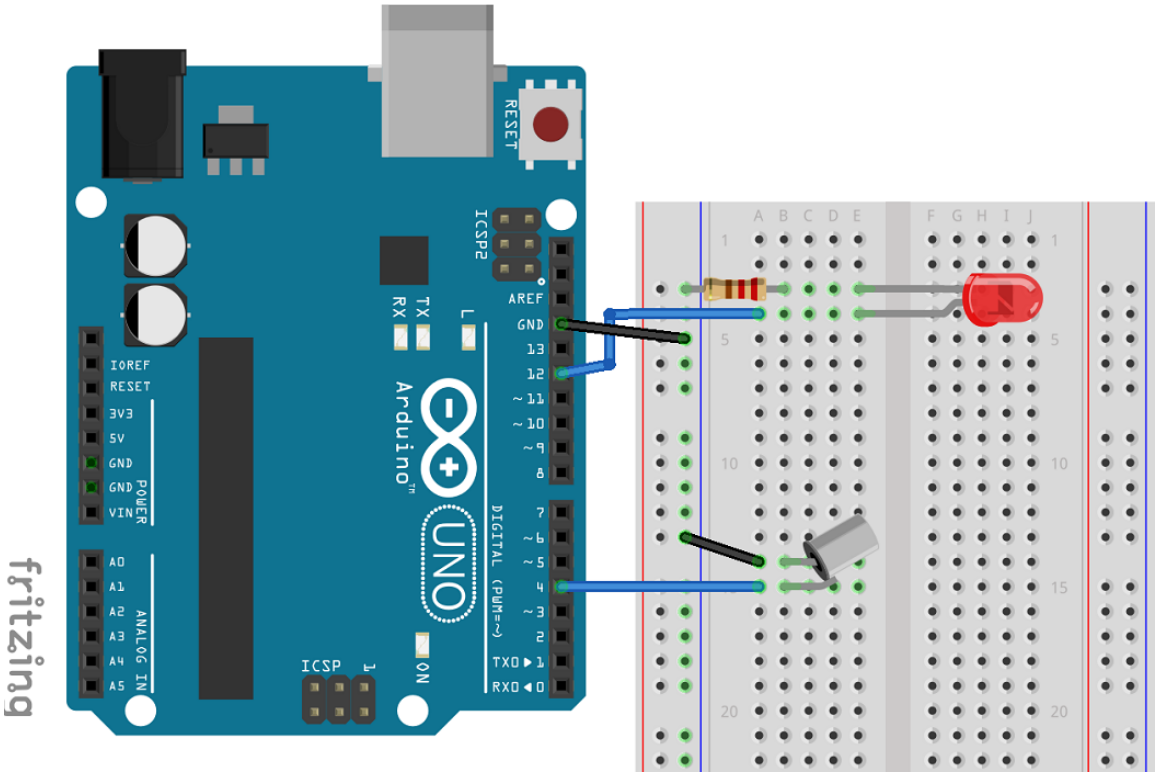
This is just a simple example for you to start put hands on your tilt sensor. In this example, an LED will be turned off if the sensor is upright, and will be turned on if the sensor is tilted.

Figure	Name	Check Price
	Arduino UNO	Find best price on Maker Advisor

	Tilt Sensor	Find best price on Maker Advisor
	Breadboard	Find best price on Maker Advisor
	2 x LEDs	Find best price on Maker Advisor
	220Ω Resistor	Find best price on Maker Advisor
	Jumper Wires	Find best price on Maker Advisor

Schematics

For this example, you just need to add an LED to the schematics in the "Pin Wiring" section.



Code

To complete this example, upload the following code.

```
/*
 * Rui Santos
 * Complete Project Details http://randomnerdtutorials.com
 */
int ledPin = 12;
int sensorPin = 4;
int sensorValue;
int lastTiltState = HIGH;   // the previous reading from the tilt sensor

// the following variables are long's because the time, measured in
miliseconds,
// will quickly become a bigger number than can be stored in an int.
long lastDebounceTime = 0;  // the last time the output pin was toggled
long debounceDelay = 50;    // the debounce time; increase if the output
flickers

void setup(){
  pinMode(sensorPin, INPUT);
  digitalWrite(sensorPin, HIGH);
  pinMode(ledPin, OUTPUT);
  Serial.begin(9600);
}

void loop(){
  sensorValue = digitalRead(sensorPin);
  // If the switch changed, due to noise or pressing:
  if (sensorValue == lastTiltState) {
    // reset the debouncing timer
    lastDebounceTime = millis();
  }
  if ((millis() - lastDebounceTime) > debounceDelay) {
    // whatever the reading is at, it's been there for longer
    // than the debounce delay, so take it as the actual current state:
    lastTiltState = sensorValue;
  }
  digitalWrite(ledPin, lastTiltState);
```

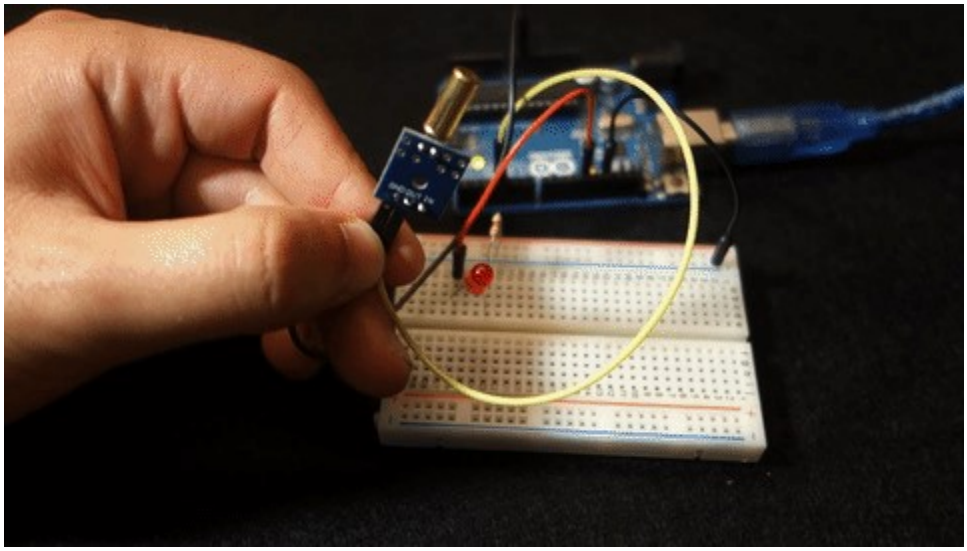
```
  Serial.println(sensorValue);
  delay(500);
}
```

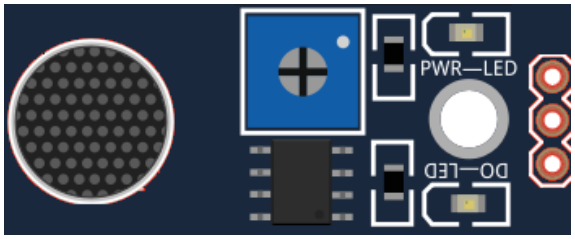
SOURCE CODE

https://github.com/RuiSantosdotme/Random-Nerd-Tutorials/blob/master/Projects/Arduino_tilt_sensor.ino

Demonstration

In the end, this is what you'll have.

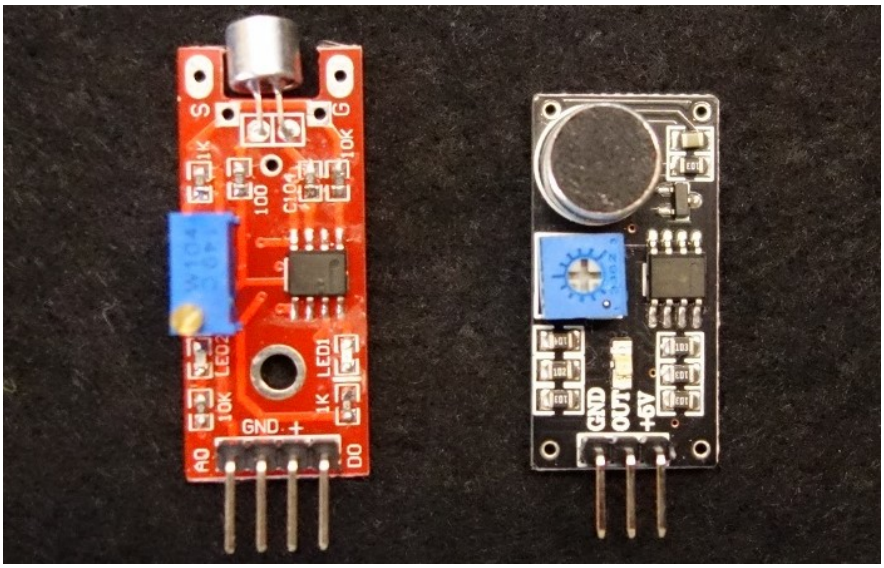




Microphone Sound Sensor

This guide shows how to use the microphone sound sensor with the Arduino. The microphone sound sensor, as the name says, detects sound. It gives a measurement of how loud a sound is.

There is a wide variety of these sensors. In the figure below you can see the most common ones used with the Arduino.



At the left you can see the KY-038 and at the right the LM393 microphone sound sensors. Both sensor modules have a built-in potentiometer to adjust the sensitivity of the digital output pins.

Where to buy?

Click the link below to compare the sensor at different stores and find the best price:

- Microphone sound sensor KY-038

Pin wiring

Sensor Pin	Wiring to Arduino Uno
A0	Analog pins
D0	Digital pins
GND	GND
VCC	5V

If you're using the LM 393 module, you should connect the OUT pin to an Arduino digital pin.

Example: Sound Sensitive Lights

In this example, a microphone sensor will detect the sound intensity of your surroundings and will light up and LED if the sound intensity is above a certain threshold.

You can modify this project by adding more LEDs. You can also adjust your sensor sensitivity so that the LEDs are turned on or off accordingly to the beat of a music.

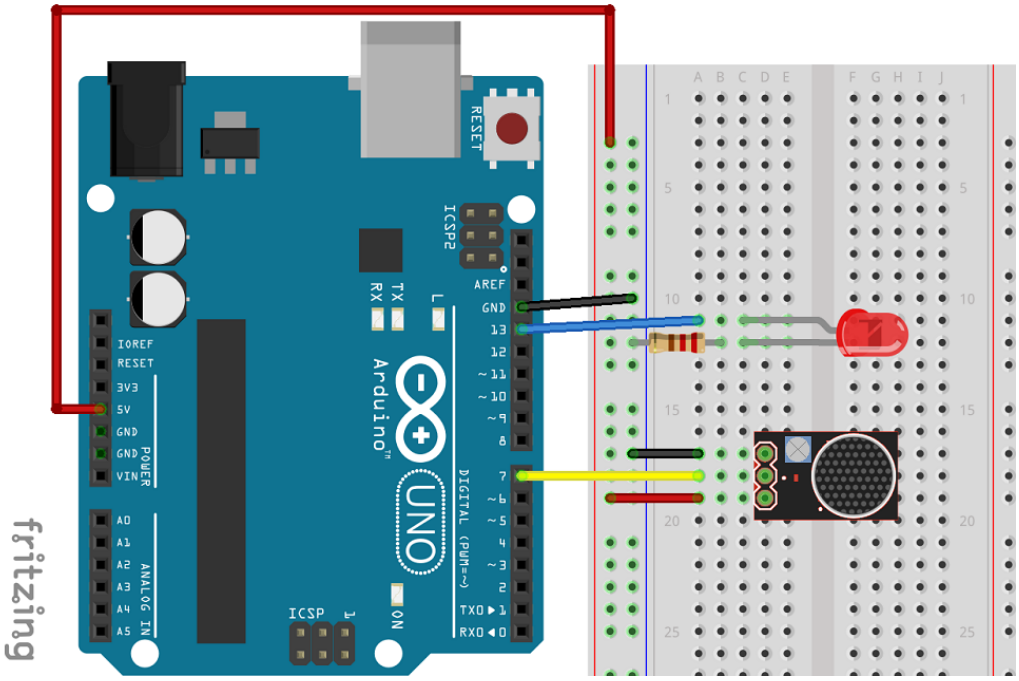
For this example you'll need the following components.

Figure	Name	Check Price
	Arduino UNO	Find best price on Maker Advisor
	Microphone sound sensor	Find best price on Maker Advisor
	Breadboard	Find best price on Maker Advisor
	LED	Find best price on Maker Advisor
	220Ω Resistor	Find best price on Maker Advisor

	Jumper Wires	Find best price on Maker Advisor

Schematic

Assemble all the parts by following the schematic below:



Code

Upload the following code to your Arduino.

```
/*
 * Rui Santos
 * Complete Project Details http://randomnerdtutorials.com
*/

int ledPin=13;
int sensorPin=7;
boolean val =0;

void setup(){
  pinMode(ledPin, OUTPUT);
```

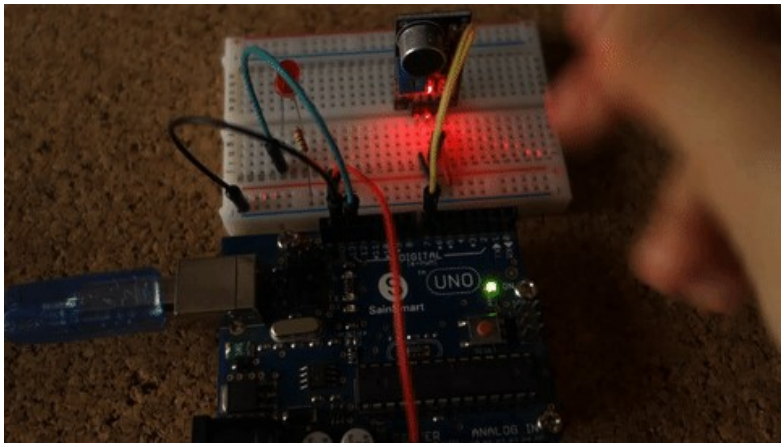
```
  pinMode(sensorPin, INPUT);
  Serial.begin (9600);
}

void loop (){
  val =digitalRead(sensorPin);
  Serial.println (val);
  // when the sensor detects a signal above the threshold value, LED flashes
  if (val==HIGH) {
    digitalWrite(ledPin, HIGH);
  }
  else {
    digitalWrite(ledPin, LOW);
  }
}
```

SOURCE CODE

https://github.com/RuiSantosdotme/Random-Nerd-Tutorials/blob/master/Projects/Arduino_microphone_sensor.ino

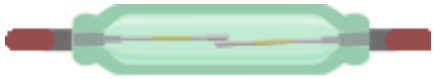
Demonstration

After uploading the code, you can clap next to the sensor. If the LED is not turning on, you need to change the sensor sensitivity by rotating the potentiometer.



You can also adjust the sensitivity so that the LED follows the beat of a certain music.

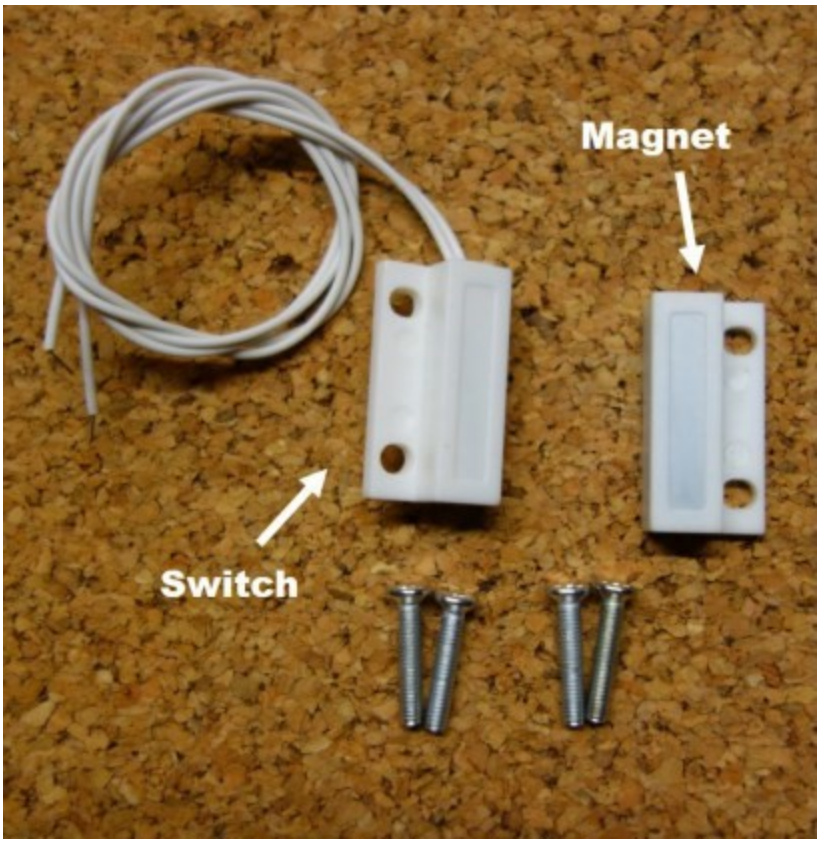
Add more LEDs to a more spectacular effect.



Reed Switch

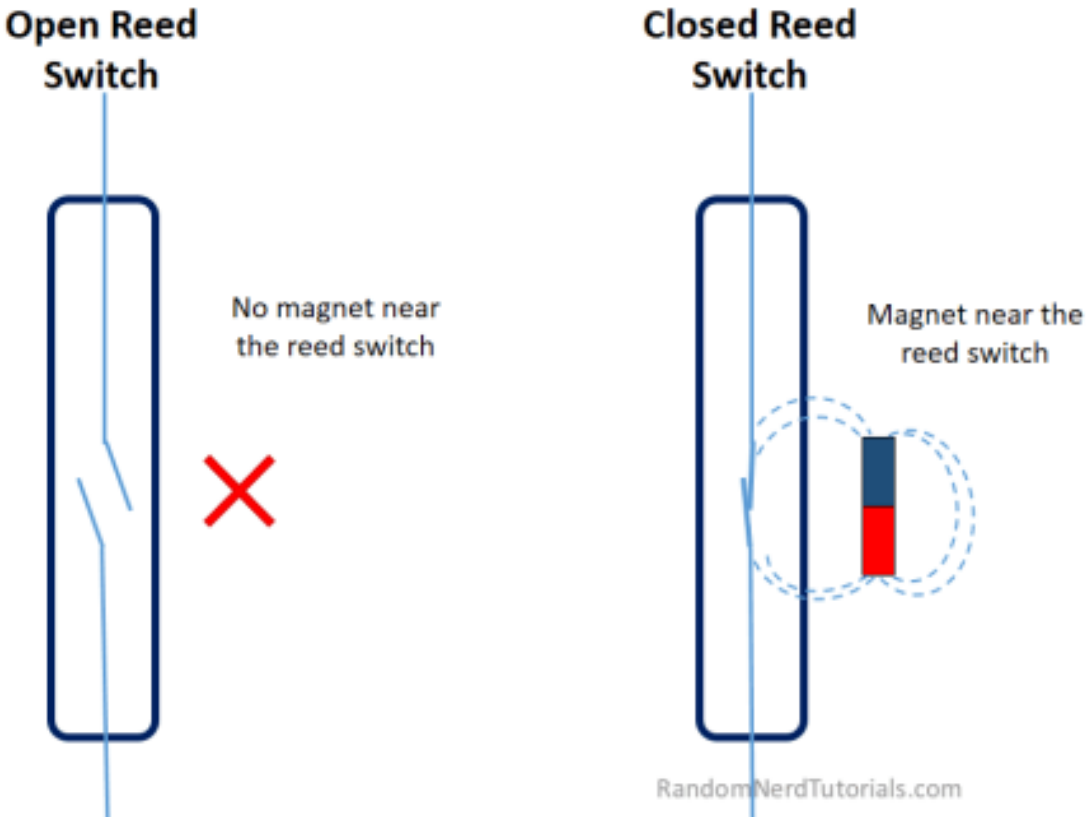
A magnetic contact switch is basically a reed switch encased in a plastic shell so that you can easily apply them in a door, a window or a drawer to detect if the door is open or closed.

The switch that we are going to use has two parts: the switch itself, that usually comes opened and the magnet. When you buy this switch, it also comes with 4 screws, so that you can attach it to your door.



How does it work?

It's very simple. The electrical circuit is closed when a magnet is near the switch (less than 13 mm (0.5") away). When the magnet is far away from the switch, the circuit is open. See the figure below.



Where to buy?

Click the link below to compare the sensor at different stores and find the best price:

- Magnetic reed switch

Project example

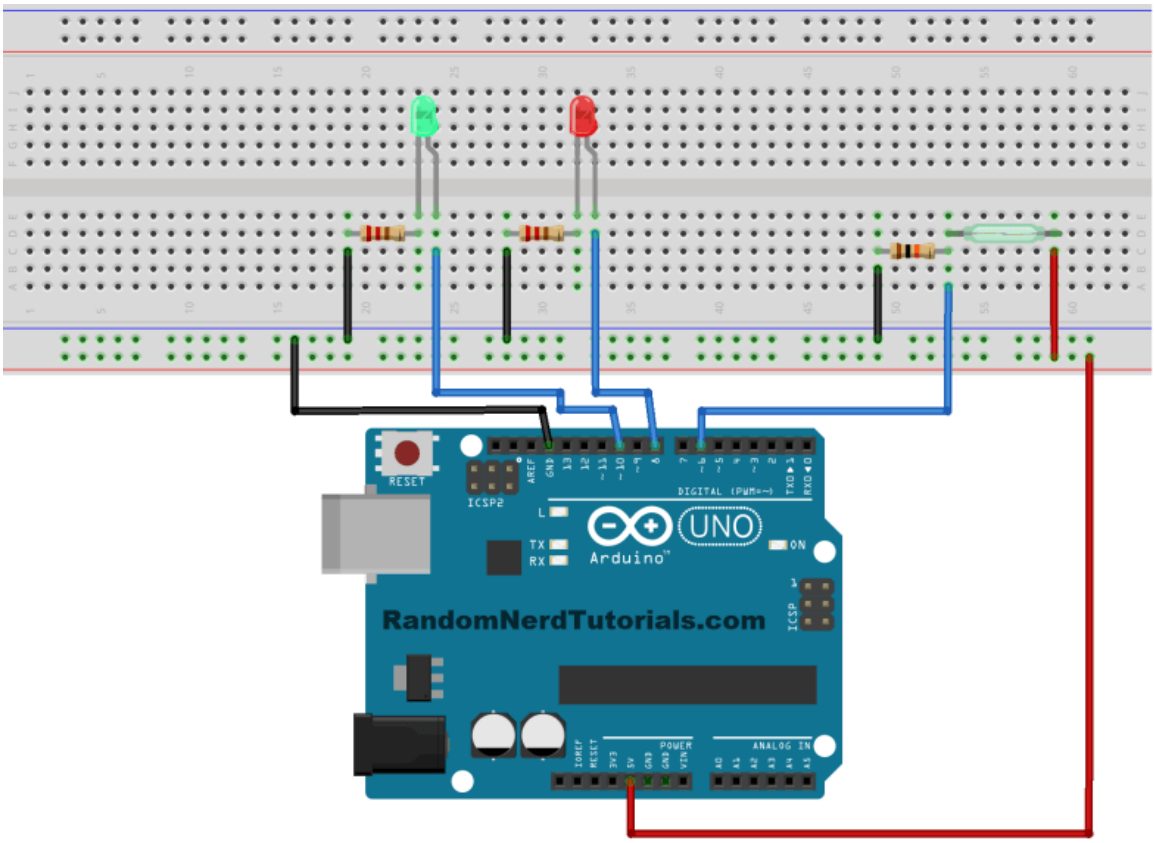
In this example, we will turn on a red LED if your door is open and a green LED if your door is closed.

Figure	Name	Check Price
	Arduino UNO	Find best price on Maker Advisor
	Magnetic reed switch	Find best price on Maker Advisor
	Breadboard	Find best price on Maker Advisor

	LED	Find best price on Maker Advisor
	2x 220Ω Resistor, 1x 10kΩ Resistor	Find best price on Maker Advisor
	Jumper Wires	Find best price on Maker Advisor

Schematic

Follow the next schematic diagram to complete this project.



Code

For this example, upload the following code:

```
/*
 Created by Rui Santos
 All the resources for this project:
```

```
 http://randomnerdtutorials.com/
*/

int ledOpen=8;
int ledClose=10;
int switchReed=6;

void setup(){
  pinMode(ledOpen, OUTPUT);
  pinMode(ledClose, OUTPUT);
  pinMode(switchReed, INPUT);
  Serial.begin(9600);
}

void loop(){

  if (digitalRead(switchReed)==HIGH){
    digitalWrite(ledOpen, LOW);
    digitalWrite(ledClose, HIGH);
    Serial.println("Your Door is Closed");
  }
  else {
    digitalWrite(ledOpen, HIGH);
    digitalWrite(ledClose, LOW);
    Serial.println("Your Door is Open");
  }
  delay(1);
}
```

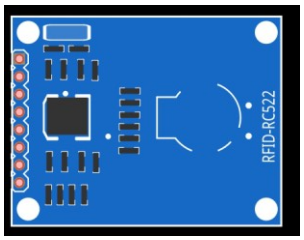
{;} SOURCE CODE

https://github.com/RuiSantosdotme/Random-Nerd-Tutorials/blob/master/Projects/Magnetic_Contact_Switch.ino

Demonstration

You can watch this video demonstration to see the project in action:

https://youtu.be/yY0DK8fe04w



MRFC522 RFID

RFID means radio-frequency identification. RFID uses electromagnetic fields to transfer data over short distances. RFID is useful to identify people, to make transactions, etc…

You can use an RFID system to open a door. For example, only the person with the right information on his card is allowed to enter. An RFID system uses:

- **tags** attached to the object to be identified, in this example we have a keychain and an electromagnetic card. Each tag has his own unique identification (**UID**).

- two-way radio transmitter-receiver, the **reader**, that send a signal to the tag and read its response.

Specifications

- Input voltage: 3.3V
- Frequency: 13.56MHz

Where to Buy?

Click the link below to compare the module at different stores and find the best price:

- MRFC522 RFID

Pin wiring

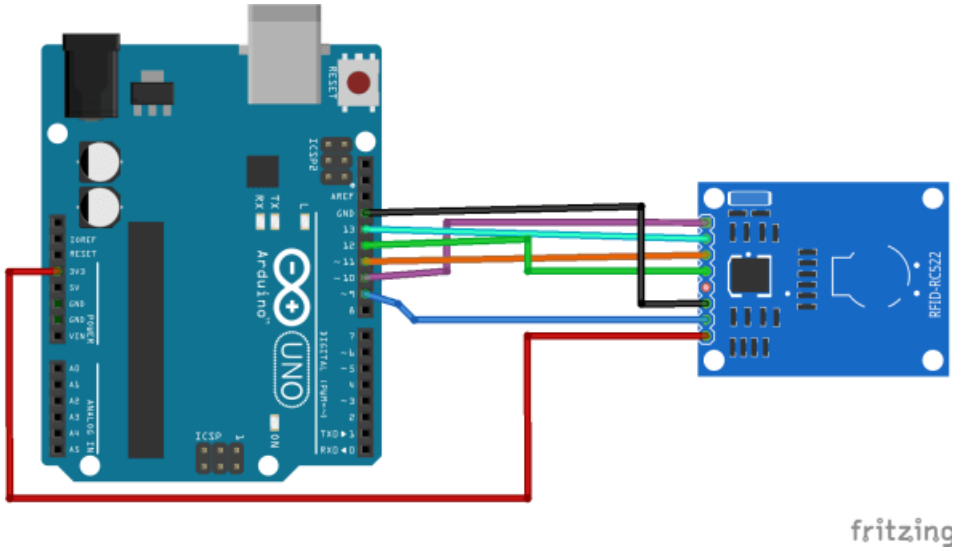
Sensor Pin	Wiring to Arduino Uno
SDA	Digital 10
SCK	Digital 13
MOSI	Digital 11
MISO	Digital 12
IRQ	Don't connect
GND	GND
RST	Digital 9
3.3V	3.3V

Reading Data from a RFID tag

In this example you will read data from an RFID tag. You will authorize a predetermined tag and deny the others.

Schematic

Wire your RFID reader module as in the schematic below.



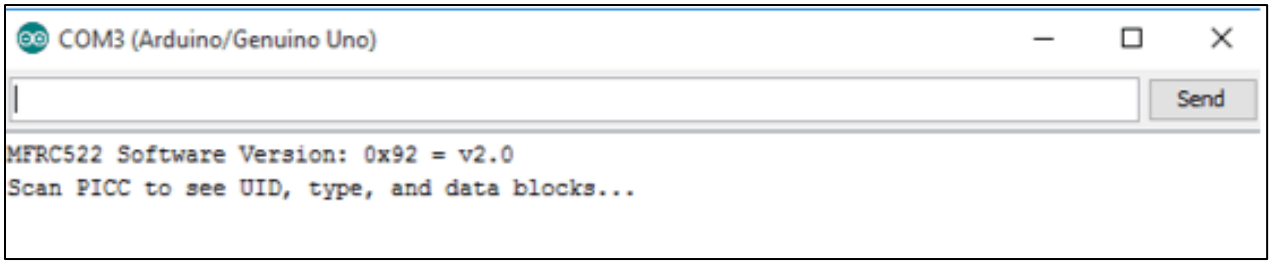
Library download

You need the RFID library for this project:

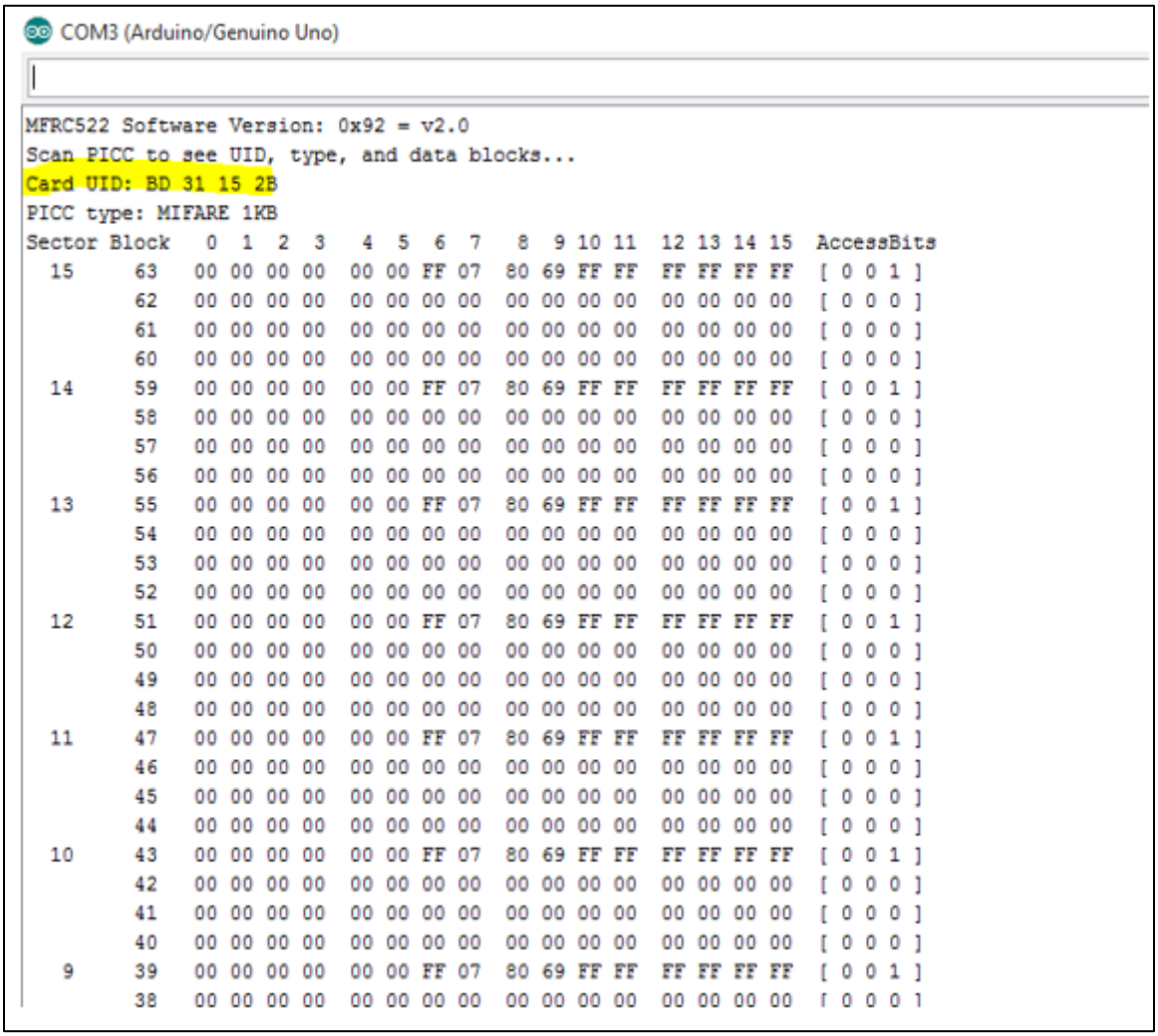
1. Click here to download the RFID library here created by miguelbalboa. You should have a .zip folder in your Downloads folder.
2. Unzip the .zip folder and you should get rfid-master folder.
3. Rename your folder from ~~rfid-master~~ to rfid.
4. Move the rfid folder to your Arduino IDE installation libraries folder.
5. Finally, re-open your Arduino IDE.
6. Restart your Arduino IDE

Go to **File ▸ Examples ▸ MFRC522 ▸ DumpInfo** and upload the code. This code will be available in your Arduino IDE (after installing the RFID library).

Then, open the serial monitor. You should see something like the figure below:



Approximate the RFID card or the keychain to the reader. Let the reader and the tag closer until all the information is displayed.



This is the information that you can read from the card, including the card UID that is highlighted in yellow. The information is stored in the memory that is divided into segments and blocks as you can see in the previous picture. You have 1024 bytes of data storage divided into 16 sectors.

Write down your UID card because you'll need it later.

Then, you need the following code (don't upload it now).

```
/*
 *
 * All the resources for this project: http://randomnerdtutorials.com/
 * Modified by Rui Santos
 *
 * Created by FILIPEFLOP
 *
 */

#include <SPI.h>
#include <MFRC522.h>

#define SS_PIN 10
#define RST_PIN 9
MFRC522 mfrc522(SS_PIN, RST_PIN);   // Create MFRC522 instance.

void setup()
{
  Serial.begin(9600);   // Initiate a serial communication
  SPI.begin();      // Initiate  SPI bus
  mfrc522.PCD_Init();   // Initiate MFRC522
  Serial.println("Approximate your card to the reader...");
  Serial.println();

}
void loop()
{
  // Look for new cards
  if ( ! mfrc522.PICC_IsNewCardPresent())
  {
    return;
  }
  // Select one of the cards
  if ( ! mfrc522.PICC_ReadCardSerial())
  {
    return;
  }
  //Show UID on serial monitor
  Serial.print("UID tag :");
  String content= "";
  byte letter;
```

```
  for (byte i = 0; i < mfrc522.uid.size; i++)
  {
     Serial.print(mfrc522.uid.uidByte[i] < 0x10 ? " 0" : " ");
     Serial.print(mfrc522.uid.uidByte[i], HEX);
     content.concat(String(mfrc522.uid.uidByte[i] < 0x10 ? " 0" : " "));
     content.concat(String(mfrc522.uid.uidByte[i], HEX));
  }
  Serial.println();
  Serial.print("Message : ");
  content.toUpperCase();
  if (content.substring(1) == "BD 31 15 2B") //change here the UID of the
card/cards that you want to give access
  {
    Serial.println("Authorized access");
    Serial.println();
    delay(3000);
  }

 else   {
    Serial.println(" Access denied");
    delay(3000);
  }
}
```

SOURCE CODE

https://github.com/RuiSantosdotme/Random-Nerd-Tutorials/blob/master/Projects/MFRC522_RFID_Reader_with_Arduino.ino

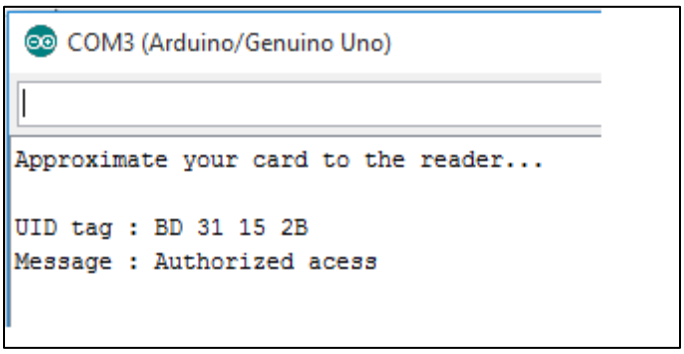
In the piece of code above you need to change the line

`if (content.substring(1) == "REPLACE WITH YOUR UID")` and type the UID card you've written previously.

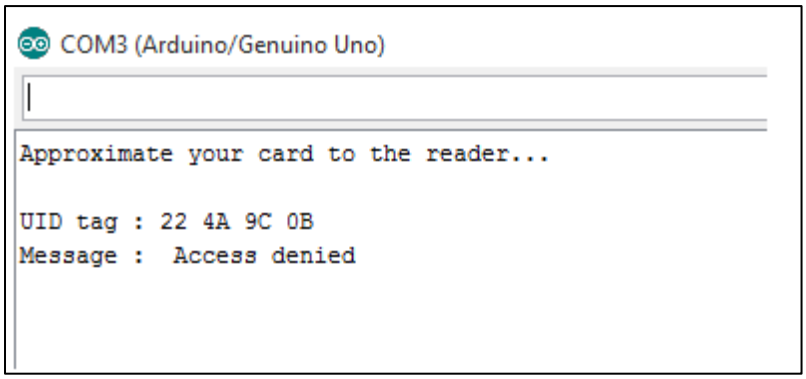
Demonstration

Now, upload the code to your Arduino and open the serial monitor.

Approximate the card you've chosen to give access and you'll see:



If you approximate another tag with another UID, the denial message will show up:

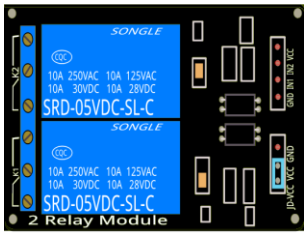


You can watch the following video demonstration to see the project in action:

https://youtu.be/mpLzcBDBl1U

Wrapping up

The MFRC522 RFID reader allows you to do a project in which you have security access to something. For example, only the right tag can open a specific door. This is just a simple guide to show you how this reader works. Now, the idea is for you to apply this knowledge and use your imagination to make your own projects.



Relay Module

A relay is an electrically operated switch of mains voltage. It can be turned on or off, letting the current go through or not. The relay module is shown in the figure below.

This module has two channels (those blue cubes).

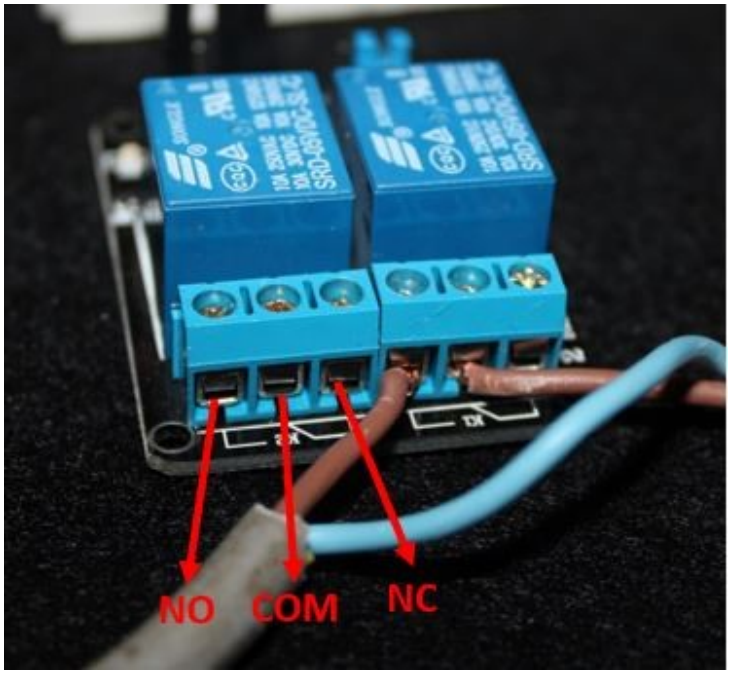
Where to buy?

Click the link below to compare the module at different stores and find the best price:

- Relay module

Mains voltage connections

In relation to mains voltage, relays have 3 possible connections:



- **COM**: common pin
- **NO**: **normally open** – there is no contact between the common pin and the normally open pin. So, when you trigger the relay, it connects to the COM pin and power is provided to the load (a desktop lamp, in our case).
- **NC: normally closed** – there is contact between the common pin and the normally closed pin. There is always connection between the COM and NC pins, even when the relay is turned off. When you trigger the relay, the circuit is opened and there is no supply provided to the load.

If you want to control a lamp for example, it is better to use a normally-open circuit, because we just want to light up the lamp occasionally.

Pin wiring

The connections between the relay and the Arduino are simple:

- **GND**: goes to ground
- **IN1**: controls the first relay. Should be connected to an Arduino digital pin
- **IN2**: controls the second relay. Should be connected to an Arduino digital pin if you are using this second relay. Otherwise, you don't need to connect it.
- **VCC**: goes to 5V.

Controlling a Lamp with a Relay Module and PIR Motion Sensor

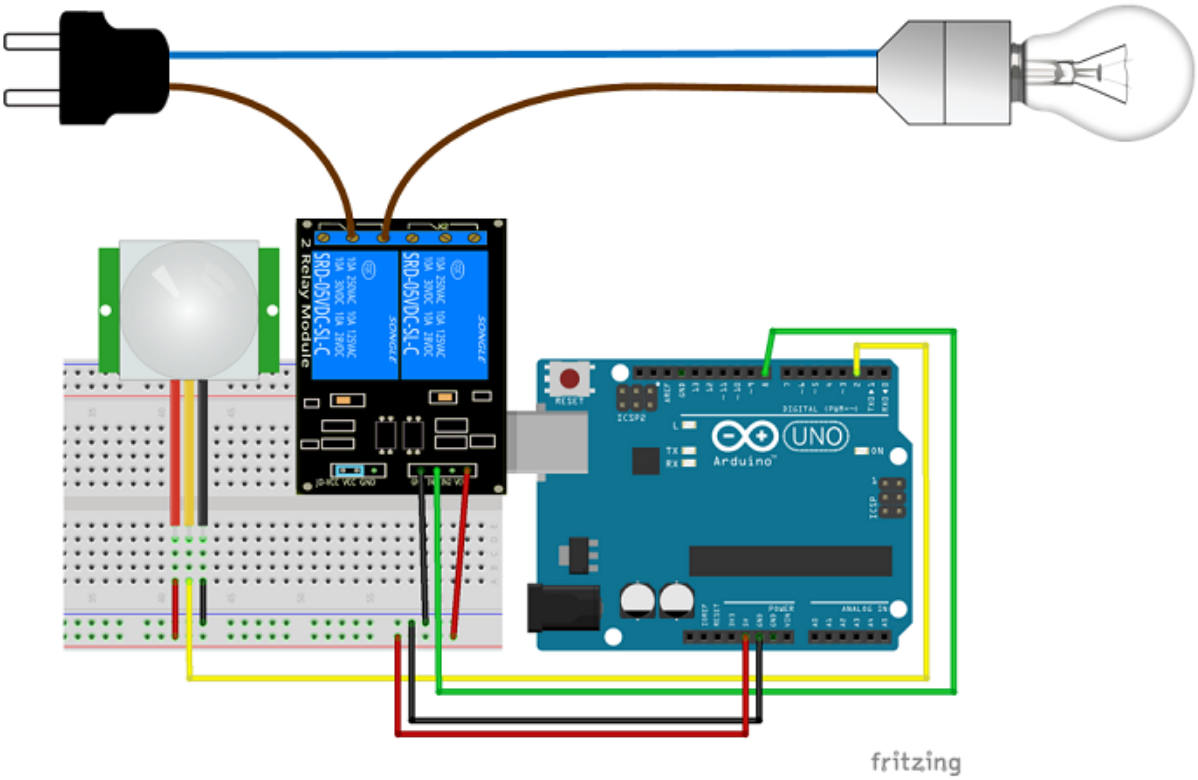
In this example you will create a motion sensitive lamp. A lamp will light up for 10 seconds every time motion is detected. Motion will be detected using a PIR motion sensor.

Figure	Name	Check Price
	Arduino UNO	Find best price on Maker Advisor
	Relay Module	Find best price on Maker Advisor
	Lamp Cord Set	-
	PIR Motion Sensor	Find best price on Maker Advisor
	Jumper Wires	Find best price on Maker Advisor

Schematic

Assemble all the parts by following the schematic diagram below.

Be very careful with the mains voltage connections. Make sure you have the relay unplugged from mains voltage when assembling the circuit.



Code

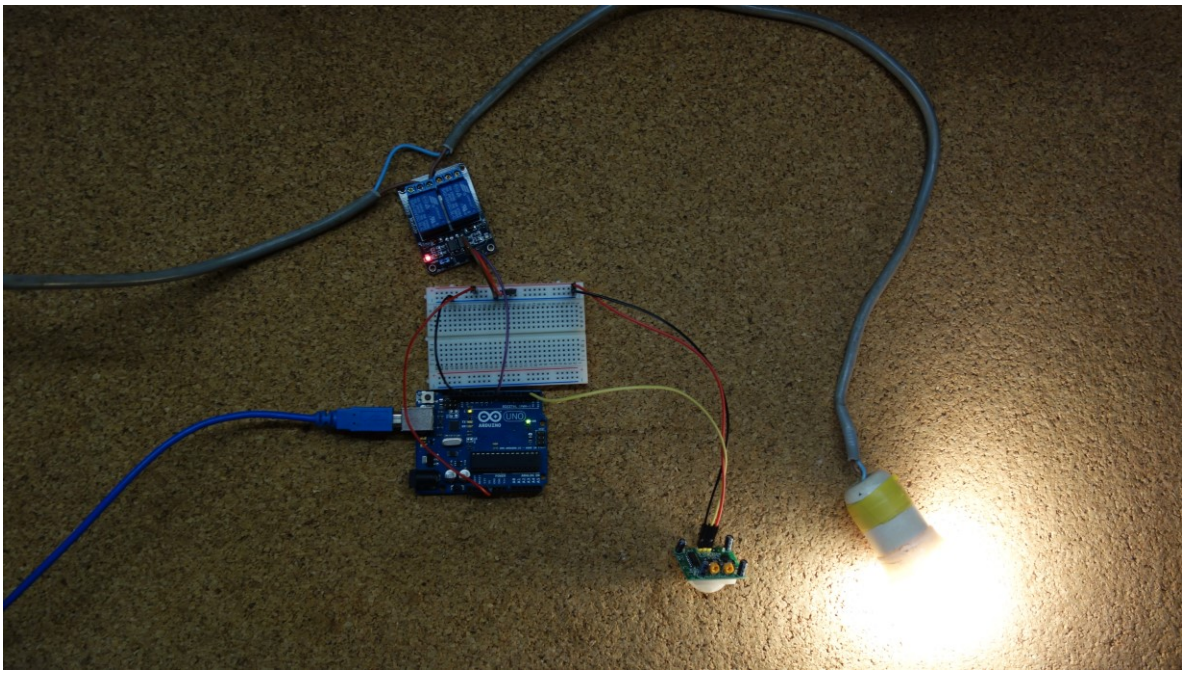
Copy the following code to your Arduino IDE and upload it to your Arduino board.

```
/*********
  Rui Santos
  Complete project details at http://randomnerdtutorials.com
*********/

// Relay pin is connected to D8. The active wire is connected to Normally
open and common
int relay = 8;
volatile byte relayState = LOW;

// PIR Motion Sensor is connected to D2.
int PIRInterrupt = 2;

// Timer Variables
long lastDebounceTime = 0;
long debounceDelay = 10000;
```

```
void setup() {
  // Pin for relay module set as output
  pinMode(relay, OUTPUT);
  digitalWrite(relay, HIGH);
  // PIR motion sensor set as an input
  pinMode(PIRInterrupt, INPUT);
  // Triggers detectMotion function on rising mode to turn the relay on, if
the condition is met
  attachInterrupt(digitalPinToInterrupt(PIRInterrupt), detectMotion, RISING);
  // Serial communication for debugging purposes
  Serial.begin(9600);
}

void loop() {
  // If 10 seconds have passed, the relay is turned off
  if((millis() - lastDebounceTime) > debounceDelay && relayState == HIGH){
    digitalWrite(relay, HIGH);
    relayState = LOW;
    Serial.println("OFF");
  }
  delay(50);
}

void detectMotion() {
  Serial.println("Motion");
  if(relayState == LOW){
    digitalWrite(relay, LOW);
  }
  relayState = HIGH;
  Serial.println("ON");
  lastDebounceTime = millis();
}
```

Demonstration

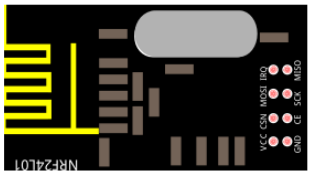
Now, when movement is detected your lamp lights up.



Wrapping up

Controlling a relay module with the Arduino is as simple as controlling an output.

With the relay module you can control almost any AC electronics appliance (not just lamps).



nRF24L01

These RF modules are very popular among the Arduino tinkerers. The nRF24L01 is used on a wide variety of applications that require wireless control. They are transceivers which means that each module can transmit and receive data.

These modules are very cheap and you can use them with any microcontroller (MCU).

Specifications nRF24L01 – 2.4GHz RF Transceiver

- Low cost single-chip 2.4GHz GFSK RF transceiver IC
- Range with Antenna: 250Kb rate (Open area) >1000 meter
- Power: Ultra low power consumption
- Input Voltage: 3.3V
- Pins: 5V tolerant
- Price: $2

Where to buy?

Click the link below to compare these modules at different stores and find the best price:

- nRF24L01/2.4GHz RF

Arduino with nRF24L01

You need the following components to complete the instructions in this example:

Figure	Name	Check Price
	Arduino UNO	Find best price on Maker Advisor
	2x nRF24L01	Find best price on Maker Advisor
	Breadboard	Find best price on Maker Advisor
	Jumper Wires	Find best price on Maker Advisor

Library

For this project you need the RadioHead library.

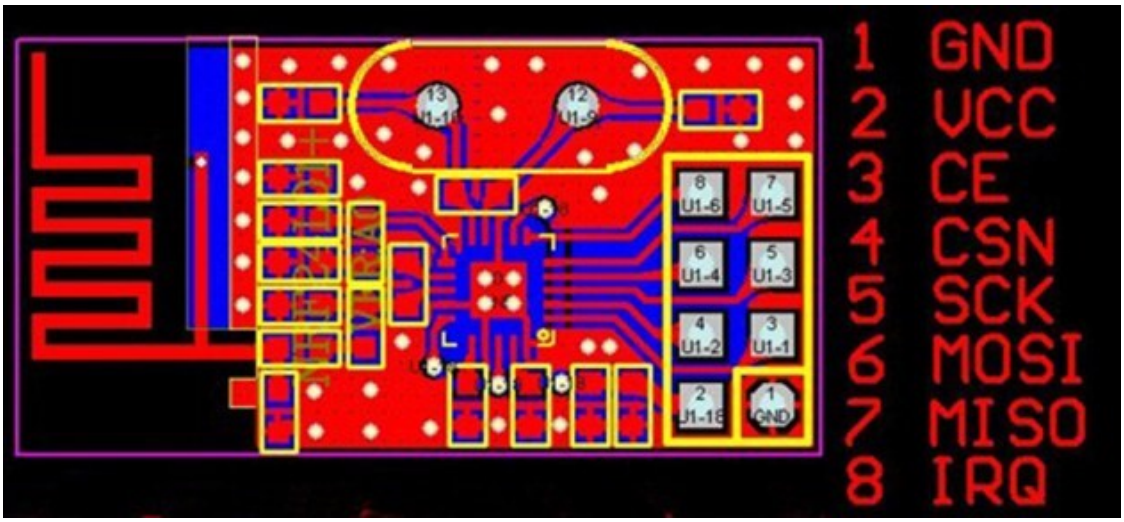
1. Click here to download the RadioHead library
2. Unzip the .zip folder and you should get RadioHead-1.46 folder.
3. Rename your folder from RadioHead-1.46 to RadioHead.
4. Move the RadioHead folder to your Arduino IDE installation libraries folder.
5. Finally, re-open your Arduino IDE.

The RadioHead library is great and it works with almost all RF modules in the market.

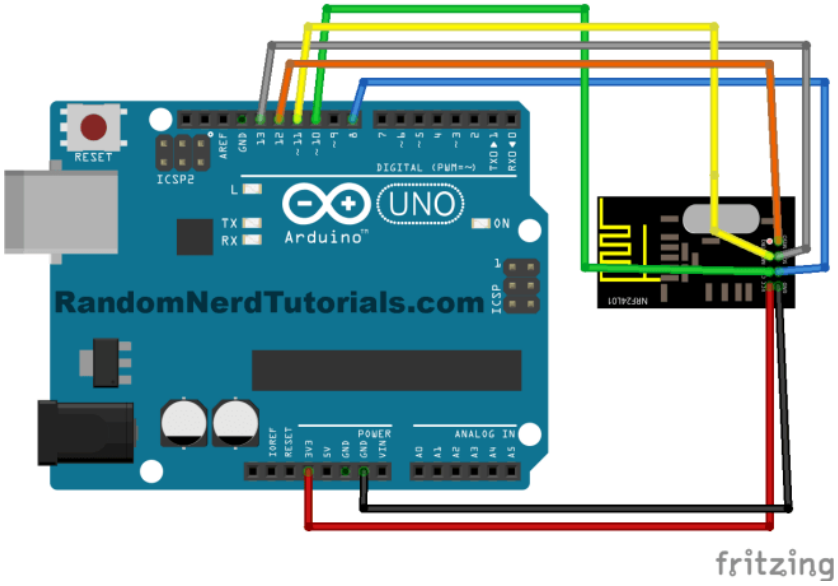
You can read more about this project here.

Pinout

Here's the top view of NRF24L01



Client circuit



WARNING

Input voltage is of 1.9V~3.6V, do not exceed this voltage, otherwise it will fry your module.

Follow the circuit above for your client. Then upload the code below which can be found in your Arduino IDE (after installing the RadioHead library).

Go to **File ▸ Examples ▸ RadioHead ▸ nrf24 ▸ nrf24_client**.

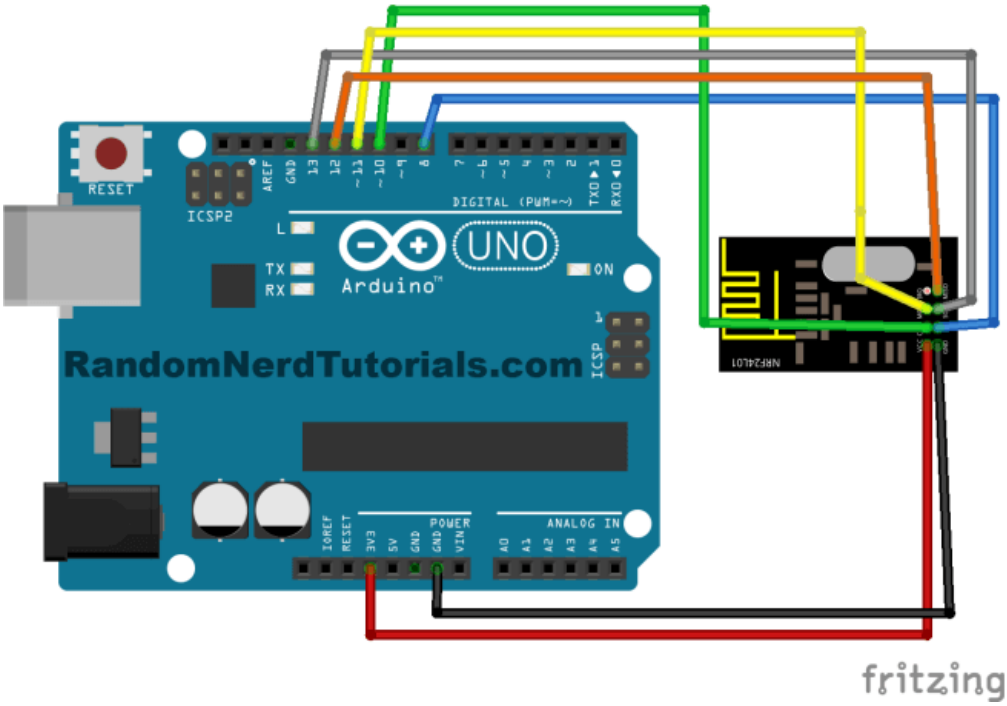
```
// nrf24_client
#include <SPI.h>
#include <RH_NRF24.h>

// Singleton instance of the radio driver
RH_NRF24 nrf24;
// RH_NRF24 nrf24(8, 7); // use this to be electrically compatible with Mirf
// RH_NRF24 nrf24(8, 10);// For Leonardo, need explicit SS pin
// RH_NRF24 nrf24(8, 7); // For RFM73 on Anarduino Mini

void setup()
{
  Serial.begin(9600);
  while (!Serial)
    ; // wait for serial port to connect. Needed for Leonardo only
  if (!nrf24.init())
    Serial.println("init failed");
  // Defaults after init are 2.402 GHz (channel 2), 2Mbps, 0dBm
  if (!nrf24.setChannel(1))
    Serial.println("setChannel failed");
  if (!nrf24.setRF(RH_NRF24::DataRate2Mbps, RH_NRF24::TransmitPower0dBm))
    Serial.println("setRF failed");
}


void loop()
{
  Serial.println("Sending to nrf24_server");
  // Send a message to nrf24_server
  uint8_t data[] = "Hello World!";
  nrf24.send(data, sizeof(data));

  nrf24.waitPacketSent();
  // Now wait for a reply
  uint8_t buf[RH_NRF24_MAX_MESSAGE_LEN];
  uint8_t len = sizeof(buf);
```

```
  if (nrf24.waitAvailableTimeout(500))
  {
    // Should be a reply message for us now
    if (nrf24.recv(buf, &len))
    {
      Serial.print("got reply: ");
      Serial.println((char*)buf);
    }
    else
    {
      Serial.println("recv failed");
    }
  }
  else
  {
    Serial.println("No reply, is nrf24_server running?");
  }
  delay(400);
}
```

SOURCE CODE

https://github.com/RuiSantosdotme/Random-Nerd-Tutorials/blob/master/Projects/nrf24_client.ino

Server circuit



WARNING

Input voltage is of 1.9V~3.6V, do not exceed this voltage, otherwise it will fry your module.

Follow the circuit above for your server. Then upload the code below which can be found in your Arduino IDE (after installing the RadioHead library).

Go to **File ▸ Examples ▸ RadioHead ▸ nrf24 ▸ nrf24_server**.

```
// nrf24_server

#include <SPI.h>
#include <RH_NRF24.h>

// Singleton instance of the radio driver
RH_NRF24 nrf24;
// RH_NRF24 nrf24(8, 7); // use this to be electrically compatible with Mirf
// RH_NRF24 nrf24(8, 10);// For Leonardo, need explicit SS pin
```

```
// RH_NRF24 nrf24(8, 7); // For RFM73 on Anarduino Mini

void setup()
{
  Serial.begin(9600);
  while (!Serial)
    ; // wait for serial port to connect. Needed for Leonardo only
  if (!nrf24.init())
    Serial.println("init failed");
  // Defaults after init are 2.402 GHz (channel 2), 2Mbps, 0dBm
  if (!nrf24.setChannel(1))
    Serial.println("setChannel failed");
  if (!nrf24.setRF(RH_NRF24::DataRate2Mbps, RH_NRF24::TransmitPower0dBm))
    Serial.println("setRF failed");
}

void loop()
{
  if (nrf24.available())
  {
    // Should be a message for us now
    uint8_t buf[RH_NRF24_MAX_MESSAGE_LEN];
    uint8_t len = sizeof(buf);
    if (nrf24.recv(buf, &len))
    {
//      NRF24::printBuffer("request: ", buf, len);
      Serial.print("got request: ");
      Serial.println((char*)buf);

      // Send a reply
      uint8_t data[] = "And hello back to you";
      nrf24.send(data, sizeof(data));
      nrf24.waitPacketSent();
      Serial.println("Sent a reply");
    }
    else
    {
      Serial.println("recv failed");
    }
  }
}
```

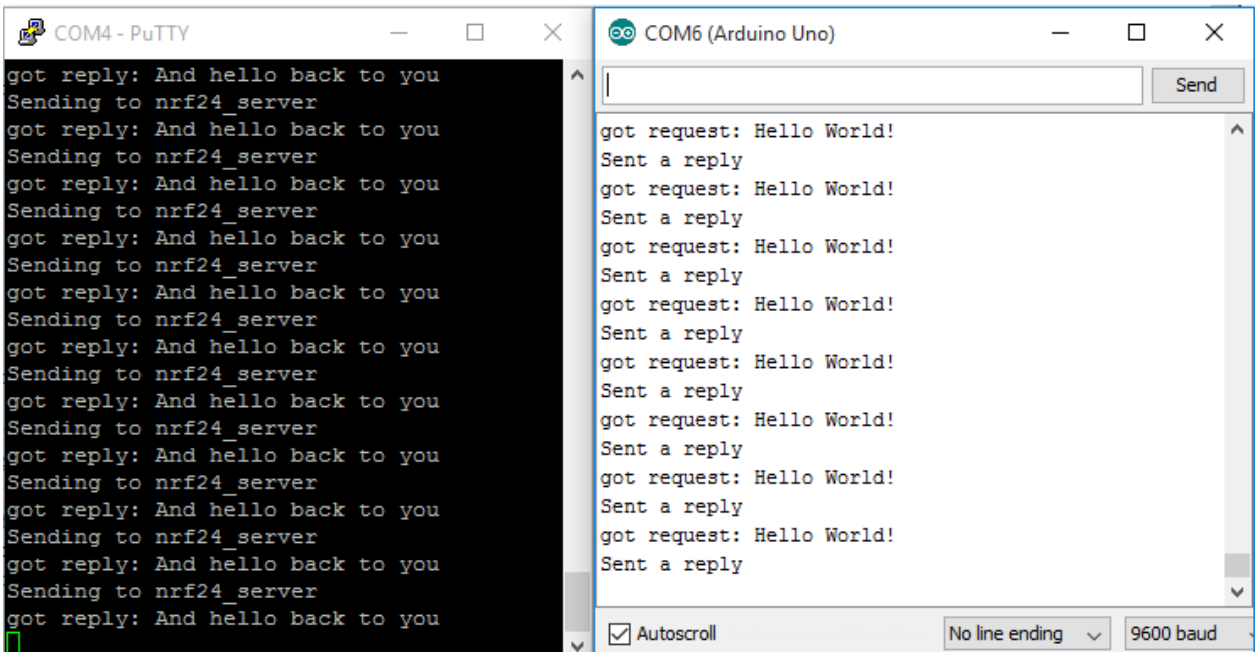
SOURCE CODE

https://github.com/RuiSantosdotme/Random-Nerd-Tutorials/blob/master/Projects/nrf24_server.ino

Demonstration

In this project the client is sending a message "Hello World!" to the server via RF and the server is sending back the following message "And hello back to you". Those messages are being displayed in the serial monitor. Here's what you should see in your serial monitor and terminal windows (see Figure below).

Client Circuit **Server Circuit**

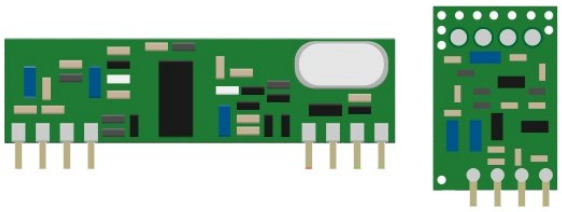


Note: On the left window, I'm establishing a serial communication with PuTTY.org. On the right window, I'm using the Arduino IDE Serial Monitor.

Wrapping up

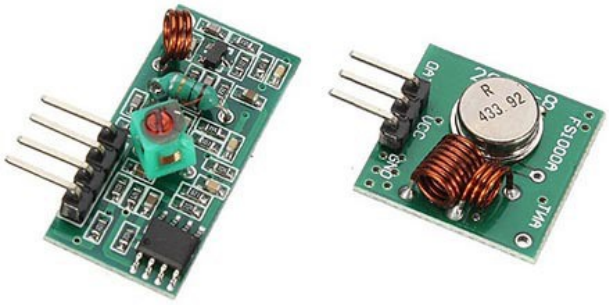
You need to have some realistic expectations when using this module. They work very well when the receiver and transmitter are quite close to each other. If you separate them too far you'll lose the communication.

The communication range will vary. It depends on how much noise in your environment, if there's any obstacles and if you're using an external antenna.



433 MHz Transmitter/Receiver

These RF modules are very popular among the Arduino tinkerers. The 433MHz is used on a wide variety of applications that require wireless control.



These modules are very cheap and you can use them with any microcontroller (MCU).

Specifications RF 433MHz Receiver

- Frequency Range: 433.92 MHz
- Modulation: ASK
- Input Voltage: 5V
- Price: $1 to $2

Specifications RF 433MHz Transmitter

- Frequency Range: 433.92MHz
- Input Voltage: 3-12V
- Price: $1 to $2

Where to buy?

Click the link below to compare the sensor at different stores and find the best price:

- 433 MHz Transmitter/Receiver

Arduino with RF 433MHz Modules

This is a simple example to transmit a message via RF.

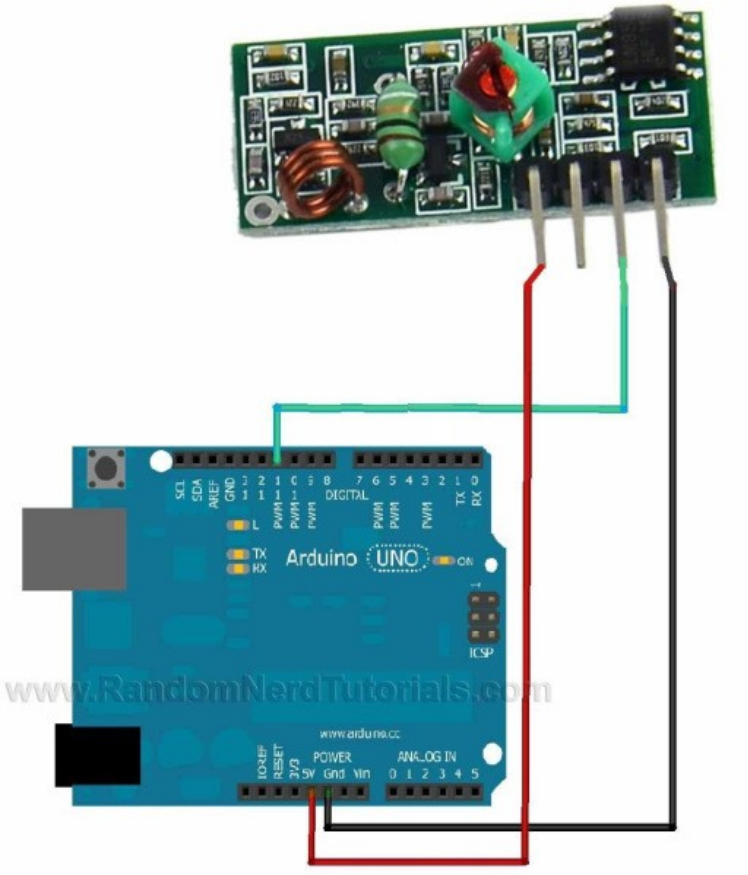
Library download

Here's the library you need for this project:

1. Download the RadioHead library
2. Unzip the RadioHead library
3. Install the RadioHead library in your Arduino IDE
4. Restart your Arduino IDE

The RadioHead library is great and it works with almost all RF modules in the market. You can read more about this project here.

Receiver circuit



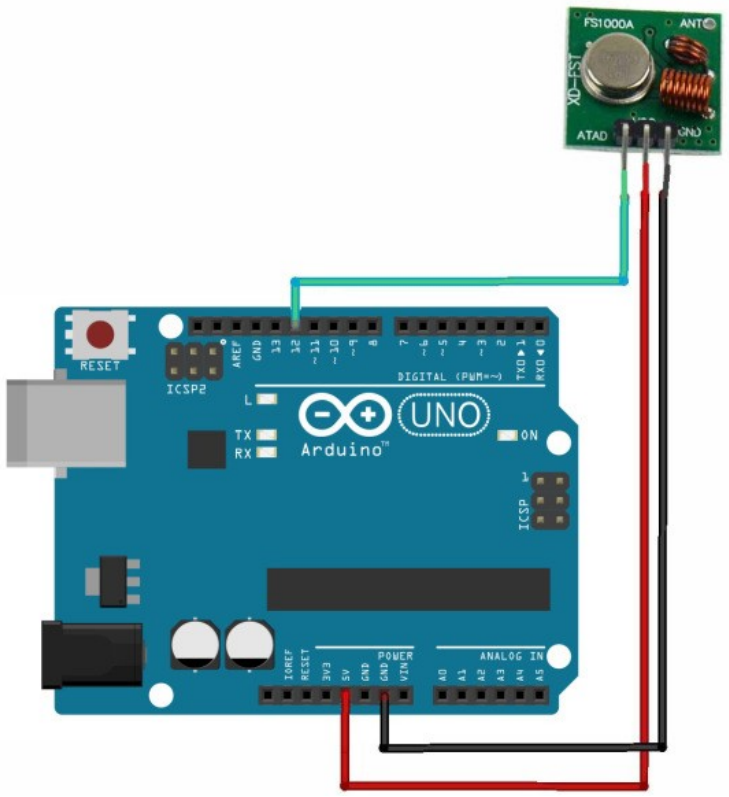
Follow the circuit above for your receiver. Then upload the code below.

```
#include <RH_ASK.h>
#include <SPI.h> // Not actualy used but needed to compile

RH_ASK driver;

void setup()
{
    Serial.begin(9600); // Debugging only
    if (!driver.init())
         Serial.println("init failed");
}

void loop()
{
    uint8_t buf[12];
    uint8_t buflen = sizeof(buf);
    if (driver.recv(buf, &buflen)) // Non-blocking
    {
      int i;
      // Message with a good checksum received, dump it.
      Serial.print("Message: ");
      Serial.println((char*)buf);
    }
}
```

SOURCE CODE

https://github.com/RuiSantosdotme/Random-Nerd-Tutorials/blob/master/Projects/433MHz/receiver.ino

Transmitter circuit



Follow the circuit above for your transmitter. Then upload the code below.

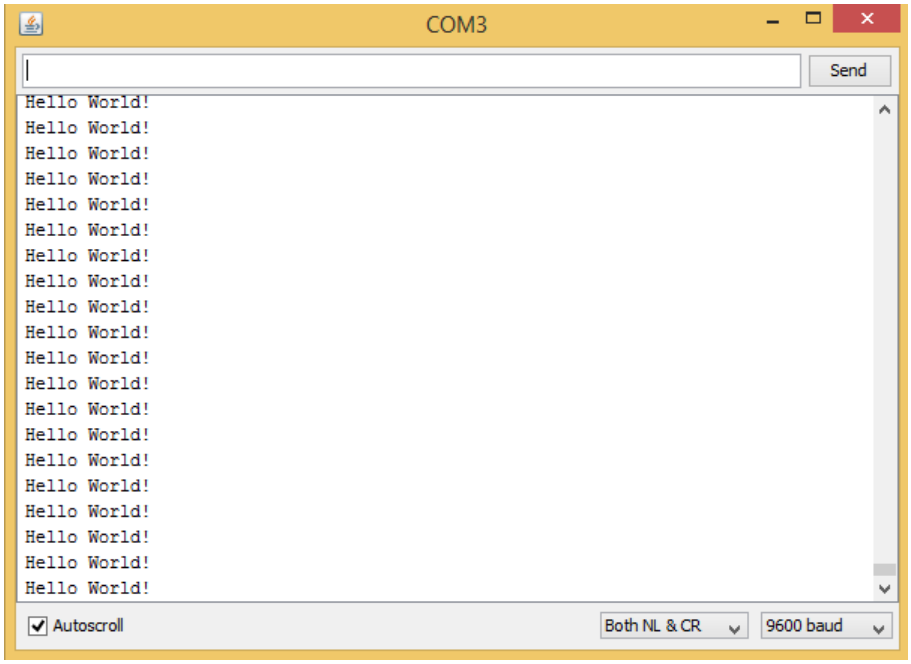
```
#include <RH_ASK.h>
#include <SPI.h> // Not actually used but needed to compile

RH_ASK driver;

void setup()
{
    Serial.begin(9600);    // Debugging only
    if (!driver.init())
         Serial.println("init failed");
}

void loop()
{
    const char *msg = "Hello World!";
    driver.send((uint8_t *)msg, strlen(msg));
    driver.waitPacketSent();
    delay(1000);
}
```

SOURCE CODE

https://github.com/RuiSantosdotme/Random-Nerd-Tutorials/blob/master/Projects/433MHz/transmitter.ino

Demonstration

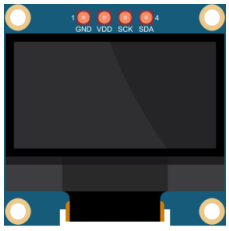
In this project the transmitter is sending a message "Hello World!" to the receiver via RF. Those messages are being displayed in the serial monitor from the receiver. Here's what you should see in your Arduino IDE serial monitor.



Wrapping up

You need to have some realistic expectations when using this module. They work very well when the receiver and transmitter are close to each other. If you separate them too far you'll lose the communication.

The communication range will vary. It depends on how much voltage that you're supplying to your transmitter module, RF noise in your environment and if you're using an external antenna.



OLED Display

The OLED display is shown in the following figure:

It is a very small display made of 128 by 64 individual OLED pixels and no backlight is required. That OLED display is monochrome (white color), but there are other models with several colors.

This display uses I2C communication. This means that it communicates with the Arduino using just 2 pins.

Where to buy?

Click the link below to compare the sensor at different stores and find the best price:

- 0.96 inch OLED display

Pin wiring

OLED Pin	Wiring to Arduino Uno
Vin	5V
GND	GND
SCL	A5
SDA	A4

If you're using other Arduino board rather than the UNO, check out what are their SCL and SDA pins.

- Nano: SDA (A4); SCL(A5);
- MEGA: SDA (20); SCL(21);
- Leonardo: SDA (20); SCL(21);

Libraries

To control the OLED display you'll need the "**adafruit_GFX.h**" library and the "**adafruit_SSD1306.h**" library.

Installing the adafruit_GFX library

1. Click here to download the adafruit GFX library. You should have a .zip folder in your Downloads folder
2. Unzip the .zip folder and you should get Adafruit-GFX-Library-master folder
3. Rename your folder from ~~Adafruit-GFX-Library-master~~ to Adafruit_GFX_Library.
4. Move the Adafruit_GFX_Library folder to your Arduino IDE installation libraries folder
5. Finally, re-open your Arduino IDE

Installing the adafruit_SSD1306 library

1. Click here to download the adafruit_SSD1306 library. You should have a .zip folder in your Downloads folder
2. Unzip the .zip folder and you should get Adafruit-GFX-Library-master folder
3. Rename your folder from ~~Adafruit_SSD1306-master~~ to Adafruit_SSD1306

4. Move the Adafruit_SSD1306 folder to your Arduino IDE installation libraries folder
5. Finally, re-open your Arduino IDE

Tips for writing text using these libraries

Here's some functions that will help you handle the OLED display library to write text or draw simple graphics.

- `display.clearDisplay()` – all pixels are off
- `display.drawPixel(x,y, color)` – plot a pixel in the x,y coordinates
- `display.setTextSize(n)` – set the font size, supports sizes from 1 to 8
- `display.setCursor(x,y)` – set the coordinates to start writing text
- `display.print("message")` – print the characters at location x,y

Example: Display temperature and humidity

In this example you will display the temperature and humidity in the OLED display. The aim of this project is to get familiar with the OLED display.

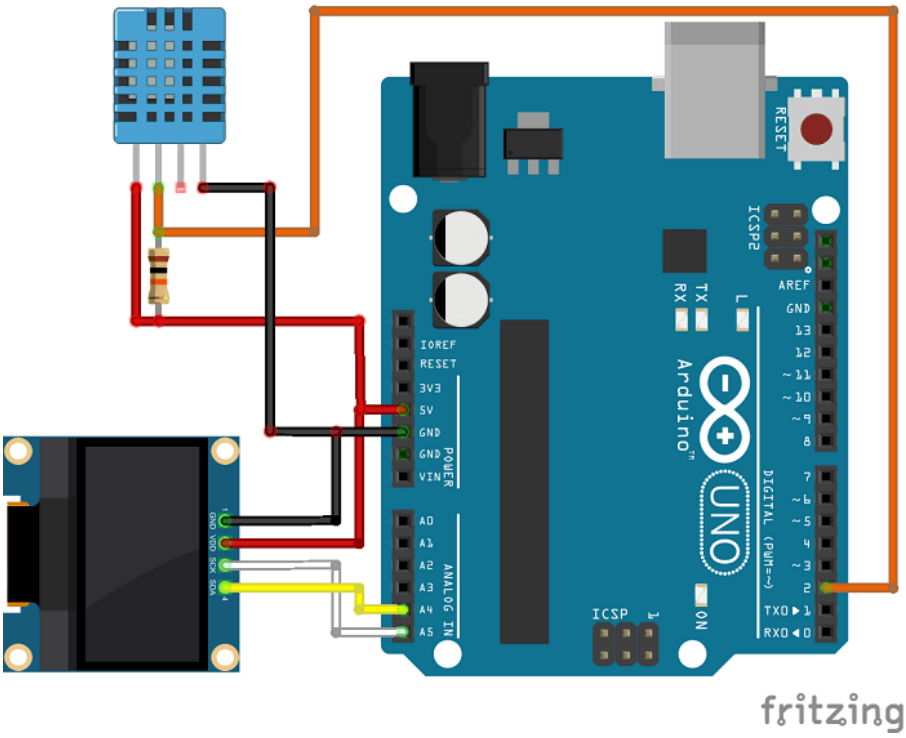
The temperature and humidity will be measured using the DHT11 temperature and humidity sensor. If you're not familiar with the DHT11 sensor we recommend that you check the DHT11/22 guide.

Figure	Name	Check Price
	Arduino UNO	Find best price on Maker Advisor
	OLED Display	Find best price on Maker Advisor
	DHT11	Find best price on Maker Advisor
	Breadboard	Find best price on Maker Advisor
	10kΩ Resistor	Find best price on Maker Advisor

	Jumper Wires	Find best price on Maker Advisor

Schematics

Assemble all the parts as in the schematics below.



Code

Make sure you've installed the necessary libraries to control the OLED display. You also need to install the DHT library. Check the DHT11/22 temperature and humidity sensor guide.

Then, you can upload the following code.

```
/*
 * Random Nerd Tutorials - Rui Santos
 * Complete Project Details http://randomnerdtutorials.com
 */

#include <Wire.h>
```

```
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#include <DHT.h>

#define DHTPIN 2     // what pin we're connected to
#define DHTTYPE DHT11   // DHT 11
#define OLED_RESET 4
Adafruit_SSD1306 display(OLED_RESET);

// Initialize DHT sensor for normal 16mhz Arduino
DHT dht(DHTPIN, DHTTYPE);

void setup()
{
  Wire.begin();
  dht.begin(); // initialize dht
  display.begin(SSD1306_SWITCHCAPVCC, 0x3C);// initialize with the I2C addr
0x3C (for the 128x32)(initializing the display)
  Serial.begin(9600);
}
void displayTempHumid(){
  delay(2000);
  // Reading temperature or humidity takes about 250 milliseconds!
  // Sensor readings may also be up to 2 seconds 'old' (its a very slow
sensor)
  float h = dht.readHumidity();
  // Read temperature as Celsius
  float t = dht.readTemperature();
  // Read temperature as Fahrenheit
  float f = dht.readTemperature(true);

  // Check if any reads failed and exit early (to try again).
  if (isnan(h) || isnan(t) || isnan(f)) {
    display.clearDisplay(); // clearing the display
    display.setTextColor(WHITE); //setting the color
    display.setTextSize(1); //set the font size
    display.setCursor(5,0); //set the cursor coordinates
    display.print("Failed to read from DHT sensor!");
    return;
  }
  display.clearDisplay();
```

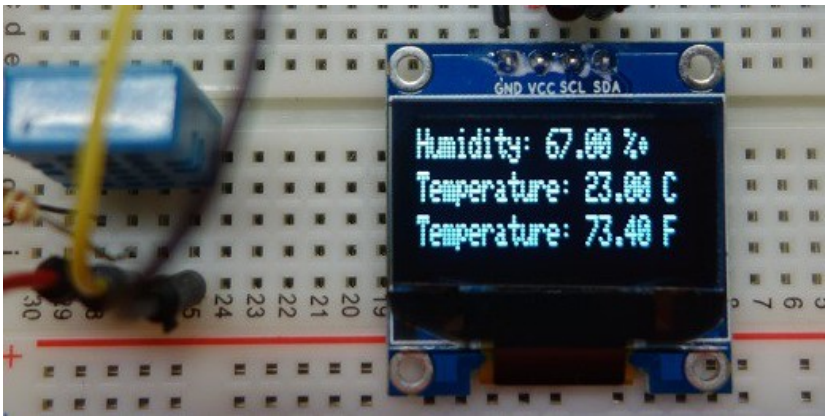
```
  display.setTextColor(WHITE);
  display.setTextSize(1);
  display.setCursor(0,0);
  display.print("Humidity: ");
  display.print(h);
  display.print(" %\t");
  display.setCursor(0,10);
  display.print("Temperature: ");
  display.print(t);
  display.print(" C");
  display.setCursor(0,20);
  display.print("Temperature: ");
  display.print(f);
  display.print(" F");
}
void loop()
{
  displayTempHumid();
  display.display();
}
```

SOURCE CODE

https://github.com/RuiSantosdotme/Random-Nerd-Tutorials/blob/master/Projects/display_tempreature_and_humidity_in_oled.ino

Demonstration

Here's the OLED display in action after uploading the code.

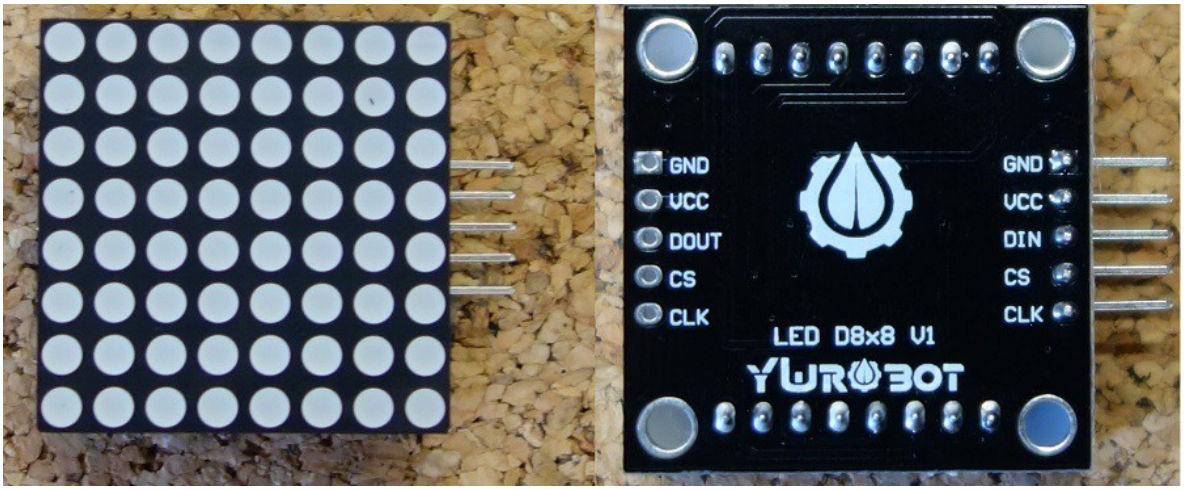


8x8 Dot Matrix

The dot matrix that we're going to use in this guide is an 8×8 matrix which means that it has 8 columns and 8 rows, so it contains a total of 64 LEDs.

The MAX7219 chip makes it easier to control the dot matrix, by just using 3 digital pins of the Arduino board.

I think the best option is to buy the dot matrix with the MAX7219 chip as a module, it will simplify the wiring.



You can control more than one matrix at a time. For that you just need to connect them to each other, as they have pins in both sides to extend the dot matrix.

Where to buy?

Click the link below to compare the matrix at different stores and find the best price:

- MAX7219 LED matrix

Pin wiring

You only need to connect 5 pins from the dot matrix to your Arduino board. The wiring is pretty straightforward:

Dot Matrix Pin	Wiring to Arduino Uno
GND	GND
VCC	5V
DIN	Digital pin
CS	Digital pin
CLK	Digital pin

Libraries

For making it easier to control the dot matrix, you need to download and install in your Arduino IDE the **LedControl** library. To install the library follow these steps:

1. Click here to download the LedControl library. You should have a .zip folder in your Downloads
2. Unzip the .zip folder and you should get LedControl-master folder
3. Rename your folder from LedControl-master to LedControl
4. Move the LedControl folder to your Arduino IDE installation libraries folder
5. Finally, re-open your Arduino IDE

Using the LedControl library functions

The easiest way to display something on the dot matrix is by using the functions `setLed()`, `setRow()` or `setColumn()`. These functions allow you to control one single led, one row or one column at a time. Here's the parameters for each function:

```
setLed(addr, row, col, state)
```

- **addr** is the address of your matrix, for example, if you have just 1 matrix, the addr will be zero.

- **row** is the row where the led is located
- **col** is the column where the led is located
- **state**
    - It's true or 1 if you want to turn the led on
    - It's false or 0 if you want to switch it off

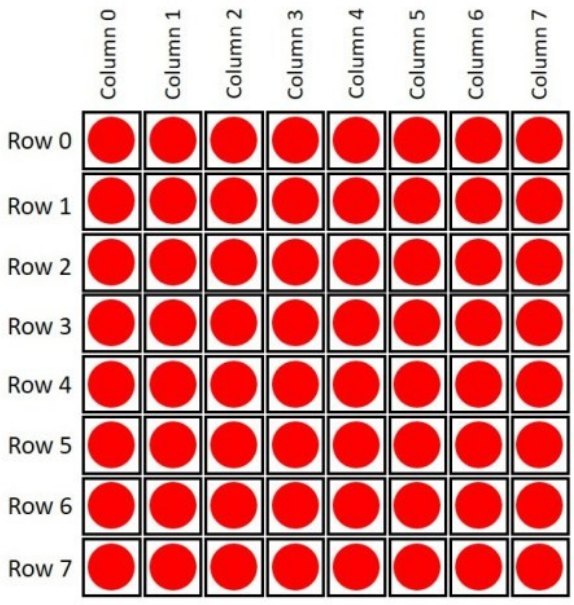
```
setRow(addr, row, value)
setCol(addr, column, value)
```

Example: Displaying icons on the dot matrix

In this example, you'll display several faces sequentially on a dot matrix: a happy, a neutral, and a sad face.

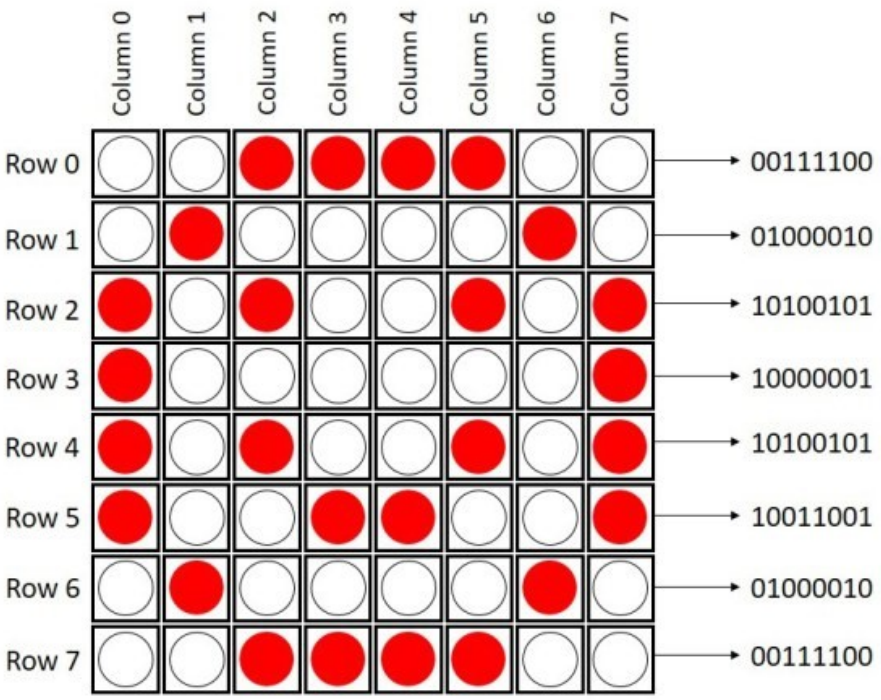
Index

As previously stated, this matrix has 8 columns and 8 rows. Each one is indexed from 0 to 7. Here's a figure for better understanding:



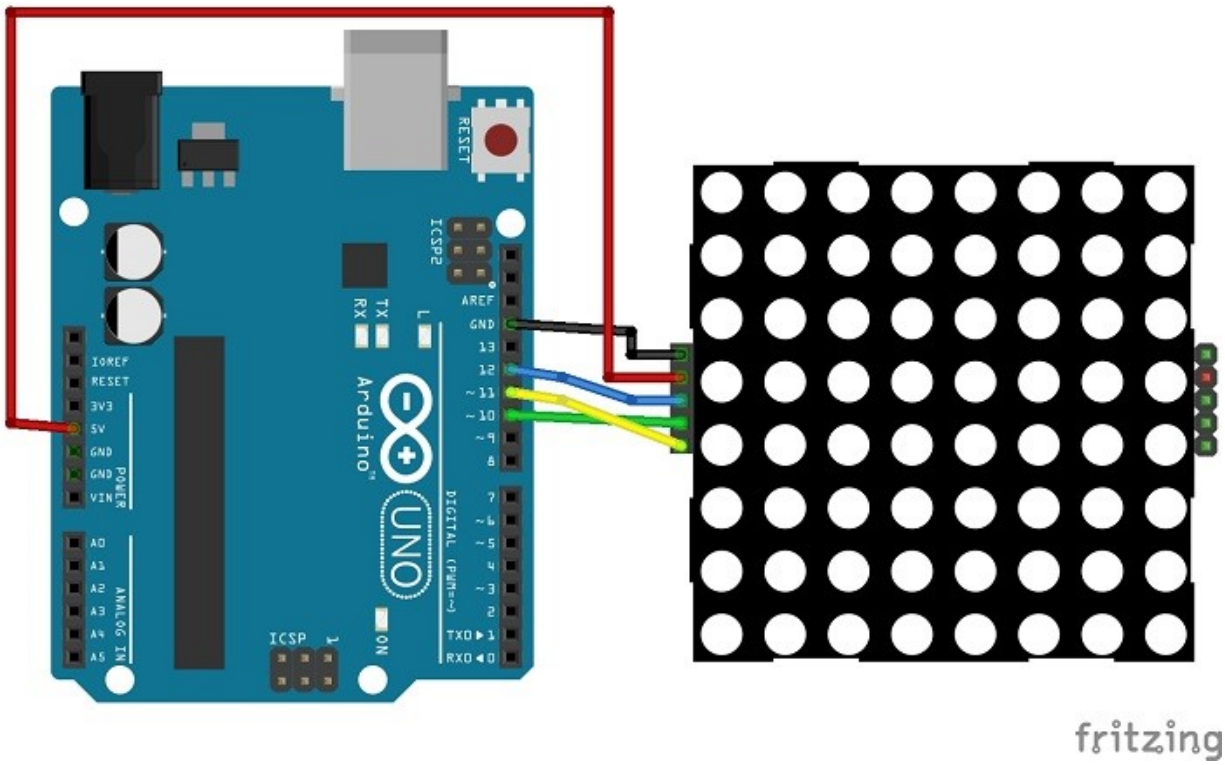
If you want to display something in the matrix, you just need to know if in a determined row or column, the LEDs that are on or off.

For example, if you want to display a happy face, here's what you need to do:



Schematic

Connect the dot matrix as in the diagram below.



Here's a simple sketch that displays three types of faces: a sad face, a neutral face and a happy face. Upload the following code to your board:

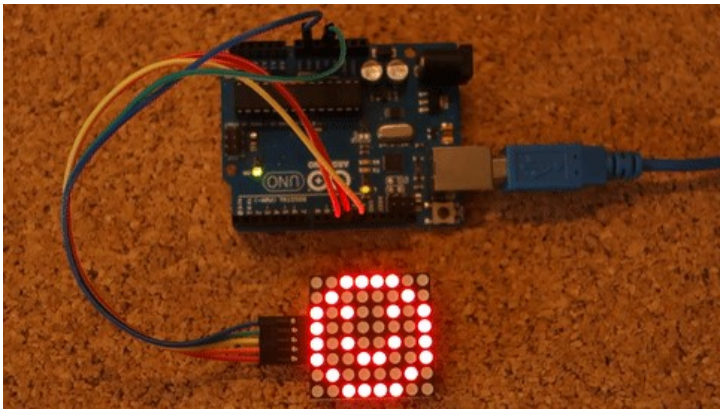
```
/*
 Created by Rui Santos

 All the resources for this project:
 http://randomnerdtutorials.com/
*/

#include "LedControl.h"
#include "binary.h"

/*
 DIN connects to pin 12
 CLK connects to pin 11
 CS connects to pin 10
*/
LedControl lc=LedControl(12,11,10,1);

// delay time between faces
unsigned long delaytime=1000;

// happy face
byte hf[8]=
{B00111100,B01000010,B10100101,B10000001,B10100101,B10011001,B01000010,B00111
100};
// neutral face
byte nf[8]={B00111100,
B01000010,B10100101,B10000001,B10111101,B10000001,B01000010,B00111100};
// sad face
byte sf[8]=
{B00111100,B01000010,B10100101,B10000001,B10011001,B10100101,B01000010,B00111
100};

void setup() {
  lc.shutdown(0,false);
  // Set brightness to a medium value
  lc.setIntensity(0,8);
  // Clear the display
  lc.clearDisplay(0);
```

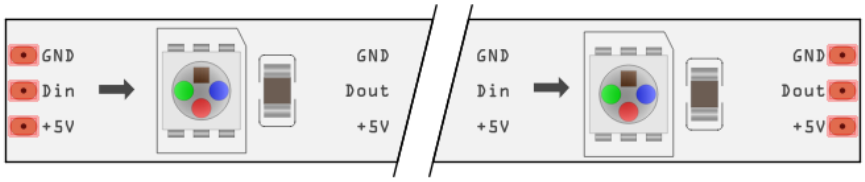
```
}

void drawFaces(){
  // Display sad face
  lc.setRow(0,0,sf[0]);
  lc.setRow(0,1,sf[1]);
  lc.setRow(0,2,sf[2]);
  lc.setRow(0,3,sf[3]);
  lc.setRow(0,4,sf[4]);
  lc.setRow(0,5,sf[5]);
  lc.setRow(0,6,sf[6]);
  lc.setRow(0,7,sf[7]);
  delay(delaytime);

  // Display neutral face
  lc.setRow(0,0,nf[0]);
  lc.setRow(0,1,nf[1]);
  lc.setRow(0,2,nf[2]);
  lc.setRow(0,3,nf[3]);
  lc.setRow(0,4,nf[4]);
  lc.setRow(0,5,nf[5]);
  lc.setRow(0,6,nf[6]);
  lc.setRow(0,7,nf[7]);
  delay(delaytime);

  // Display happy face
  lc.setRow(0,0,hf[0]);
  lc.setRow(0,1,hf[1]);
  lc.setRow(0,2,hf[2]);
  lc.setRow(0,3,hf[3]);
  lc.setRow(0,4,hf[4]);
  lc.setRow(0,5,hf[5]);
  lc.setRow(0,6,hf[6]);
  lc.setRow(0,7,hf[7]);
  delay(delaytime);
}

void loop(){
  drawFaces();
}
```

SOURCE CODE

https://github.com/RuiSantosdotme/Random-Nerd-Tutorials/blob/master/Projects/dot_matrix_faces.ino

In the end, you'll have something like this:

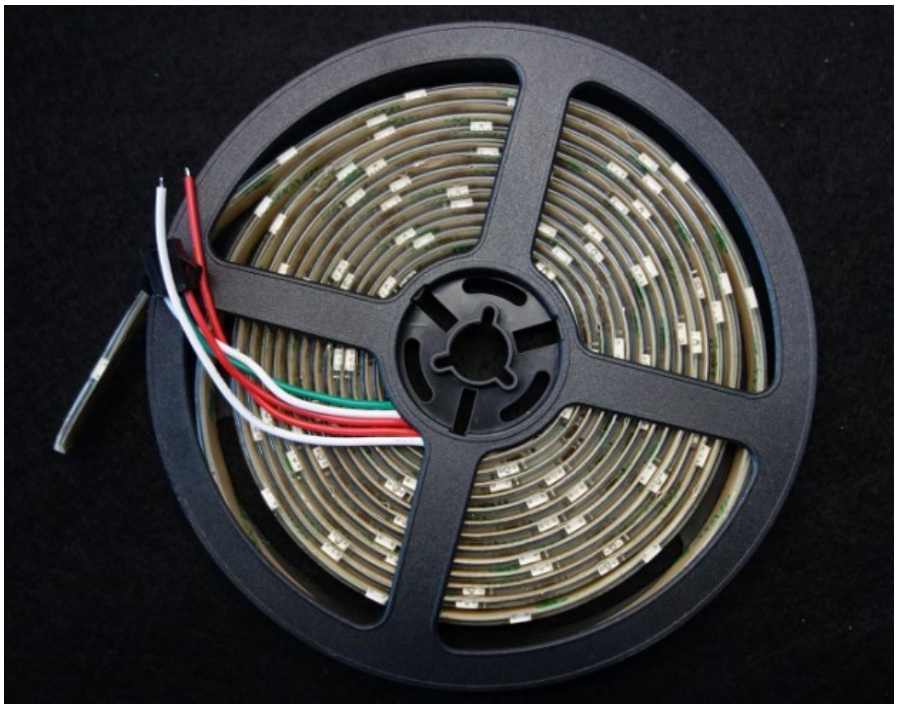




WS1812B Addressable RGB LED Strip

The WS2812B LED strip is an addressable RGB LED strip. The information in this guide also works with other similar LED strips, such as strips of the WS28XX family, Neopixel strip and others.

The WS2812B addressable LED strip comes in several varieties that differ in size, sealant or LED density. Choose the one that best fits your purposes.



Where to buy?

Click the link below to compare the LED strip at different stores and find the best price:

- WS2812B addressable RGB LED Strip

WS1812B Addressable RGB LED strip

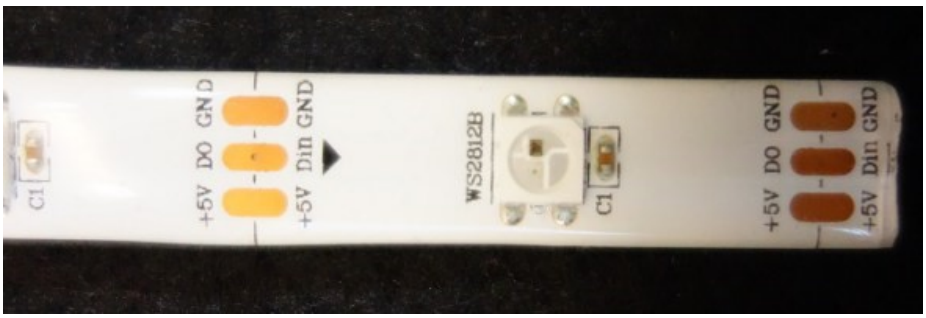
In the figure above you can see my WS2812B LED strip. It is 5 meters long and the LEDs are enclosed in a weatherproof silicone.

In my opinion, this is the coolest type of LED strips. You can control the brightness and the color of each LED individually, which allows you to produce amazing and complex effects in a simple way.

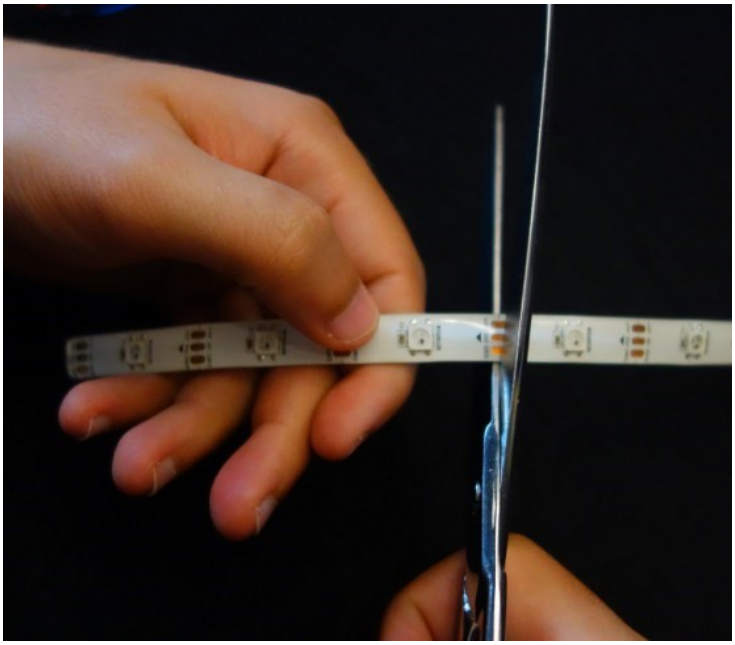
This LED strip is made by WS2812B LEDs wired in series. These LEDs have an IC built right into the LED. This allows a communication via a one-wire interface. This means that you can control lots of LEDs using just one digital pin of your Arduino.

In the following figure you can see the chip inside the LED. The LED is an RGB LED and works like so.

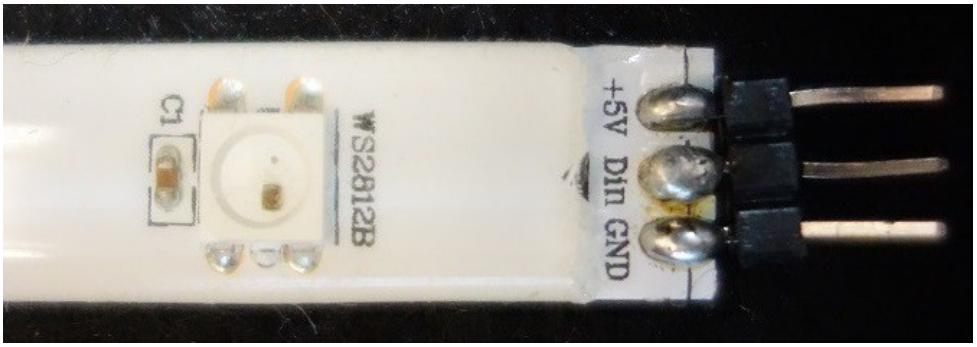
This kind of strips are very flexible and can be cut to any length you want. As you can see, the strip is divided into segments, and each segment contains one RGB LED.



You can adjust its size by cutting the strip with a scissors in the right place (the proper places to cut the strip are marked).



These strips come with connectors at each end. I've decided to cut the connectors, and solder header pins. It's handier if you want to connect the strip to an Arduino or to a breadboard.



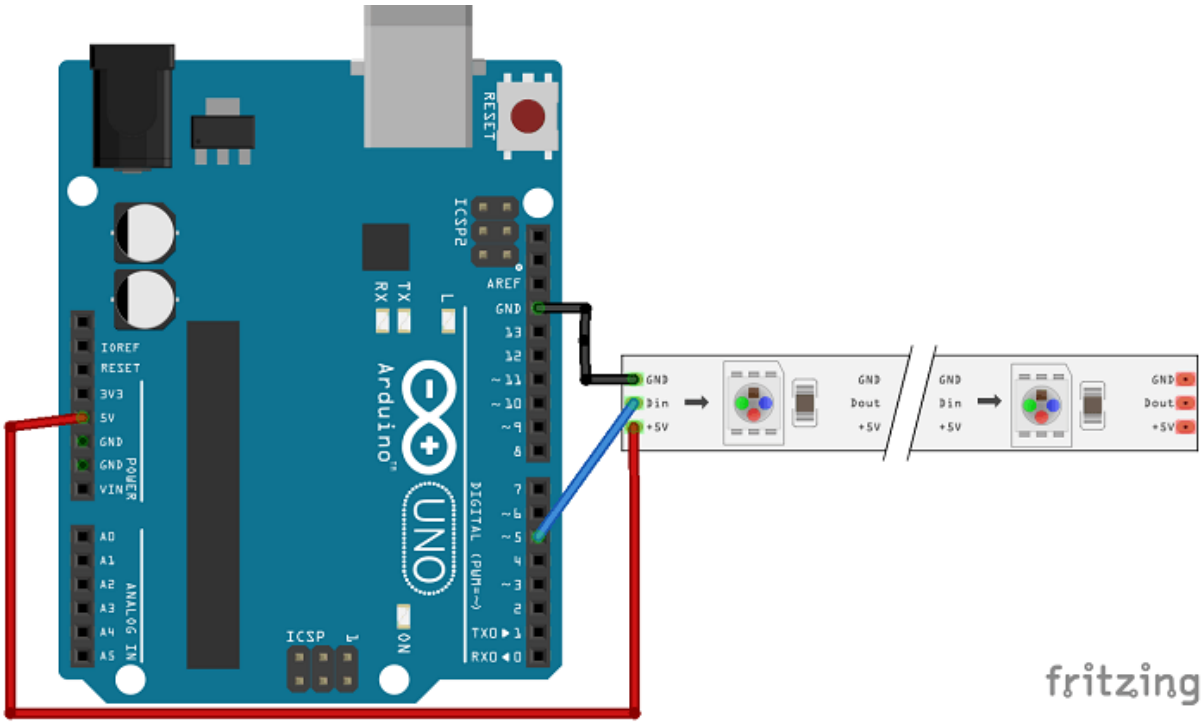
Powering the WS2812B LED Strip

The LED strip should be powered using a 5V power source. At 5V, each LED draws about 50mA, when set to its full brightness. This means that for every 30 LEDs, the strip may draw as much as 1.5 A. Make sure you select a power source that matches the strip's needs.

If you end up using an external supply, don't forget to connect the power supply ground to the Arduino ground.

Schematics

In this example, the WS2812B LED strip will be powered using the 5V Arduino pin. In my case, I'm controlling 14 LEDs. Bear in mind that if you want to control many LEDs, you'll need to use an external power supply.



Useful tips:

- Connect a capacitor with a capacitance between 100uF and 1000uF from power to ground to smooth out the power supply.
- Add a 220 or 470 Ohm resistor between the Arduino digital output pin and the strip data input pin to reduce noise on that line.
- Make your wires between the Arduino, power supply and the strip as short as possible to minimize voltage loss.
- If your strip gets damaged and doesn't work, check if the first LED is broken. If so, cut it, resolder the header pins, and it should be working again.

Code

To control the WS2812B LED strip, you'll need to download the **FastLED** library.

Installing the FastLED library

1. Click here to download the FastLED library. You should have a .zip folder in your Downloads folder
2. Unzip the .zip folder and you should get FastLED-master folder
3. Rename your folder from FastLED-master to FastLED
4. Move the FastLED folder to your Arduino IDE installation libraries folder
5. Finally, re-open your Arduino IDE

After installing the needed library, upload the following code to your Arduino board (this is an example sketch provided in the library examples folder). Go to **File ▸Examples ▸ FastLED ▸ ColorPalette** or copy the code below.

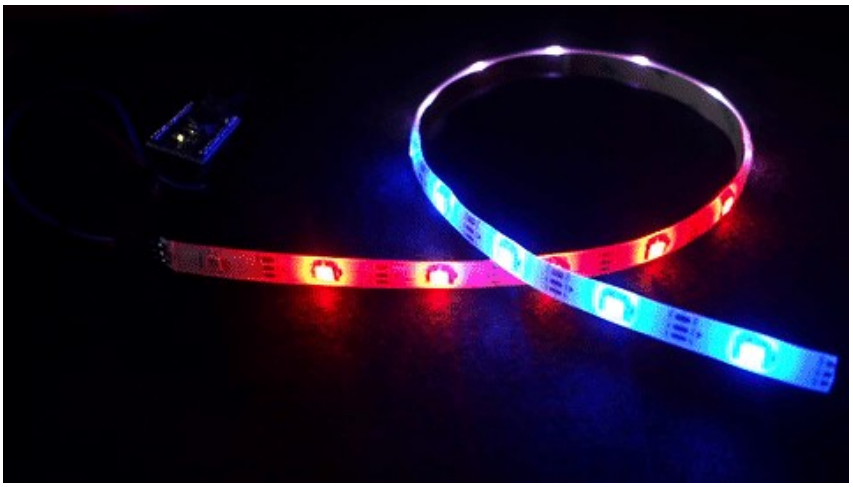
Note: you have to change `#define NUM_LEDS` to the number of LEDs that your strip currently has.

{;} SOURCE CODE

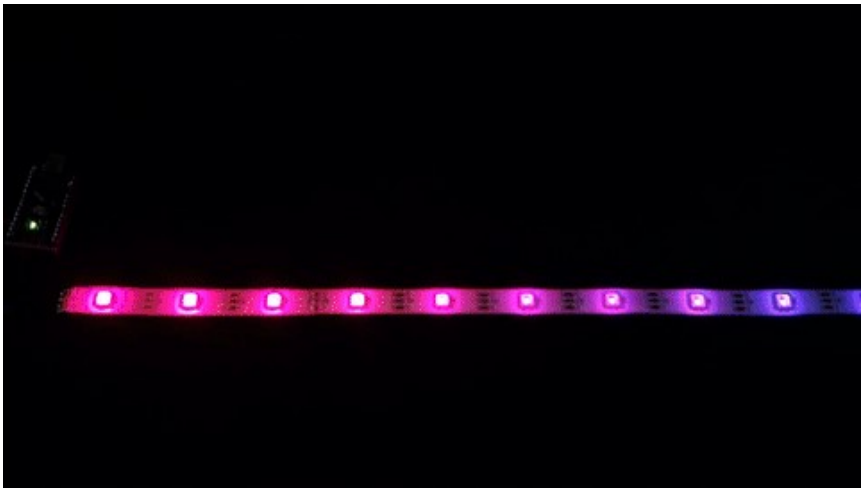
https://github.com/RuiSantosdotme/Random-Nerd-Tutorials/blob/master/Projects/Arduino_WS2812B_Color_Palette.ino

Demonstration

In the end, this is what you'll have. Amazing effects like this one:



Or this one:



And so on (…)

Using an LED Strip Case

These strips usually come with a removable tape, so that you can stick them wherever you want. The problem is that they don't stick very well, so chances are that you'll find your strip in the floor the following day.

The solution: I found this strip case that diffuses the light well and you can screw it to a shelf, for example, if you want a permanent solution.

Membrane Keypad

A keypad allows you to interact with a microcontroller. You can salvage these keypads from old telephones or you can purchase them from most electronics stores for less than $2. They come in wide variety of shapes and sizes. The most common sizes are 3×4 and 4×4 and you can get keypads with words, letters and numbers written on the keys.

These keypads very popular among the Arduino tinkerers. They are very cheap and you can use them with any microcontroller (MCU). You can even create your own keypad from scratch.

Where to buy?

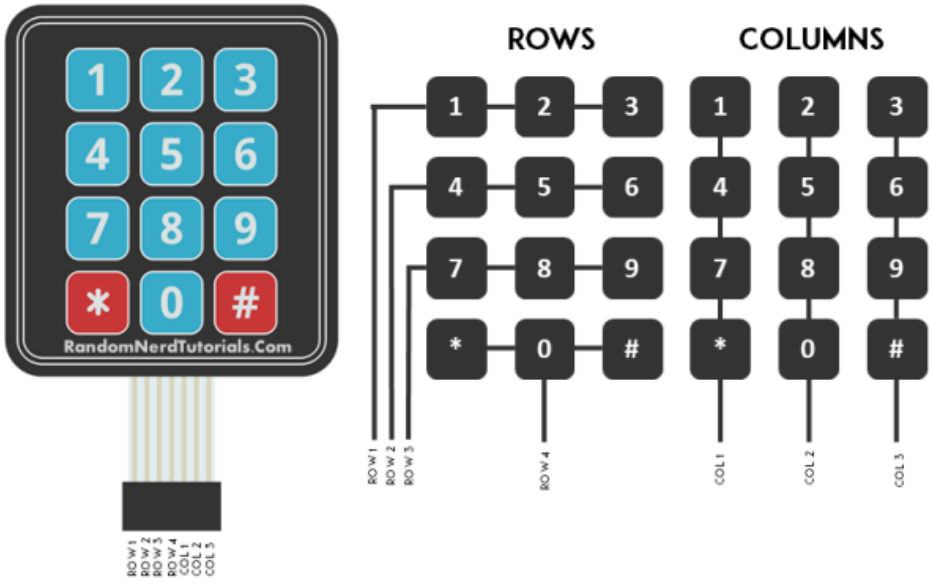
Click the link below to compare the sensor at different stores and find the best price:

- Membrane keypad

How it works?

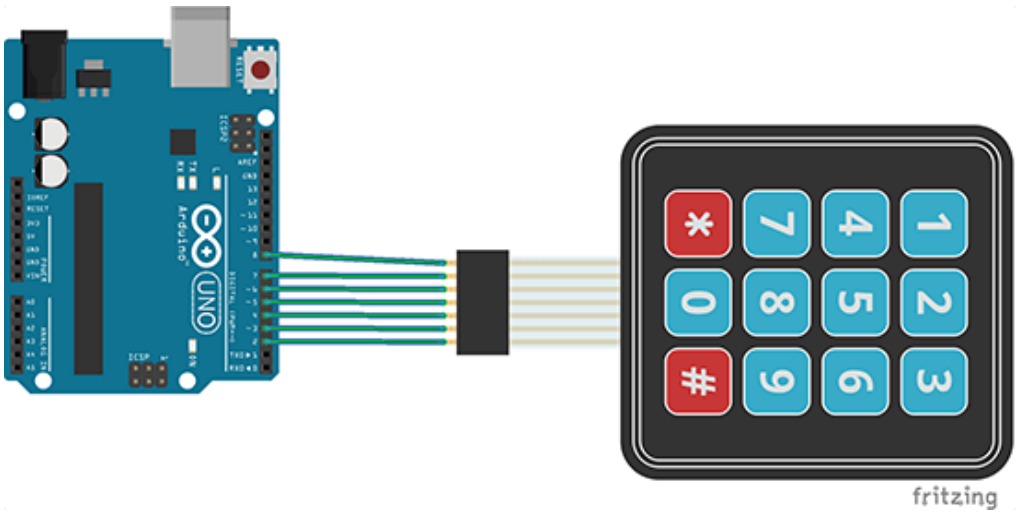
A membrane keypad is a matrix consisting of rows and columns. Each key is assigned to a certain row and column (see the picture below).

On a 12 button keypad you have 4 rows and 3 columns. The first key would make a link between Row 1 and Column 1 (R1C1). 2 would be R1C2, 3 R1C3, * R4C1, 9 R3C3 and so on.



Schematic

Follow the next schematics. If your keypad is different from the one below, search for the datasheet online.



Library download

1. Click here to download the Keypad library . You should have a .zip folder in your Downloads folder
2. Unzip the .zip folder and you should get Keypad folder
3. Move the Keypad folder to your Arduino IDE installation libraries folder
4. Finally, re-open your Arduino IDE

Code

If your keypad doesn't work with code below you might have to change the connections from the previous schematics. You have to make sure you follow your keypad's datasheet.

Note: If your keypad has more keys you can change lines 3 and 4 to add the right number of rows and columns. Then in line 5 you can change the array to match your keypad keys.

```
#include "Keypad.h"

const byte ROWS = 4; // number of rows
const byte COLS = 3; // number of columns
char keys[ROWS][COLS] = {
{'1','2','3'},
{'4','5','6'},
{'7','8','9'},
{'#','0','*'}
};

byte rowPins[ROWS] = {8, 7, 6, 5}; // row pinouts of the keypad R1 = D8, R2 =
D7, R3 = D6, R4 = D5
byte colPins[COLS] = {4, 3, 2};    // column pinouts of the keypad C1 = D4,
C2 = D3, C3 = D2
Keypad keypad = Keypad(makeKeymap(keys), rowPins, colPins, ROWS, COLS);

void setup(){
  Serial.begin(9600);
}
void loop(){
  char key = keypad.getKey();
  if (key != NO_KEY)
```

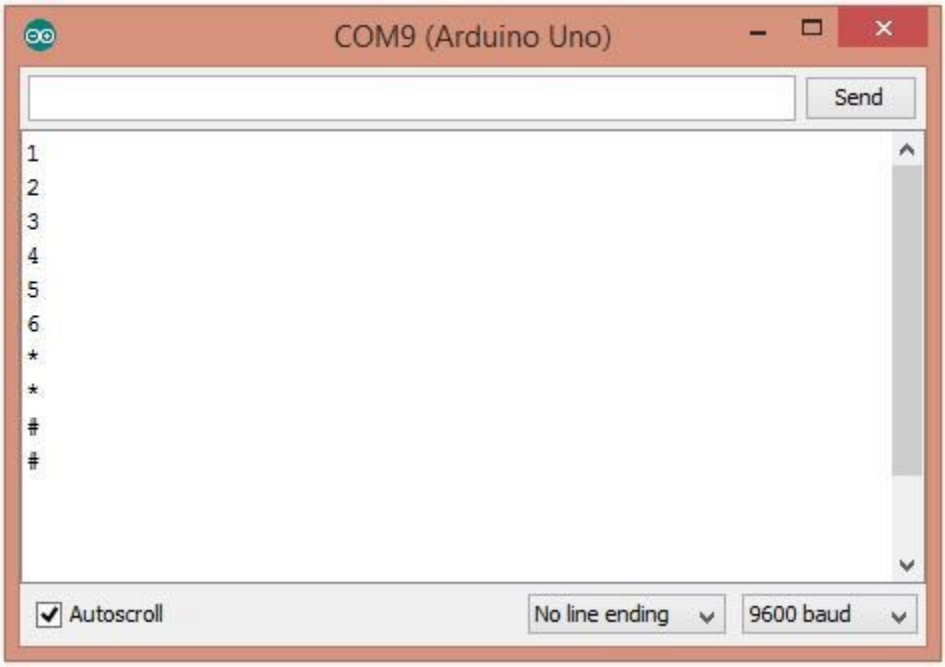
```
    Serial.println(key);
}
```

SOURCE CODE

https://github.com/RuiSantosdotme/Random-Nerd-Tutorials/blob/master/Projects/Arduino_with_Keypad.ino

Demonstration

In this project when you press a key, its value is displayed on the Arduino serial monitor. Here's what you should see in your Arduino IDE serial monitor when you start pressing the keypad keys.

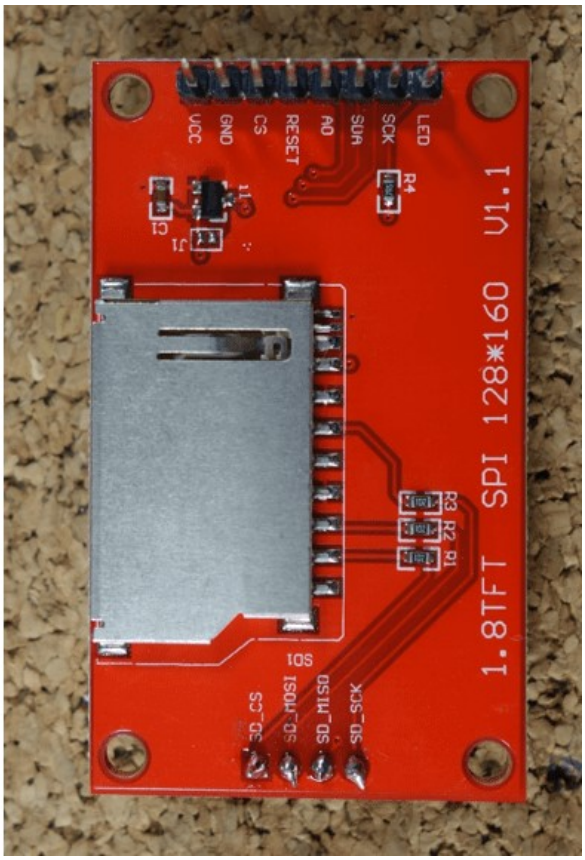


Wrapping up

Now you can create an interface for your Arduino using a keypad. You can also add an LCD to this project.

1.8 TFT Display

The 1.8 TFT is a colorful display with 128 x 160 color pixels. The display can load images from an SD card – it has an SD card slot at the back. The following figure shows the screen front and back view.



This module uses SPI communication – see the wiring below. To control the display we'll use the TFT library, which is already included with Arduino IDE 1.0.5 and later.

Where to buy?

Click the link below to compare the display at different stores and find the best price:

- 1.8 TFT display

Pin wiring

The table below shows the 1.8 TFT wiring to Arduino UNO.

1.8 TFT Display Pin	Wiring to Arduino Uno
LED	3.3 V
SCK	13
SDA	11
A0 or DC	9
RESET	8
CS	10
GND	GND

Note: different Arduino boards have different SPI pins. If you're using another Arduino board, check the Arduino official documentation.

Initializing the display

The TFT display communicates with the Arduino via SPI communication, so you need to include the SPI library on your code. We also use the TFT library to write and draw on the display.

```
#include <TFT.h>

#include <SPI.h>
```

Then, you need to define the CS, A0 (or DC) and RST pins:

```
#define cs 10
```

```
#define dc 9

#define rst 8
```

Create an instance of the library called `TFTscreen`:

```
TFT TFTscreen = TFT(cs, dc, rst);
```

Finally, in the `setup()`, you need to initialize the library:

```
TFTscreen.begin();
```

Display text

To write text on the display, you can customize the screen background color, font size and color.

To set the background color, use:

```
TFTscreen.background(r, g, b);
```

In which, **r**, **g** and **b** are the RGB values for a given color.

To choose font color use:

```
TFTscreen.stroke(r, g, b);
```

To set the font size:

```
TFTscreen.setTextSize(2);
```

You can increase or decrease the number given as argument, to increase or decrease font size.

Finally, to draw text on the display you use the following line:

```
TFTscreen.text("Hello, World!", x, y);
```

In which **"Hello, World!"** is the text you want to display and the (**x, y**) coordinate is the location where you want to start display text on the screen.

Code

The following example displays "Hello, World!" in the middle of the screen and changes the font color every 200 milliseconds.

Copy the following code to your Arduino IDE and upload it to your Arduino board.

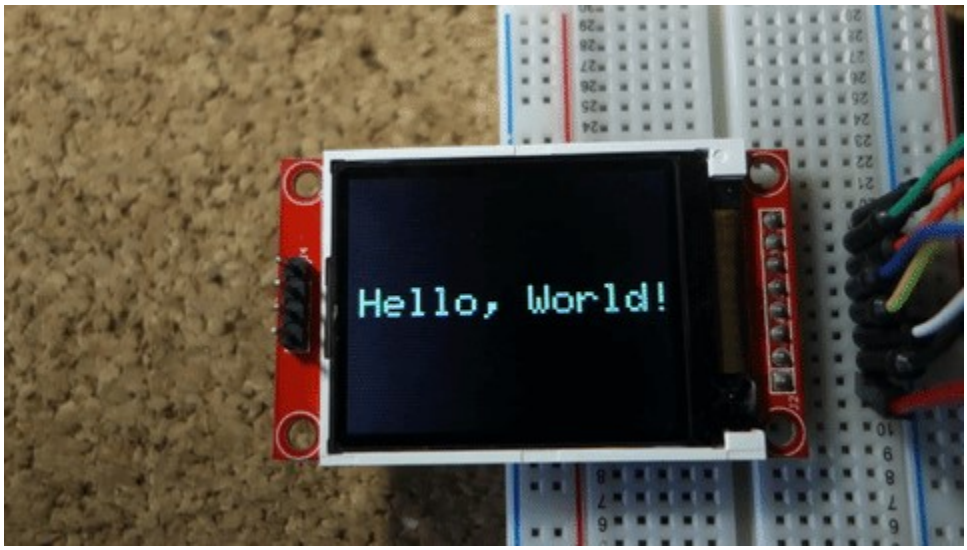
```
/*
 * Rui Santos
 * Complete Project Details http://randomnerdtutorials.com
 */

// include TFT and SPI libraries
#include <TFT.h>
#include <SPI.h>

// pin definition for Arduino UNO
#define cs   10
#define dc   9
#define rst  8


// create an instance of the library
TFT TFTscreen = TFT(cs, dc, rst);

void setup() {

  //initialize the library
  TFTscreen.begin();

  // clear the screen with a black background
  TFTscreen.background(0, 0, 0);
  //set the text size
  TFTscreen.setTextSize(2);
}

void loop() {

  //generate a random color
  int redRandom = random(0, 255);
  int greenRandom = random (0, 255);
```

```
  int blueRandom = random (0, 255);

  // set a random font color
  TFTscreen.stroke(redRandom, greenRandom, blueRandom);

  // print Hello, World! in the middle of the screen
  TFTscreen.text("Hello, World!", 6, 57);

  // wait 200 miliseconds until change to next color
  delay(200);
}
```

SOURCE CODE

https://github.com/RuiSantosdotme/Random-Nerd-Tutorials/blob/master/Projects/tft/write_text.ino

Here's your "Hello, World!" text on the 1.8 TFT display.



Display shapes

The TFT library provides useful functions to draw shapes on the display:

- `TFTscreen.point(x, y)` – displays a point at the (`x`, `y`) coordinate

- **`TFTscreen.line(xStart, yStart, xEnd, yEnd)`** – draws a line that starts at (**`xStart`**, **`yStart`**) and ends at (**`xEnd`**, **`yEnd`**)
- **`TFTscreen.rect(xStart, yStart, width, height)`** – draws a rectangle with the top left corner at (**`xStart`**, **`yStart`**) with the defined width and height
- **`TFTscreen.circle(x, y, radius)`** – draws a circle with center at (**`x`**, **`y`**) with the specified radius

Code

The following example displays several shapes. Every time the code goes through the loop, the shapes change color.

Copy the following code to your Arduino IDE and upload it to your Arduino board.

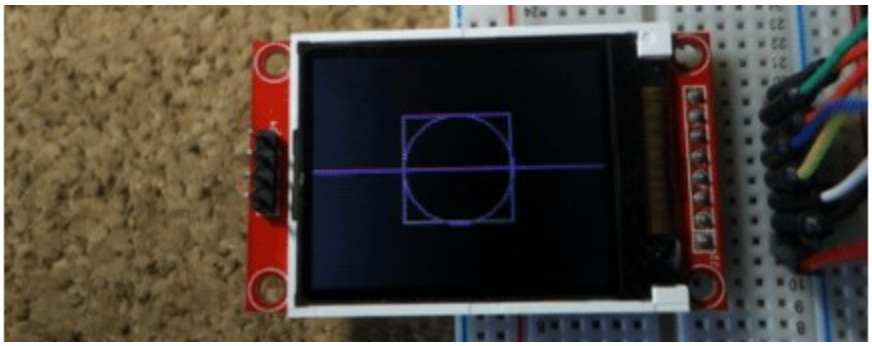
```
/*
 * Rui Santos
 * Complete Project Details http://randomnerdtutorials.com
 */

// include TFT and SPI libraries
#include <TFT.h>
#include <SPI.h>

// pin definition for Arduino UNO
#define cs   10
#define dc   9
#define rst  8


// create an instance of the library
TFT TFTscreen = TFT(cs, dc, rst);

void setup() {

  //initialize the library
  TFTscreen.begin();

  // clear the screen with a black background
```

```
  TFTscreen.background(0, 0, 0);
}

void loop() {

  //generate a random color
  int redRandom = random(0, 255);
  int greenRandom = random (0, 255);
  int blueRandom = random (0, 255);

  // set the color for the figures
  TFTscreen.stroke(redRandom, greenRandom, blueRandom);

  // light up a single point
  TFTscreen.point(80,64);
  // wait 200 miliseconds until change to next figure
  delay(500);

  // draw a line
  TFTscreen.line(0,64,160,64);
  delay(500);

  //draw a square
  TFTscreen.rect(50,34,60,60);
  delay(500);

  //draw a circle
  TFTscreen.circle(80,64,30);
  delay(500);

  //erase all figures
  TFTscreen.background(0,0,0);
}
```

SOURCE CODE

https://github.com/RuiSantosdotme/Random-Nerd-Tutorials/blob/master/Projects/tft/draw_shapes.ino

Here's the shapes on the display:



Display images

The 1.8 TFT display can load images from the SD card. To read from the SD card you use the **SD** library, already included in the Arduino IDE software. Follow the next steps to display an image on the display:

1) Solder header pins for the SD card. There are four pins opposite to the display pins, as shown in figure below.

2) The display can load images bigger or smaller than the display size (160 x 128 px), but for better results, edit your image size to 160 x 128 px.

3) The image should be in *.bmp* format. To do that, you can use a photo editing software and save the image as *.bmp* format.

4) Copy the image to the SD card and insert it on the SD card slot at the back of the display.

5) Wire the SD card pins to the Arduino by following the table below:

SD card pins on TFT display	Wiring to Arduino Uno
CS	4
MOSI	11
MISO	13
SCK	12

Both the display and the SD card work with SPI communication, so you'll have pins on the Arduino with two connections.

6) In the Arduino IDE go to **File ▸ Examples ▸ TFT ▸ Arduino ▸ TFTBitmaLogo**.

7) Edit the code, so that it searches for your image. Replace the `"arduino.bmp"` with the name of your image:

```
// now that the SD card can be access, try to load the image
file

logo = TFTscreen.loadImage("arduino.bmp");
```

8) Upload the code to your Arduino.

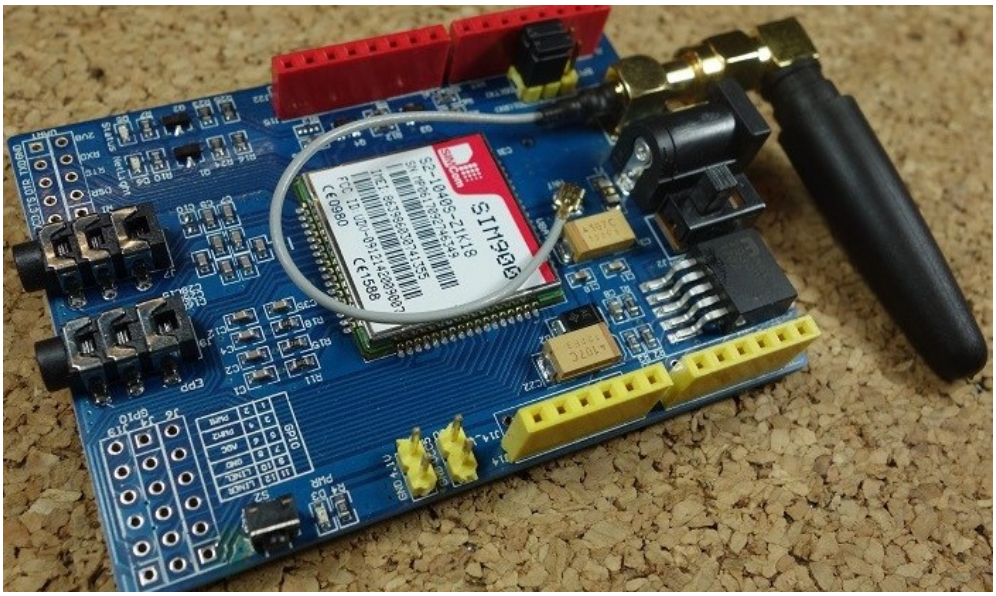
Note: some people find issues with this display when trying to read from the SD card. We don't know why that happens. In fact, we tested a couple of times and it worked well, and then, when we were about to record to show you the final result, the display didn't recognized the SD card anymore – we're not sure if it's a problem with the SD card holder that doesn't establish a proper connection with the SD card. However, we are sure these instructions work, because we've tested them before.

Wrapping up

In this guide we've shown you how to use the 1.8 TFT display with the Arduino: display text, draw shapes and display images. You can easily add a nice visual interface to your projects using this display.

SIM900 GSM GPRS Shield

The SIM900 GSM GPRS Shield is shown in figure below.



GSM stands for Global System for Mobile Communications and is the global standard for mobile communications.

GPRS stands for General Packet Radio Service. GPRS is a mobile service on the 2G and 3G cellular communication.

Applications:

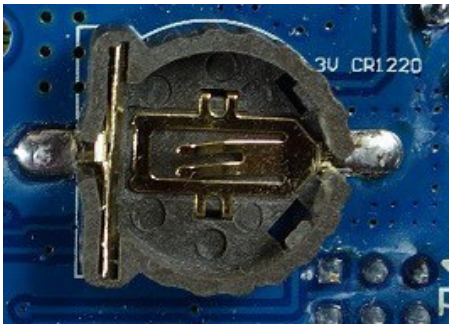
The GSM GPRS shield is particularly useful as it allows to:

- Connect to the Internet over GPRS network
- Send and receive SMS
- Place and receive phones calls
- Its capabilities make it perfect for projects with Arduino like:
- Remote control of electronic appliances – sending an SMS to turn something on;
- Receive notifications – send SMS to your cell phone if movement is detected in your house;
- Receive sensor data – send periodic SMS to your cell phone with daily weather data.

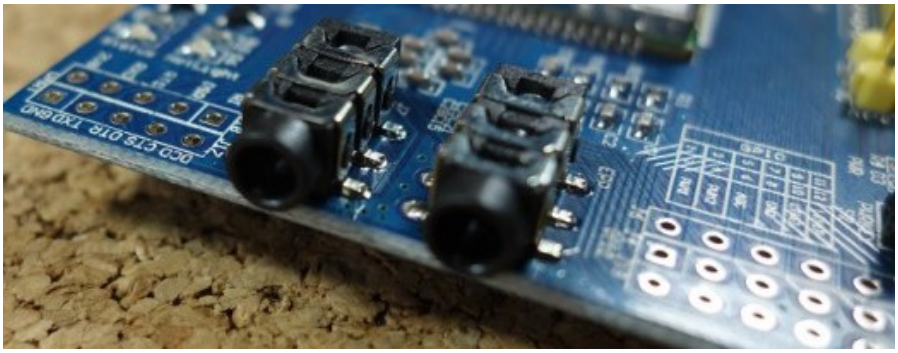
Features

Here's some of the most important features of the shield:

- Compatible with Arduino and clones
- Based on SIM900 module from SIMCOM
- Allows you to send SMS, MMS, GPRS and Audio via UART using AT commands.
- It has 12 GPIOs, 2 PWMs and built-in ADC of the SIM900 module
- Quad Band: 850; 900; 1800 and 1900 MHZ, so it should work in all countries with GSM (2G) networks
- Control via AT commands
- Supports RTC (real time clock) – it has a holder for a 3V CR1220 battery at the back



- Has microphone and headphone jacks for phone calls



Where to buy?

Click the link below to compare the sensor at different stores and find the best price:

- SIM900 GSM/GPRS Shield

Preliminary steps

Before getting started with your SIM900 GSM GPRS module, you need to consider some aspects about the SIM card and the shield power supply.

GSM coverage

Ensure you have coverage on a GSM 850 MHz, GSM 900 MHz, DCS 1800 MHz or PCS 1900 MHz network. By GSM we mean 2G.

Prepaid SIM Card

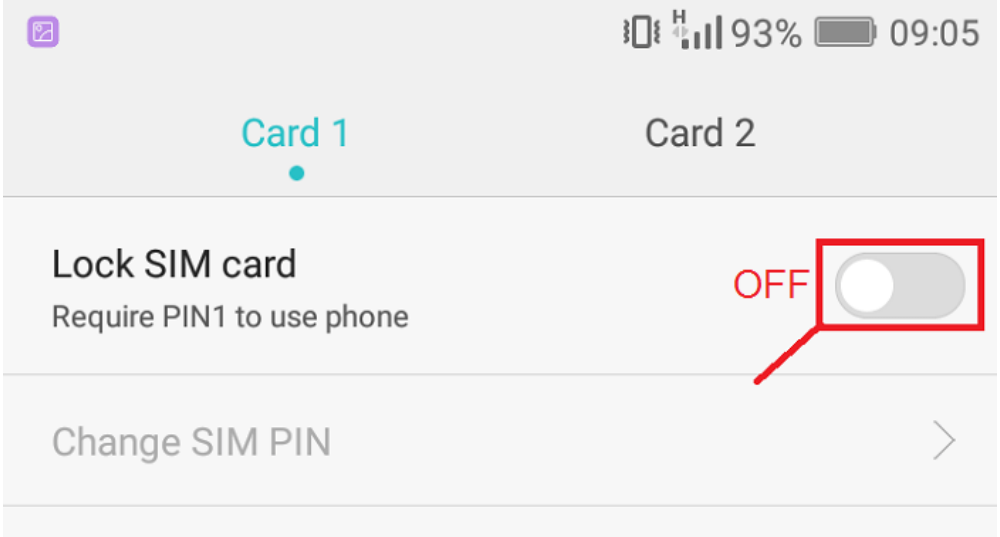
We recommend that you use a prepaid plan or a plan with unlimited SMS for testing purposes. Otherwise, if something goes wrong, you may need to pay a huge bill for hundreds of SMS text messages sent by mistake. In this tutorial we're using a prepaid plan with unlimited SMS.

The shield uses the original SIM card size, not micro or nano. If you have micro or nano you may consider getting a SIM card size adapter.

Turn off the PIN lock

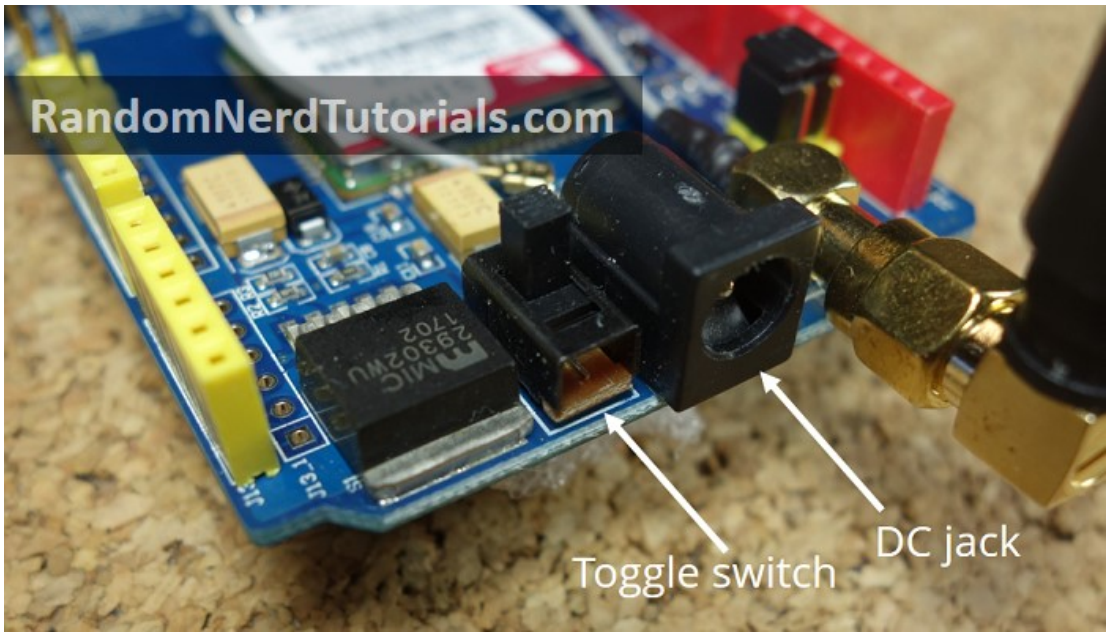
To use the SIM card with the shield, you need to turn off the pin lock. The easiest way to do this, is to insert the SIM card in your smartphone and turn off the pin lock in the phone security settings.

In my case, I needed to go through: **Settings ▸ Advanced Settings ▸ Security ▸ SIM lock** and turn off the Lock SIM card with pin.



Getting the right power supply

The shield has a DC socket for power as shown in figure below.



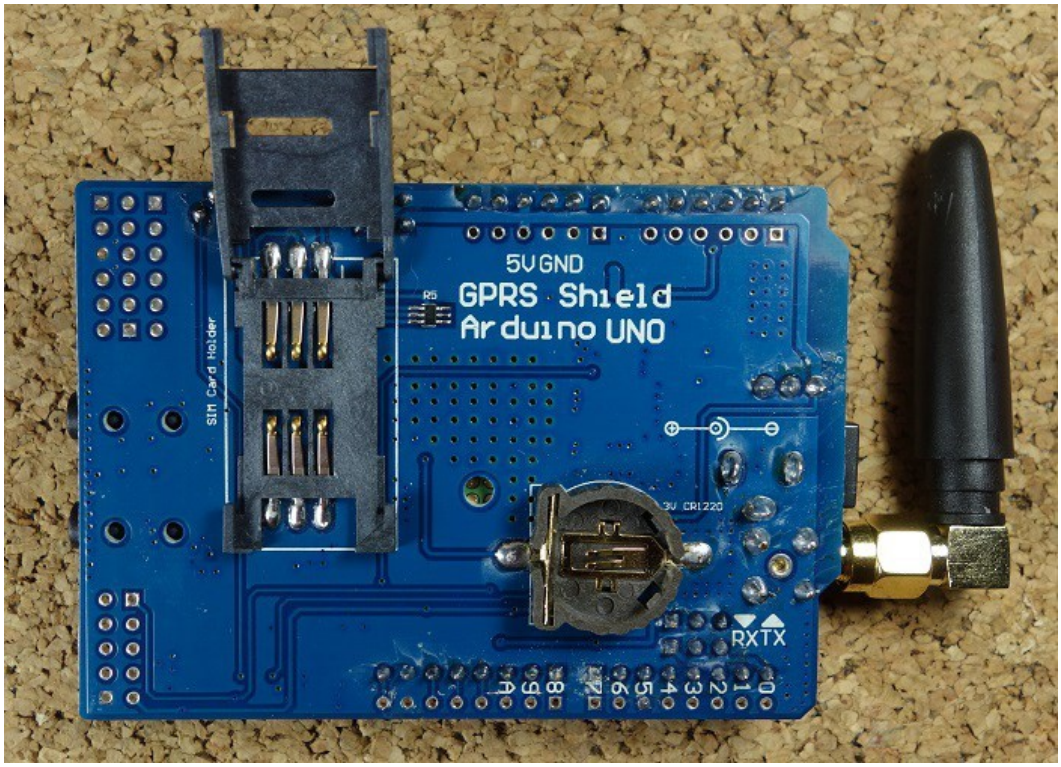
Next to the power jack there is a toggle switch to select the power source. Next to the toggle switch on the board, there is an arrow indicating the toggle position to use an external power supply – move the toggle switch to use the external power supply as shown above.

To power up the shield, it is advisable to use a 5V power supply that can provide 2A as the one shown below. It can also be powered with 9V 1A, or 12V 1A.

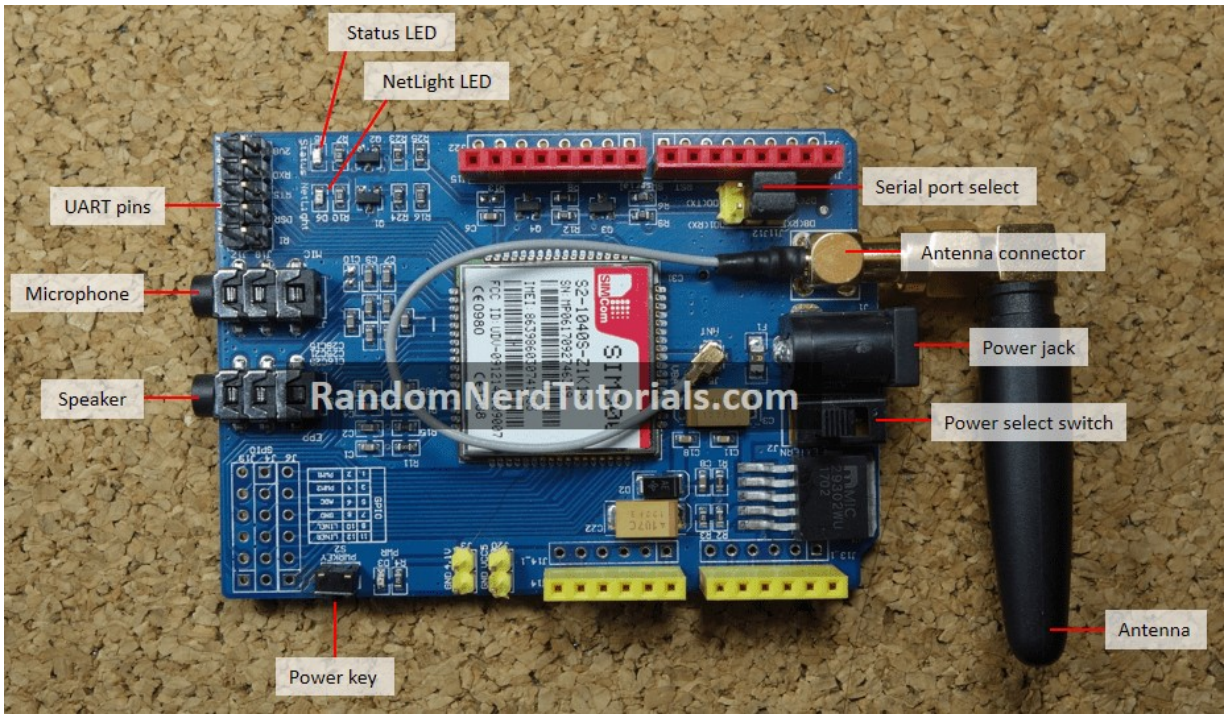
You can find the right power adapter for this shield here. Make sure you select the model with 5V and 2A.

SIM900 GSM GPRS Shield Hardware

The figure below shows the back of the shield. It has a holder for the SIM card and for a 3V CR1220 battery for the RTC (real time clock).

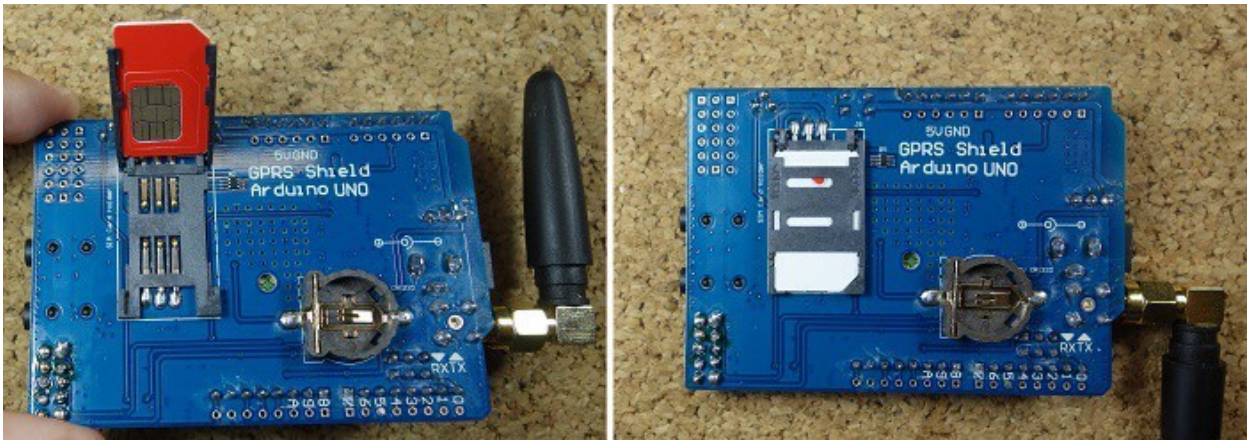


The figure below shows the shield most important components on the board that you need to pay attention to.



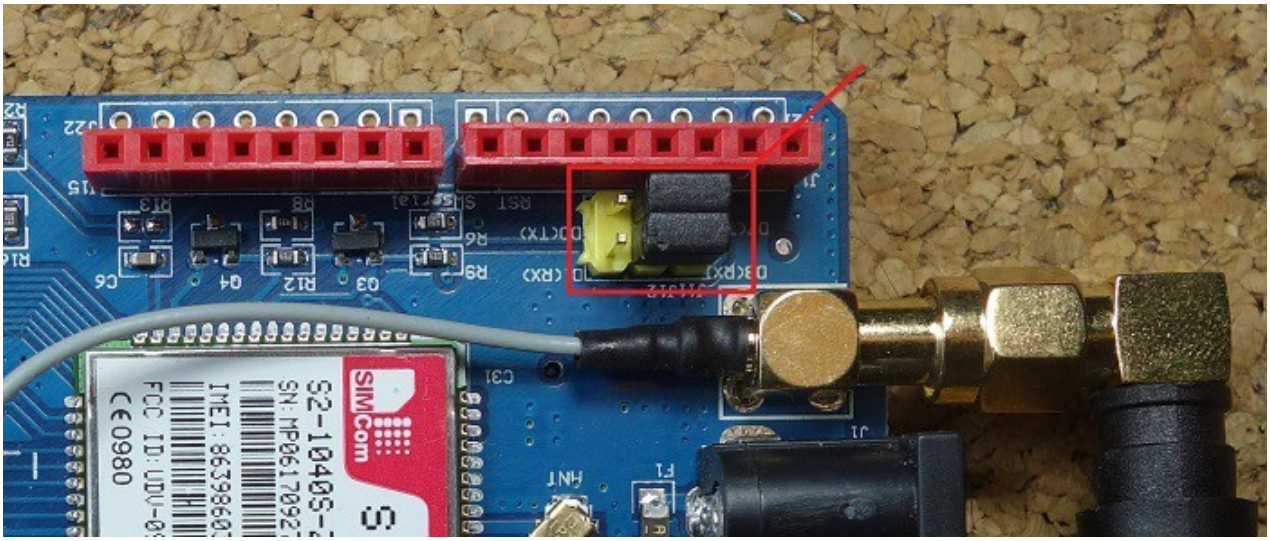
Getting started

1) Insert the SIM card into the SIM card holder – make sure you've read the preliminary steps in the previous section.

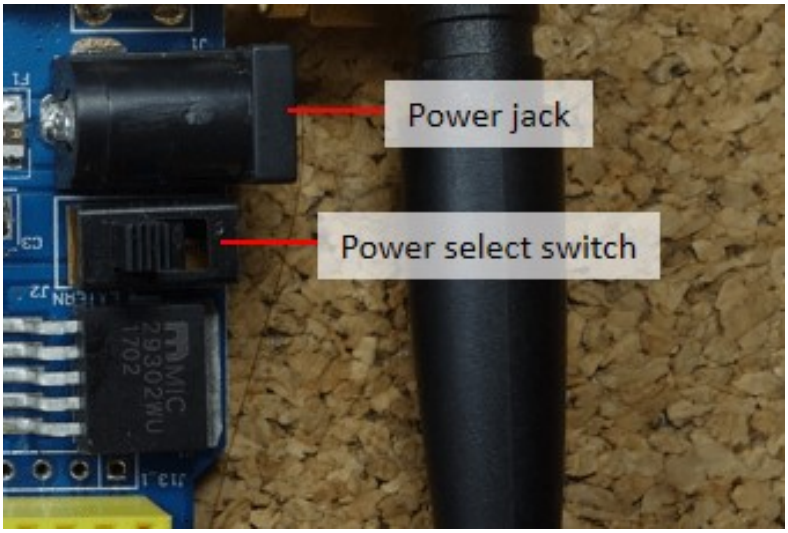


2) Make sure the antenna is well connected.

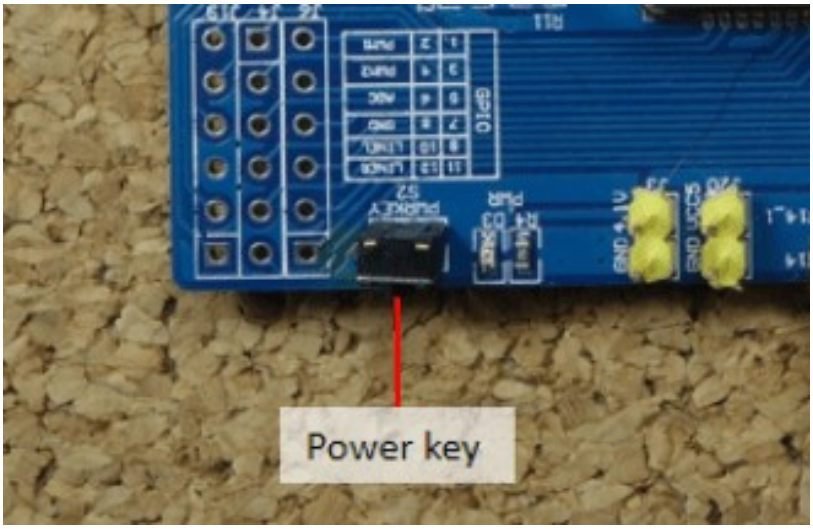
3) On the serial port select, make sure the jumper cap is connected as shown in figure below to use software serial.



4) Power the shield using an external 5V power supply. Make sure you select the external power source with the toggle switch next to the DC jack.



5) To power up/down the shield press the power key for about 2 seconds.



6) Then, the Status LED will light up and the NetLight LED will blink every 800 ms until it finds the network. When it finds the network, the NetLight LED will start blinking every three seconds.

Note: you can automatically turn on the shield via software. See how to do that in the **Automatically turn on the shield** section, after the code examples.

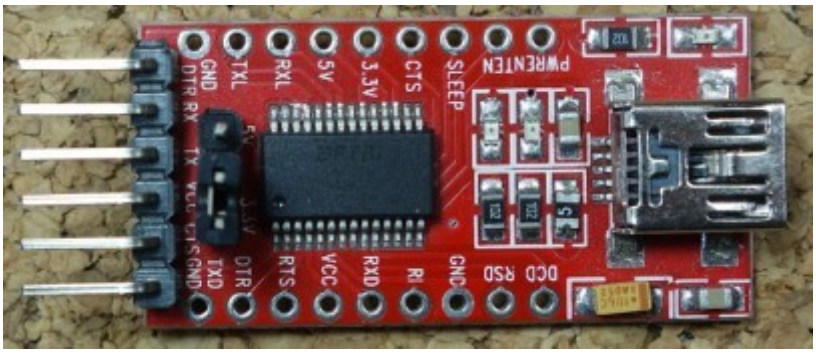
7) You can test if the shield is working properly by sending AT commands from the Arduino IDE using an FTDI programmer – as we'll shown later in this guide.

SIM900 AT commands

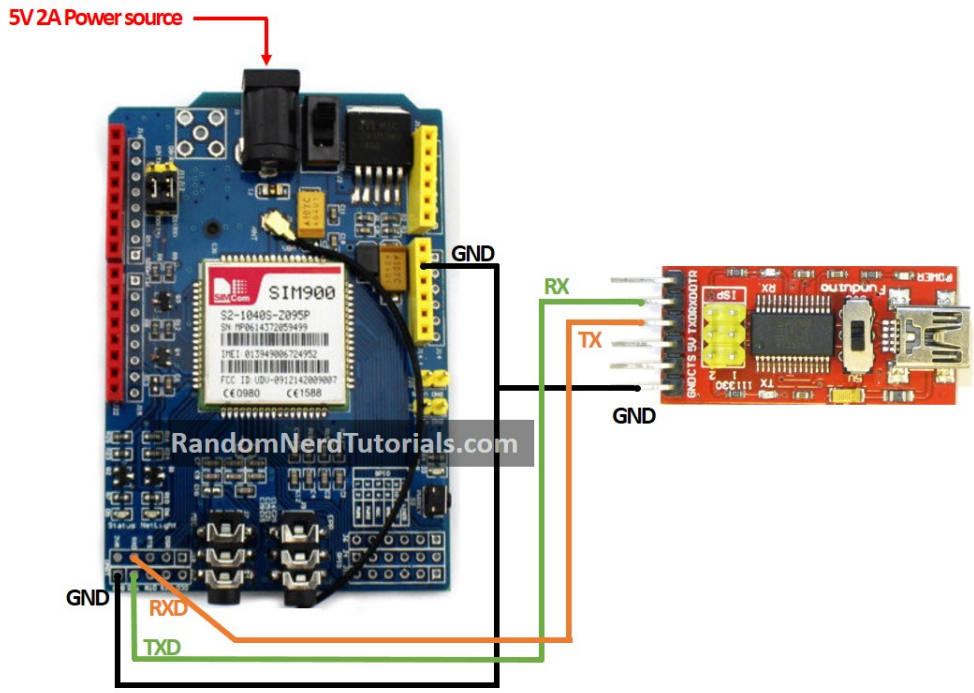
- set the SIM900 to text mode: **AT+CMGF=1\r**
- send SMS to a number: A**T+CMGS=PHONE_NUMBER** (in international format)
- read the first SMS from the inbox: **AT+CMGR=1\r**
- read the second SMS from the inbox: **AT+CMGR=2\r**
- read all SMS from the inbox: **AT+CMGR=ALL\r**
- call to a number: **ATDP+ PHONE_NUMBER** (in international format)
- hang up a call: **ATH**
- receive an incoming call: **ATA**
- For more information, you can check the SIM900 AT commands manual here.

Testing the Shield with FTDI programmer

To test if everything is working properly, you can test the shield by sending AT commands from the Arduino IDE serial monitor. For that, you need an FTDI programmer as the one shown in figure below. You can get an FTDI programmer like this here.



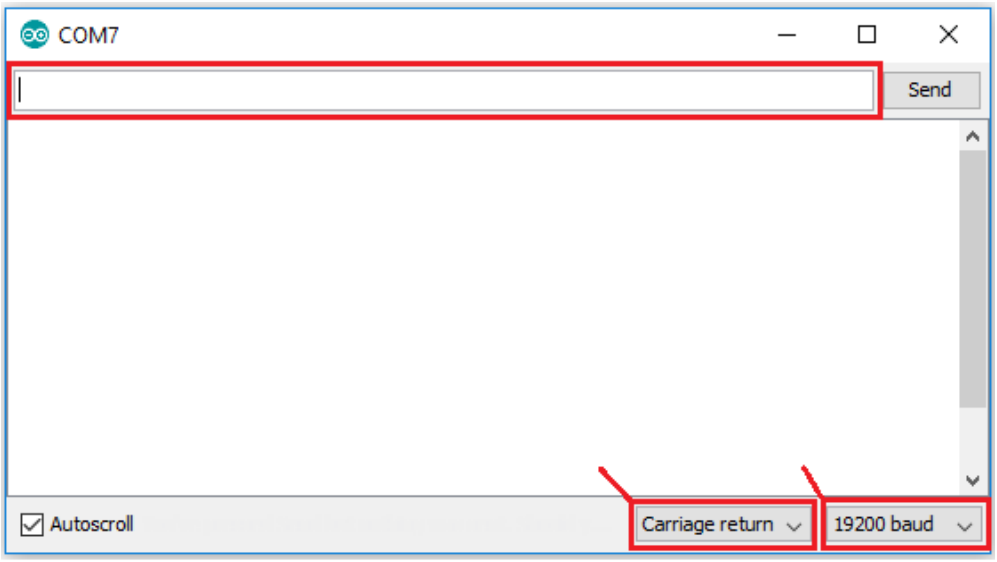
1) Connect the FTDI programmer to the GSM shield as shown in figure below.



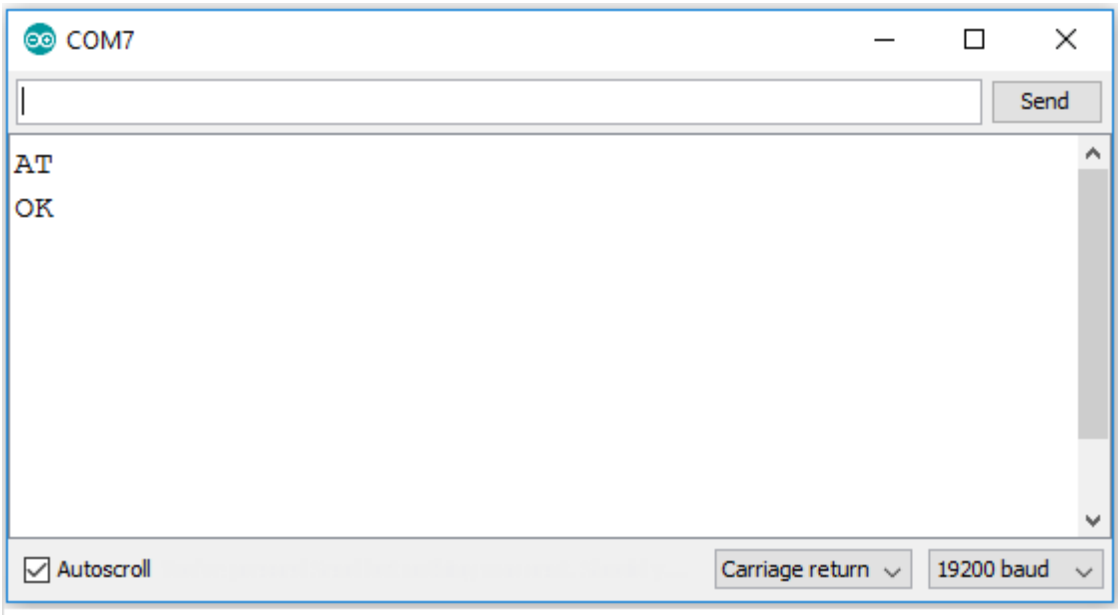
2) Open the Arduino IDE and select the right COM port.

3) Open the Serial monitor.

4) Select 19200 baud rate – the shield default setting is 19200 – and Carriage return. Write AT at the box highlighted in red and then press enter. See figure below.

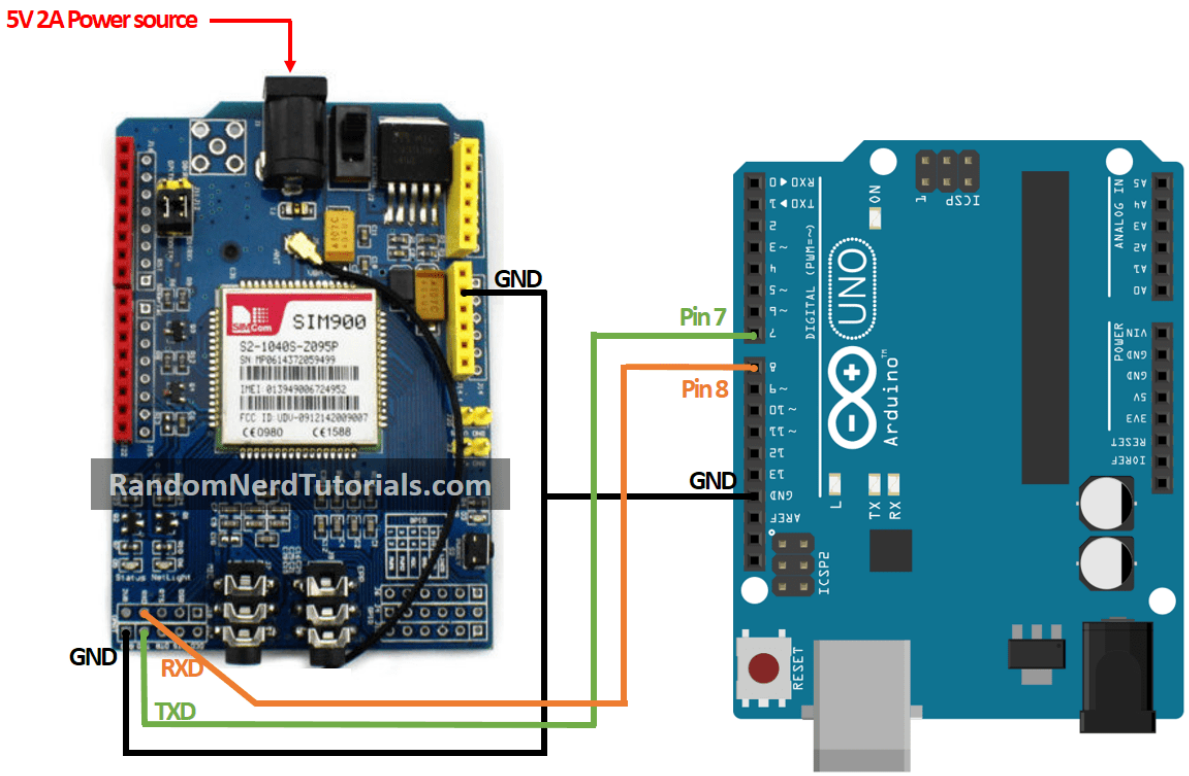


5) The shield will respond with **OK**, if everything is working properly.



Connecting the Shield to Arduino

Connect the shield to the Arduino as shown in the schematic diagram below.



Sending an SMS

To send an SMS, upload the code below to your Arduino board.

```
/*********
  Complete project details at http://randomnerdtutorials.com
*********/

#include <SoftwareSerial.h>

// Configure software serial port
SoftwareSerial SIM900(7, 8);

void setup() {
  // Arduino communicates with SIM900 GSM shield at a baud rate of 19200
  // Make sure that corresponds to the baud rate of your module
  SIM900.begin(19200);
  // Give time to your GSM shield log on to network
  delay(20000);

  // Send the SMS
  sendSMS();
}

void loop() {

}

void sendSMS() {
  // AT command to set SIM900 to SMS mode
  SIM900.print("AT+CMGF=1\r");
  delay(100);

  // REPLACE THE X's WITH THE RECIPIENT'S MOBILE NUMBER
  // USE INTERNATIONAL FORMAT CODE FOR MOBILE NUMBERS
  SIM900.println("AT + CMGS = \"+XXXXXXXXXXXX\"");
  delay(100);

  // REPLACE WITH YOUR OWN SMS MESSAGE CONTENT
  SIM900.println("Message example from Arduino Uno.");
  delay(100);
```

```
  // End AT command with a ^Z, ASCII code 26
  SIM900.println((char)26);
  delay(100);
  SIM900.println();
  // Give module time to send SMS
  delay(5000);
}
```

SOURCE CODE

https://github.com/RuiSantosdotme/Random-Nerd-Tutorials/blob/master/Projects/GSM/sendSMS.ino

In this code, you start by including the **SoftwareSerial.h** library and create a software serial port on pins 7 and 8 (Pin 7 is being set as RX and 8 as TX).

```
#include <SoftwareSerial.h>

SoftwareSerial SIM900(7, 8);
```

The `sendSMS()` function created is what actually sends the SMS. This function uses the AT commands: **AT+CMGF=1\r** and **AT + CMGS**.

You need to change the recipient's mobile number at: (replace the X's with the recipient's phone number)

```
SIM900.println("AT + CMGS = \"+++++++++++++++"");
```

The recipient's mobile number should be in international format.

Then, at the following line you can edit the text you want to send.

```
// REPLACE WITH YOUR OWN SMS MESSAGE CONTENT

SIM900.println("Message example from Arduino Uno.")
```

Reading received SMS

To read incoming SMS, upload the code below to your Arduino. After uploading, wait 20 seconds for the shield to establish communication. Then, test the script by sending an SMS to the shield SIM card number. The SMS is shown on the Arduino serial monitor – baud rate: 19200.

```
/*********
  Complete project details at http://randomnerdtutorials.com
*********/

#include <SoftwareSerial.h>

// Configure software serial port
SoftwareSerial SIM900(7, 8);
//Variable to save incoming SMS characters
char incoming_char=0;

void setup() {
  // Arduino communicates with SIM900 GSM shield at a baud rate of 19200
  // Make sure that corresponds to the baud rate of your module
  SIM900.begin(19200);
  // For serial monitor
  Serial.begin(19200);
  // Give time to your GSM shield log on to network
  delay(20000);

  // AT command to set SIM900 to SMS mode
  SIM900.print("AT+CMGF=1\r");
  delay(100);
  // Set module to send SMS data to serial out upon receipt
  SIM900.print("AT+CNMI=2,2,0,0,0\r");
  delay(100);
}

void loop() {
  // Display any text that the GSM shield sends out on the serial monitor
  if(SIM900.available() >0) {
    //Get the character from the cellular serial port
    incoming_char=SIM900.read();
```

```
    //Print the incoming character to the terminal
    Serial.print(incoming_char);
  }
}
```

SOURCE CODE

https://github.com/RuiSantosdotme/Random-Nerd-Tutorials/blob/master/Projects/GSM/receiveSMS.ino

In this code, you set the module to send the SMS data to the serial output:

```
SIM900.print("AT+CNMI=2,2,0,0,0\r");
```

You store the incoming characters from the SMS message on the **incoming_char** variable. You read the chars using the **SIM900.read()** function.

Making a phone call

To make a phone call, upload the following code to your Arduino.

Don't forget to edit the code with the phone number you want to call.

```
/*********
  Complete project details at http://randomnerdtutorials.com
*********/

#include <SoftwareSerial.h>

// Configure software serial port
SoftwareSerial SIM900(7, 8);

void setup() {
  // Arduino communicates with SIM900 GSM shield at a baud rate of 19200
  // Make sure that corresponds to the baud rate of your module
  SIM900.begin(19200);
  // Give time to your GSM shield log on to network
  delay(20000);
```

```
  // Make the phone call
  callSomeone();
}

void loop() {

}

void callSomeone() {
  // REPLACE THE X's WITH THE NUMER YOU WANT TO DIAL
  // USE INTERNATIONAL FORMAT CODE FOR MOBILE NUMBERS
  SIM900.println("ATD + +XXXXXXXXX;");
  delay(100);
  SIM900.println();

 // In this example, the call only last 30 seconds
 // You can edit the phone call duration in the delay time
  delay(30000);
  // AT command to hang up
  SIM900.println("ATH"); // hang up
}
```

SOURCE CODE

https://github.com/RuiSantosdotme/Random-Nerd-Tutorials/blob/master/Projects/GSM/makePhoneCall.ino

To make the call, you use the `callSomeone()` function that uses the **ATD** command.

```
SIM900.println("ATD + +XXXXXXXXX;");
```

You need to replace the X's (highlighted in red) with the phone number you want to call.

Don't forget to connect a microphone and earphones to make the call.

In this code example, the call is hang up after 30 seconds, using the **ATH** command:

```
SIM900.println("ATH");
```

Hanging up after 30 seconds is not very useful, but it works well for an example. The idea is that you use the ATH command when an event is triggered. For example, connect a push button to the Arduino, that when pressed sends the ATH command to hang up the phone.

Answering incoming phone calls

The following code answers incoming calls.

```
/*********
  Complete project details at http://randomnerdtutorials.com
*********/

#include <SoftwareSerial.h>

// Configure software serial port
SoftwareSerial SIM900(7, 8);
char incoming_char=0;

void setup() {
  // Arduino communicates with SIM900 GSM shield at a baud rate of 19200
  // Make sure that corresponds to the baud rate of your module
  SIM900.begin(19200); // for GSM shield
  // For serial monitor
  Serial.begin(19200);
  // Give time to log on to network.
  delay(20000);

  SIM900.print("AT+CLIP=1\r"); // turn on caller ID notification
  delay(100);
}
void loop() {
  // Display any text that the GSM shield sends out on the serial monitor
  if(SIM900.available() >0) {
    // Get the character from the cellular serial por
    // With an incomming call, a "RING" message is sent out
    incoming_char=SIM900.read();
    //  Check if the shield is sending a "RING" message
    if (incoming_char=='R') {
```

```
      delay(10);
      Serial.print(incoming_char);
      incoming_char=SIM900.read();
      if (incoming_char =='I') {
        delay(10);
        Serial.print(incoming_char);
        incoming_char=SIM900.read();
        if (incoming_char=='N') {
          delay(10);
          Serial.print(incoming_char);
          incoming_char=SIM900.read();
          if (incoming_char=='G') {
            delay(10);
            Serial.print(incoming_char);
            // If the message received from the shield is RING
            // Send ATA commands to answer the phone
            SIM900.print("ATA\r");
          }
        }
      }
    }
  }
}
```

When someone calls the SIM900 number, it sends a message saying "RING". To know if someone is calling you, you can wait for incoming characters from the SIM900 and then, compare if it was a RING message. That's what is done in this code. When it receives a RING message, you send the **ATA** command to answer the phone.

SOURCE CODE

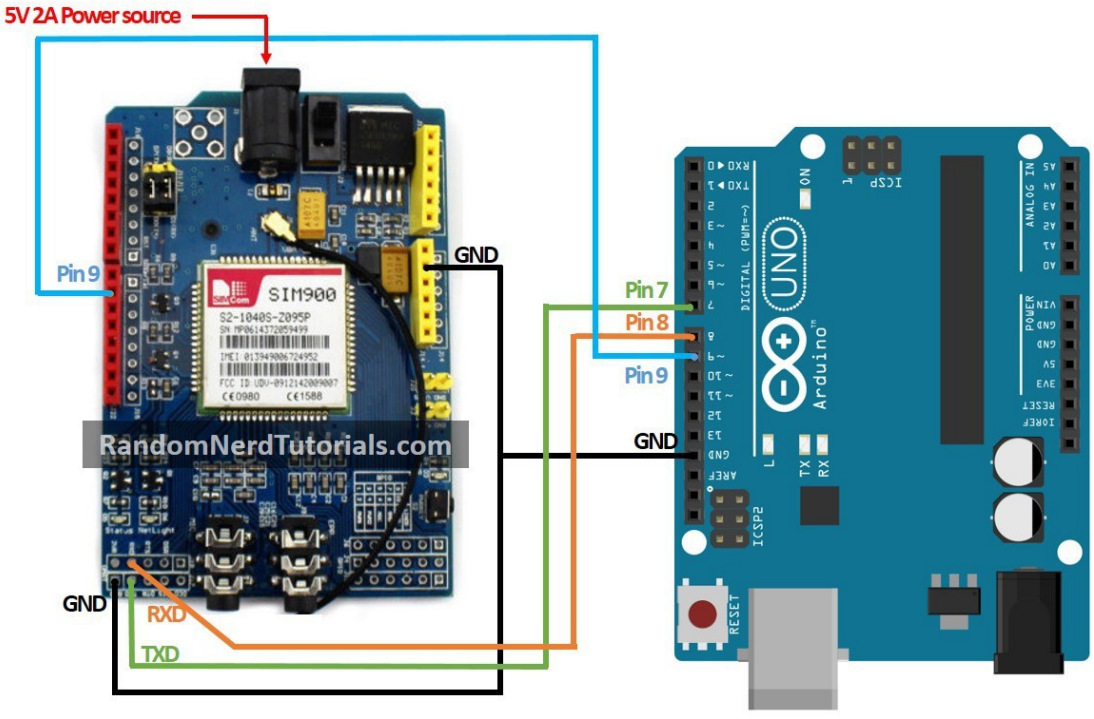
https://github.com/RuiSantosdotme/Random-Nerd-Tutorials/blob/master/Projects/GSM/receiveCall.ino

Automatically turn on the shield

Instead of manually pressing the "power" key to turn on the shield, you can automatically turn on the shield via software.

1) First, you need to solder **R13** connections on the shield as shown in the figure below – highlighted in red.

2) Connect D9 on the shield to the D9 Arduino pin as shown in the schematic below.



3) Add the following code snippet in the `setup()` function. This is the equivalent of pressing the shield "power" button.

```
digitalWrite(9, HIGH);

delay(1000);
```

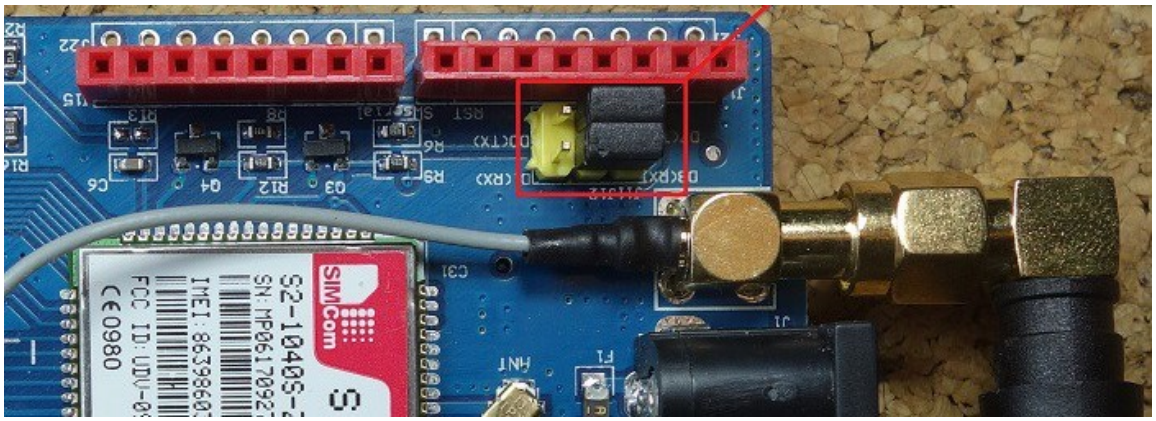
```
digitalWrite(9, LOW);

delay(5000);
```

Troubleshooting

Shield doesn't respond with OK

Check your TX and RX connections to the Arduino. Try repeating the process by changing the TX with the RX pins.

Also check if you have selected the software serial by placing the jumper cap on the appropriate place on the serial selector.



Cannot see messages in the serial monitor

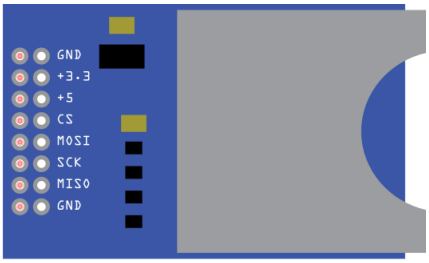
To see the messages in the serial monitor, the shield and the Arduino's serial port baud rate should be the same. The SIM900 GSM GPRS shield default baud rate is 19200. So, select the Arduino's baud rate to 19200.

However, if you need to change the shield baud rate, you can send the following AT command to change it to 19200 or other appropriate baud rate.

```
AT+IPR=19200
```

Wrapping up

This tutorial shows you how to send and receive SMS and making and receiving phone calls with the Arduino. You can apply the concepts learned in this tutorial to build your own projects to communicate over a cell network.

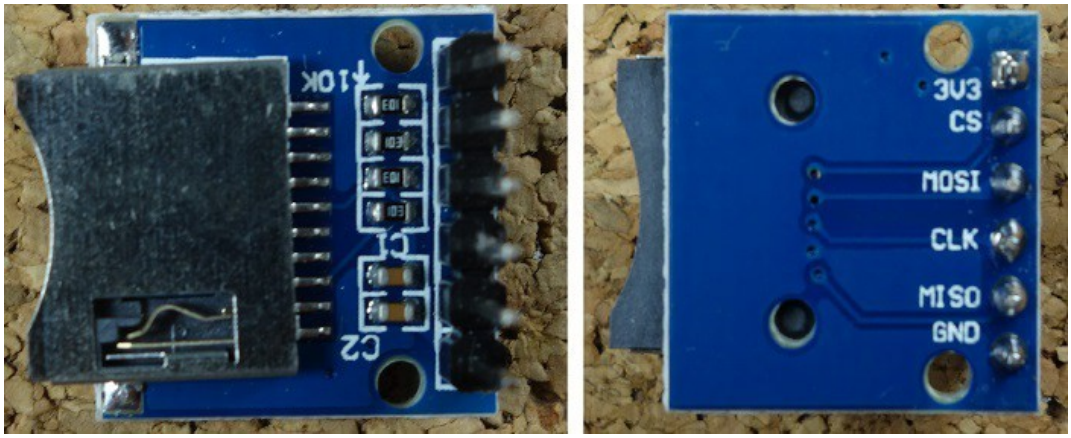


SD Card Module

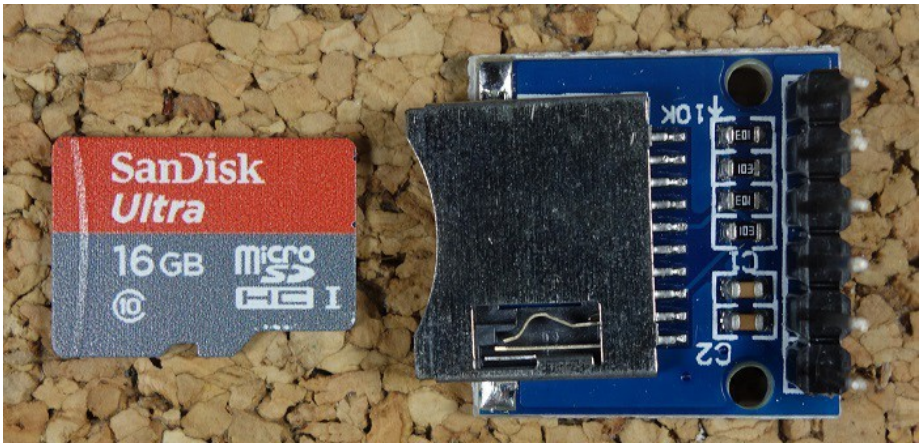
The SD card module is especially useful for projects that require data logging.

The Arduino can create a file in an SD card to write and save data using the **SD** library.

There are different models from different suppliers, but they all work in a similar way, using the SPI communication protocol. The module used in this tutorial is the one shown in figure below (front and back view).



This module works with micro SD card but there are others that work with SD card.



Where to buy?

Click the link below to compare the sensor at different stores and find the best price:

- Micro SD card module

Pin wiring

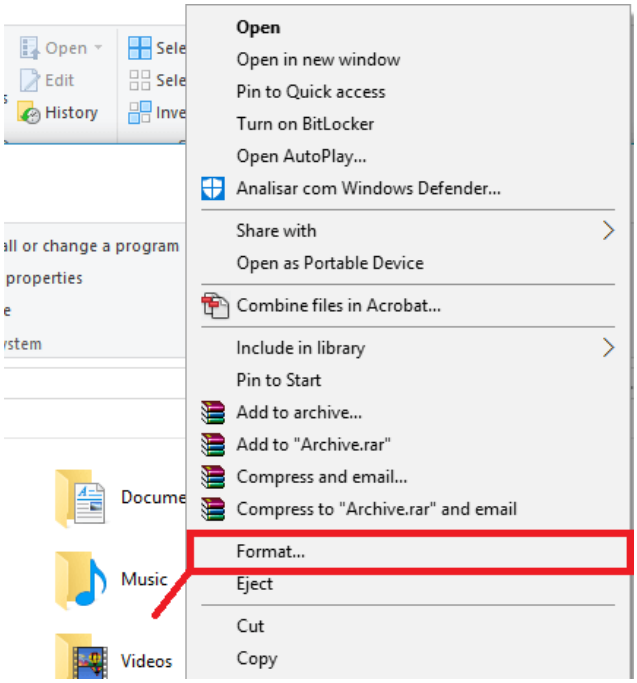
SD card module	Wiring to Arduino Uno	Wiring to Arduino Mega
VCC	3.3V or 5V (check module's datasheet)	3.3V or 5V (check module's datasheet)
CS	4	53
MOSI	11	51
CLK	13	52
MISO	12	50
GND	GND	GND

Note: different Arduino boards have different SPI pins. If you're using another Arduino board, check the Arduino official documentation.

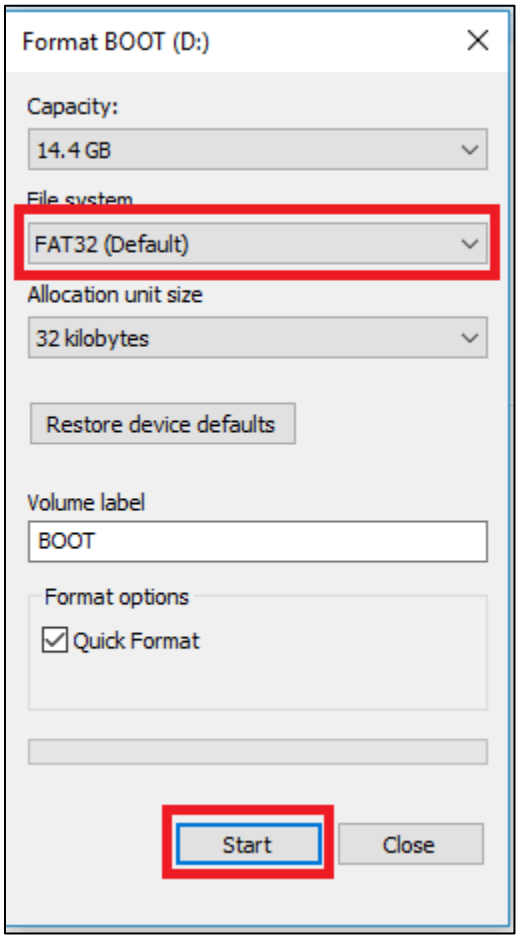
Preparing the SD card

The first step when using the SD card module with Arduino is formatting the SD card as FAT16 or FAT32. Follow the instructions below.

1) To format the SD card, insert it in your computer. Go to **My Computer** and right click on the SD card. Select **Format** as shown in figure below.



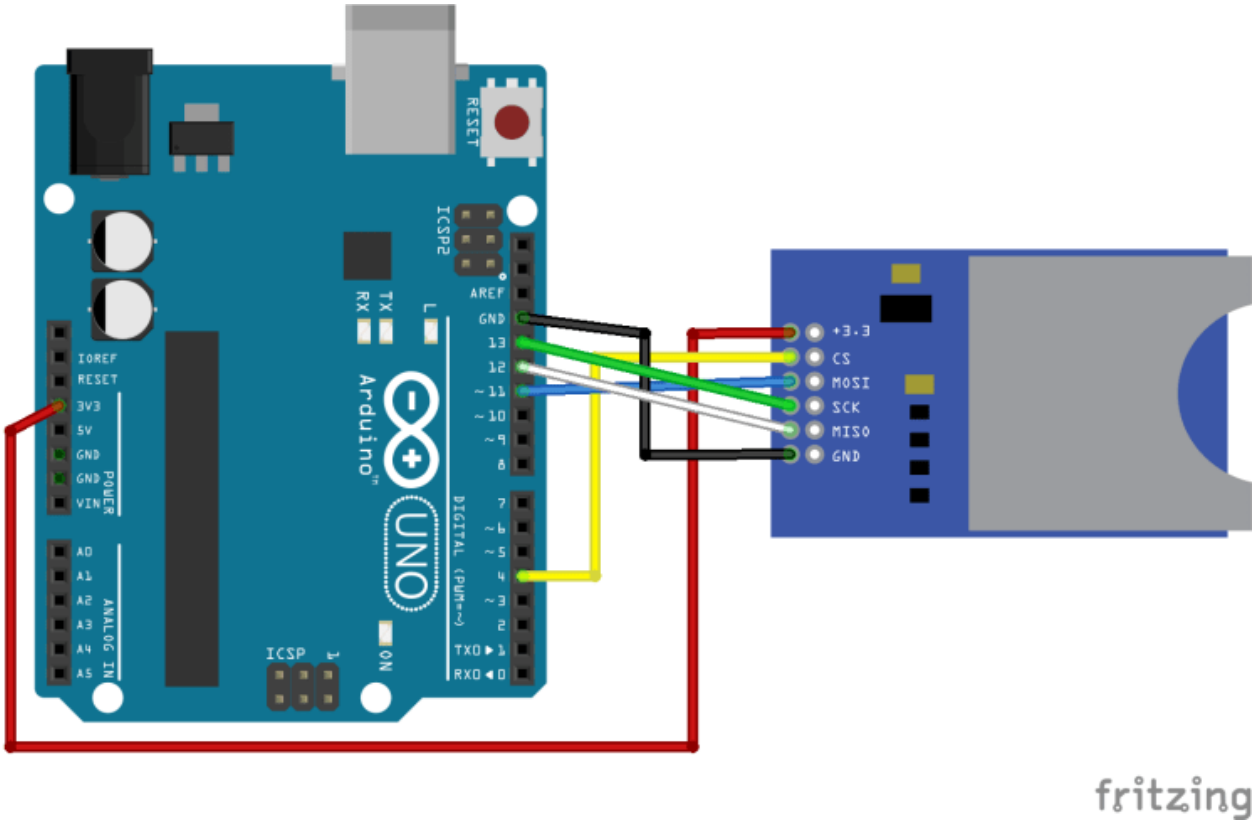
2) A new window pops up. **Select FAT32**, press **Start** to initialize the formatting process and follow the onscreen instructions.



Testing the SD card module

Insert the formatted SD card in the SD card module.

Connect the SD card module to the Arduino as shown in the circuit schematic below or check Pin wiring in previous section.



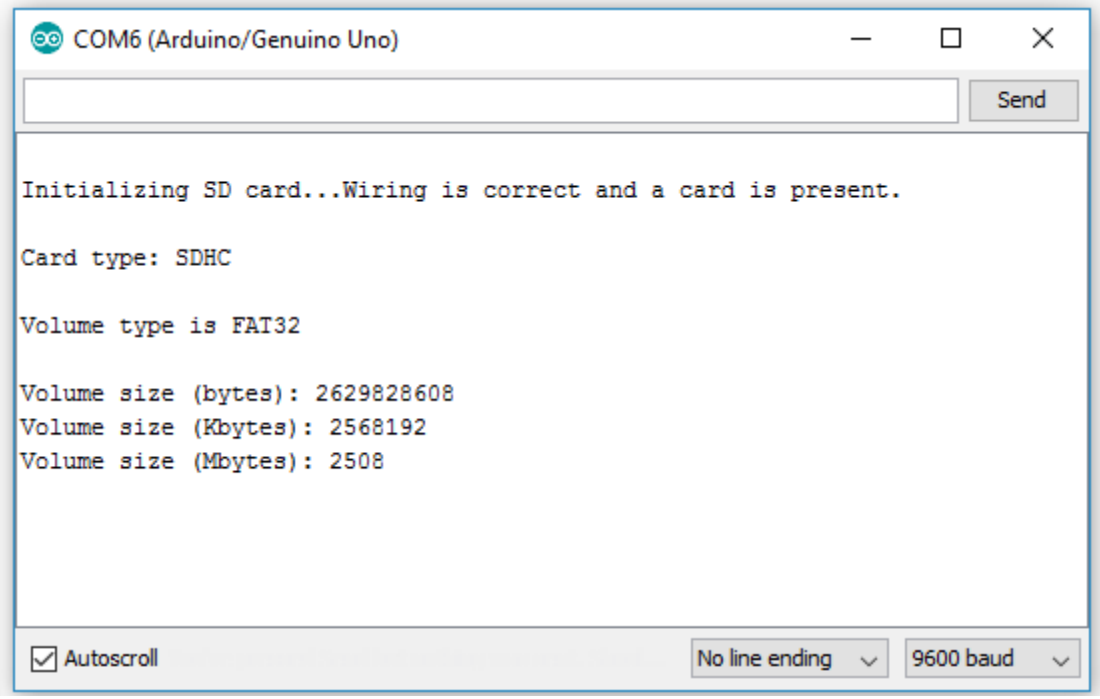
Note: depending on the module you're using, the pins may be in a different order.

Code – CardInfo

To make sure everything is wired correctly and the SD card is working properly, in the Arduino IDE window go to **File ▸ Examples ▸ SD ▸ CardInfo**.

Upload the code to your Arduino board. Make sure you have the right board and COM port selected. Open the Serial Monitor at a baud rate of 9600 and you should see your SD card information.

If everything is working properly you'll see a similar message on the serial monitor.



Read and write to the SD card

The **SD** library provides useful functions for easily write in and read from the SD card.

To write and read from the SD card, first you need to include the **SPI** and **SD** libraries:

```
#include <SPI.h>

#include <SD.h>
```

You also have to initialize the SD card module at the Chip Select (**CS**) pin – in our case, pin 4.

```
SD.begin(4);
```

To open a new file in the SD card, you need to create a file object that refers to your data file. For example:

```
dataFile = SD.open("data.txt", FILE_WRITE);
```

The first parameter of this function is the name of the file, `data.txt`, and the `FILE_WRITE` argument enables you to read and write into the file.

This line of code creates a file called `data.txt` on your SD card. If the `data.txt` file already exists, the Arduino will open the file instead of creating another one.

To write data to the currently open file, you use:

```
dataFile.write(data);
```

In which the **dataFile** is the file object created previously and the **data** is what you want to write in the file.

You can also use the **print()** or **println()** functions to print data into the file:

```
dataFile.print(data);

dataFile.println(data); // followed by a new line
```

To read the data saved on your file:

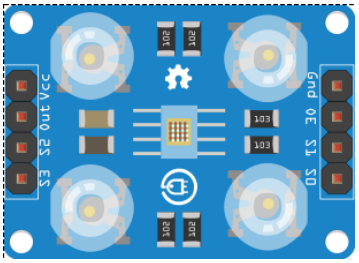
```
dataFile.read();
```

You can only write within a file at once, so you need to close a file before proceeding to the next one. To close the **data.txt** file we've just created:

```
SD.close("data.txt");
```

The argument of this function is the file you want to close, in this case **data.txt**.

For a complete sketch on how to read and write, in your Arduino IDE go to **File▸ Examples ▸ SD ▸ ReadWrite**.



TCS3200 Color Sensor

The TCS3200 color sensor can detect a wide variety of colors based on their wavelength. This sensor is especially useful for color recognition projects such as color matching, color sorting, test strip reading and much more.

The TCS3200 color sensor – shown in the figure below – uses a TAOS TCS3200 RGB sensor chip to detect color. It also contains four white LEDs that light up the object in front of it.

Specifications

- Power: 2.7V to 5.5V
- Size: 28.4 x 28.4mm (1.12 x 1.12”)
- Interface: digital TTL
- High-resolution conversion of light intensity to frequency
- Programmable color and full-scale output frequency
- Communicates directly to microcontroller

Where to buy?

Click the link below to compare the sensor at different stores and find the best price:

- Color Sensor TCS3200

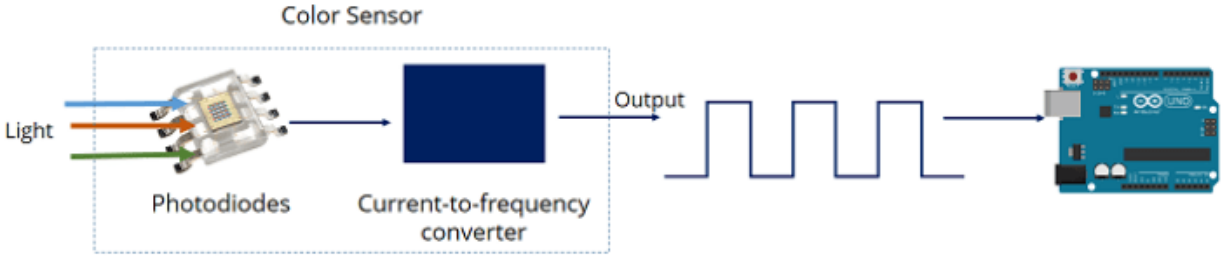
How does the TCS3200 sensor work?

The TCS3200 has an array of photodiodes with 4 different filters. A photodiode is simply a semiconductor device that converts light into current. The sensor has:

- 16 photodiodes with red filter – sensitive to red wavelength
- 16 photodiodes with green filter – sensitive to green wavelength
- 16 photodiodes with blue filter – sensitive to blue wavelength
- 16 photodiodes without filter

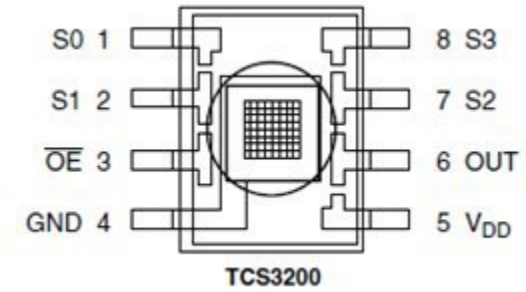
If you take a closer look at the TCS3200 chip you can see the different filters.

By selectively choosing the photodiode filter's readings, you're able to detect the intensity of the different colors. The sensor has a current-to-frequency converter that converts the photodiodes' readings into a square wave with a frequency that is proportional to the light intensity of the chosen color. This frequency is then, read by the Arduino – this is shown in the figure below.



Pin wiring

Here's the sensor pinout:



Sensor Pin	I/O	Description
GND (4)		Power supply ground
OE (3)	I	Enable for output frequency (active low)
OUT (6)	O	Output frequency
S0, S1（1，2）	I	Output frequency scaling selection inputs
S2, S3（7，8）	I	Photodiode type selection inputs
VDD（5）		Voltage supply

Filter selection

To select the color read by the photodiode, you use the control pins S2 and S3. As the photodiodes are connected in parallel, setting the S2 and S3 LOW and HIGH in different combinations allows you to select different photodiodes. Take a look at the table below:

Photodiode type	S2	S3
Red	LOW	LOW
Blue	LOW	HIGH
No filter (clear)	HIGH	LOW
Green	HIGH	HIGH

Frequency scaling

Pins S0 and S1 are used for scaling the output frequency. It can be scaled to the following preset values: 100%, 20% or 2%. Scaling the output frequency is useful to optimize the sensor readings for various frequency counters or microcontrollers. Take a look at the table below:

Output frequency scaling	S0	S1
Power down	L	L
2%	L	H
20%	H	L
100%	H	H

For the Arduino, it is common to use a frequency scaling of 20%. So, you set the S0 pin to HIGH and the S1 pin to LOW.

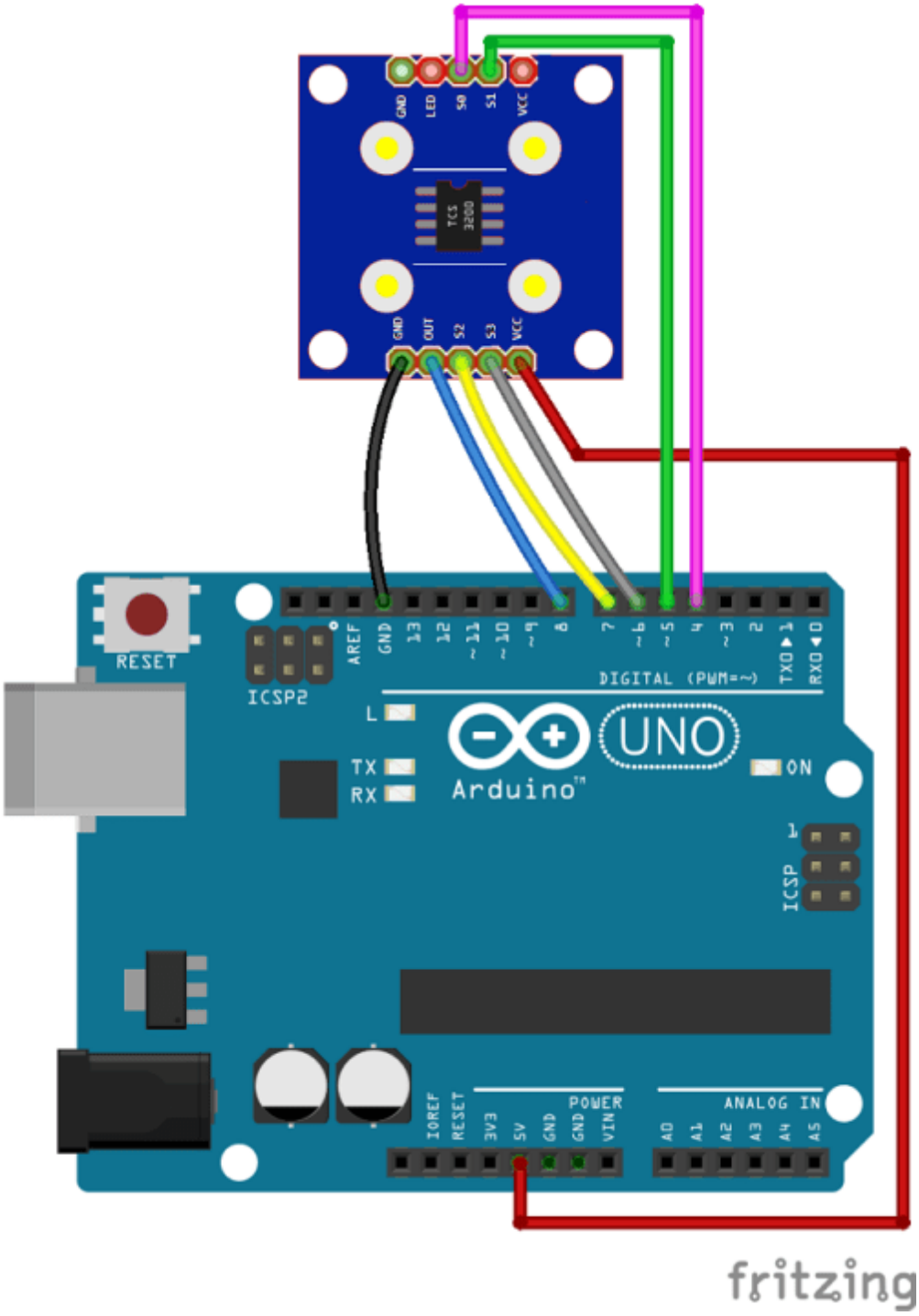
Color Sensing with Arduino and TCSP3200

In this example you're going to detect colors with the Arduino and the TCSP3200 color sensor. This sensor is not very accurate, but works fine for detecting colors in simple projects. For this example you'll need the following parts:

Figure	Name	Check Price
	Arduino UNO	Find best price on Maker Advisor
	TCS3200 Color Sensor	Find best price on Maker Advisor
	Jumper Wires	Find best price on Maker Advisor

Schematic

Wiring the TCSP3200 sensor to your Arduino is pretty straightforward. Simply follow the next schematic diagram.



Pin wiring

Here's the connections between the TCSP3200 and the Arduino:

Sensor Pin	Wiring to Arduino Uno
S0	Digital pin 4
S1	Digital pin 5
VCC	5V
S3	Digital pin 6
S4	Digital pin 7
OUT	Digital pin 8

Code

You need two sketches for this project:

1. **Reading and displaying the output frequency** on the serial monitor. In this part you need to write down the frequency values when you place different colors in front of the sensor.
2. **Distinguish between different colors**. In this section you'll insert the frequency values picked previously on your code, so that your sensor can distinguish between different colors. We'll detect red, green and blue colors.

1. Reading the output frequency

Upload the following code to your Arduino board.

```
/*********
  Rui Santos
  Complete project details at http://randomnerdtutorials.com
*********/

// TCS230 or TCS3200 pins wiring to Arduino
#define S0 4
```

```
#define S1 5
#define S2 6
#define S3 7
#define sensorOut 8

// Stores frequency read by the photodiodes
int redFrequency = 0;
int greenFrequency = 0;
int blueFrequency = 0;

void setup() {
  // Setting the outputs
  pinMode(S0, OUTPUT);
  pinMode(S1, OUTPUT);
  pinMode(S2, OUTPUT);
  pinMode(S3, OUTPUT);

  // Setting the sensorOut as an input
  pinMode(sensorOut, INPUT);

  // Setting frequency scaling to 20%
  digitalWrite(S0,HIGH);
  digitalWrite(S1,LOW);

   // Begins serial communication
  Serial.begin(9600);
}
void loop() {
  // Setting RED (R) filtered photodiodes to be read
  digitalWrite(S2,LOW);
  digitalWrite(S3,LOW);

  // Reading the output frequency
  redFrequency = pulseIn(sensorOut, LOW);

   // Printing the RED (R) value
  Serial.print("R = ");
  Serial.print(redFrequency);
  delay(100);

  // Setting GREEN (G) filtered photodiodes to be read
```

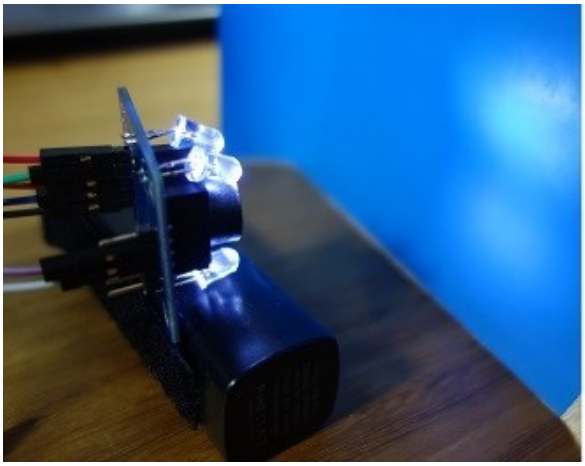
```
  digitalWrite(S2,HIGH);
  digitalWrite(S3,HIGH);

  // Reading the output frequency
  greenFrequency = pulseIn(sensorOut, LOW);

  // Printing the GREEN (G) value
  Serial.print(" G = ");
  Serial.print(greenFrequency);
  delay(100);

  // Setting BLUE (B) filtered photodiodes to be read
  digitalWrite(S2,LOW);
  digitalWrite(S3,HIGH);

  // Reading the output frequency
  blueFrequency = pulseIn(sensorOut, LOW);

  // Printing the BLUE (B) value
  Serial.print(" B = ");
  Serial.println(blueFrequency);
  delay(100);
}
```

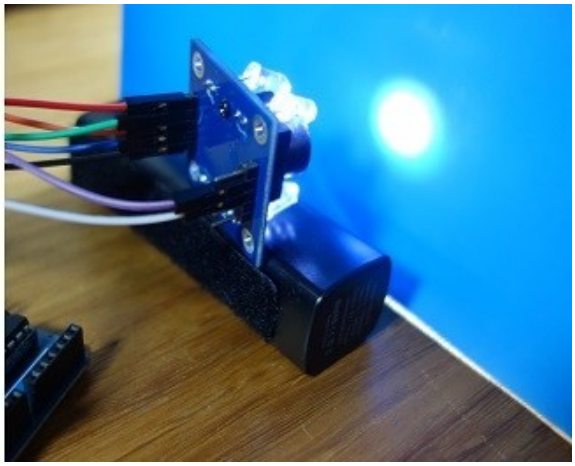
SOURCE CODE

https://github.com/RuiSantosdotme/Random-Nerd-Tutorials/blob/master/Projects/color/Color_Sensor_TCS230_TCS3200_1.ino

Open the serial monitor at a baud rate of 9600.

Place a blue object in front of the sensor at different distances. You should save two measurements: when the object is placed far from the sensor and when the object is close to it.





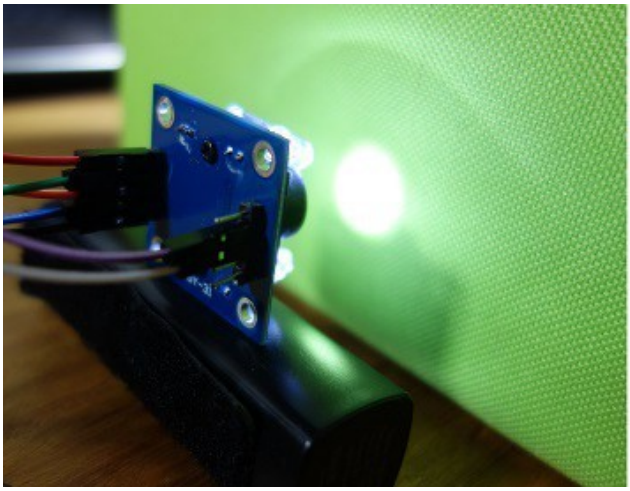
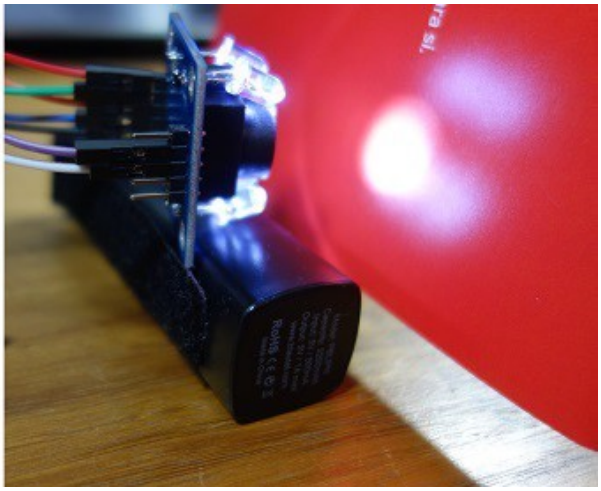
Check the values displayed on the serial monitor. The blue frequency (B) should be the lowest compared to the red (R) and green (G) frequency readings – see figure below.

```
R = 1141 G = 590 B = 178
R = 1182 G = 626 B = 191
R = 1289 G = 645 B = 197
R = 1315 G = 664 B = 209
R = 1381 G = 719 B = 223
R = 1401 G = 707 B = 219
R = 1425 G = 733 B = 220
R = 1395 G = 722 B = 209
R = 1275 G = 627 B = 193
R = 1221 G = 619 B = 177
R = 1018 G = 486 B = 141
R = 842 G = 449 B = 145
R = 932 G = 473 B = 148
R = 904 G = 482 B = 146
R = 431 G = 235 B = 73
R = 432 G = 263 B = 75
R = 393 G = 220 B = 63
R = 367 G = 207 B = 65
R = 369 G = 210 B = 62
R = 322 G = 195 B = 59
R = 329 G = 207 B = 63
```

When we place the blue object in front of the sensor, the blue frequency (B) values oscillate between 59 and 223 (see highlighted values).

Note: you can't use these frequency values (59 and 223) in your code, you should measure the colors for your specific object with your own color sensor. Then, save your upper and bottom frequency limits for the blue color, because you'll need them later.

Repeat this process with a green and red objects and write down the upper and bottom frequency limits for each color.

2. Distinguish between different colors

This next sketch maps the frequency values to RGB values (that are between 0 and 255).

In the previous step the maximum frequency for blue as 233 and the minimum was 59. So, 59 in frequency corresponds to 255 (in RGB) and 223 in frequency to 0 (in RGB). We'll do this with the Arduino `map()` function. In the `map()` function you need to replace XX parameters with your own values.

```
/*********
  Rui Santos
  Complete project details at http://randomnerdtutorials.com
*********/

// TCS230 or TCS3200 pins wiring to Arduino
#define S0 4
#define S1 5
#define S2 6
#define S3 7
#define sensorOut 8
// Stores frequency read by the photodiodes
int redFrequency = 0;
int greenFrequency = 0;
int blueFrequency = 0;
// Stores the red. green and blue colors
int redColor = 0;
int greenColor = 0;
int blueColor = 0;
```

```
void setup() {
  // Setting the outputs
  pinMode(S0, OUTPUT);
  pinMode(S1, OUTPUT);
  pinMode(S2, OUTPUT);
  pinMode(S3, OUTPUT);

  // Setting the sensorOut as an input
  pinMode(sensorOut, INPUT);

  // Setting frequency scaling to 20%
  digitalWrite(S0,HIGH);
  digitalWrite(S1,LOW);

  // Begins serial communication
  Serial.begin(9600);
}

void loop() {
  // Setting RED (R) filtered photodiodes to be read
  digitalWrite(S2,LOW);
  digitalWrite(S3,LOW);

  // Reading the output frequency
  redFrequency = pulseIn(sensorOut, LOW);
  // Remaping the value of the RED (R) frequency from 0 to 255
  // You must replace with your own values. Here's an example:
  // redColor = map(redFrequency, 70, 120, 255,0);
  redColor = map(redFrequency, XX, XX, 255,0);

  // Printing the RED (R) value
  Serial.print("R = ");
  Serial.print(redColor);
  delay(100);

  // Setting GREEN (G) filtered photodiodes to be read
  digitalWrite(S2,HIGH);
  digitalWrite(S3,HIGH);

  // Reading the output frequency
```

```
  greenFrequency = pulseIn(sensorOut, LOW);
  // Remaping the value of the GREEN (G) frequency from 0 to 255
  // You must replace with your own values. Here's an example:
  // greenColor = map(greenFrequency, 100, 199, 255, 0);
  greenColor = map(greenFrequency, XX, XX, 255, 0);

  // Printing the GREEN (G) value
  Serial.print(" G = ");
  Serial.print(greenColor);
  delay(100);

  // Setting BLUE (B) filtered photodiodes to be read
  digitalWrite(S2,LOW);
  digitalWrite(S3,HIGH);

  // Reading the output frequency
  blueFrequency = pulseIn(sensorOut, LOW);
  // Remaping the value of the BLUE (B) frequency from 0 to 255
  // You must replace with your own values. Here's an example:
  // blueColor = map(blueFrequency, 38, 84, 255, 0);
  blueColor = map(blueFrequency, XX, XX, 255, 0);

  // Printing the BLUE (B) value
  Serial.print(" B = ");
  Serial.print(blueColor);
  delay(100);
  // Checks the current detected color and prints
  // a message in the serial monitor
  if(redColor > greenColor && redColor > blueColor){
      Serial.println(" - RED detected!");
  }
  if(greenColor > redColor && greenColor > blueColor){
    Serial.println(" - GREEN detected!");
  }
  if(blueColor > redColor && blueColor > greenColor){
    Serial.println(" - BLUE detected!");
  }
}
```

{;} SOURCE CODE

https://github.com/RuiSantosdotme/Random-Nerd-Tutorials/blob/master/Projects/color/Color_Sensor_TCS230_TCS3200_2.ino

To distinguish between different colors we have three conditions:

- When the R is the maximum value (in RGB parameters) we know we have a red object
- When G is the maximum value, we know we have a green object
- When B is the maximum value, we know we have a blue object

Now, place something in front of the sensor. It should print in your serial monitor the color detected: red, green or blue. Your sensor can also detect other colors with more if statements.

Wrapping up

In this post you've learned how to detect colors with the TCSP3200 color sensor. You can easily build a color sorting machine by simply adding a servo motor.

Download Other RNT Products

Random Nerd Tutorials is an online resource with electronics projects, tutorials and reviews.

Creating and posting new projects takes a lot of time. At this moment, Random Nerd Tutorials has nearly 100 free blog posts with complete tutorials using open-source hardware that anyone can read, remix and apply to their own projects: http://randomnerdtutorials.com

To keep free tutorials coming, there's also paid content or as I like to call "Premium Content".

To support Random Nerd Tutorials, you can download Premium content here. If you enjoyed this eBook, make sure you check all the others.

Thanks for taking the time to read my work!

Good luck with all your projects,

-Rui Santos

P.S. Click here for more Courses and eBooks like this one.

Click here to Download other Courses and eBooks

http://randomnerdtutorials.com/products