

2ND EDITION

18 + ARDUINO PROJECTS



Parts + Schematics + Code + Demonstration



Compilation of 18+ Arduino Projects and Tutorials!
by Rui Santos of The RandomNerdTutorials.com Blog

RANDOM NERD TUTORIALS - PROJECTS

Table of Contents

Parts Required.....	6
Introducing the Arduino.....	8
Traffic Lights	16
LED Brightness on a 16x2 LCD	22
Complete Guide for Ultrasonic Sensor HC-SR04 with Arduino.....	27
Parking Sensor.....	34
Gesture Slider Swiper	39
Arduino with PIR Motion Sensor.....	46
Control LEDs with IR Remote Control.....	49
Teensy/Arduino - Memory Game.....	58
Guide for MQ-2 Gas/Smoke Sensor with Arduino	66
Guide for 8x8 Dot Matrix MAX7219 + Pong Game	72
Security Access using MFRC522 RFID Reader with Arduino	86
Arduino Time Attendance System with RFID.....	93
Arduino Temperature Data Logger with SD Card Module	111
Android App – RGB LED with Arduino and Bluetooth.....	118
Control DC Motor via Bluetooth.....	128
Request Sensor Data via SMS.....	133
Night Security Light with Arduino	149
Ethernet Web Server with Relay.....	154
Resources.....	163
Wrapping Up.....	165
Arduino Step-by-step Projects Course	166
Download Other RNT Products.....	168

Disclaimer

This eBook has been written for information purposes only. Every effort has been made to make this eBook as complete and accurate as possible.

The purpose of this eBook is to educate. The author (Rui Santos) does not warrant that the information contained in this eBook is fully complete and shall not be responsible for any errors or omissions. The author (Rui Santos) shall have neither liability nor responsibility to any person or entity with respect to any loss or damage caused or alleged to be caused directly or indirectly by this eBook.

This eBook contains code examples which you can use on your own projects, excepted where otherwise noted.

You cannot redistribute this eBook.

This eBook is only available for free download at:

- <http://randomnerdtutorials.com/download>

Please send an email to the author (Rui Santos - hello@ruisantos.me), if you find this eBook anywhere else.

Introduction

This eBook is a compilation of some of my most popular Arduino projects. For more Arduino projects, take a look at our [Arduino project's repository](#).

I encourage you to watch some of the video demonstrations. Some of my projects are easier to understand if you can see the circuit in action.

This eBook has the purpose to inspire you create something amazing with electronics and programing. After you create something cool, I hope you share it with others. That's the whole goal of this awesome community.

To all my readers, thank you for your interest in my work. I really appreciate it!

Have fun with your projects,

Rui Santos

P.S. Make sure you visit my website to see the latest projects!

<http://RandomNerdTutorials.com>

Connect with Rui

If you have any questions, please don't hesitate to contact me. Here are some ways to stay in touch.

Random Nerd Tutorials

Electronics Projects, Tutorials and Reviews!

[Visit my website](#)

(<http://RandomNerdTutorials.com>)



[Subscribe on YouTube](#)

(<https://www.youtube.com/user/RandomNerdTutorials>)



[Like on facebook](#)

(<https://www.facebook.com/RandomNerdTutorials>)



[Follow me on Twitter](#)

(<https://twitter.com/RuiSantosdotme>)



[Fork me on GitHub](#)

(<https://github.com/RuiSantosdotme>)



[Follow me on Instagram](#)

(<https://www.instagram.com/ruisantosme/>)

Parts Required

To build Arduino projects you need some electronics components beside the bare Arduino board. In each project we provide a complete list of the needed parts and links to [Maker Advisor](#), so that you can find the parts you're looking for on your favorite store at the best price.



If you buy your parts through Maker Advisor links, we'll earn a small affiliate commission (you won't pay more for it). By getting your parts through our affiliate links you are supporting our work. If there's a component or tool you're looking for, we advise you to take a look at [our favorite tools and parts here](#).

What do you need to get started?

In our opinion, the best way to get started with the Arduino is by getting one [Arduino starter kit](#) that contains all the components you need to learn the basics and start doing projects.



[Elegoo Arduino Uno R3 Complete Starter Kit](#)

There are a wide variety of Arduino Starter Kits. The best kit for you depends on what you want to do and how much you are willing to spend. We recommend reading the following article about the best Arduino Starter Kits for Beginners:

- [Best Arduino Starter Kits - Buying Guide](#)

There are also other tools we recommend you getting like a multimeter and a soldering iron.



We have some articles to help you chose the best multimeter and soldering iron for beginners:

- [Best Soldering Irons for Beginners and Hobbyists](#)
- [Best Multimeters Under \\$50](#)

You may also find useful taking a look at the following article that gives you tips to set up your own electronics hobbyist lab:

- [How To Set Up an Electronics Lab: Tools and Equipment](#)

Introducing the Arduino

The Arduino is a small computer that you can program to read information from the world around you and send commands to the outside world. All of this is possible because you can connect several devices and components to the Arduino to do what you want.

You can do amazing projects with it, there is no limit for what you can do, and using your imagination everything is possible!

What is an Arduino?

The Arduino is the board shown in the figure below.

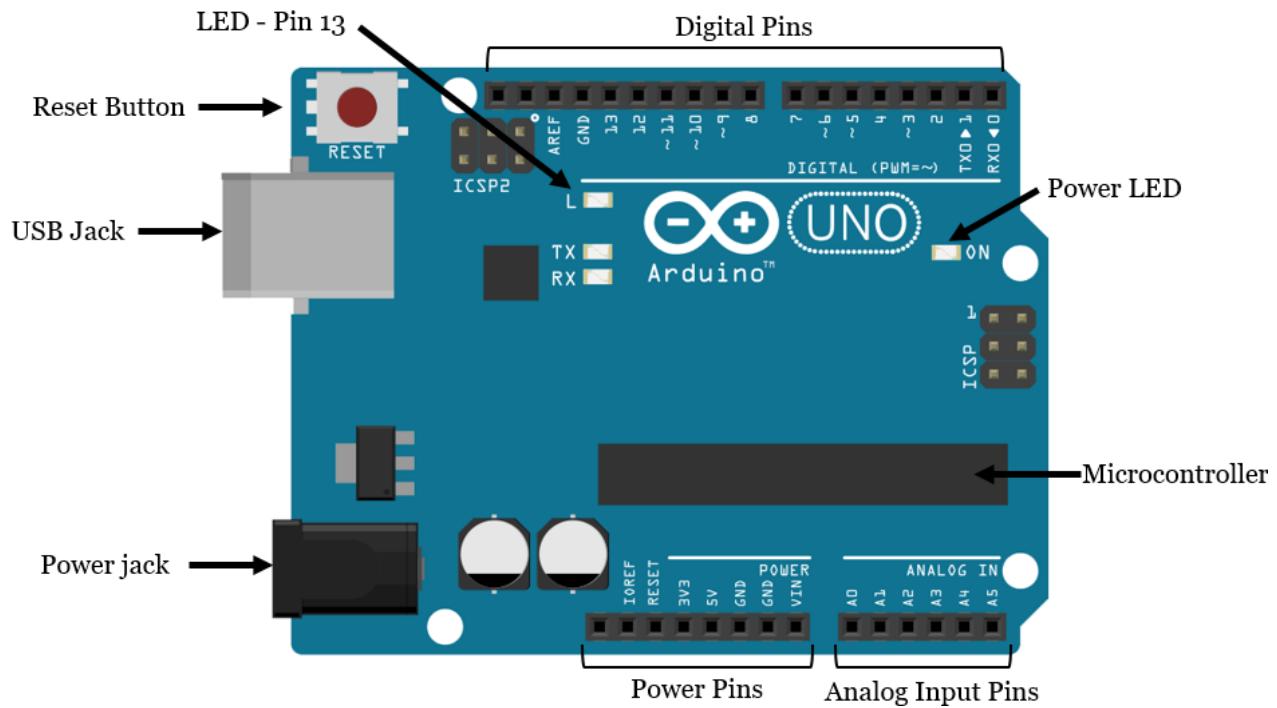


[Arduino UNO R3 board with ATmega328P](#)

Basically, it is a small development board with a brain (also known as a microcontroller) that you can connect to electrical circuits. This makes it easy to read inputs – read data from the outside – and control outputs - send a command to the outside. The brain of this board (Arduino Uno) is an **ATmega328p** chip where you can store your programs that will tell your Arduino what to do.

Exploring the Arduino Uno Board

In the figure below you can see an Arduino board labeled. Let's see what each part does.



- **Microcontroller:** the **ATmega328p** is the Arduino brain. Everything on the Arduino board is meant to support this microcontroller. This is where you store your programs to tell the Arduino what to do.
- **Digital pins:** Arduino has 14 digital pins, labeled from 0 to 13 that can act as inputs or outputs.
 - When set as inputs, these pins can read voltage. They can only read two states: HIGH or LOW.
 - When set as outputs, these pins can apply voltage. They can only apply 5V (HIGH) or 0V (LOW).
- **PWM pins:** These are digital pins marked with a ~ (pins 11, 10, 9, 6, 5 and 3). PWM stands for “pulse width modulation” and allows the digital pins output “fake” varying amounts of voltage. You’ll learn more about PWM later.
- **TX and RX pins:** digital pins 0 and 1. The T stands for “transmit” and the R for “receive”. The Arduino uses these pins to communicate with other electronics via Serial. Arduino also uses these pins to communicate with your computer when uploading new code. Avoid using these pins for other tasks other than serial communication, unless you’re running out of pins.
- **LED attached to digital pin 13:** This is useful for an easy debugging of the Arduino sketches.
- **TX and RX LEDs:** these leds blink when there are information being sent between the computer and the Arduino.

- **Analog pins:** the analog pins are labeled from A0 to A5 and are often used to read analog sensors. They can read different amounts of voltage between 0 and 5V. Additionally, they can also be used as digital output/input pins like the digital pins.
- **Power pins:** the Arduino provides 3.3V or 5V through these pins. This is really useful since most components require 3.3V or 5V to operate. The pins labelled as "GND" are the ground pins.
- **Reset button:** when you press that button, the program that is currently being run in your Arduino restarts. You also have a Reset pin next to the power pins that acts as reset button. When you apply a small voltage to that pin, it will reset the Arduino.
- **Power ON LED:** will be on since power is applied to the Arduino.
- **USB jack:** you need a male USB A to male USB B cable (shown in figure below) to upload programs from your computer to your Arduino board. This cable also powers your Arduino.



- **Power jack:** you can power the Arduino through the power jack. The recommended input voltage is 7V to 12V. There are several ways to power up your Arduino: rechargeable batteries, disposable batteries, wall-warts and solar panel, for example. For more information about this subject you can read this blog post on Random Nerd Tutorials [Arduino – 5 Ways to Power Up your Arduino](#).

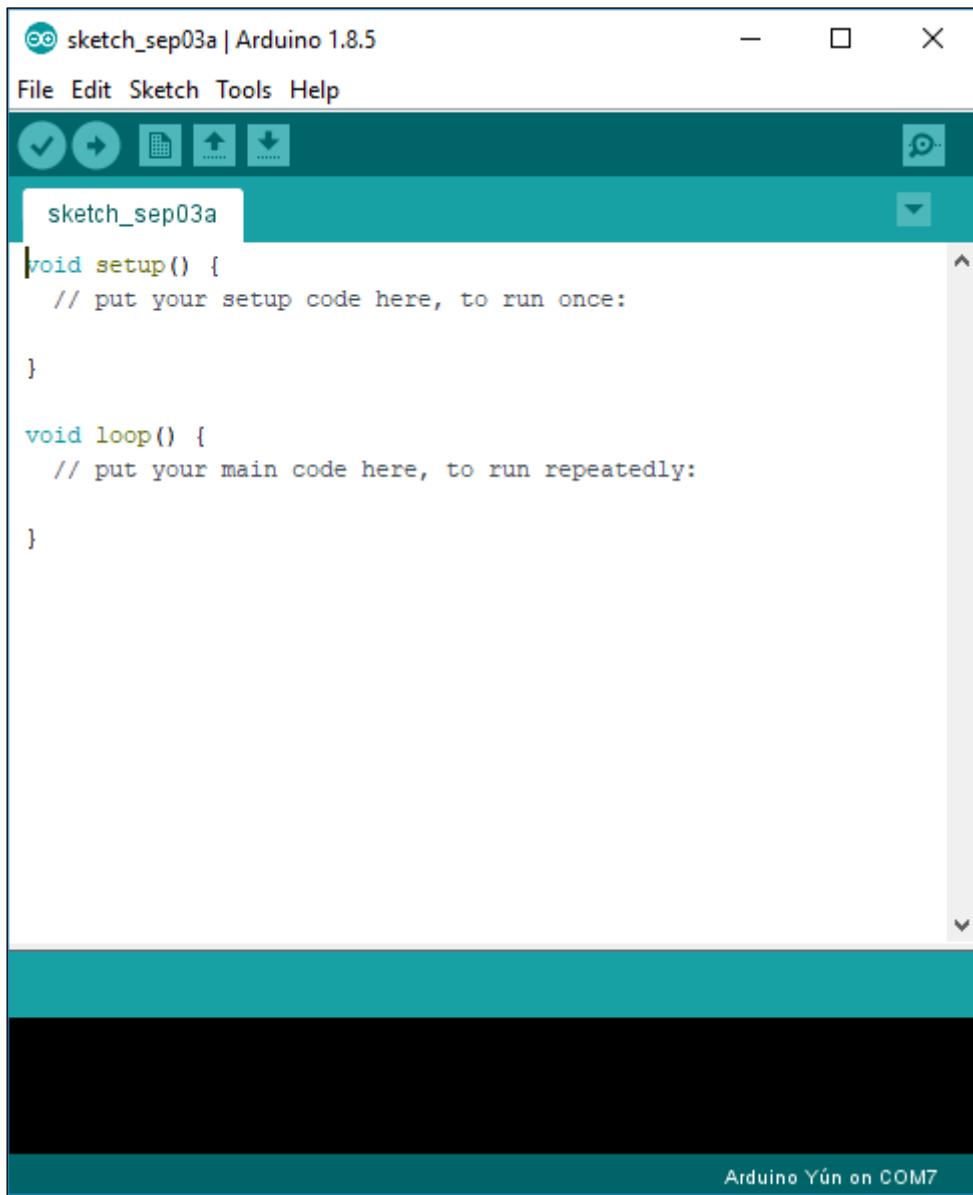
Note: For more information about the Arduino hardware parts, visit the [Arduino official web page](#).

Downloading the Arduino IDE

The Arduino IDE (Integrated Development Environment) is where you develop your programs that will tell the Arduino what to do.

You can load new programs onto the main chip, the ATmega328p, via USB using the Arduino IDE. To download the Arduino IDE, please click on the following link: <https://www.arduino.cc/en/Main/Software>. Select which Operating System you're using and download it. Then, simply follow the installation wizard to install the Arduino IDE.

When you first open the Arduino IDE, you should see something similar to the figure below:

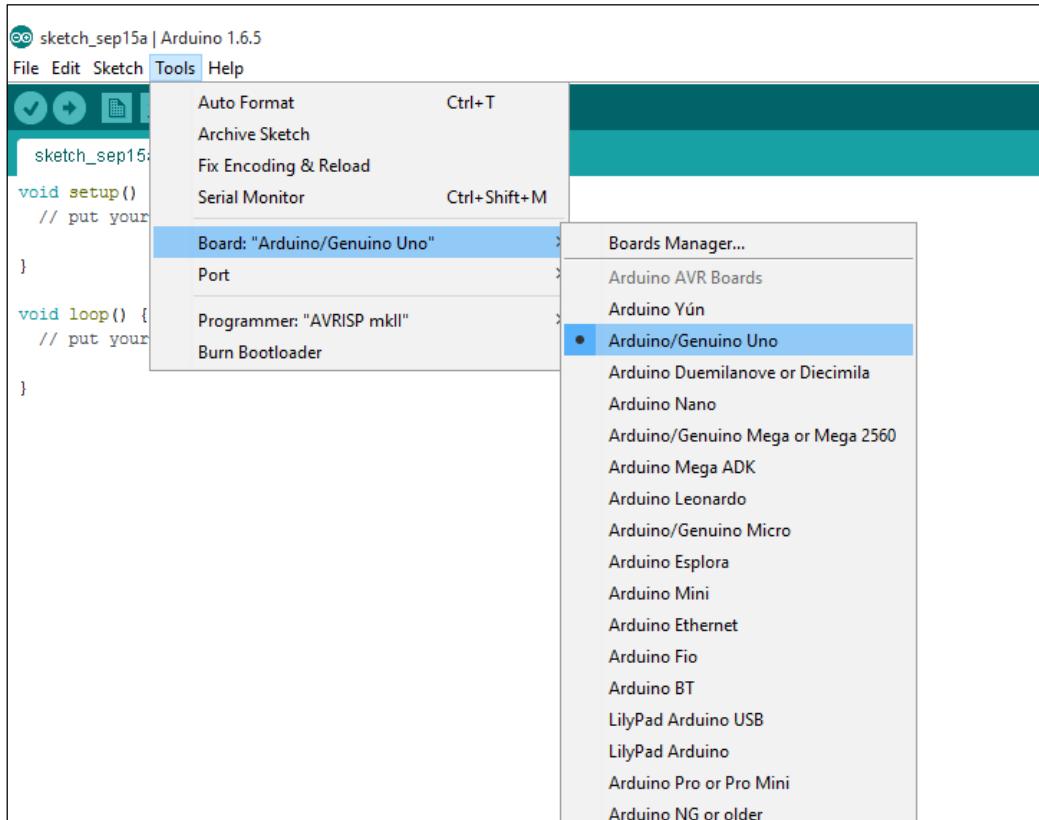


Connecting your Arduino

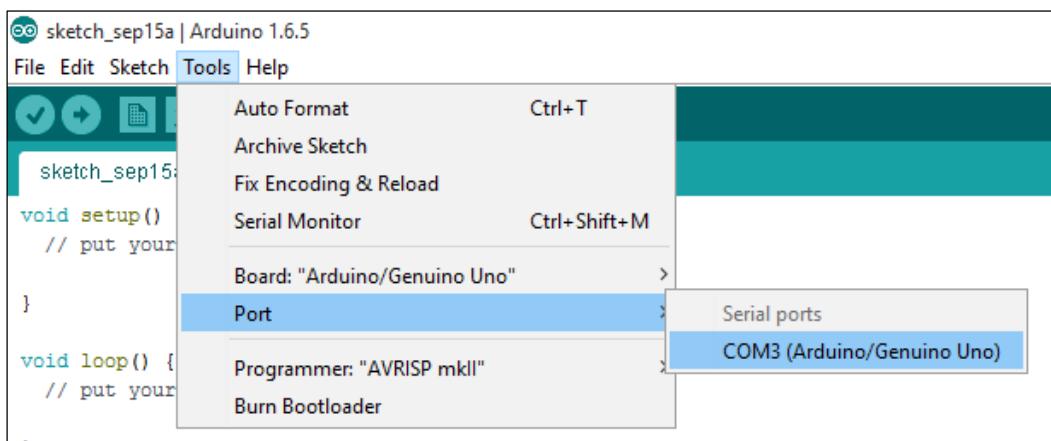
Connect your Arduino UNO to your computer via USB.

After connecting your Arduino with a USB cable, you need to make sure that the Arduino IDE has selected the right board.

In our case, we're using Arduino Uno, so we should go to **Tools** ▶ **Board:** ▶ **Arduino/Genuino Uno.**



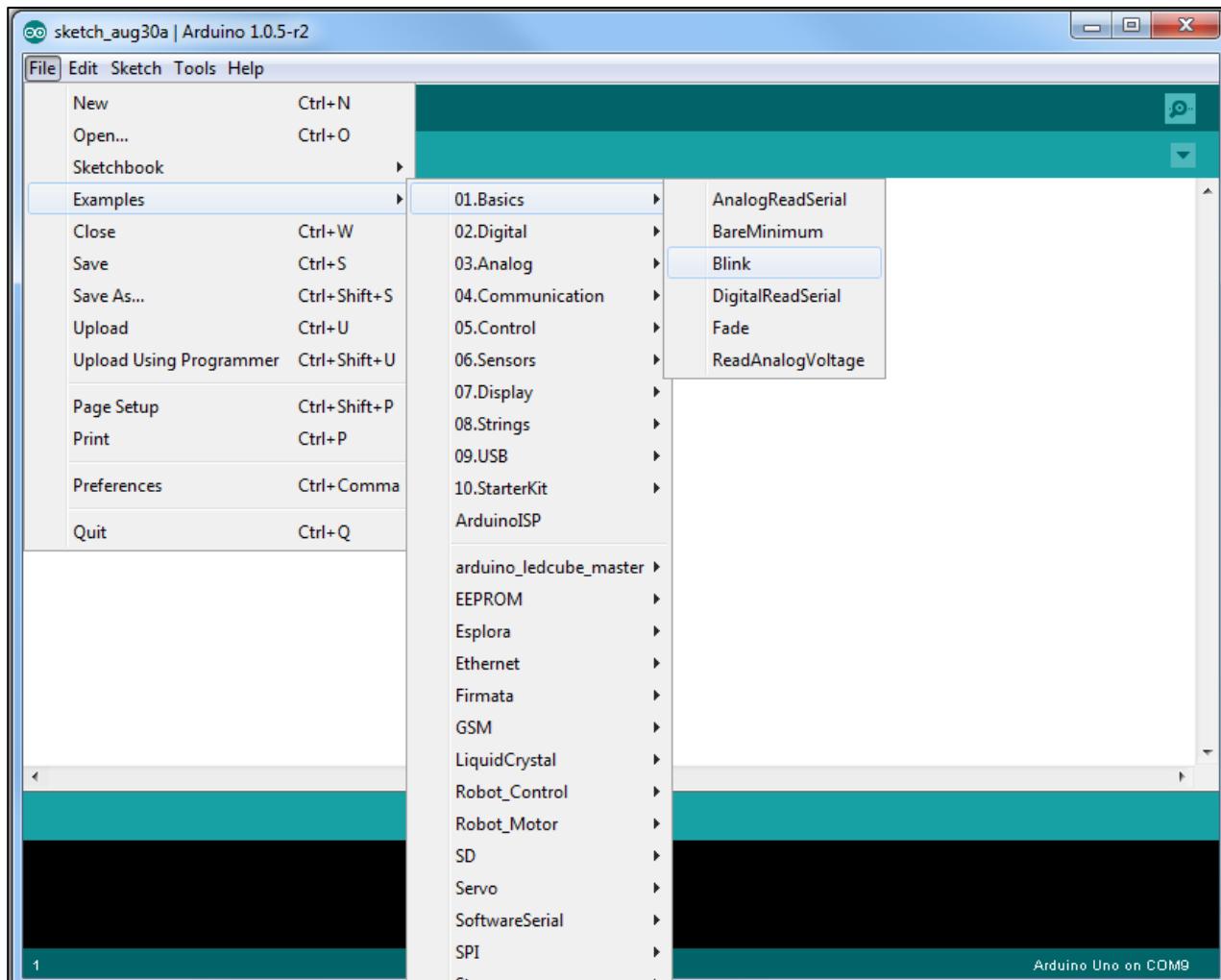
Then, you should select the serial port where your Arduino is connected to. Go to **Tools** ▶ **Port** and select the right port.



Uploading an Arduino Sketch

To show you how to upload code to your Arduino board, we'll show you a simple example. This is one of the most basic examples – it consists in blinking the on-board LED or digital pin 13 every second.

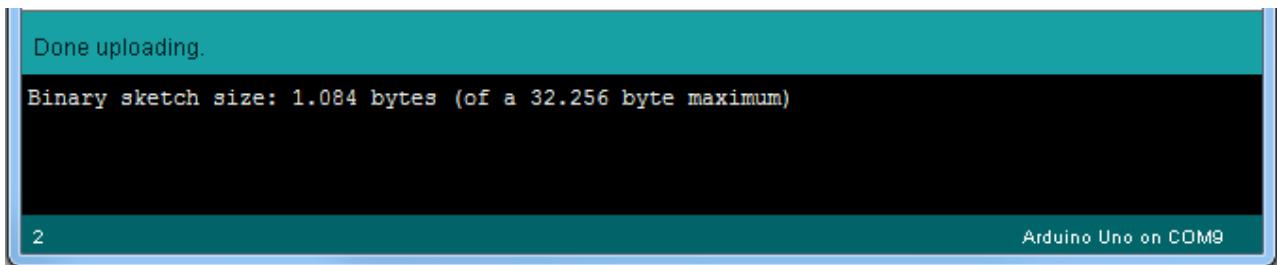
1. Open your Arduino IDE.
2. Go to **File** ▶ **Examples** ▶ **01.Basics** ▶ **Blink**



By default, the Arduino IDE comes pre-configured for the Arduino UNO. Click the **Upload** button and wait a few seconds.



After a few seconds, you should see a **Done uploading** message.



The screenshot shows the Arduino IDE's serial monitor window. The top bar is teal with the text "Done uploading.". Below it is a black bar with the text "Binary sketch size: 1.084 bytes (of a 32.256 byte maximum)". At the bottom left is a small number "2", and at the bottom right is the text "Arduino Uno on COM9".

This code simply blinks the on-board LED on your Arduino UNO (highlighted with red color). You should see the little LED turn on for one second, and turn off for another second repeatedly.



Control an Output and Read an Input

An Arduino board contains digital pins, analog pins and PWM pins.

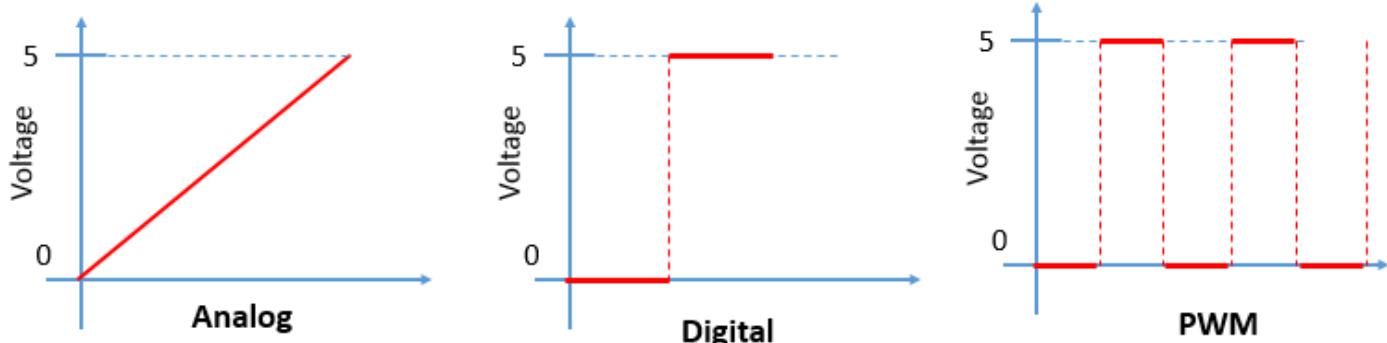
Difference between digital, analog and PWM

In **digital pins**, you have just two possible states, which are on or off. These can also be referred as High or Low, 1 or 0 and 5V or 0V.

For example, if an LED is on, then, its state is High or 1 or 5V. If it is off, you'll have Low, or 0 or 0V.

In **analog pins**, you have unlimited possible states between 0 and 1023. This allows you to read sensor values. For example, with a light sensor, if it is very dark, you'll read 1023, if it is very bright you'll read 0. If there is a brightness between dark and very bright you'll read a value between 0 and 1023.

PWM pins are digital pins, so they output either 0 or 5V. However these pins can output “fake” intermediate voltage values between 0 and 5V, because they can perform “Pulse Width Modulation” (PWM). PWM allows to “simulate” varying levels of power by oscillating the output voltage of the Arduino.



Controlling an output

To control a digital output you use the **digitalWrite()** function and between brackets you write, the pin you want to control, and then HIGH or LOW.

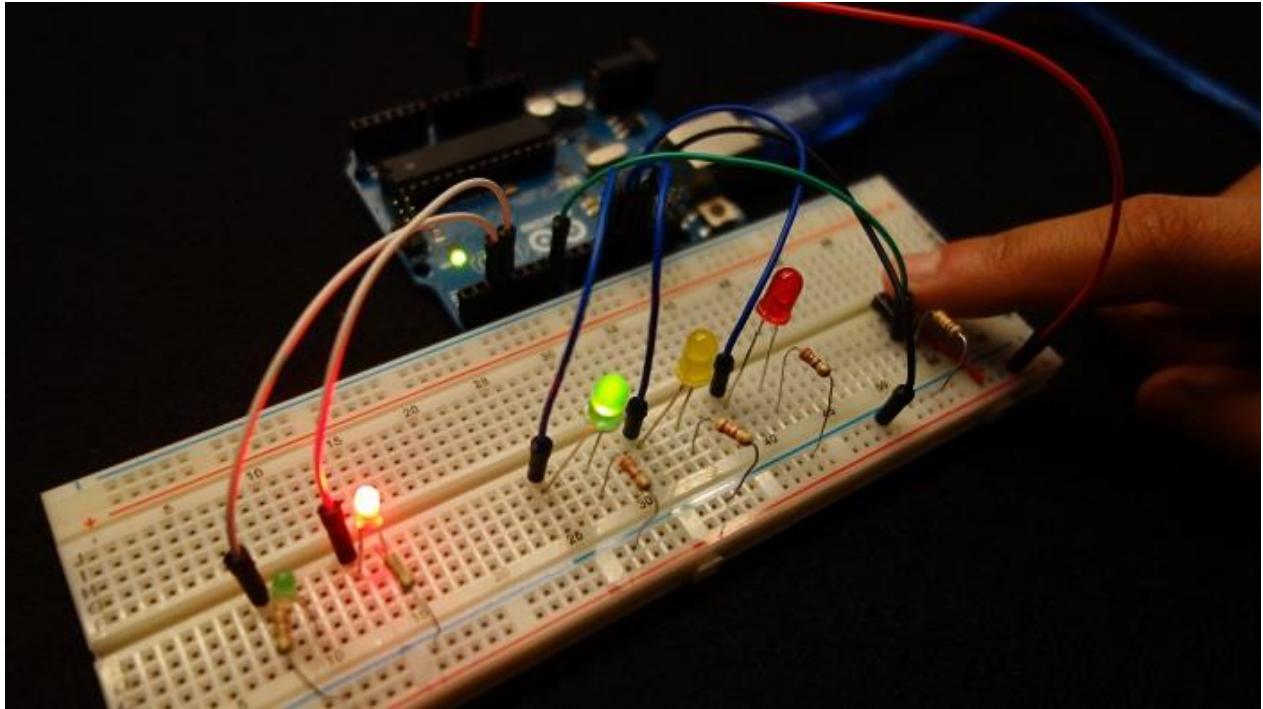
To control a PWM pin you use the **analogWrite()** function and between brackets you write the pin you want to control and a number between 0 and 255.

Reading an input

To read an analog input you use the function **analogRead()** and for a digital input you use **digitalRead()**.

The best way for you to learn Arduino is practising. So, choose a project and start building something.

Traffic Lights



[View code on GitHub](#)

Click [here](#)

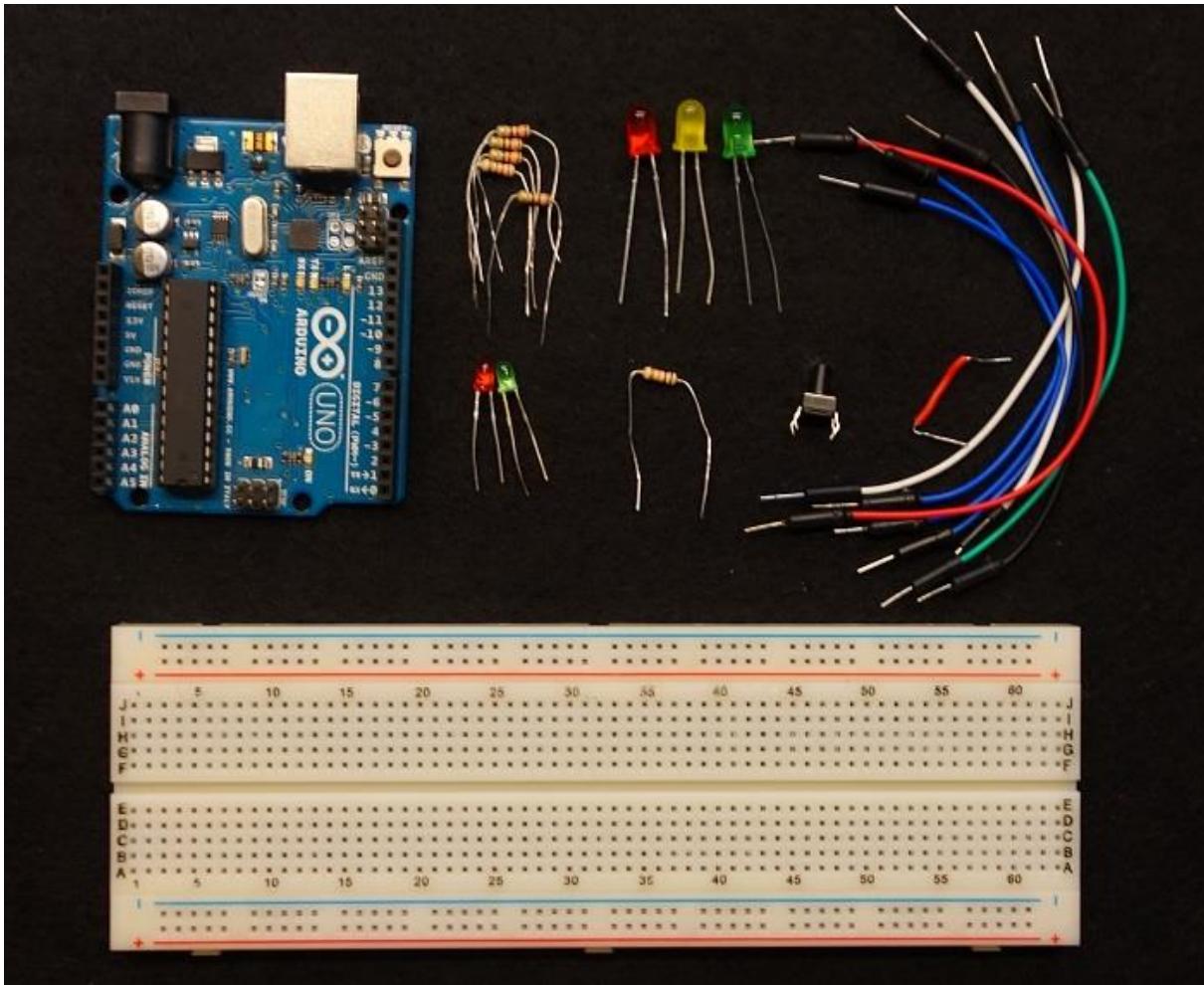
Introduction

In this project you're going to build a traffic lights system:

- There are 3 LEDs with different colors (green, yellow and red) to mimic the traffic lights for the cars
- There are 2 LEDs with different colors (green and red) to mimic the traffic lights for the pedestrians
- There is a pushbutton to mimic the ones in the pedestrians traffic lights

Parts Required

Grab all the needed components for this project.

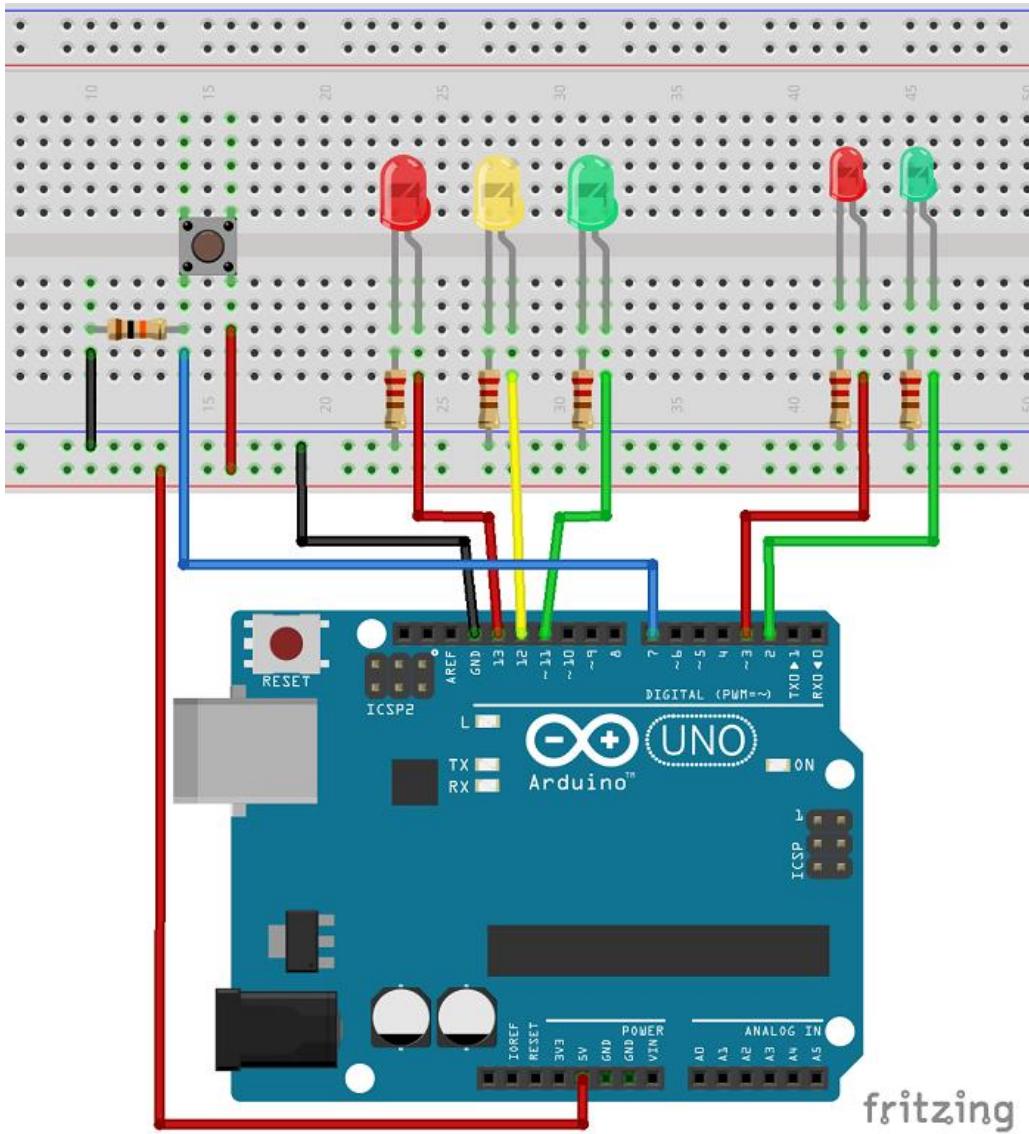


- [1x Breadboard](#)
- [Arduino UNO](#) – read [Best Arduino Starter Kits](#)
- [3x 5mm LED](#) (1x red, 1x yellow, 1x green)
- [2x 3mm LED](#) (1x red, 1x green)
- [5x 220Ohm Resistor](#)
- [1x 10kOhm Resistor](#)
- [1x pushbutton](#)
- [Jumper Wires](#)

I'm using LEDs of different sizes but if you don't have LEDs of different sizes, it is ok.
The project still works.

Schematics

Assemble all the parts by following the schematics below.



Code

You don't need any library for this code. The code is very simple. Here's some tips to better understand what's going on.

- The car light is always green, and so the pedestrian light is always red unless someone presses the button.
 - When someone presses the button here's what happens:
 - The car light changes to yellow and then to red
 - The pedestrian light changes to green

- The lights are in this state for a while (in the code this time is the variable crossTime)
- The pedestrian green light flashes and goes to red
- The car light changes from red to green

All these actions will be inside the function **changeLights()**. Everytime you want to change the lights, you just need to call the **changeLights()** function.

Copy the following code to your Arduino IDE, and upload it to your Arduino board. Make sure you have the right board and COM port selected.

[View code on GitHub](#)

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

int redCar = 13;
int yellowCar = 12;
int greenCar = 11;
int greenPed = 2;
int redPed = 3;
int button = 7;
int crossTime = 2000;
unsigned long changeTime;

void setup() {
    // initialize timer
    changeTime = millis();
    // here we are initializing our pins as outputs
    pinMode(redCar, OUTPUT);
    pinMode(yellowCar, OUTPUT);
    pinMode(greenCar, OUTPUT);
    pinMode(redPed, OUTPUT);
    pinMode(greenPed, OUTPUT);
    pinMode(button, INPUT);
    //turn on the green light
    digitalWrite(greenCar, HIGH);
    digitalWrite(redPed, HIGH);
    digitalWrite(redCar, LOW);
    digitalWrite(yellowCar, LOW);
    digitalWrite(greenPed, LOW);
```

```

Serial.begin(9600);
}

void loop() {
    // this variable will tell us if the button is pressed
    int state = digitalRead(button);
    Serial.println(state);
    // if the button is pressed and if it has passed 5 seconds since last button
    // press
    if (state == HIGH && (millis() - changeTime) > 5000) {
        //call the function to change the lights
        changeLights();
    }
}

void changeLights() {
    digitalWrite(greenCar, LOW);      // the green LED will turn off
    digitalWrite(yellowCar, HIGH); // the yellow LED will turn on for 2 second
    delay(2000);

    digitalWrite(yellowCar, LOW); // the yellow LED will turn off
    digitalWrite(redCar, HIGH); // the red LED will turn on for 5 seconds

    digitalWrite(redPed, LOW);
    digitalWrite(greenPed, HIGH);
    delay(crossTime);

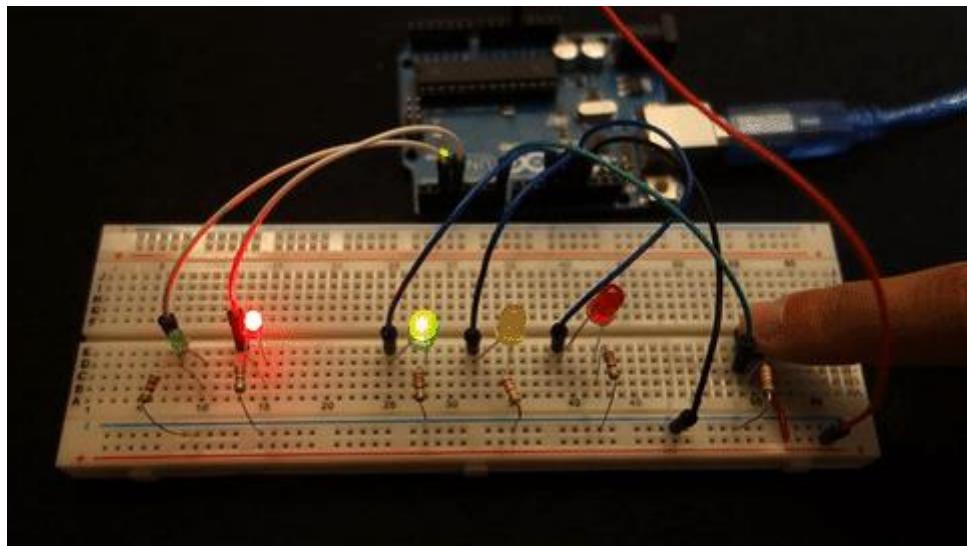
    // flash the ped green
    for (int x=0; x<10; x++) {
        digitalWrite(greenPed, LOW);
        delay(100);
        digitalWrite(greenPed, HIGH);
        delay(100);
    }
    digitalWrite(greenPed, LOW);
    digitalWrite(redCar, LOW);
    digitalWrite(redPed, HIGH);
    digitalWrite(greenCar, HIGH);

    changeTime = millis();
}

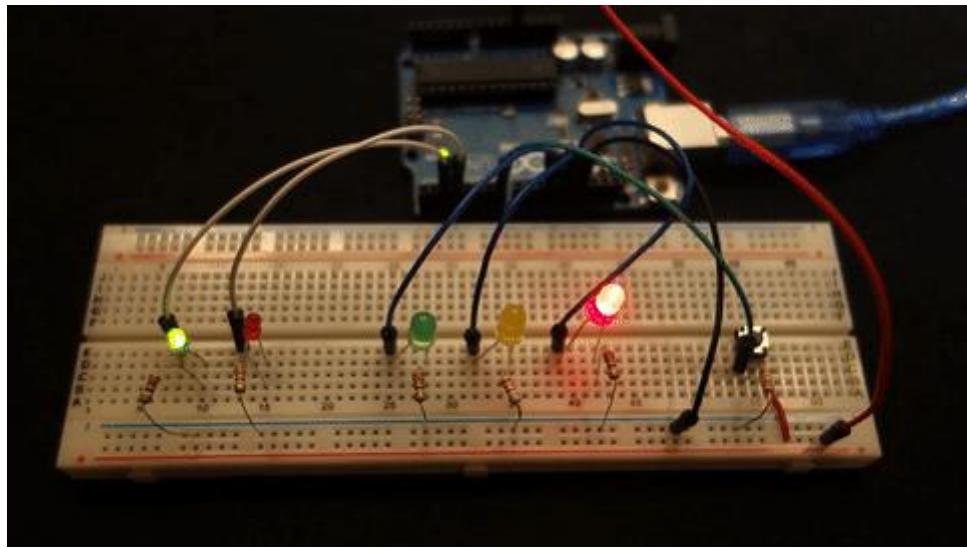
```

Demonstration

When you press the button, the light for the cars changes from green to red, and the pedestrian light changes from red to green.



After the crosstime, the pedestrian green led flashes and changes to red. The light for the cars changes from red to green.

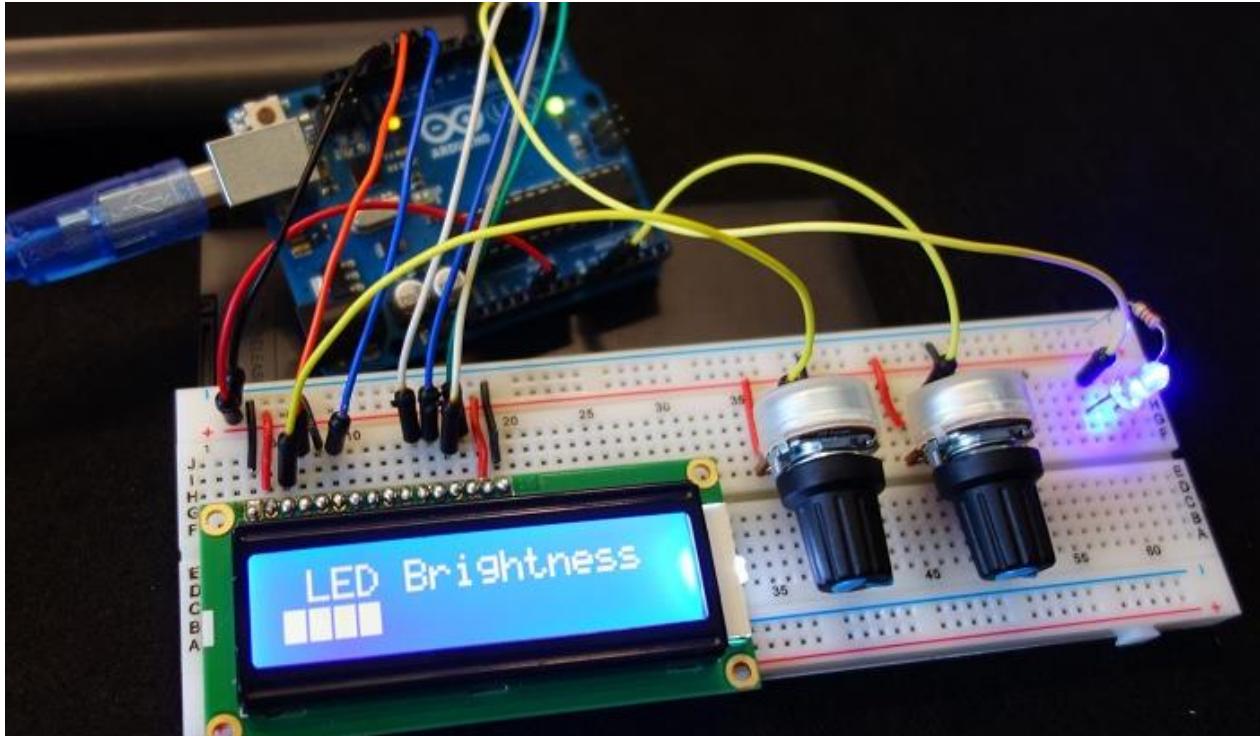


Wrapping Up

If you're starting with the Arduino, a good exercise is to change the value of some variables like **crossTime** and **changeTime** and see what happens.

If you want something a little bit more challenging, try to mimic what happens in a junction, with several lights for several cars and pedestrians.

LED Brightness on a 16x2 LCD



View Project on Random Nerd Tutorials	Click here
Watch on YouTube	Click here
View code on GitHub	Click here

Introduction

This is a beginner project where you'll use a 16x2 LCD to display the LED brightness. Shortly, in this project we'll control an LED brightness using a potentiometer. The LED brightness will be displayed on the LCD screen using a progress bar

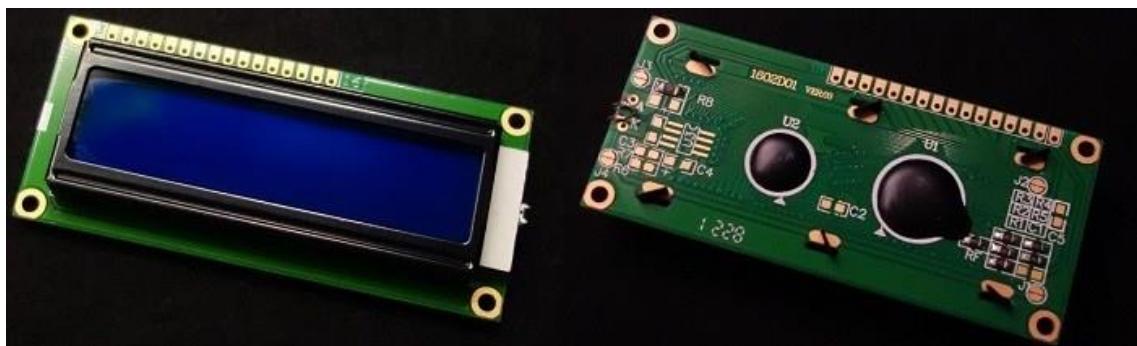
Watch the video below



Watch on YouTube: <http://youtu.be/sAklcqiywPw>

Introducing the LCD

The simplest and inexpensive way to display information is with an LCD (liquid crystal display). These are found in everyday electronics devices such as vending machines, calculators, parking meters, printers, and so on. These are ideal for displaying text or small icons. The figure below shows a 16×2 LCD front and back view.



This LCD has 2 rows, and each row can display 16 characters. It also has LED backlight to adjust the contrast between the characters and the background.

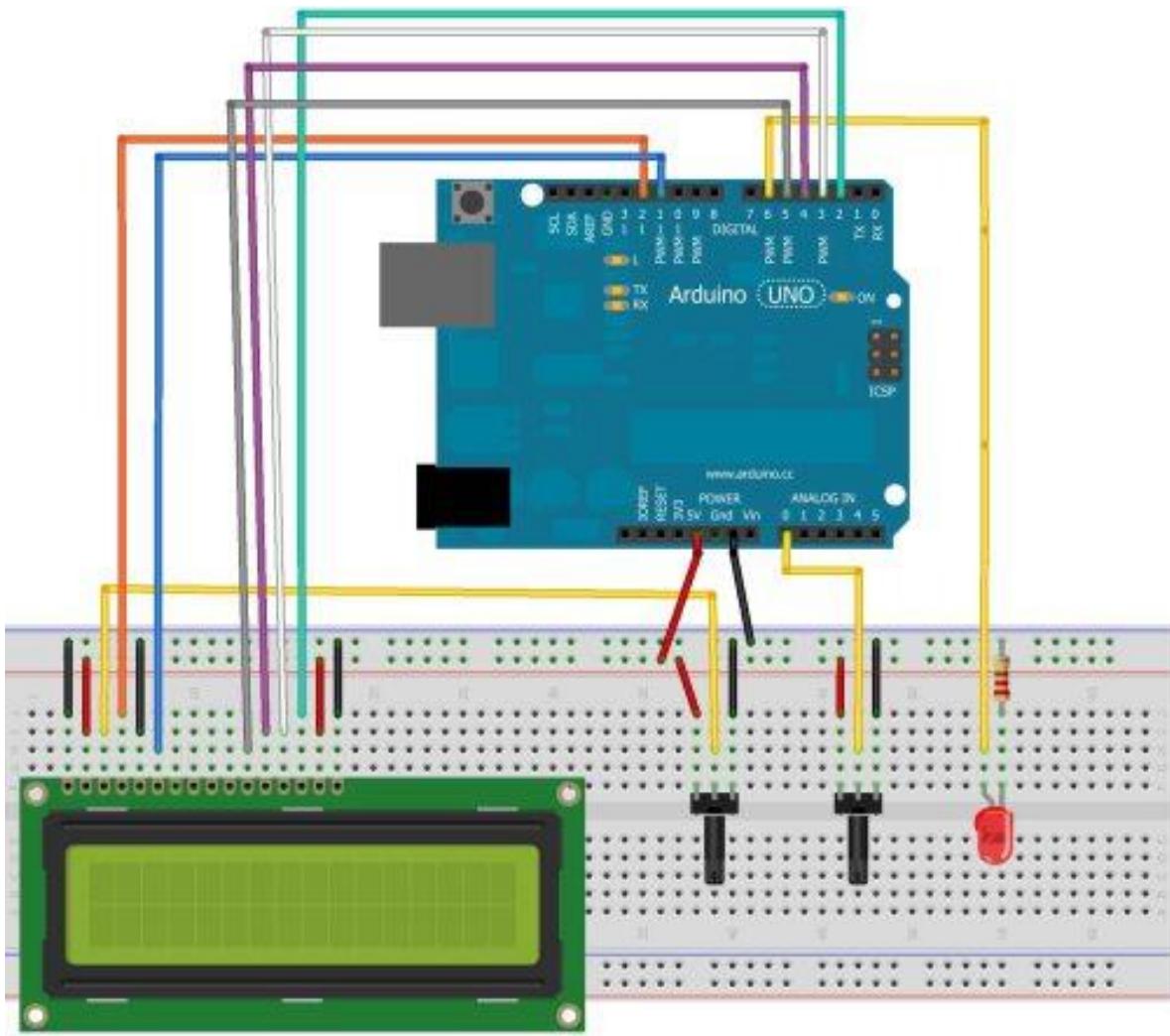
Parts Required

For this project you need the following parts:

- [Arduino UNO](#) – read [Best Arduino Starter Kits](#)
- [1x Breadboard](#)
- [1x LCD 16×2](#)
- [2x 10k Ohm Potentiometers](#)
- [1x 5mm LED](#)
- [1x 220Ohm Resistor](#)
- [Jumper wires](#)

Schematics

Wire all the parts by following the next schematic diagram.



The next table shows a brief description of each pin of the LCD display. Make sure your LCD uses the same pinout.

INTERFACE PIN CONNECTIONS

Pin No.	Symbol	Level	Description
1	VSS	---	Ground for Logic (0V)
2	VDD	---	Power supply for Logic (+5.0V)
3	V0	---	Power supply for LCD drive
4	RS	H/L	Register selection (H:Data register,L:Instruction register)
5	R/W	H/L	Read/Write selection (H:read,L:Write)
6	E	H/L → L	Enable signal for LCM
7-14	DB0~DB7	H/L	Data Bus Lines
15	LEDA	---	Power supply for backlight(+5.0V)
16	LEDK	---	Power supply for backlight(-)

Code

Copy the following code and upload it to your Arduino board. The code is well commented so that you can easily understand how it works, and modify it to include in your own projects.

[View code on GitHub](#)

```
/*
Created by Rui Santos

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

Based on some Arduino code examples
*/



// include the library code
#include <LiquidCrystal.h>

// initialize the library with the numbers of the interface pins
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);
int potPin = A0;          // Analog pin 0 for the LED brightness potentiometer
int ledPin = 6;           // LED Digital Pin with PWM
int potValue = 0;          // variable to store the value coming from the
potentiometer
int brightness = 0;        // converts the potValue into a brightness
int pBari = 0;             // progress bar
int i = 0;                 // foor loop

//progress bar character for brightness
byte pBar[8] = {
    B11111,
    B11111,
    B11111,
    B11111,
    B11111,
    B11111,
```

```

B1111,
};

void setup() {
    // setup our led as an OUTPUT
    pinMode(ledPin, OUTPUT);
    // set up the LCD's number of columns and rows:
    lcd.begin(16, 2);
    // Print a message to the LCD
    lcd.print(" LED Brightness");
    //Create the progress bar character
    lcd.createChar(0, pBar);
}

void loop() {
    // clears the LCD screen
    lcd.clear();
    // Print a message to the LCD
    lcd.print(" LED Brightness");
    //set the cursor to line number 2
    lcd.setCursor(0,1);
    // read the value from the potentiometer
    potValue = analogRead(potPin);
    // turns the potValue into a brightness for the LED
    brightness=map(potValue, 0, 1024, 0, 255);
    //lights up the LED according to the bightness
    analogWrite(ledPin, brightness);
    // turns the brightness into a percentage for the bar
    pBari=map(brightness, 0, 255, 0, 17);
    //prints the progress bar
    for (i=0; i<pBari; i++)
    {
        lcd.setCursor(i, 1);
        lcd.write(byte(0));
    }
    // delays 750 ms
    delay(750);
}

```

Complete Guide for Ultrasonic Sensor HC-SR04 with Arduino



View Project on Random Nerd Tutorials

Click [here](#)

View code on GitHub

Click [here](#)

Click [here](#)

This project is about the Ultrasonic Sensor HC – SR04. We'll explain how it works, show some features and share an Arduino Project example. We provide a schematic diagram on how to wire the ultrasonic sensor, and an example sketch to use with your Arduino.

Description

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 2cm to 400 cm or 1" to 13 feet. Its operation is not affected by sunlight or black material like sharp rangefinders are (although acoustically soft materials like cloth can be difficult to detect). It comes complete with ultrasonic transmitter and receiver module.

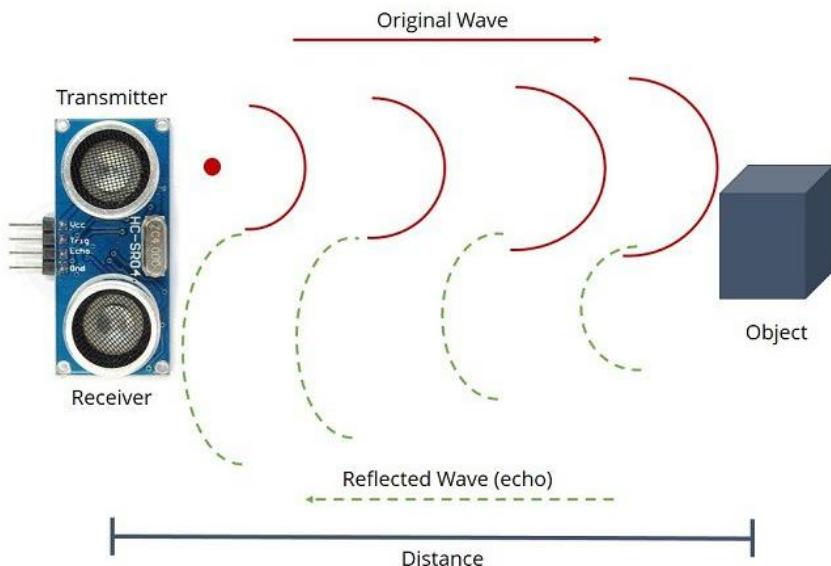
Features

- 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

How Does it Work?

The ultrasonic sensor uses sonar to determine the distance to an object. Here's what happens:

1. The transmitter (trig pin) sends a signal: a high-frequency sound;
2. When the signal finds an object, it is reflected and...
3. The transmitter (echo pin) receives it.



The time between the transmission and reception of the signal allows us to calculate the distance to an object. This is possible because we know the sound's velocity in the air.

Sensor Pins



- VCC: +5VDC
- Trig : Trigger (INPUT)
- Echo: Echo (OUTPUT)
- GND: GND

Where to buy?

You can check the [Ultrasonic Sensor HC-SR04 sensor on Maker Advisor](#) and find the best price.



Arduino with HC - SR04 Sensor

In this project the ultrasonic sensor reads and writes the distance to an object in the Serial Monitor.

The goal of this project is to help you understand how this sensor works. Then, you can use this example in your own projects.

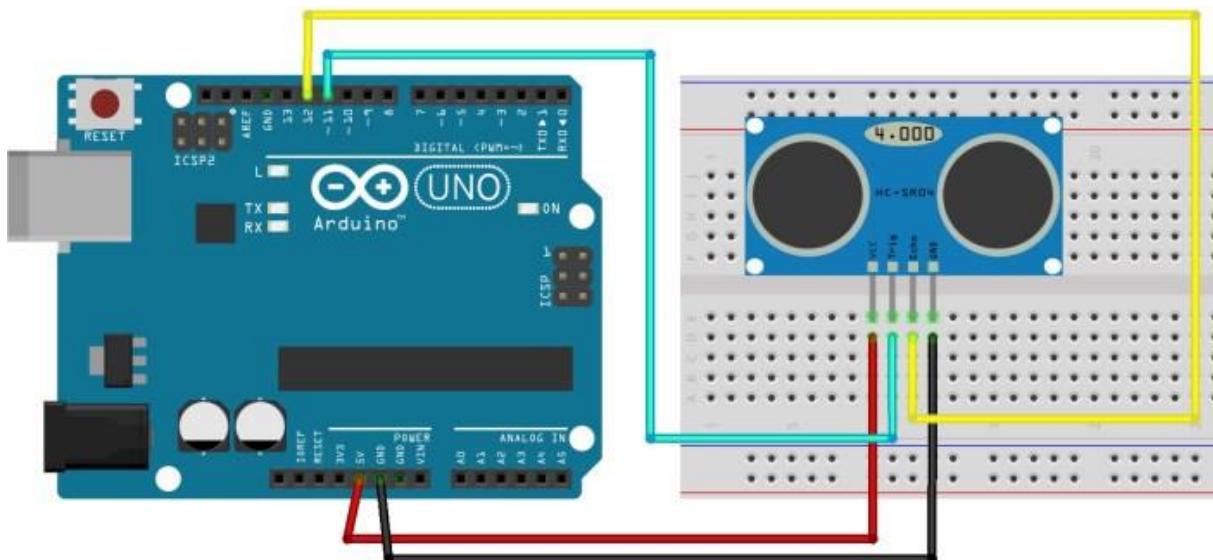
Note: There's an Arduino library called [NewPing](#) that can make your life easier when using this sensor.

Parts Required

- [Arduino UNO](#) – read [Best Arduino Starter Kits](#)
- [Ultrasonic Sensor \(HC-SR04\)](#)
- [Breadboard](#)
- [Jumper wires](#)

Schematics

Follow the next schematic diagram to wire the HC-SR04 ultrasonic sensor to the Arduino.



The following table shows the connections you need to make:

Ultrasonic Sensor HC-SR04	Arduino
VCC	5V
Trig	Pin 11
Echo	Pin 12
GND	GND

Code

Upload the following code to your Arduino IDE.

[View code on GitHub](#)

```
/*
 * 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;      //Trig - green Jumper
int echoPin = 12;      //Echo - yellow Jumper
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);
}

```

Code with NewPing

You can also use the the NewPing library. Download the library [here](#). After installing the NewPin library, you can upload the code provided below.

[View code on GitHub](#)

```

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

#include <NewPing.h>

#define TRIGGER_PIN 11
#define ECHO_PIN 12
#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");
}

```

Troubleshooting

NOTE: "If the HC-SR04 does not receive an echo then the output never goes low. Devantec and Parallax sensors time out after 36ms and I think 28ms respectively. If you use Pulsin as above then with no return echo the program will hang for 1 second which is the default timeout for Pulsin. You need to use the timeout parameter.

<http://arduino.cc/en/Reference/PulseIn>

The HC-SR04 barely works to 10 feet giving a total path length of 20 feet and a path time of about 20ms so set the timeout to something above that, say 25 or 30ms.

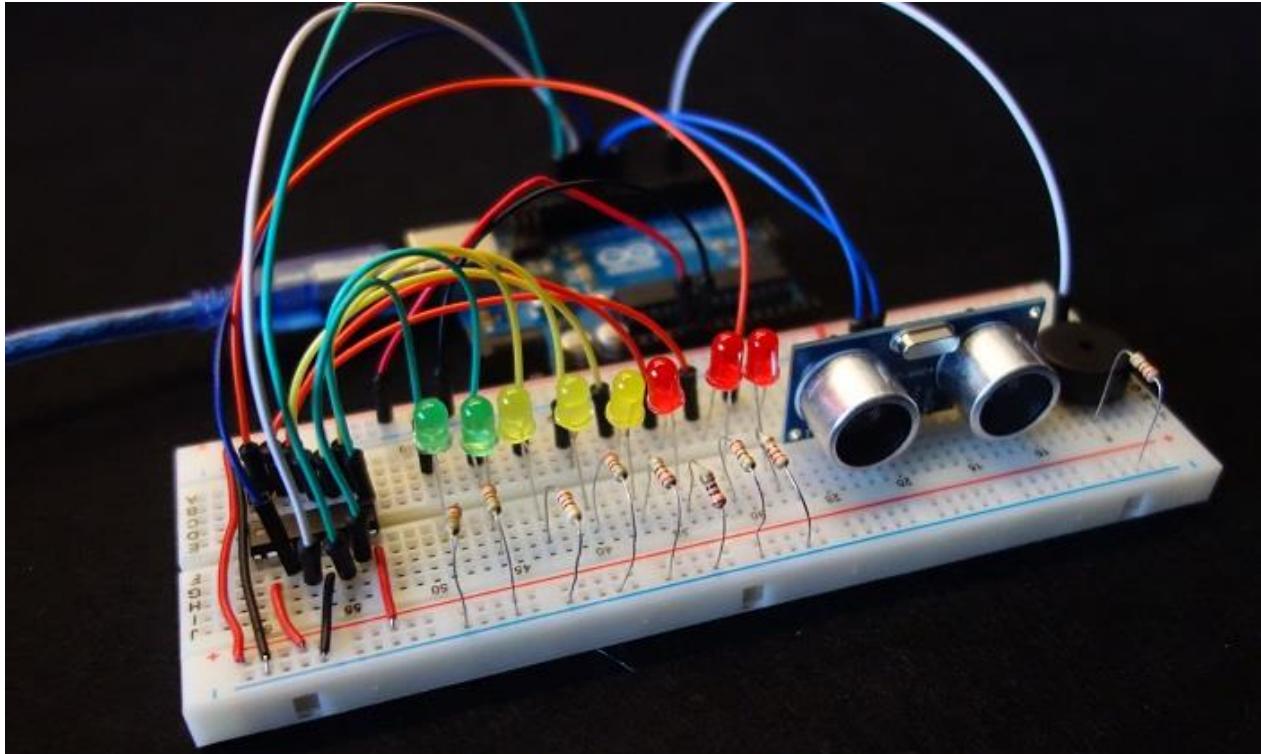
If you put a resistor, say 2k2 between E and T then only connect to T you can use the HC-SR04 from just one Arduino pin. Look up single pin operation of ultrasonic sensors.

Also if you are using a HC-SR04 with a PicAxe you need to up the clockspeed to at least 8MHz otherwise they don't see the start of the echo pulse so pulsin never starts. The HC-SR04 works fine with a BS2." by David Buckley

Wrapping Up

In this post we've shown you how the HC-SR04 ultrasonic sensor works, and how you can use it with Arduino. If you are a beginner to the Arduino, we recommend following our [Arduino Mini Course](#) that will help you quickly getting started with this amazing board.

Parking Sensor



View Project on Random Nerd Tutorials

Click [here](#)

Watch on YouTube

Click [here](#)

View code on GitHub

Click [here](#)

Introduction

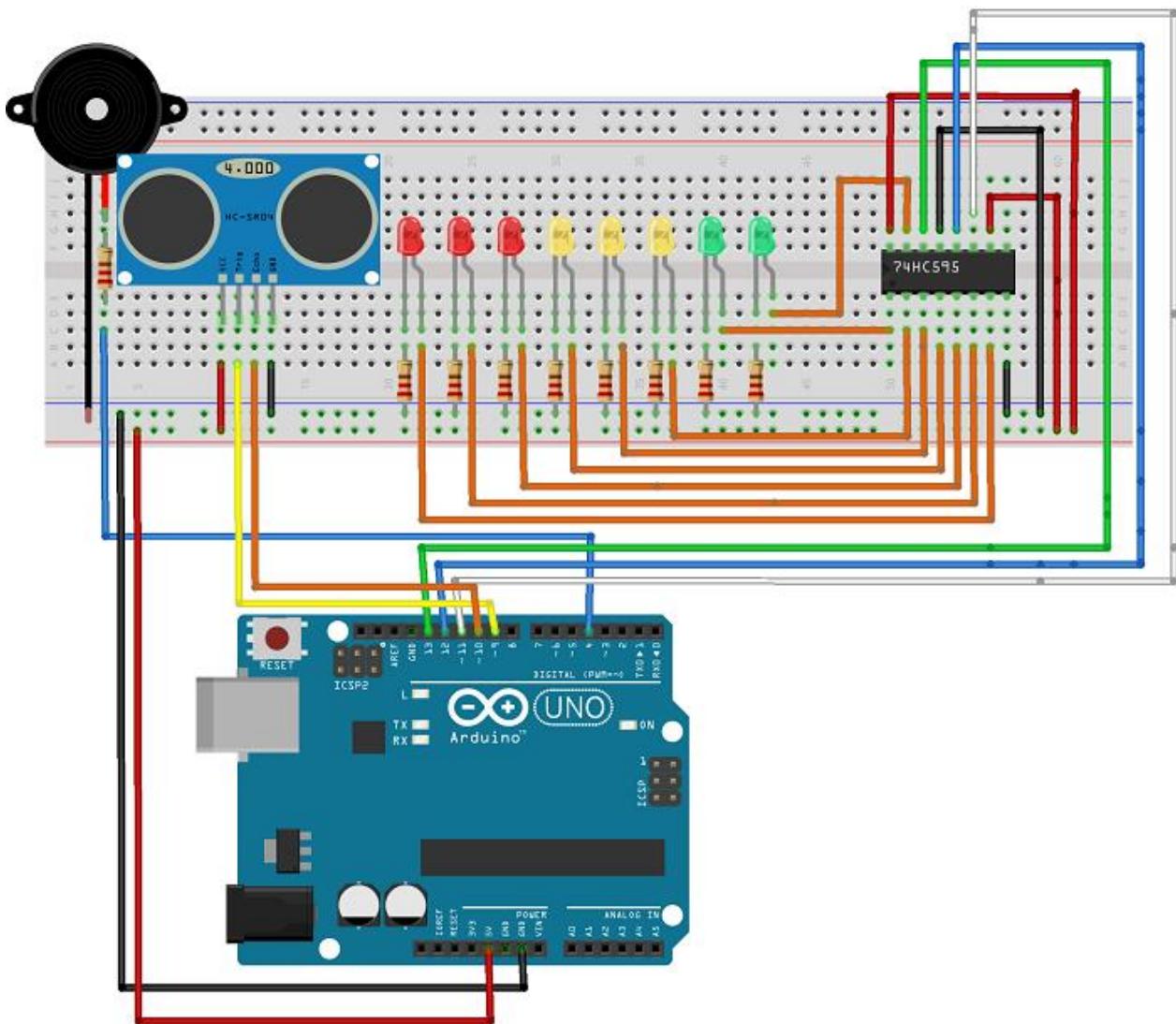
In this project we have an ultrasonic sensor that measures the distance and an LED bar graph that lights up accordingly to our distance from the sensor. As we get closer to the sensor the buzzer beeps in a different way. This circuit can work as a parking sensor.

Parts Required

- [Arduino UNO](#) – read [Best Arduino Starter Kits](#)
- 1x 74HC595 8 Bit Shift Register
- [1x Breadboard](#)
- [8x LEDs \(for example: 3x red, 3x yellow, 2x green\)](#)
- [9x 220 Ohm Resistors](#)
- [1x Buzzer](#)
- [1x Ultrasonic Sensor \(for example: HC-SR04\)](#)
- [Jumper Wires](#)

Schematics

Assemble the circuit by following the next schematic diagram:



fritzing

Code

Upload the following code to your Arduino board:

[View code on GitHub](#)

```
/*
 * created by Rui Santos, http://randomnerdtutorials.com
 * Ultrasonic Sensor with LED's bar graph and buzzer
*/
int tonePin = 4;      //Tone - Red Jumper
int trigPin = 9;      //Trig - violet Jumper
int echoPin = 10;     //Echo - yellow Jumper
int clockPin = 11;    //IC Pin 11 - white Jumper
int latchPin = 12;    //IC Pin 12 - Blue Jumper
int dataPin = 13;     //IC Pin 14 - Green Jumper

byte possible_patterns[9] = {
B00000000,
B00000001,
B00000011,
B00000111,
B00001111,
B00011111,
B00111111,
B01111111,
B11111111,
};

int proximity=0;
int duration;
int distance;

void setup() {
//Serial Port
Serial.begin (9600);

pinMode(trigPin, OUTPUT);
pinMode(echoPin, INPUT);
pinMode(clockPin, OUTPUT);
pinMode(latchPin, OUTPUT);
pinMode(dataPin, OUTPUT);
pinMode(tonePin, OUTPUT);
}
```

```

void loop() {
    digitalWrite(latchPin, LOW);
    digitalWrite(trigPin, HIGH);
    delayMicroseconds(1000);
    digitalWrite(trigPin, LOW);
    duration = pulseIn(echoPin, HIGH);
    distance = (duration/2) / 29.1;

    /*if (distance >= 45 || distance <= 0) {
        Serial.println("Out of range");
    }
    else {
        Serial.print(distance);
        Serial.println(" cm");
    }*/
}

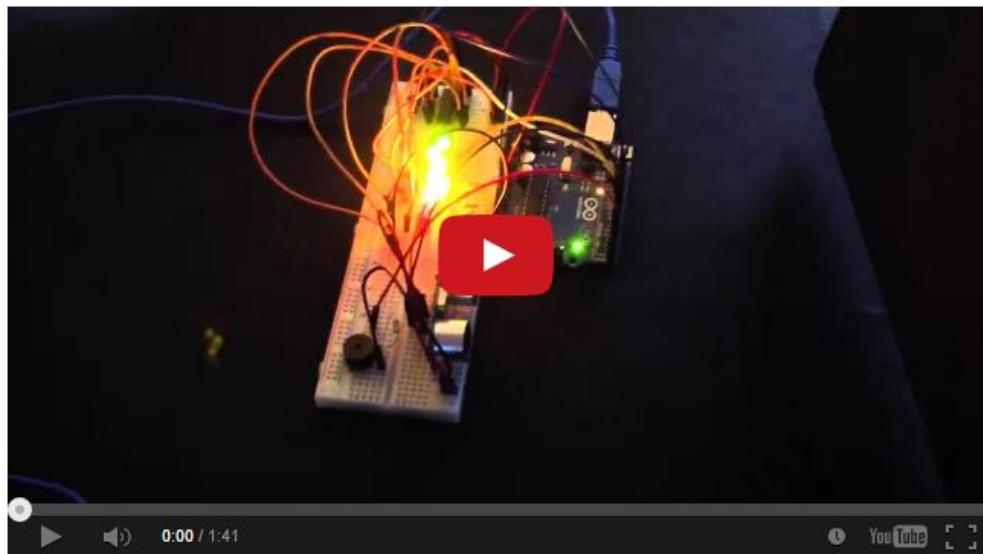
proximity=map(distance, 0, 45, 8, 0);
//Serial.println(proximity);

if (proximity <= 0){
    proximity=0;
}
else if (proximity >= 3 && proximity <= 4){
    tone(tonePin, 200000, 200);
}
else if (proximity >= 5 && proximity <= 6){
    tone(tonePin, 5000, 200);
}
else if (proximity >= 7 && proximity <= 8){
    tone(tonePin, 1000, 200);
}
shiftOut(dataPin, clockPin, MSBFIRST, possible_patterns[proximity]);
digitalWrite(latchPin, HIGH);

delay(600);
noTone(tonePin);
}

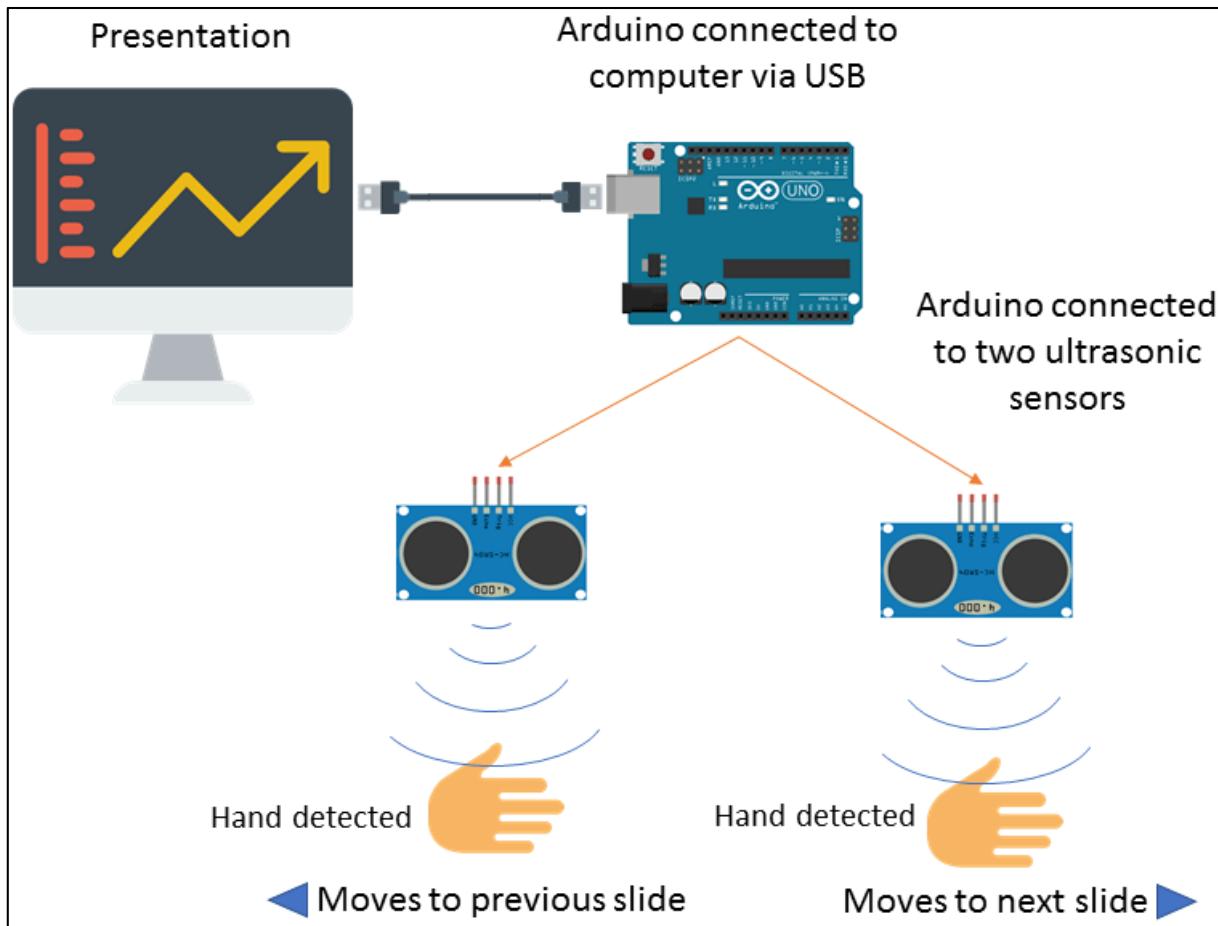
```

Watch the video demonstration



Watch on YouTube: http://youtu.be/7ZPc_5tL3c

Gesture Slider Swiper



View Project on Random Nerd Tutorials

Click [here](#)

View code on GitHub

Click [here](#)

In this project you're going to learn how to set an Arduino UNO board as a USB HID keyboard and use gestures to swipe slides during a presentation.

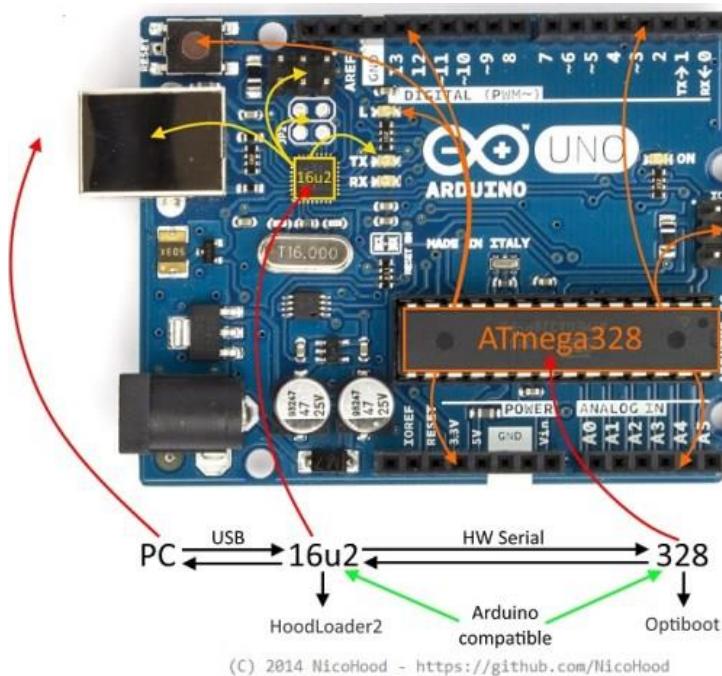
Note: This project was written by [Emmanuel Odunlade](#) and edited by [Rui Santos](#).

Arduino UNO or Mega as a USB HID Keyboard

The Arduino can be configured to emulate a keyboard and perform useful tasks. Attaching a pushbutton to an Arduino digital pin and with a button press, [you can write a password](#), copy+paste, up/down volume, up/down screen brightness, etc...

Although boards like the Arduino UNO can't do this by default if you flash a new bootloader, you can make the Arduino UNO board turn into a USB HID keyboard.

An Arduino Uno/Mega has two microcontrollers: ATmega328 and 16u2. The 16u2 is normally used for USB to Serial communication. We can also use it as standalone AVR Microcontroller with (or without) USB functions as well.



HoodLoader2 gives you the option to reprogram the 16u2 of a normal Arduino Uno/Mega R3 with custom sketches. This means you can use the 16u2 as a normal USB AVR like an Arduino Leonardo board. You have a full compatible USB-HID core.

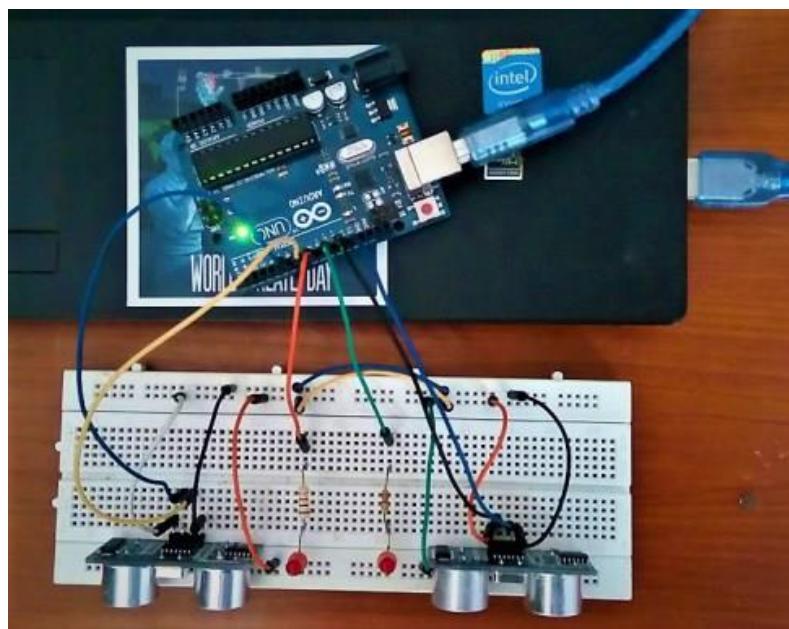
You can prepare your Arduino UNO or Arduino Mega to act as a keyboard with the [HoodLoader2](#) bootloader by following the instructions in their [official Wiki page](#).

- Supported Arduino boards using Arduino IDE 1.6.7 or higher:
- Uno (needs HoodLoader2 bootloader)
- Mega (needs HoodLoader2 bootloader)
- Leonardo
- (Pro) Micro

Parts Required

Here's a complete list of the components you need for this project:

- [Arduino UNO](#), Mega, Leonardo, ProMicro or any other 8u2/16u2/at90usb8/162/32u2/32u4 compatible board
- [2x Ultrasonic sensor – HC-SR04](#)
- [2x LEDs](#)
- [2x 220 ohm resistors](#)
- [Breadboard](#)
- [Jumper wires](#)



Code

To use your Arduino as a keyboard, you need to install the HID Library.

Installing the HID Library

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

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

[View code on GitHub](#)

```
/*
 * Author: Emmanuel Odunlade
 * Complete Project Details http://randomnerdtutorials.com
 */

#include <HID.h> // include the library

int swipe = 0; //slide
boolean left = false;
boolean right = false;
int maxD = 5; // (in cm) maximum distance from ultrasonic at which the
obstruction will be considered a gesture
long int lastTouch = -1;
int resetAfter = 1000; //ms
int afterslideDelay = 500; //
int slideleft_Begin = -1;
int slideNone = 0;
int slideright_Begin = 1;

//Declaring the connected pins
int lLed = 7;
int rLed = 6;
const int lEcho = 2;
const int lTrig = 3;
const int rEcho = 4;
const int rTrig = 5;

void setup() {
    pinMode(lLed, OUTPUT);
    pinMode(rLed, OUTPUT);
    pinMode(rEcho, INPUT);
    pinMode(rTrig, OUTPUT);
    pinMode(lEcho, INPUT);
    pinMode(lTrig, OUTPUT);

    Serial.begin(9600);
    // Sends a clean report to the host. This is important because
    // the 16u2 of the Uno/Mega is not turned off while programming
    // so you want to start with a clean report to avoid strange bugs after
    reset.
}
```

```

    pressRawKeyboard(0, 0);
}

//get distance
//echo is input, trigger is output
unsigned long measureD(int input, int output){
    digitalWrite(output, HIGH);
    delayMicroseconds(10);
    digitalWrite(output, LOW);
    long range= pulseIn(input, HIGH);
    int distance= range / 29 / 2;// to get distance in cm
    return distance;
}

// 
boolean act (int input, int output, int led){
    int d = measureD(input,output);
    boolean pinActivated = false;
    if (d < maxD){
        digitalWrite(led,HIGH);
        pinActivated = true;
    }
    else{
        digitalWrite(led,LOW);
        pinActivated = false;
    }
    return pinActivated;
}

void slideNow(char directn){
    if ('R' == directn)
    {
        // press the right rrow key
        //Serial.println("F");
        pressRawKeyboard(0,RAW_KEYBOARD_RIGHT_ARROW); //modifiers + key
        pressRawKeyboard(0, 0); // release! Important
    }
    if ('L' == directn)
    {
        //press the left arrow key
        //Serial.println("B"); for debug
        pressRawKeyboard(0,RAW_KEYBOARD_LEFT_ARROW); // //modifiers + key
        pressRawKeyboard(0, 0); // release! Important
    }
}

```

```

}

delay(afterslideDelay);
swipe = slideNone;
}

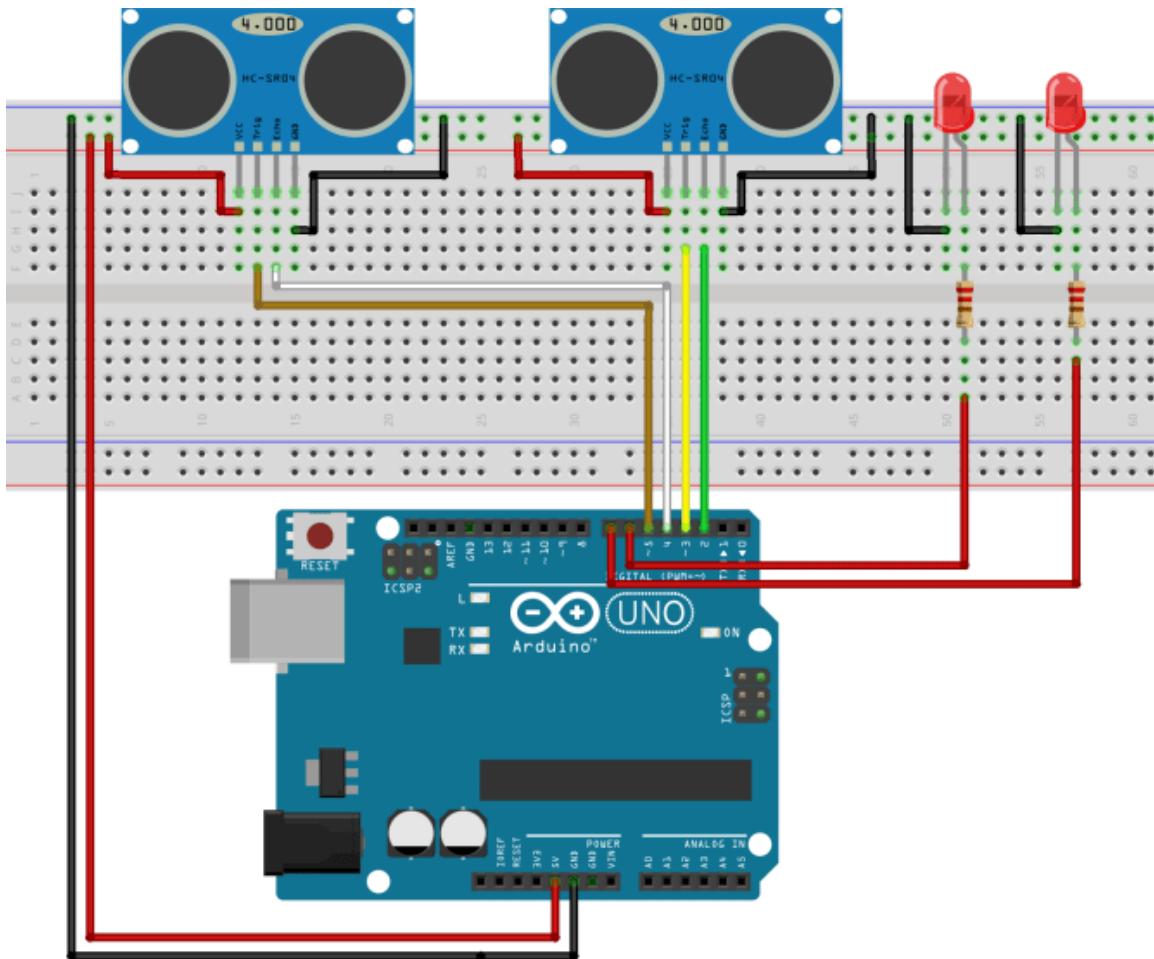
void pressRawKeyboard(uint8_t modifiers, uint8_t key) {
    uint8_t keys[8] = {
        modifiers, 0, key, 0, 0, 0, 0, 0      }; //modifiers, reserved, key[0]
    HID_SendReport(HID_REPORTID_KeyboardReport, keys, sizeof(keys));
}

void loop() {
    left = act(lEcho,lTrig,lLed);
    right = act(rEcho,rTrig,rLed);
    if (left || right){
        lastTouch = millis();
    }
    if (millis() - lastTouch > resetAfter){
        swipe = 0;
        //serial.println(@reset slide& timer);
    }
    if (swipe >= slideNone){
        if ((left)&&(!right)){
            swipe = slideright_Begin;
        }
        if ((right)&&(swipe == slideright_Begin)){
            slideNow('R');
        }
    }
    if (swipe <= slideNone ){
        if ((right)&&(!left)){
            swipe = slideleft_Begin;
        }
        if ((left) && (swipe == slideleft_Begin)){
            slideNow('L');
        }
    }
    delay(50);
}

```

Schematics

Wire your circuit accordingly to the schematic below:



Now, if you move a hand in front of the left ultrasonic sensor, your Arduino sends a Left keyboard command resulting in moving the presentation to the previous slide.

Wrapping Up

I have always been fascinated by natural user interfaces, the idea of things like the sixth sense (which I hope to share a tutorial on someday) and generally all things that makes telekinesis look like a possibility.

Thus I believe there are several updates that could be made to this project. Instead of using ultrasonic sensors and controlling sides what if we attached an Electroencephalogram (EEG) headset and then do more complex stuff like draw shapes and achieve all this within the simplicity frame the Arduino provides.

So the possibilities are endless. Hack it up!

Arduino with PIR Motion Sensor



View Project on Random Nerd Tutorials	Click here
Watch on YouTube	Click here
View code on GitHub	Click here

In this project we're going to create a simple circuit with an Arduino and PIR motion sensor that can detect movement.

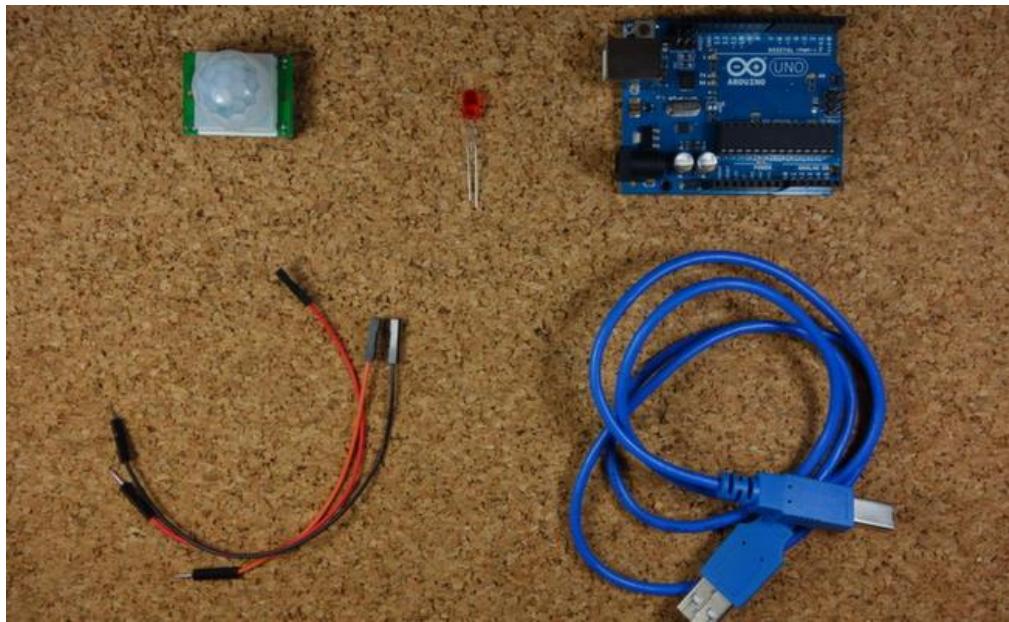
Watch the video below



Watch on YouTube: <http://youtu.be/vJgtckLzokM>

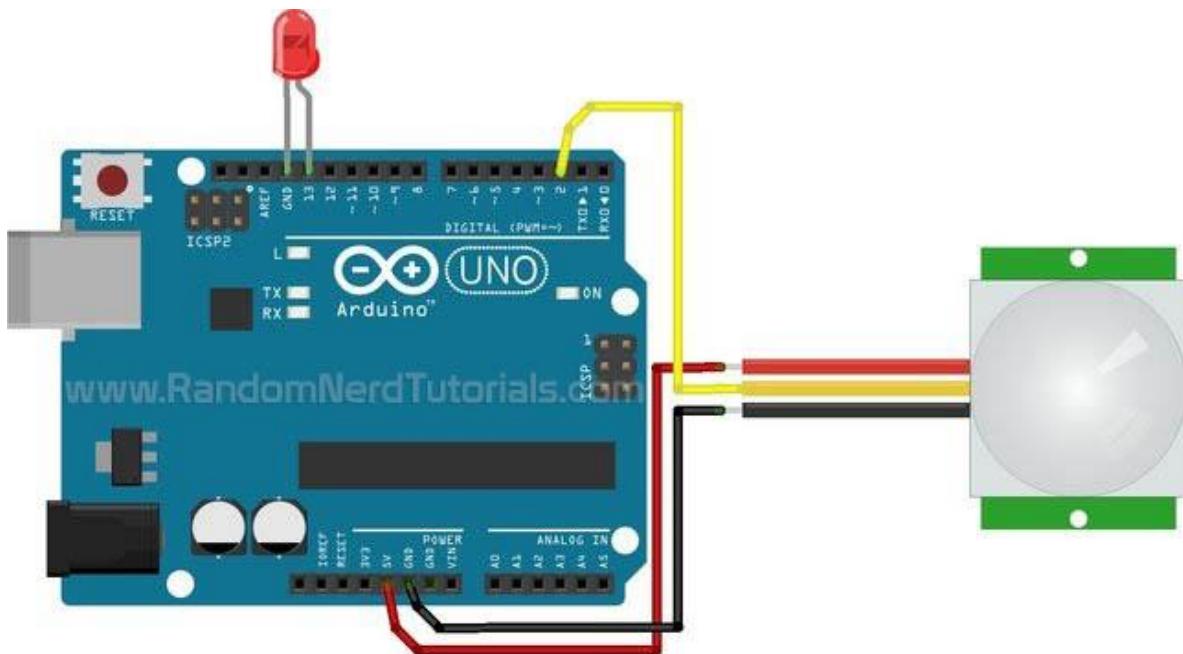
Parts Required

- [1x PIR Motion Sensor \(HC-SR501\)](#)
- [Arduino UNO](#) – read [Best Arduino Starter Kits](#)
- [1x LED](#)
- [Jumper wires](#)



Schematics

Assemble all the parts by following the schematic below.



Code

Upload the following code to your Arduino board.

[View code on GitHub](#)

```
/*
    Arduino with PIR motion sensor
    Modified by Rui Santos based on PIR sensor by Limor Fried
*/
s
int led = 13;                      // the pin that the LED is attached to
int sensor = 2;                     // the pin that the sensor is attached to
int state = LOW;                    // by default, no motion detected
int val = 0;                        // variable to store the sensor status (value)

void setup() {
    pinMode(led, OUTPUT);           // initialize 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

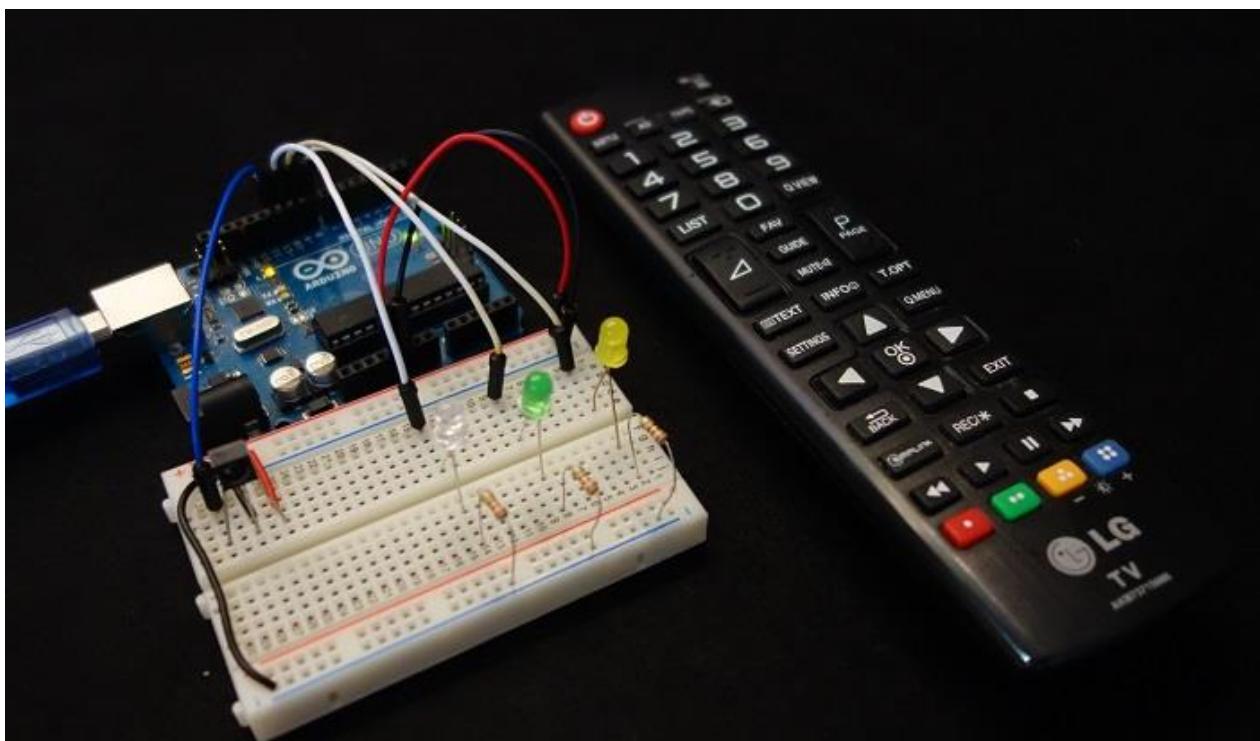
        if (state == LOW) {
            Serial.println("Motion detected!");
            state = HIGH;          // update variable state to HIGH
        }
    }
    else {
        digitalWrite(led, LOW);   // turn LED OFF
        delay(200);              // delay 200 milliseconds

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

Wrapping Up

This project shows a simple example on how to use the PIR motion sensor with the Arduino. Now, you can use the PIR motion sensor in more advanced projects. For example, you can build a [Night Security Light project](#).

Control LEDs with IR Remote Control



View Project on Random Nerd Tutorials

Click [here](#)

Watch on YouTube

Click [here](#)

View code on GitHub

Click [here](#)

Click [here](#)

Introduction

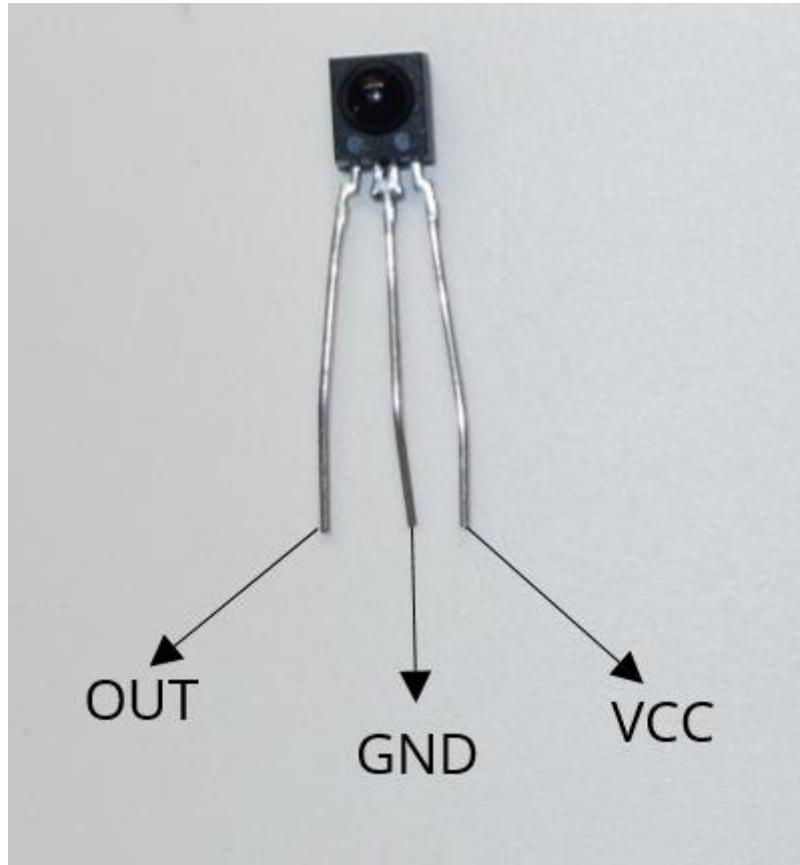
In this project you'll use an infrared (IR) receiver to control 3 LEDs with a remote control. You'll turn them on and off with the buttons of your remote control. This is a good beginner project to get you introduced to the infrared receiver.

This project is divided into two parts:

- You'll decode the IR signals transmitted by your remote control
- You'll use that info to perform a task with your Arduino (control 3 LEDs)

Infrared (IR) Receiver

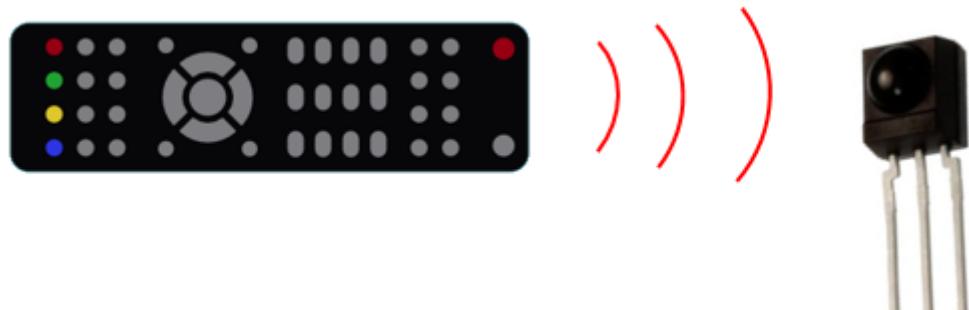
The infrared receiver is the component in the figure below. This is the TSOP4838.



As you can see, it has three pins:

- OUT (OUTPUT)
- GND (Ground)
- VCC (Voltage source)

When you press your remote control, it sends infrared modulated signals. These signals contain information that your receiver collects.

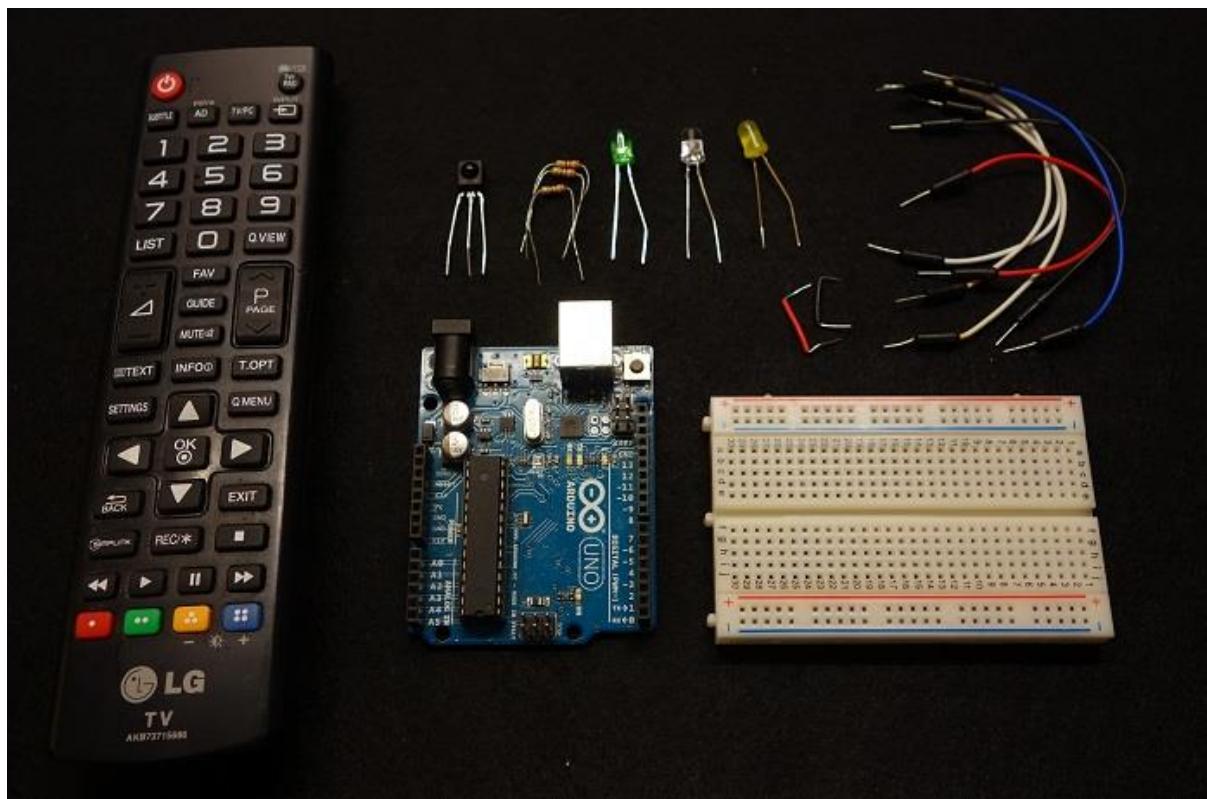


Each button sends specific information. So, we can assign that information to a specific button.

Parts Required

For this project you'll need a remote control. If you don't have any remote control at home, you need to buy one.

Any remote control will do, even if it is a very old one that you don't use. Actually, if it is an old one that you have laying around that's perfect. It's a new purpose for something old.



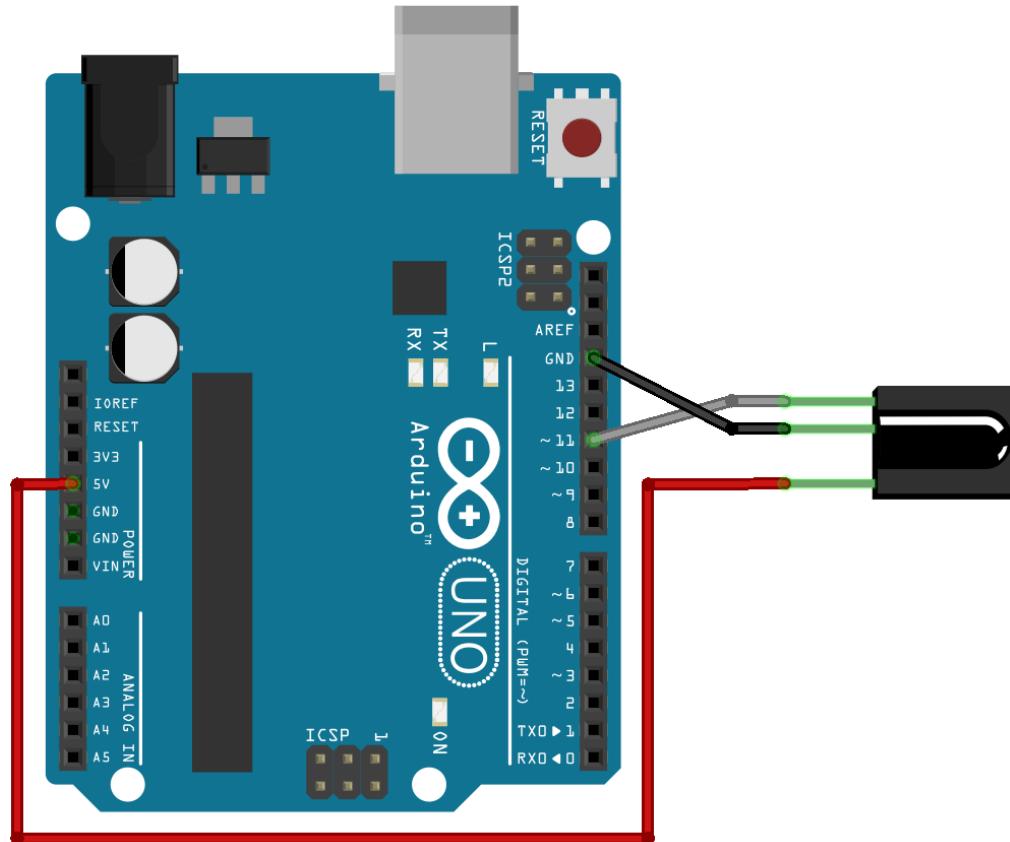
- [Arduino UNO](#) – read [Best Arduino Starter Kits](#)
- [1x Breadboard](#)
- [1x Remote control](#)
- [1x IR receiver \(I'll be using TSOP4838\)](#)
- [3x LEDs](#)
- [4x 220ohm resistors](#)
- [Jumper cables](#)

1. Decode the IR signals

In this part of the project you need to decode the IR signals associated with each button.

Schematics

Connect the IR receiver accordingly to the schematics below.



fritzing

Code

To control the IR receiver, you need to install the **IRremote Library** in the Arduino IDE.

Installing the IRremote library

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

Note: this library conflicts with the RobotIRremote library. So, in order to use it, you should move temporarily the RobotIRremote library out of the Arduino IDE libraries folder. If you don't do this, the code will not compile!

Copy the following code to your Arduino IDE, and upload it to your Arduino board. Make sure that you have the right board and COM port selected.

[View code on GitHub](#)

```
/*
 * IRremote: IRrecvDemo - demonstrates receiving IR codes with IRrecv
 * An IR detector/demodulator must be connected to the input RECV_PIN.
 * Version 0.1 July, 2009
 * Copyright 2009 Ken Shirriff
 * http://arcfn.com
 */

#include <IRremote.h>

int RECV_PIN = 11;

IRrecv irrecv(RECV_PIN);

decode_results results;

void setup()
{
    Serial.begin(9600);
    irrecv.enableIRIn(); // Start the receiver
}

void loop() {
    if (irrecv.decode(&results)) {
        Serial.println(results.value, HEX);
        irrecv.resume(); // Receive the next value
    }
    delay(100);
}
```

Open the serial monitor at a baud rate of 9600.



In this project you want to control 3 LEDs. Choose 6 buttons for the following tasks:

1. LED1 – ON
2. LED1 – OFF
3. LED2 – ON
4. LED2 – OFF
5. LED3 – ON
6. LED3 – OFF

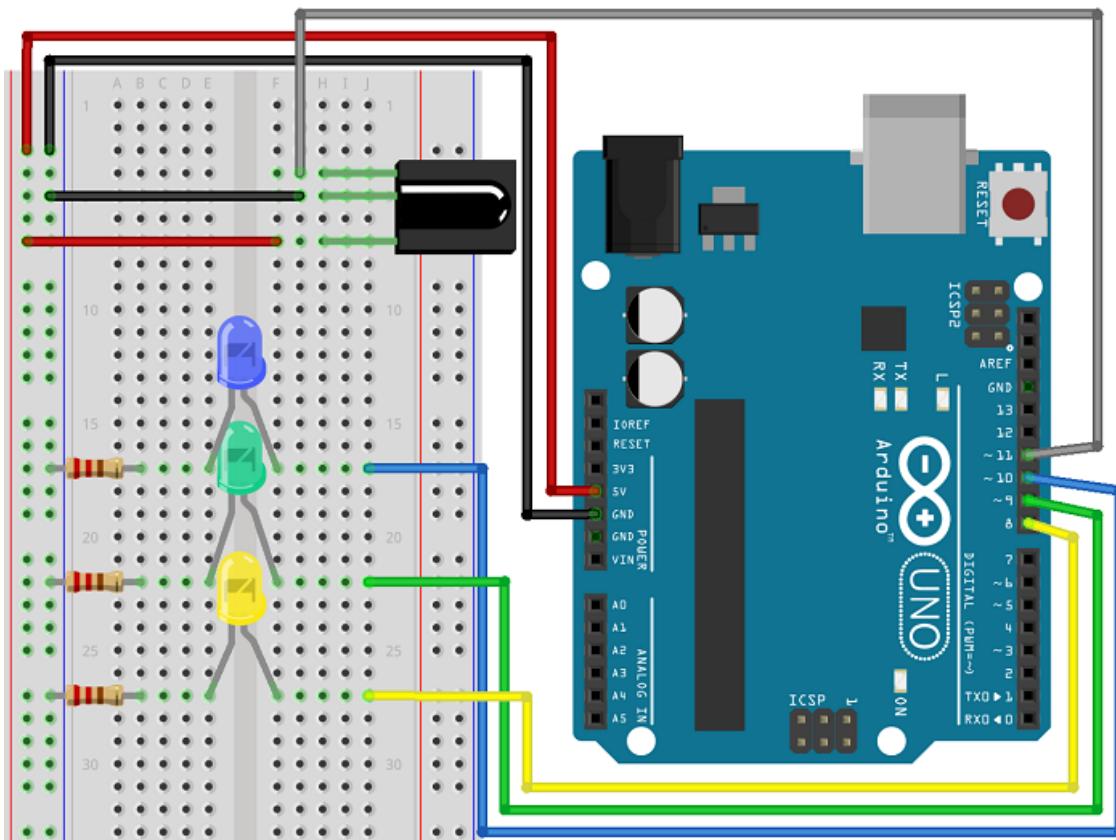
Press button number 1. You should see a code on the serial monitor. Press the same button several times to make sure you have the right code for that button. If you see something like FFFFFFF ignore it, it's trash.

Do the same for the other buttons.

Write down the code associated with each button, because you'll need that information later.

2. Building the final circuit

In this part, you'll build the circuit with three LEDs that will be controlled using your remote. Assemble all the parts by following the schematics below.

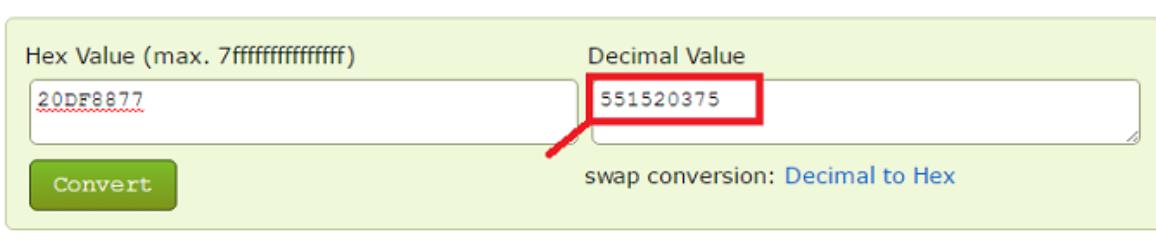


Code

Now, grab the codes you captured in the previous step. You need to convert your codes from hex to decimal.

For that, you can use the following website: www.binaryhexconverter.com/hex-to-decimal-converter

Here's a conversion example for one of my codes:



Repeat that process to all your hex values and save the decimal values. These are the ones you need to replace in the code below.

Copy the following sketch to your Arduino IDE. Write your own decimal values in the sketch provided in the *case* lines and upload it to your Arduino board. Make sure that you have the right board and COM port selected.

[View code on GitHub](#)

```
/*
 * Modified by Rui Santos, http://randomnerdtutorials.com
 * based on IRremote Library - Ken Shirriff
 */

#include <IRremote.h>

int IR_Recv = 11;    //IR Receiver Pin 3
int bluePin = 10;
int greenPin = 9;
int yellowPin = 8;

IRrecv irrecv(IR_Recv);
decode_results results;

void setup() {
    Serial.begin(9600);    //starts serial communication
    irrecv.enableIRIn();   // Starts the receiver
    pinMode(bluePin, OUTPUT);      // sets the digital pin as output
```

```

pinMode(greenPin, OUTPUT);           // sets the digital pin as output
pinMode(yellowPin, OUTPUT);         // sets the digital pin as output

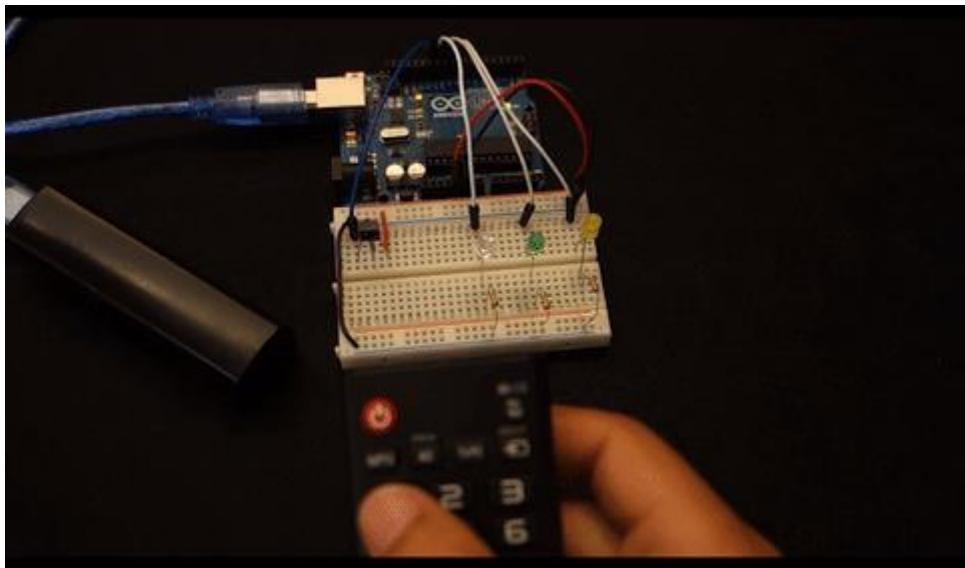
}

void loop() {
    //decodes the infrared input
    if (irrecv.decode(&results)){
        long int decCode = results.value;
        Serial.println(results.value);
        //switch case to use the selected remote control button
        switch (results.value){
            case 551520375: //when you press the 1 button
                digitalWrite(bluePin, HIGH);
                break;
            case 551495895: //when you press the 4 button
                digitalWrite(bluePin, LOW);
                break;
            case 551504055: //when you press the 2 button
                digitalWrite(greenPin, HIGH);
                break;
            case 551528535: //when you press the 5 button
                digitalWrite(greenPin, LOW);
                break;
            case 551536695: //when you press the 3 button
                digitalWrite(yellowPin, HIGH);
                break;
            case 551512215: //when you press the 6 button
                digitalWrite(yellowPin, LOW);
                break;
        }
        irrecv.resume(); // Receives the next value from the button you press
    }
    delay(10);
}

```

Demonstration

Now you can control each LED individually using the buttons of your remote control.



Wrapping up

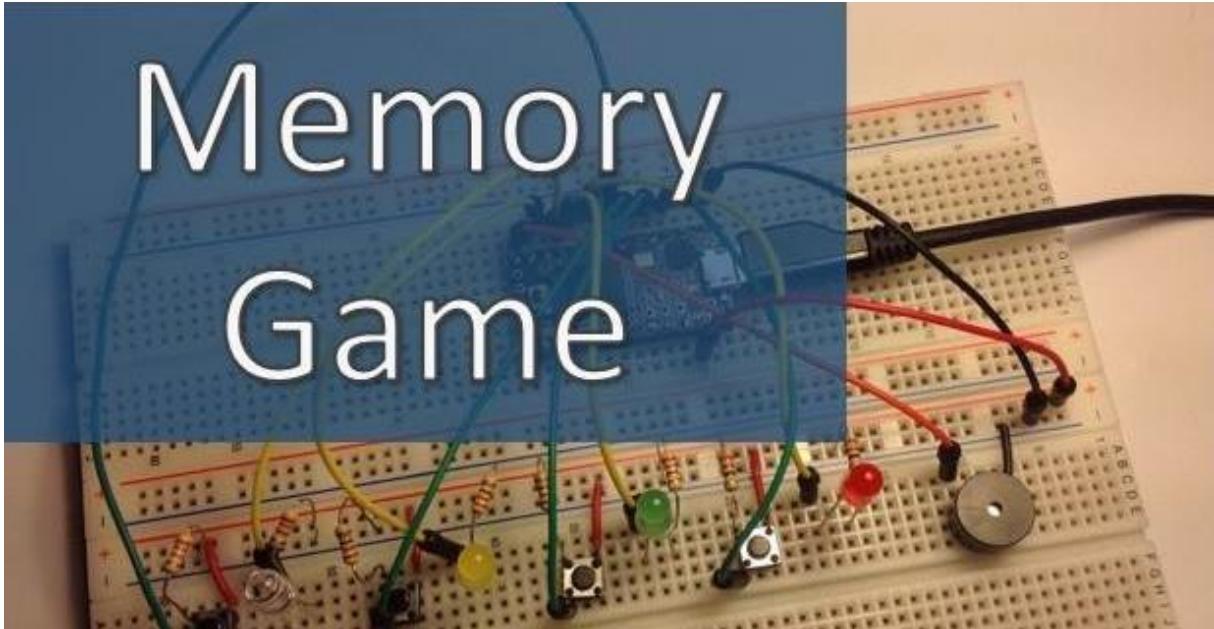
This is a great project to learn about the IR receiver. There are endless possibilities for what you can do with it.

For example, you can replace those LEDs with a relay to control your house appliances.

This can be particularly useful because some remotes have a bunch of buttons that you never use. So, why not use them to do something?

P.S. I should warn you that I've found some issues with the IRremote library. For example, I've found that this library conflicts with the analogwrite() arduino function.

Teensy/Arduino - Memory Game



View Project on Random Nerd Tutorials

Click [here](#)

Watch on YouTube

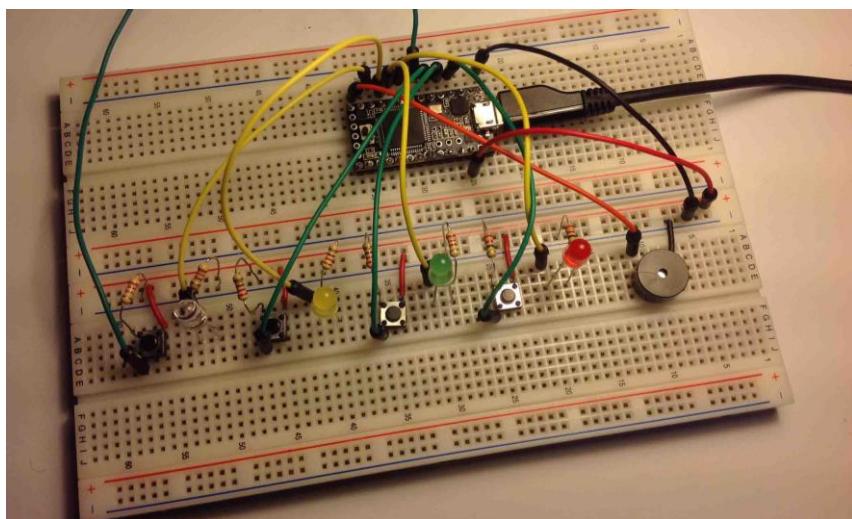
Click [here](#)

View code on GitHub

Click [here](#)

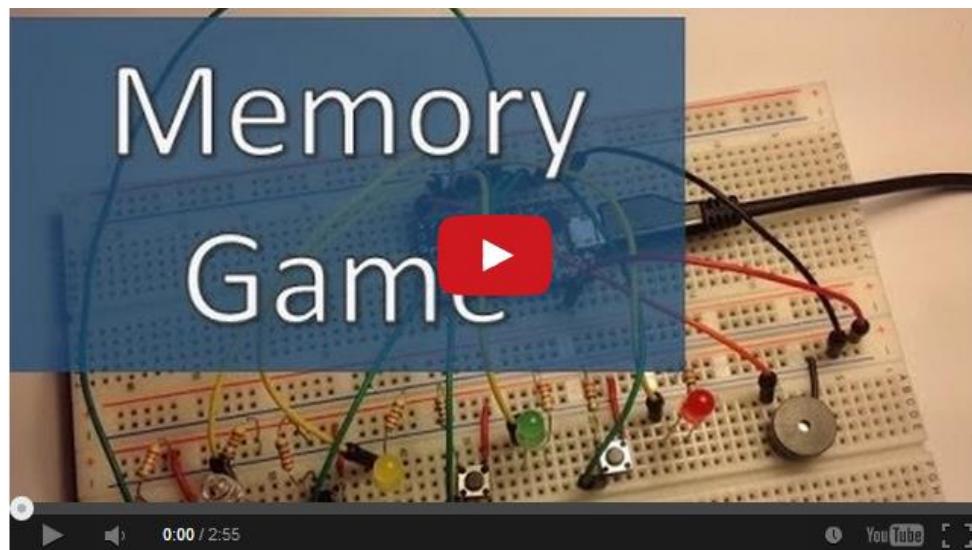
Introduction

In this project you'll create a simple game to test your memory.



I'll be using a Teensy 3.0 board. If you want to know more about this board please [click here](#) to read a Getting Started Guide. This project is also 100% compatible with the Arduino.

Watch the video below



Watch on YouTube: <http://youtu.be/cDEmH0iguMw>

Parts Required

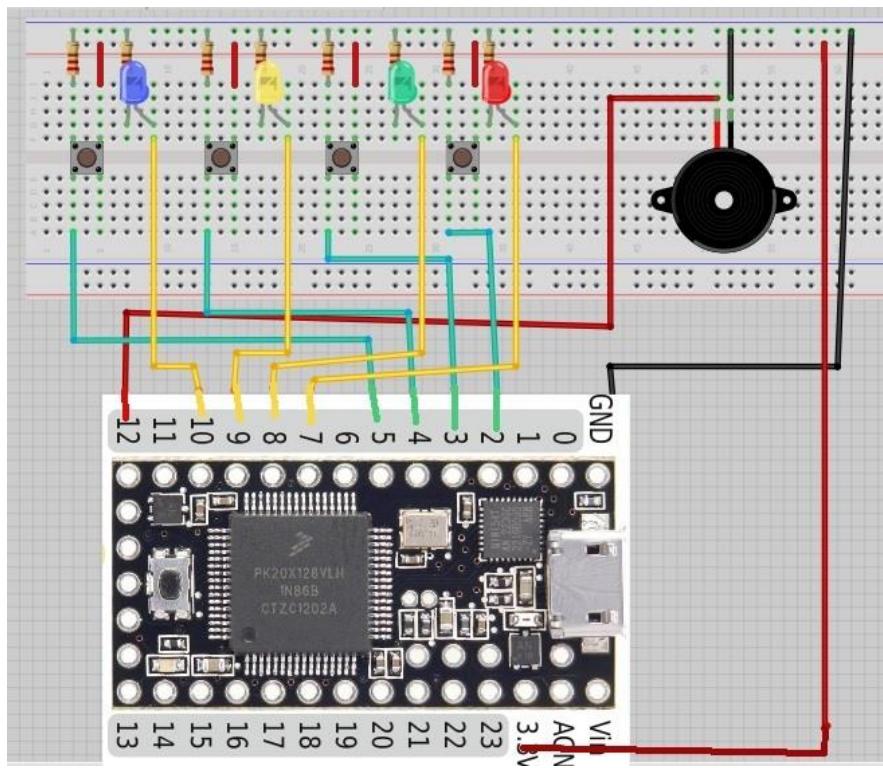
Here's a list of the parts required to build this project:

- 1x [Teensy 3.0](#)
- (or [Arduino UNO](#) – read [Best Arduino Starter Kits](#))
- [8x 220 Ohm Resistor](#)
- [4x LED's](#)
- [4x Pushbuttons](#)
- [1x Buzzer](#)
- [1x Breadboard](#)
- [Jumper Cables](#)

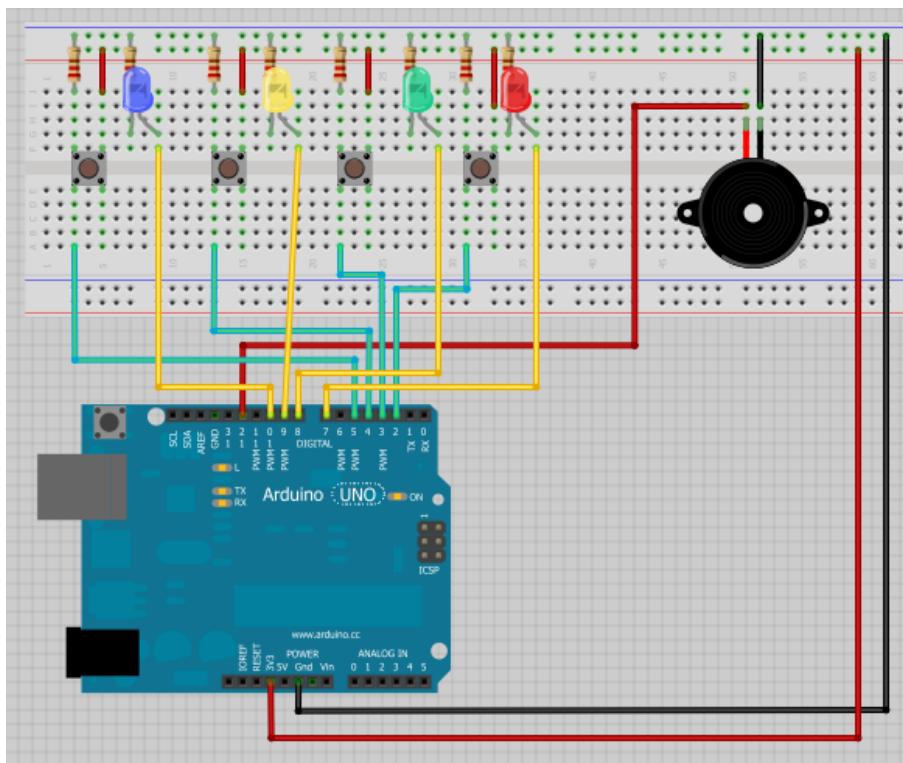
Schematics

Follow one of the next schematic diagram.

Teensy



Arduino



Code

This code works both with Teensy and Arduino.

[View code on GitHub](#)

```
/*
Memory Game with Arduino
Based on a project by Jeremy Wilson
Modified by Rui Santos
Visit: http://randomnerdtutorials.com
*/

// Constants
const int button1 = 2;          // 1st button controls Blue LED
const int button2 = 3;          // 2nd button controls Yellow LED
const int button3 = 4;          // 3rd button controls Green LED
const int button4 = 5;          // 4th button controls Red LED
const int led1 = 7;             // Blue LED
const int led2 = 8;             // Yellow LED
const int led3 = 9;             // Green LED
const int led4 = 10;            // Red LED
const int buzzer = 12;          // Buzzer Output
const int tones[] = {1915, 1700, 1519, 1432, 2700}; // tones when you press
the LED's - the last one is when you fail.

// Variables
int buttonState[] = {0,0,0,0};      // current state of the button
int lastButtonState[] = {0,0,0,0};    // previous state of the button
int buttonPushCounter[] = {0,0,0,0};

void playTone(int tone, int duration) {
    for (long i = 0; i < duration * 1000L; i += tone * 2) {
        digitalWrite(buzzer, HIGH);
        delayMicroseconds(tone);
        digitalWrite(buzzer, LOW);
        delayMicroseconds(tone);
    }
}

void setup() {
    // initialize inputs :
    randomSeed(analogRead(0));
```

```

pinMode(button1, INPUT);
pinMode(button2, INPUT);
pinMode(button3, INPUT);
pinMode(button4, INPUT);
// initialize outputs:
pinMode(led1, OUTPUT);
pinMode(led2, OUTPUT);
pinMode(led3, OUTPUT);
pinMode(led4, OUTPUT);
pinMode(buzzer, OUTPUT);
// initialize serial communication for debugging:
//Serial.begin(9600);
}

int game_on = 0;
int wait = 0;
int currentlevel = 1; // This is the level (also the number of button presses to pass to next level)
long rand_num = 0; //initialize long variable for random number from 0-100.
int rando = 0; //initialize random integer for loopgame_on. Will be from 1-4 later.
int butwait = 500; //amount of time to wait for next button input (ghetto debounce)
int ledtime = 500; //amount of time each LED flashes for when button is pressed
int n_levels = 10; //number of levels until the game is won
int pinandtone = 0; //This integer is used when the sequence is displayed
int right = 0; //This variable must be 1 in order to go to the next level
int speedfactor = 5; //This is the final speed of the lights and sounds for the last level. This increases as more games are won
int leddelay = 200; //Initializing time for LED. This will decrease as the level increases

void loop() {

int n_array[n_levels];
int u_array[n_levels];

int i;
//clears arrays both "n_array" and "u_array" and starts a new game
if (game_on == 0){
for(i=0; i<n_levels; i=i+1){
    n_array[i]=0;
    u_array[i]=0;
    rand num = random(1,200);
    if (rand_num <= 50)
}
}
}

```

```

        rando=0;
    else if (rand_num>50 && rand_num<=100)
        rando=1;
    else if (rand_num>100 && rand_num<=150)
        rando=2;
    else if (rand_num<=200)
        rando=3;
    //saves a random number in our n_array
    n_array[i]=rando;
}
game_on = 1;

}

//shows the user the current sequence
if (wait == 0){
    delay (200);
i = 0;
for (i = 0; i < currentlevel; i= i + 1){
    leddelay = ledtime/(1+(speedfactor/n_levels)*(currentlevel - 1));
    pinandtone = n_array[i];
    digitalWrite(pinandtone+7, HIGH);
    playTone(tones[pinandtone], leddelay);
    digitalWrite(pinandtone+7, LOW);
    delay(100/speedfactor);
}
wait = 1;
}
i = 0;
int buttonchange = 0;
int j = 0; // This is the current position in the sequence
while (j < currentlevel){
    while (buttonchange == 0){
        for (i = 0; i < 4; i = i + 1){
            buttonState[i] = digitalRead(i+2);
            buttonchange = buttonchange + buttonState[i];
        }
    }
    for (i = 0; i < 4; i = i + 1){
        if (buttonState[i] == HIGH) {
            digitalWrite(i+7, HIGH);
            playTone(tones[i], ledtime);
        }
    }
}

```

```

        digitalWrite(i+7, LOW);
        wait = 0;
        u_array[j]=i;
        buttonState[i] = LOW;
        buttonchange = 0;
    }
}

if (u_array[j] == n_array[j]){
    j++;
    right = 1;
}
else{
    right = 0;
    i = 4;
    j = currentlevel;
    wait = 0;
}

if (right == 0){
    delay(300);
    i = 0;
    game_on = 0;
    currentlevel = 1;
    for (i = 0; i < 4; i = i + 1){
        digitalWrite(i+7, HIGH);
    }
    playTone(tones[4], ledtime);
    for (i = 0; i < 4; i = i + 1){
        digitalWrite(i+7, LOW);
    }
    delay (200);
    for (i = 0; i < 4; i = i + 1){
        digitalWrite(i+7, HIGH);
    }
    playTone(tones[4], ledtime);
    for (i = 0; i < 4; i = i + 1){
        digitalWrite(i+7, LOW);
    }

    delay(500);
    game_on = 0;
}

```

```

}

//if you insert the right sequence it levels up
if (right == 1){
    currentlevel++;
    wait = 0;
}
//if you finish the game
if (currentlevel == n_levels){
    delay(500);
    // The following is the victory sound:
    int notes[] = {2, 2, 2, 2, 0, 1, 2, 1, 2};
    int note = 0;
    int tempo[] = {200, 200, 200, 400, 400, 400, 200, 200, 600};
    int breaks[] = {100, 100, 100, 200, 200, 200, 300, 100, 200};
    for (i = 0; i < 9; i = i + 1){
        note = notes[i];
        digitalWrite(note+7, HIGH);
        playTone(tones[note], tempo[i]);
        digitalWrite(note+7, LOW);
        delay(breaks[i]);
    }
    //sets game_on to 0, so it restarts a new game
    game_on = 0;
    currentlevel = 1;
    n_levels = n_levels + 2;
    speedfactor = speedfactor + 1;
}
}

```

Guide for MQ-2 Gas/Smoke Sensor with Arduino



View Project on Random Nerd Tutorials	Click here
Watch on YouTube	Click here
View code on GitHub	Click here

This guide shows how to build a smoke detector that beeps when it detects flammable gas or smoke.

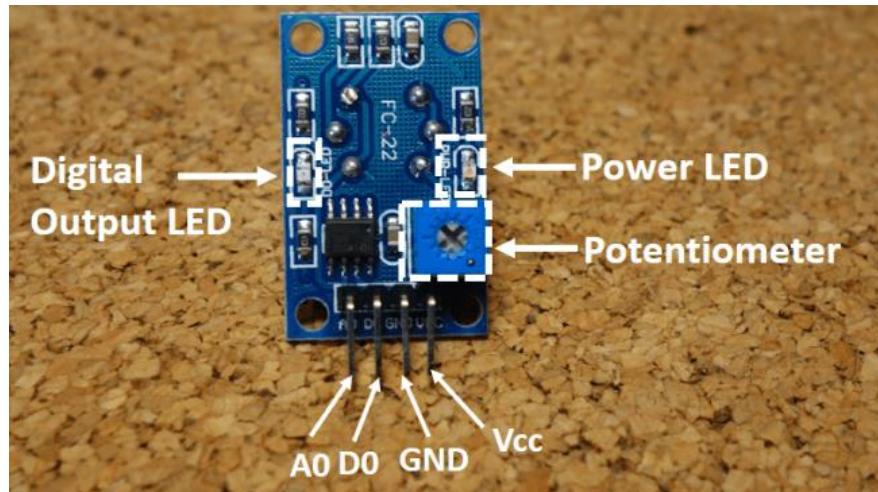
The MQ-2 Gas Sensor

The MQ-2 smoke sensor is sensitive to smoke and to the following flammable gases:

- LPG
- Butane
- Propane
- Methane
- Alcohol
- Hydrogen

The resistance of 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.

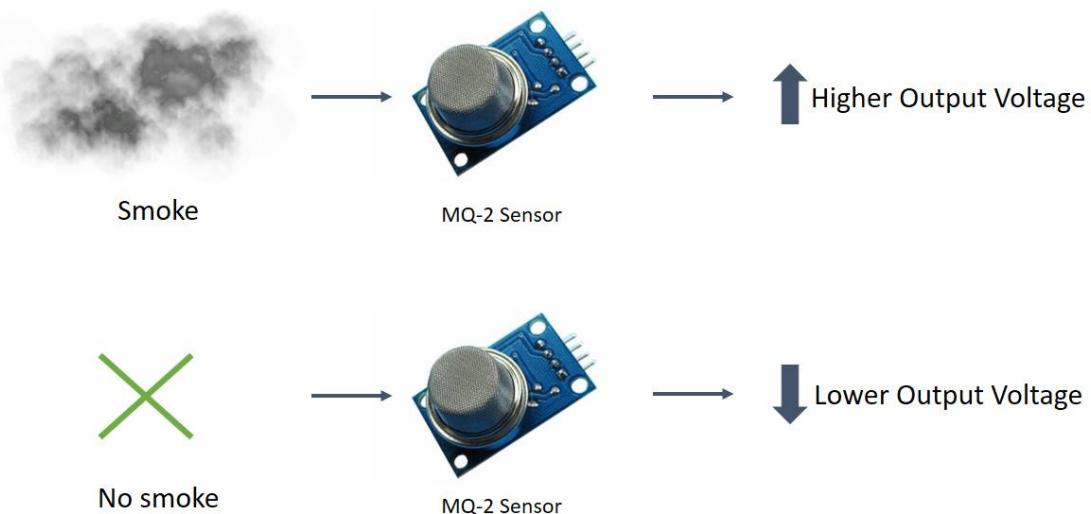


How does it work?

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 as shown:

- 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.

Pin Wiring

The MQ-2 sensor has 4 pins.

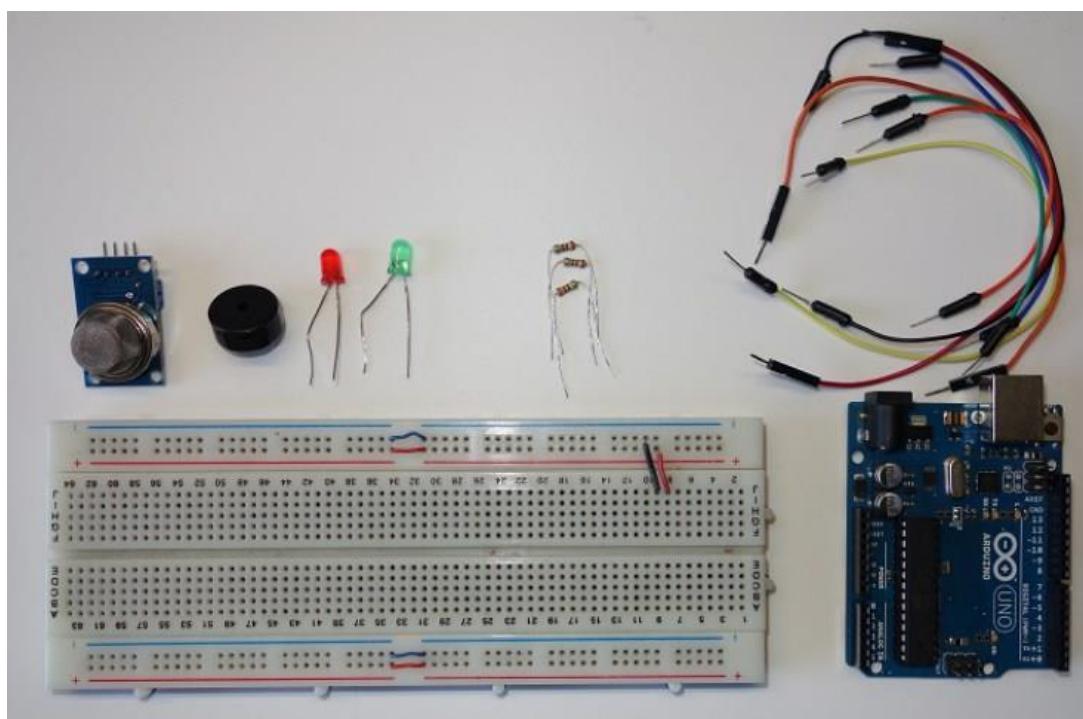
Pin	Wiring to Arduino Uno
A0	Analog pins
D0	Digital pins
GND	GND
VCC	5V

Example: Gas Sensor with Arduino

In this example, you will read the sensor analog output voltage and 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.

Parts Required

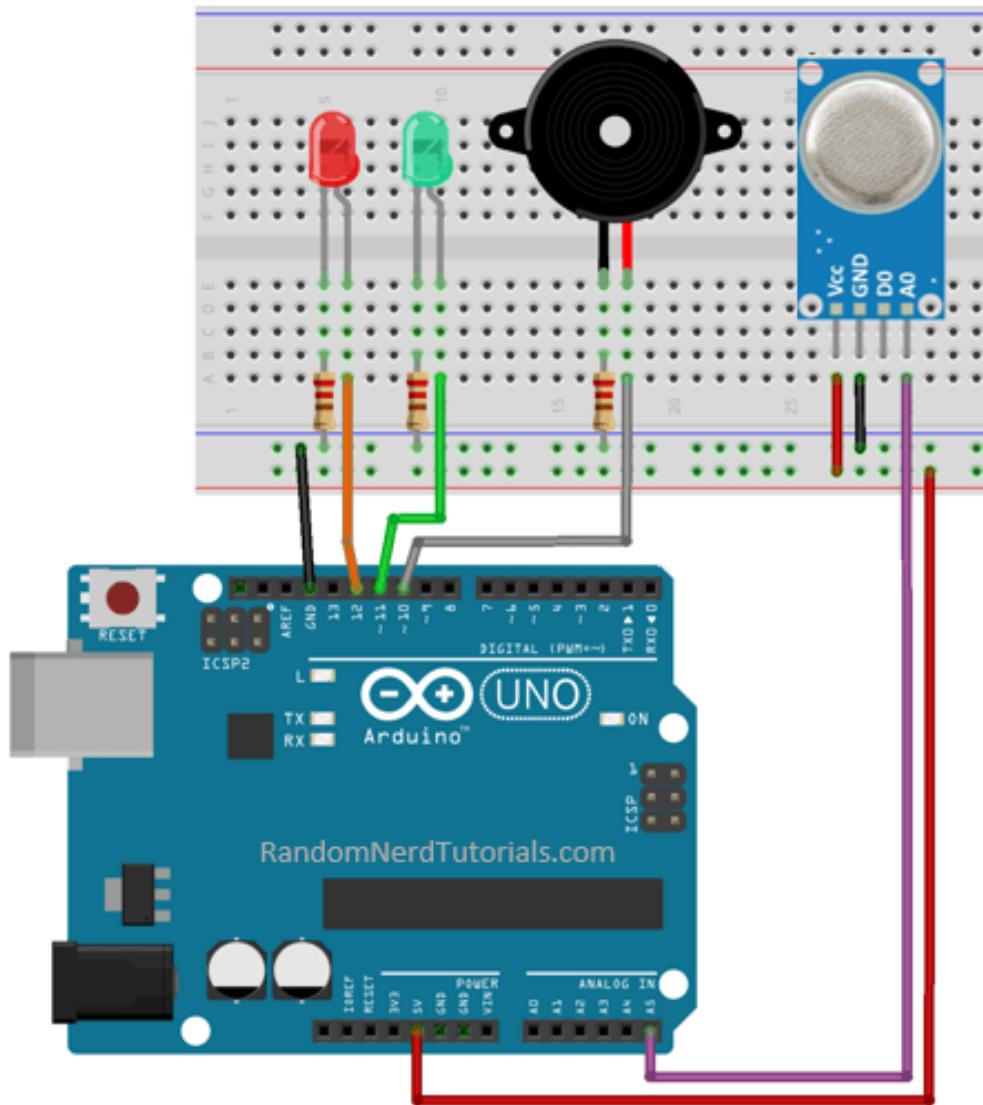
For this example, you'll need the following parts:



- [1 x MQ-2 gas sensor](#)
- [Arduino UNO](#) – read [Best Arduino Starter Kits](#)
- [1x Breadboard](#)
- [1 x red LED](#)
- [1 x green LED](#)
- [1 x buzzer](#)
- [3 x 220 \$\Omega\$ resistor](#)
- [Jumper wires](#)

Schematics

Follow these schematic diagram to complete the project:



Code

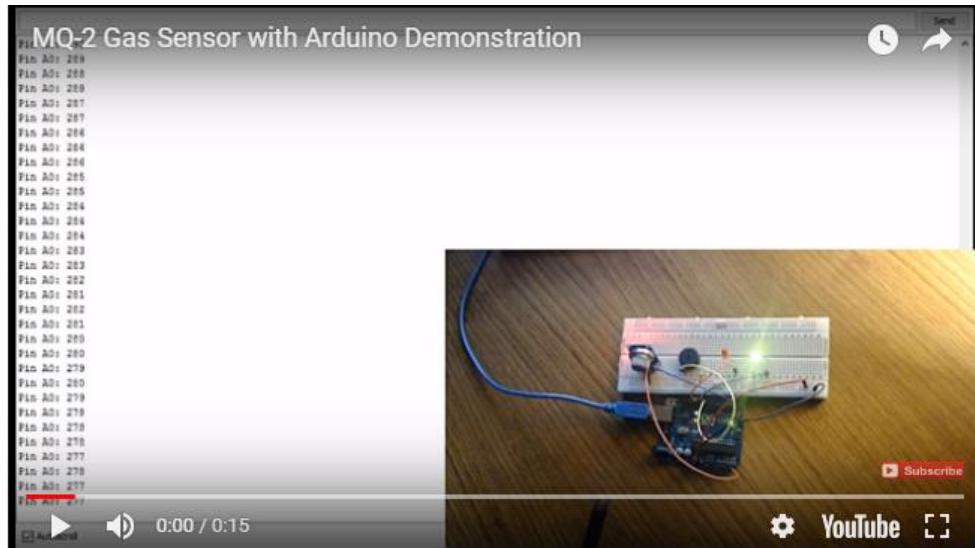
Upload the following sketch to your Arduino board (feel free to adjust the variable **sensorThres** variable with a different threshold value):

[View code on GitHub](#)

```
*****  
  
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);  
}
```

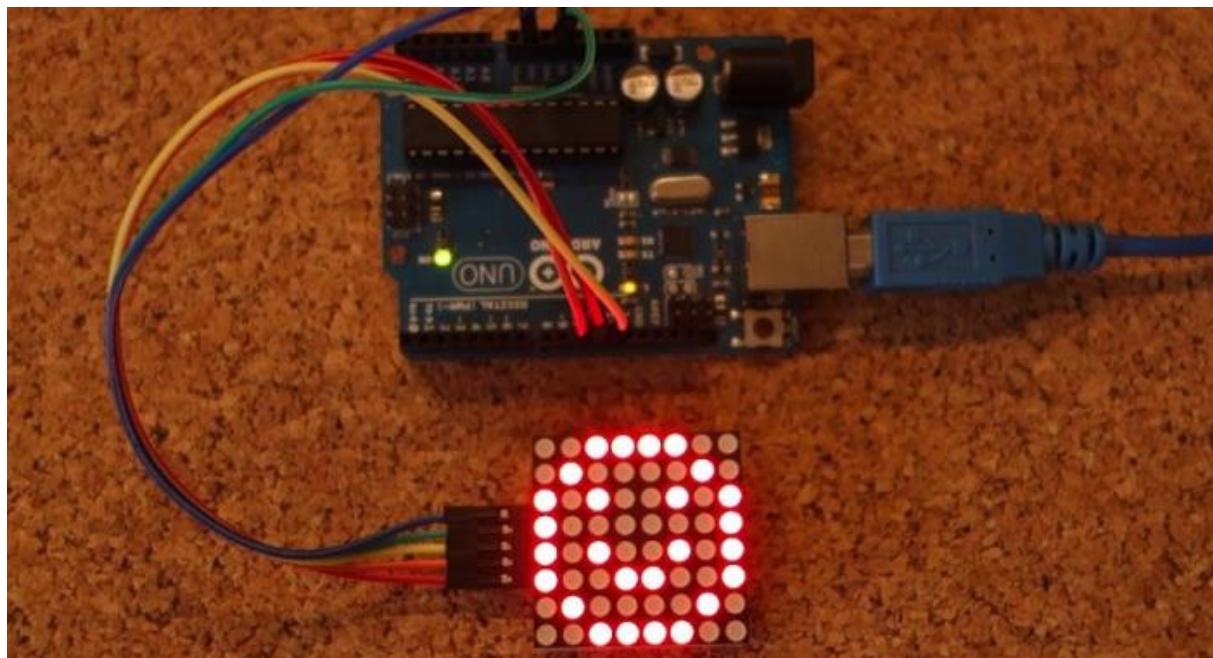
Video demonstration

Watch this quick video demonstration to see this project in action:



Watch on YouTube: https://youtu.be/_mEUszHqWDg

Guide for 8x8 Dot Matrix MAX7219 + Pong Game



View Project on Random Nerd Tutorials

Click [here](#)

View code on GitHub

Click [here](#)

Click [here](#)

Introduction

The dot matrix that we're going to use in this project is a 8x8 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 using just 3 digital pins of the Arduino board.

The best option is to buy the dot matrix with the MAX7219 chip as a module, it will simplify the wiring. You can check the [dot matrix at Maker Advisor](#) and find the best price.



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

Parts Required

For this guide you'll need the following parts:

- [1x 8x8 Dot Matrix with MAX7219](#)
- [Arduino UNO](#) – read [Best Arduino Starter Kits](#)
- [1x 1k ohm Potentiometer](#)
- [Jumper wires](#)

Pin Wiring

You 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

How to control the dot matrix with Arduino

To make 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)

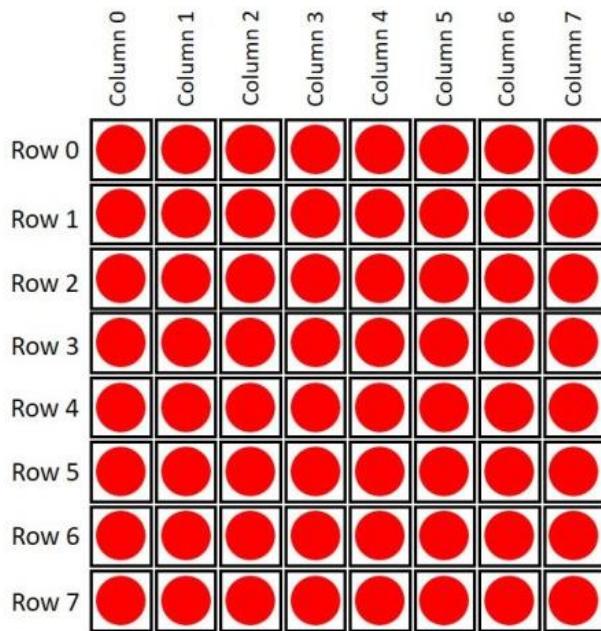
- addr is the address of your matrix, for example, if you have just 1 matrix, the int addr will be zero.
- row is the row where the led is located
- col is the column where the led is located
- state
 - It's true or 1 if you want to turn the led on
 - It's false or 0 if you want to switch it off

setRow(addr, row, value)

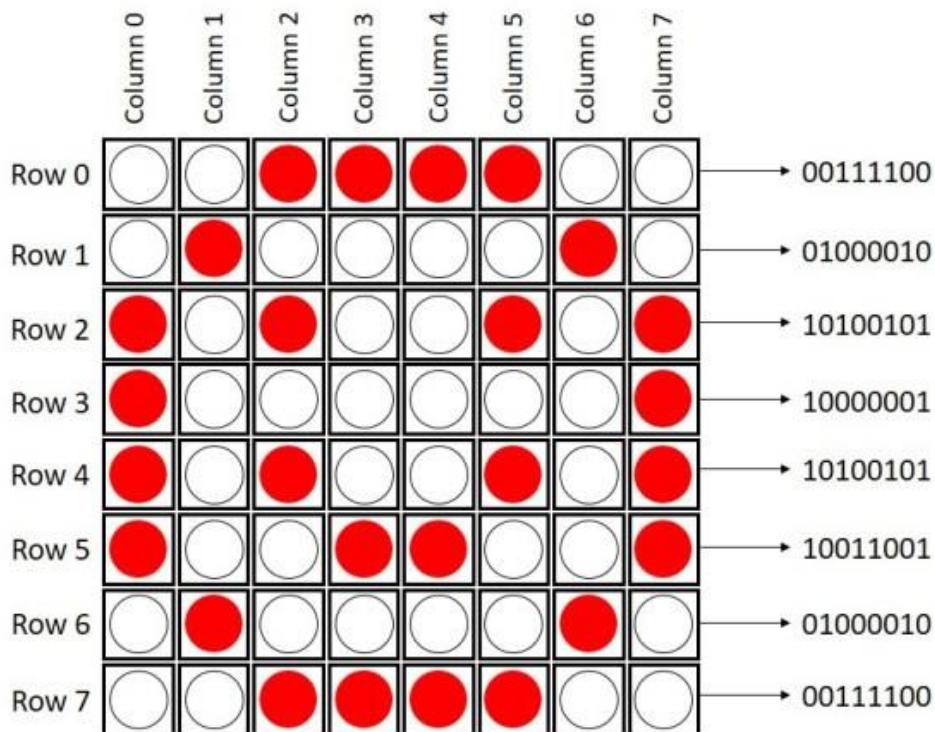
setCol(addr, column, value)

Index

As previously stated, this matrix has 8 columns and 8 rows. These are 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 take note of the LEDs that are on or off on the matrix. For example, if you want to display a happy face, here's what you need to do:



Code

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:

[View code on GitHub](#)

```
/*
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,B00111100};

// neutral face
byte nf[8]={B00111100,
B01000010,B10100101,B10000001,B1011101,B10000001,B01000010,B00111100};

// sad face
byte sf[8]=
{B00111100,B01000010,B10100101,B10000001,B10011001,B10100101,B01000010,B00111100};

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);

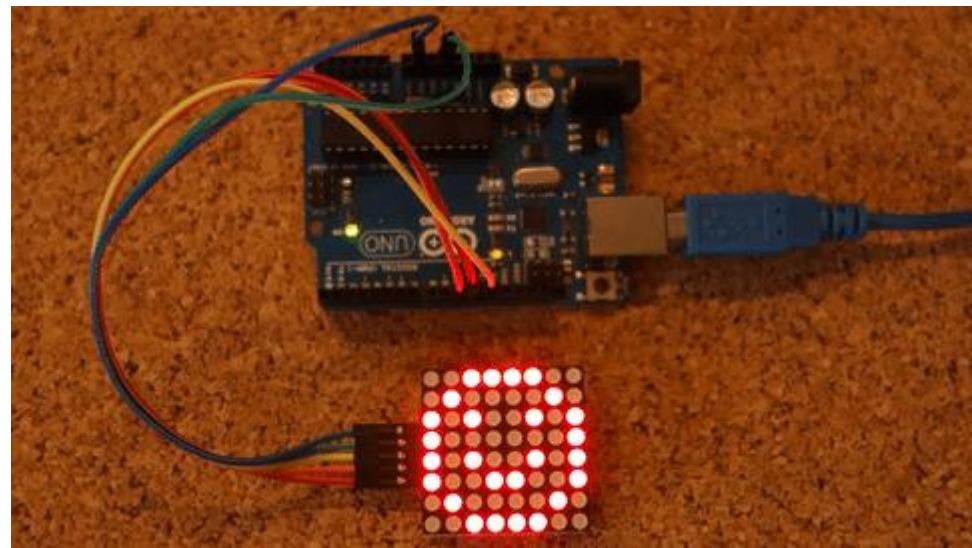
    // Display neutral face
    lc.setRow(0,0,nf[0]);
    lc.setRow(0,1,nf[1]);
    lc.setRow(0,2,nf[2]);
    lc.setRow(0,3,nf[3]);
    lc.setRow(0,4,nf[4]);
    lc.setRow(0,5,nf[5]);
    lc.setRow(0,6,nf[6]);
    lc.setRow(0,7,nf[7]);
    delay(delaytime);

    // 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();
}

```

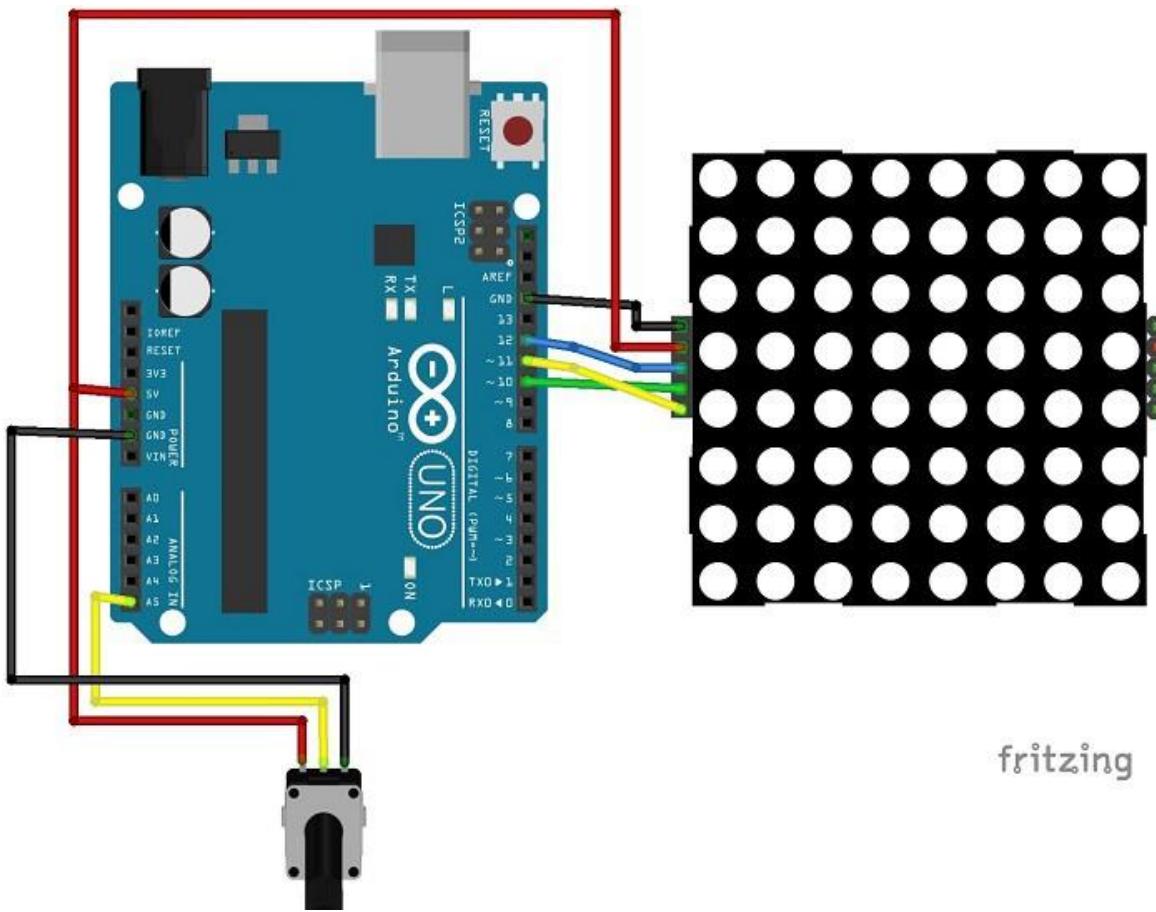
In the end, you'll have something like this:



Pong Game

The pong game that you're about to try was created by [Alessandro Pasotti](#).

For the pong game, you just need to add a 1k ohm potentiometer to the previous schematic. Assemble the new circuit as shown in the schematic below:



Code

Upload the following code to your Arduino board:

[View code on GitHub](#)

```
/*
 *      Play pong on an 8x8 matrix - project from itopen.it
 */

#include "LedControl.h"
#include "Timer.h"

#define POTPIN A5 // Potentiometer
#define PADSIZE 3
#define BALL_DELAY 200
#define GAME_DELAY 10
#define BOUNCE_VERTICAL 1
#define BOUNCE_HORIZONTAL -1
#define NEW_GAME_ANIMATION_SPEED 50
#define HIT_NONE 0
#define HIT_CENTER 1
#define HIT_LEFT 2
#define HIT_RIGHT 3

//#define DEBUG 1

byte sad[] = {
B00000000,
B01000100,
B00010000,
B00010000,
B00010000,
B00000000,
B00111000,
B01000100,
B00000000
};

byte smile[] = {
B00000000,
B01000100,
B00010000,
B00010000,
```

```

B00010000,
B01000100,
B00111000,
B00000000
};

Timer timer;

LedControl lc = LedControl(12,11,10,1);

byte direction; // Wind rose, 0 is north
int xball;
int yball;
int yball prev;
byte xpad;
int ball_timer;

void setSprite(byte *sprite){
    for(int r = 0; r < 8; r++){
        lc.setRow(0, r, sprite[r]);
    }
}

void newGame() {
    lc.clearDisplay(0);
    // initial position
    xball = random(1, 7);
    yball = 1;
    direction = random(3, 6); // Go south
    for(int r = 0; r < 8; r++){
        for(int c = 0; c < 8; c++){
            lc.setLed(0, r, c, HIGH);
            delay(NEW_GAME_ANIMATION_SPEED);
        }
    }
    setSprite(smile);
    delay(1500);
    lc.clearDisplay(0);
}

void setPad() {
    xpad = map(analogRead(POTPIN), 0, 1020, 8 - PADSIZE, 0);
}

```

```

void debug(const char* desc) {
#ifdef DEBUG
    Serial.print(desc);
    Serial.print(" XY: ");
    Serial.print(xball);
    Serial.print(", ");
    Serial.print(yball);
    Serial.print(" XPAD: ");
    Serial.print(xpad);
    Serial.print(" DIR: ");
    Serial.println(direction);
#endif
}

int checkBounce() {
    if(!xball || !yball || xball == 7 || yball == 6) {
        int bounce = (yball == 0 || yball == 6) ? BOUNCE_HORIZONTAL : BOUNCE_VERTICAL;
#ifdef DEBUG
        debug(bounce == BOUNCE_HORIZONTAL ? "HORIZONTAL" : "VERTICAL");
#endif
        return bounce;
    }
    return 0;
}

int getHit() {
    if(yball != 6 || xball < xpad || xball > xpad + PADSIZE) {
        return HIT_NONE;
    }
    if(xball == xpad + PADSIZE / 2) {
        return HIT_CENTER;
    }
    return xball < xpad + PADSIZE / 2 ? HIT_LEFT : HIT_RIGHT;
}

bool checkLoose() {
    return yball == 6 && getHit() == HIT_NONE;
}

void moveBall() {
    debug("MOVE");
}

```

```

int bounce = checkBounce();
if(bounce) {
    switch(direction) {
        case 0:
            direction = 4;
        break;
        case 1:
            direction = (bounce == BOUNCE_VERTICAL) ? 7 : 3;
        break;
        case 2:
            direction = 6;
        break;
        case 6:
            direction = 2;
        break;
        case 7:
            direction = (bounce == BOUNCE_VERTICAL) ? 1 : 5;
        break;
        case 5:
            direction = (bounce == BOUNCE_VERTICAL) ? 3 : 7;
        break;
        case 3:
            direction = (bounce == BOUNCE_VERTICAL) ? 5 : 1;
        break;
        case 4:
            direction = 0;
        break;
    }
    debug("->");
}

// Check hit: modify direction is left or right
switch(getHit()) {
    case HIT_LEFT:
        if(direction == 0) {
            direction = 7;
        } else if (direction == 1) {
            direction = 0;
        }
    break;
    case HIT_RIGHT:
        if(direction == 0) {
            direction = 1;
        }
}

```

```

        } else if(direction == 7) {
            direction = 0;
        }
        break;
    }

// Check orthogonal directions and borders ...
if((direction == 0 && xball == 0) || (direction == 4 && xball == 7)) {
    direction++;
}
if(direction == 0 && xball == 7) {
    direction = 7;
}
if(direction == 4 && xball == 0) {
    direction = 3;
}
if(direction == 2 && yball == 0) {
    direction = 3;
}
if(direction == 2 && yball == 6) {
    direction = 1;
}
if(direction == 6 && yball == 0) {
    direction = 5;
}
if(direction == 6 && yball == 6) {
    direction = 7;
}

// "Corner" case
if(xball == 0 && yball == 0) {
    direction = 3;
}
if(xball == 0 && yball == 6) {
    direction = 1;
}
if(xball == 7 && yball == 6) {
    direction = 7;
}
if(xball == 7 && yball == 0) {
    direction = 5;
}

```

```

yball_prev = yball;
if(2 < direction && direction < 6) {
    yball++;
} else if(direction != 6 && direction != 2) {
    yball--;
}
if(0 < direction && direction < 4) {
    xball++;
} else if(direction != 0 && direction != 4) {
    xball--;
}
xball = max(0, min(7, xball));
yball = max(0, min(6, yball));
debug("AFTER MOVE");
}

void gameOver() {
    setSprite(sad);
    delay(1500);
    lc.clearDisplay(0);
}

void drawGame() {
    if(yball_prev != yball){
        lc.setRow(0, yball_prev, 0);
    }
    lc.setRow(0, yball, byte(1 << (xball)));
    byte padmap = byte(0xFF >> (8 - PADSIZE) << xpad) ;
#ifdef DEBUG
    //Serial.println(padmap, BIN);
#endif
    lc.setRow(0, 7, padmap);
}

void setup() {
    // The MAX72XX is in power-saving mode on startup,
    // we have to do a wakeup call
    pinMode(POTPIN, INPUT);

    lc.shutdown(0, false);
    // Set the brightness to a medium values
    lc.setIntensity(0, 8);
    // and clear the display
}

```

```

lc.clearDisplay(0);
randomSeed(analogRead(0));

#ifndef DEBUG
  Serial.begin(9600);
  Serial.println("Pong");
#endif

newGame();
ball_timer = timer.every(BALL_DELAY, moveBall);

}

void loop() {
  timer.update();
  // Move pad
  setPad();

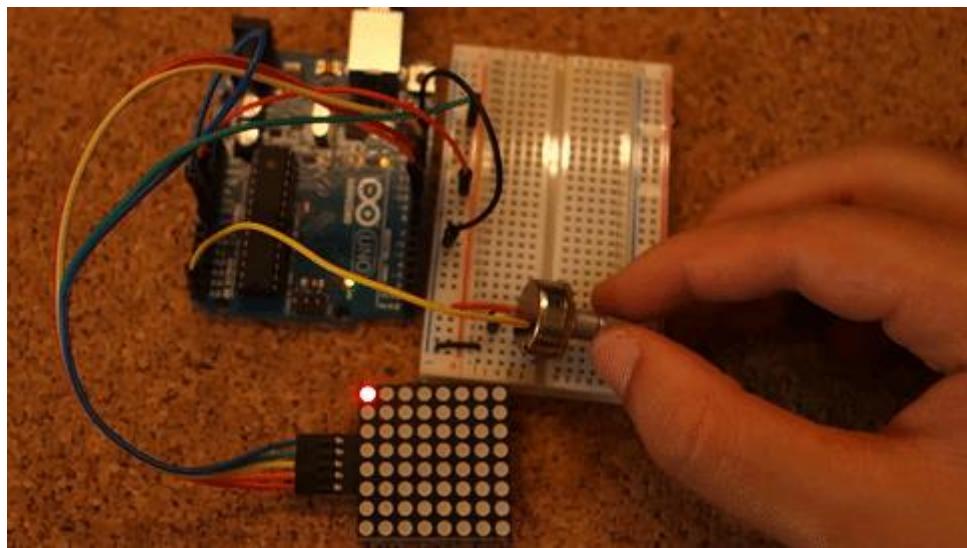
#ifndef DEBUG
  Serial.println(xpad);
#endif

  // Update screen
  drawGame();
  if(checkLoose()) {
    debug("LOSE");
    gameOver();
    newGame();
  }
  delay(GAME_DELAY);
}

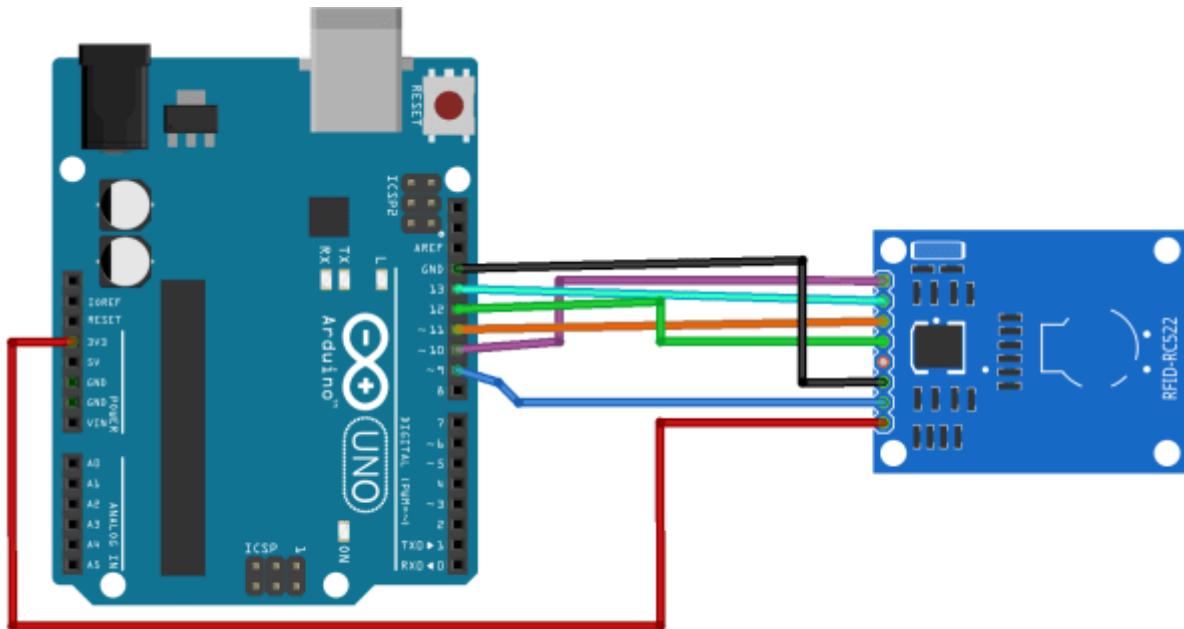
```

Demonstration

Here's the final demonstration of me playing the pong game. Have fun!



Security Access using MFRC522 RFID Reader with Arduino



fritzing

View Project on Random Nerd Tutorials

Click [here](#)

Watch on YouTube

Click [here](#)

View code on GitHub

Click [here](#)

This project shows a simple example on how to use the MFRC522 RFID reader. I'll do a quick overview of the specifications and demonstrate a project example using an Arduino.

Description

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 get in. 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 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
- Price: approximately 3\$ ([check best price on Maker Advisor](#))
- Frequency: 13.56MHz

Library Download

Here's the library you need for this project:

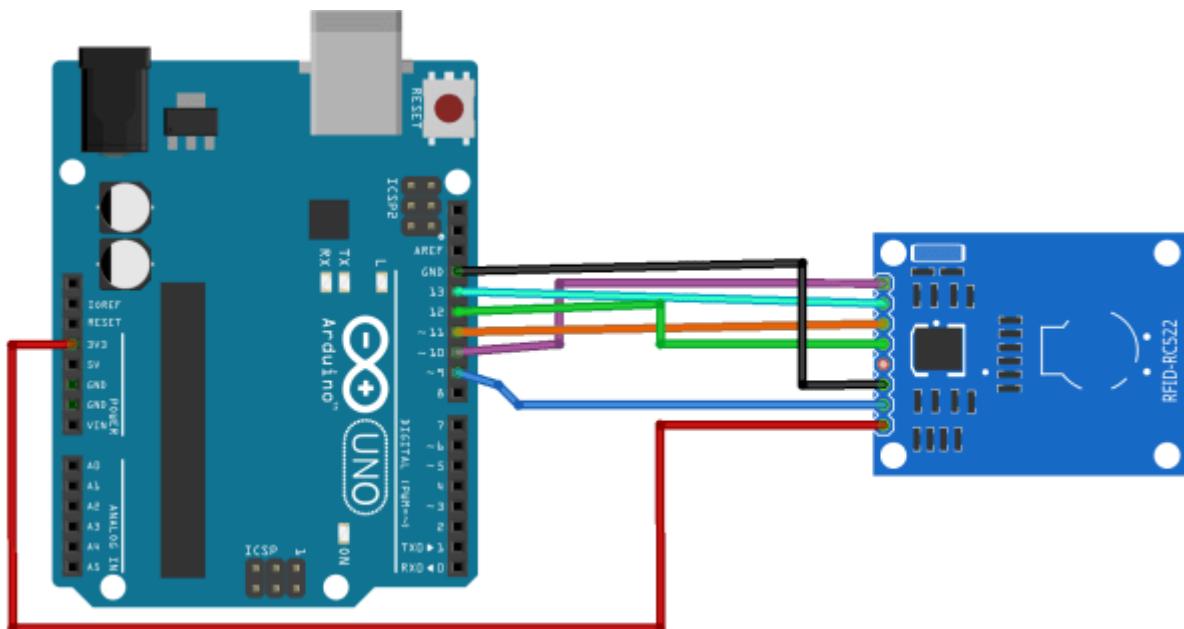
1. [Click here to download the RFID library](#). You should have a .zip folder in your Downloads
2. Unzip the .zip folder and you should get **RFID-master** folder
3. Rename your folder from **RFID-master** to **RFID**
4. Move the **RFID** folder to your Arduino IDE installation libraries folder
5. Finally,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

Pin Wiring

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

Schematic

Wire the RFID reader to the Arduino as shown in the following schematic diagram:

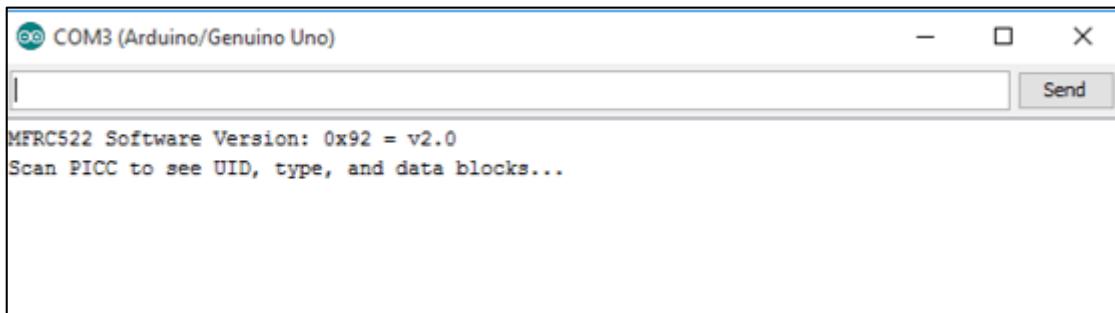


fritzing

Reading Data from a RFID Tag

After having the circuit ready, go to **File > Examples > MFRC522 > DumpInfo** and upload the code. This code is 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.

A screenshot of the Arduino Serial Monitor window titled "COM3 (Arduino/Genuino Uno)". The text area displays the following detailed RFID tag data:
MFRC522 Software Version: 0x92 = v2.0
Scan PICC to see UID, type, and data blocks...
Card UID: BD 31 15 2B
PICC type: MIFARE 1KB
Sector Block 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 AccessBits
15 63 00 00 00 00 00 00 FF 07 80 69 FF FF FF FF FF FF FF [0 0 1]
62 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 [0 0 0]
61 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 [0 0 0]
60 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 [0 0 0]
14 59 00 00 00 00 00 00 FF 07 80 69 FF FF FF FF FF FF FF [0 0 1]
58 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 [0 0 0]
57 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 [0 0 0]
56 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 [0 0 0]
13 55 00 00 00 00 00 00 FF 07 80 69 FF FF FF FF FF FF FF [0 0 1]
54 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 [0 0 0]
53 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 [0 0 0]
52 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 [0 0 0]
12 51 00 00 00 00 00 00 FF 07 80 69 FF FF FF FF FF FF FF [0 0 1]
50 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 [0 0 0]
49 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 [0 0 0]
48 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 [0 0 0]
11 47 00 00 00 00 00 00 FF 07 80 69 FF FF FF FF FF FF FF [0 0 1]
46 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 [0 0 0]
45 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 [0 0 0]
44 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 [0 0 0]
10 43 00 00 00 00 00 00 FF 07 80 69 FF FF FF FF FF FF FF [0 0 1]
42 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 [0 0 0]
41 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 [0 0 0]
40 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 [0 0 0]
9 39 00 00 00 00 00 00 FF 07 80 69 FF FF FF FF FF FF FF [0 0 1]
38 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 [0 0 0]

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.

Code

Upload the following code to your Arduino board:

[View code on GitHub](#)

```
/*
 *
 * 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);
}
}

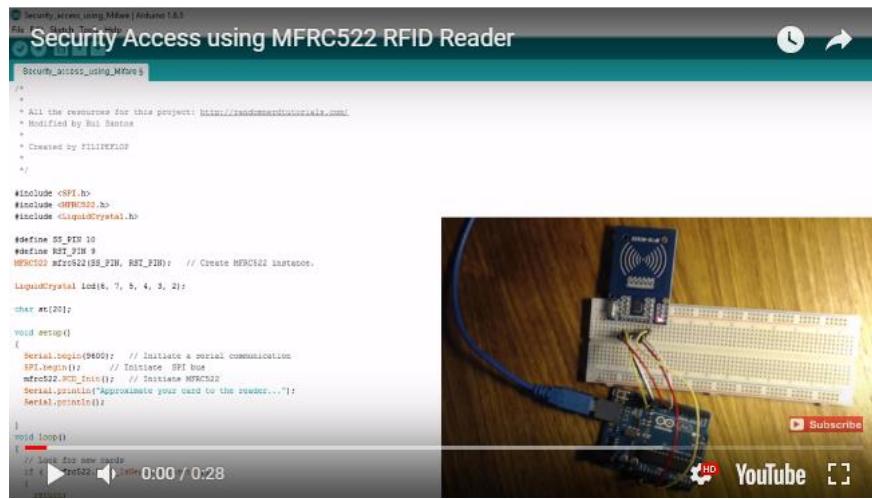
```

In code you need to change the following line with the UID card you've found previously.

```
if (content.substring(1) == "REPLACE WITH YOUR UID")
```

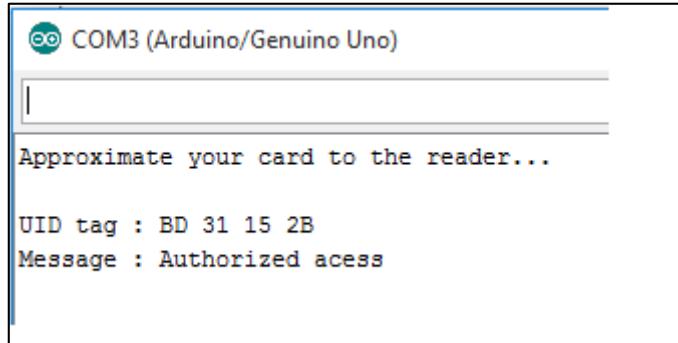
Demonstration

Open the Serial Monitor at a baud rate of 115200.

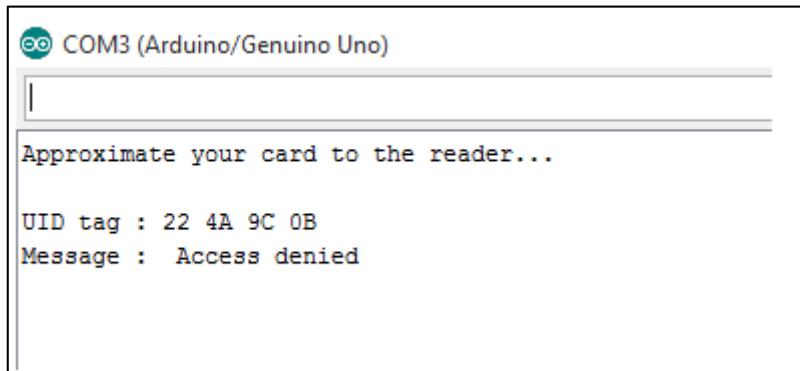


Watch on YouTube: <https://youtu.be/mpLzcBDBI1U>

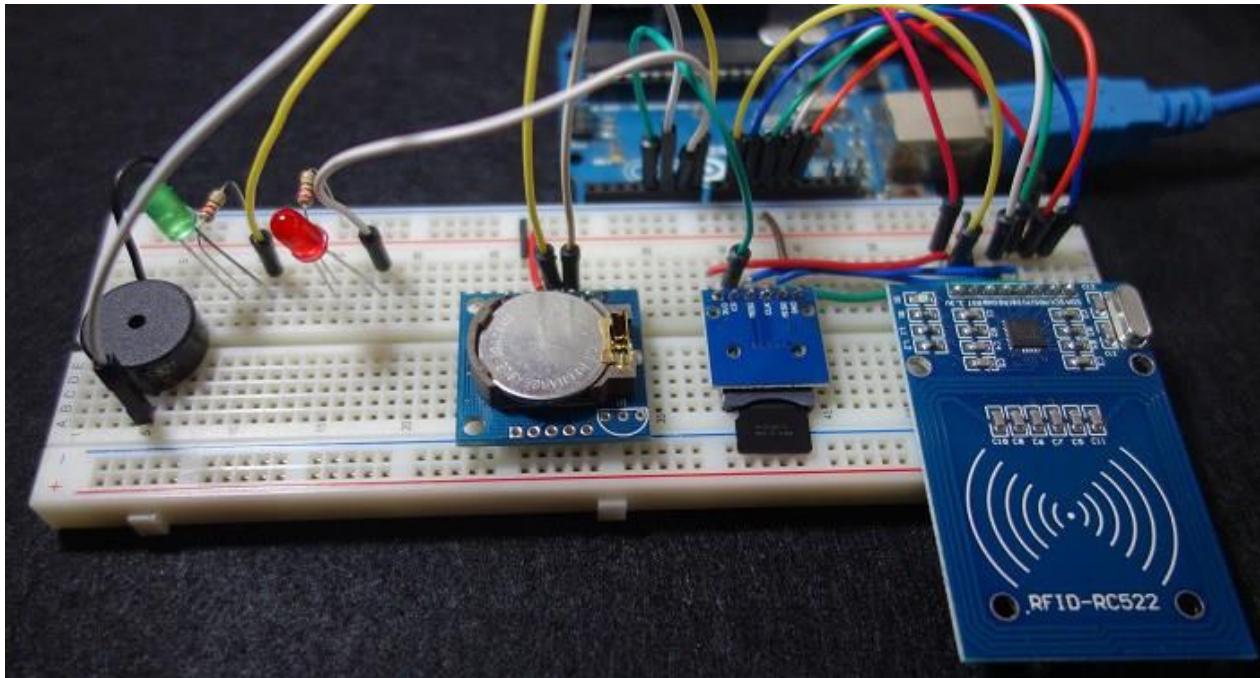
Approximate the card you've chosen to give access and you'll see:



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



Arduino Time Attendance System with RFID

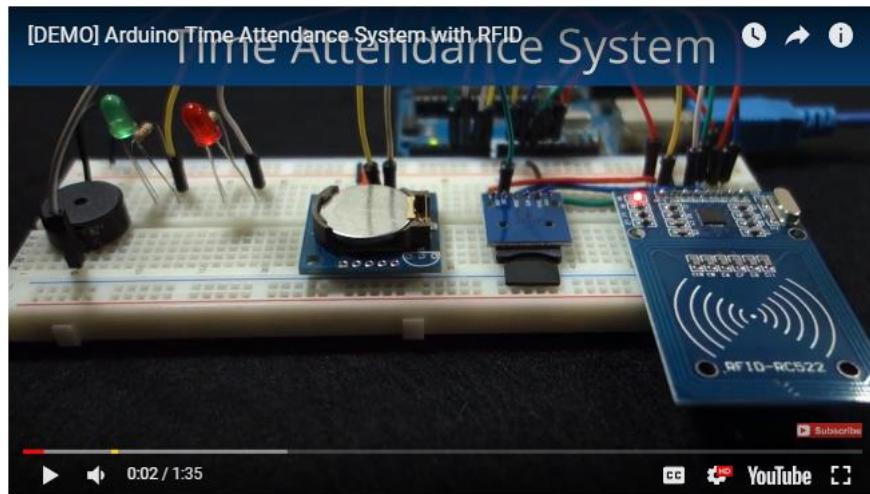


View Project on Random Nerd Tutorials	Click here
Watch on YouTube	Click here
View code on GitHub	Click here

Introduction

In this project you're going to build a time attendance system with the MFRC522 RFID Reader and Arduino. When you swipe an RFID tag next to the RFID reader, it saves the user UID and time in an SD card. It also shows if you are late or in time accordingly to a preset hour and minute.

Watch the video intro



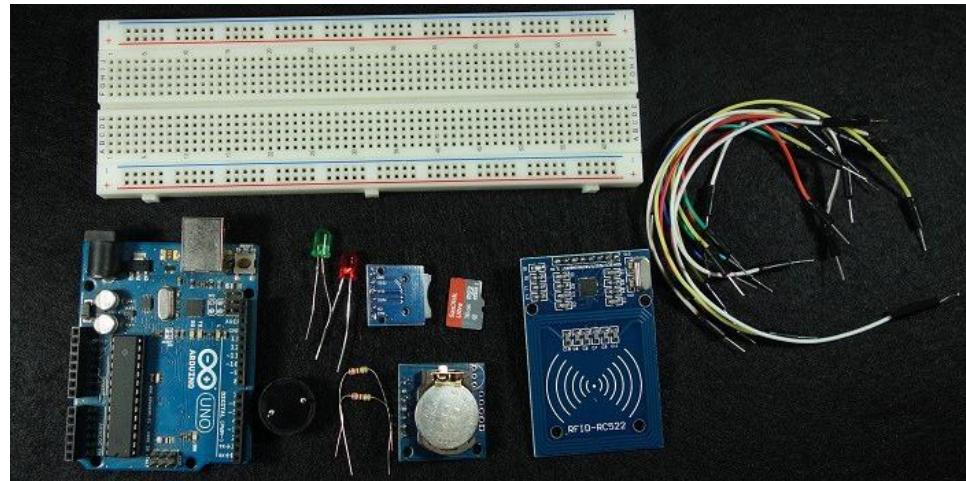
Watch video on Youtube: <https://youtu.be/b01k6YlaHfA>

Project Overview

Before getting started it's important to layout the project main features:

- It contains an RFID reader that reads RFID tags;
- Our setup has a real time clock module to keep track of time;
- When the RFID reader reads an RFID tag, it saves the current time and the UID of the tag in an SD card;
- The Arduino communicates with the SD card using an SD card module;
- You can set a check-in time to compare if you are in time or late;
- If you are on time, a green LED lights up; if you are late, a red LED lights up;
- The system also has a buzzer that beeps when a tag is read.

Parts Required



Here's a list of the required components for this project:

- [Arduino UNO](#) – read [Best Arduino Starter Kits](#)
- [MFRC522 RFID reader + tags](#)
- [SD card module](#)
- [Micro SD card](#)
- [SD1307 RTC module](#)
- [2x LEDs \(1x red + 1x green\)](#)
- [2x 220 Ohm resistor](#)
- [Breadboard](#)
- [Jumper wires](#)

MFRC522 RFID Reader

In this project we're using the MFRC522 RFID reader and that's the one we recommend you to get.

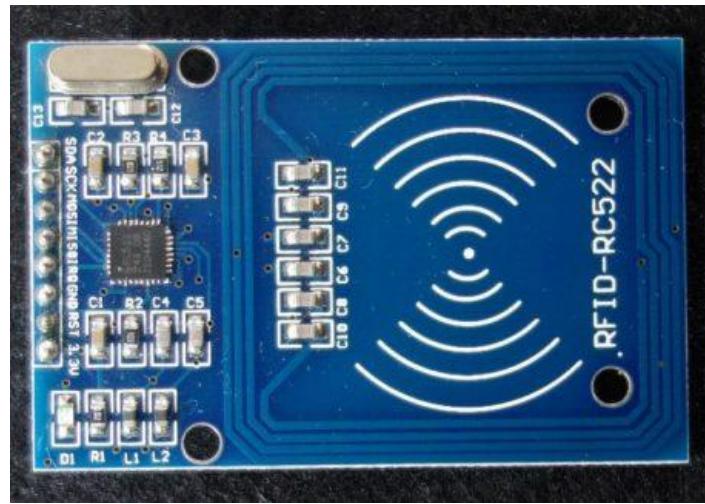
RFID means radio-frequency identification. RFID uses electromagnetic fields to transfer data over short distances and it's useful to identify people, to make transactions, etc.

An RFID system needs tags and a reader:

- **Tags** are attached to the object to be identified, in this example we have a keychain and an electromagnetic card. Some stores also use RFID tags in their products' labels to identify them. Each tag has its own unique identification (UID).



- **Reader** is a two-way radio transmitter-receiver that sends a signal to the tag and reads its response.



The MFRC522 RFID reader works at 3.3V and it can use SPI or I2C communication. The [library](#) we're going to use to control the RFID reader only supports SPI, so that's the communication protocol we're going to use.

To learn more about the RFID reader with the Arduino read: [Security Access using MFRC522 RFID Reader with Arduino](#)

Installing the MFRC522 Library

This project uses the MFRC522.h library to control the RFID reader. This library doesn't come installed in Arduino IDE by default, so you need to install it. Go to **Sketch > Include library > Manage libraries** and search for **MFRC522** or follow the next steps:

1. [Click here to download the MFRC522 library](#). 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

MFRC522 RFID Reader Pinout

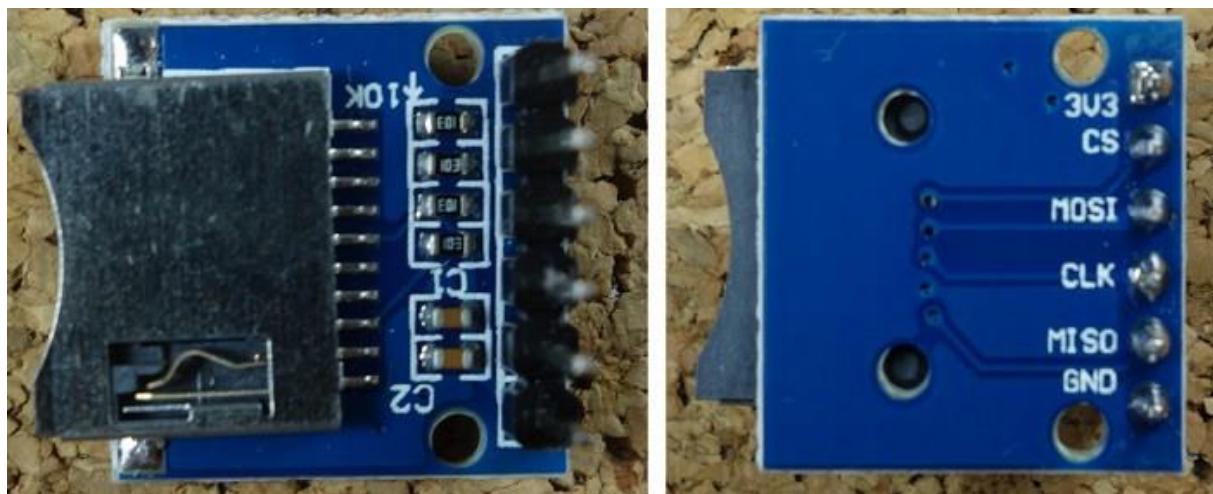
The following table shows the reader pinout for a future reference:

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

Note: different Arduino boards have different SPI pins. If you're using another Arduino board, check the Arduino [documentation](#).

SD Card Module

When a tag is read, its UID and time are saved on an SD card so that you can keep track of check-ins. There are different ways to use an SD card with the Arduino. In this project we're using the SD card module shown in figure below – it works with micro SD card.



There are different models from different suppliers, but they all work in a similar way, using the SPI communication protocol. To communicate with the SD card we're going to use a library called **SD.h** that comes already installed in Arduino IDE by default. To learn more about the SD card module with the Arduino read: [Guide to SD Card Module with Arduino](#)

SD Card Module Pinout

The following table shows the SD card module pinout for a future reference:

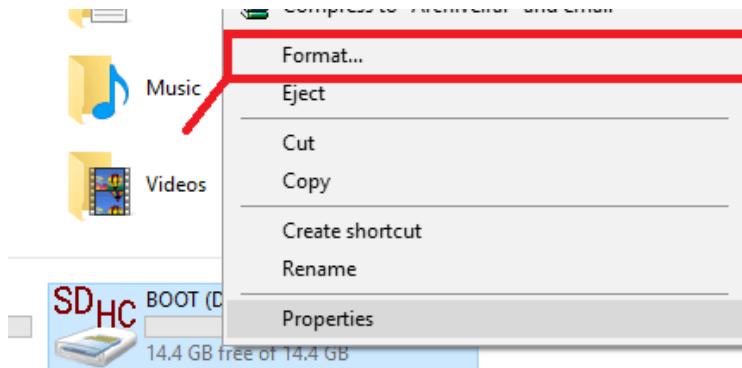
PIN	WIRING TO ARDUINO UNO
VCC	3.3V
CS	Digital 4
MOSI	Digital 11
CLK	Digital 13
MISO	Digital 12
GND	GND

Note: different Arduino boards have different SPI pins. If you're using another Arduino board, check the Arduino [documentation](#).

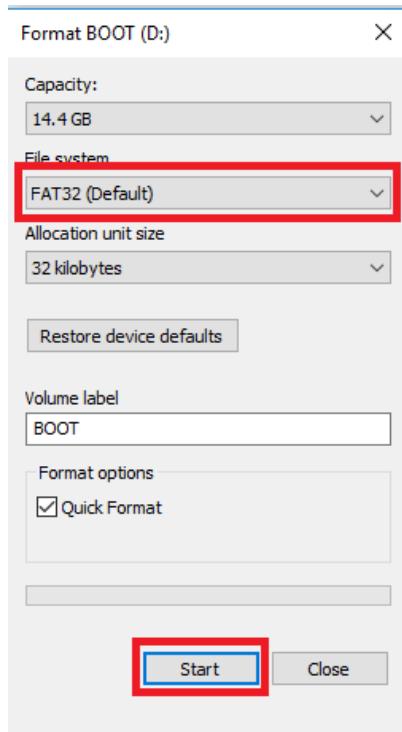
Preparing the SD Card

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.

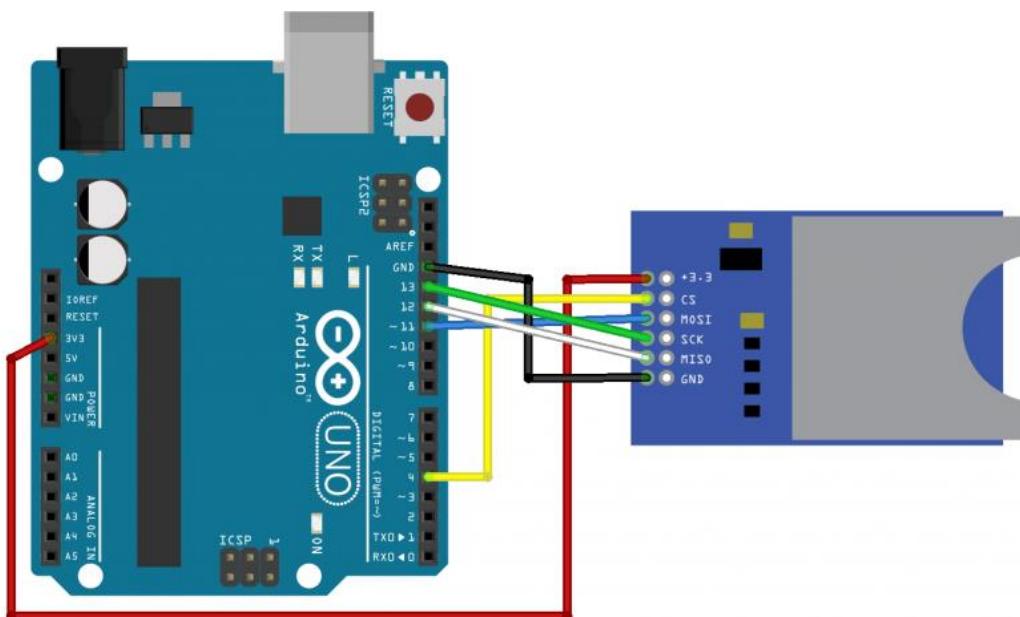


2) A new window pops up. Select **FAT32**, press **Start** to initialize the formatting process and follow the onscreen instructions.



Testing the SD Card Module

This step is optional. This is an additional step to make sure the SD card module is working properly. Insert the formatted SD card in the SD card module and connect the SD card module to the Arduino as shown in the circuit schematic below or check the pinout table.



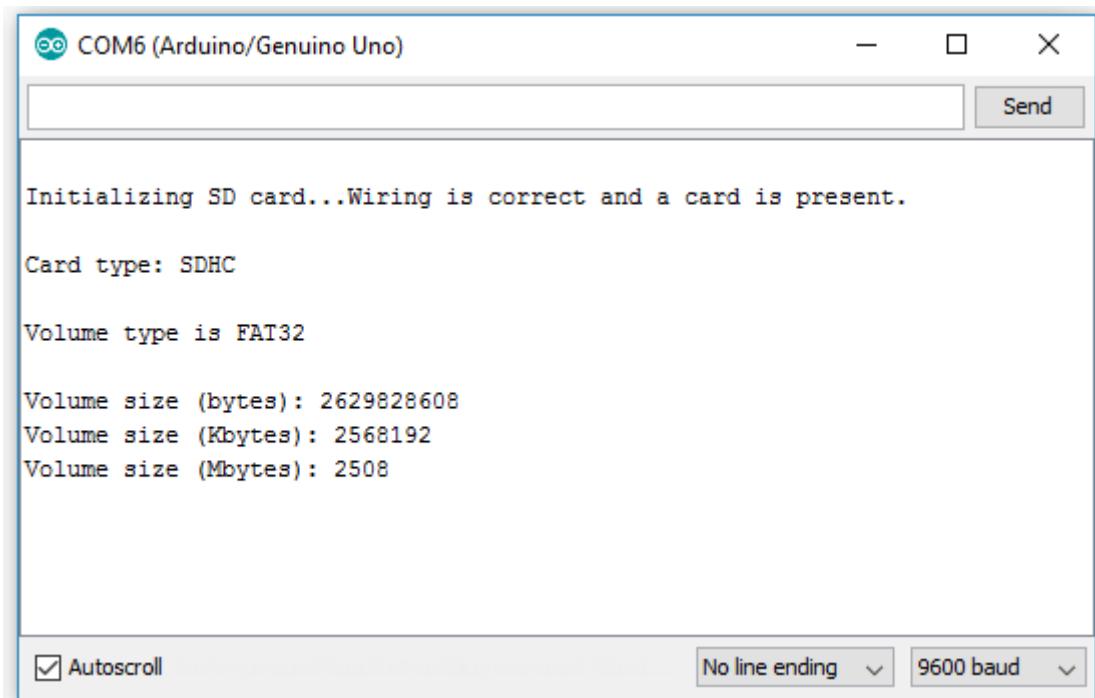
Note: depending on the module you're using, the pins may be placed in a different location.

Uploading CardInfo Sketch

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 the SD card information will be displayed. If everything is working properly you'll see a similar message on the serial monitor.



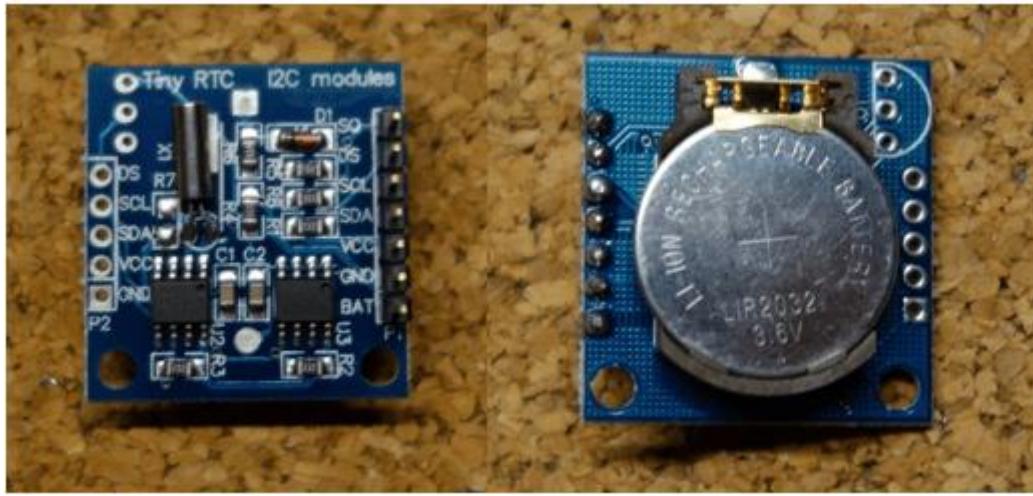
The screenshot shows the Arduino Serial Monitor window titled "COM6 (Arduino/Genuino Uno)". The window contains the following text output:

```
Initializing SD card...Wiring is correct and a card is present.  
Card type: SDHC  
Volume type is FAT32  
Volume size (bytes): 2629828608  
Volume size (Kbytes): 2568192  
Volume size (Mbytes): 2508
```

At the bottom of the window, there are three buttons: "Autoscroll" (checked), "No line ending", and "9600 baud".

RTC (Real Time Clock) Module

To keep track of time, we're using the SD1307 RTC module. However, this project works just fine with the DS3231, which is very similar. One main difference between them is the accuracy. The DS3231 is much more accurate than the SD1307. The figure below shows the SD1307 model.



The module has a backup battery installed. This allows the module to retain the time, even when it's not being powered up.

This module uses I2C communication and we'll use the **RTClib.h** library to read the time from the RTC.

To learn more about the DS1307 real time clock with the Arduino read: [Guide for Real Time Clock \(RTC\) Module with Arduino \(DS1307 and DS3231\)](#)

RTC Module Pinout

The following table shows the RTC module pinout for a future reference:

Pin	Wiring to arduino uno
SCL	A5
SDA	A4
VCC	5V (check your module datasheet)
GND	GND

Note: different Arduino boards have different I2C pins. If you're using another Arduino board, check the Arduino [documentation](#).

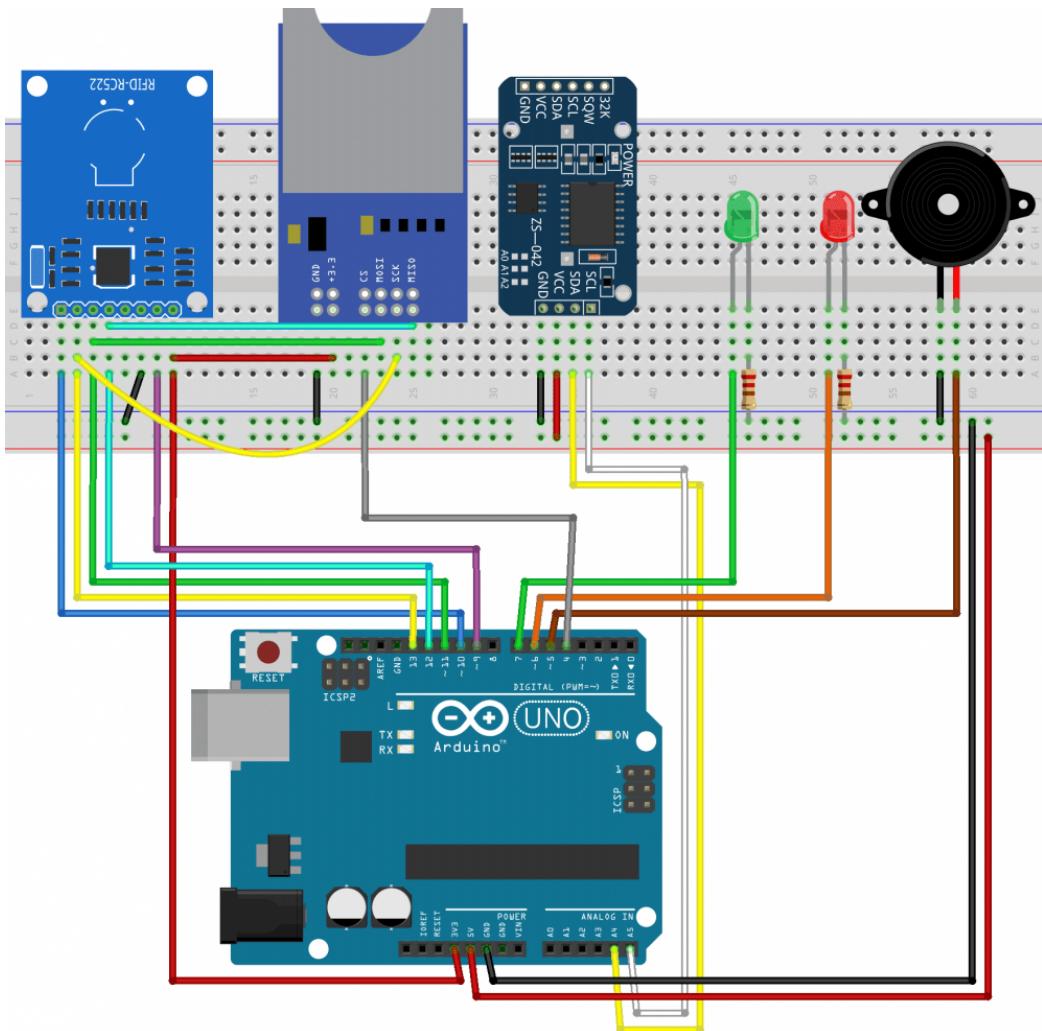
Installing the RTCLib library

To install the RTCLib.h go to **Sketch > Include library > Manage libraries** and search for **RTCLib** or follow the next steps:

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

Schematics

The circuit for this project is shown in the circuit schematics below. In this circuit there are 3.3V and 5V devices, make sure you wire them correctly. Also, if you're using different modules, check the recommend voltage before powering the circuit.



Code

Upload the following code to your Arduino. Make sure you have the right Board and COM Port selected.

[View code on GitHub](#)

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

#include <MFRC522.h> // for the RFID
#include <SPI.h> // for the RFID and SD card module
#include <SD.h> // for the SD card
#include <RTClib.h> // for the RTC

// define pins for RFID
#define CS_RFID 10
#define RST_RFID 9
// define select pin for SD card module
#define CS_SD 4

// Create a file to store the data
File myFile;

// Instance of the class for RFID
MFRC522 rfid(CS_RFID, RST_RFID);

// Variable to hold the tag's UID
String uidString;

// Instance of the class for RTC
RTC_DS1307 rtc;

// Define check in time
const int checkInHour = 9;
const int checkInMinute = 5;

//Variable to hold user check in
int userCheckInHour;
int userCheckInMinute;
```

```

// Pins for LEDs and buzzer
const int redLED = 6;
const int greenLED = 7;
const int buzzer = 5;

void setup() {

    // Set LEDs and buzzer as outputs
    pinMode(redLED, OUTPUT);
    pinMode(greenLED, OUTPUT);
    pinMode(buzzer, OUTPUT);

    // Init Serial port
    Serial.begin(9600);
    while(!Serial); // for Leonardo/Micro/Zero

    // Init SPI bus
    SPI.begin();
    // Init MFRC522
    rfid.PCD_Init();

    // Setup for the SD card
    Serial.print("Initializing SD card...");
    if(!SD.begin(CS_SD)) {
        Serial.println("initialization failed!");
        return;
    }
    Serial.println("initialization done.");

    // Setup for the RTC
    if(!rtc.begin()) {
        Serial.println("Couldn't find RTC");
        while(1);
    }
    else {
        // following line sets the RTC to the date & time this sketch was compiled
        rtc.adjust(DateTime(F(__DATE__), F(__TIME__)));
    }
    if(!rtc.isrunning()) {
        Serial.println("RTC is NOT running!");
    }
}

```

```

void loop() {
    //look for new cards
    if(rfid.PICC_IsNewCardPresent()) {
        readRFID();
        logCard();
        verifyCheckIn();
    }
    delay(10);
}

void readRFID() {
    rfid.PICC_ReadCardSerial();
    Serial.print("Tag UID: ");
    uidString = String(rfid.uid.uidByte[0]) + " " + String(rfid.uid.uidByte[1])
+ " " +
    String(rfid.uid.uidByte[2]) + " " + String(rfid.uid.uidByte[3]);
    Serial.println(uidString);

    // Sound the buzzer when a card is read
    tone(buzzer, 2000);
    delay(100);
    noTone(buzzer);

    delay(100);
}

void logCard() {
    // Enables SD card chip select pin
    digitalWrite(CS_SD, LOW);
    // Open file
    myFile=SD.open("DATA.txt", FILE_WRITE);
    // If the file opened ok, write to it
    if (myFile) {
        Serial.println("File opened ok");
        myFile.print(uidString);
        myFile.print(", ");

        // Save time on SD card
        DateTime now = rtc.now();
        myFile.print(now.year(), DEC);
        myFile.print('/');
        myFile.print(now.month(), DEC);
        myFile.print('/');
    }
}

```

```

myFile.print(now.day(), DEC);
myFile.print(',');
myFile.print(now.hour(), DEC);
myFile.print(':');
myFile.println(now.minute(), DEC);
// Print time on Serial monitor
Serial.print(now.year(), DEC);
Serial.print('/');
Serial.print(now.month(), DEC);
Serial.print('/');
Serial.print(now.day(), DEC);
Serial.print(' ');
Serial.print(now.hour(), DEC);
Serial.print(':');
Serial.println(now.minute(), DEC);
Serial.println("successfully written on SD card");
myFile.close();
// Save check in time;
userCheckInHour = now.hour();
userCheckInMinute = now.minute();
}

else {
    Serial.println("error opening data.txt");
}

// Disables SD card chip select pin
digitalWrite(CS_SD, HIGH);
}

void verifyCheckIn() {
    if((userCheckInHour < checkInHour) || ((userCheckInHour==checkInHour) &&
(userCheckInMinute <= checkInMinute))){
        digitalWrite(greenLED, HIGH);
        delay(2000);
        digitalWrite(greenLED, LOW);
        Serial.println("You're welcome!");
    }
    else{
        digitalWrite(redLED, HIGH);
        delay(2000);
        digitalWrite(redLED, LOW);
        Serial.println("You are late...");
    }
}
}

```

How the Code Works

Continue reading to learn how the code works.

Importing libraries

The code starts by importing the needed libraries. The **MFRC522** for the RFID reader, the **SD** for the SD card module and the **RTClib** for the RTC. You also include the **SPI** library for SPI communication with the RFID and SD card module.

```
#include <MFRC522.h> // for the RFID
#include <SPI.h>      // for the RFID and SD card module
#include <SD.h>        // for the SD card
#include <RTClib.h>    // for the RTC
```

Preparing RFID reader, SD card and RTC

Then, you define the pins for the RFID reader and the SD card module. For the RFID, the SCK pin (**CS_RFID**) is connected to pin 10 and the RST pin (**RST_RFID**) is connected to pin 9. For the SD card module, the Chip Select pin (**CS_SD**) is connected to pin 4.

```
// define pins for RFID
#define CS_RFID 10
#define RST_RFID 9
// define chip select pin for SD card module
#define CS_SD 4
```

You create a File called **myFile** to store your data.

```
File myFile;
```

Then, you create an instance for the RFID and for the RTC:

```
// Instance of the class for RFID
MFRC522 rfid(CS_RFID, RST_RFID);
// Instance of the class for RTC
RTC_DS1307 rtc;
```

Variables

You create a string variable **uidString** that holds the UID tags.

```
String uidString;
```

The following lines create variables to define the check in time hour and minute. In this case, we're defining the check in hour to 9h05m AM. You can change the check in time by changing these values:

```
// Define check in time  
const int checkInHour = 9;  
const int checkInMinute = 5;
```

You also need to create variables to hold the user's check in hour. These variables will save the hour a certain UID tag was read. The following variables hold the check-in hour and the check-in minute.

```
//Variable to hold user check in  
int userCheckInHour;  
int userCheckInMinute;
```

Finally you attribute the pin numbers to the LEDs and buzzer.

```
// Pins for LEDs and buzzer  
const int redLED = 6;  
const int greenLED = 7;  
const int buzzer = 5;
```

setup()

Next, in the **setup()** you set the LEDs and buzzer as outputs.

```
// Set LEDs and buzzer as outputs  
pinMode(redLED, OUTPUT);  
pinMode(greenLED, OUTPUT);  
pinMode(buzzer, OUTPUT);
```

After that, each module is initialized.

Functions

In this code you create 3 functions: **readRFID()**, **logCard()** and **verifyCheckIn()**.

The **readRFID()** function reads the tag UID, saves it in the **uidString** variable and displays it on the serial monitor. Also, when it reads the tag, the buzzer makes a beep sound.

The **logCard()** function creates a file on your SD card called **DATA.txt**. You can edit the name of the file, if you want, on the following line.

```
myFile=SD.open("DATA.txt", FILE_WRITE);
```

Then, it saves the **uidString** (that holds the UID of the tag) on the SD card and the current time.

```
myFile.print(uidString);  
// Save time on SD card  
DateTime now = rtc.now();  
myFile.print(now.year(), DEC);  
myFile.print('/');  
myFile.print(now.month(), DEC);  
myFile.print('/');  
myFile.print(now.day(), DEC);  
myFile.print(',');  
myFile.print(now.hour(), DEC);  
myFile.print(':');  
myFile.print(now.minute(), DEC);
```

It also saves the user check-in hour and minute in the following variables for further comparison with the predefined check-in time.

```
userCheckInHour = now.hour();  
userCheckInMinute = now.minute();
```

The **verifyCheckIn()** function simply compares the user check-in time with the predefined check in hour and gives feedback accordingly. If the user is late, the red LED lights up; if the user is on time, the green LED lights up.

loop()

After studying the created functions, the loop() is pretty straightforward to understand.

First, the code checks if an RFID tag was swiped. If yes, it will read the RFID UID, log the UID and the time into the SD card, and then it will give feedback to the user by lighting up one of the LEDs.

Grabbing the SD Card Data

To check the data saved on the SD card, remove it from the SD card module and insert it on your computer.

Open the SD card folder and you should have a file called **DATA.txt**.



Open the file using a text editor. You'll have something as follows:

A screenshot of a Windows Notepad window titled "DATA - Notepad". The window shows a list of data entries, each consisting of a sequence of numbers separated by commas, followed by a timestamp. The data is as follows:
189 49 21 43, 2017/8/30,14:23
34 74 156 11, 2017/8/30,14:23
130 114 159 11, 2017/8/30,14:23
98 236 236 14, 2017/8/30,14:23
130 31 148 111, 2017/8/30,14:23
189 49 21 43, 2017/8/30,14:23
189 49 21 43, 2017/8/30,14:23
98 236 236 14, 2017/8/30,14:23
34 30 114 11, 2017/8/30,14:23
34 74 156 11, 2017/8/30,14:23
130 31 148 111, 2017/8/30,14:24

Notice that each value is separated by commas. This makes it easier if you want to import this data to Excel, Google Sheets, or other data processing software.

Wrapping Up

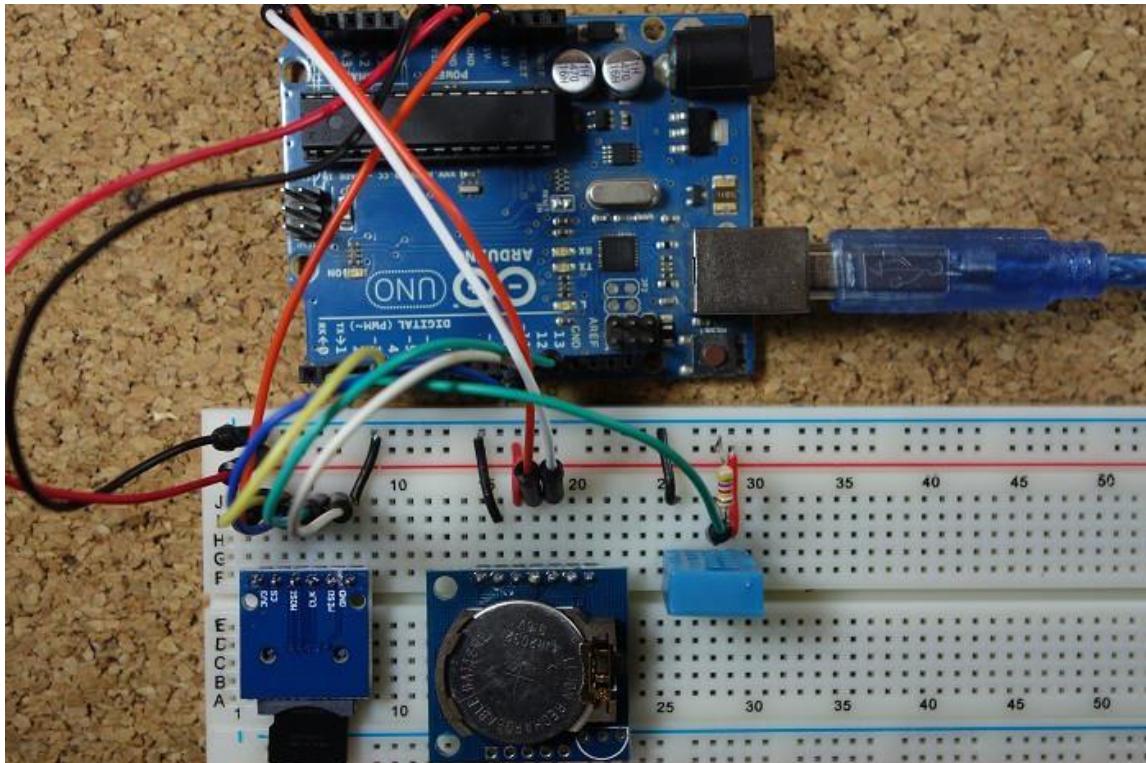
In this project you've learned how to use an RFID card reader and the SD card module with Arduino. You can modify this project to your own needs or you can use the functions created here in other projects that require data logging or reading RFID tags.

You can take this project further and add a display to give extra feedback to the user. You can associate a name with each UID and display the user name when the tag is read. You may want to take a look at some tutorials with the Arduino using displays:

- [Guide for 0.96 inch OLED Display with Arduino](#)
- [Nextion Display with Arduino – Getting Started](#)

This is an excerpt from our course "[Arduino Step-by-step Projects](#)". If you like Arduino and you want to make more projects, we recommend downloading our course: [Arduino Step-by-step Projects](#).

Arduino Temperature Data Logger with SD Card Module



View Project on Random Nerd Tutorials

Click [here](#)

View code on GitHub

Click [here](#)

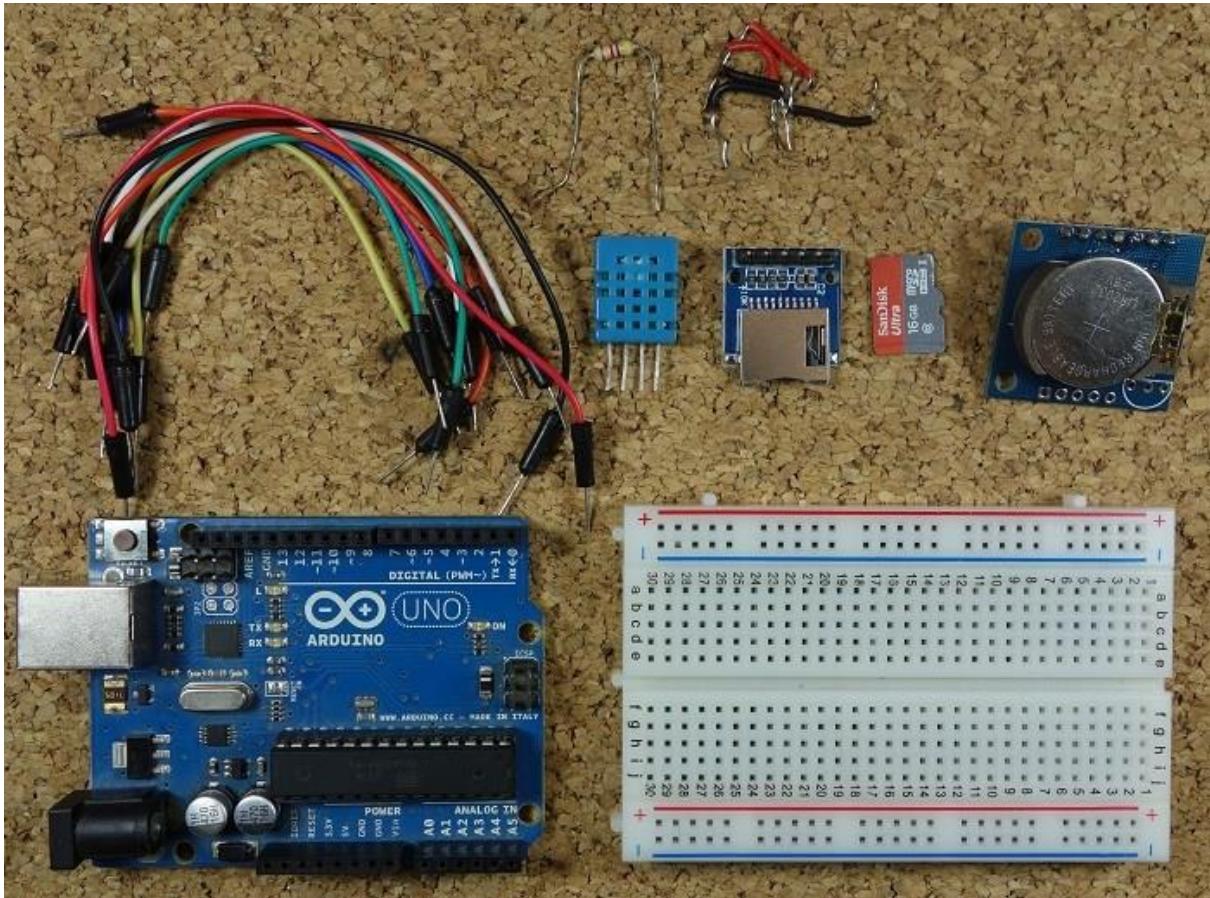
This project shows you how to create an Arduino temperature data logger. We'll use the DHT11 to measure temperature, the real time clock (RTC) module to take time stamps and the SD card module to save the data on the SD card.

Recommended resources:

- [Guide for Real Time Clock \(RTC\) Module with Arduino \(DS1307 and DS3231\)](#)
- [Complete Guide for DHT11/DHT22 Humidity and Temperature Sensor With Arduino](#)
- [Guide to SD Card module with Arduino](#)

Parts Required

Here's a complete list of the parts required for this project:

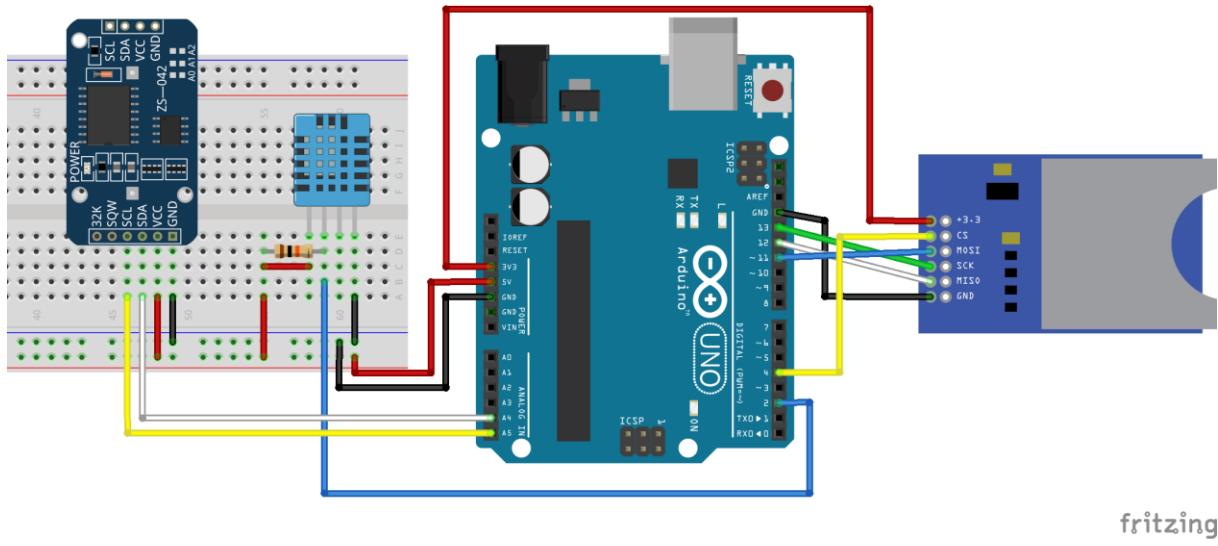


- [Arduino UNO](#) – read [Best Arduino Starter Kits](#)
- [SD card module](#)
- [Micro SD card](#)
- [DHT11 temperature and humidity sensor](#)
- [RTC module](#)
- [Breadboard](#)
- [Jumper wires](#)

Note: alternatively to the SD card module, you can use a [data logging shield](#). The data logging shield comes with built-in RTC and a prototyping area for soldering connections, sensors, etc..

Schematics

The following figure shows the circuit's schematics for this project.



Note: make sure your SD card is formatted and working properly. Read "[Guide to SD card module with Arduino](#)".

Installing the DHT Sensor Library

For this project you need to install the DHT library to read from the DHT11 sensor.

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

Installing the Adafruit_Sensor library

To use the DHT temperature and humidity sensor, you also need to install the [Adafruit Sensor library](#). Follow the next steps to install the library in your Arduino IDE:

1. [Click here to download](#) the Adafruit_Sensor library. You should have a .zip folder in your Downloads folder
 2. Unzip the .zip folder and you should get **Adafruit_Sensor-master** folder
 3. Rename your folder from **Adafruit_Sensor-master** to **Adafruit_Sensor**

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

Code

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

[View code on GitHub](#)

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

#include <SPI.h> //for the SD card module
#include <SD.h> // for the SD card
#include <DHT.h> // for the DHT sensor
#include <RTClib.h> // for the RTC

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

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

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

// change this to match your SD shield or module;
// Arduino Ethernet shield and modules: pin 4
// Data loggin SD shields and modules: pin 10
// Sparkfun SD shield: pin 8
const int chipSelect = 4;

// Create a file to store the data
File myFile;

// RTC
RTC_DS1307 rtc;

void setup() {
  //initializing the DHT sensor
  dht.begin();

  //initializing Serial monitor
  Serial.begin(9600);
```

```

// setup for the RTC
while(!Serial); // for Leonardo/Micro/Zero
if(! rtc.begin()) {
    Serial.println("Couldn't find RTC");
    while (1);
}
else {
    // following line sets the RTC to the date & time this sketch was
compiled
    rtc.adjust(DateTime(F(__DATE__)), F(__TIME__));
}
if(! rtc.isrunning()) {
    Serial.println("RTC is NOT running!");
}

// setup for the SD card
Serial.print("Initializing SD card...");

if(!SD.begin(chipSelect)) {
    Serial.println("initialization failed!");
    return;
}
Serial.println("initialization done.");

//open file
myFile=SD.open("DATA.txt", FILE_WRITE);

// if the file opened ok, write to it:
if (myFile) {
    Serial.println("File opened ok");
    // print the headings for our data
    myFile.println("Date,Time,Temperature °C");
}
myFile.close();
}

void loggingTime() {
    DateTime now = rtc.now();
    myFile = SD.open("DATA.txt", FILE_WRITE);
    if (myFile) {
        myFile.print(now.year(), DEC);
        myFile.print('/');
        myFile.print(now.month(), DEC);
        myFile.print('/');
        myFile.print(now.day(), DEC);
        myFile.print(',');
        myFile.print(now.hour(), DEC);
        myFile.print(':');
        myFile.print(now.minute(), DEC);
        myFile.print(':');
        myFile.print(now.second(), DEC);
        myFile.print(",");
    }
}

```

```

}

Serial.print(now.year(), DEC);
Serial.print('/');
Serial.print(now.month(), DEC);
Serial.print('/');
Serial.println(now.day(), DEC);
Serial.print(now.hour(), DEC);
Serial.print(':');
Serial.print(now.minute(), DEC);
Serial.print(':');
Serial.println(now.second(), DEC);
myFile.close();
delay(1000);
}

void loggingTemperature() {
    // Reading temperature or humidity takes about 250 milliseconds!
    // Sensor readings may also be up to 2 seconds 'old' (its a very slow
    sensor)
    // 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(t) /*|| isnan(f)*/) {
        Serial.println("Failed to read from DHT sensor!");
        return;
    }

    //debugging purposes
    Serial.print("Temperature: ");
    Serial.print(t);
    Serial.println(" *C");
    //Serial.print(f);
    //Serial.println(" *F\t");

    myFile = SD.open("DATA.txt", FILE_WRITE);
    if (myFile) {
        Serial.println("open with success");
        myFile.print(t);
        myFile.println(",");
    }
    myFile.close();
}

void loop() {
    loggingTime();
    loggingTemperature();
    delay(5000);
}

```

In this code we create two functions that are called in the loop():

- **loggingTime()**: log time to the **DATA.txt** file in the SD card
- **loggingTemperature()**: log temperature to the **DATA.txt** file in the SD card.

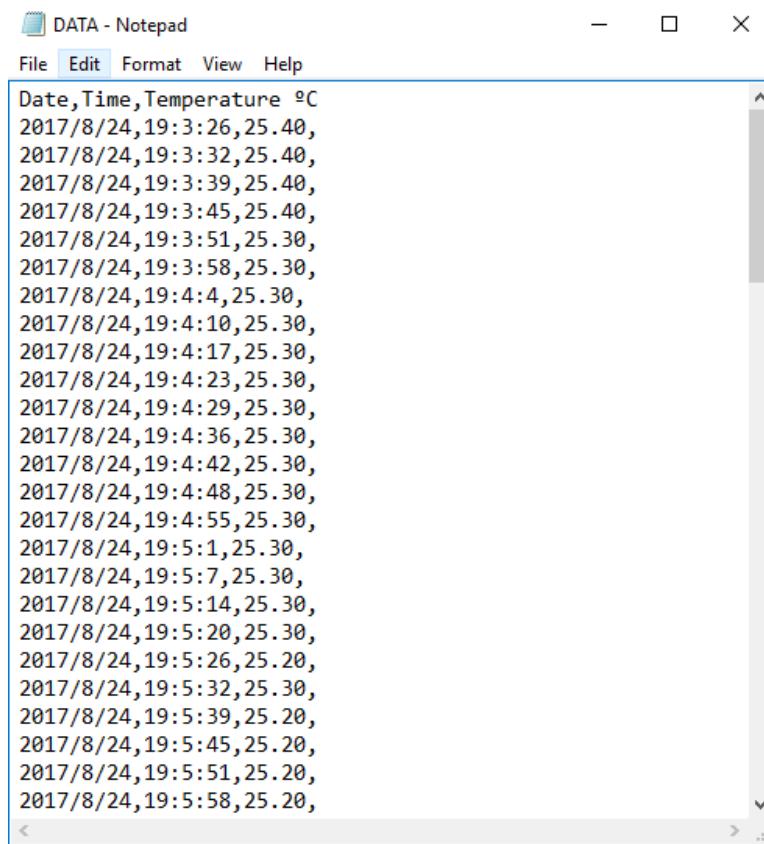
Open the Serial Monitor at a baud rate of 9600 and check if everything is working properly.

Getting the Data from the SD Card

Let this project run for a few hours to gather a decent amount of data, and when you're happy with the data logging period, shut down the Arduino and remove the SD from the SD card module.

Insert the SD card on your computer, open it, and you should have a **DATA.txt** file with the collected data.

You can open the data with a text editor, or use a spreadsheet to analyse and manipulate your data.

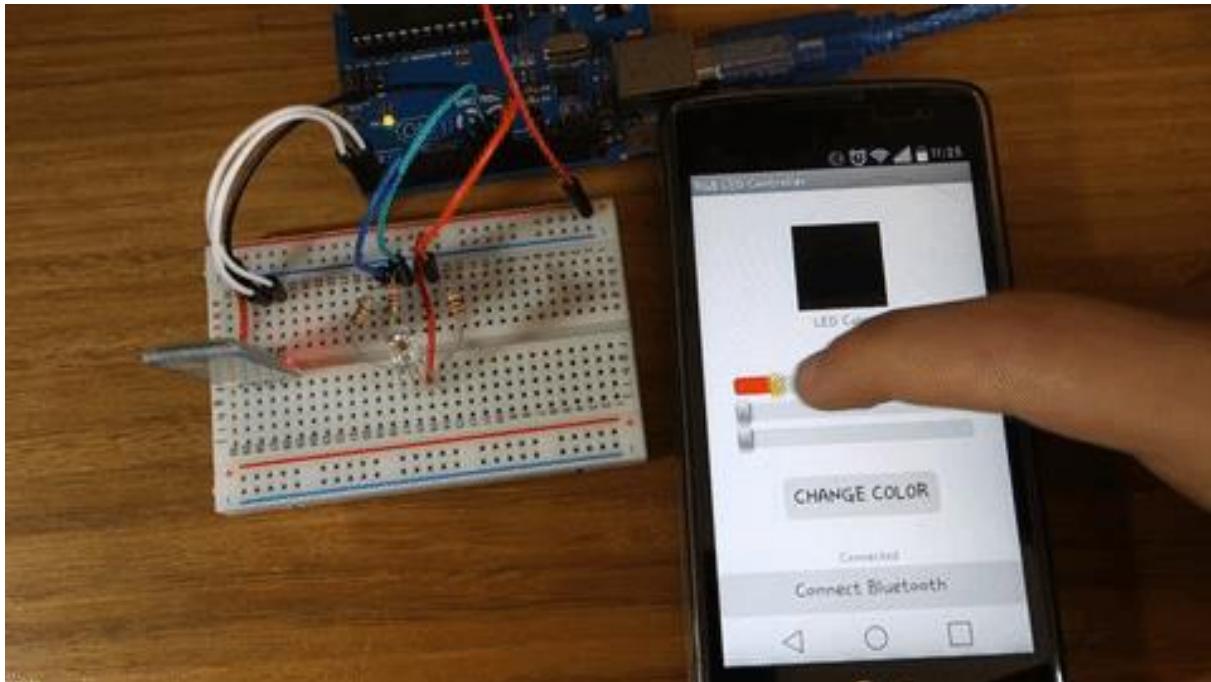


The screenshot shows a Windows Notepad window titled "DATA - Notepad". The menu bar includes File, Edit, Format, View, and Help. The main content area displays a series of log entries, each consisting of three fields separated by commas: Date, Time, and Temperature in degrees Celsius. The data is as follows:

Date	Time	Temperature (°C)
2017/8/24	19:3:26	25.40
2017/8/24	19:3:32	25.40
2017/8/24	19:3:39	25.40
2017/8/24	19:3:45	25.40
2017/8/24	19:3:51	25.30
2017/8/24	19:3:58	25.30
2017/8/24	19:4:4	25.30
2017/8/24	19:4:10	25.30
2017/8/24	19:4:17	25.30
2017/8/24	19:4:23	25.30
2017/8/24	19:4:29	25.30
2017/8/24	19:4:36	25.30
2017/8/24	19:4:42	25.30
2017/8/24	19:4:48	25.30
2017/8/24	19:4:55	25.30
2017/8/24	19:5:1	25.30
2017/8/24	19:5:7	25.30
2017/8/24	19:5:14	25.30
2017/8/24	19:5:20	25.30
2017/8/24	19:5:26	25.20
2017/8/24	19:5:32	25.30
2017/8/24	19:5:39	25.20
2017/8/24	19:5:45	25.20
2017/8/24	19:5:51	25.20
2017/8/24	19:5:58	25.20

The data is separated by commas, and each reading is in a new line. In this format, you can easily import data to Excel or other data processing software.

Android App – RGB LED with Arduino and Bluetooth



View Project on Random Nerd Tutorials	Click here
View code on GitHub	Click here
Download .apk files	Click here
Download .aia files	Click here

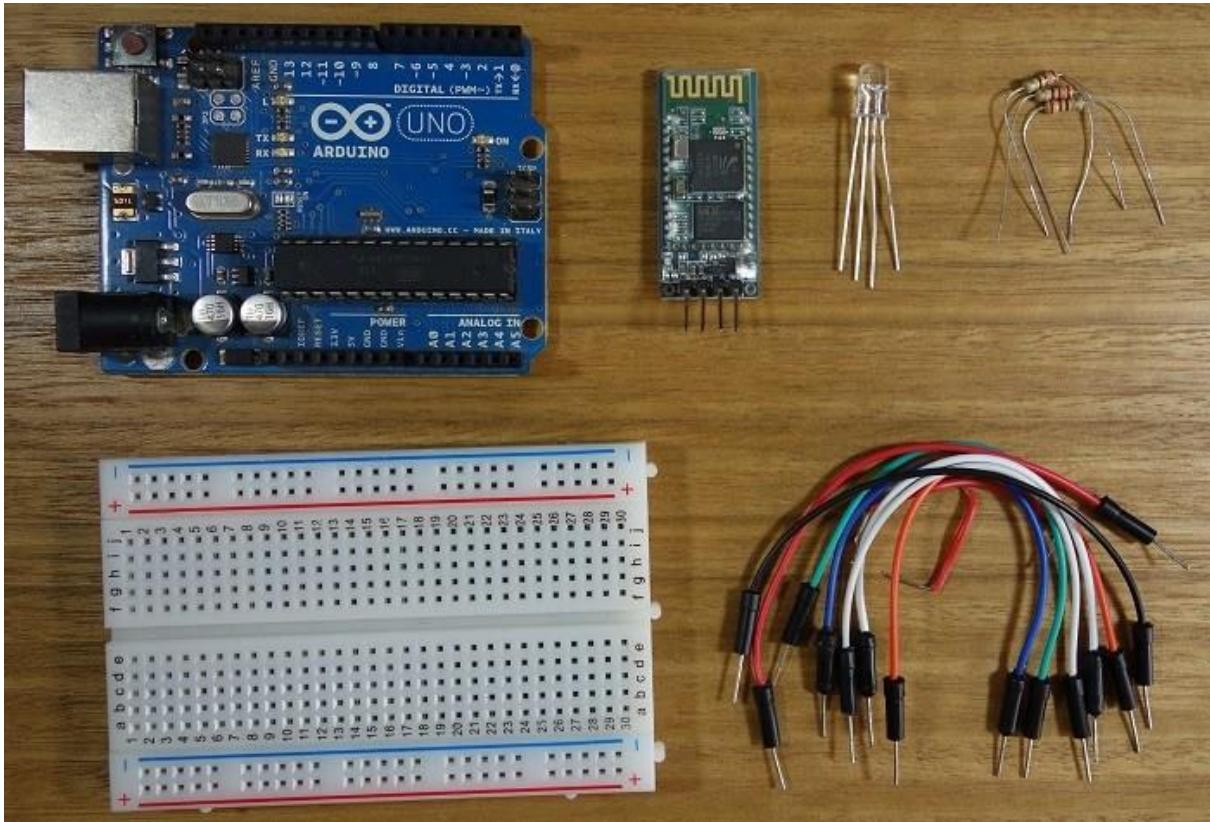
In this project you're going to build an Android app to control the color of an RGB LED with a smartphone via bluetooth.

You're going to build the Android app using a free web based software called MIT App Inventor 2. This is a great project to learn how to interface the Arduino with a smartphone.

If you're not familiar with RGB LEDs, read the following post: [How do RGB LEDs work?](#)

Parts Required

Here's a complete list of the parts required for this project:



- [Arduino UNO](#) – read [Best Arduino Starter Kits](#)
- [Bluetooth module HC-04 or HC-05 or HC-06](#)
- RGB LED (common anode)
- [3 x 220Ohm resistor](#)
- [Breadboard](#)
- [Jumper wires](#)

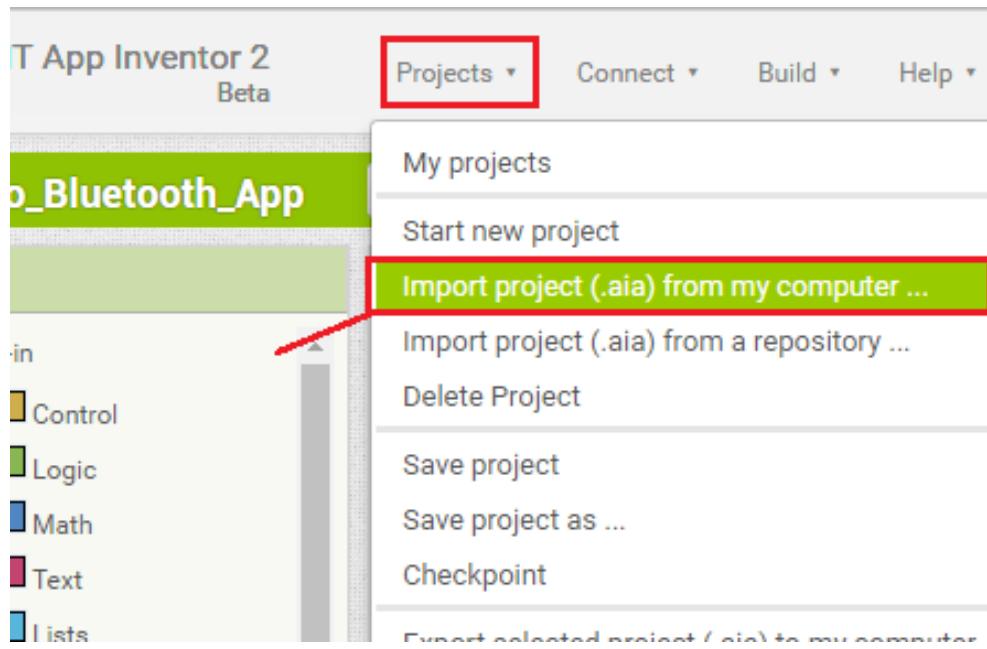
Additionally, you also need a smartphone with bluetooth.

Creating the Android App

The Android App will be created using a free web application called [MIT App Inventor](#). MIT App Inventor is a great place to get started with Android development, because it allows you to build simple apps with drag-n-drop.

You need a Google account to sign up for MIT App Inventor and here's the login page: <http://ai2.appinventor.mit.edu>.

After login in go to **Projects** ▶ **Import project (.aia)** from my computer and upload the .aia file. [Click here to download the .aia file.](#)

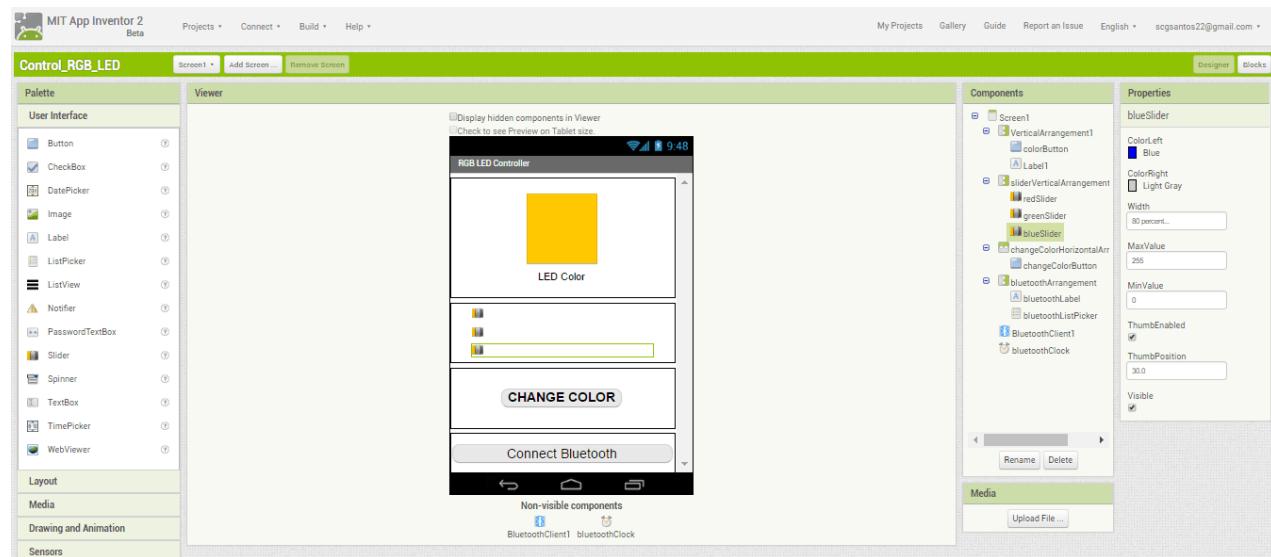


After [uploading the .aia file](#). You'll see the application on the MIT App Inventor Software.

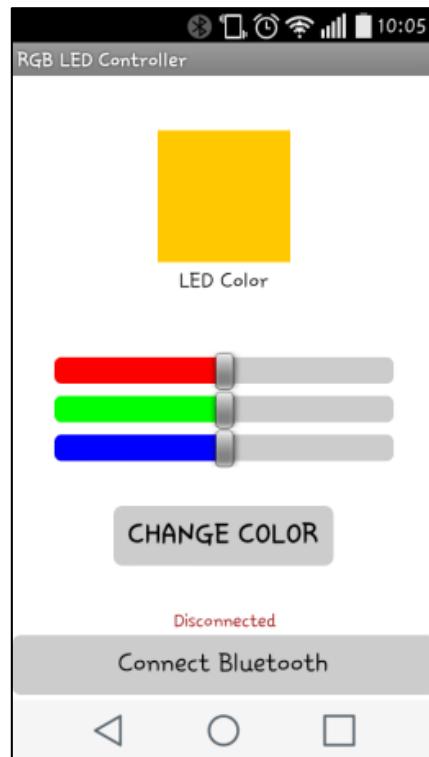
Designer

With MIT App Inventor you have 2 main sections: designer and blocks.

The designer is what gives you the ability to add buttons, add text, add screens and edit the overall app look. In the App Inventor software the app looks like this:

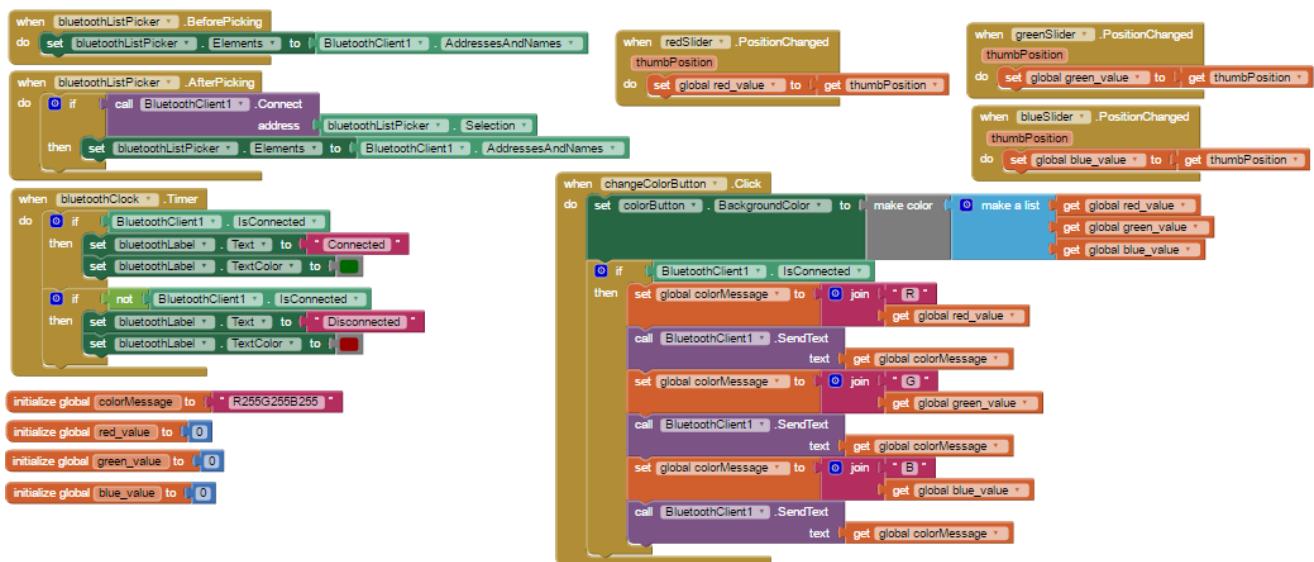


The app looks different in the software and in your smartphone. This is the how the app looks in our smartphone:



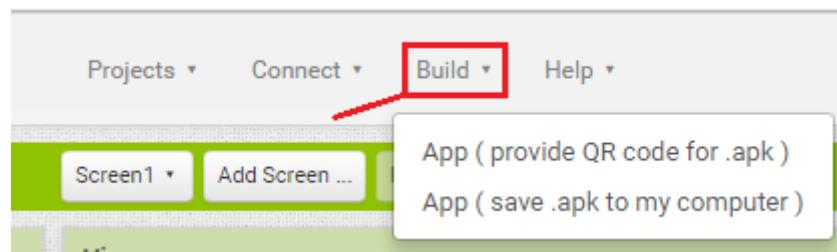
Blocks

You also have the blocks section. The blocks sections is what allows to create custom functionality for your app, so when you press the buttons it actually does something with that information. These are the blocks for this app – visit the project page for a high-resolution figure:



Installing the App

To install the app in your smartphone, go to the Build tab.



You can either generate a QR code that you can scan with your smartphone and automatically install the app in your smartphone.

Or you [can download the .apk file](#), connect your smartphone to your computer and move the .apk file to the phone.

Simply follow the installation wizard to install the App and it's done!

Code

Copy the following code to your Arduino IDE, and upload it to your Arduino board. Make sure you have the right Board and COM port selected.

Note: before uploading the code, make sure you have the TX and RX pins disconnected from the bluetooth module!

[View code on GitHub](#)

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

#define max_char 12
char message[max_char];      // stores your message
char r_char;                  // reads each character
byte index = 0;                // defines the position into your array
int i;
int redPin = 11;              // Red RGB pin -> D11
int greenPin = 10;             // Green RGB pin -> D10
int bluePin = 9;               // Blue RGB pin -> D9
int redValue = 255;            // Red RGB pin -> D11
int greenValue = 255;           // Green RGB pin -> D10
int blueValue = 255;            // Blue RGB pin -> D9
```

```

String redTempValue;      // Red RGB pin -> D11
String greenTempValue;    // Green RGB pin -> D10
String blueTempValue;     // Blue RGB pin -> D9
int flag = 0;
char currentColor;
void setup() {
    pinMode(redPin,OUTPUT);
    pinMode(bluePin,OUTPUT);
    pinMode(greenPin, OUTPUT);
    // initialize serial communication at 9600 bits per second:
    Serial.begin(9600);
}
void loop() {
    //while is reading the message
    while(Serial.available() > 0){
        flag = 0;
        //the message can have up to 12 characters
        if(index < (max_char-1)){
            r_char = Serial.read();           // Reads a character
            message[index] = r_char;         // Stores the character in message array
            if(r_char=='R'){
                currentColor = 'R';
                redTempValue = "";
            }
            else if(r_char=='G'){
                currentColor = 'G';
                greenTempValue = "";
            }
            else if(r_char=='B'){
                currentColor = 'B';
                blueTempValue = "";
            }
            if(currentColor == 'R' && r_char!='R'){
                redTempValue += r_char;
            }
            else if(currentColor == 'G' && r_char!='G'){
                greenTempValue += r_char;
            }
            else if(currentColor == 'B' && r_char!='B'){
                blueTempValue += r_char;
            }
            index++;                         // Increment position
            message[index] = '\0';          // Delete the last position
        }
    }
}

```

```

    }

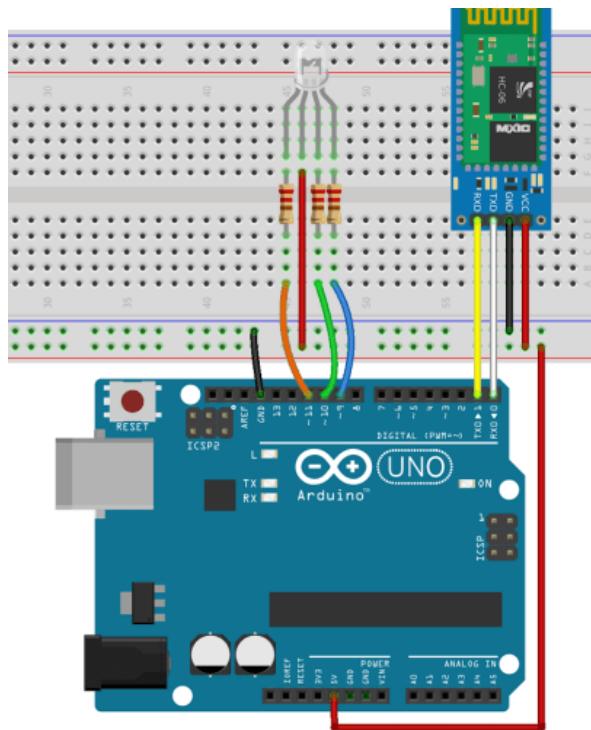
}

if(flag == 0) {
    analogWrite(redPin, 255-redTempValue.toInt());
    analogWrite(greenPin, 255-greenTempValue.toInt());
    analogWrite(bluePin, 255-blueTempValue.toInt());
    /*Serial.print('R');
    Serial.println(redTempValue);
    Serial.print('G');
    Serial.println(greenTempValue);
    Serial.print('B');
    Serial.println(blueTempValue);
    Serial.print("MESSAGE ");
    Serial.println(message);
    flag=1;
    for(i=0; i<12; i++) {
        message[i] = '\0';
    }
    //resests the index
    index=0;
}
}

```

Schematics

Follow the schematic diagram in the following figure to wire your circuit.

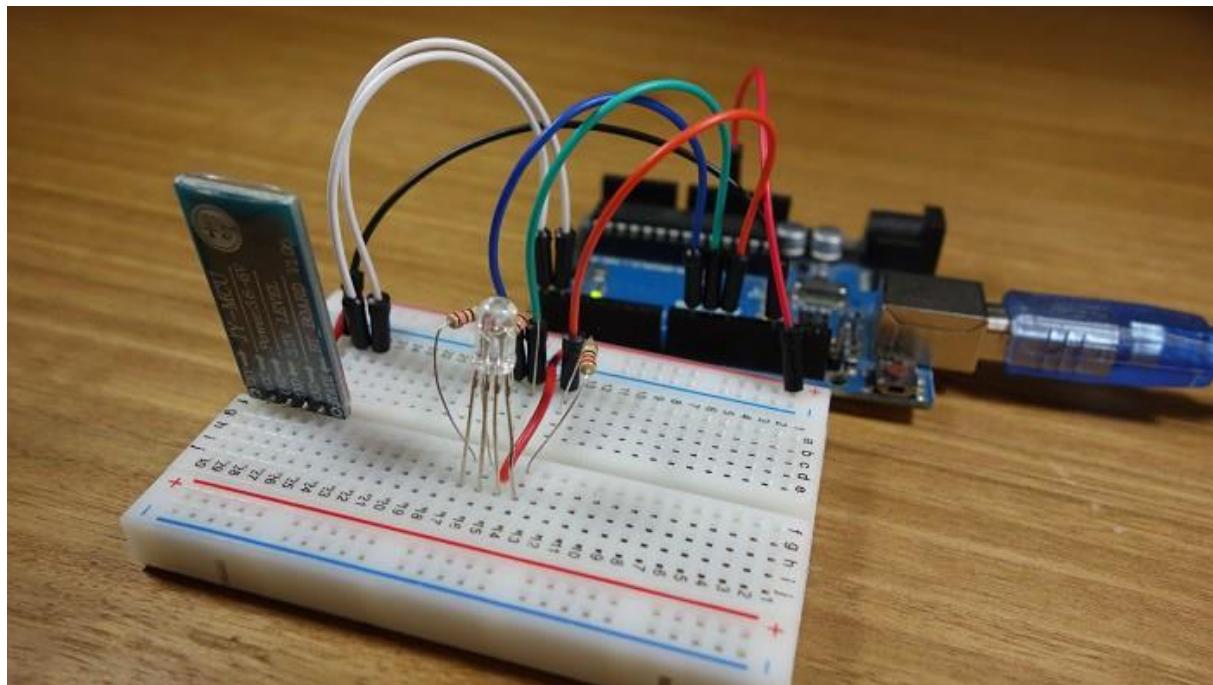


Note: If you're using an RGB LED common cathode, you need to connect the longer lead to GND.

Important: Here's the bluetooth module connections to the Arduino:

- Bluetooth module TX connects to Arduino RX
 - Bluetooth module RX connects to Arduino TX

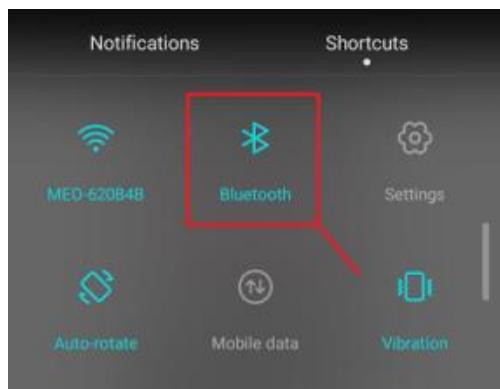
Here's how your circuit should look:



Launching the App

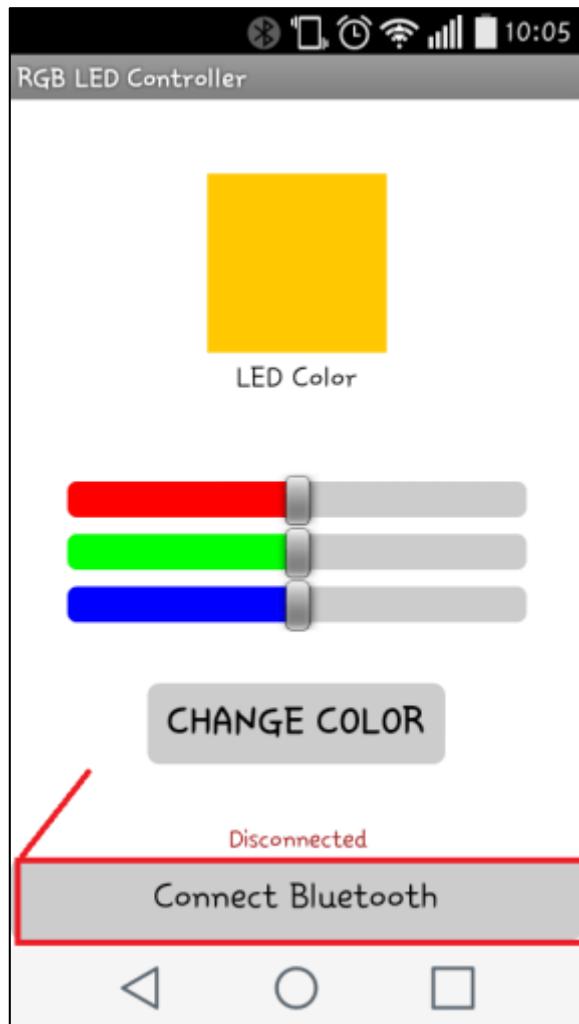
If you haven't generated the .apk file in a previous step, you can [click here to download the .apk file](#) (which is the Android App installation file). Move that file to your smartphone and open it. Follow the installation wizard to install the app.

Enable your smartphone's Bluetooth.

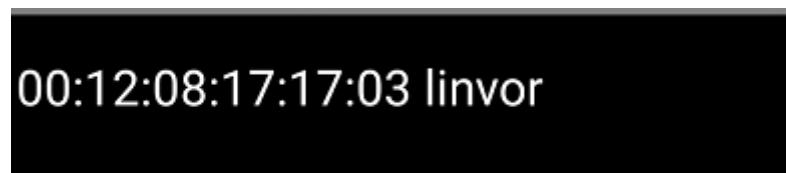


Make sure you pair your smartphone with the bluetooth module – search for paired devices in your smartphone's bluetooth settings.

Then, open the newly installed app. Tap on the Connect bluetooth button to connect via bluetooth to the arduino bluetooth module.



Select your Bluetooth module (it should be named linvor).



Now, it's ready to use! Move the sliders and click on CHANGE COLOR to set your RGB LED color.

Troubleshooting

1. I can't upload code to my Arduino board.

Check if you have the TX and RX cables from the bluetooth module disconnected.

When you're uploading code to your Arduino you should disconnect the TX and RX cables from the bluetooth module. These pins are needed for serial communication between the Arduino and your computer.

2. I can't find my bluetooth module device.

Make sure you have paired your smartphone with your bluetooth module. Go to your bluetooth settings and search for the available devices. Your bluetooth module device should appear (it's often called: linvor, HC-06, HC-04, HC-05 ...). Pair with it. If it asks for a password, it's 1234.

3. The app doesn't interact with the Arduino.

If your Android app is connected to your bluetooth module, it should display the "Connected" message (as shown below). Otherwise, press the "Connect Bluetooth" to establish a bluetooth communication.



Double-check your bluetooth module connections:

- Bluetooth module TX connects to Arduino RX
- Bluetooth module RX connects to Arduino TX

4. My bluetooth module is asking for a password.

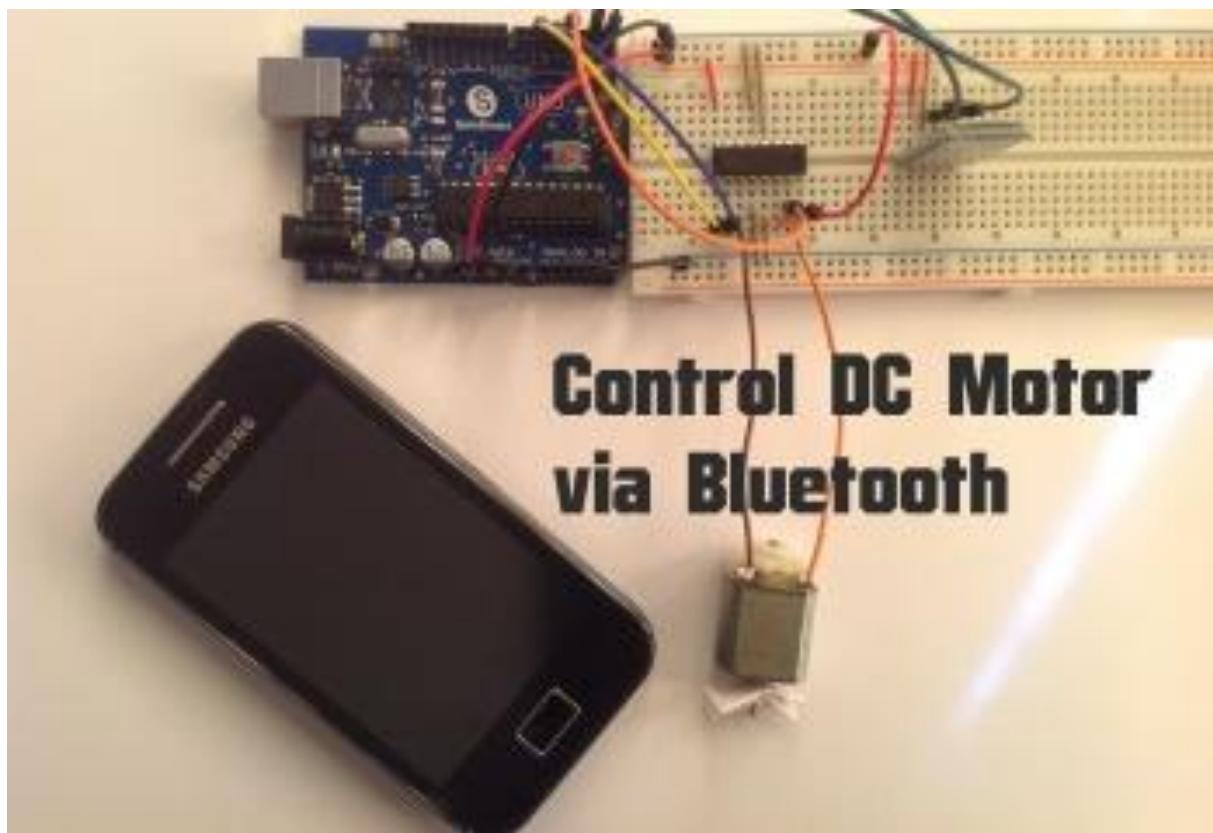
If your bluetooth module asks for a password, type 1234.

Wrapping Up

In this project you learned how to control the color of an RGB LED with an Android App built with the MIT App Inventor 2 software.

Now, feel free to change how the app looks and give it more functionalities. If you like this project, make sure you check our course: [Android Apps for Arduino with MIT App Inventor 2](#).

Control DC Motor via Bluetooth



View Project on Random Nerd Tutorials	Click here
Watch on YouTube	Click here
View code on GitHub	Click here

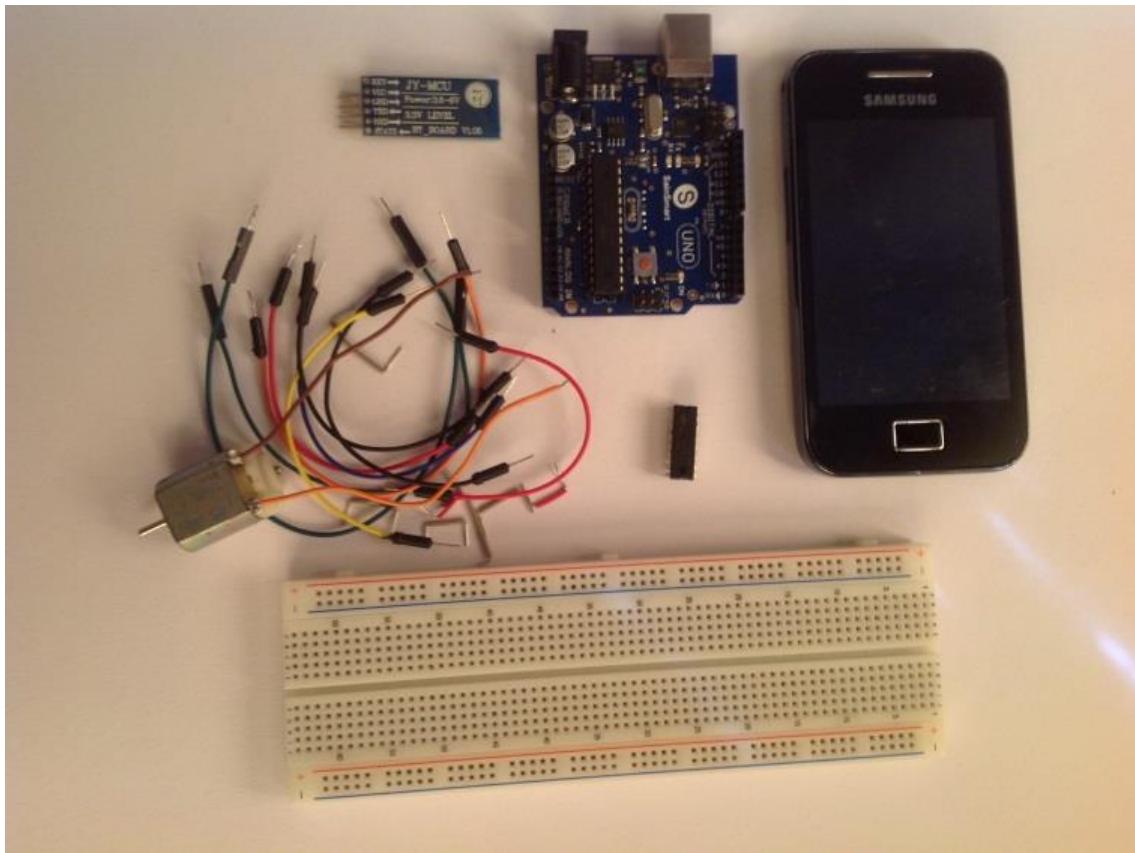
Introduction

In this project we will control a DC motor with a smartphone via bluetooth. This project is great to learn more about:

- DC motors
- Interfacing Arduino with your smartphone
- Bluetooth
- L293D IC

Parts Required

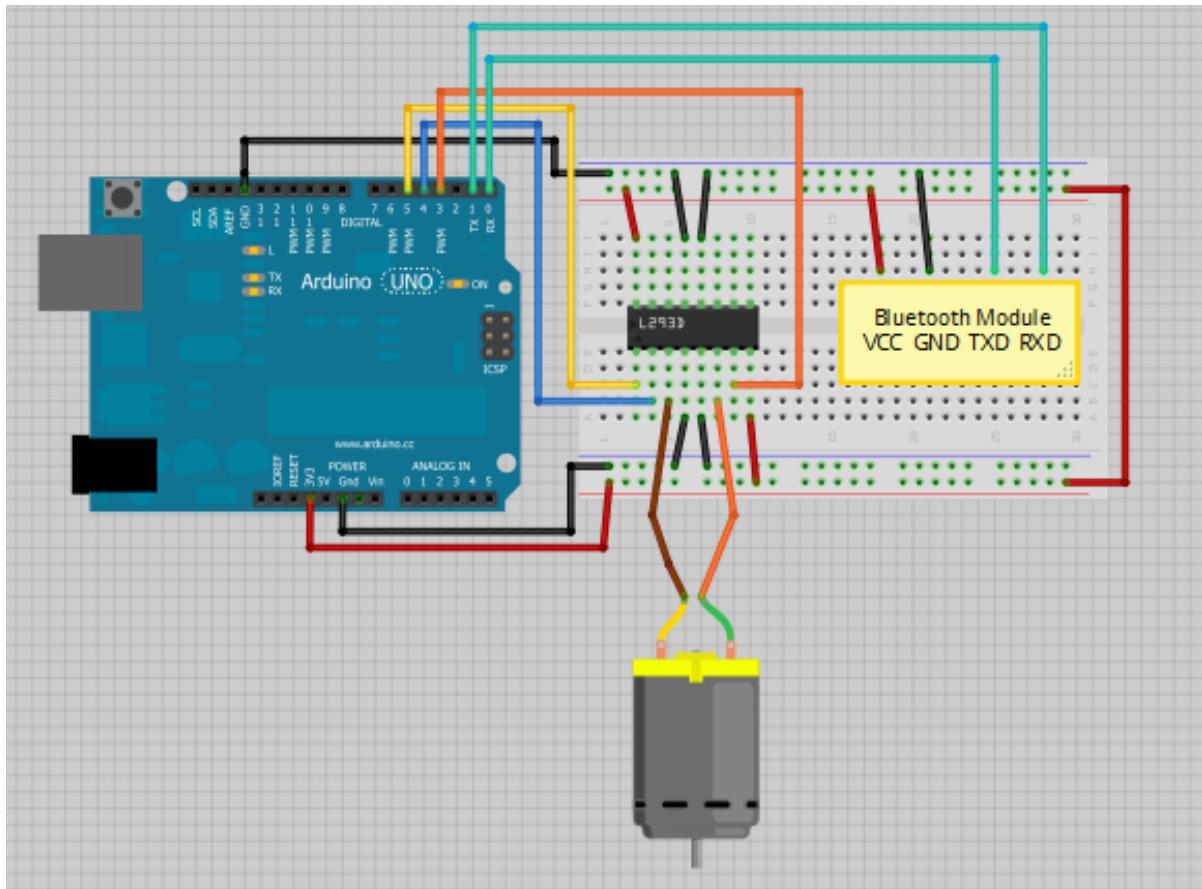
- [Arduino UNO](#) – read [Best Arduino Starter Kits](#)
- [1x Bluetooth Module \(for example: HC-05 or 06\)](#)
- 1x Smartphone (any Android will work)
- BlueTerm application
- 1x L293D IC
- [1x DC motor](#)
- [1x Breadboard](#)
- [Jumper Cables](#)



Schematics

Wire the circuit by following the next schematic diagram.

You can only connect TX and RX cables to your Arduino, after you upload the code.



Two common mistakes

1. You need to remove the RX and TX cables when you're uploading the sketch to your Arduino.
2. Sometimes people connect the TX from the bluetooth module to the TX of the Arduino... that's wrong and it won't work. Make sure you connect it properly, the TX into RX and the RX into the TX.

Note

If the HC-05 Bluetooth Module asks for a password, it's '1234'.

Code

Make sure you remove the wires from RX and TX otherwise the code won't upload properly!

[View code on GitHub](#)

```
/*
 * Control DC motor with Smartphone via bluetooth
 * created by Rui Santos, http://randomnerdtutorials.com
 */

int motorPin1 = 3; // pin 2 on L293D IC
int motorPin2 = 4; // pin 7 on L293D IC
int enablePin = 5; // pin 1 on L293D IC
int state;
int flag=0; //makes sure that the serial only prints once the state

void setup() {
    // sets the pins as outputs:
    pinMode(motorPin1, OUTPUT);
    pinMode(motorPin2, OUTPUT);
    pinMode(enablePin, OUTPUT);
    // sets enablePin high so that motor can turn on:
    digitalWrite(enablePin, HIGH);
    // initialize serial communication at 9600 bits per second:
    Serial.begin(9600);
}

void loop() {
    //if some date is sent, reads it and saves in state
    if(Serial.available() > 0){
        state = Serial.read();
        flag=0;
    }
    // if the state is '0' the DC motor will turn off
    if (state == '0') {
        digitalWrite(motorPin1, LOW); // set pin 2 on L293D low
        digitalWrite(motorPin2, LOW); // set pin 7 on L293D low
        if(flag == 0){
            Serial.println("Motor: off");
            flag=1;
        }
    }
    // if the state is '1' the motor will turn right
    else if (state == '1') {
        digitalWrite(motorPin1, LOW); // set pin 2 on L293D low
        digitalWrite(motorPin2, HIGH); // set pin 7 on L293D high
        if(flag == 0){
            Serial.println("Motor: right");
            flag=1;
        }
    }
    // if the state is '2' the motor will turn left
    else if (state == '2') {
        digitalWrite(motorPin1, HIGH); // set pin 2 on L293D high
        digitalWrite(motorPin2, LOW); // set pin 7 on L293D low
        if(flag == 0){
```

```
    Serial.println("Motor: left");
    flag=1;
}
}
```

For the android communication with the bluetooth module I've used the BlueTerm app. It's completely free, so you just need to go to "Play store" and download it. Then you connect your smartphone with the bluetooth module. Remember to remove the TX and RX cables. (you can see in the youtube video below how that's done).

I've only set 3 commands to control the DC motor:

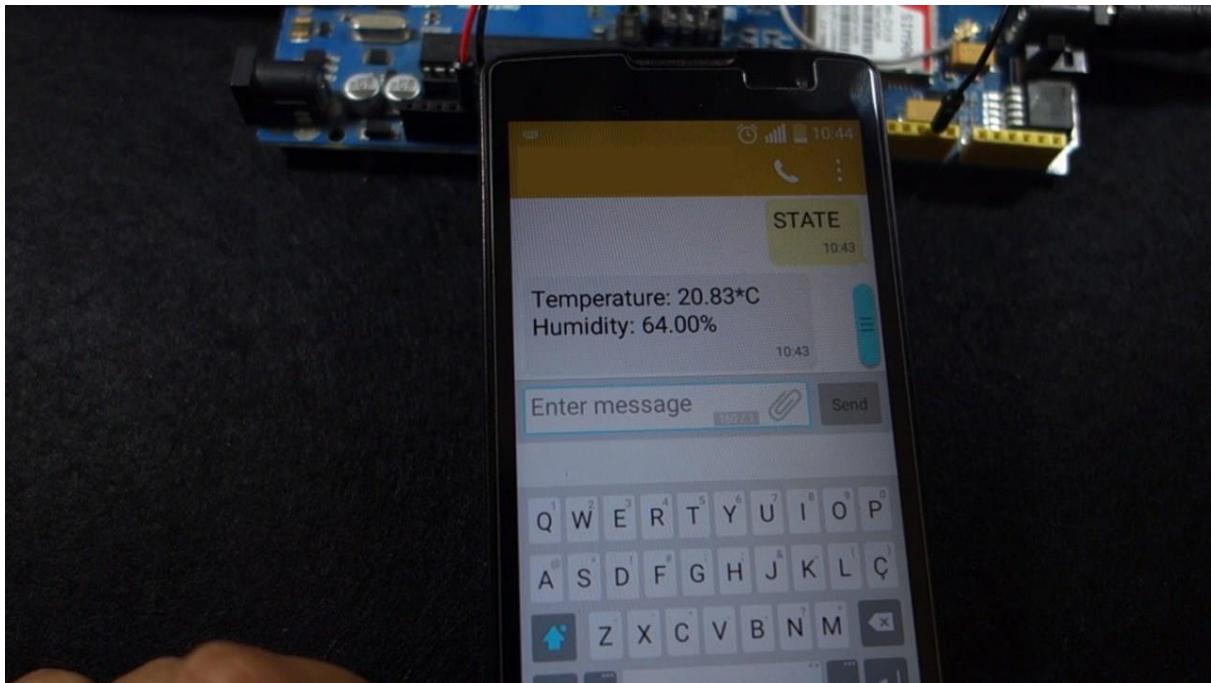
- '0' - Turns off the DC motor
- '1' - DC motor rotates to right
- '2' - DC motor rotates to left

Watch the video demonstration



Watch on YouTube: <http://youtu.be/nXymP6txD4>

Request Sensor Data via SMS



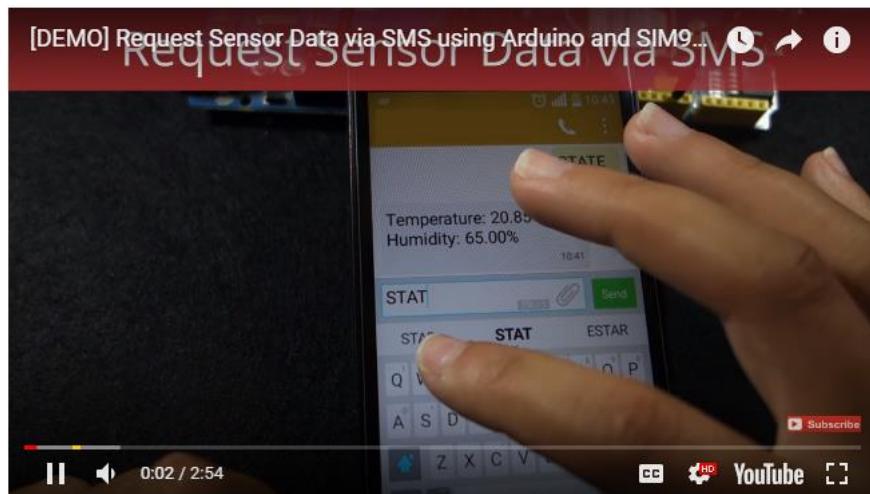
View Project on Random Nerd Tutorials	Click here
Watch on YouTube	Click here
View code on GitHub	Click here

In this project we're going to show you how to request sensor data via SMS with the Arduino. As an example we're going to request the temperature and humidity from a DHT11 sensor. To send and receive SMS with the Arduino we're going to use the SIM900 GSM shield. When you send an SMS to the Arduino with the message "STATE", it replies with the latest temperature and humidity readings.

Before proceeding with this tutorial we recommend the following resources:

- [Guide to SIM900 GSM GPRS Shield with Arduino](#)
- [Guide for DHT11/DHT22 Humidity and Temperature Sensor With Arduino](#)
- [Enroll in our free Arduino Mini Course](#)

Watch the video demonstration

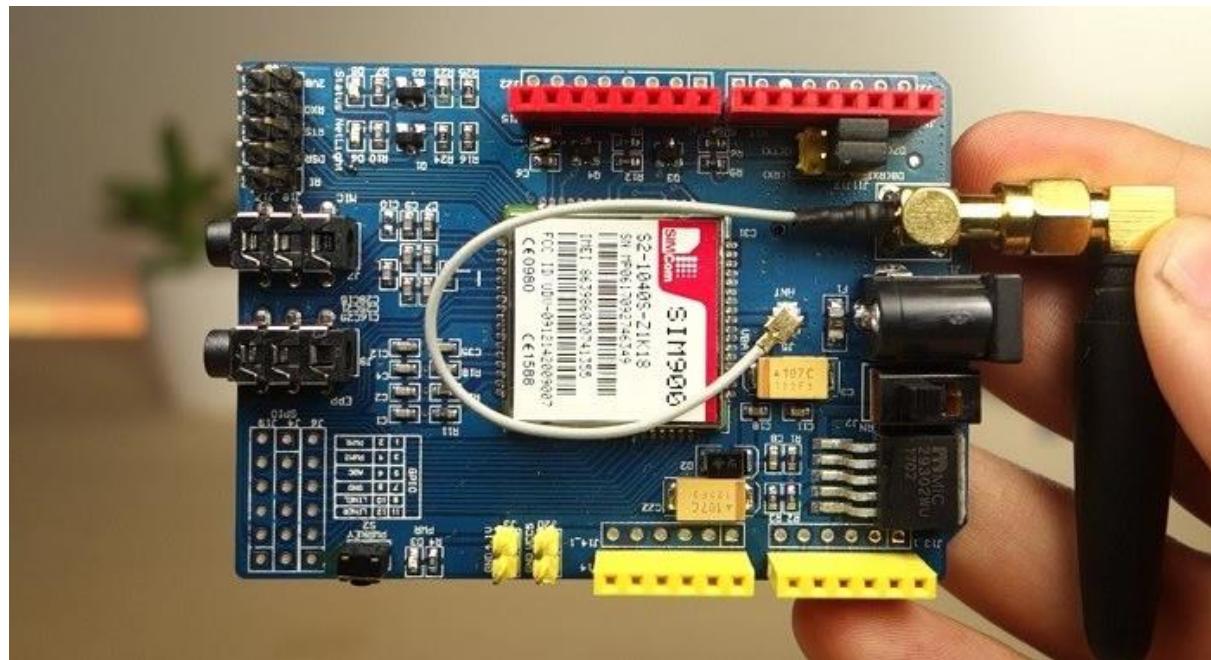


Watch on YouTube: <https://youtu.be/2Cz1GOvEbrw>

SIM900 GSM Shield

There are several modules you can use to send and receive SMS with the Arduino. We did this project using the SIM900 GSM shield and that's the shield we recommend you to get.

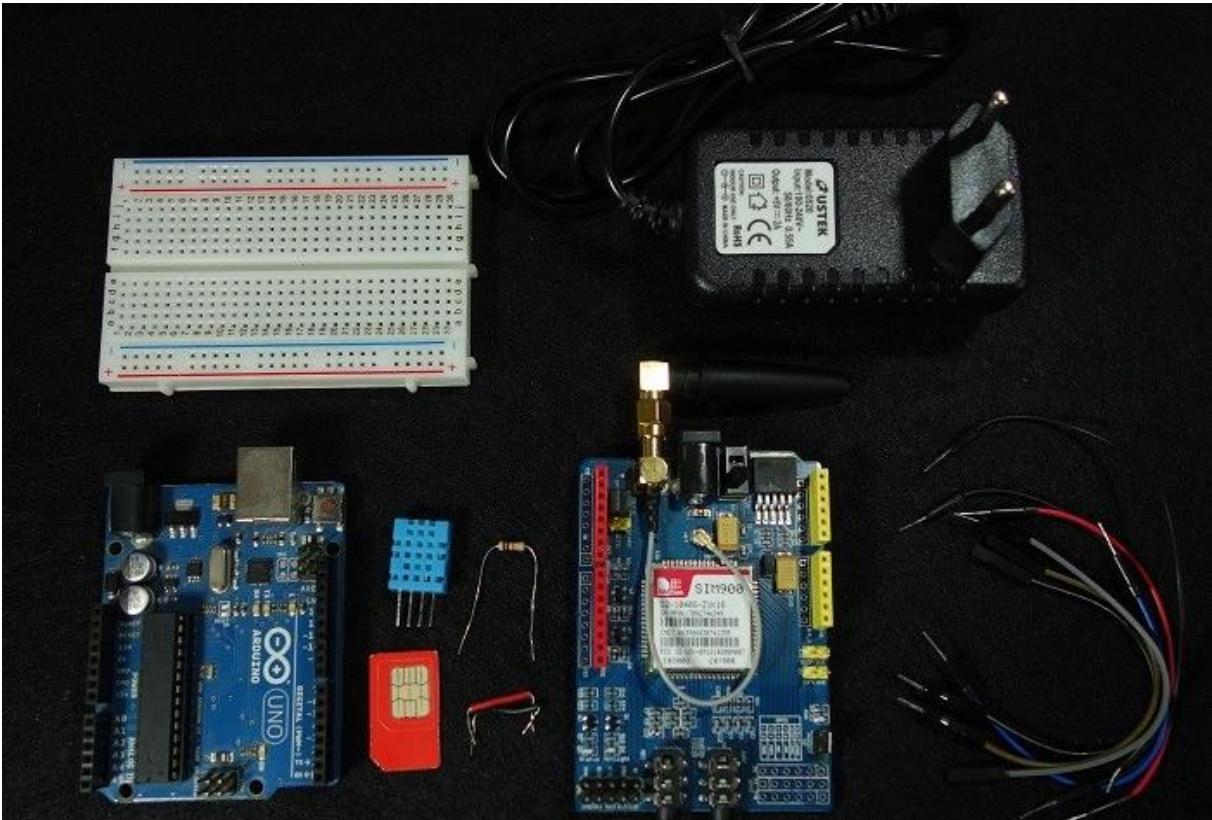
The SIM900 GSM shield is shown in figure below:



For an introduction to the GSM shield, and how to set it up, read the [Guide to SIM900 GSM GPRS Shield with Arduino](#).

Parts Required

Here's a list of all the components needed for this project:



- [Arduino UNO](#) – read [Best Arduino Starter Kits](#)
- [SIM900 GSM Shield](#)
- [5V 2A Power Adaptor](#)
- [FTDI programmer](#) (optional)
- SIM Card
- [DHT11](#) or [DHT22](#) Temperature and Humidity Sensor
- [10 kOhm resistor](#)
- [Breadboard](#)
- [Jumper Wires](#)

Preliminary Steps

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

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.

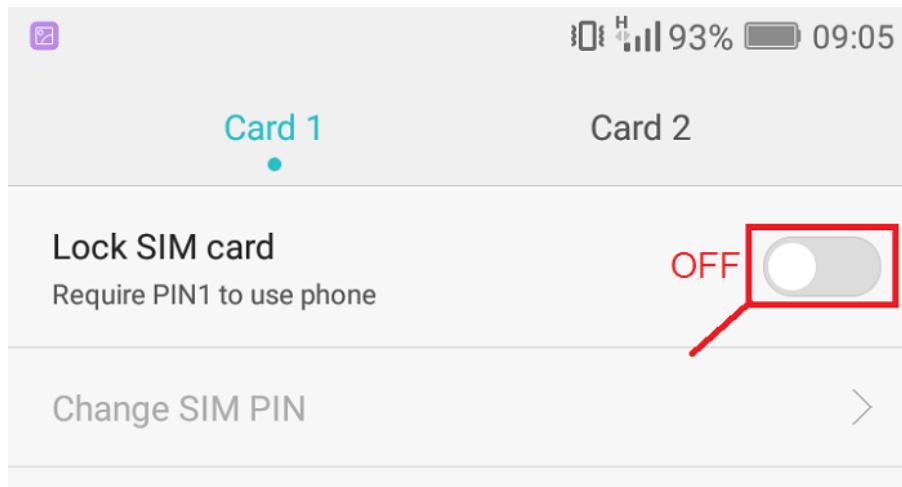


The shield uses the original SIM card size, not micro or nano. If you have micro or nano you may consider getting a [SIM card size adapter](#).

Turn off the PIN lock

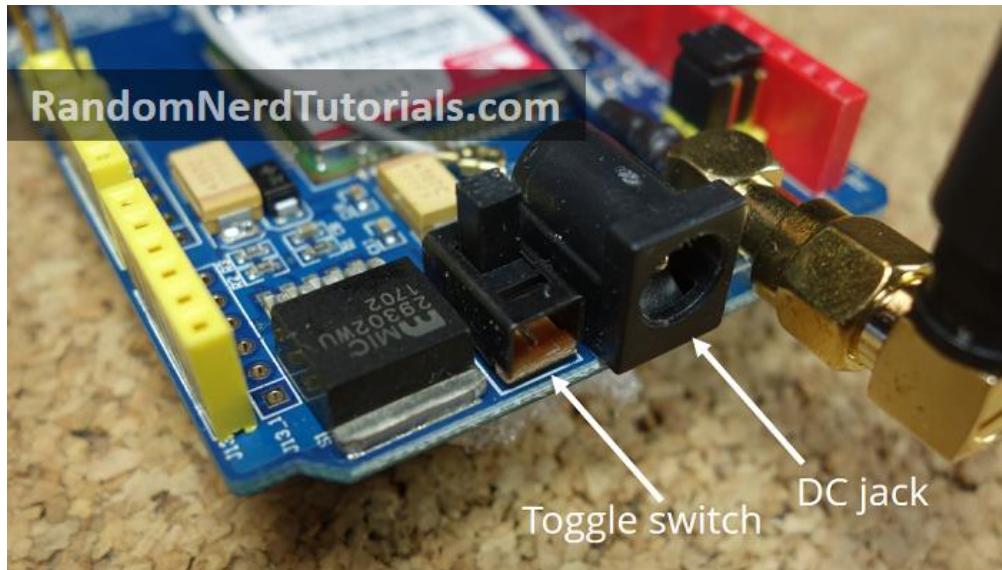
To use the SIM card with the shield, you need to turn off the pin lock. The easiest way to do this, is to insert the SIM card in your smartphone and turn off the pin lock in the phone security settings.

In my case, I need 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 jack 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.



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

Setting up the SIM900 GSM Shield

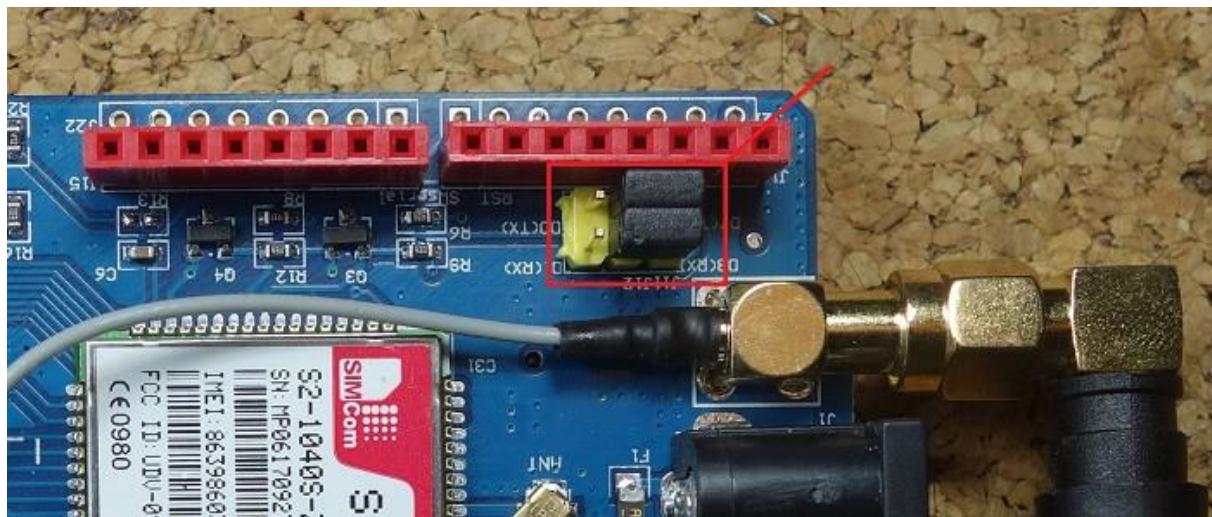
The following steps show you how to set up the SIM900 GSM shield.

1) Insert the SIM card into the SIM card holder. You need a SIM card with the standard size. The shield is not compatible with micro or nano SIM cards. If you need, you may get a sim card size adapter. Also, it is advisable to use a SIM card with a prepaid plan or unlimited SMS.

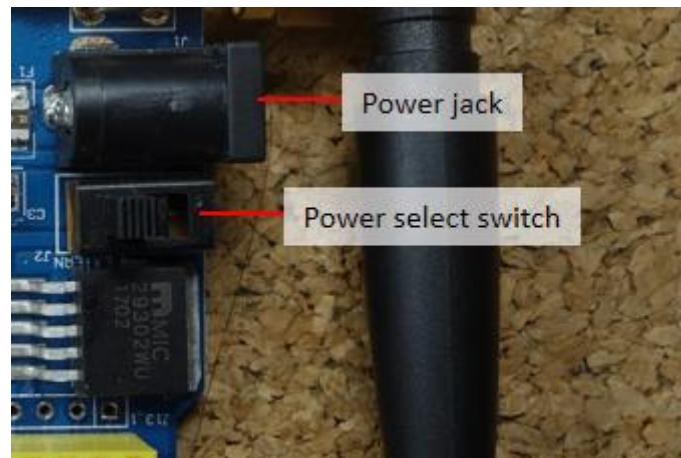


2) Check the antenna is well connected.

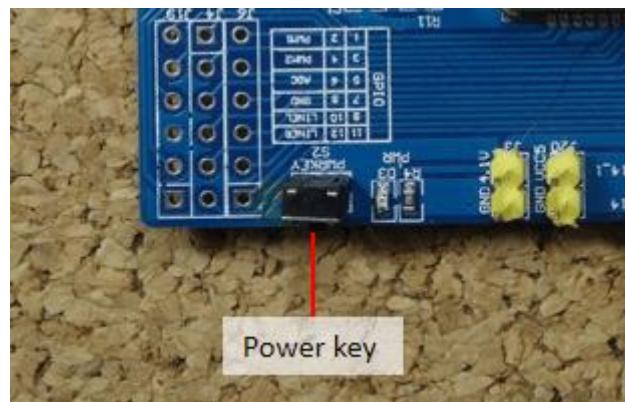
3) On the serial port select, make sure the jumper cap is connected as shown in figure below to use software serial.



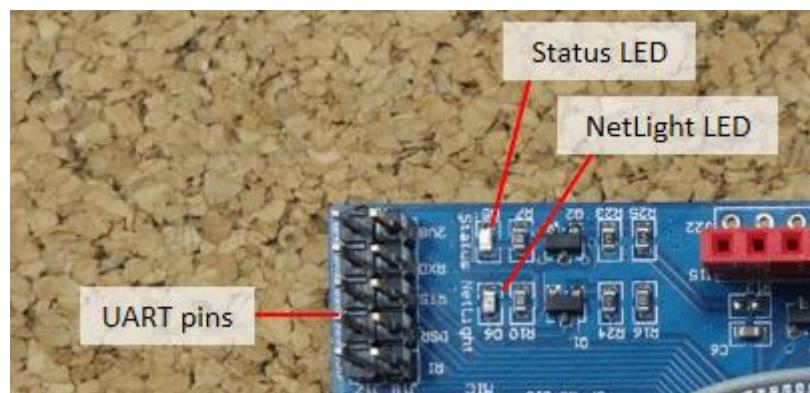
4) Power the shield using an external 5V power supply. Double-check that you have the external power source selected as we've mentioned earlier.



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



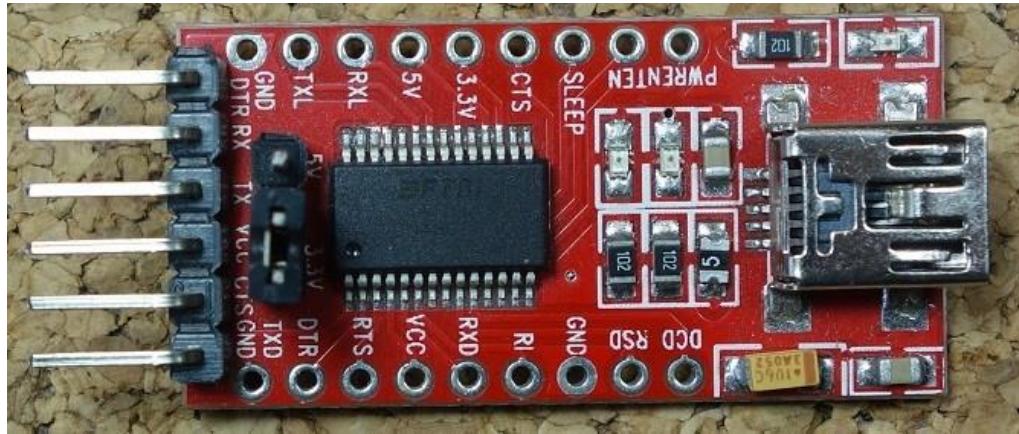
6) Then, the Status LED will light up and the NetLight LED will blink every 800 ms until it finds the network. When it finds the network the NetLight LED will start blinking every three seconds.



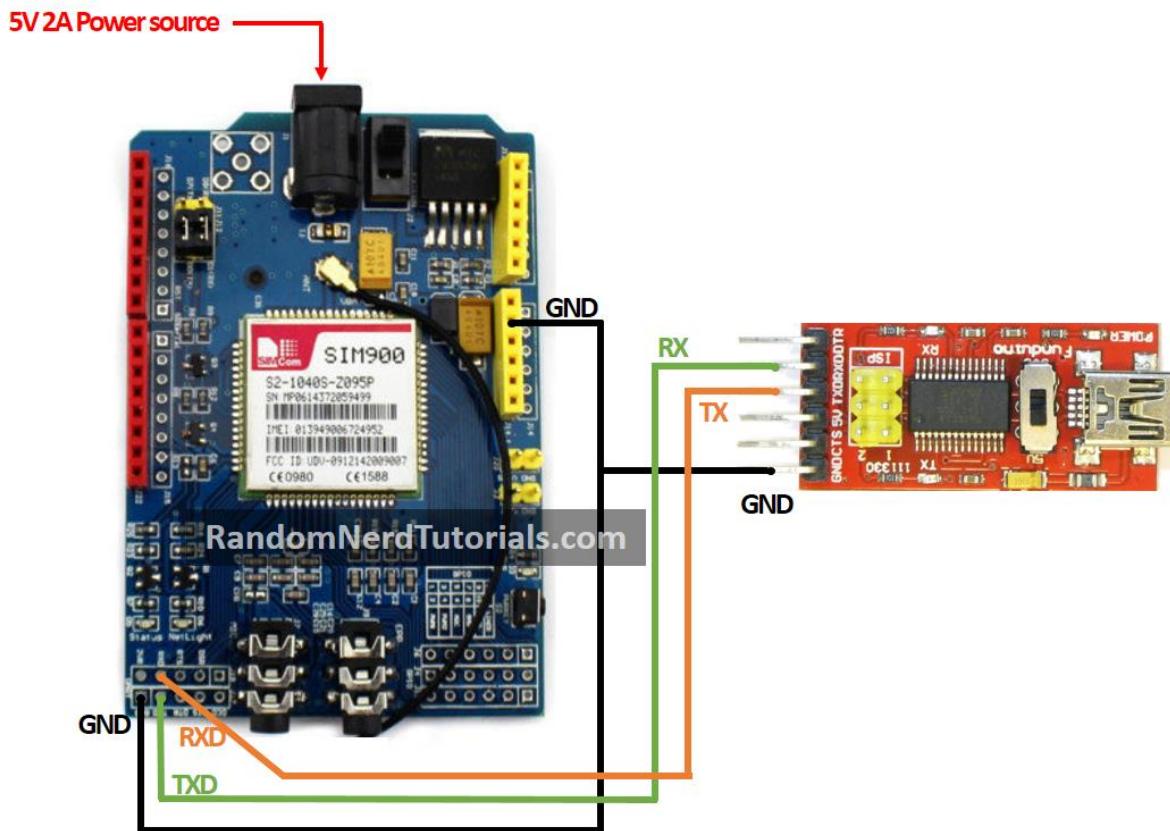
7) You can test if the shield is working properly by sending AT commands from the Arduino IDE using an FTDI programmer – as shown below.

Testing the Shield with FTDI Programmer

You don't need to do this step to get the shield working properly. This is an extra step to ensure that you can communicate with your GSM shield and send AT commands from the Arduino IDE serial monitor. For that, you need an FTDI programmer as the one shown in figure below.



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

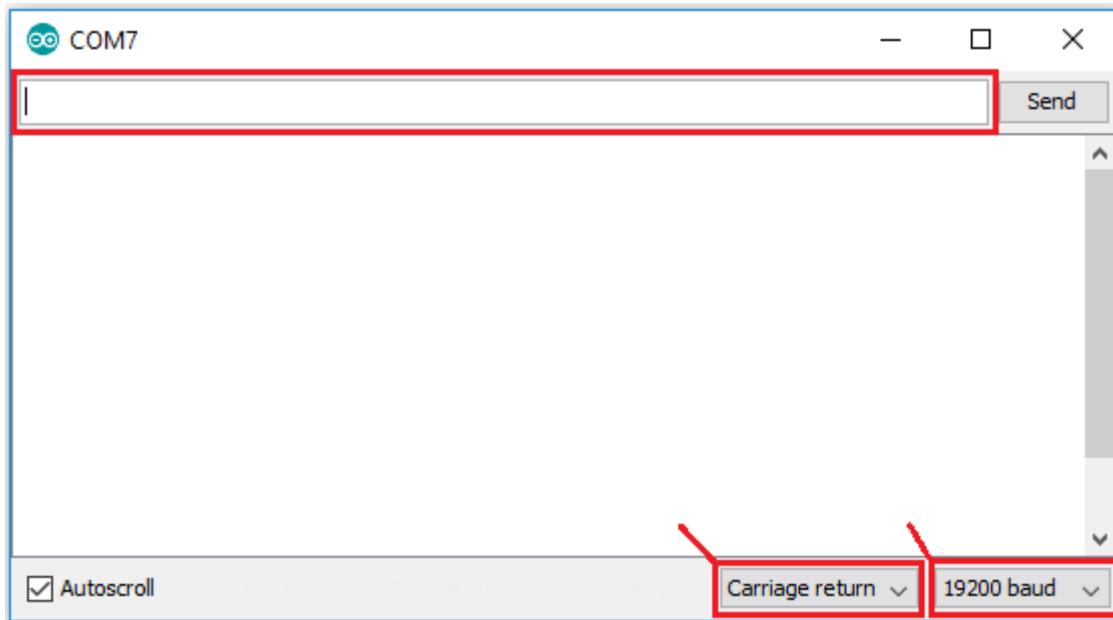


2) Open the Arduino IDE and select the right COM port.

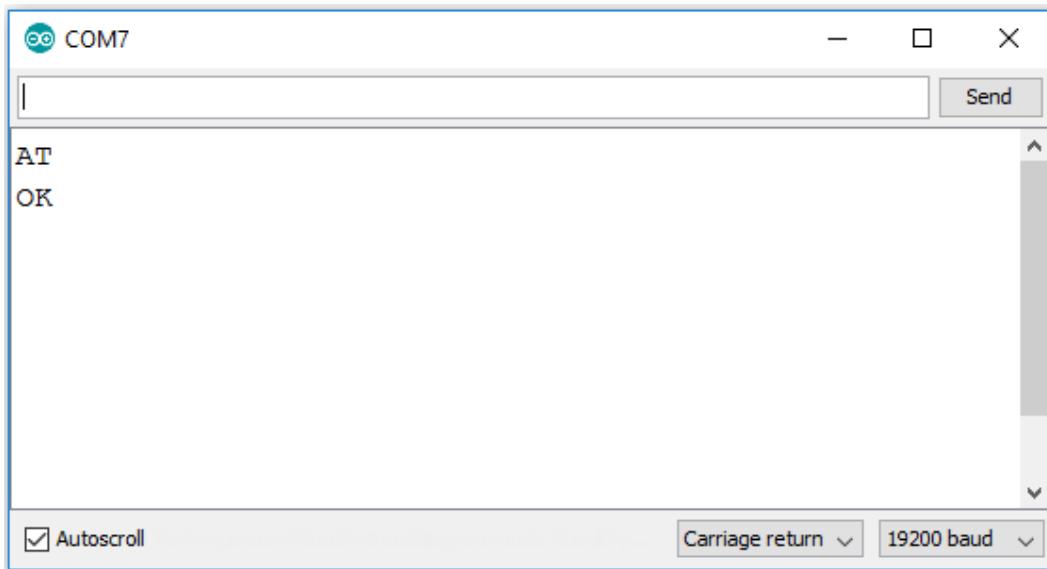
3) Open the Serial monitor.



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.



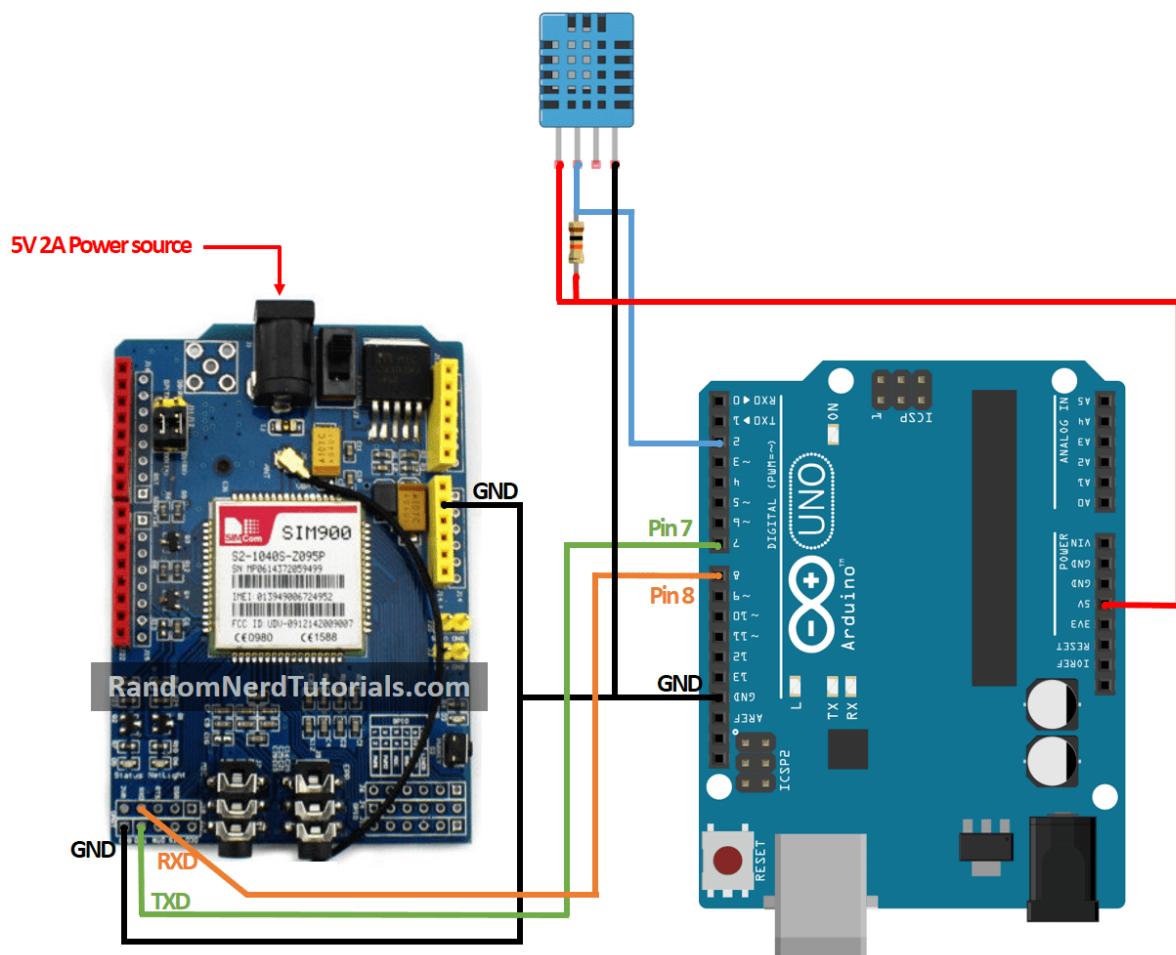
5) The shield will respond with OK, if everything is working properly.



Now that you know the shield is working properly, you are ready to start building the project.

Schematics

The figure below shows the circuit schematics for this project. You have to connect the SIM900 GSM shield and the DHT11 temperature and humidity sensor to the Arduino as shown in the figure below.



Installing the DHT library

To read from the DHT sensor, you must have the DHT library installed. If you don't have the DHT library installed, follow the instructions below:

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

Installing the Adafruit_Sensor library

To use the DHT temperature and humidity sensor, you also need to install the [Adafruit_Sensor library](#). Follow the next steps to install the library in your Arduino IDE:

6. [Click here to download](#) the Adafruit_Sensor library. You should have a .zip folder in your Downloads folder
7. Unzip the .zip folder and you should get **Adafruit_Sensor-master** folder
8. Rename your folder from **Adafruit_Sensor-master** to **Adafruit_Sensor**
9. Move the **Adafruit_Sensor** folder to your Arduino IDE installation libraries folder
10. Finally, re-open your Arduino IDE

Code

The following code reads the temperature and humidity from the DHT sensor and sends them via SMS when you send an SMS to the Arduino with the message "STATE".

You need to modify the code provided with the phone number your Arduino should reply the readings to. The code is well commented for you to understand the purpose of each line of code.

Don't upload the code now. Scroll down and read the explanation below the code.

[View code on GitHub](#)

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

// Include DHT library and Adafruit Sensor Library
#include "DHT.h"
#include <Adafruit_Sensor.h>
//Include Software Serial library to communicate with GSM
#include <SoftwareSerial.h>

// Pin DHT is connected to
#define DHTPIN 2

// Uncomment whatever type of sensor you're using
#define DHTTYPE DHT11    // DHT 11
```

```

//#define DHTTYPE DHT22      // DHT 22  (AM2302)
//#define DHTTYPE DHT21      // DHT 21  (AM2301)

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

// Create global variables to store temperature and humidity
float t; // temperature in celcius
float f; // temperature in fahrenheit
float h; // humidity

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

// Create variable to store incoming SMS characters
char incomingChar;

void setup() {
  dht.begin();

  Serial.begin(19200);
  SIM900.begin(19200);

  // Give time to your GSM shield log on to network
  delay(20000);
  Serial.print("SIM900 ready...");

  // 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(){
  if (SMSRequest()){
    if(readData()){
      delay(10);
      // REPLACE THE X's WITH THE RECIPIENT'S MOBILE NUMBER
      // USE INTERNATIONAL FORMAT CODE FOR MOBILE NUMBERS
      SIM900.println("AT + CMGS = \"+XXXXXXXXXX\"");
      delay(100);
      // REPLACE WITH YOUR OWN SMS MESSAGE CONTENT
      String dataMessage = ("Temperature: " + String(t) + "*C " + " Humidity: "
      " + String(h) + "%");
      // Uncomment to change message with farenheit temperature
      // String dataMessage = ("Temperature: " + String(f) + "*F " + "
      Humidity: " + String(h) + "%");

      // Send the SMS text message
      SIM900.print(dataMessage);
    }
  }
}

```

```

    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);
}
}

delay(10);
}

boolean readData() {
    //Read humidity
    h = dht.readHumidity();
    // Read temperature as Celsius
    t = dht.readTemperature();
    // Read temperature as Fahrenheit
    f = dht.readTemperature(true);

    // Compute temperature values in Celcius
    t = dht.computeHeatIndex(t,h,false);

    // Uncomment to compute temperature values in Fahrenheit
    //f = dht.computeHeatIndex(f,h,false);

    // 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 false;
    }
    Serial.print("Humidity: ");
    Serial.print(h);
    Serial.print(" %\t");
    Serial.print("Temperature: ");
    Serial.print(t);
    Serial.print(" *C ");
    //Uncomment to print temperature in Farenheit
    //Serial.print(f);
    //Serial.print(" *F\t");
    return true;
}

boolean SMSRequest() {
    if(SIM900.available() >0) {
        incomingChar=SIM900.read();
        if(incomingChar=='S') {
            delay(10);
            Serial.print(incomingChar);
            incomingChar=SIM900.read();
            if(incomingChar =='T') {
                delay(10);

```

```

    Serial.print(incomingChar);
    incomingChar=SIM900.read();
    if(incomingChar=='A') {
        delay(10);
        Serial.print(incomingChar);
        incomingChar=SIM900.read();
        if(incomingChar=='T') {
            delay(10);
            Serial.print(incomingChar);
            incomingChar=SIM900.read();
            if(incomingChar=='E') {
                delay(10);
                Serial.print(incomingChar);
                Serial.print("...Request Received \n");
                return true;
            }
        }
    }
}
return false;
}

```

Importing libraries

First, you include the libraries needed for this project: the DHT libary to read from the DHT sensor and the SoftwareSerial library to communicate with the SIM900 GSM module.

```

#include "DHT.h"
#include <Adafruit_Sensor.h>
#include <SoftwareSerial.h>

```

DHT sensor

Then, you tell the Arduino that the DHT data pin is connected to pin 2, select the DHT sensor type and create a dht instance. The code is compatible with other DHT sensors as long as you define the one you're using in the code.

```

#define DHTPIN 2
#define DHTTYPE DHT11
DHT dht(DHTPIN, DHTTYPE);

```

You also create float variables to store the temperature and humidity values.

```

float t; // temperature in celcius
float f; // temperature in fahrenheit

```

```
float h; // humidity
```

GSM shield

The following line configures the software serial on pins 7 and 8. Pin 7 is being configured as RX and pin 8 as TX.

```
SoftwareSerial SIM900(7, 8);
```

You also create a char variable to store the incoming SMS characters.

```
char incomingChar;
```

setup()

In the setup(), you begin the DHT and the SIM900 shield. The SIM900 shield is set to text mode and you also set the module to send the SMS data to the serial monitor when it receives it. This is done with the following two lines, respectively:

```
SIM900.print("AT+CMGF=1\r");  
SIM900.print("AT+CNMI=2,2,0,0,0\r");
```

Functions

We create a function to read the temperature and humidity called **readData()**. This function stores the values on the t and h variables. The code uses temperature in Celsius, but it is prepared if you want Fahrenheit instead – the code is commented on where you should make the changes.

We also create a function that checks if the incoming message is equal to STATE – the **SMSRequest()** function. This function returns true if the Arduino receives a message with the text STATE and false if not. You read the SMS incoming characters using:

```
incomingChar = SIM900.read();
```

loop()

In the loop(), you check if there was an SMS request with the **SMSRequest()** function – you check if the Arduino received a STATE message. If true, it will read the temperature and humidity and send it via SMS to you.

The number the Arduino answers to is set at the following line:

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

Replace the **XXXXXXXXXXXX** with the recipient's phone number.

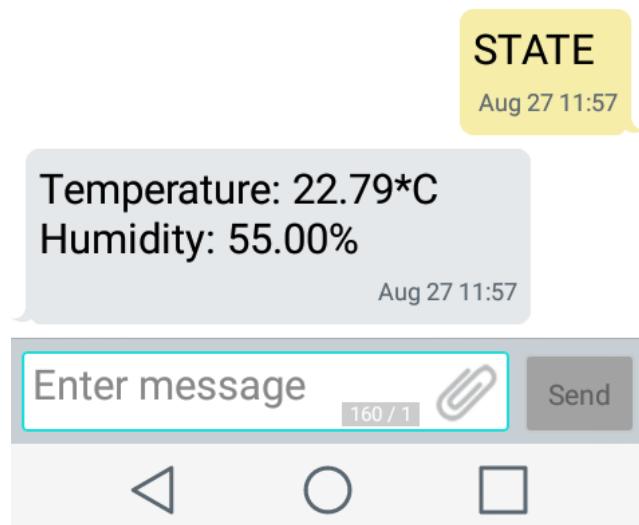
Note: you must add the number according to the international phone number format. For example, in Portugal the number is preceded by **+351XXXXXXXXX**.

Then, you store the message you want to send in the dataMessage variable. Finally you send the SMS text message using:

```
SIM900.print(dataMessage);
```

Demonstration

When you send the STATE message to the Arduino, it replies with the sensor data.



Watch the video at the beginning of the project for a more in depth project demo.

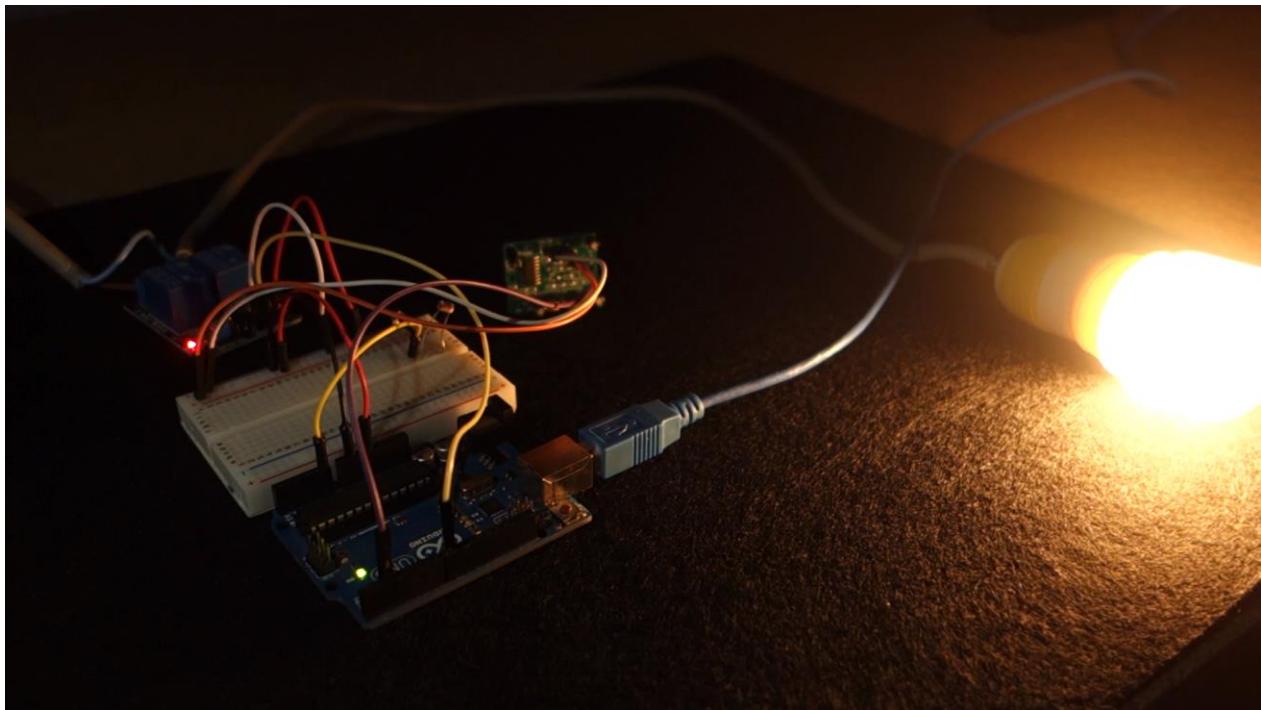
Wrapping Up

This is a great project to get you started with the SIM900 GSM shield. You've learned how to read and send SMS text messages with the Arduino.

You can apply the concepts learned in pretty much any project. Here's some ideas of projects:

- Surveillance system that sends an SMS when it detects movement
- [Control a relay via SMS](#)
- Request specific sensor data from a collection of sensors by adding more conditions to the code

Night Security Light with Arduino



View Project on Random Nerd Tutorials

Click [here](#)

View code on GitHub

Click [here](#)

Introduction

In this project you're going to build a night security light with a relay module, a photoresistor and an Arduino.

A night security light only turns on when it's dark and when movement is detected.

Here's the main features of this project:

- the lamp turns on when it's dark AND movement is detected;
- when movement is detected the lamp stays on for 10 seconds;
- when the lamp is ON and detects movement, it starts counting 10 seconds again;
- when there's light, the lamp is turned off, even when motion is detected.

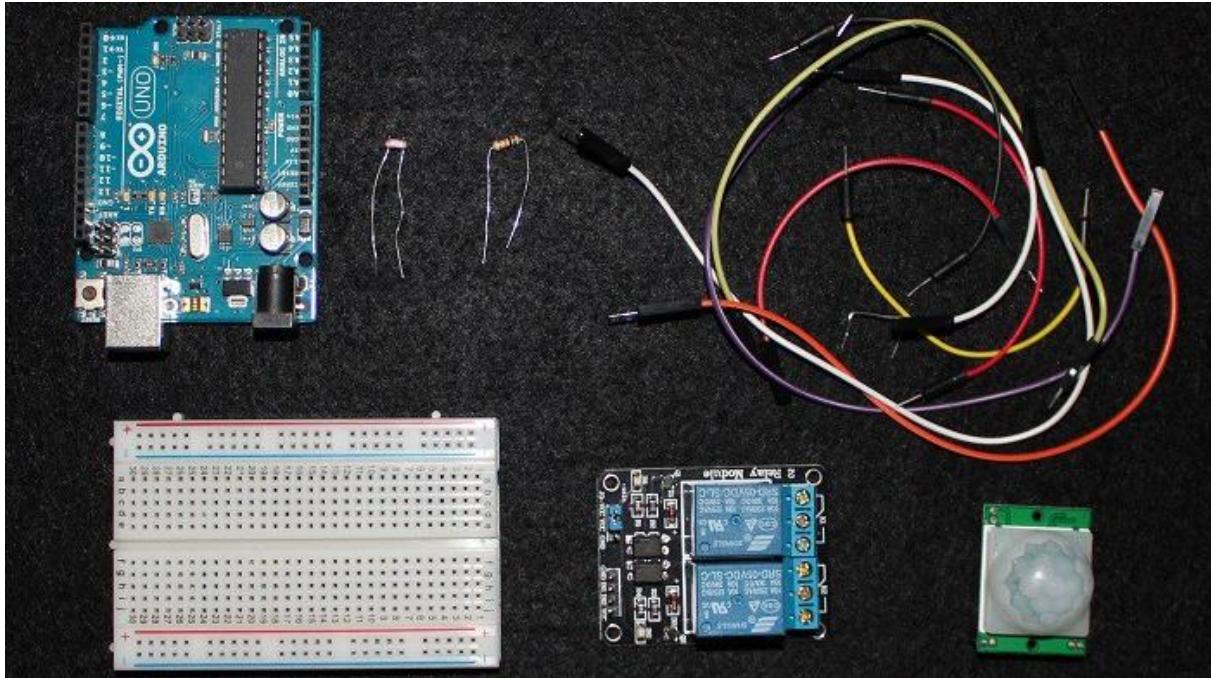
Recommended resources

The following resources include guides on how to use the relay module and the PIR motion sensor with the Arduino, which might be useful for this project.

- [Guide for Relay Module with Arduino](#)
- [Arduino with PIR Motion Sensor](#)

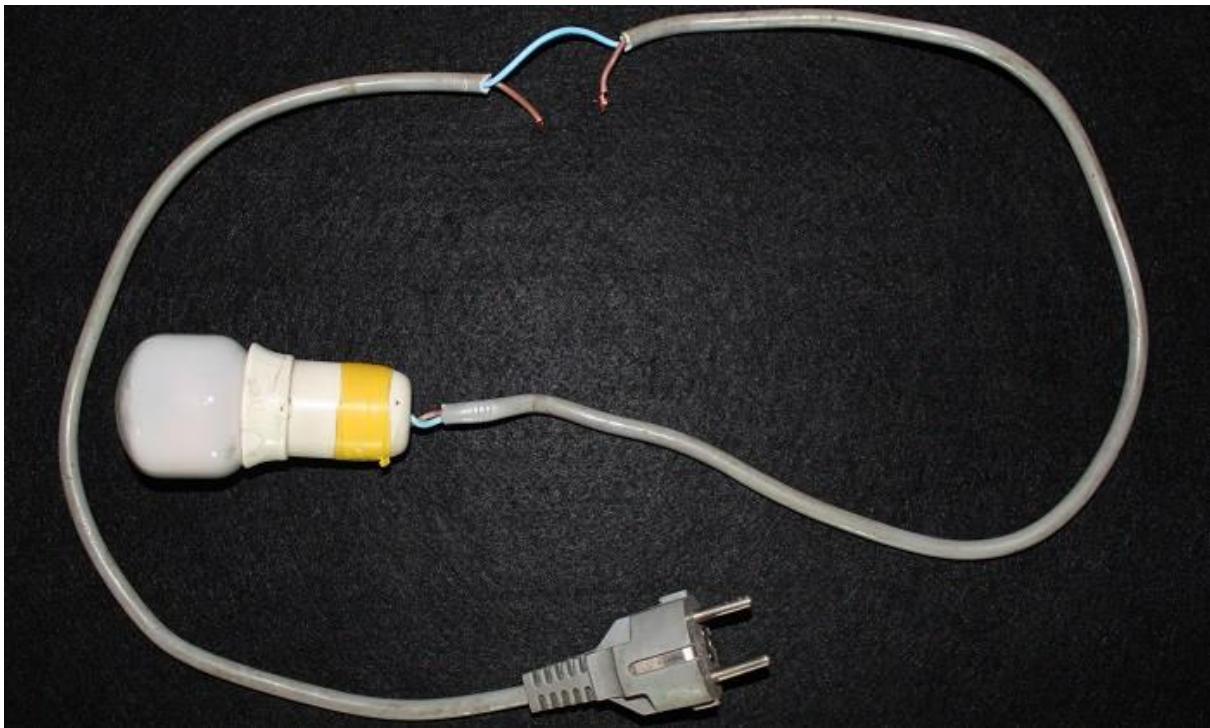
Parts Required

Here's a complete list of the parts required for this project:



- [Arduino UNO](#) – read [Best Arduino Starter Kits](#)
- [PIR Motion Sensor](#)
- [Photoresistor](#)
- [10kOhm resistor](#)
- [Relay module](#)
- Lamp cord set ([view on eBay](#))
- [Breadboard](#)
- [Jumper wires](#)

Besides these electronics components, you also need an AC male socket, an AC wire and a lamp bulb holder (a lamp cord set) as shown in the figure below.



Code

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

Warning: do not upload a new code to your Arduino board while your lamp is connected to the mains voltage. You should unplug the lamp from mains voltage, before upload a new sketch to your Arduino.

[View code on GitHub](#)

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

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

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

// LDR pin is connected to Analog 0
int LDRPin = A0;
// LDR value is stored on LDR reading
```

```

int LDRReading;
// LDR Threshold value
int LDRThreshold = 300;

// 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");
    LDRReading = analogRead(LDRPin);
    // LDR Reading value is printed on serial monitor, useful to get your
    LDRThreshold
    //Serial.println(LDRReading);
    // Only turns the Relay on if the LDR reading is higher than the
    LDRThreshold
    if(LDRReading > LDRThreshold) {
        if(relayState == LOW) {
            digitalWrite(relay, LOW);
        }
        relayState = HIGH;
    }
}

```

```
Serial.println("ON");

lastDebounceTime = millis();

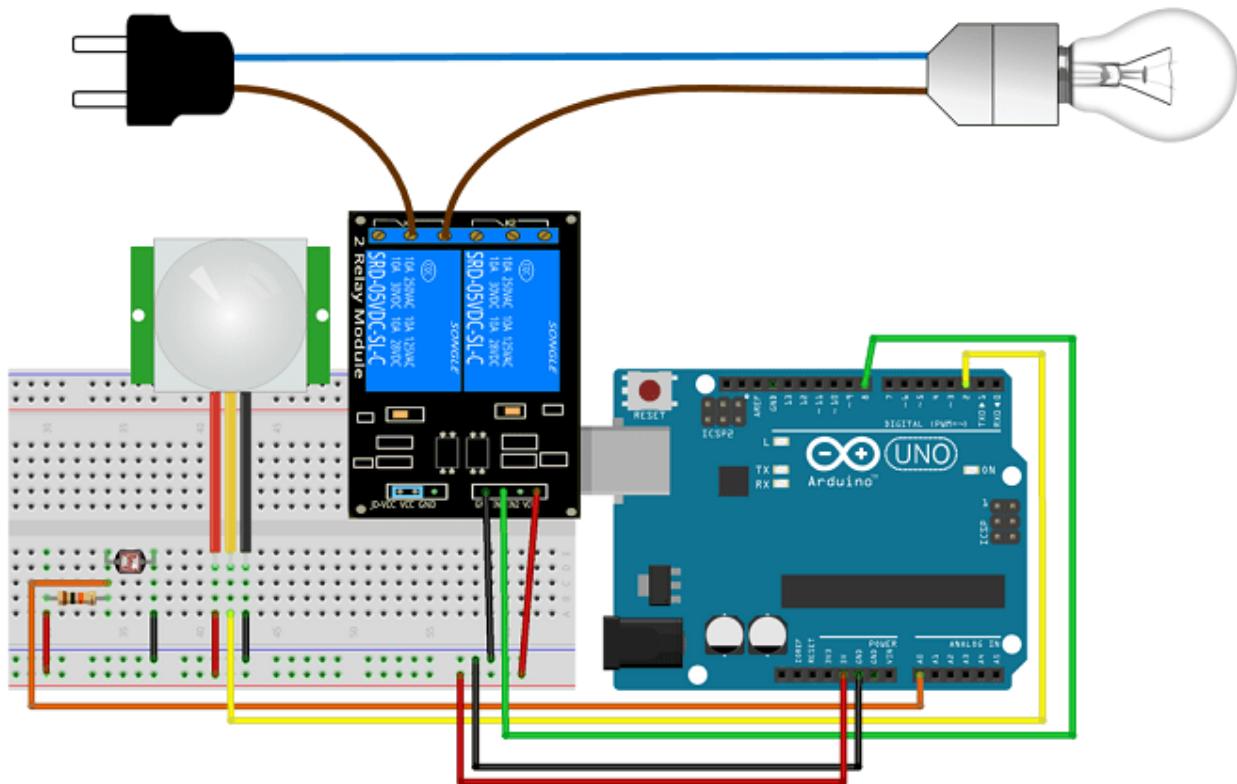
}

}
```

Schematics

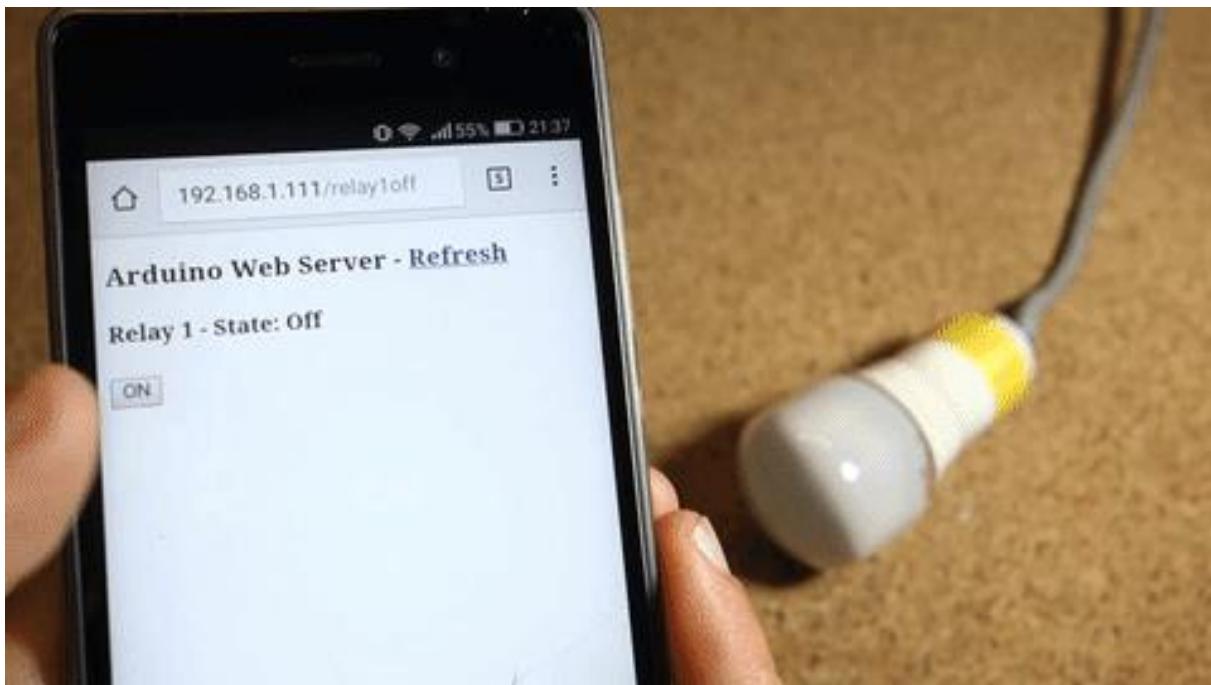


Here's the schematics for this project.



Note: if you have an earth (GND) connection in the mains voltage cable – a yellow and green cable – it should go outside the relay module, like the blue wire (neutral).

Ethernet Web Server with Relay



View Project on Random Nerd Tutorials

Click [here](#)

View code on GitHub

Click [here](#)

Introduction

This project shows how to build an Arduino Ethernet web server that controls a relay that is attached to a lamp. You can access your web server with any device that has a browser and it's connected to the same network.

Recommended resources:

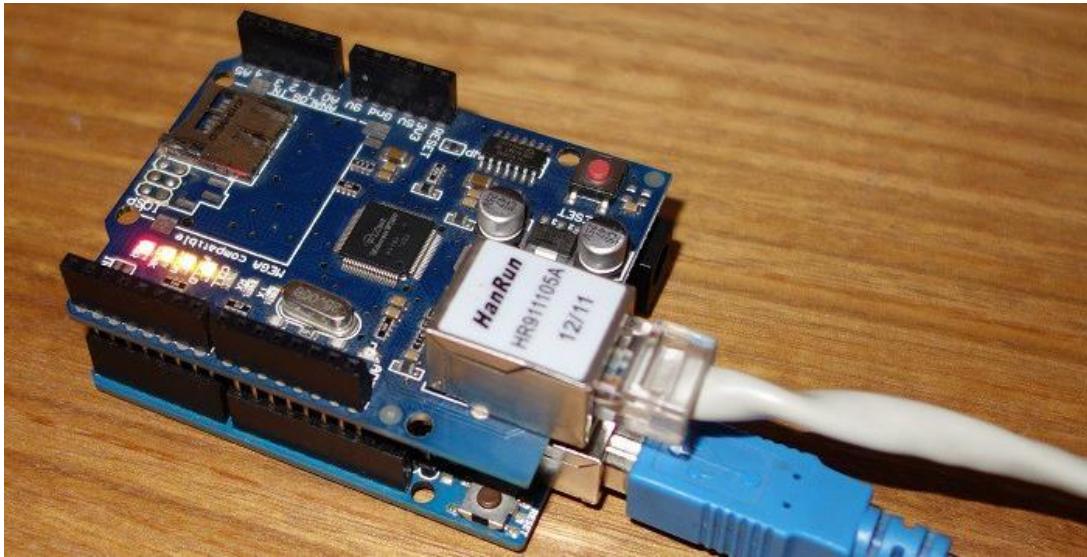
- [Guide for Relay Module with Arduino](#)
- [Arduino – Webserver with an Arduino + Ethernet Shield](#)
- [ESP8266 Web Server with Arduino IDE](#)
- [ESP8266 Web Server Tutorial](#)

Note: if you're not comfortable dealing with mains voltage, you can still try this project by replacing the relay module with an LED, for example.

Ethernet Shield

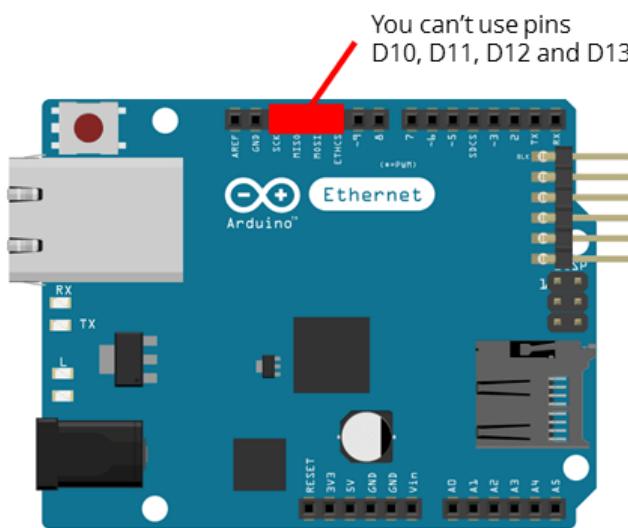
The Arduino Ethernet shield connects your Arduino to the internet in a simple way. Just mount this module onto your Arduino board, connect it to your network with an RJ45 cable and follow a few simple steps to start controlling your projects through the web.

Note: you must connect an Ethernet cable from your router to your Ethernet shield as shown in the following figure.



Pin usage

When the Arduino is connected to an Ethernet shield, you can't use Digital pins from 10 to 13, because they are being used in order to establish a communication between the Arduino and the Ethernet shield.



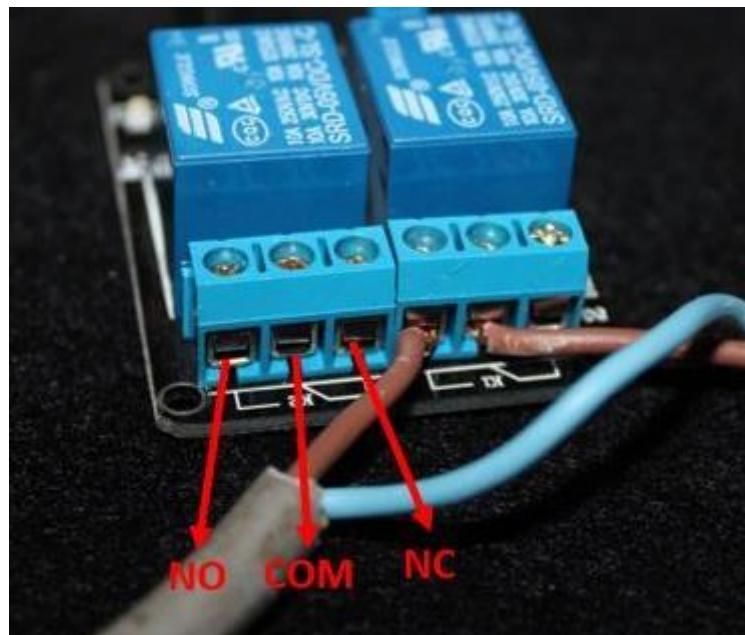
Relay Module

A [relay](#) is an electrically operated switch. It means that it can be turned on or off, letting the current going through or not. The relay module is shown in the figure below.



This particular relay module comes with two relays (those blue cubes).

About 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 lamp, in our case).
- **NC:** normally closed – there is contact between the common pin and the normally closed pin. There is always contact 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 power provided to the load.

Relating this project, it is better to use a normally open circuit, because we want to light up the lamp occasionally. [Read this tutorial](#) to learn more about using a relay module with the Arduino board.

The connections between the relay and the Arduino are really 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
- **VCC:** goes to 5V

Parts Required

- Here's a complete list of the components you need for this project:
- [Arduino UNO](#) – read [Best Arduino Starter Kits](#)
- [Ethernet Shield](#)
- [Relay module](#)
- Lamp cord set ([view on eBay](#))
- [Breadboard](#)
- [Jumper Wires](#)

Code

Copy the following code to your Arduino IDE and before uploading it to your Arduino board read the “Configuring your network” section below.

[View code on GitHub](#)

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

#include <SPI.h>
#include <Ethernet.h>

// Enter a MAC address and IP address for your controller below.
// The IP address will be dependent on your local network:
byte mac[] = {
  0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };
IPAddress ip(192,168,1, 111);

// Initialize the Ethernet server library
// with the IP address and port you want to use
// (port 80 is default for HTTP):
EthernetServer server(80);

// Relay state and pin
String relay1State = "Off";
const int relay = 7;

// Client variables
char linebuf[80];
int charcount=0;

void setup() {
  // Relay module prepared
  pinMode(relay, OUTPUT);
  digitalWrite(relay, HIGH);

  // Open serial communication at a baud rate of 9600
  Serial.begin(9600);

  // start the Ethernet connection and the server:
```

```

Ethernet.begin(mac, ip);
server.begin();
Serial.print("server is at ");
Serial.println(Ethernet.localIP());
}

// Display dashboard page with on/off button for relay
// It also print Temperature in C and F
void dashboardPage(EthernetClient &client) {
    client.println("<!DOCTYPE HTML><html><head>");
    client.println("<meta name=\"viewport\" content=\"width=device-width,
initial-
scale=1\"></head><body>");

    client.println("<h3>Arduino Web Server - <a href=\"/\">Refresh</a></h3>");
    // Generates buttons to control the relay
    client.println("<h4>Relay 1 - State: " + relay1State + "</h4>");
    // If relay is off, it shows the button to turn the output on
    if(relay1State == "Off"){
        client.println("<a href=\"/relay1on\"><button>ON</button></a>");
    }
    // If relay is on, it shows the button to turn the output off
    else if(relay1State == "On"){
        client.println("<a
href=\"/relay1off\"><button>OFF</button></a>");
    }
    client.println("</body></html>");
}

void loop() {
    // listen for incoming clients
    EthernetClient client = server.available();
    if (client) {
        Serial.println("new client");
        memset(linebuf, 0, sizeof(linebuf));
        charcount=0;
        // an http request ends with a blank line
        boolean currentLineIsBlank = true;
        while (client.connected()) {
            if (client.available()) {
                char c = client.read();
                //read char by char HTTP request

```

```

linebuf[charcount]=c;
if (charcount<sizeof(linebuf)-1) charcount++;
// if you've gotten to the end of the line (received a newline
// character) and the line is blank, the http request has ended,
// so you can send a reply
if (c == '\n' && currentLineIsBlank) {
    dashboardPage(client);
    break;
}
if (c == '\n') {
    if (strstr(linebuf,"GET /relayloff") > 0){
        digitalWrite(relay, HIGH);
        relay1State = "Off";
    }
    else if (strstr(linebuf,"GET /relayon") > 0){
        digitalWrite(relay, LOW);
        relay1State = "On";
    }
    // you're starting a new line
    currentLineIsBlank = true;
    memset(linebuf,0,sizeof(linebuf));
    charcount=0;
}
else if (c != '\r') {
    // you've gotten a character on the current line
    currentLineIsBlank = false;
}
}
// give the web browser time to receive the data
delay(1);
// close the connection:
client.stop();
Serial.println("client disconnected");
}
}

```

Configuring your Network

Take a look at the configuring your network code snippet:

```

byte mac[] = {
0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };

```

```
IPAddress ip(192,168,1,xxx);
```

Important: you actually might need to replace that variable highlighted in red with appropriate values that are suitable for your network, otherwise your Arduino will not establish a connection with your network.

Replace the following line with an IP that is available and suitable for your network:

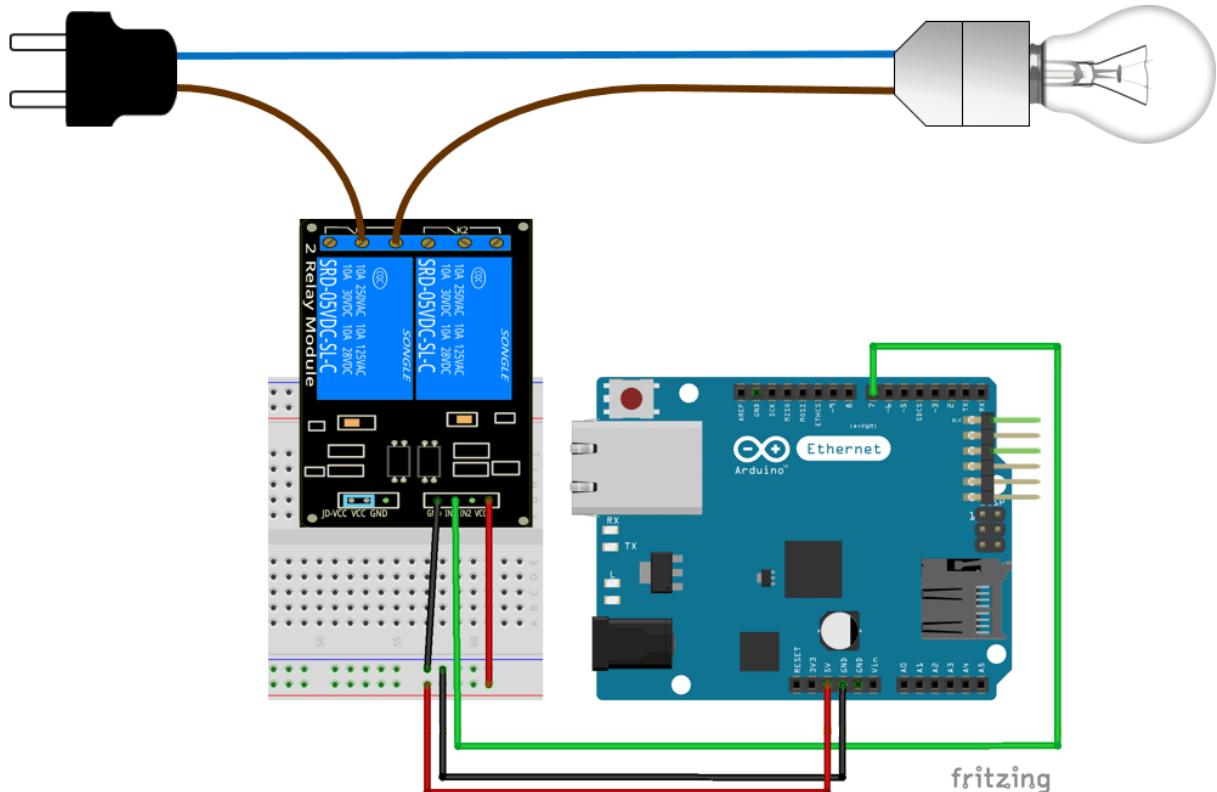
```
IPAddress ip(x, x, x, x);
```

In my case, my IP range is 192.168.1.X and with the software [Angry IP Scanner](#) I know that the IP 192.168.1.111 is available in my network, because it doesn't have any active device in my network with that exact same IP address:

```
IPAddress ip(192, 168, 1, 111);
```

Schematics

Wire your circuit accordingly to the schematic below:

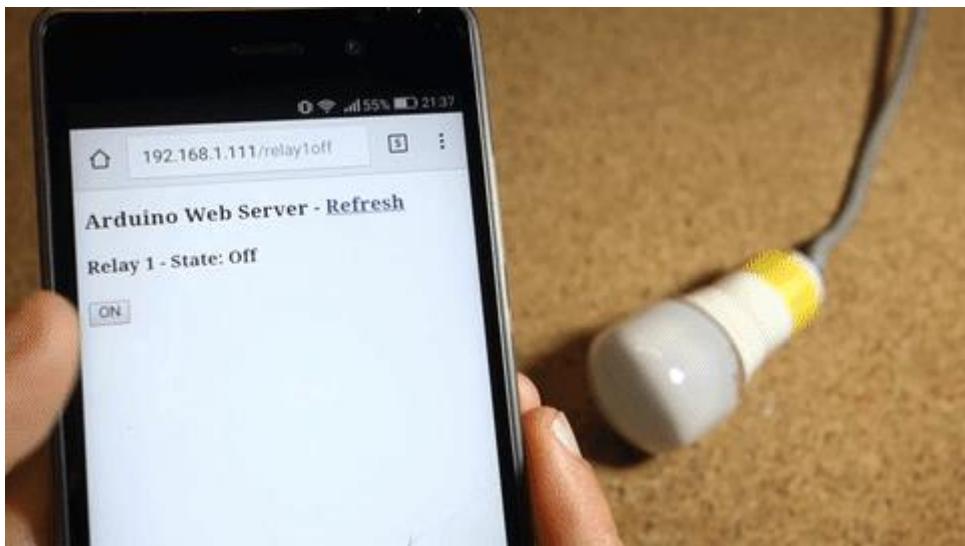


Demonstration

Your Arduino Web Server looks like the figure below:



Here's a demonstration showing what you have at the end of this project:



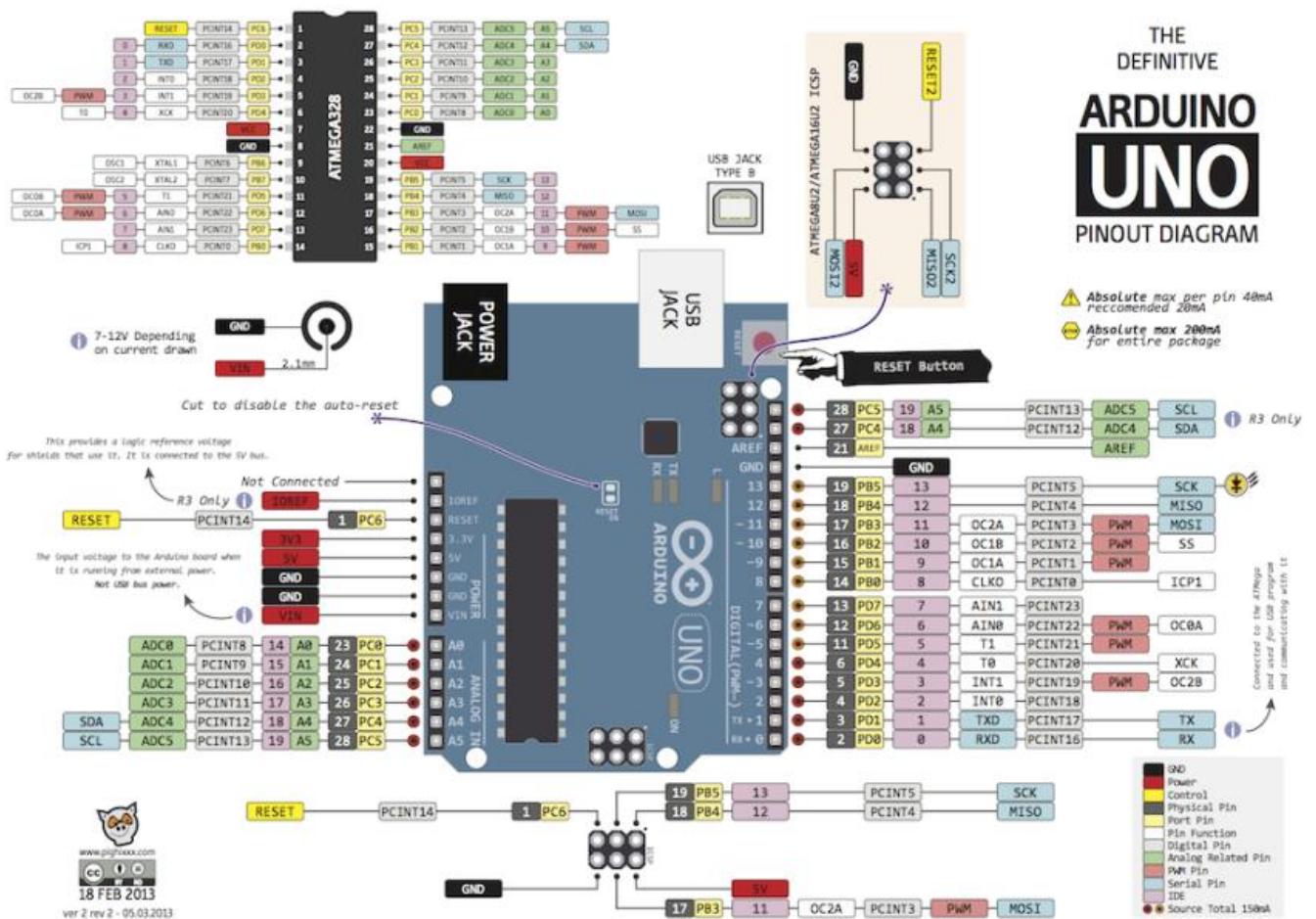
Wrapping Up

With this project you've built an Arduino web server that turns a relay on and off. Now, you can use this project to control any electronics appliance you want.

This project is a simplified version of one of our projects in the [Arduino Step-by-step Projects](#). If you liked this project make sure you check out the [course](#) that includes 25 Arduino projects.

Resources

Arduino Pinout



Resistor Color Code

The diagram illustrates two types of resistors and their color coding:

- 4-Band-Code:** A resistor with four color bands. The first three bands represent the resistance value (in ohms), and the fourth band represents the tolerance. The value is calculated as $(1st\ band \times 10^{2nd\ band}) \times 10^{3rd\ band}$.
- 5-Band-Code:** A resistor with five color bands. The first four bands represent the resistance value, and the fifth band represents the tolerance.

Reference Table:

COLOR	1 ST BAND	2 ND BAND	3 RD BAND	MULTIPLIER	TOLERANCE
Black	0	0	0	1Ω	
Brown	1	1	1	10Ω	± 1% (F)
Red	2	2	2	100Ω	± 2% (G)
Orange	3	3	3	1KΩ	
Yellow	4	4	4	10KΩ	
Green	5	5	5	100KΩ	± 0.5% (D)
Blue	6	6	6	1MΩ	± 0.25% (C)
Violet	7	7	7	10MΩ	± 0.10% (B)
Grey	8	8	8		± 0.05%
White	9	9	9		
Gold				0.1Ω	± 5% (J)
Silver				0.01Ω	± 10% (K)

Wrapping Up

If you've made it to this point, thank you so much!

Seriously, thanks for your interest in my projects. That's why I keep sharing new projects. So you can read and have fun building them.

Lastly, I hope you enjoyed this eBook as much as I loved creating it! If you did, please share one of my projects with your friends! Or you can send them the link below, so they can also download our free resources:

<http://RandomNerdTutorials.com/download>

Thanks again and have fun with your projects,

Rui Santos.

P.S. Make sure you visit my website to see the latest projects!

<http://RandomNerdTutorials.com>

Arduino Step-by-step Projects Course

This [Arduino Course](#) is a compilation of 25 projects divided in 5 Modules that you can build by following clear step-by-step instructions with schematics and downloadable code.



Included Projects:

Module #1: Lights and LEDs

- Traffic Lights
- LED Brightness on LCD
- Light Sensitive LEDs
- Remote Controlled LEDs
- Monitor Stand with Ambient Light
- Night Security Ligh

Module #2: Sensor Projects

- Smoke Detector Alarm
- Parking Sensor

- Arduino Weather Station (AWS) with Temperature, Humidity, Pressure, Date and Time
- Weather Station – Data Logging to Excel
- Request Sensor Data via SMS

Module #3: Sound Projects

- Pseudo-Theremin
- Sound Sensitive Lights
- Piano Keyboard

Module #4: Creating Applications

- Bluetooth Android App
- Voice Controlled Relay App
- Arduino Ethernet Web Server (Relay + Temperature)
- Shake Controlled LED
- Intruder Detector with Notifications

Module #5: Miscellaneous Projects

- Arduino Stopwatch
- Electronic Dice Roller
- Memory Game
- Light following Robot
- Bluetooth Remote Controlled Robot
- Time Attendance System with RFID

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 more than 200 free blog posts with complete tutorials using open-source hardware that anyone can read, remix and apply to their own projects. To keep free tutorials coming, there's also "premium content". To support Random Nerd Tutorials you can [download premium content here](#).

Learn ESP32 with Arduino IDE

This is practical course where you'll learn how to take the most out of the ESP32 using the Arduino IDE. This is our complete guide to program the ESP32 with Arduino IDE, including projects, tips, and tricks! [Read product description](#).

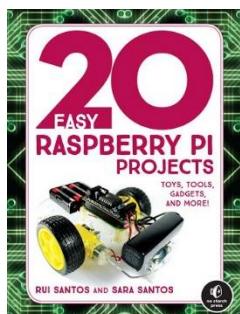


Android Apps for Arduino with MIT App Inventor 2

This eBook is our step-by-step guide to get you building Android applications for Arduino, even with no prior experience! You're going to build 8 Android applications to interact with the Arduino. [Read product description](#).



20 Easy Raspberry Pi Projects



20 Easy Raspberry Pi Projects book is a beginner-friendly collection of electronics projects using the Raspberry Pi. This book was a collaboration with the NoStarch Press Publisher and it is available in paperback format. [Read 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 product description.](#)



Home Automation Using ESP8266 (3rd Edition)

This course is a step-by-step guide designed to help you get started with the ESP8266. If you're new to the world of ESP8266, this eBook is perfect for you. If you've already used the ESP8266 before, I'm sure you'll also learn something new. [Read product description.](#)

