Arduino Exodus

Beginner Arduino & Project

Electronic Technology
Basic Arduino Coding

Arduino Exodus
Beginner Arduino
& Project

Electronic Technology
Basic Arduino Coding

Arduino Catenary

What is a microcontroller?

A large Arduino family was introduced

About hardware prototyping

Arduino software properties

Beginner Arduino

Intermediate Arduino: Inputs and Outputs

Project 01- IoT Fidget

Project 02 - 3 LED With Arduino 101

Project 03 - Ultrasonic Distance Sensor in Arduino

Project 04 - Flowing LED Lights With Arduino Uno R3

Project 05 - Light Sensor With Arduino in Tinkercad

Project 06 - DIY | 3x3x3 LED Cube for Arduino Nano+

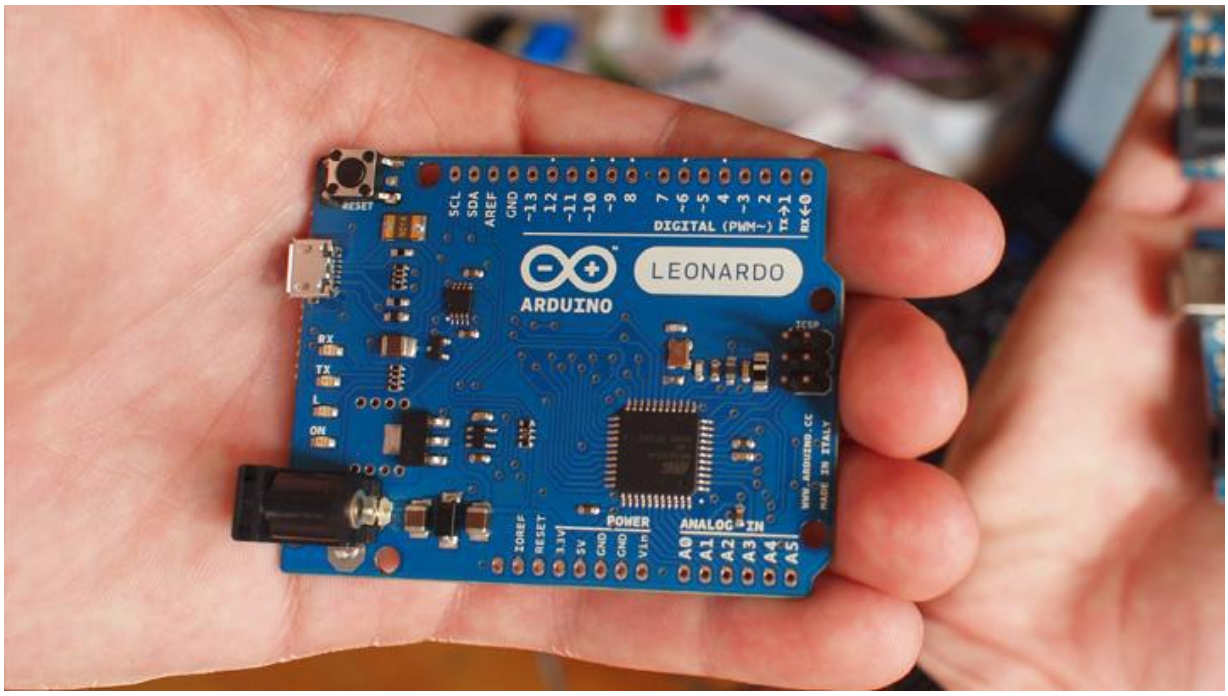
Project 07 - Ultrasonic Sensor (HC-SR04)

Project 08 - How to Use an RGB LED

Project 09 - PIR Motion Sensor

Project 10 - DIY Arduino Obstacle Avoiding Car at Home

What is a microcontroller?



Arduino is about connecting things. We'll do that in a few minutes after we learned more about microcontrollers in general and in particular
a large and wonderful Arduino family. This chapter will teach you how to be completely perfect
ready to enter code, phone, and check things with your new hardware friend. Yes, this will do
it happened quickly, very quickly; now let's go inside!
What is a microcontroller?
A microcontroller is an integrated circuit (IC) that contains all the main components of a standard

Computer, the following:

• Processor
• Memories
• Edges
• Inputs and outputs

The brain processor, the part where all the decisions are made and what
he can count.

Memories are often the two spaces where both the internal system and the user
elements are active (commonly called Read Only Memory (ROM) and Random
Access Memory (RAM)).

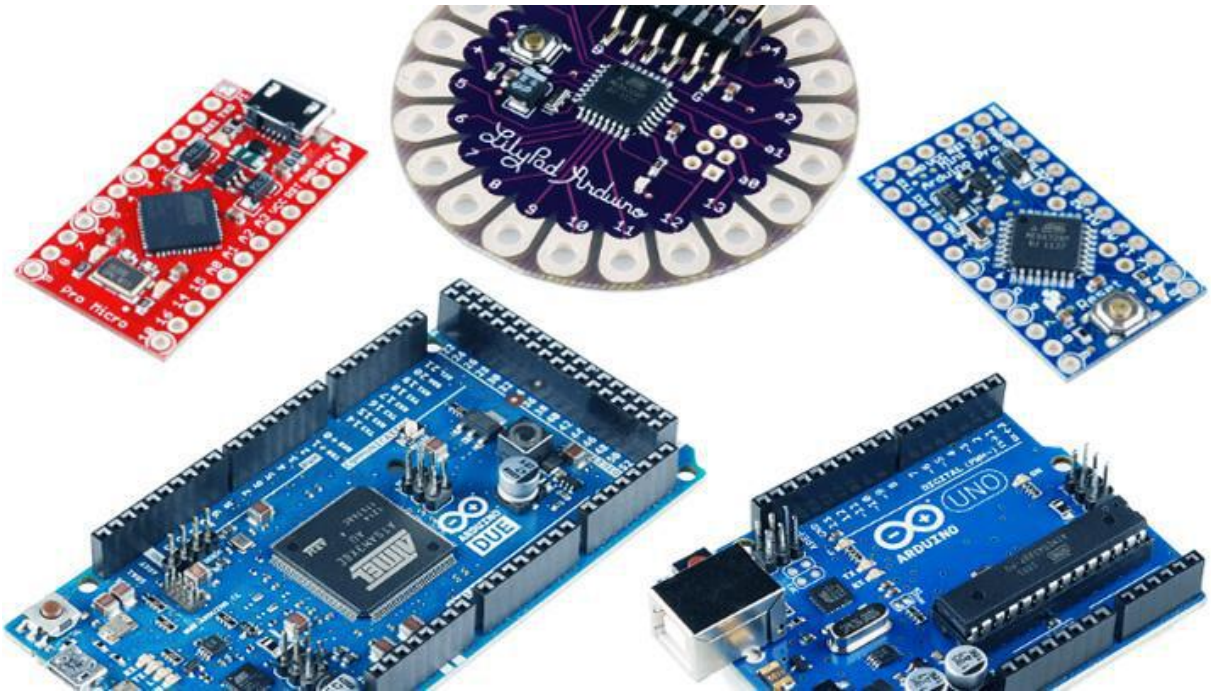
I describe in detail the artificial materials contained in the global board; these are
a very diverse range of integrated circuits with a major purpose: to support
processor and extend its power.

Inputs and outputs are means of international communication (worldwide
microcontroller) and the microcontroller itself.

The first single-chip processor was developed and developed by Intel Corporation in
In 1971 under the name Intel 4004. It was a 4-bit central processing unit (CPU).
Since the 70s, things have changed a lot and we have a lot of processors around us.
Look around, you will see your phone, your computer, and your screen. Processors or
microprocessors drive almost everything.

Compared to microprocessors, microcontrollers provide a way to reduce power
usage, size, and cost. Indeed, microprocessors, albeit faster than ever
processors embedded in microcontroller, requiring multiple interruptions in order to operate
work. The high level of integration provided by the microcontroller makes it friendly
of embedded programs that control the engine of the car, the remote control of your TV,
desktop equipment including your beautiful printer, household items, children's games,
cell phones, and I can go on…

A large Arduino family was introduced



microcontroller based on a single board. It is the most popular platform targeted from the file
The cable platform is also first designed for popularity
the use of electronics in building collaborative projects for university students.
It is based on the Atmel AVR processor and offers many inputs and effects
on only one hardware item.

The project was launched in Italy in 2005 by founders Massimo Banzi and David
Cuartielles. Today it is one of the best examples of open source concept, brought to the world of hardware and is often used only in the file
world software.

We are talking about the Arduino family because today we can count about 15 boards
‘Arduino-based’, which is a funny meta-term to describe a different type of board

designs everything made using the Atmel AVR processor. The main difference between

Those boards are:

• Processor type
• Number of inputs and outputs
• Form factor

Some Arduino boards have great power, considering the calculation speed,
some have more memory, some have more input / output (check more
Arduino Mega), some are designed to be integrated into complex projects as well
I have a very small form with inputs and very few results… as I have been using
to tell my students each can find his friend in the Arduino family. There is also
boards that include parameters such as Ethernet Connectors or Bluetooth
modules, including antennas.

The magic behind this family is the fact that we can use the same Combined
Development Environment (IDE) on our computers on any of those boards.
Some bits need to be set properly but this is the same software as well
the language we will use:

An excellent but incomplete reference page for this can be found in Hardware.

I especially want you to check out the following models:

• The Arduino Uno is the basic one with an interchangeable chipset
• Arduino Mega, 2560 offers a plethora of inputs and effects
• Arduino LilyPad, worn as accessories

• Arduino Nano, very small

Throughout this book I will use Arduino Mega and Arduino Uno as well; but do not do it
fear, once you are familiar with the Arduino system, you will be able to use any
their

About hardware prototyping

We can easily design and build software today using many open sources
forums where you can find many useful communities on the web. I am
you are considering Processing, as well
openFrameworks (C ++ - based, check out, but
there are many others who sometimes use very different paradigms such as clicks
programming languages such as Pure Data For Windows.

Because we, the makers, are fully involved in artificial insemination, all of us
demand and need to build and build our tools and often means hardware
and electronic tools. We want to expand our computers with sensors, blinking
lamps, and even make independent gears.

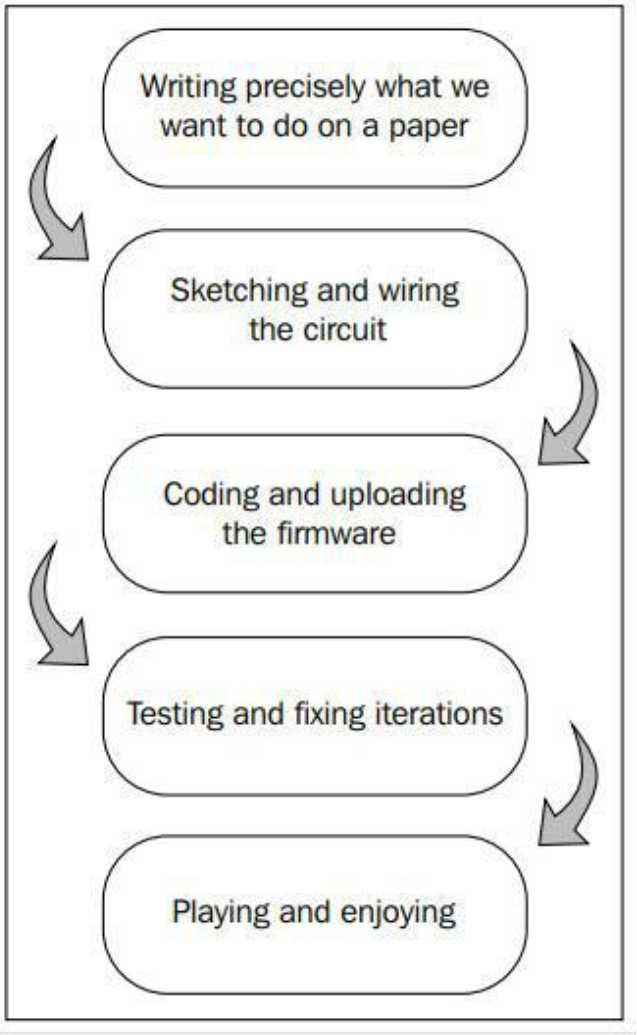
Even testing for basic things like flashing a light emitting diode (LED), is true
incorporates many things from enabling low-level chipset systems, from resistors value calculations to voltage-driven quartz clock setup. All those steps
it just gives heads to students and even those who are interested can be postponed to just do the initial testing.

Arduino appeared and changed everything in the world by showing that i

An inexpensive and all-encompassing solution (you have to pay $ 30 for Arduino Uno
R3), a cross-platform toolchain running Windows, OS X, and Linux, very simple
high-quality C language and a library that can also convert low-level pieces, and completely
expandable open source framework.
Indeed, with a small, well-equipped board, a USB cable, and your computer,
you can read electronica, embedded hardware program using C-language, and blink
your LED.

Hardware prototyping became (almost) as simple as software similarity due to
high level of integration between software and hardware provided by the whole frame.

One of the most important things to understand here is the prototyping display cycle.



From our point of view to our final offer, we usually have to follow these steps.
If we want to do that LED flashing, we have to define a few flashing features
For example. It will help to accurately describe the project, which is the key to success.
After that we will have to draw a scheme with our Arduino board and our LED; will do
catch the question, "How are they connected together?"

A firmware program that uses the C language can be started directly after we have it
underline the circuit because, as we will see later, it is directly related to the hardware.

This is one of the strongest forces of Arduino development. Do You Remember? Board
design is only designed to make us think about our project and not to confuse
us with invisible pieces of reading that are at a very low level.

The upload step is very important. It can give us many details especially if it happens to continue to solve the problem. We will learn that this step is not necessary
more than a few clicks when the board is properly installed on our computer.
After that, testing and adjustment of the subcycle will take place. We will learn by doing, by testing, and by

it means failure again. It is an important part of the process and will teach you the
a lot. I have to admit something important here: when I first started my career
bonome project, RGB monome clone tool,
I spent two hours repairing the LED matrix with wires back. Now, I know them very well
because I failed one day.

The last step is the coldest. I say that because we have to keep it in mind
the ultimate goal, the one that will bring us happiness in the end; it is the secret to success.

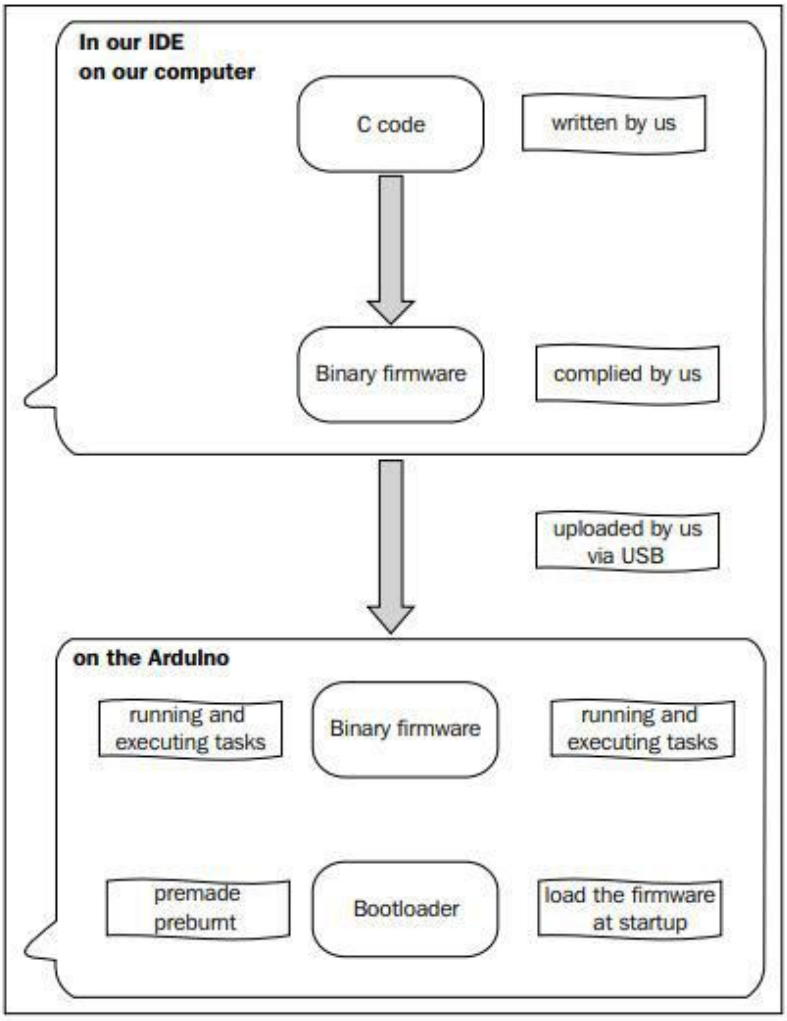
Understanding Arduino software properties

To understand how to make our beautiful Arduino board work like us I want, we need to understand the design of global software and tool tools
which we will use soon.

Take your Arduino board by hand. You will see a rectangular IC by name

ATMEL listed above; Here is the processor.
This processor is the place that will contain the whole program that we will rewrite
that will make things happen.
When purchasing Arduino, processor, also called chipset,
is pre burnt. It is organized by caring people to make our lives better Easier. The system that already exists on the chipset is called bootloader Basically, it takes care of itself
the first moment of the life of the processor when you give it some power.
But its main role is the responsibility of our firmware, I mean, our valuable integrated system.

Let's take a look at a little drawing to better understand:



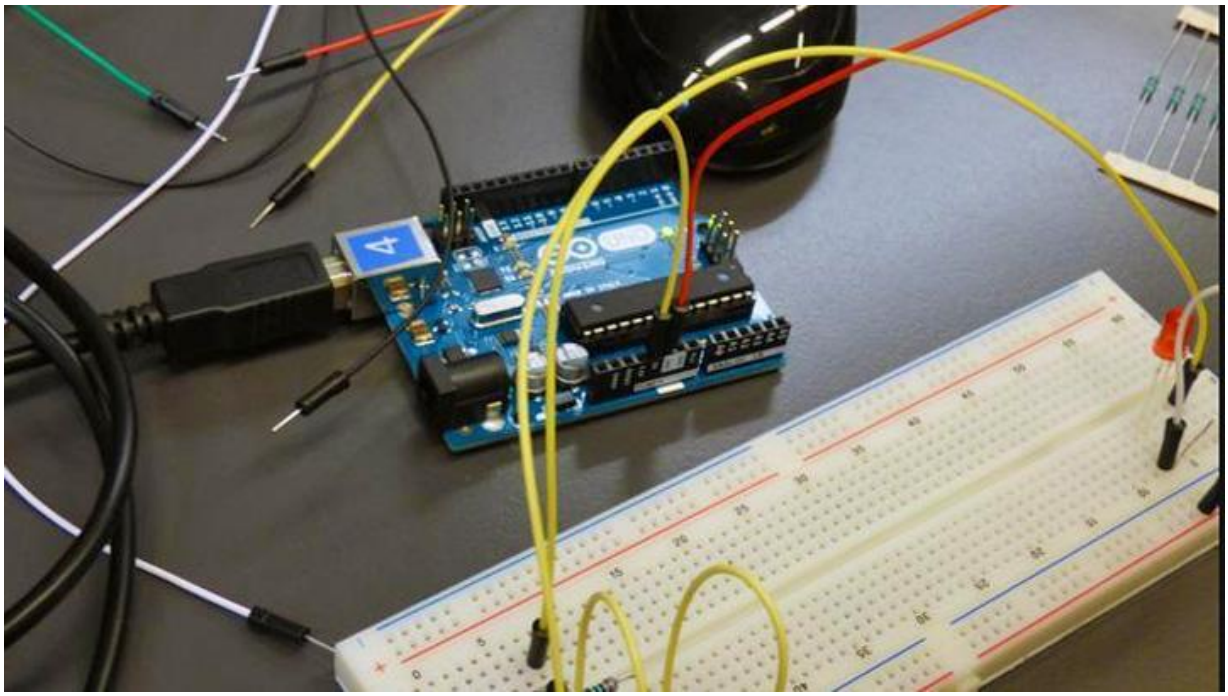
I would like to explain to you that bootloader is hardware and firmware software
is user software. Indeed, and there is some value for memory spaces
on the chipset does not match the writing functions (within the specific hardware
we will discuss in future sections of this book). We use a program designer, we can't
override the bootloader (the safest at this point in our reading) but only
firmware. This will suffice for the purpose of improvement, as you will see the whole book.
Not all bootloaders of Arduino boards are the same size. Indeed, they are

very specific to the hardware part, which gives us more output than the
hardware; we can focus on higher design standards because bootloader provides
Us services such as firmware upload via USB and serial monitoring.

Now let's download the required software:

FTDI USB drivers: http://www.ftdichip.com/Drivers/VCP.htm
Arduino IDE: http://arduino.cc/en/Main/Software
Processing: http://processing.org/download/

Beginner Arduino



Arduino is a pocket-sized computer (also called a "microcontroller") that you can use to control circuits. Works with a foreign name through sensors, lead, engines, speakers ... even the internet; this makes it a flexible platform for many creative projects. Other popular uses include:

Structured lighting that reflects responsiveness to music or social media.
Robots that use information from sensors to navigate or perform other tasks.
Different controls, default and social media for music, games, and more.
Connecting real world objects online (twitter is very popular).
Anything connected.
Automation and prototyping.

There are tons of amazing Arduino Projects posted online, here are some of my favorites:

Twitter Mood Light with RandomMatrix, a color that changes color depending on what types of emotional words are best on Twitter

Nebulophone Synth with Bleep Labs:
Location by Mads Hobye:
Sandy Noble's Polargraph Imaging Machine:
Flame-Throwing Jack-O-Lantern by Randy Sarafan and Noah Weinstein:
Umbrella that illuminates the rain in snl017:

There are a few microcontrollers on the market today, but Arduino stands out from the rest of the active online community around you. If you search on google or youtube, you will find tons of great project ideas and information to get you started. Whether you have no experience program or working with a microcontroller, Arduino is easy to get up and running, and is a great way to learn about electronic experimentation.

This Reading is written for the Intro to Arduino section I am teaching at Women's Audio Mission this month. I will be posting tutorials on

advanced Arduino topics and the construction of customized MIDI controls with Arduino over the next few weeks as the class progresses. More information about Arduino can be found on the Arduino reference page.

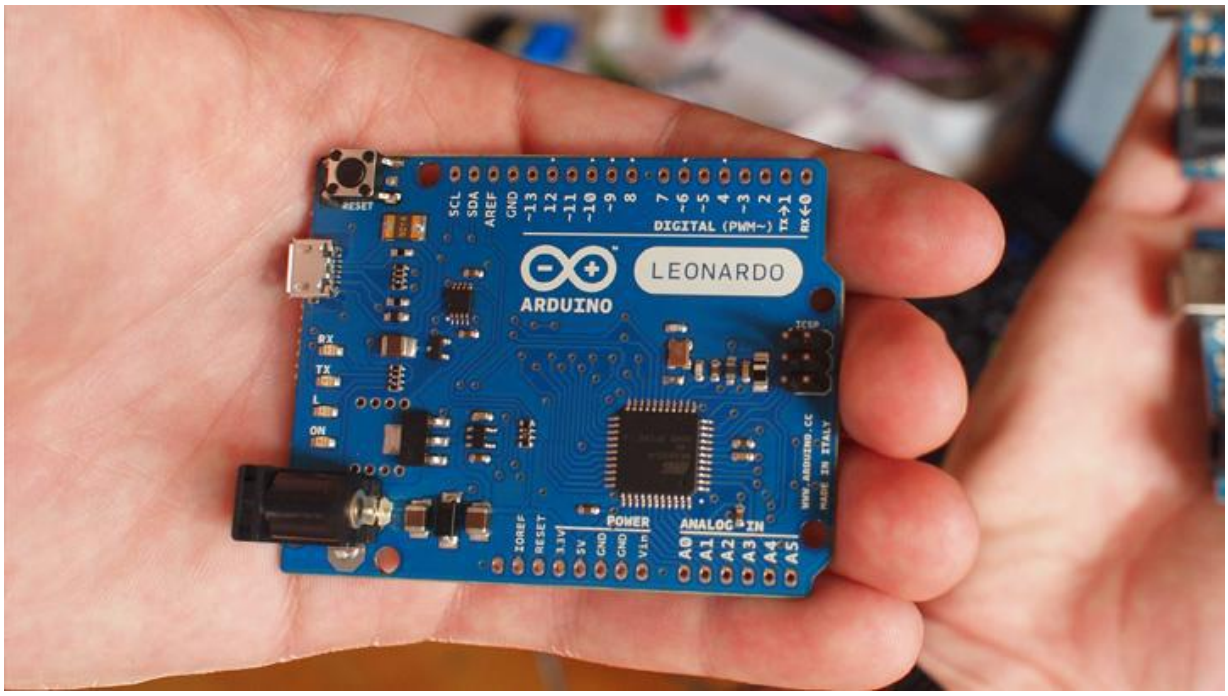
For this class you will need:

(1x) Arduino
(1x) Amazon usb cable
(1x) bread board (this one comes with jumper straps)
(1x) Amazon jumper cables
(4x) Digikey red LEDs
(4x) 220Ohm resistors
(1x) 10kOhm Resistant
(1x) Digikey strategy button
(1x) 10kOhm potentiometer
(1x) RGB LED (standard cathode)

Items to order items: Digikey is usually the cheapest place to find parts and shipped very quickly, but sometimes it's hard to find what you want because it has so many things. If Digikey gives you a lot of trouble try Jameco, you'll pay a few cents more for each part, but it's much easier to navigate their list. If you need things right away, you can find things, bread boards, cables, and Arduinos at your local Radioshack, but you will usually pay more. Adafruit and Sparkfun are great online stores for finding cool sensors or other Arduino accessories and usually have tutorials and sample code for their more sophisticated components. Amazon is also a great place to explore, they currently have Arduino Unos for $ 22, the cheapest I've ever seen.

In this Edited I will be using 123D circuits to show and emulate circuits, embedded circuit simulations work well with the Chrome browser.

Step 1: What is Arduino



First we will look at all parts of Arduino. Arduino is actually a small computer that can connect to electrical circuits. The Arduino Uno is powered by the Atmega 328P chip, which is the largest chip on the board (see photo note in the picture above). This chip is able to perform programs stored in its memory (very limited).

We can download applications to the chip via USB using Arduino IDE (download this if you have not already done so). The USB port also enables Arduino. Alternatively, we can power the built-in board using a power jack, in which case we do not need a USB connection.

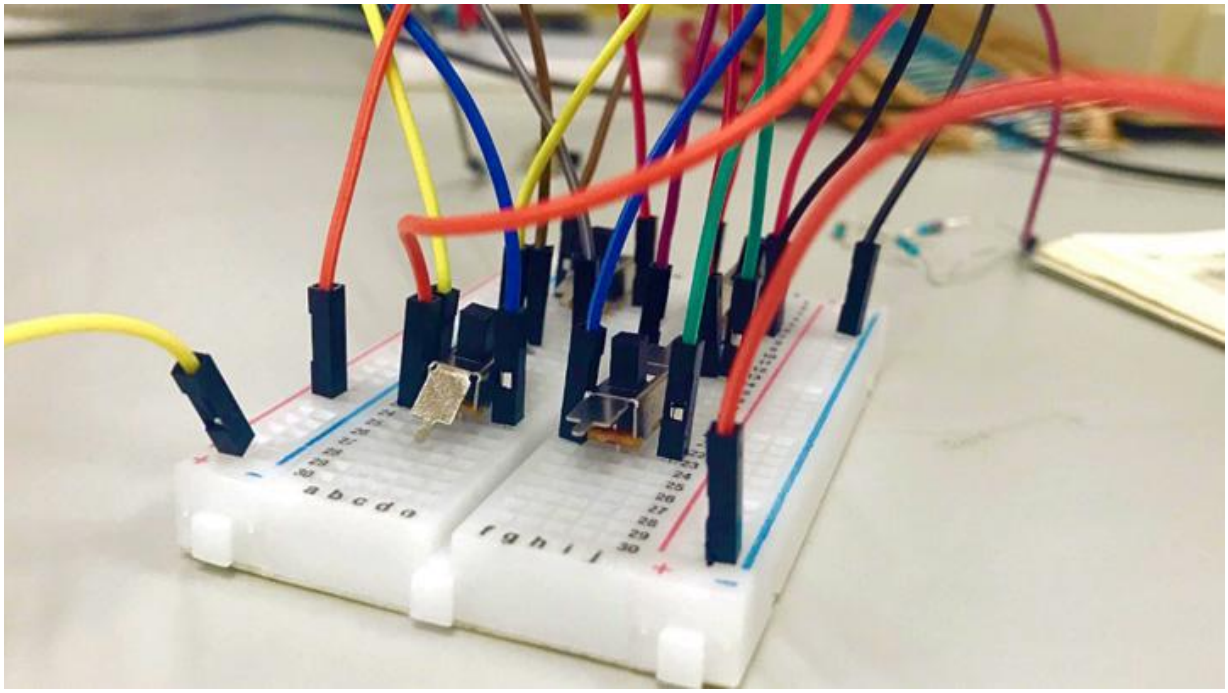
Arduino has a few rows of pins that we can connect wires to. The power pins are labeled in the image above. Arduino has both 3.3V or 5V specifications; In this section we will use the 5V supply, but you can get chips or items that require 3.3V to work, in which case the 3.3V supply will be useful. You will also find some pins marked "GND" in Arduino, these are ground pins (ground the same thing as 0V). At the moment electricity flows from a certain amount of electricity to the ground, so these anchors are useful in finishing circuits, we will use them more often.

Arduino has 14 digital anchors, labeled 0-14, that connect to circuits to turn it on or off, or to measure buttons with 2 other state circuits

(button is state two because it is pressed or not pressed, as dialing, which has a variety of possible conditions). These anchors can act as an inlet or outlet, which means they can control the circuit or adjust it.

Next to the power connector are the Analog input pins, labeled A0-A5. These pins are used to make analog measurements of sensors or other objects. Analog input is ideal for measuring objects in the range of available values. For example, an analog input pin will allow us to estimate the flex sensor flex value, or the number of turned dials. You can use analog input to measure a digital component (like a button) or act as a digital output, primarily more powerful digital pins.

Step 2: How to use BreadBoard



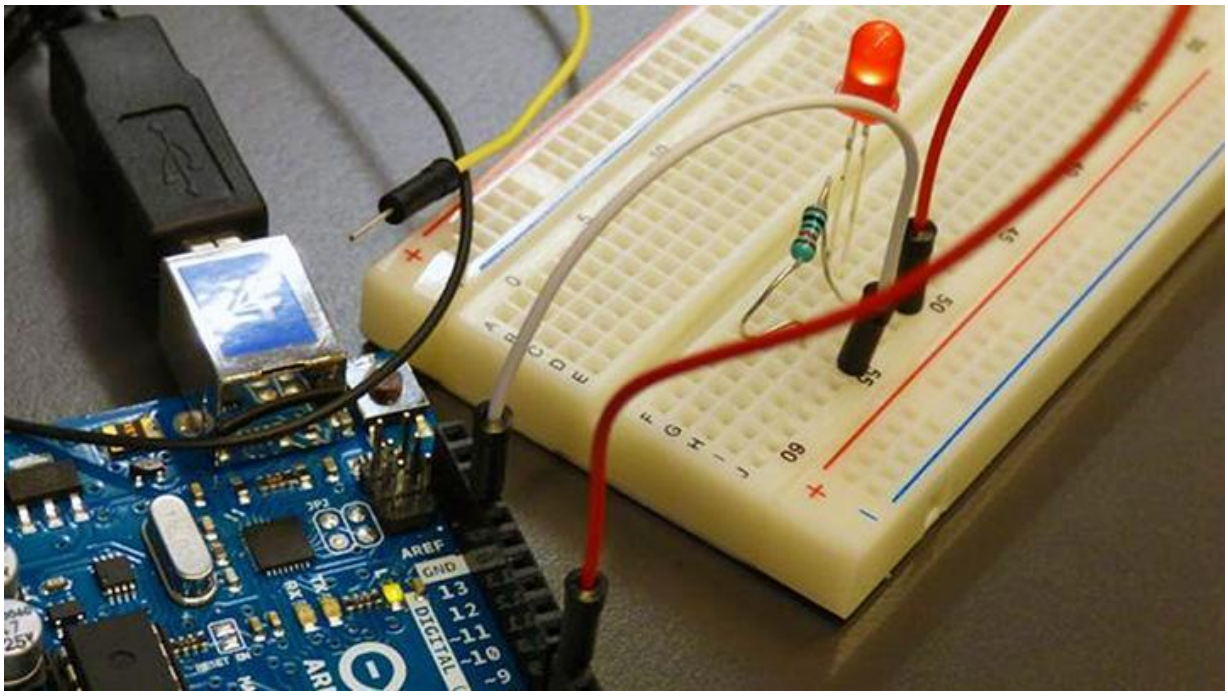
Bread boards should make temporary electrical connections between objects so that we can inspect the circuits before fully assembling them permanently. The whole class will be held on the bread board so we can reuse items and make quick changes to the circuit.

Bread boards have holes in which you can insert wires or other electrical appliances. Some of these holes are electrically connected to each other by means of metal straps under the bread board. Here's how communication works:

On each side of the loaf board, two rows of holes are connected throughout the length of the board (pictures 1 and 2 above). Typically, you will connect these long "rails" to 0V (also called "ground") and any gas power you used (in this class we will use 5V from Arduino), so that those connections are available everywhere on board. In this case, the first thing you want to do is plug in these links to your Arduino as shown in Figure 4, be careful how I connect the line to the "-" line and 5V to the line marked "+", your breadboard or it may not be written. Note: sometimes these side strands will only stretch half the length of the bread board, using the strings to complete the connection (Figure 5).

All the other holes in the breadboard are arranged in five rows in the middle of the bread board (Figure 3). This is where you will connect electrical appliances to each other and form circuits.

Step 3: Turn on the LED With 5V



As I explained earlier, current electricity flows from high power to low power. At this stage we will be removing all 5V emissions from Arduino, so for now it will flow from 5V out of Arduino, past our circuit, back to the Arduino "ground" pin. The first thing we will empower is the LED.

The LED opening circuit includes two components: a resistor and an LED. The formal representation of the region is shown in Figure 4 above. The resistor is represented by a rectangular box (you can also see that it represents a zigzag line). The LED is represented by a linear triangle, and usually some arrows point outwards representing the light emanating from the object.

So why do we need an enemy in this region? This Resistor is called the current Resistor Limit, which means that the Resistor limits the amount of electrical energy flowing through the LED. Every LED is limited to a certain current value, if you exceed that amount you will probably damage the LED. Using the Ohms rule, we can calculate the amount of current resistor current that we must use with our LED.

Ohms Law is very simple, stating that there is a balanced relationship between current and electrical power in the resistor:

increasing the voltage across the Resistor will increase the flow power. It specifically states:

V = Mina * R
where
V = voltage across the calculator
I = current through resistor
R = resistance - this is what we want to calculate
so if we know the values of V and me, we can calculate the correct R for our region

First we need to calculate the volume power across the calculator. In the circuit shown in Figure 4, a total of 5V is applied to the circuit. Most 3mm or 5mm LEDs you will use require 3V lighting, so the remaining 2V (5V - 3V = 2V) is used across the resistor.

Next we calculate the current flow using a resistor. Most 3mm or 5mm LEDs operate at full brightness at about 20mA current; passing this can damage the LED, and going below this will make the LED light dim (but not harmful). If we think we want to use our LED at 20mA, we know that the same current value should run through the resistor because the components are connected together in series. This leaves us with:

2V = 20mA * R
2V = 0.02A * R
R = 100 Oms

So 100 Ohms is the perfect little resistance we need to make sure we don’t damage the LED. For your own good, it is a good idea to use something a little higher, in case your LED has slightly different dimensions than I have used here. I like to use 220Ohms because it always seems like I have a lot of people around. If you know your LED measurements (you can find it in the LED database) and want to do this calculation yourself, you can also try using an online calculator.

Next we will install the LED on the bread wire. Connect the resistor and LED to the central part of the bread board so that the long LED lead is connected electrically to one of the resistor guides (Figure 3).

Then connect the remaining Resistor end to 5V and the remaining LED end to the ground. You should see the LED light glow.

Some things to try:

Note that LED tracks are not the same length, this is because LEDs need to be in a circuit in a certain direction in the circuit. It currently flows continuously with LEDS from long lead to short lead (in system representation, the current flow points where the triangle points, here is a good picture of that). Try turning on your LED position - you should find that the LED will not light up when placed in the rear region.

Resistors, on the other hand, have no shape, you will see that their lead is at the same length (their formal representation also shows the balance). Investigating the position of the opponent in the region will not affect the region - try it.

Now try to change the position of the LED and the Resistor in the region (picture 5). You should find out that this also does not make a cycle. Regardless of whether the current opposing enemy is on one side of the LED or the other, it will still be effective in protecting the current LED.

Step 4: Anatomy of Arduino Drawing

Programs in the Arduino language are called "drawings". Arduino drawing consists of two main parts: setup function and loop function.

setup () - The setup () function is performed automatically at the beginning of the Arduino program. Within this function you will start the variables, pins, and any libraries you may be using in your drawing. The setup () function only works during Arduino drawing, when the board is enabled or reset.

loop () - loop () is where most of your system will stay. This operation is performed after the setup () has been completed. Arduino will execute commands inside the loop over and over until the board is enabled.

From here on out, the Arduino index page will be very helpful in finding out about Arduino language and editing environment.

Step 5: Arduino LED Blink

In this example we will plug our LED circuit into one of the Arduino digital pins and turn on and off the LED with the code. This example introduces a few useful functions built into the Arduino language, namely:

pinMode (pinNumber, mode) - pinMode is used at the time of setting () the part of the drawing to launch each pin we use as input or output. Cannot read or write the pin before pinMode is set. pinMode () picks up two issues - the pin number (each Arduino pin has a label with the number) and that mode we want the pin (either "INPUT" or "OUTPUT"). In the case of LED flashing, we send data from Arduino to control the LED status, so we use "OUTPUT" as a secondary argument.

DigitalWrite (pinNumber, status) - digitalWrite is a command that allows us to set the pin power to 5V or ground (remember "earth" is equal to 0 Volts). In the last example we connected the LED to a 5V feed and saw it turn on, if we connected the LED to one of the Arduino digital pins instead, we could turn the LED on by setting the pin to 5V and turn it off by setting the pin down. DigitalWrite () also takes up two issues - pin number and pin status ("HIGH" 5V and "LOW" by ground).

timeInMs - delays stop the program for a given period of time. For example, delay (2000) will slow down the system by 2000 milliseconds (2000 milliseconds = 2 seconds), delay (100) will stop the system for 100 milliseconds (1/10 seconds), etc ...

Below the LED Blink code, apply this code to your Arduino.

```
//LED Blink

int ledPin = 7;//the Arduino pin that is connected to the LED

void setup() {
  pinMode(ledPin, OUTPUT);// initialize the pin as an output
}

void loop() {
  digitalWrite(ledPin, HIGH);//turn LED on
  delay(1000);// wait for 1000 milliseconds (one second)
  digitalWrite(ledPin, LOW);//turn LED off
  delay(1000);//wait one second
}
```

A few notes in the code:

Lines starting with "//" in the comments - Arduino ignores this.

You may have noticed all the semicolons, semicolons used at the end of every command in the Arduino language. If you forget the semicolon, you will get an error. You will find that many other editing languages use semicolons at the end of each line.

In this code "ledPin" varies. Variables are used to store data in programs, in this diagram, I use the "ledPin" variable to store the number 7. Later in the system when Arduino hits the line with the flexible "ledPin", it will check the flexibility according to its current saved value. So the line:

pinMode (ledPin, OUTPUT);

tested by Arduino as:

pinMode (7, OUTPUT);

In fact, we can switch to other uses of pinMode by number 7 and the system will work the same, but the flexible use helps us to easily read and understand the code.

"int" from the first line of data type - in Arduino language, you should always start the variables by declaring their type. There are many different types (you can read them all here), because now all you need to know is whether int variables are absolute or negative values - you'll use them often.

Below the impersonation of the drawing, try pressing the play button to see how it works (works well in Chrome)

As expected, the LED turns on for one second, then turns off for one second. Try changing the duration of the delay () to see how it affects the LED blink time.

Another thing to look for - the most common mistake people make is to leave the last delay () in the loop (). Try it - you will find that the LED stays on without blinking. This may confuse you, as we still have a digitalWrite command (ledPin, LOW) in the system. What happens here is the lead is off, but Arduino immediately hits the end of the loop () and begins to pull out the first row of loop () again (to open the LED). This happens so fast that the human eye cannot see the LED turning off that brief moment while the loop restarts.

Step 6: Control multiple LEDs with Arduino

In this example we will plug the wires into as many as three LEDs as we did in the last example, and then control them with multiple digital pins. First start with three more LEDs and limited current resistors as shown below:

If we want to rotate all the LEDs and turn them on and off we can write our Arduino diagram as follows:

```
//Multi LED Blink

int led1Pin = 4;
int led2Pin = 5;
int led3Pin = 6;
int led4Pin = 7;

void setup() {
```

```
  //initialize the led pins as an outputs
  pinMode(led1Pin, OUTPUT);
  pinMode(led2Pin, OUTPUT);
  pinMode(led3Pin, OUTPUT);
  pinMode(led4Pin, OUTPUT);
}

void loop() {
  digitalWrite(led1Pin, HIGH);//turn LED on
  delay(1000);// wait for 1000 milliseconds (one second)
  digitalWrite(led1Pin, LOW);//turn LED off
  delay(1000);//wait one second

  //do the same for the other 3 LEDs
  digitalWrite(led2Pin, HIGH);//turn LED on
  delay(1000);// wait for 1000 milliseconds (one second)
  digitalWrite(led2Pin, LOW);//turn LED off
  delay(1000);//wait one second

  digitalWrite(led3Pin, HIGH);//turn LED on
  delay(1000);// wait for 1000 milliseconds (one second)
  digitalWrite(led3Pin, LOW);//turn LED off
  delay(1000);//wait one second

  digitalWrite(led4Pin, HIGH);//turn LED on
  delay(1000);// wait for 1000 milliseconds (one second)
  digitalWrite(led4Pin, LOW);//turn LED off
  delay(1000);//wait one second
}
```

This works, and we can leave it like that and everything will work fine, but it is not the most effective way to write our code. Instead, we will use a structure called a loop to rotate the LEDs. For loops it helps to repeat a piece of code over and over again. In the above case we repeat the lines:

```
digitalWrite (led4Pin, HIGH);
delay (1000);
digitalWrite (led4Pin, LOW);
delay (1000);
```

Here's how to write a loop:

```
of (int ledPin = 4; ledPin <8; ledPin ++) {
```

```
digitalWrite (ledPin, HIGH);
delay (1000);
digitalWrite (ledPin, LOW);
delay (1000);

}
```

In the first row we start the "ledPin" variable as 4 and tell Arduino that we would like to rotate it with the starting variable values by 4, up to 7 (ledPin <8). LedPin ++ tells Arduino to increase ledPin value by 1 each time we repeat the loop. After that we make lines inside the loop using the flexible ledPin. So first ledPin = 4, and pin 4 is turned on and off, then ledPin is raised to 5 and then the loop starts again, this time the pin 5 is opened and closed, and so on ... The result is exactly the same as the diagram of verbose more, where we repeat digitalWrite commands and frequent delays. Here is the full diagram:

```
//Multi LED Blink

int led1Pin = 4;
int led2Pin = 5;
int led3Pin = 6;
int led4Pin = 7;

void setup() {
  //initialize the led pins as an outputs
  pinMode(led1Pin, OUTPUT);
  pinMode(led2Pin, OUTPUT);
  pinMode(led3Pin, OUTPUT) ;
  pinMode(led4Pin, OUTPUT);
}

void loop() {
  for (int ledPin=4;ledPin<8;ledPin++){//for pins 4-7
    digitalWrite(ledPin, HIGH);//turn LED on
    delay(1000);// wait for 1000 milliseconds (one second)
    digitalWrite(ledPin, LOW);//turn LED off
    delay(1000);//wait one second
  }
}
```

Step 7: Fade LEDs With AnalogWrite

Sometimes we will want to control the LED light, in which case we can use a command called analogWrite (). AnalogWrite works by turning on and off the LED very quickly, very quickly so that our eyes do not see blinking. If the LED spends its break time and half of its time, then it will appear as the light part. This method is called pulse wide modulation (PWM), which is often used electronically because it allows us to control the part in an "analog" way using a digital pin. Not all digital pins in Arduino can make PWM, if you look at your Arduino, you will see that some pins have "~" next to them (pins 3, 5, 6, 9, 10, 11), these are enabled pins for PWM.

Insert one of your LEDs into the PWM-enabled pin, using pin 9. Try using a blink diagram from the front, but use analogWrite instead of digitalWrite to turn on the LED (see diagram below). analogWrite () takes up two issues: pin number and light level (between 0 and 255).

```
//LED Blink (half brightness)

int ledPin = 9;//the Arduino pin that is connected to the LED

void setup() {
  pinMode(ledPin, OUTPUT);// initialize the pin as an output
}

void loop() {
  analogWrite(ledPin, 255);//turn LED on at full brightness (255/255 = 1)
  delay(1000);// wait for 1000 milliseconds (one second)
  digitalWrite(ledPin, LOW);//turn LED off
  delay(1000);//wait one second

  analogWrite(ledPin, 191);//turn LED on at 3/4 brightness (191/255 ~= 0.75)
  delay(1000);// wait for 1000 milliseconds (one second)
  digitalWrite(ledPin, LOW);//turn LED off
  delay(1000);//wait one second

  analogWrite(ledPin, 127);//turn LED on at half brightness (127/255 ~= 0.5)
  delay(1000);// wait for 1000 milliseconds (one second)
  digitalWrite(ledPin, LOW);//turn LED off
```

```
  delay(1000);//wait one second

  analogWrite(ledPin, 63);//turn LED on at one quarter brightness (63/255 ~= 0.25)
  delay(1000);// wait for 1000 milliseconds (one second)
  digitalWrite(ledPin, LOW);//turn LED off
  delay(1000);//wait one second
}
```

Try changing the brightness of the analogWrite commands to see how it affects the brightness of your LED.

Next we will write the code so that the light runs smoothly from all the way to full light. We can copy the same piece of code:

analogWrite (ledPin, light);
delays (5); // short delay
brightness = brightness + 1;

Repeatedly (255 times), increasing light at the same time. Here's what it might look like:

```
//LED Blink (half brightness)

int ledPin = 9;//the Arduino pin that is connected to the LED

void setup() {
  pinMode(ledPin, OUTPUT);// initialize the pin as an output
}

void loop() {
  int brightness = 0;
  analogWrite(ledPin, brightness);//brightness = 0
  delay(5);// short delay
  brightness += 1;

  analogWrite(ledPin, brightness);//brightness = 1
  delay(5);// short delay
  brightness += 1;

analogWrite(ledPin, brightness);//brightness = 2
  delay(5);// short delay
  brightness += 1;

  analogWrite(ledPin, brightness);//brightness = 3
  delay(5);// short delay
  brightness += 1;
```

```
  analogWrite(ledPin, brightness);//brightness = 4
  delay(5);// short delay
  brightness += 1;

  analogWrite(ledPin, brightness);//brightness = 5
  delay(5);// short delay
  brightness += 1;

  analogWrite(ledPin, brightness);//brightness = 6
  delay(5);// short delay
  brightness += 1;

  //keep repeating until brightness = 255 (full brightness)
}
```

Or we can use the loop again to make the code much shorter. In the following boat I have two wires, the first LED lines rise from (0) to full light (255):

of (int bright = 0; brightness <256; bright ++) {

analogWrite (ledPin, light);
delays (5);

}

The second step across the street from full light to off:

of (int bright = 255; light> = 0; light -) {

analogWrite (ledPin, light);
delays (5);

}

(delay (5) is used to reduce dryness, so it takes 5 * 256 = 1280ms = 1.28seconds)

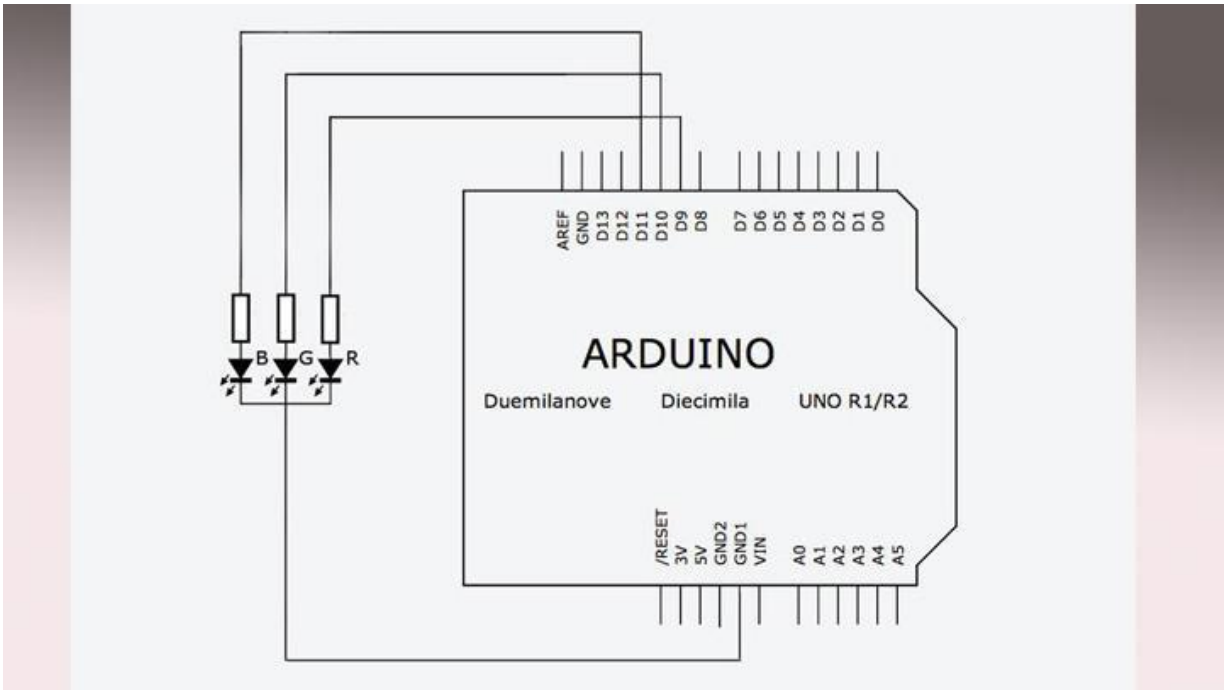
In the first line, we use "brightness--" to tell the loop to decrease the amount of light by 1 each time the loop repeats. Also note how the loop will work until light> = 0, using> = instead of> we enter the number 0 in width.

```
//LED fade
```

```
int ledPin = 9;//the Arduino pin that is connected to the LED

void setup() {
  pinMode(ledPin, OUTPUT);// initialize the pin as an output
}

void loop() {
  //ramp LED up to full brightness (0 to 255)
  for (int brightness=0;brightness<256;brightness++){
    analogWrite(ledPin,brightness);
    delay(5);
  }

  delay(1000);// wait one second

  //ramp LED down to no brightness (255 to 0)
  for (int brightness=255;brightness>=0;brightness--){
    analogWrite(ledPin,brightness);
    delay(5);
  }

  delay(1000);//wait one second
}
```

Here's what it looks like (This impersonation isn't really good, but you get the idea). Try setting the delay to see how it affects the speed of the ramps.

Step 8: RGB LED and Arduino



RGB LEDs are really fun, as you can see in the first picture above, each RGB LED is actually made up of three LEDs: one red, one green, and one blue. When you turn on a few LEDs at the same time they will combine to form new colors.

The RGB LEDs we use in this class are standard cathodes, which means all three LEDs share the same ground axle (some RGB LEDs, called standard anodes, share a normal supply pin and for different reasons). We will plug our circuit into the circuit as the first picture above, each LED on the RGB LED has a single 220Ohm resistor in the series which also has a cord that reaches the Arduino pin powered by PWM (I used pins 9-11). In this way, we can optionally turn each LED on the RGB LED and turn it off individually.

See the second image above to find out which RGB LED direction corresponds to red, green, blue, and ground (counting numbers 1-4).

The first drawing will rotate with each color on the LED:

```
//RGB LED - test

//pin connections
int red = 9;
int green = 10;
int blue = 11;
```

```
void setup(){
  pinMode(red, OUTPUT);
  pinMode(blue, OUTPUT);
  pinMode(green, OUTPUT);
}

void loop(){
  //turn red led on
  digitalWrite(red, HIGH);
  delay(500);
  digitalWrite(red, LOW);
  delay(500);

  //turn green led on
  digitalWrite(green, HIGH);
  delay(500);
  digitalWrite(green, LOW);
  delay(500);

  //turn blue led o n
  digitalWrite(blue, HIGH);
  delay(500);
  digitalWrite(blue, LOW);
  delay(500);
}
```

Then use analogWrite () and random () to set random light levels for each color in the LED. The three colors will combine in different sizes (depending on their brightness) to make a variety of colors (255 ^ 3 = 16,581,375 possible colors).

```
//RGB LED - random colors

//pin connections
int red = 9;
int green = 10;
int blue = 11;
void setup(){
  pinMode(red, OUTPUT);
  pinMode(blue, OUTPUT);
  pinMode(green, OUTPUT);
}
void loop(){
  //pick a random color
  analogWrite(red, random(256));
```

```
  analogWrite(blue, random(256));
  analogWrite(green, random(256));
  delay(1000);//wait one second
}
```

random (256); // returns the number between 0 and 255

Step 9: Arduino functions

The following diagram shows the LED from red to green to red to green and more ...

```
//RGB LED - fading between colors
//pin connections
int red = 9;
int green = 10;
int blue = 11;
void setup(){
  pinMode(red, OUTPUT);
  pinMode(blue, OUTPUT);
  pinMode(green, OUTPUT);
}
void loop(){
  for (int brightness=0;brightness<256;brightness++){
    analogWrite(red, 255-brightness);
    analogWrite(green, brightness);
    delay(10);
  }
  for (int brightness=0;brightness<256;brightness++){
    analogWrite(green, 255-brightness);
    analogWrite(blue, brightness);
    delay(10);
  }
  for (int brightness=0;brightness<256;brightness++){
    analogWrite(blue, 255-brightness);
    analogWrite(red, brightness);
    delay(10);
  }
}
```

The diagram above works, but there is a lot of code repeated. We can make it easier by writing our own assistant work that fades from

one color to another. Here's what the job will look like:

```
void fader (int color1, int color2) {

of (int bright = 0; brightness <256; bright ++) {
analogWrite (color1, 255-light);
analogWrite (color2, light);
delays (10);
}

}
```

Let's take a closer look at this job description. The function is called "fader" and takes two arguments. Each argument is comma-separated and has the type declared in the first line of the job description:

```
void fader (int color1, int color2) {
```

We recognize that both fader-accepted arguments are ints, and we use the words "color1" and "color2" as dummy variations in our definition of work. "Void" refers to the type of data that is returned to work, because our work returns nothing (simply issuing commands), we set the return type to zero. If we were to build a two-digit operation and return a product we would describe it as follows:

```
int multiplier (int number1, int number2) {

int product = number1 * number2;
return product;

}
```

Note how we declared int as a return type here instead of void.

Intestinal function is something we have seen before. It is the same with the loop we repeated in our last drawing, but the pin numbers have been replaced by the color change1 and color2. When we call:

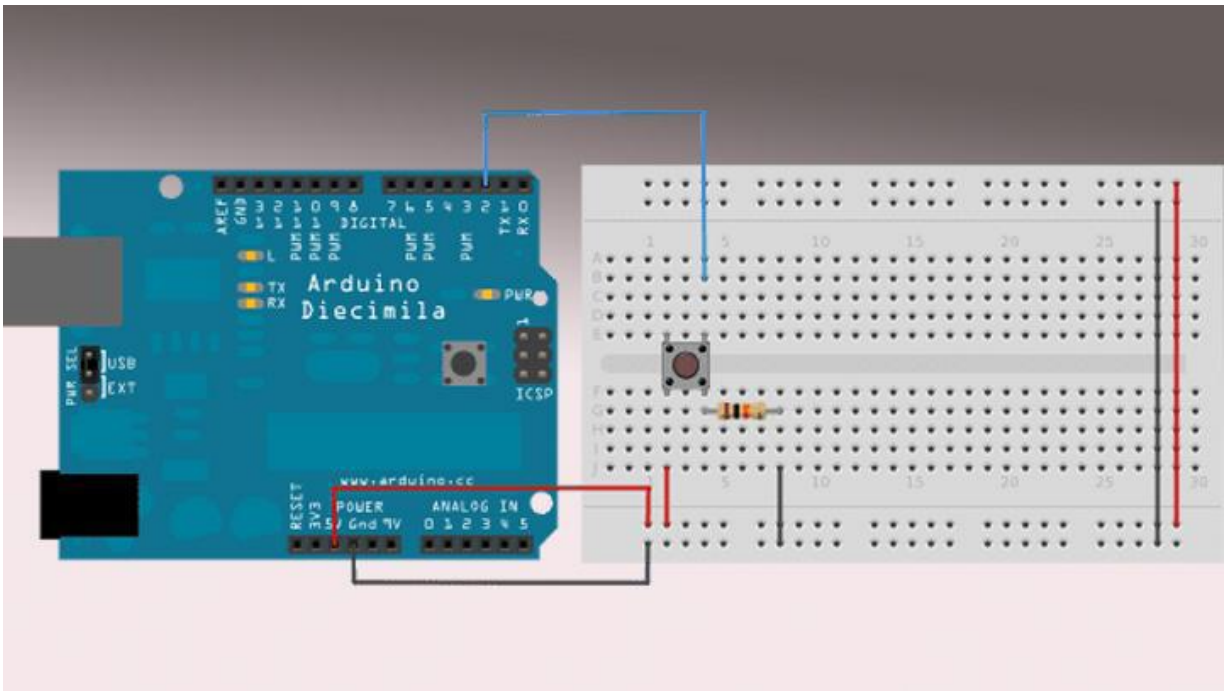
```
fader (red, green);
```

from Arduino loop (), Arduino checks the fader function for color 1 = red and color2 = green.

To sum up all this we can rewrite this drawing and use this function as follows, this will work exactly the same as the diagram above this step.

```
//RGB LED - fading between colors
//pin connections
int red = 9;
int green = 10;
int blue = 11;
void setup(){
  pinMode(red, OUTPUT);
  pinMode(blue, OUTPUT) ;
  pinMode(green, OUTPUT);
}
void loop(){
  fader(red,green);
  fader(green,blue);
  fader(blue, red);
}
void fader(int color1, int color2){
    for (int brightness=0;brightness<256;brightness++){
        analogWrite(color1, 255-brightness);
        analogWrite(color2, brightness);
        delay(10);
    }
}
```

Step 10: Button



Time for a new type of circuit, we will now look at how we use Arduino pressure buttons. Buttons are a type of switch, the type of button we use is called the "normal temporary open button". "Normally open" means that when the button is not pressed, no current will flow to the button because both sides are not connected - it forms like an open circuit (see first image above). "Temporary" means the fact that this switch only opens when you press it with your finger; this makes it different from the switch switch, which switches between open and closed positions every time you press it.

The button circuit we will use is made up of two elements - the Push button and the pixel. Unlike an LED circuit, we are not concerned with the current value of the pass button (at very high current levels we may have to worry about melting the button, but the Arduino is less powerful), so the resistor does not work as a current resistor in the LED cycle. Instead this Resistor acts as a pull resistor. The pull resistor binds the button down, so that the voltage measurement at the junction between the button and the resistor will remain 0V (set) when the button is not pressed (and the circuit is open). In this region, the amount of gravity resistor does not matter, I like to use something around 10kOhms.

Here is a button drawing:

```
//Button Press Detection
int buttonPin = 7;
void setup(){
  pinMode(buttonPin, INPUT);//this time we will set the pin as INPUT
  Serial.begin(9600);//initialize Serial connection
}
void loop(){
  if (digitalRead(buttonPin)==HIGH){//if button pressed
    Serial.println("pressed");
  } else {
    Serial.println("unpressed");
  }
}
```

Button drawing introduces a few new ideas:

digitalRead (pinNumber) - similar to digitalWrite (), but used to measure the HIGH or LOW value in our region. DigitalRead () takes up one issue - the pin number we read from. We must also ensure that we implement the correct installation pin:

pinMode (PIN button, INPUT);

Serial Connections - Coupling connections allow Arduino to send messages to your computer while the system is running, useful for debugging, sending messages to other devices or applications, or getting a better experience of what is happening in your region. To enable serial connection in your drawing, you must establish a serial connection in Arduino's () setting with the command Serial.begin (). Serial.begin () takes one argument, baud rate, which is the data transfer rate between Arduino and your computer, 9600 is the best baud rate yet. In the next box we will use Serial.println () to print messages in Arduino IDE (Tools >> Serial Monitor).

if / not - If / other statements give us more control over what orders are made there. In the button diagram I used the following statement if / else:

uma (digitalRead (buttonPin) == HIGH) {

Serial.println ("pressed");

} more {

Serial.println ("can be released");

}

if the digitalRead (buttonPin) effect returns to HIGH and Arduino prints the word "compressed", if digitalRead (buttonPin) returns something other than HIGH (like LOW), Arduino prints the word "unpressed" ". If the statements can check == ("equal to") ,! = ("Not equal to"),>, <,> =, and <=. Try using the following when making a statement in the Arduino loop ():

if (4> 3) {

Serial.println ("true");

} more {

Serial.println ("false");

}

Try changing the ifif statement to check other items.

Step 11: Input and Effects of Arduino Digital

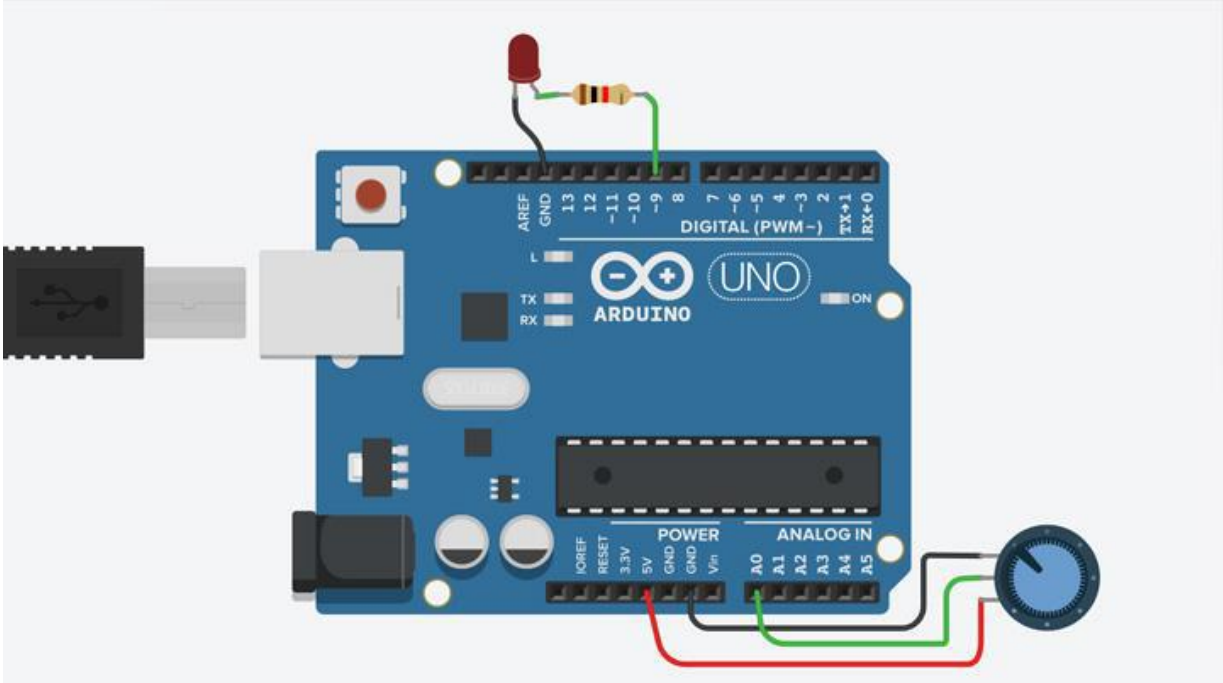
We will now use the data at the press of a button to turn on and off the LED. Change the drawing from the last step to turn on the LED connected to pin 8:

```
//button press detection with LED output
int buttonPin = 7;
int ledPin = 8;
void setup(){
  pinMode(buttonPin, INPUT);//this time we will set button pin as INPUT
  pinMode(ledPin, OUTPUT);
  Serial.begin(9600);
}
```

```
void loop(){
  if (digitalRead(buttonPin)==HIGH){
    digitalWrite(ledPin,HIGH);
    Serial.println("pressed");
  } else {
    digitalWrite(ledPin,LOW);
    Serial.println("unpressed");
  }
}
```

Here's what to look for:

Step 12: Arduino Analog Input



analogRead (pinNumber) - analogRead () read from one of the Arduino analog pins and output a value between 0 (voltage per pin = 0V) and 1023 (voltage per pin = 5V), if the analog pin voltage was 2.5V, then it would print:

2.5 / 5 * 1023 = 512

analogRead () takes one argument - the name of the analog pin (A0-A5) to be read.

A potentiometer is a Resistor with a central pin that connects to a certain length of resistor. As you turn on the potentiometer move the center pin next to the resistor and change the resistance level on either side of the pin. This allows the potentiometer to act as a variable power separator.

Connect the potentiometer so that the outer pins are connected at 5V to the ground (position does not matter), and the center pin connects pin A0 to the Arduino. Generate the following code and view the output in Serial Monitor.

```
//analog input

int potPin = A0;//center pin of the potentiometer is attached to pin A0

void setup(){
  //analog pins are initilized as INPUT by default, no need for pinMode() command
  Serial.begin(9600);
}

void loop(){
  int potVal = analogRead(potPin);//potVal is a number between 0 and 1023
  Serial.println(potVal);
}
```

Now open the pot and see how the printed value of potVal changes. You should see the arduino 1023 print when you open the pot all the way to the connected side in 5V, and 0 when you open the pot to the other side. You should also see a list of printed prices among those extremes.

Step 13: Working with Analog Input Data

Before you can use analog data to control some items in your system, you may need to measure or compress it between certain days and sizes. For example, suppose you want to use your analog input reading to control the LED light with analogWrite (). analogRead () returns numbers between 0 and 1023, but analogWrite () only accepts numbers between 0 and 255. In this

case you can use map () to measure the range of values from analogRead () to other analogWrite ();

map (value, fromLow, fromHigh, toLow, toHigh) - measure one distance to another. The map () receives four inputs: the amount we are trying to measure, the minute we measure it, the distance we measure, the minute we measure, and the distance we measure.

Here is an example:

```
//analog input with map

int potPin = A0;
int ledPin = 9;

void setup(){
  pinMode(ledPin, OUTPUT);
  Serial.begin(9600);
}

void loop(){
  int analogVal = analogRead(potPin);//analogVal is between 0 and 1023
  int scaledVal  = map(analogVal, 0, 1023, 0, 255);//scaled val is between 0 and
255
  Serial.print("analogVal = ");
  Serial.print(analogVal);
  Serial.print("   scaledVal = ");
  Serial.print(scaledVal);
  analogWrite(ledPin, scaledVal);
}
```

Also check the pressure (x, a, b) - prevents the number x between a and b. If x is under the return pressure a, x is greater than b constrain Return b, otherwise the return is x.

Step 14: Practice Arduino Installation and Results

The following example includes a button acquisition diagram and an analog LED control diagram. Insert the pin 7 pin button, as shown in step 10, then connect the pot to A0 and LED to pin 9, as shown in section 13. Then download the following code

```
//button press detection with LED output and variable intensity
int buttonPin = 7;
int ledPin = 9;
int potPin = A0;
void setup(){
  pinMode(buttonPin, INPUT) ;
  pinMode(ledPin, OUTPUT);
  Serial.begin(9600);
}
void loop(){
  if (digitalRead(buttonPin)==HIGH){//if button pressed
    int analogVal = analogRead(potPin);
    int scaledVal  = map(analogVal, 0, 1023, 0, 255);
    analogWrite(ledPin, scaledVal);//turn on led with intensity set by pot
    Serial.println("pressed");
  } else {
    digitalWrite(ledPin, LOW);//turn off if button is not pressed
    Serial.println("unpressed");
  }
}
```

The diagram turns off the LED and continues depending on the position of the button (pressed / pressed), and at the same time uses a potentiometer to control the LED light when it is in “on” position.

Step 15: Button As Change Switch

Sometimes you will be interested in the exact moment when the button is pressed or released, so you can start the event in your drawing. In this case you will need to save the currentState button and compare it to the last recorded country. If currentState is high and lastState is low, you will know that the button has just been pressed. See code below:

```
//Button Press Detection - single message

int buttonPin = 7;
boolean currentState = LOW;//stroage for current button state
boolean lastState = LOW;//storage for last button state

void setup(){
  pinMode(buttonPin, INPUT);//this time we will set the pin as INPUT
```

```
  Serial.begin(9600);//initialize Serial connection
}

void loop(){
  currentState = digitalRead(buttonPin);
  if (currentState == HIGH && lastState == LOW){//if button has just been pressed
    Serial.println("pressed");
    delay(1);//crude form of button debouncing
  } else if(currentState == LOW && lastState == HIGH){
    Serial.println("released");
    delay(1);//crude form of button debouncing
  }
  lastState = currentState;
}
```

I used something new in my statement if:

uma (currentState == HIGH && lastState == LOW)

this read "if currentState is high and lastState is low", && allows us to test the truth of many things with the same statement. You can also use || ("or") to check one thing or another is true. Learn more here.

You will also see that the following line appears twice in the code above:

Delays (1);

This delay was incorporated when giving the button time to settle the power supply before we started measuring again, this is called the charge deduction button; prevents us from counting one machine as two machines due to the dialogue of buttons. Use delays to make the button release okay in this simple example, but if you measure multiple buttons the delay will add and make your code much slower. This can end up giving your Hardware a feeling of backwardness. I will address some of the best ways to get out of this class later.

This code also introduces a new type of data: boolean. Booleans are used to store 1 piece of information, things like true / false, open / close, 1/0, HIGH / LOW. In my code I used it to keep the current and last button position (HIGH or LOW).

Here's how we can use this to switch on or off the LED each time the button is pressed:

```
//Button Toggle LED

int ledPin = 9;
int buttonPin = 7;
boolean currentState = LOW;//stroage for current button state
boolean lastState = LOW;//storage for last button state
boolean ledState = LOW;//storage for the current state of the LED (off/on)

void setup(){
  pinMode(buttonPin, INPUT);//this time we will set the pin as INPUT
  pinMode(ledPin, OUTPUT);
  Serial.begin(9600);//initialize Serial connection
}

void loop(){
  currentState = digitalRead(buttonPin);
  if (currentState == HIGH && lastState == LOW){//if button has just been pressed
    Serial.println("pressed");
    delay(1);//crude form of button debouncing

    //toggle the state of the LED
    if (ledState == HIGH){
      digitalWrite(ledPin, LOW);
      ledState = LOW;
    } else {
      digitalWrite(ledPin, HIGH);
      ledState = HIGH;
    }
  }

  lastState = currentState;
}
```

In the code above I set a variable called "ledState" to keep the current LED state, where every time a button is pressed, I use digitalWrite to set the LED in the opposite position and keep the new ledState.

To continue, you can use the code change code with the fader code from the RGB LED

example for the following:

```
//Button Press Detection - single message

//pin connections
int red = 9;
int green = 10;
int blue = 11;
int buttonPin = 7;

boolean currentState = LOW;//stroage for current button state
boolean lastState = LOW;//storage for last button state
int currentColor = red;//storage for current color

void setup(){
  pinMode(buttonPin, INPUT);//this time we will set the pin as INPUT
  pinMode(red, OUTPUT);
  pinMode(blue, OUTPUT);
  pinMode(green, OUTPUT);
  Serial.begin(9600);//initialize Serial connection
  digitalWrite(currentColor, HIGH);//initialize with currentColor on (full brightness)
}

void loop(){
  currentState = digitalRead(buttonPin);
  if (currentState == HIGH && lastState == LOW){//if button has just been pressed
    Serial.println("pressed");
    delay(1);//crude form of button debouncing

    int nextColor = getNextColor(currentColor);
    fader(currentColor, nextColor);
    currentColor = nextColor ;

  }

  lastState = currentState;

}

int getNextColor(int color){//helper function that gives us the next color to fade to
  if (color == red) return green;
  if (color == green) return blue;
  if (color == blue) return red;
}

void fader(int color1, int color2){
    for (int brightness=0;brightness<256;brightness++){
        analogWrite(color1, 255-brightness);
        analogWrite(color2, brightness);
```

```
        delay(2);
    }
}
```

I added an extra assistant function in the code above to help select the next color that will end in:

int getNextColor (int color) {

if (color == red) returns green;
if (color == green) return to blue;
if (color == blue) returns red;

}

I announced this function by int to tell Arduino that he should expect the function to return the number (in this case, the Arduino pin value connected to one of the RGB LED pins.

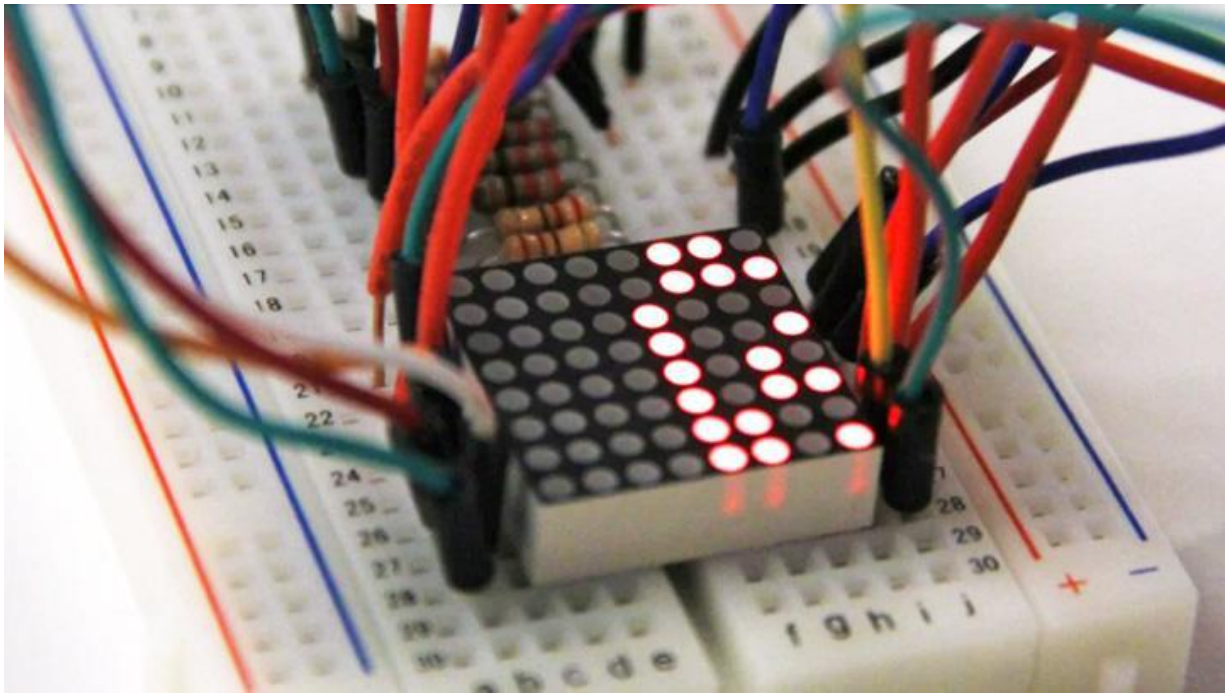
```
int getNextColor(int color){
  if (color == red) {
    return green ;
  }
  if (color == green) {
    return blue;
  }
  if (color == blue) {
    return red;
  }
}
```

and it will work the same way. If you only need to make one line in the if statement, you can use shorthand:

if (something) doSomething;

without curly braces or linebreaks.

Intermediate Arduino: Inputs and Outputs

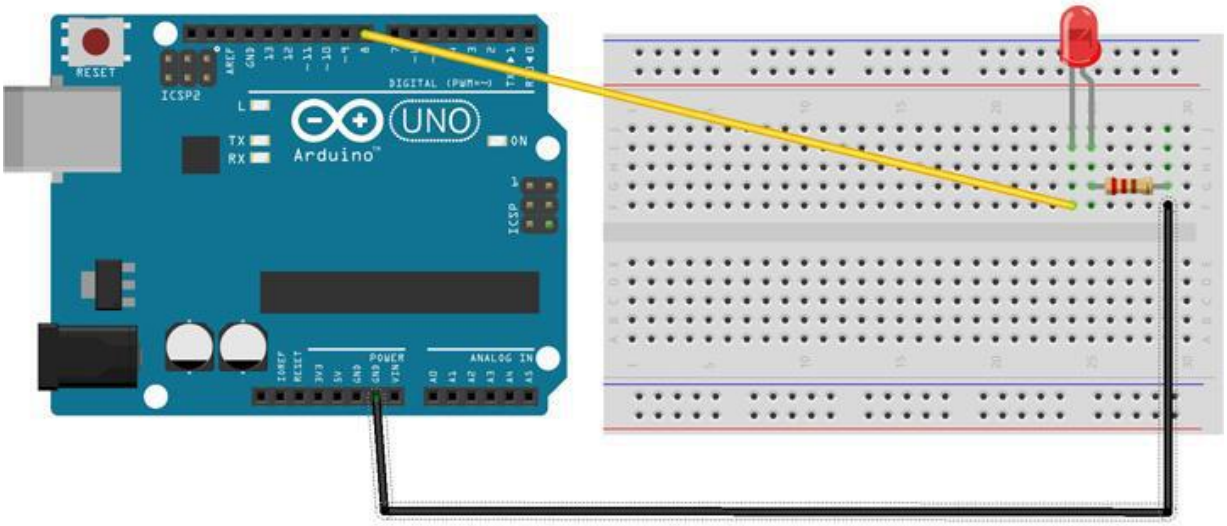


Continuing from my Intro to Arduino post, this Instructable will go over some slightly more advanced topics with Arduino, specifically relating to controlling and managing many inputs and outputs. The next class covers how to connect the Arduino's inputs and outputs to MIDI.

Parts List:

(1x) Arduino Uno
(1x) usb cable
(1x) breadboard (this one comes with jumper wires)
(1x) jumper wires
(8x) red LEDs
(8x) 220Ohm resistors
(1x) 10kOhm resistor
(1x) tact button
(1x) 595 shift register
(1x) red LED dot matrix Adafruit 454

Step 1: Blink Without Delay



So far we have been using the delay function () to pause the Arduino drawing so that the time interval between the two Arduino commands passes. In the LED blink panel, we used the delay to set the Arduino turn on and off:

```
digitalWrite (ledPin, HIGH); // turn on the LED
delay (1000); // wait 1000 milliseconds (one second)
digitalWrite (ledPin, LOW); // turn off the LED
delay (1000); // wait one second
```

Sometimes using a delay () is not a good option because Arduino is unable to perform any second operations while the delay occurs. Suppose we wanted to flash the LED and get a press button simultaneously using the delay ():

```
loop () {

digitalWrite (ledPin, HIGH);
```

```
delay (1000);
digitalWrite (ledPin, LOW);
delay (1000);
booleanState = digitalRead button (7);
}
```

In the code above, we measure the button once every two seconds, so it can take up to two seconds before a button is detected, and very short machines may not be available at all.

millis gives us control over when events occur without pausing in the diagram. Each time we call millis in an Arduino diagram, it returns the number of milliseconds since the Arduino was opened.

Generate the following code to see how millis works:

```
//recording time with Arduino millis()

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

void loop()
{
  unsigned long currentMillis = millis();
  Serial.println(currentMillis);
}
```

Here's how to use a millis to flash an LED without using a delay.

```
//blink led without delay()

int ledPin =  7;

int ledState = LOW;//current state of the LED
unsigned long timeOfLastLedEvent = 0;//the last time the LED was updated

int intervalON = 1000;//how long we want the LED to stay on
int intervalOFF = 500;//how long we want the LED to stay off

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

```
  digitalWrite(ledPin, ledState);
}

void loop() {

  unsigned long currentMillis = millis();

  if (ledState == LOW){//if the LED is already off
    if (currentMillis - timeOfLastLedEvent > intervalOFF){//and enough time has
passed
      digitalWrite(ledPin, HIGH);//turn it on
      ledState = HIGH;//store its current state
      timeOfLastLedEvent = currentMillis;//update the time of this new event
    }
  } else {//if the LED is already on
    if (currentMillis - timeOfLastLedEvent > intervalON){
      digitalWrite(ledPin, LOW);
      ledState = LOW;
      timeOfLastLedEvent = currentMillis;
    }
  }
}
```

The diagram above introduces a few new features:

unsigned long other type of data (so far we have seen int and boolean). Unsigned lengths are the same as int, but larger, I will explain ... Each type of data requires a certain amount of space in Arduino memory, and the amount of Arduino space that we release with a given variable defines min and max values that the variable can store. For example, the int can be from 32,768 to 32,767, if you try to do something like this:

```
int myVariable = 100,000;
```

You will end up with a very strange bug in your code. This may seem like an undeniable range, but it turns out that int requires 16 space in Arduino memory, and with 16 bits of binary you can store numbers from 0 to (2 ^ 16-1) = 65535. But people have decided that The int should be able to store negative numbers as well, so one of the 16-bit numbers is used to store the symbol (good or bad) and the remaining 15 pieces retain the value: 2 ^ 15 = 32768. 0, we store the

width -32,768 to -32,767. Another type of data called ins ins does not save the mark, so it gets 0 to 65535 range I mentioned earlier, but you can't store the wrong number in the ins ins.

When we need to use numbers greater than 65535 or less than 32768, we use a data type called long. Long time allocated 32 pieces of space in Arduino memory. 2 ^ 32 = 4,294,967,296, place this around zero to find the distance: -2,147,483,648 to 2,147,483,647. Unsigned long's, as unsigned int's are always constructive, so they range from 0 to 4,294,967,295.

There is no bigger data type to store numbers than long, so if you need to save a number larger than 4,294,967,295, you will have to come up with a different way of storing it (perhaps the first 9 pieces in one number and the last nine in one?). This limit has positive effects on millis () performance. Since millis returns an unsigned length, and is calculated every millisecond, the millis () will actually return to zero once it reaches:

4,294,967,295 ms

= 4,294,967 seconds

= 49.71 days

If you use millis () and plan to keep the project running for a long time without shutting down or resetting, you should keep this in mind.

One comment about data types: We may have been using the remote or unsigned all this time when announcing pin numbers or other variables in the image to date, but it is usually a good idea to use very small data type to change, thus having more space in Arduino memory for other items . In Arduino, desires are rarely used, but millis () are a good example when they come in handy.

Going back to the diagram, it is a common idea to save one last time when you turn on or off the LED and compare it to the current restored by millis (). When the difference between the two times is

greater than the interval, you know it is time to change the LED again. To do this I set a new storage variable:

```
int ledState = LOW; // current LED status
unsigned longOfLastLedEvent = 0; // last time the LED is updated
int intervalON = 1000; // we want the LED to last longer
int interOFOF = 500; // we want the LED to last longer
```

In the loop () there are a number of views that check to see if enough time has passed, and if so, it changes the LED, regenerates the flexible "timeOfLastLedEvent", and then changes the saved LED status. The mind is doubled, once in the event that the LED is HIGH, and once in the event that the LED is low, I will repeat the LOW case below:

```
if (currentMillis - timeOfLastLedEvent> interval OFF)
{// sufficient time has elapsed

digitalWrite (ledPin, HIGH); // open it
ledState = HIGH; // maintain its current state
timeOfLastLedEvent = currentMillis; // update the timing of this new event

}
```

currentMillis is a long undeveloped term representing the current time renewed each time Arduino's loop () function begins. (currentMillis - timeOfLastLedEvent) provides the time for the LED status to be changed at the end, comparing this with the OFF time to see if it is time to turn off the LED, if not Arduino will continue to update currentMillis and re-check until later.

Step 2: Removal of Arduino button

To proceed from the connect button I introduced in my last Instructable, we can use millis () to make buttons without using delay ():

```
//Button Press Detection - debounce with millis()

int buttonPin = 7;

boolean currentState = LOW;//stroage for current measured button state
boolean lastState = LOW;//storage for last measured button state
boolean debouncedState = LOW;//debounced button state

int debounceInterval = 20;//wait 20 ms for button pin to settle
unsigned long timeOfLastButtonEvent = 0;//store the last time the button state changed

void setup(){
  pinMode(buttonPin, INPUT);//this time we will set the pin as INPUT
  Serial.begin(9600);//initialize Serial connection
}

void loop(){

  currentState = digitalRead(buttonPin);
  unsigned long currentTime = millis();

  if (currentState != lastState){
    timeOfLastButtonEvent = currentTime;
  }

  if (currentTime - timeOfLastButtonEvent > debounceInterval){//if enough time has passed
    if (currentState != debouncedState){//if the current state is still different than our last stored debounced state
      debouncedState = currentState;//update the debounced state

      //trigger an event
      if (debouncedState == HIGH){
        Serial.println("pressed");
      } else {
        Serial.println("released");
      }
    }
  }
```

```
  lastState = currentState;
}
```

In this code, I have added the last new variable:

```
boolean debutedState = DOWN;
int debounceInterval = 20;
unsigned timeOfLastButtonEvent = 0;
```

debouncedState retains the current status of the discarded button, this is the condition that the button is certain of. In contrast, currentState and lastState keep current and final ratings we made with the button, but they do not tell us the status of the button with certainty because it can be affected by a button chat.

debounceInterval the value of ms waiting for the button to be resolved before we know its status. I am my last example using 1ms, here I am using 20ms.

timeOfLastButtonEvent is similar to the TimeOfLastLedEvent time on the last deck, giving the comparison time with currentTime so that we can count how many seconds have passed since the button was first found.

We reset timeOfLastButtonEvent each time the currentState does not match the lastState:

```
for (currentState! = lastState) {

timeOfLastButtonEvent = currentTime;

}
```

When sufficient time has elapsed without the need to reset timeOfLastButtonEvent, we know that the button is sitting in the discarded state:

currentTime - timeOfLastButtonEvent> debounceInterval

We can then update the current dump status if it has changed, and if so, start the event according to the new dump state:

```
for (currentState! = debutedState) {

debutedState = currentState;

if (debutedState == HIGH) {

Serial.println ("pressed");

} more {

Serial.println ("extracted");

}

}
```

Step 3: Shift Registers

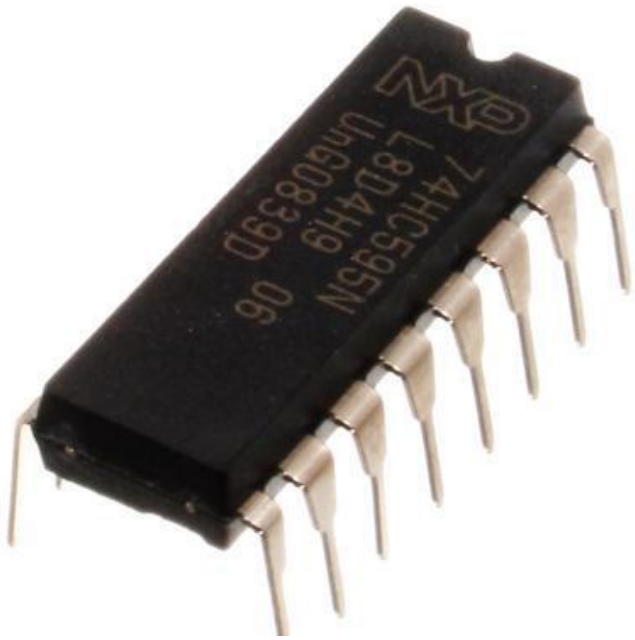
So far we've seen how we can use Arduino to control multiple inputs and outputs for digital at the same time, but sometimes we'll want to control more things than Arduino has its anchors. In this case, we can use an integrated external circuit (also called a "chip") to maximize input and Arduino effects.

Shift chip chips use sensible gates to control multiple inputs or outputs simultaneously. They are digital by nature, like digital anchors in Arduino - this means they can read or write 0V and 5V (LOW or HIGH). If you want something to extend the analog input you will need a demultiplexer like 4051 (learn more about how to use it here). In this quote we will look at the 74HC595 shift register (called "595"), which is very popular because it can extend 3 Arduino output to 8 outputs.

The 595 has 8 output pins labeled Q0-Q7 (sometimes also called Qa-Qh), can't read data from these pins, can only be used as output (if you want to register the switch with 8 input pins, see -74HC165, tutorial here). The 595 is controlled by three connectors, called data pin, latch pin, and pin pin. See the flow diagram above to see how to control the output pins (repeated below):

first, the latch pin (labeled "latch clock" in the second diagram above) is set to LOW to disable the output pins (labeled "corresponding data results"), this way the output pins will not change as we send new data to 595
next, new data is sent to 595 by tapping the clock clock ("shift clock") and then sending each of the eight provinces with "serial data input" one by one. Arduino has an active function in its library called shiftOut which takes care of this, I will explain how you can use this in the next step.
finally, set the latch pin HIGH. This sends your new data to all output pins simultaneously (called the same output).

Step 4: 595 with ShiftOut



Next we will look at a data sheet 595 to find the appropriate connecting pins for them. The first image above shows 595 pin connections. There are 16 pins in 595, with a 1-16 label. Note the half-marking on one side of the chip, the # 1 pin is always located on the left side of the chip. All pins are numbered around the chip to the opposite side of the clock.

The second image shows a pin name, ordered with a pin number, and a brief description. Pines 1-7 and 15 are the results of Q0-Q7, leave those pins unlit for now. Pin 8 is a ground pin, connect this to Arduino ground. Pin 9 is extracted by data extraction, this is used to connect another 595's for daisy chaining. Daisy chaining allows you to drive 16 or more results using three Arduino pins. It's a little out of this lesson, but you can learn more about it here. Since we cannot use pin 9, we can leave it disconnected (also called "floating"). The Pin 10 key reset, when this pin goes DOWN, causes the switch register to be reset - losing any data we could have stored earlier. We don't want this to work right now, so connect a reset to 5V to prevent a reset from happening. Pin 11 is a clock input or "clock clock", connect this to Arduino digital pin 7. Pin 12 is a wiring registry or "latch pin", connect this to Arduino digital pin 6. Pin 13 is a pin that enables the pin, if the bottom allows 595 to send data to its output, we want this, to connect this pin down. Pin 14 is a serial data entry, connect this to Arduino digital pin 5. Pin 16 is the power supply to the chip, connect this to 5V. Your region should look like Figure 3.

Now connect the LED and resistor to the ground on six outlet pins. Your circuit will now look like Figure 4. Now setting the output pin of 595 HIGH will turn on the corresponding LED, then setting it LOW will turn off the LED. Enter the following code:

```
//set 595 state

int clockPin = 7;
int latchPin = 6;
int dataPin = 5;

byte numberToDisplay = 154;


void setup() {
```

```
  //all connections to 595 are outputs
  pinMode(latchPin, OUTPUT);
  pinMode(clockPin, OUTPUT);
  pinMode(dataPin, OUTPUT);
}

void loop() {
    //first set latch pin low so we can shift in data without disrupting the outputs
    digitalWrite(latchPin, LOW);
    // shift out bits of data
    shiftOut(dataPin, clockPin, LSBFIRST, numberToDisplay);
    //set latch pin high to send data to output pins
    digitalWrite(latchPin, HIGH);
}
```

This code introduces a new type of data called byte, byte is the same as an int, but since bytes require only 8 pieces of memory, they store numbers between 0 and 255 (2 ^ 8 = 256).

Some code is straight, out of line:

shiftOut (dataPin, clockPin, LSBFIRST, numberToDisplay);

On line, Arduino uses data and 595 clock pins to send the 154 number (current number ToDisplay) to the switch register. Number 154 contains the provinces of all 8 of the 595 anchors:

154 converted to binary is 10011010

If you check the status of your LEDs, you will see that the LED connected to Q0 is on, Q1 and Q2 are off, Q3 and Q4 are on, Q5 is off, Q6 is on, and Q7 is off. So the LEDs follow the same pattern as the binary number, 1 represents LED and 0 represents closed LED.

Now try some numbers, the number 15 is binary 00001111, you can get another decimal in the google conversion by typing # and then the phrase "to binary" (the output number will start with 0b, ignore that section and hold the last 8 digits) . Keep in mind that we can only send 8-digit binary numbers to them (8-bit) in the exchange

register because it has only 8 output pins, so the number of ToTisDisplay values must be between 0 and 255.

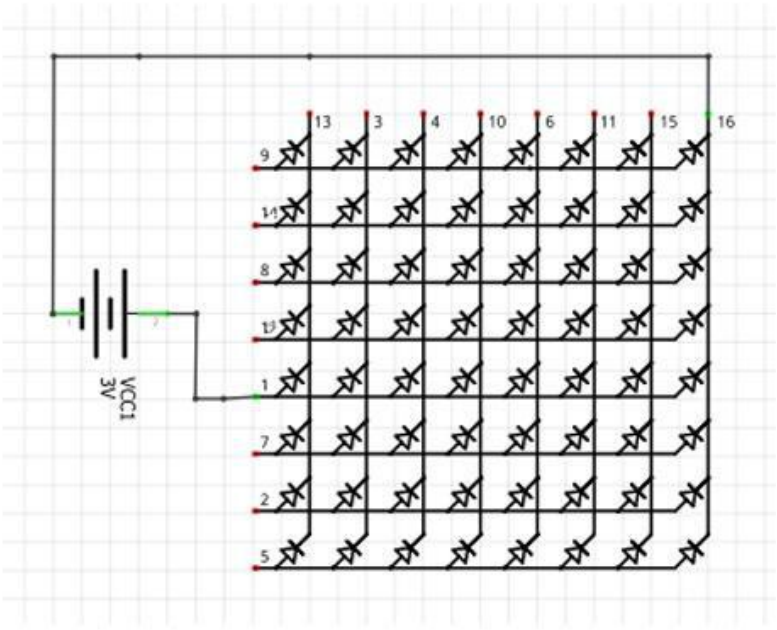
Now try changing the parameter LSBFIRST to MSBFIRST, you should see the LED setting back, this switch sets the indicator that we are sending the binary number to 595: LSBFIRST means "less important first" and MSBFIRST means " most importantly first ".

To make it a little more interesting, try the following:

```
int clockPin = 7;
int latchPin = 6;
int dataPin = 5;

void setup() {
  //all connections to 595 are outputs
  pinMode(latchPin, OUTPUT);
  pinMode(clockPin, OUTPUT);
  pinMode(dataPin, OUTPUT);
}

void loop() {
  for (byte numberToDisplay=0;numberToDisplay<256; numberToDisplay++){
    digitalWrite(latchPin, LOW);
    // shift out bits of data
    shiftOut(dataPin, clockPin, MSBFIRST, numberToDisplay);
    //set latch pin high to send data to output pins
    digitalWrite(latchPin, HIGH);

    delay(500);
  }
}
```

You have now turned your LEDs into a binary counter:

Step 5: Arduino controlled Matrix LED



Next we will look at using Arduino to control an 8x8 LED matrix, a 64-LED grid. The 8x8 matrix we are going to use has 16 connectors: eight pins connect the positive LED track to each column of the matrix, and the other eight pins connect the ground track of all the LEDs to each matrix line. This gives us control over each individual LED. Look at the picture in the second picture above. The image that all the columns are fixed except column 8, connected (with the current fixed resistance) to 5V. There is no image that all lines are connected to 5V except line 1, which is connected to the ground. The only LED that will light up in this case is found in line 1 and column 1.

Place the LED matrix on the bread board as shown in the first picture. Use a current-limit resistor to connect the columns (see pin numbers in the second picture) to 5V and then use a standard jumper wire to connect the lines to the ground. You should see all the LED light light up. Now try unplugging to connect the line to the ground and connect it to 5V instead, all the LEDs in that list will turn off. Try connecting column down, all LEDs in that column will be turned off.

Now disconnect all but one of the connectors from the line pins to the bottom, so only one line of LEDs glows. Instead of connecting the columns to 5V, connect them to Arduino (you are still installing current limiters that limit the region). See the third picture for a better idea of what this should look like. Here's how the columns should be linked to Arduino:

column 1 - Arduino A0 (analog pin 0)
column 2 - Arduino A1
column 3 - Arduino A2
column 4 - Arduino A3
column 5 - Arduino A4
column 6 - Arduino A5
column 7 - Arduino D2 (2-digit digital)
column 8 - Arduino D3

Generate the following code:

```
void setup(){
  //set pins A0-A6 as outputs
  for (int pinNum=A0;pinNum<A6;pinNum++){
    pinMode(pinNum, OUTPUT);
  }
  //set pins 2 and 3 as outputs
  for (int pinNum=2;pinNum<4;pinNum++){
    pinMode(pinNum, OUTPUT);
  }
}

void loop(){
  //some arbitrary set of states
  digitalWrite(A0, HIGH);
  digitalWrite(A1, LOW);
  digitalWrite(A2, HIGH);
  digitalWrite(A3, LOW);
  digitalWrite(A4, HIGH);
  digitalWrite(A5, LOW);
  digitalWrite(2, HIGH);
  digitalWrite(3, LOW);
```

```
}
```

The only unusual thing about this code is that we use analog pins as digital output, this is approved by Arduino. Analog anchors can work as digital inputs and outputs, but they have additional functionality for analog inputs. We're going to use a lot of Arduino pins in this example (a total of 16), so I had to start by assembling some analog pins. Also, I deliberately left 0 pins and 1 with no attachments. Arduino uses these pins to communicate via USB, and sometimes having objects connected to pins 0 and 1 limits your ability to edit the board.

You should see a pattern of LEDs shining on the bottom line. One LED on, one off, one on, one off ... and so on. This pattern is shown in Figure # 3.

Now remove the ground connection from the LED matrix, and then tick a separate line at the bottom. You should see the same pattern on a different line (picture # 4). In the next step we will use Arduino to select each line selectively.

First try one thing, change the pattern on and off the LEDs, here's what I did:

```
void setup(){
  //set pins A0-A6 as outputs
  for (int pinNum=A0;pinNum<A6;pinNum++){
    pinMode(pinNum, OUTPUT);
  }
  //set pins 2 and 3 as outputs
  for (int pinNum=2;pinNum<4;pinNum++){
    pinMode(pinNum, OUTPUT);
  }
}

void loop(){
  //some arbitrary set of different states
  digitalWrite(A0, LOW);
  digitalWrite(A1, HIGH);
  digitalWrite(A2, LOW) ;
  digitalWrite(A3, HIGH);
  digitalWrite(A4, LOW);
```

```
  digitalWrite(A5, HIGH);
  digitalWrite(2, LOW);
  digitalWrite(3, HIGH);
}
```

The output is shown in the last image above.

Step 6: LED Matrix Multiplexing

Now hit all the ground LED matrix pins on Arduino 6-13 pins. Here is the pin connection:

Line 1 - Arduino D6 (6-digit digital pin)
line 2 - Arduino D7
line 3 - Arduino D8
line 4 - Arduino D9
line 5 - Arduino D10
line 6 - Arduino D11
line 7 - Arduino D12
line 8 - Arduino D13

Also use the following code:

```
void setup(){
  //set pins 6-13 as outputs  and initialize HIGH (so all LEDs are off to start)
  for (int pinNum=6;pinNum<14;pinNum++){
    pinMode(pinNum, OUTPUT);
    digitalWrite(pinNum, HIGH);
  }
  //set pins A0-A6 as outputs
  for (int pinNum=A0;pinNum<A6;pinNum++){
    pinMode(pinNum, OUTPUT);
  }
  //set pins 2 and 3 as outputs
  for (int pinNum=2;pinNum<4;pinNum++){
    pinMode(pinNum, OUTPUT) ;
  }
}

void loop(){
  for (int pinNum=6;pinNum<14;pinNum++){
```

```
    //set columns
    digitalWrite(A0, LOW);
    digitalWrite(A1, HIGH);
    digitalWrite(A2, LOW);
    digitalWrite(A3, HIGH);
    digitalWrite(A4, LOW);
    digitalWrite(A5, HIGH);
    digitalWrite(2, LOW);
    digitalWrite(3, HIGH);

    //pulse row low to light LEDs
    digitalWrite(pinNum, LOW);
    delay(500);
    digitalWrite(pinNum, HIGH);
  }
}
```

You should see the same pattern of LEDs illuminating one line at a time (second image) as each line is placed one by one. Now try to reduce the delay, the pattern will go across the lines faster and faster. Now remove the delay completely, you should see all lines highlighted in (what appears to be) at the same time (third image). Here is a video sign (in this case, I use different patterns to illuminate each line, I will explain more about that soon):

This is called multiplexing. Although it may seem like we are lighting up many lines of LEDs at the same time, we know that we are actually lighting up one line at a time, but we are doing it so fast that we deceive our eyes into thinking everything is happening at once. Multiplexing helps in any time you want to control a lot of things with a few pins. Instead of connecting each LED individually, we can duplicate it and reduce the cost and difficulty significantly.

Now try posting different patterns in different lines:

```
void setup(){
  //set pins 6-13 as outputs  and initialize HIGH
  for (int pinNum=6;pinNum<14;pinNum++){
    pinMode(pinNum, OUTPUT);
    digitalWrite(pinNum, HIGH);
  }
  //set pins A0-A6 as outputs
```

```
  for (int pinNum=A0;pinNum<A6;pinNum++){
    pinMode(pinNum, OUTPUT);
  }
  //set pins 2 and 3 as outputs
  for (int pinNum=2;pinNum<4;pinNum++){
    pinMode(pinNum, OUTPUT);
  }
}

void loop(){
  for (int pinNum=6;pinNum<14;pinNum++){

    if (pinNum%2==1){//is pinNum is odd
      //some arbitrary set of states
      digitalWrite(A0, HIGH);
      digitalWrite(A1, LOW);
      digitalWrite(A2, HIGH);
      digitalWrite(A3, LOW);
      digitalWrite(A4, HIGH);
      digitalWrite(A5, LOW);
      digitalWrite(2, HIGH);
      digitalWrite(3, LOW);
    } else {
      //some arbitrary set of different states
      digitalWrite(A0, LOW);
      digitalWrite(A1, HIGH);
      digitalWrite(A2, LOW);
      digitalWrite(A3, HIGH);
      digitalWrite(A4, LOW);
      digitalWrite(A5, HIGH);
      digitalWrite(2, LOW);
      digitalWrite(3, HIGH);
    }

    digitalWrite(pinNum, LOW) ;
    delay(500);
    digitalWrite(pinNum, HIGH);
  }
}
```

To implement this code slightly at the beginning, we see that one line is sent to one pattern, and then the next row is sent to a different

pattern, this alternates across all lines in the matrix. Here's how I do it:

if (pinNum% 2 == 1) {// pinNum is unique

% is called "modulo", which is similar to a split, but instead of returning pinNum / 2, it returns the rest of that function. So if pinNum = 6, pinNum% 2 = 0, because 2 goes into 6 equally. If pinNum = 7, then pinNum% 2 = 1. This gives me an easy way to find out which lines are even and odd numbers. Using this information, I can post different patterns for line exchanges.

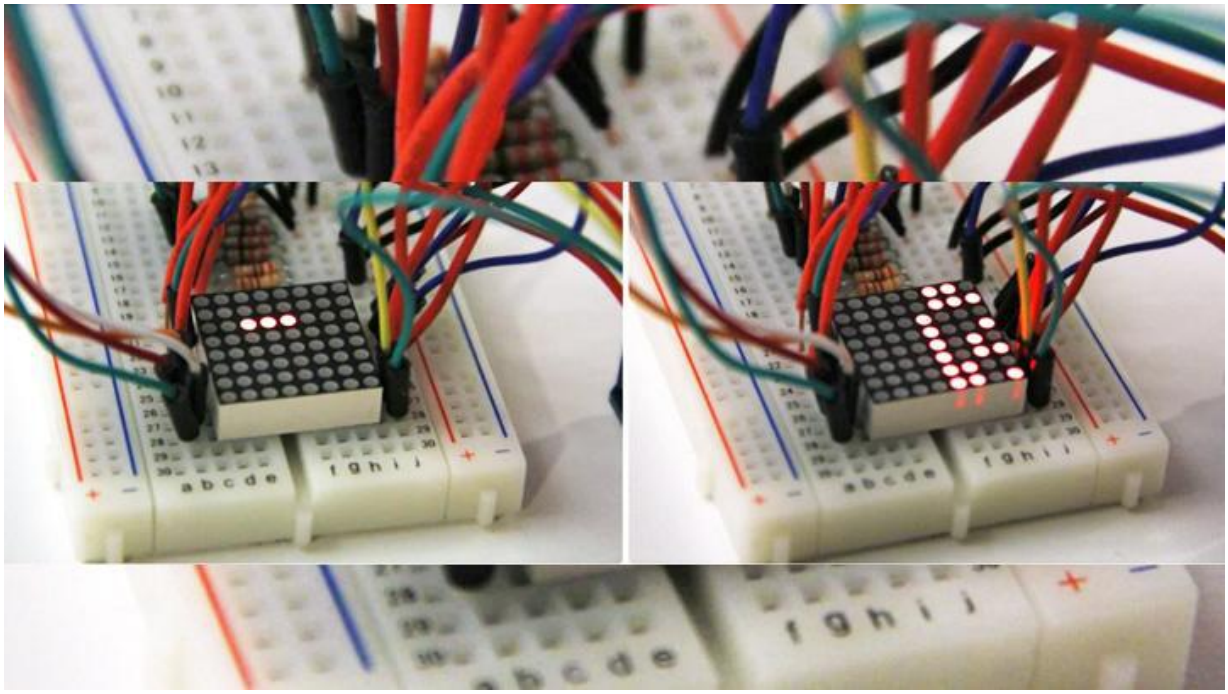
Now try to remove the code delay. You should see a test pattern from the LED matrix (picture four). By sending a different pattern to each line and cycling in individual rows, you can better control which LEDs are on and off. Try the following:

```
void setup(){
  //set pins 6-13 as outputs  and initialize HIGH
  for (int pinNum=6;pinNum<14;pinNum++){
    pinMode(pinNum, OUTPUT);
    digitalWrite(pinNum, HIGH);
  }
  //set pins A0-A6 as outputs
  for (int pinNum=A0;pinNum<A6;pinNum++){
    pinMode(pinNum, OUTPUT);
  }
  //set pins 2 and 3 as outputs
  for (int pinNum=2;pinNum<4;pinNum++){
    pinMode(pinNum, OUTPUT);
  }
}

void loop(){
  for (int pinNum=6;pinNum<14;pinNum++){

    if (pinNum==8){
      digitalWrite(A5, HIGH);
    } else {
      digitalWrite(A5, LOW);
    }

    digitalWrite(pinNum, LOW);
```

```
    digitalWrite(pinNum, HIGH);
  }
}
```

This code will set only the A5 HIGH line connected to pin 8. This illuminates only one LED in the matrix (picture five).

Step 7: Sending Bytes to Multiplexed LED Matrix



So far the code we used to send data to LED is hard to write. Includes explicit allocation of HIGH or LOW status to each Arduino pin connected to the column. If we wanted to edit the matrix to show a different pattern for each of the 8 lines, we would have to write a line of multiple code. Here's how to put things together:

```
void setup(){
  //set pins 6-13 as outputs  and initialize HIGH
  for (int pinNum=6;pinNum<14;pinNum++){
    pinMode(pinNum, OUTPUT);
    digitalWrite(pinNum, HIGH);
```

```
  }
  //set pins A0-A6 as outputs
  for (int pinNum=A0;pinNum<A6;pinNum++){
    pinMode(pinNum, OUTPUT);
  }
  //set pins 2 and 3 as outputs
  for (int pinNum=2;pinNum<4;pinNum++){
    pinMode(pinNum, OUTPUT);
  }
}

void loop(){
  for (int pinNum=6;pinNum<14;pinNum++){

    if (pinNum==8){
      setStates(56);
    } else {
      setStates(0);
    }

    digitalWrite(pinNum, LOW);
    digitalWrite(pinNum, HIGH);
  }
}

void setStates(byte states){
  zeroStates();//first turn all pins off
  //look at each bit of binary number and set HIGH if needed
  if (states & 1) digitalWrite(A0, HIGH);//check the first bit (least significant bit )
  if (states & 2) digitalWrite(A1, HIGH);//check the second bit
  if (states & 4) digitalWrite(A2, HIGH);
  if (states & 8) digitalWrite(A3, HIGH);
  if (states & 16) digitalWrite(A4, HIGH);
  if (states & 32) digitalWrite(A5, HIGH);
  if (states & 64) digitalWrite(2, HIGH);
  if (states & 128) digitalWrite(3, HIGH);//check the most significant bit
}

void zeroStates(){
  digitalWrite(A0, LOW);
  digitalWrite(A1, LOW);
  digitalWrite(A2, LOW);
  digitalWrite(A3, LOW);
  digitalWrite(A4, LOW);
  digitalWrite(A5, LOW);
```

```
  digitalWrite(2, LOW);
  digitalWrite(3, LOW);
}
```

Remember how we used a number between 0 and 255 (a byte) to set 8 LED states connected to 595? Now I have added a function called setStates () which allows us to set up 8 LED circuits in each LED matrix line using a byte. The first thing SetStates does is set up all the pins connected to the LOW LED matrix columns to turn off any possible LEDs. It then looks at each 8-digit binary byte using the & operator; if another 1 digit, sets the corresponding PIN HIGH.

The code above places a line connected to an 8-digit digital digit using the number 56. For binary, 56 is represented as:

00111000

And the LED output shown is shown in the first image above, you can see that each 1 in the binary number is the same as the LED illuminated in the matrix.

Next try to arrange each line in the LED matrix to indicate the digital pin number attached to it:

```
void setup(){
  //set pins 6-13 as outputs  and initialize HIGH
  for (int pinNum=6;pinNum<14;pinNum++){
    pinMode(pinNum, OUTPUT);
    digitalWrite(pinNum, HIGH);
  }
  //set pins A0-A6 as outputs
  for (int pinNum=A0;pinNum<A6;pinNum++){
    pinMode(pinNum, OUTPUT);
  }
  //set pins 2 and 3 as outputs
  for (int pinNum=2;pinNum<4;pinNum++){
    pinMode(pinNum, OUTPUT);
  }
}

void loop(){
  for (int pinNum=6;pinNum<14;pinNum++){
```

```
    setStates(pinNum);

    digitalWrite(pinNum, LOW);
    digitalWrite(pinNum, HIGH);
  }
}

void setStates(byte states){
  zeroStates();//first turn all pins off
  //look at each bit of binary number and set HIGH if need
  if (states & 1) digitalWrite(A0, HIGH);
  if (states & 2) digitalWrite(A1, HIGH);
  if (states & 4) digitalWrite(A2, HIGH);
  if (states & 8) digitalWrite(A3, HIGH);
  if (states & 16) digitalWrite(A4, HIGH);
  if (states & 32) digitalWrite(A5, HIGH);
  if (states & 64) digitalWrite(2, HIGH);
  if (states & 128) digitalWrite(3, HIGH);
}

void zeroStates(){
  digitalWrite(A0, LOW);
  digitalWrite(A1, LOW) ;
  digitalWrite(A2, LOW);
  digitalWrite(A3, LOW);
  digitalWrite(A4, LOW);
  digitalWrite(A5, LOW);
  digitalWrite(2, LOW);
  digitalWrite(3, LOW);
}
```

The output is shown in the second image above.

Step 8: Use Array to store and set the LED Matrix countries

In the last step we saw how using bytes has simplified the process of setting the regions for each row in the LED matrix. We will now look at a strategy to keep that information in order. The same members are the place where we can keep a collection of related variables, in this case, I will be using a larger list to keep the regions of each row in the matrix. Since we represent the regions of each row in a byte

using a byte, and there are 8 lines, I will need a list of 8 bytes to keep the circuits of all 64 LEDs on the LED matrix. Here's how the list is structured

byte ledStates [8] = {0, 15, 0, 0, 0, 84, 0, 0};

I started a lot of lists with a set of bytes, throwing some non-zero bytes to see how they appear in the LED matrix. Here's how the list is used:

```
byte ledStates[8] = {0, 15, 0, 0, 0, 84, 0, 0};

void setup(){
  //set pins 6-13 as outputs  and initialize HIGH
  for (int pinNum=6;pinNum<14;pinNum++){
    pinMode(pinNum, OUTPUT);
    digitalWrite(pinNum, HIGH);
  }
  //set pins A0-A6 as outputs
  for (int pinNum=A0;pinNum<A6;pinNum++){
    pinMode(pinNum, OUTPUT);
  }
  //set pins 2 and 3 as outputs
  for (int pinNum=2;pinNum<4;pinNum++){
    pinMode(pinNum, OUTPUT);
  }
}

void loop(){
  for (int i=0;i<8;i++){

    setStates(ledStates[i]);

    int pinNum = getPinNumForLEDIndex(i);
    digitalWrite(pinNum, LOW);
    digitalWrite(pinNum, HIGH);
  }
}

int getPinNumForLEDIndex(int index){
  return index+6;
}

void setStates(byte states){
  zeroStates();//first turn all pins off
```

```
  //look at each bit of binary number and set HIGH if need
  if (states & 1) digitalWrite(A0, HIGH);
  if (states & 2) digitalWrite(A1, HIGH);
  if (states & 4) digitalWrite(A2, HIGH);
  if (states & 8) digitalWrite(A3, HIGH);
  if (states & 16) digitalWrite(A4, HIGH);
  if (states & 32) digitalWrite(A5, HIGH);
  if (states & 64) digitalWrite(2, HIGH);
  if (states & 128) digitalWrite(3, HIGH);
}

void zeroStates(){
  digitalWrite(A0, LOW);
  digitalWrite(A1, LOW);
  digitalWrite(A2, LOW) ;
  digitalWrite(A3, LOW);
  digitalWrite(A4, LOW);
  digitalWrite(A5, LOW);
  digitalWrite(2, LOW);
  digitalWrite(3, LOW);
}
```

To find variations stored in the same members, use the following syntax:

arrayName [variableIndex]

where variableIndex is a place of diversity for the same members. For example:

ledStates [0] equals 0

ledStates [1] equals 15

because the first item (index = 0) in the array is 0, and the second item (index = 1) in the array is 15.

Since I no longer go beyond pinNums in my loop, I create a help function that switches between the variation of my iterator "i" and the Arduino pin connected to the interest line:

int getPinNumForLEDIndex (int index) {

return index +6;

}

Try switching bytes on ledStates of the same members to see how it affects the output of the matrix. And try changing the variables stored in ledStates as the drawing works, use the following syntax:

ledStates [indexNum] = newValue

You can also set LEDs using bytes marked as follows:

B11011011

B informs Arduino that he has to wait for the binary number, and then the next 8 numbers are 8 digits on the banner you want to show. This makes it slightly easier to design a 2D pattern on an LED matrix. Try using this listing:

byte ledStates [8] = {B0111100, B01000010, B10110101, B10000101, B10000101, B10110101, B01000010, B0111100};

Step 9: Control of LED Matrix With Chip

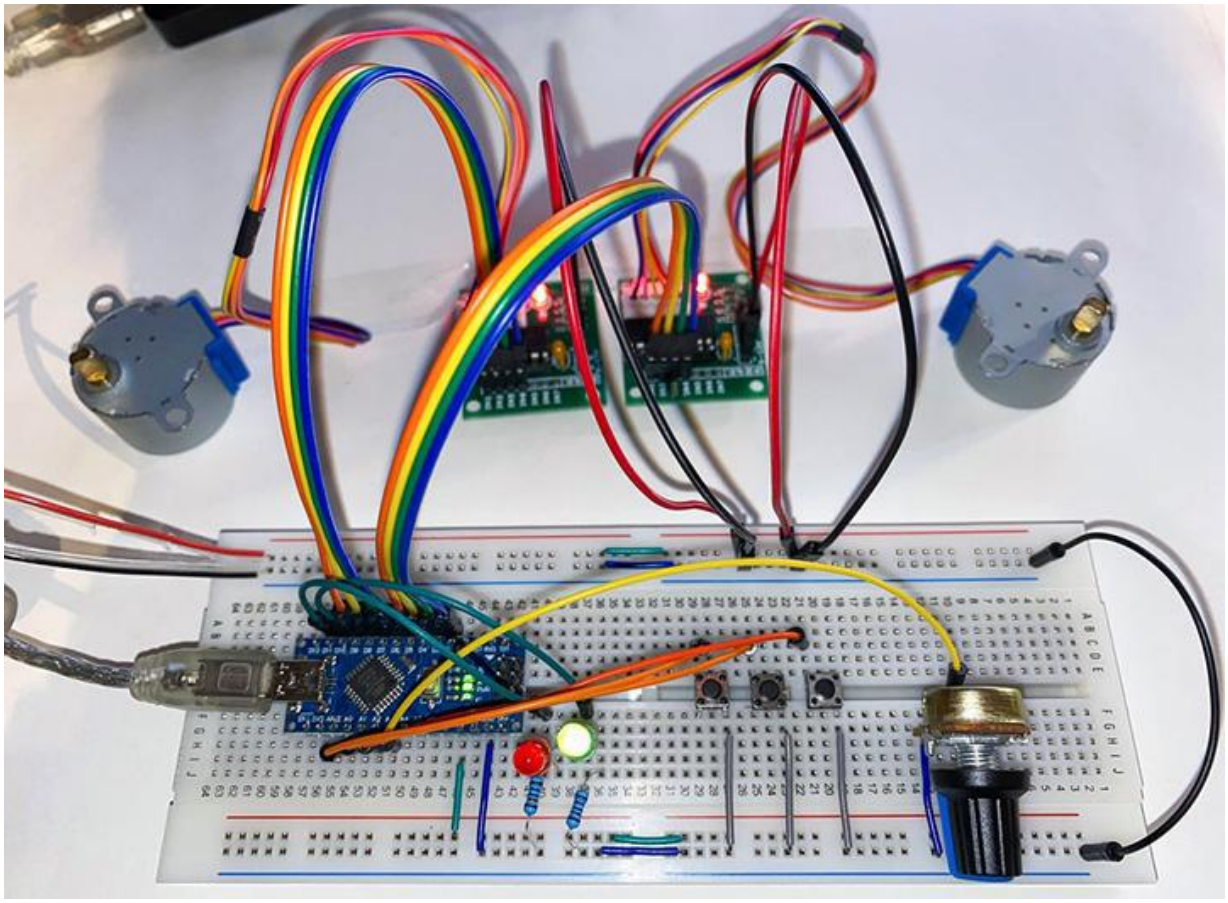
This region uses the tone of Arduino pins so far. In case you need to connect other items to your Arduino, try using 595 code and matrix code to control the matrix using a few Arduino pins (touch-up 595's 8 results in 8 columns in rows 8 LED matrix). You can even use two 595's, one to control the rows and the other to control the columns. One thing you should know is that there are no Arduino or 595 pins capable of providing enough current to drive the entire LED line with full brightness, so you may want to use something like TPIC6B595 (high power register) if you are worried about the light.

The same concept (multiplexing) can be used to control the input grid, such as buttons. I won't go into details on this post, but you can find more details about this here (one small difference between multiplexing buttons vs multiplexing LEDs is that when you add

buttons to the grid, you'll also need to connect the dial with each button - 1N4148 is fine). If you are interested in working with button grids and LEDs, you may want to check out this illuminated button bed and PCB from Sparkfun.

The easiest way I found to control the LED matrix with the MAX7219 chip, basically manages all the multiplexing inside, each chip can control up to 64 LEDs, and requires only three Arduino digital output pins, however at a relatively low cost at $ 11 each. There is a ton of information about using that chip on the Arduino website. MAX7219 will not allow you to adjust the brightness of the LEDs individually, it will only control its turn off / off mode. If you need to control the brightness of most LEDs, you can check out the TLC5940 (Arduino library here), or I agree, it is tricky to duplicate this chip for some time view - that could be an advanced project - but each chip easily controls no LEDs -16 and you can put them together for more control.

Project 01- Motor Variable Speed



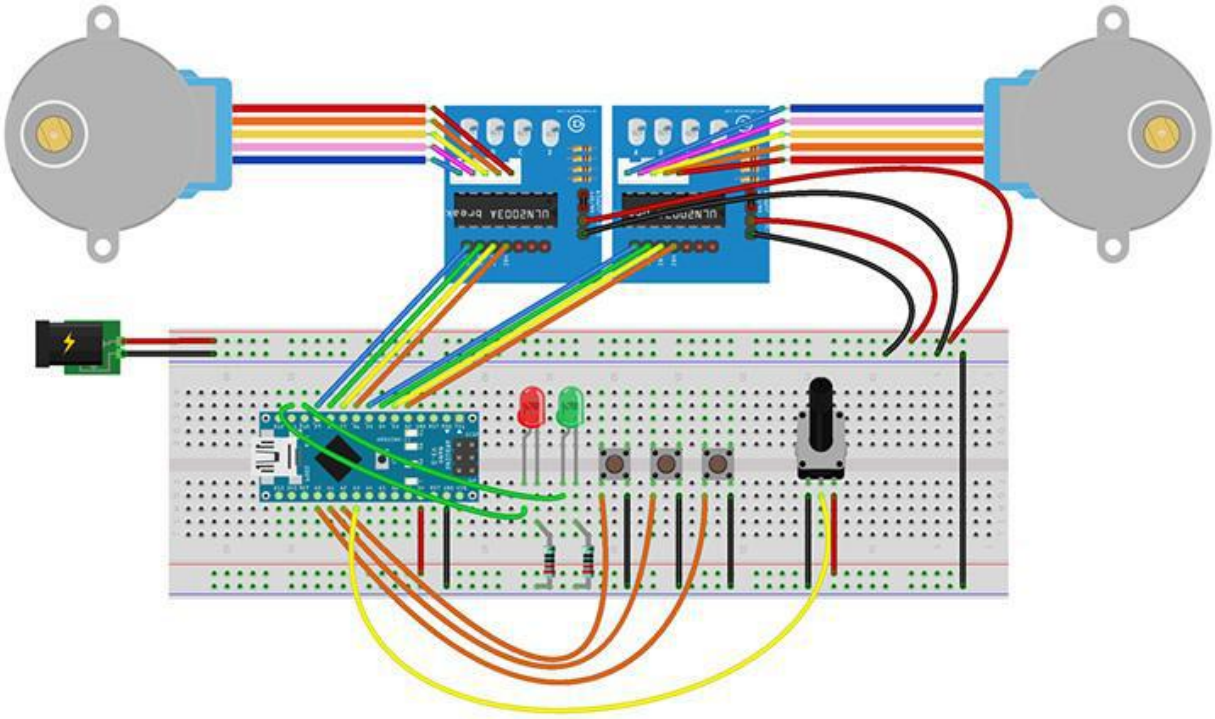
This tutorial shows you how to move stepper motors to get them to a consistent destination. There are many sophisticated solutions to do this such as GRBL or Marlin. Using Arduino, there are also "Multi Stepper" libraries that can support you. However, we will use the basic method here which makes it as easy as possible while adding a simple user interface using three LEDs and other microswitches.

Goods:

1 Arduino Nano (or Uno, with 328P CPU)
1 Bread Board (830 Holes)
14 Jumper Cables (male to male, 10 ... 20 cm)
12 Jumper threads (male to female, 10 ... 20 cm)
2 LEDs (5mm, 20mA, around 2V)
2 opponents (220 Ohm)
2 Microswitches

1 Potentionmeter (5 or 10 kOhm)
1 5V Power Supply
2 28BYJ-48 Stepper Motors (5V Version) + ULN 2003 Driver Boards

Step 1: Prepare Your Breadboard



Set up your bread board as shown in the pictures above:

Connect the GND and 5V rail to the rail according to the bread board.

Arduino pins D2 to D5 to In1 to In4 of the first board of ULN2003.

Arduino pins D6 to D9 to In1 to In4 of the second ULN2003 board.

Arduino's Pin D10 goes to the leg + (long leg) of the first (red) LED. Connect the short leg (-) to the 220 resistor and then the other end of this resistor to the ground rail.

Connect the D11 in the same way as the second LED and its Resistor.

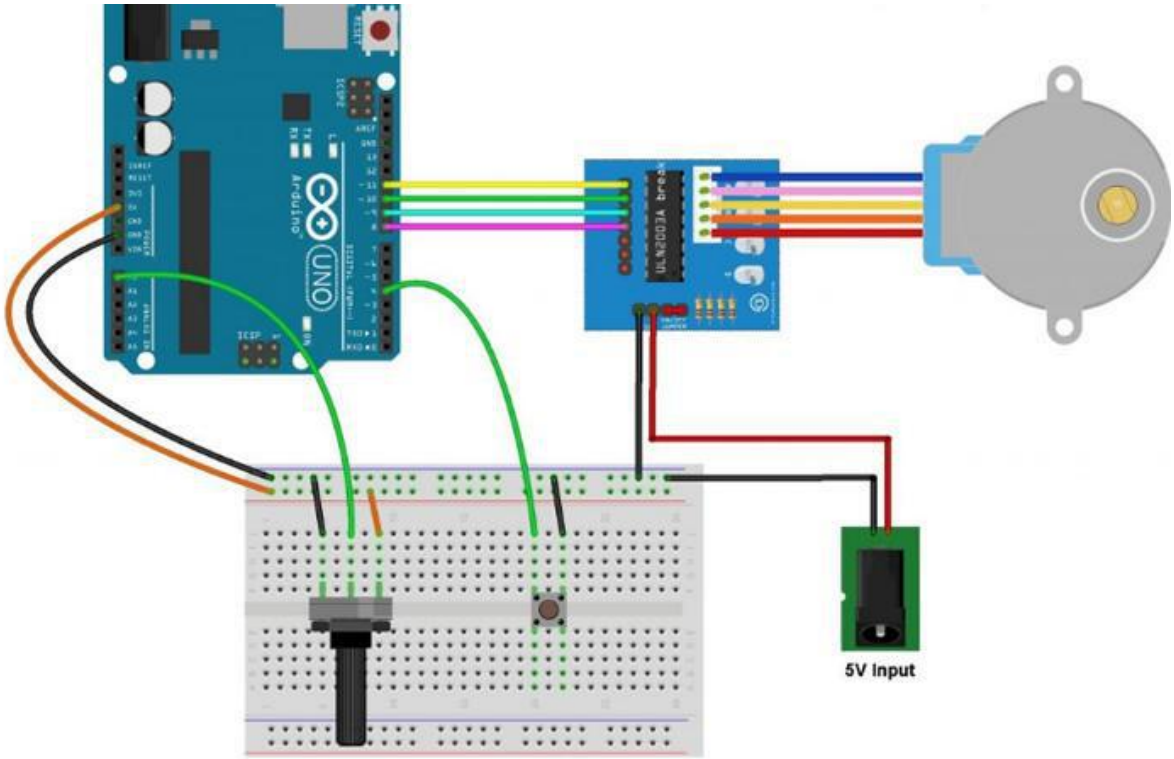
Connect the Arduino A0, A1, and A2 pins to the microswitches. Connect the other side of the microswitches to the GND.

We will not use any pull resistors to avoid short circuits here, because we are using drag already integrated in Arduino.

Connect the left leg of the potentiometer to the ground, the right leg to 5V and the middle A3 Arduino leg.

Arduino itself will be powered by a USB port during initial testing. However, motors should not be powered by a USB port. Therefore, connect the separate power supply to the connecting / ground rail on the other side of the bread board. Connect the pins + and pins of each ULN2003 board to these connectors / ground. As a last resort, build a bridge between the two world tracks.

Step 2: A First Test



We can now perform the first test of our setup with the installed "user interface".

Open the Arduino IDE and upload the file linked to "Two_Steppers _-_ First_Test.ino". It should cover without additional libraries. Most of the code describes itself well.

The setup method only specifies which pins will be used for input and output actions and opens the serial connection with the PC.

The loop method will now proceed continuously in the following order:

Read potentiometer value from analog input, display value range between 0 and 100%, read buttons and print all input items on the serial interface.

Replace one LED and turn off the other.

Using two loops, move each car for about a quarter of a full turn (8 * 128 = 1,024-step-quarter per quarter; a full turn is usually 4,076 steps for 28BYJ-48 motors). During each second loop () operation, steps will be performed straight forward. During another run, the engines will return to the starting position.

```
PORTB = (PORTB & B11111100) | stepPatternB[k];
PORTD = (PORTD & B00000011) | stepPatternD[k];
```

The main idea of moving stepper motors is to turn on and off the coils on the motors using digital output pins. In Arduino, the PORTB and PORTD registers are represented, where D2… 5 (= motor 1) is 2… 5 bits of PORTD, D7… 8 (the first two wires to motor 2) are bits 6… 7 for PORTD and D9… 10 (two second engine cables 2) are 0… 1 pieces of PORTB. Therefore, the lines

“Keep” the current state of all unrelated antenna motors (AND operation) and then apply the output pattern for the next step (half) (OR operation).

28BYJ-48 motors are discussed on many other websites, so more information - if needed - can be found here, here or here.

After compiling and uploading the code, select "Tools → Serial Monitor" in Arduino IDE. You should see running motors, LEDs (on the bread board and ULN2003 boards) flashing. Response to pressing (and holding) / removing microswitches and switching ports will be built into the serial monitor monitor.

However: we are not done yet, as the program is only going back and forth. So let’s make the next impression next.

```
/*
* Unipolar stepper motor speed and direction control with Arduino.
* Full step control.
* This is a free software with NO WARRANTY.
* https://simple-circuit.com/
*/

// include Arduino stepper motor library
#include <Stepper.h>

// change this to the number of steps on your motor
#define STEPS 32

// create an instance of the stepper class, specifying
// the number of steps of the motor and the pins it's
// attached to
Stepper stepper(STEPS, 8, 10, 9, 11);

const int button =  4; // direction control button is connected to Arduino pin 4
const int pot    = A0; // speed control potentiometer is connected to analog pin 0
```

```
void setup()
{
  // configure button pin as input with internal pull up enabled
  pinMode(button, INPUT_PULLUP);
}

int direction_ = 1, speed_ = 0;

void loop()
{
  if ( digitalRead(button) == 0 )  // if button is pressed
    if ( debounce() )  // debounce button signal
    {
      direction_ *= -1;  // reverse direction variable
      while ( debounce() ) ;  // wait for button release
    }

  // read analog value from the potentiometer
  int val = analogRead(pot);

  // map digital value from [0, 1023] to [2, 500]
  // ===> min speed = 2 and max speed = 500 rpm
  if ( speed_ != map(val, 0, 1023, 2, 500) )
  { // if the speed was changed
    speed_ = map(val, 0, 1023, 2, 500);
    // set the speed of the motor
    stepper.setSpeed(speed_);
  }

  // move the stepper motor
```

```
  stepper.step(direction_);

}

// a small function for button debounce
bool debounce()
{
  byte count = 0;
  for(byte i = 0; i < 5; i++) {
    if (digitalRead(button) == 0)
      count++;
    delay(10);
  }
  if(count > 2)  return 1;
  else           return 0;
}
```

Step 3: Motion Vision

Suppose we have stepper motors called "x" and "y". Both start at 0 degrees. The “X” wants to move a thousand steps, while the “y” has to go only 250 steps. Together they have to move at a top speed of 100 steps per second, and they are expected to stop at the same time.

Now, there are two major obstacles:

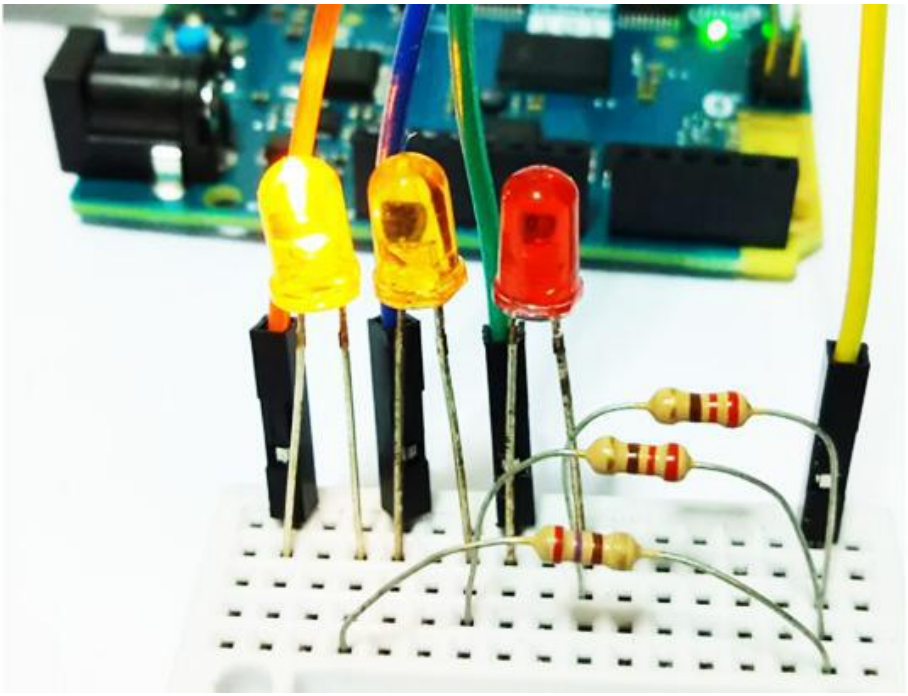
If both are moving at a speed of 100 steps per second, the "y" will only need a quarter of the "x" time. As a result, we use only the “most sought-after speed” in the “x” while reducing the “y” speed to 25 steps per second. Therefore, after 10 seconds both axes will arrive at the same destination. To make this case, you have to always find an axis that has to go a step further, and then

reduce the speed of this axis by
speed (multi-step axis) = speed (user-defined) * steps (few-step axis) / steps (multi-step axis).
If we start both axes at their full speed as calculated in the previous step, the axis will be expected to move at this speed which is not possible due to the inertia and collision of the car, its gears combined with any other motor vehicle. In addition, 28BYJ-48 motors are not really powerful. Therefore, they can easily lose steps if we start at full speed - or worse, integrated gears can be damaged if the load is heavy, because they are made of plastic in most cases. Therefore, we should accelerate each car with each movement. This can be easily accessed using the AccelStepper library as shown in the next step. To compensate for the resulting delays, we apply the above formula not only to the speed but also to the acceleration value of each axis.

Step 4: Final Test and Applications

A demonstration of the completed "machine" is shown in the video above this tutorial where one car travels around a quarter of a cycle while another travels around only one quarter. You can redo this "machine" using the attached mask. Simply print, paste in some cardboard, cut out the marked sides and use line tape to fix the ULN2003 boards and motors on them. As a final step, cut out the arrows and attach / attach them to the car parts.

Project 02 - 3 LED With Arduino 101



Learn how to turn on LEDs using Digital Write and AnalogWrite in Arduino 101!

One of the most popular features of Arduino Basics is how to flash the LED.

So if you are just starting out, you are in the right place!

As you learn this basics, you gain insight into how to configure Arduino IDE and how Arduino and PWM anchors or Pulse Width Modulation work.

Understand LED specifications, the value of resistors and the basic breadth of bread.

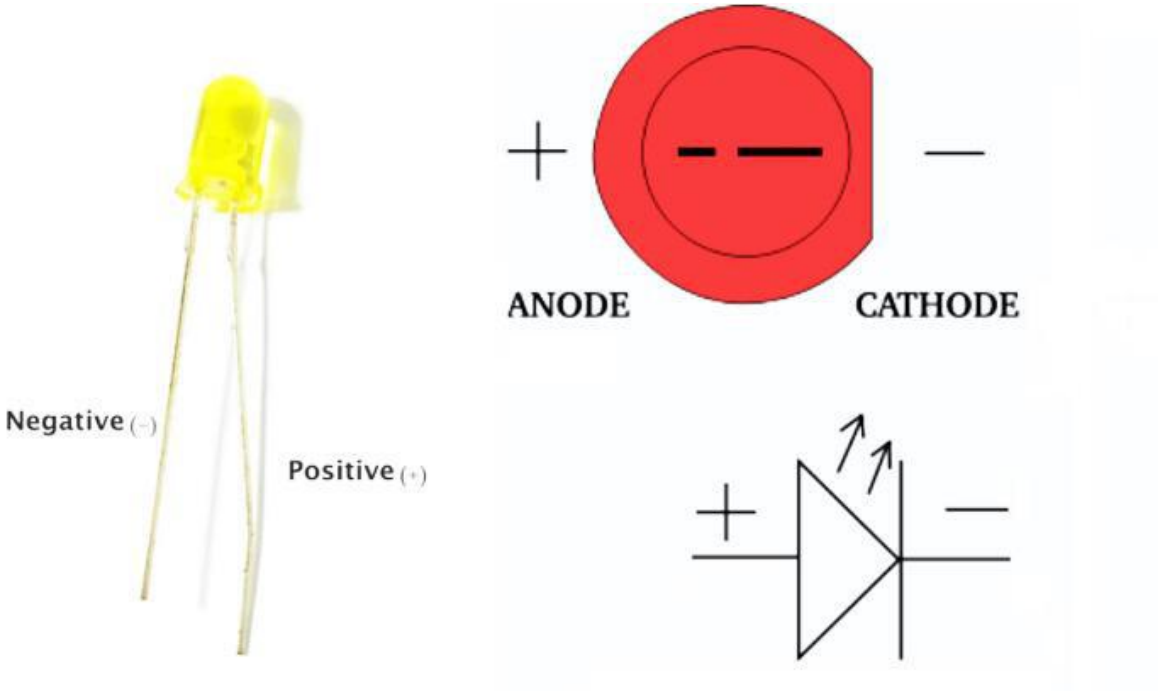
Glad to be an LED rockstar? Let's get started!

Requirements

Arduino Microcontroller
LED
Resistor (220 ohms or less)

Bread board
4pcs Jumper cables
The latest Arduino IDE

Step 1: Know What a LED Is



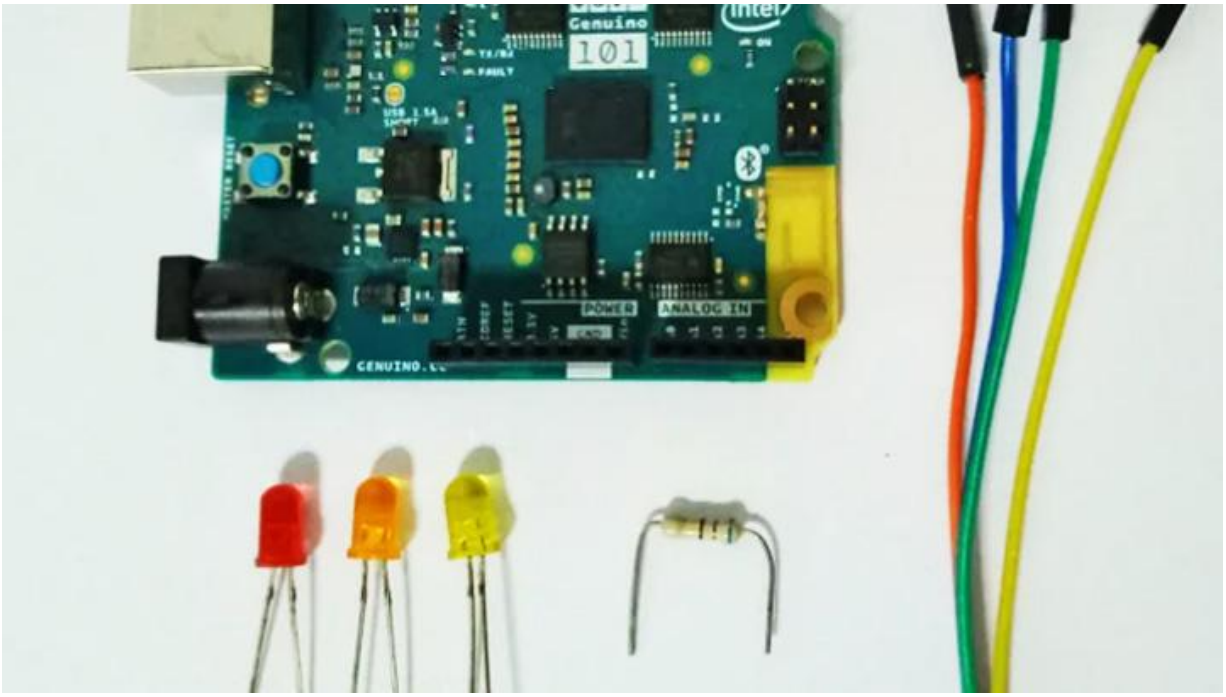
Light Emitting Diode or LED is basically a Diode that can emit light.

Being a diode means having a Cathode pin, a badly charged electrode where electrons enter the electrical field, and an Anode pin, a well-charged electrode where the electrons leave the device.

Take an LED and look !. One leg long and one short. The long leg is Positve (Ano). The short leg is the Cathode.

Same length? No problem brother! Look inside the LED, the largest metal does not have and the smallest goes to the Anode pin

Step 2: Build Our Circuit



I used a small breadboard. It has large divisions or horizontal lines in the middle that separate the connections of the anchors from top to bottom. All the pins from the top are connected to each column. Same goes down.

Connect the Arduino 101 pin GND (ground) to one of the bread board pins. I chose the top left pin.
Insert 1 resistor on the bread board and one pin connected to the jumper wire (in the same column). This means that our resistance is also connected to the ground.
Enter the wrong direction of our lead to the other end of the Resistor (in the same column) and it's right leg in the other column.
Lastly, connect the jumper wire from the Arduino 101 pin 13 to the column where the good LED leg is connected.
To summarize our region,

LED and resistor in series, this means they are connected like a chain
A good LED leg is connected to a digital pin 13
The bad LED leg is attached to the ground
Our opponent will help us reduce the amount of electrical energy to protect the LED from burning. calculated as voltage = current *

resistance.

Step 3: Light Up

Open your Arduino IDE.

Be sure to add the Arduino 101 board when using it. Just open tools> boards> board manager> look for Arduino 101 and install!

Select your Board and Port from the tools again.

Download the attached program file and open it.

Click the arrow icon or the upload button.

And see what happens to the magic! Your LED is flashing! It's scary!

About the code

we set our pin (13) as a removal from setup operation ()
turns on LED using digitalWrite function (ledPin, HIGH)
wait 1000 seconds using the delay function (1000)
turns off LED using digitalWrite (ledPin, LOW)
wait 1000 seconds using the delay function (1000)
and the loop continues forever!
As long as the microcontroller is enabled, it will blink forever.

```
/*
  Blink
  Turns on an LED on for one second, then off for one second, repeatedly.
*/

int ledPin = 13;

// the setup function runs once when you press reset or power the board
void setup() {
  // initialize digital pin ledPin, as an output.
  pinMode(ledPin, OUTPUT);
```

```
}

// the loop function runs over and over again forever
void loop() {
  digitalWrite(ledPin, HIGH);   // turn the LED on (HIGH is the voltage level)
  delay(1000);                  // wait for a second
  digitalWrite(ledPin, LOW);    // turn the LED off by making the voltage LOW
  delay(1000);                  // wait for a second
}
```

Step 4: LED lighting

Changing the light is different.

We will still use our circuit but we have to change PIN 13 to PWM with a powerful PWM.

If you see this symbol ~ 9 or another number it means it has PWM power.

Change the end of the jumper wire from digital pin 13 to pin 9.

Download the attached program file and open it.

Click the arrow icon or the upload button. And see what happens to the magic!

Your LED is flashing at the end of the flash!

The LED is illuminated by changing the power supply provided.

About the code

Add an enlightened state

```
bool onLed = true;
```

Enter light intensity (0 ~ 255)

```
int val = 0;
```

If the onLed condition is true turn up the light and the other goes down

```
if(onLed)
    val++;       // increase brightness
  else
    val--;       //decrease brightness
```

Update LED light

```
analogWrite(ledPin, val);
```

When the light reaches a high value, change the Led mode to false

otherwise if 0 converts Led status to true then wait 1000 miliseconds

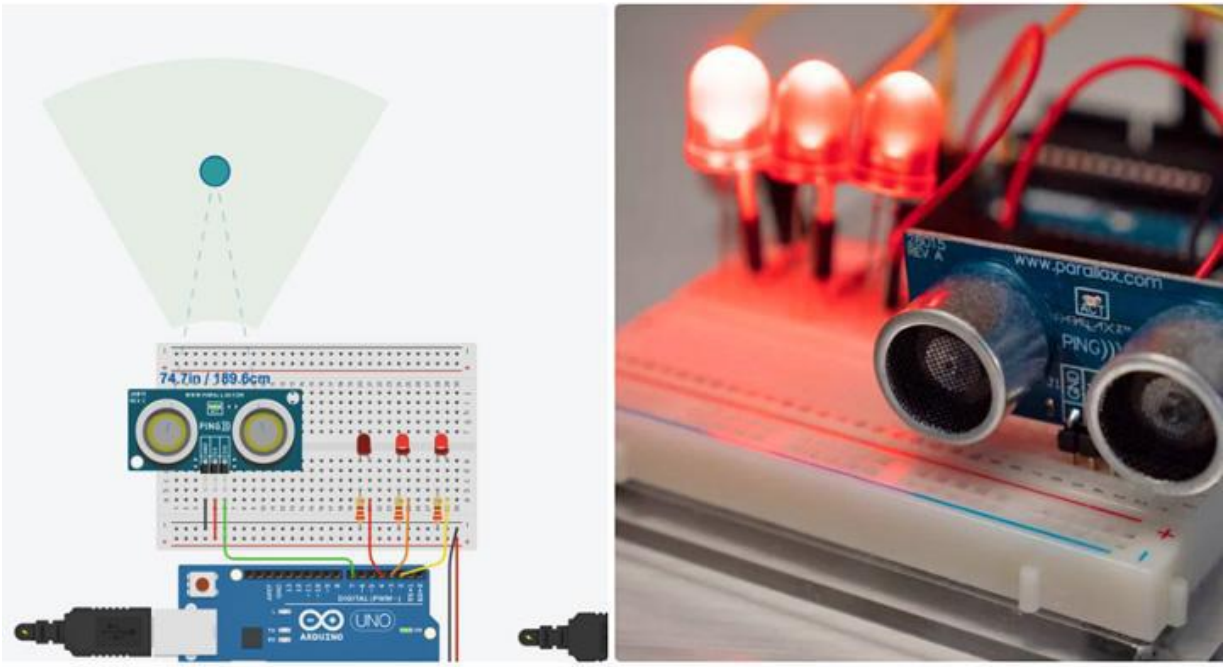
then wait 10 milliseconds

```
if(val==255)
   onLed = false;              // turns off LED
  else if(val==0){
    onLed = true;               // lights on LED
    delay(1000);                // waits for a second before on again
  }
    delay(10);
```

Step 5: More LED Be a Maker

Control what you can do! Insert more LED, add more code and start having fun!

Project 03 - Ultrasonic Distance Sensor in Arduino With Tinkercad



Let's measure distances with an ultrasonic rangefinder (sensor distance) and Arduino's digital input. We will connect the circuit using a breadboard and use a simple Arduino code to control a single LED.

You may have already learned to read pushbutton and PIR motion sensor with Arduino's digital input, and we will build on those skills in this tutorial.

Ultrasonic rangefinders use sound waves to strike objects in front of them, such as bats that use echolocation to sense their environment. The proximity sensor sends a signal and measures how long it takes to return. The Arduino system acquires this information and calculates the distance between the sensor and the object.

Find this region in Tinkercad

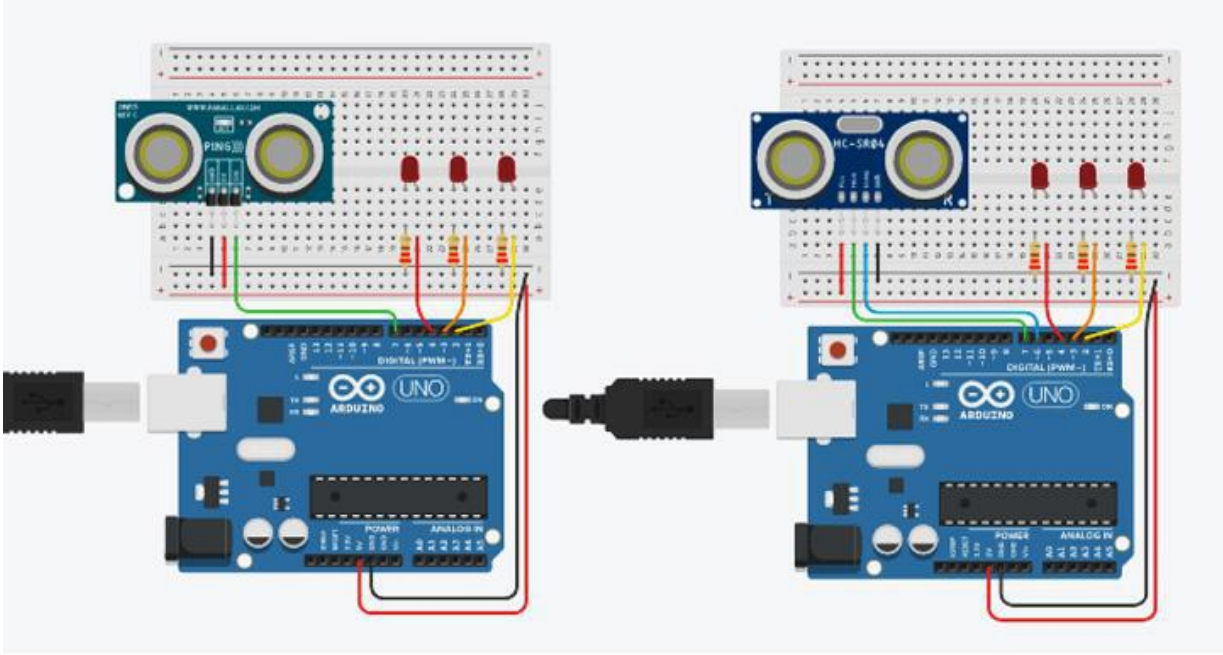
Check the regional sample embedded here by starting the simulation and clicking on the proximity sensor. This will activate the highlighted area in front of the sensor with the "object" circle inside. You may need to expand the view when the circle is closed on the screen.

Click and drag the "object" circle closer and higher, seeing the variable distance values on the screen. Most LEDs will light up when you get close to the sensor.

In this tutorial, you will create this simulated circuit next to the sample. To voluntarily build a virtual circuit, connect your Arduino Uno board, USB cable, solderless breadboard, three LEDs, resistors (any value from 100-1K), ultrasonic rangefinder, and cables of bread board.

You can follow up almost using Tinkercad Circuits. You can watch this tutorial within Tinkercad (free sign-in required)! Examine the regional sample and create your own right next to it. Tinkercad Circuits is a free browser-based program that allows you to create and emulate circuits. It is ideal for learning, teaching and prototyping.

Step 1: Create an LED Circuit



As you learned from the introduction, first connect your Arduino wires to the bread board with power and ground near the circuit

pattern, and then place the three red LEDs on the bread board, as shown. These will be "bar graph" lights to visually display the distance of the sensor.

Drag the Arduino Uno with the bread board from the object panel to the workspace, next to the existing circuit.

Connect 5 volt and ground pins to Arduino power (+) and ground (-) to the breadboard with wires. You can change the phone colors if you want! It can be the use of a checker download or numeric keys on your keyboard.

Drag the three LEDs on the bread board in line E, separated by 2 socket sockets. You can change the color of the LED using the tester that appears when you click on each one.

Use a 220 Ohm resistor to connect each LED cathode (left leg) to the lower (black) rail of the bread board. You can change the value of the resistor by highlighting it and using the drop-down menu.

Connect LED anodes (right legs) to digital pins 4, 3, and 2 in Arduino. The LED (+) anode is the only current flowing through it. This will connect to digital output pins in Arduino. The cathode (-) is the only current flowing through it. This will connect to the subway.

Step 2: Install the proximity sensor

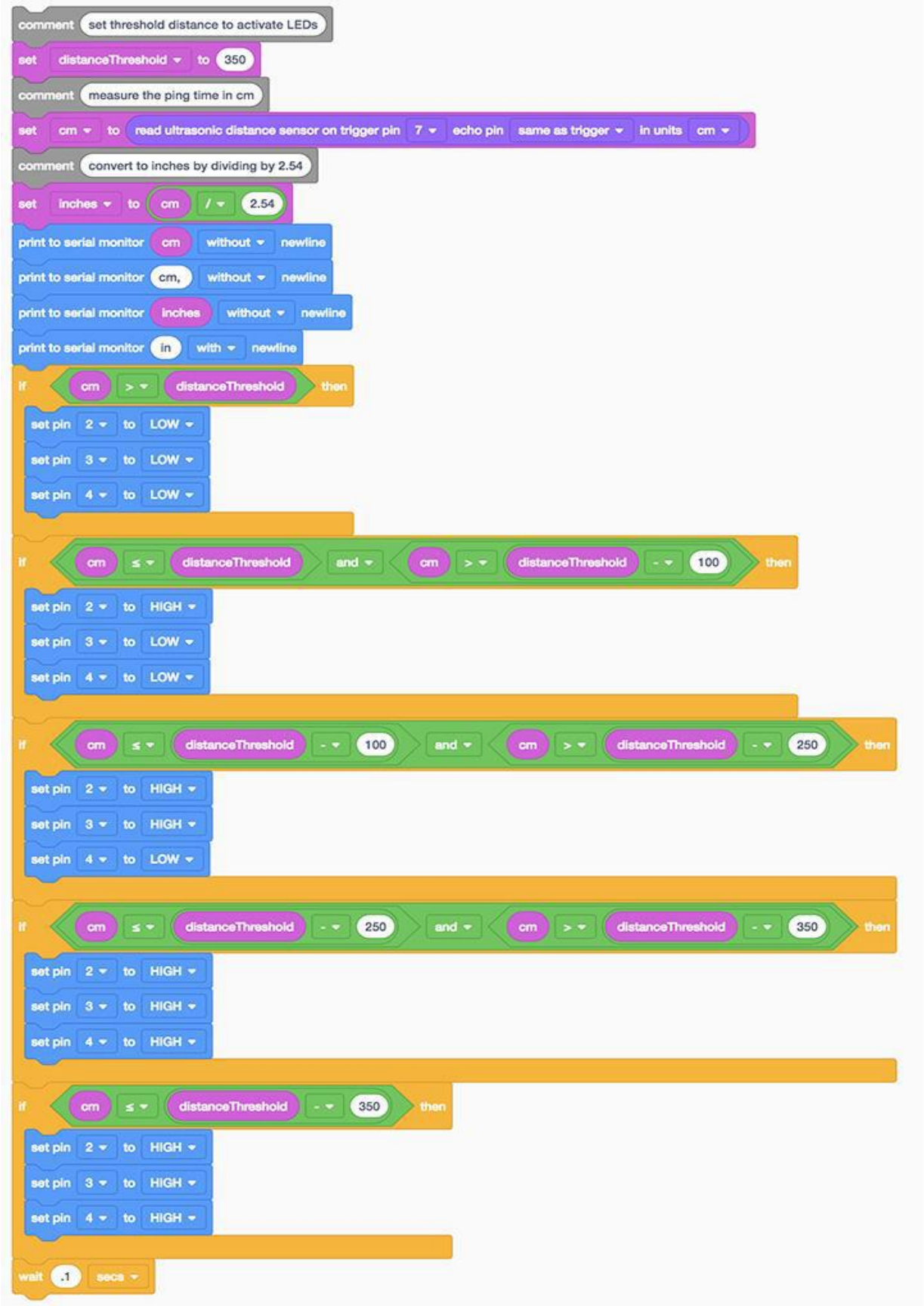
Nearby sensors come in many flavors. Here at Tinkercad Circuits, you can choose between a three-pin marker or a four-pin sensor. Typically, ultrasonic rangefinders have one ground connector, another 5-volt connector, a third signal transmission, and a fourth signal signal. The 'send' and 'receive' pins are combined into a single pin with a three-pin flavor.

In the circuit editor, find the ultrasonic rangefinder in the cabinet. To find the four-pin sensor, view "All" in the sections panel (using the drop-down menu).

Place the sensor on the breadboard to the left of the LEDs in line E, as shown in the picture.

Insert the wire sensor so that the 5V pin connects to the 5V voltage rail, the GND pin connects to the lower rail, the SIG or TRIG pin to Arduino pin 7, and, if you use the four-pin flavor, the ECHO pin connects to Arduino anywhere 6.

Step 3: Blocked Code



Let's use the code block editor to listen to the sensor state, and then make decisions about which LEDs will light up depending on the sensor value.

Click the "Code" button to open the code editor. Gray Notation blocks are comments for writing what you intend for your code to do, but this text does not need or function as part of the program.

Click the Variables section in the code editor. Create a new variable called distanceThreshold and use the "set" block to set it to 350 (inches).

To save the sensor value, create a variable called "cm".

Drag the "set" block and adjust the downtime for our new dynamic cm.

In the installation phase, drag the “read-in-the-range ultrasonic sensor” to the block, and place it inside the set block.

Adjust the drop-down menus within the input block to set the pin pin to 7, the echo pin to be "the same as the trigger" and units up to cm.

Voluntarily create new flexibility to convert inches to inches with set block and arithmetic block reading “set inches to (cm / 2.54)”.

Add serial monitoring blocks to print sensor distance in inches and inches.

Click the Control section and drag when blocking, then navigate to Math and drag the comparison block to block if.

In the Variables section, hold the cm variable and the distanceThreshold variable and drag it to the comparison block, adjust the dropdown to read "if cm> distanceThreshold then".

Insert three digital output blocks inside the if statement to set pins 2, 3, and 4 LOW.

Repeat this if the statement four times and insert the arithmetic blocks and / or blocks to form a complete five state discovery if the statements. The first scenario says “distance is far beyond our limit”

so there are no bright LEDs. When the distance is close to or equal to the distanceThe limit is greater than the distanceThreshold-100, only turn on pin 2's LED. When the temperature is between range Threshold-100 and distanceThreshold-250, turn on two LEDs. And more to answer for all the required provinces.

Step 4: Explaining the Ultrasonic Rangefinder Arduino code

When the code editor is open, you can click the drop-down menu on the left and select "Blocks + Text" to display the Arduino code made of code blocks. Follow along as we examine the code in detail.

```
int distanceThreshold = 0;
int cm = 0;
int inches = 0;
```

Before setting (), we create a variable to keep the target distance limit, and the sensor value in inches (cm) and inches. They are called int because they are whole numbers, or another whole number.

```
long readUltrasonicDistance(int triggerPin, int echoPin)
{
  pinMode(triggerPin, OUTPUT);  // Clear the trigger
  digitalWrite(triggerPin, LOW);
  delayMicroseconds(2);
  // Sets the trigger pin to HIGH state for 10 microseconds
  digitalWrite(triggerPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(triggerPin, LOW);
  pinMode(echoPin, INPUT);
  // Reads the echo pin, and returns the sound wave travel time in microsecond s
  return pulseIn(echoPin, HIGH);
}
```

The next section is a special code for sensing the ultrasonic range sensor. It's called work. So far you are accustomed to setting () and

loop (), but in this diagram, the readUltrasonicDistance () function is used to define sensor code and keep it separate from the main body of the system. The job description starts with what kind of data the job will return, or send back to the main program. In this case the function returns the length, which is the number of multi-digit decimal points. Next up is the job title, which is up to you. After that in brackets there are conflicts the work you take. int triggerPin, int echoPin flexible announcements for your sensor connection pins. The pin numbers will be specified when calling the function in the main loop loop (). Within a function, this local variation is used to refer to information you have transferred from a large loop (or other function). The function itself sends the signal via triggerPin and reports the time it takes to retrieve the signal over echoPin.

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

  pinMode(2, OUTPUT);
  pinMode(3, OUTPUT);
  pinMode(4, OUTPUT);
}
```

Within setup, anchors are configured using the pinMode () function. A serial monitoring connection is established with Serial.begin. Anchors 2, 3, and 4 are configured as LED control effects.

```
void loop()
{
  // set threshold distance to activate LEDs
  distanceThreshold = 350;
  // measure the ping time in cm
  cm = 0.01723 * readUltrasonicDistance(7, 6);
```

For a large loop, the distanceThreshold is set at its target of 350cm.

```
// convert to inches by dividing by 2.54
  inches = (cm / 2.54);
  Serial.print(cm);
  Serial.print("cm, ");
  Serial.print(inches);
  Serial.println("in");
```

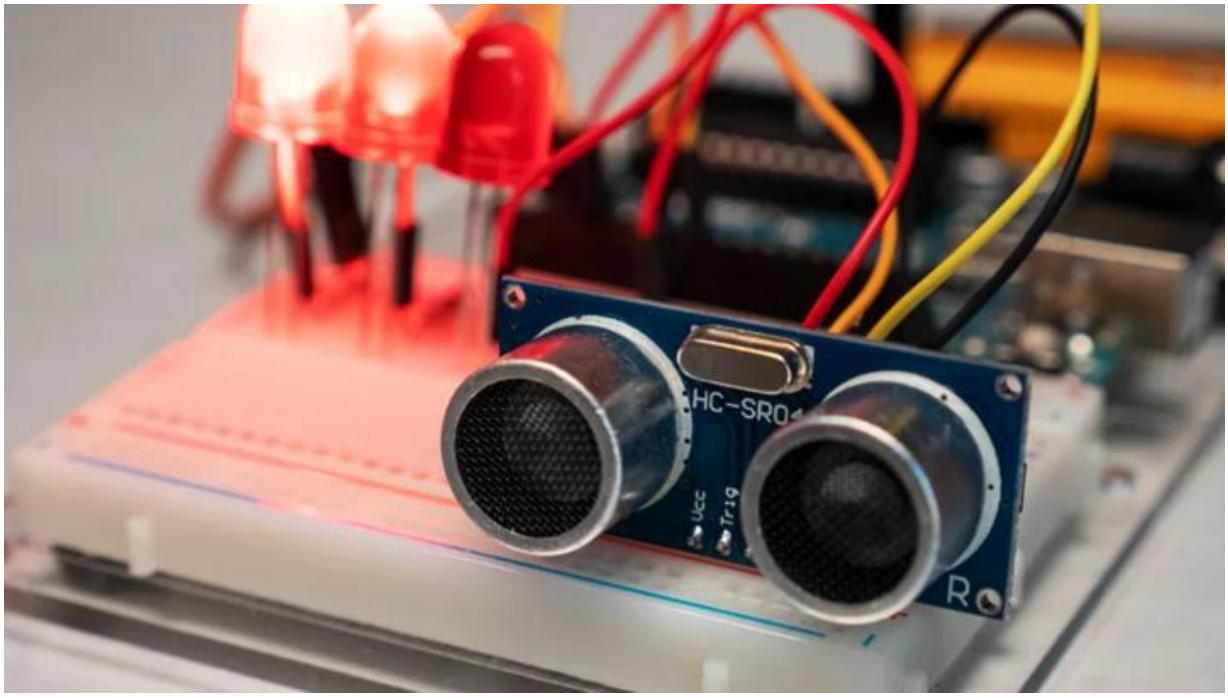
To convert inches to centimeters, divide by 2.54. Printing on a serial track helps you see far more extreme changes than what the LED says.

```
if (cm > distanceThreshold) {
    digitalWrite(2, LOW);
    digitalWrite(3, LOW);
    digitalWrite(4, LOW);
  }
  if (cm <= distanceThreshold && cm > distanceThreshold - 100) {
    digitalWrite(2, HIGH);
    digitalWrite(3, LOW);
    digitalWrite(4, LOW);
  }
  if (cm <= distanceThreshold - 100 && cm > distanceThreshold - 250) {
    digitalWrite(2, HIGH);
    digitalWrite(3, HIGH);
    digitalWrite(4, LOW);
  }
  if (cm <= distanceThreshold - 250 && cm > distanceThreshold - 350) {
    digitalWrite(2, HIGH);
    digitalWrite(3, HIGH);
    digitalWrite(4, HIGH);
  }
  if (cm <= distanceThreshold - 350) {
    digitalWrite(2, HIGH);
    digitalWrite(3, HIGH);
    digitalWrite(4, HIGH);
  }
  delay(100); // Wait for 100 millisecond(s)
}
```

Six loop statements test different distances between 0 and 350cm, illuminating multiple LEDs when an object is too close.

If you want to see the most obvious change in graph bar lights, you can change the variableThreshold distance and / or the view width by changing the arguments in the if () statements. This is called balancing.

Step 5: Create a Body Circuit (Optional)



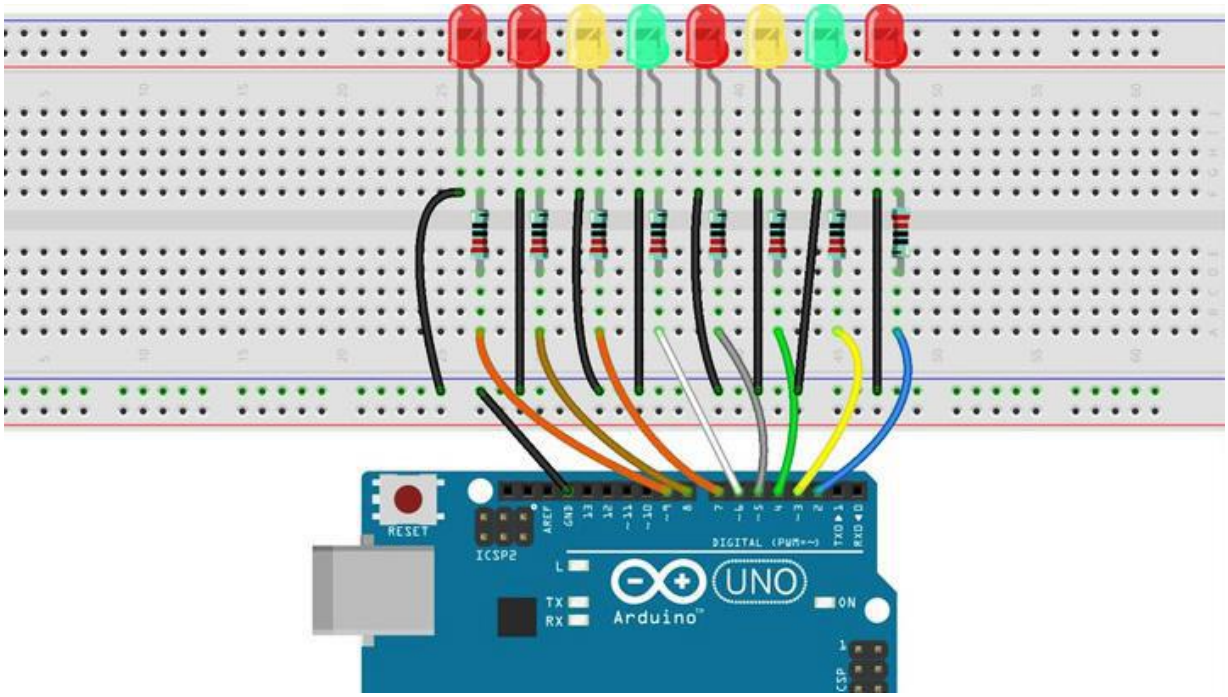
If you are creating a visual version of this region, you can try it with a serial monitor for Arduino software (magnifying the glass button at the top right of the sketch window), activating the sensor with your hand, body, notebook, etc.

When using the body board, place something in front of the sensor and check the distance reading using a serial monitor, and set the distanceThreshold to that value.

Adjust the "buckets" for a different distance to the correct distance for your first value, for example if your hand was 60cm away, your distances could be 60-40, 40-20, and 20-0.

Upload your code again, and try to walk before the sensor. As the distance decreases, you should see the LEDs open one by one.

Project 04 - Flowing LED Lights With Arduino Uno R3



In this study, we will conduct a simple yet fun test - we use LEDs to create flowing LED lights. As the name suggests, these eight consecutive LEDs glow and dim in sequence, like flowing water.

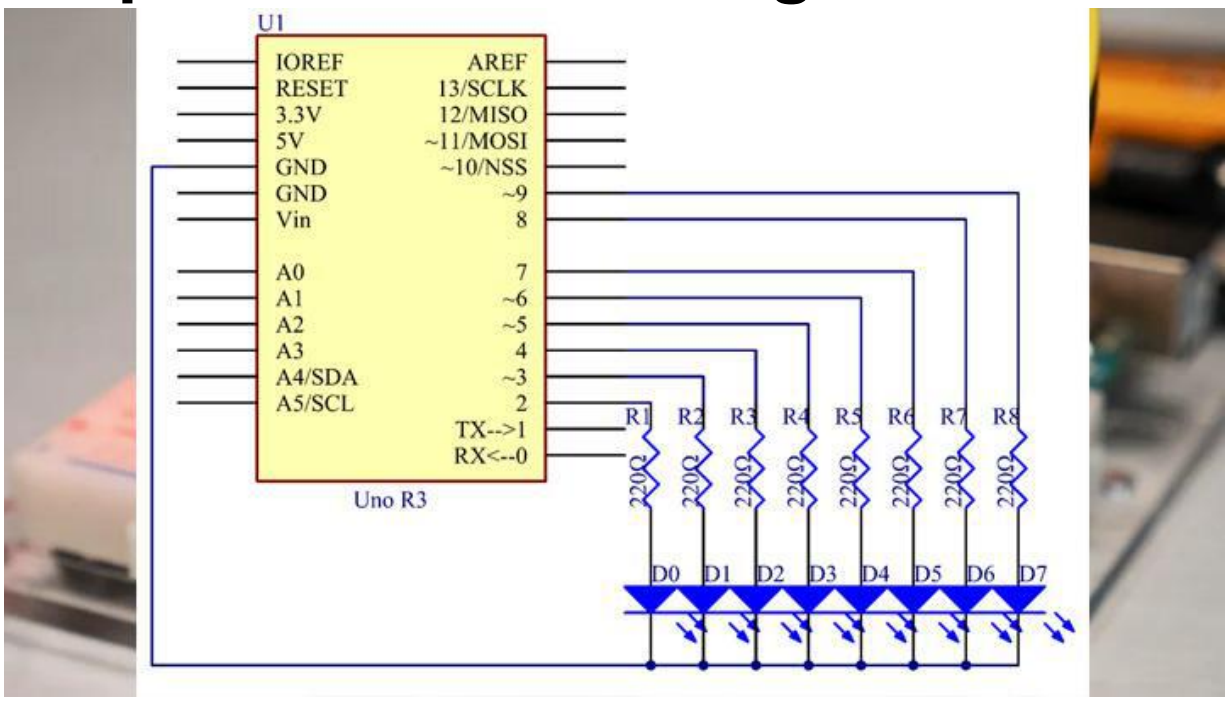
Step 1: Parts

Arduino Board Uno * 1
USB cable * 1
Resistor (220Ω) * 8
LED * 8
Potentiometer * 1
Bread board * 1
Jumping ropes

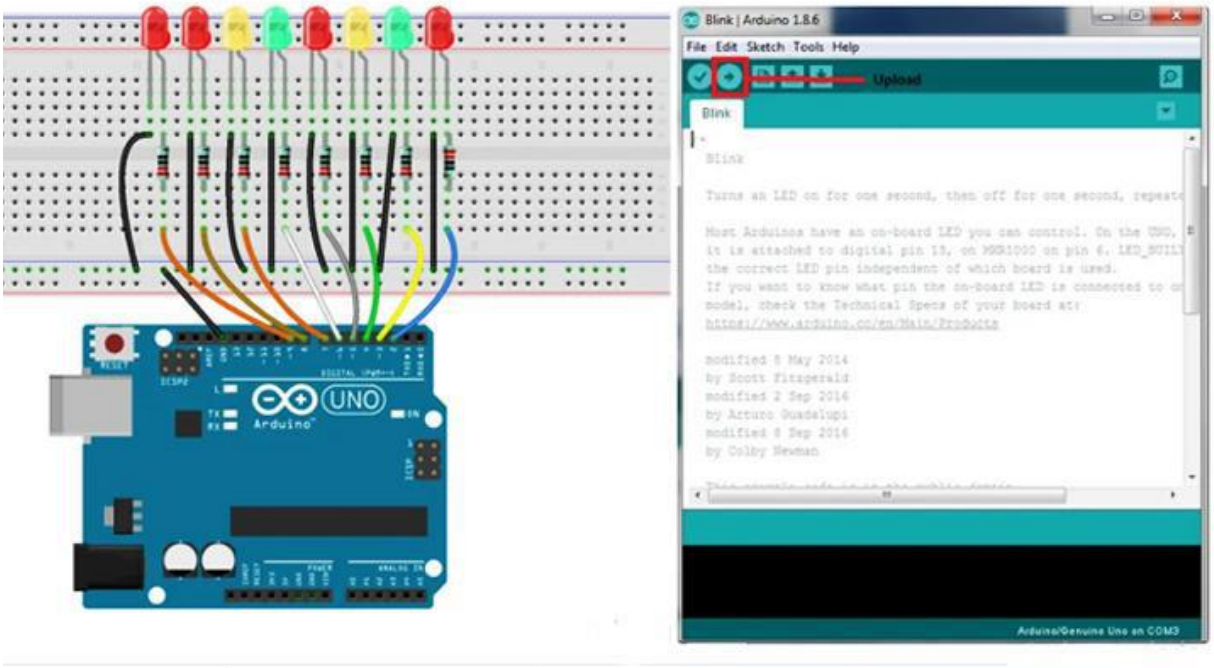
Step 2: Schedule

8 consecutive LEDs light up and dimmer respectively. The goal of this experiment is simply to turn on eight LEDs in a row. Eight LEDs are connected to a 2-pin 9 pin respectively. Set them as high quality and the corresponding LED on the pins will light up. Control each LED light time and you will see flowing LED lights.

Step 3: Formal drawing



Step 4: Procedures



The goal of this experiment is simply to turn on eight LEDs in a row. Eight LEDs are connected to a 2-pin 9 pin respectively. Set them as high quality and the corresponding LED on the pins will light up. Control each LED light time and you will see flowing LED lights.

Step 1:

Create a circuit.

Step 2:

Download code from https://github.com/primerobotics/Arduino

Step 3:

Insert the drawing on the Arduino Uno board

Click the upload icon to upload the code to the control board.

If "Uploaded" appears at the bottom of the window, it means that the drawing has been successfully uploaded.

Now, you should see eight LEDs that light up one by one from the LED connected to pin 2 to point 9, and then blur the rotation from

LED to pin 9 to pin 2. After that, the LEDs will light up from LED pin 9 to pin 2 and blur from LED pin 2 to pin 9. The whole process will be repeated until the circuit is turned off.

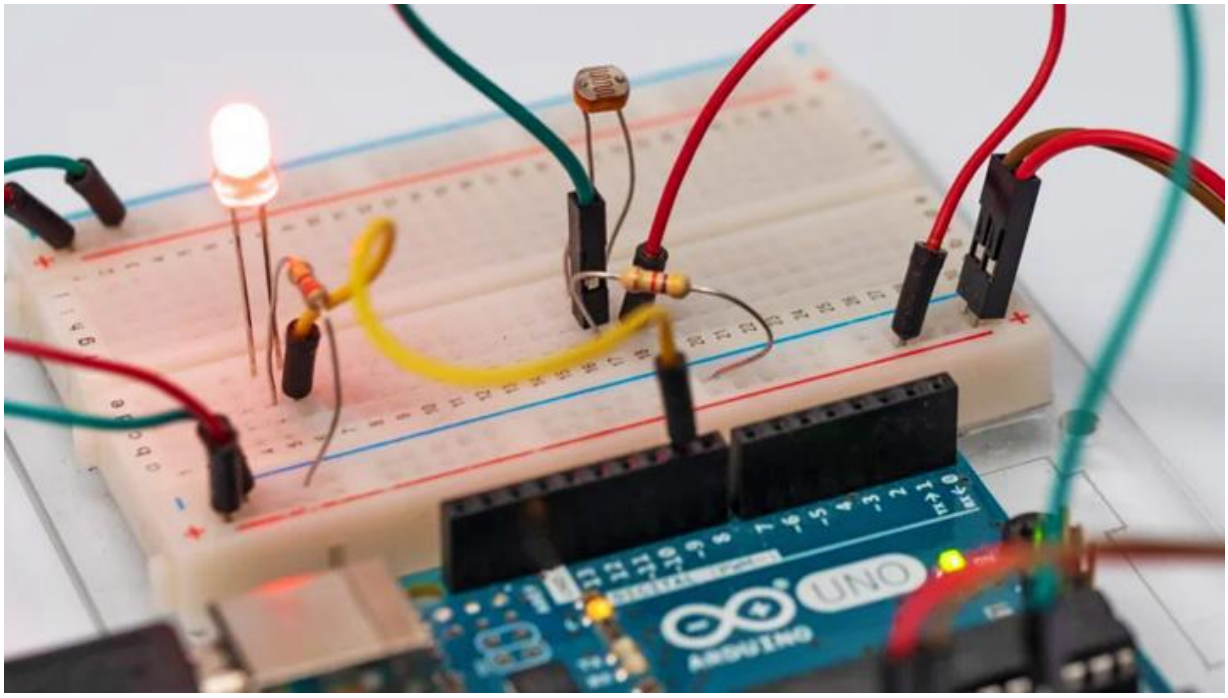
Step 5: Code

```
//Flowing

LED Lights

/*
Eight LEDs will light up one by one from left to right, and then go out one by
one from right to left.

After
that, the LEDs will light up one by one from right to left, and then go out one
by one from left to right.

This
process will repeat indefinitely.*

/Email:info@primerobotics.in

//Website:www.primerobotics.in

/*************************************/

const
int lowestPin = 2;//the lowest one attach to

const
int highestPin = 9;//the highest one attach to

/*************************************/

void
setup()

{
```

```
//set pins 2 through 9 as output

for(int thisPin = lowestPin;thisPin <=
highestPin;thisPin++)

{

    pinMode(thisPin,OUTPUT); //initialize
thisPin as an output

}

}

/**************************************/

void
loop()

{

//iterate over the pins

//turn the led on from lowest to the highest

for(int thisPin = lowestPin;thisPin <=
highestPin;thisPin++)

{

   digitalWrite(thisPin,HIGH);//turn this led
on

   delay(100);//wait for 100 ms

}

//fade from the highest to the lowest

for(int thisPin =
highestPin;thisPin>=lowestPin;thisPin--)
```

```
{
   digitalWrite(thisPin,LOW);//turn this led
off
   delay(100);//wait for 100 ms
}
for(int thisPin = highestPin;thisPin>=lowestPin;thisPin--)
{
   digitalWrite(thisPin,HIGH);//turn this led
on
   delay(100);//wait for 100 ms
}
for(int thisPin = lowestPin;thisPin <=
highestPin;thisPin++)
{
   digitalWrite(thisPin,LOW);//turn this led
off
   delay(100);//wait for 100 ms
}
}
```

Project 05 - Light Sensor (Photoresistor) With Arduino in Tinkercad



Let's learn how to read photoresistor, a light-sensitive type of flexible resistor, using Arduino Analog input. Also called LDR (light-based antagonist).

So far you have already learned to control the LEDs with Arduino analog output, and to read the potentiometer, which has some kind of flexible resistance, so we will build on those skills in this study. Keep in mind that Arduino analog inputs (pins marked A0-A6) can detect a slow-moving electrical signal, and translate that signal to a number between 0 and 1023.

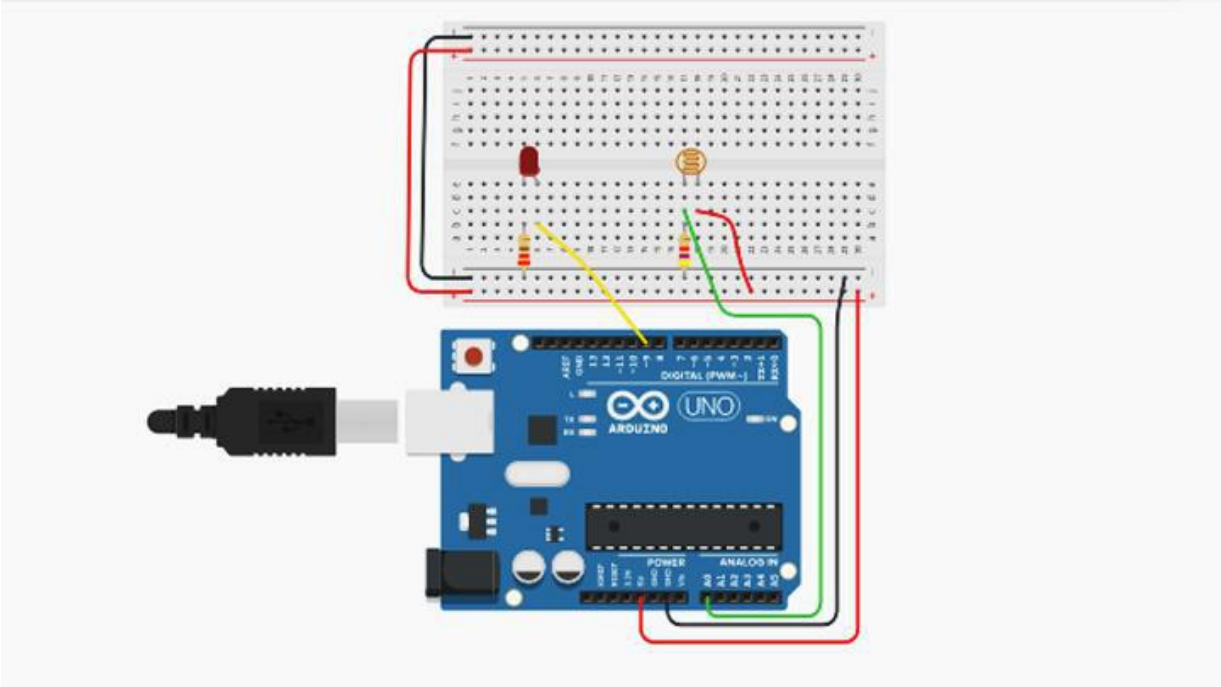
Check the circuit sample attached here on the operating aircraft by clicking on Start Simulation and click on the photoresistor (brown oval with a squiggly line at the bottom in the center), and drag the light slide to adjust the input.

In this tutorial, you will create this simulated circuit next to the sample. To voluntarily build a virtual circuit, connect your Arduino Uno board, USB cable, solderless breadboard, LED, resistors (220 ohm and 4.7k ohm), photoresistor, and breadboard wiring.

You can follow up almost using Tinkercad Circuits. You can watch this tutorial within Tinkercad (free sign-in required)! Examine the regional sample and create your own right next to it. Tinkercad

Circuits is a free browser-based program that allows you to create and emulate circuits. It is ideal for learning, teaching and prototyping.

Step 1: Build the Circuit



Look at the illustrated breadboard circuit. It would be helpful to look at the free cord type of this circuit sample for comparison, illustrated. In this step, you will create your own type of circuit alongside the sample in the operating aircraft.

Next, download the new Tinkercad Circuits window and create your own version of this region next to the sample.

Identify the photoresistor, LED, resistors, and wires connected to the Arduino on the Tinkercad Circuits operating aircraft.

Drag the Arduino Uno with the bread board from the object panel to the workspace, next to the existing circuit.

Connect the power of the breadboard (+) and the rail (-) rail to the Arduino 5V with the ground (GND), respectively, by clicking to create wires.

Extend the power and rail of the earth to their buses along the edge of the bread board (voluntarily in this region but a common practice).

Connect the LED to different loops of bread so that the cathode (negative, short leg) connects to one leg of the resistor (any from 100-1K ohms is fine). The resistor can enter any position because the separate resistors, unlike the LEDs, have to be connected in a certain way in order to work.

Connect the other leg to the ground resistance.

Insert the LED anode (good leg, long leg) into Arduino pin 9.

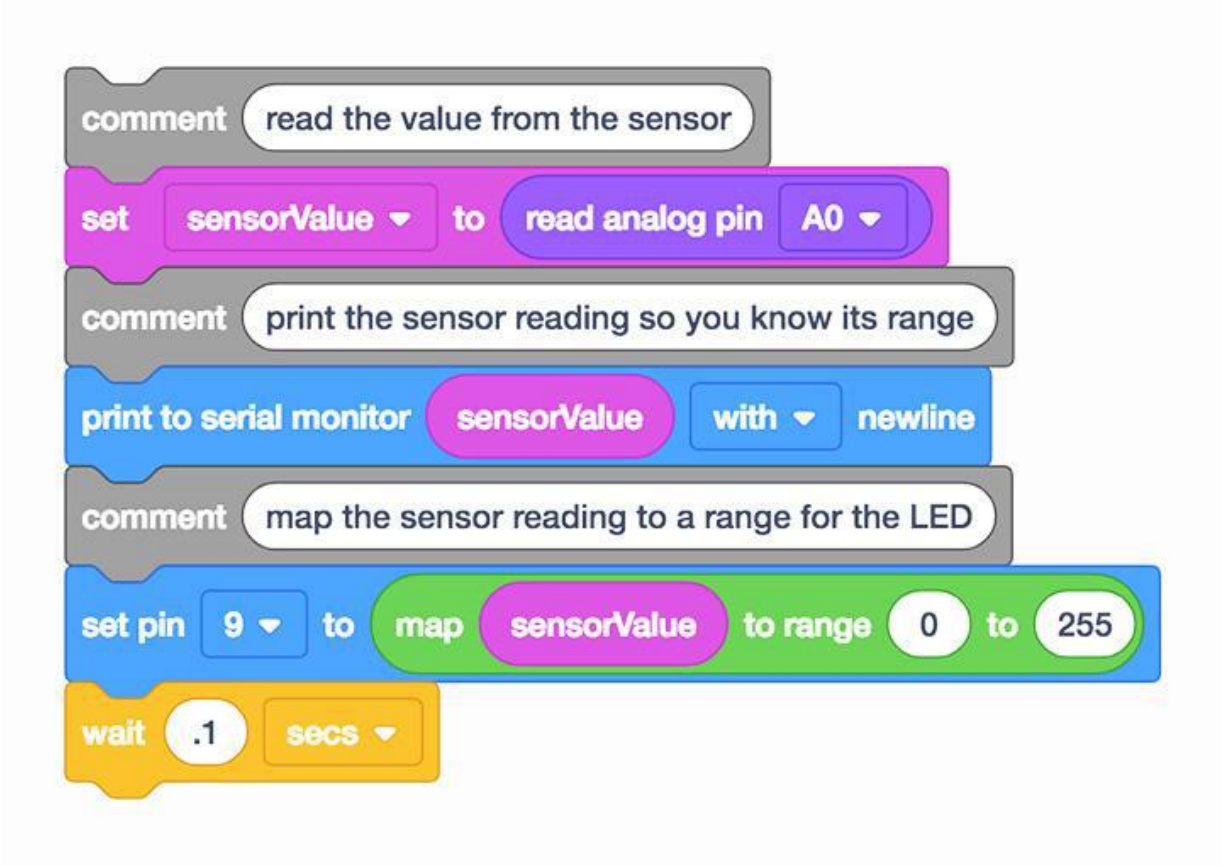
Drag the photoresistor from the component panel to your breadboard, so its legs connect in two separate lines.

Click to create a wire that connects one leg of the photoresistor to power.

Connect the other leg to the Arduino analog pin A0.

Drag the resistor from the component panel to connect the photoresistor leg connected to the A0 through the ground, and adjust its value to 4.7k ohms.

Step 2: Blocked Code



Let's use the code block editor to listen to the position of the photoresistor, and then set the LED to the relative light depending on how much light the sensor sees. You may wish to refresh your memory of the LED analog output in the Fading LED lesson.

Click the "Code" button to open the code editor. Gray Notation blocks are comments for writing what you intend for your code to do, but this text is not included as part of the program.

Click the Variables section in the code editor.

To maintain the photoresistor resistance value, create a variable called "sensorValue".

Drag the "set" block. We will maintain the status of our photoresistor in the flexible sensorValue.

Click on the input section and drag out the "analog pin" block, then place it in the "set" block after the word "to"

Since our potentiometer is connected to the Arduino on pin A0, change the drop to A0.

Click the output section and drag out the "print to monitor serial" block

Navigate to the Variables section and drag your dynamic Value sensor to the "Print to monitor serial" block, and make sure the drop-down is set to print with a new line. Voluntarily start the simulation and turn on the serial monitor to make sure the reading is in and out as you adjust the sensor. Analog input values range from 0-1023.

Since we want to write to the LED with a number between 0 (off) and 255 (full light), we will use the "map" block for some duplication. Navigate to the Math section and drag out the "map" block.

In the first space, drag to blockValue block, then set the distance from 0 to 255.

Back in the removal section, drag the analog "set pin" block, which automatically says "set pin 3 to 0." Adjust it to set pin 9.

Drag the map block you created earlier in the "set pin" block to "to" field to write the converted number to the LED pin using PWM.

Click the Control section and drag out the wait block, then adjust it to delay the program for 1 seconds.

Step 3: Photoresistor Arduino Code explained

When the code editor is open, you can click the drop-down menu on the left and select "Blocks + Text" to display the Arduino code made of code blocks. Follow along as we examine the code in detail.

```
int sensorValue = 0;
```

Before setting (), we create a variable to store the read current value from the potentiometer. It is called an int because it is a whole number or another whole number.

```
void setup()
{
  pinMode(A0, INPUT);
  pinMode(9, OUTPUT);
  Serial.begin(9600);
}
```

Within setup, anchors are configured using the pinMode () function. Pin A0 is configured as an input, so we can "listen" to the electrical status of the potentiometer. Pin 9 is configured as an LED control switch. In order to send messages, Arduino opens a new serial communication channel. Start (), which takes the baud rate (any connection speed), in this case, 9600 bits per second.

```
void loop()
{
  // read the value from the sensor
  sensorValue = analogRead(A0);
  // print the sensor reading so you know its range
  Serial.println(sensorValue);
```

Anything behind a collection of slashes // comments, which helps people understand in clear language what the program is intended to do, but is not included in your Arduino running program. In a large loop, a function called analogRead (); checks the position of pin A0 (which will be a whole number from 0-1023), and stores that value in the variable sensorValue.

```
// map the sensor reading to a range for the LED
  analogWrite(9, map(sensorValue, 0, 1023, 0, 255));
  delay(100); // Wait for 100 millisecond(s)
}
```

The line that follows the next comment does a lot at once. Remember analogWrite () takes two arguments, the pin number (9 of us), and the write value, which should be between 0 and 255. The performance line in line () captures five issues: the number to test (constant- to change the sensitivity), the minimum expected and the maximum size, and the required dates and sizes. So the map function () for us checks the incoming sensorValue, and performs certain repetitions to output output from 0-1023 to 0-255. The effect

is returned to the second analogWrite () ;, set the brightness of the LED connected to pin 9.

Step 4: Create a Physical Arduino Circuit (optional)

To customize your Arduino Uno, you will need to install free software (or a web editor plugin) and open it. Different photocelles have different values, so if your physical circuit is not working, you may need to replace the Paired Resistor. Learn more about power dividers in the Instructables Electronics Class study on resistors.

Connect the Arduino Uno circuit by inserting parts and wires to match the connections shown here in the Tinkercad Circuits. To get deeper into working with your Arduino Uno board, check out the free section of Arduino Instructables.

Copy the code from the Tinkercad Circuits code window and paste it into a blank diagram in your Arduino software, or click the download button (down arrow) and open the file using Arduino. navigate to file -> Examples -> 03.Analog -> AnalogInOutSerial.

Connect your USB cable and select your board and hole in the software tools menu.

Download the code and use your hand to cover the sensor for light, and / or turn on your sensor light!

Turn on serial monitoring to see the values of your sensor. Real world values cannot be expanded to 0 or up to 1023, depending on your lighting conditions. Feel free to adjust the range 0-1023 to the minimum you have seen and set it up too far to get the range of maximum brightness on the LED.

Step 5: Next, Try …

Now that you have learned to read the photoresistor and map out its output to control the LED light, you are ready to use those and other

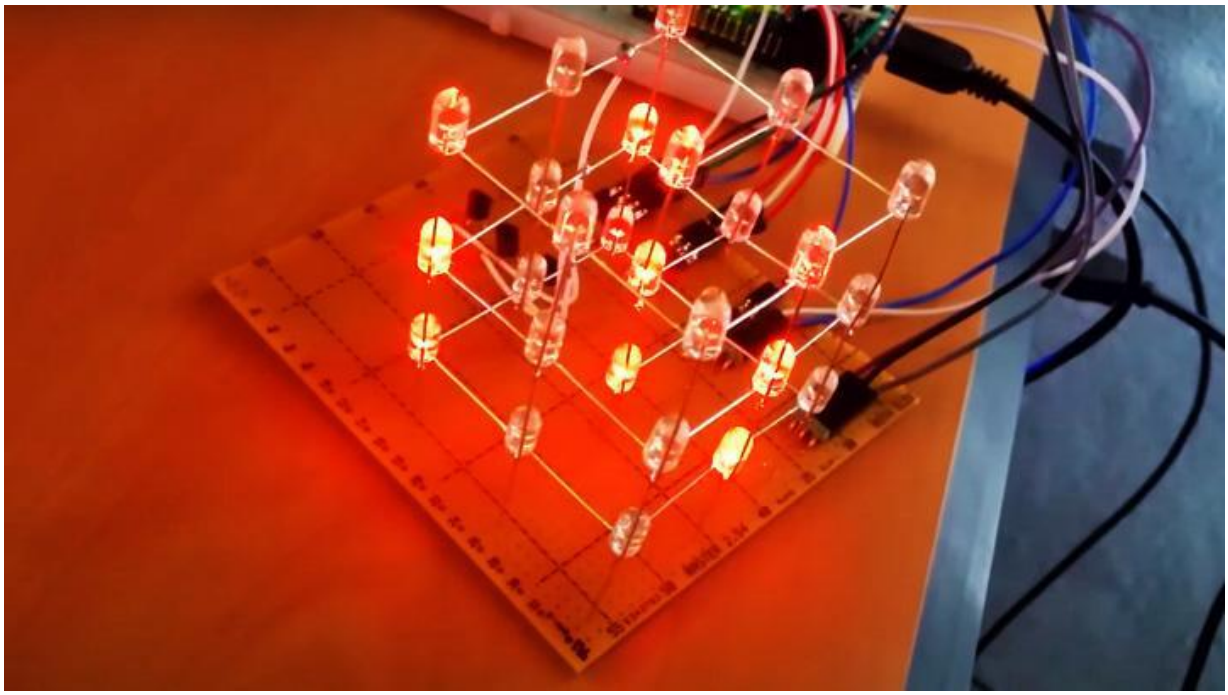
skills you have learned so far.

Can you switch the LED on to another type of output, such as a servo motor, and make a certain code to indicate the level of current light sensing as a specific position next to the gauge?

Try switching your photoresistor to other analog inputs such as the ultrasonic distance sensor or potentiometer.

Learn more about how to monitor Arduino digital and analog input using a computer using Serial Monitor.

Project 06 - DIY | 3x3x3 LED Cube for Arduino Nano+

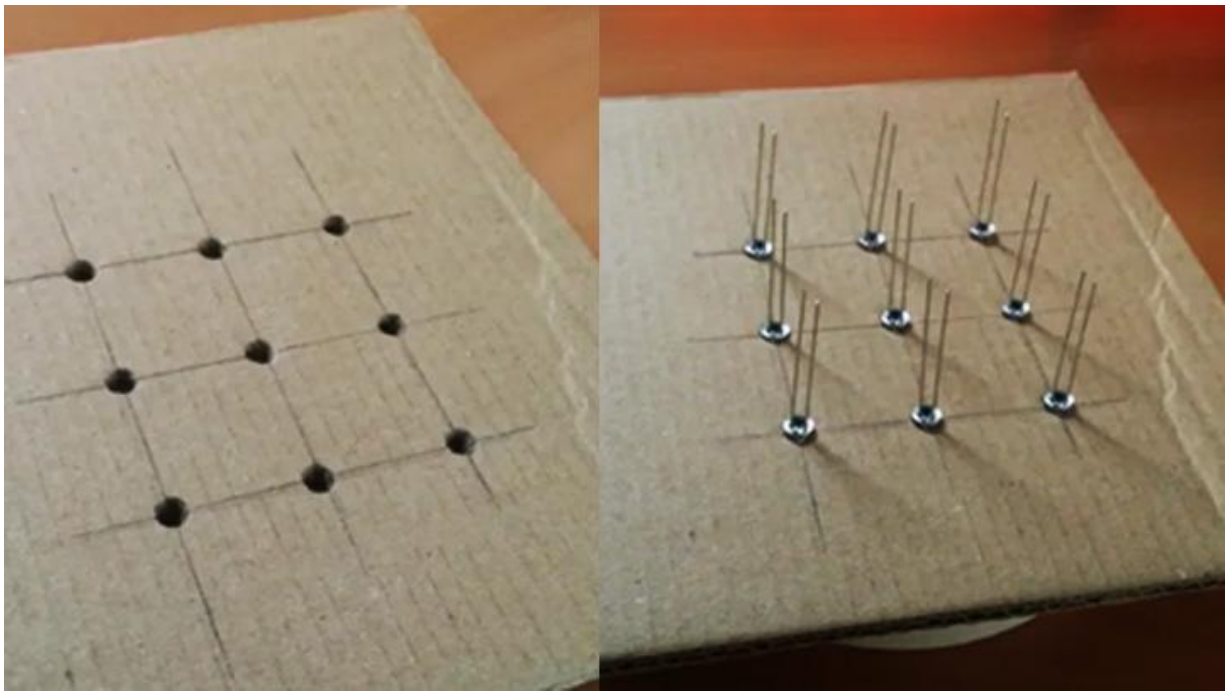


You will need the following items to make this LED Cube:

27 LEDs in one color.
Arduino Nano or one of his older brothers.

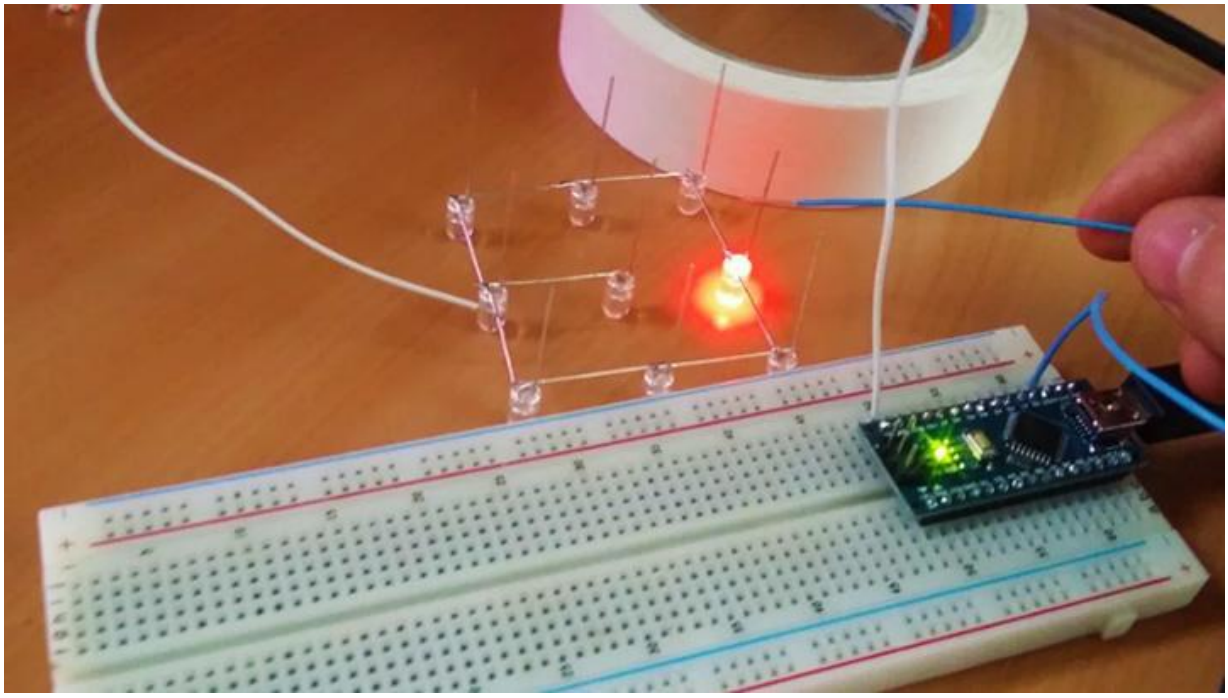
3 NPN Transistors. I used BC547.
3 pin heads with 3 pins.
1 pin head with four pins.
A piece of board.
A few small wires.

Step 1: Building 3x3 Layers



To start we have to make 3 layers of 3x3 LED; s. An easy way to do this is to get a piece of cardboard and draw holes with a 3x3 pattern. Make sure the LEDs will not come in when you press them in the holes. In the end it should look like a picture.

Step 2: Wrap all the Cathodes in each frame

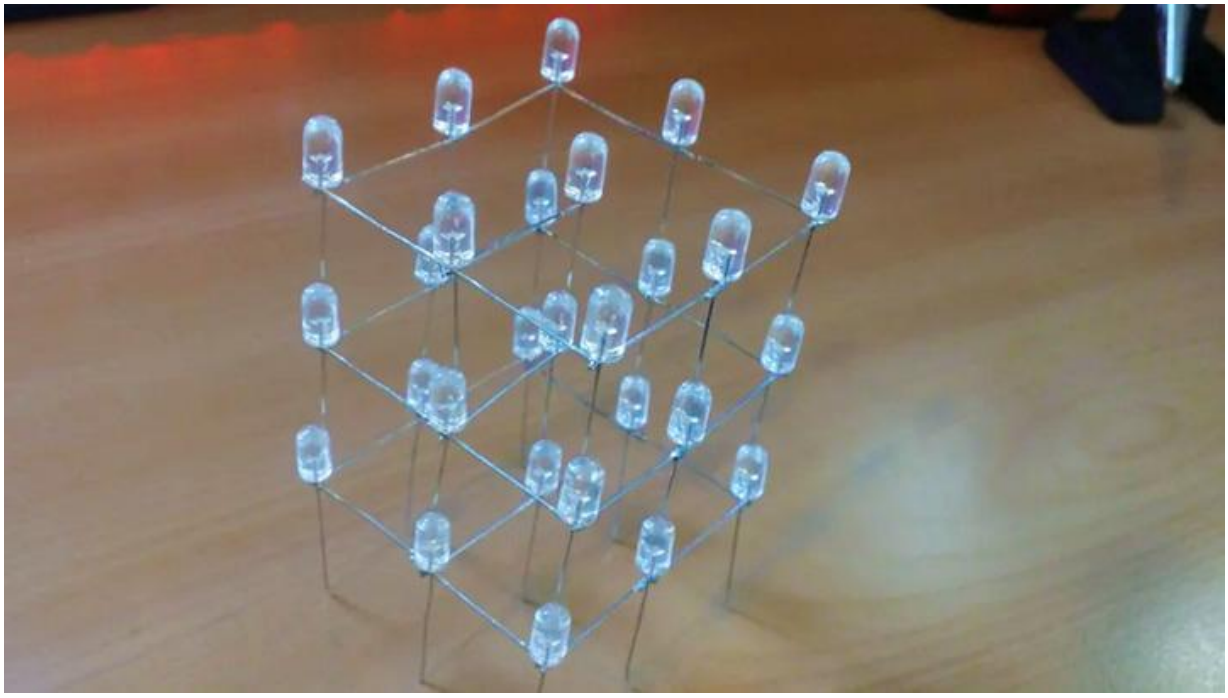


We must connect all the cathodes in each layer now. Ofcourse we use solder for this.
Repeat steps 1 and 2 twice and you should end up with 3 layers that look exactly the same.

NOTE: Cathodes and anodes MUST NOT be in contact.

It is best to check all your solder connections before proceeding with step 3. The easiest way is to hold your Arduino and connect one of the cathodes (pins you sold) to your Arduino GND and connect the cable to your Arduino 3V3. You can now touch each anode with a 3V3 wire and when everything is running the LED should turn on.

Step 3: Binding Layers Together



Use something like a soldering hand to solder all the anodes in the anode in the layer below.
Also, make sure you do not accidentally sell the cathode to the anode.

After you have finished wrapping all 3 layers together your cube is made! Now all that is left is to install a few electrical components and other connectors.

Step 4: Install the Cube on your Perfboard

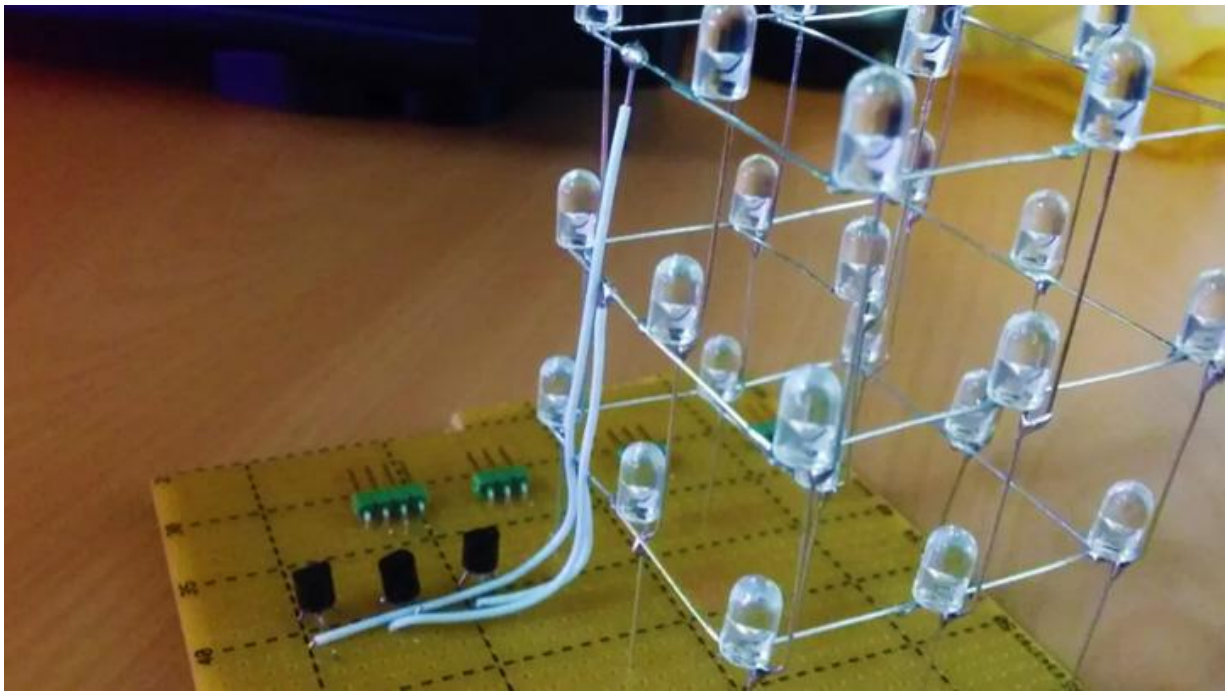
Now it's time to connect the remaining 9 anodes to the board.
You get the best result when you leave a gap between the perboard and the layer below. This will make it look as if the cube is floating in the middle of the air.

Step 5: Complete the Perfboard

Solder is a perfboard as shown in the picture provided.

Black lines interact between different copper strips. You can use solder or wire to connect it.
Blue lines are where you have to scratch the copper so that it no longer works.

Step 6: Connecting Layers to Perfboard



Use 3 wires to connect each layer to one of the transistors.
You can see which pin you should be solder to the perfboard structure in step 5.

Your cube is now finished! To connect it to your Arduino you have to connect wires to the headers.
You can see how to connect the cables using the image provided in step 7 ("Configuration").

After connecting your Arduino to the cake you can edit my code to check if everything is working properly. If you are not sure if everything is working properly I suggest you check out my youtube video because it is the same code I used in the video.

Are some LEDs not on? Check that all your solder connections are correct and check the short circuits on your perfboard.
You may encounter a problem that one of the layers is not working. This may be because one of the transistors is damaged .... Don't panic! If you replace a damaged transistor your LED hub should work properly.

Step 7: Setting up the LED Cube

I'm really new to Arduino editing myself, so I can't tell you much about it. I don’t know how I use arrays and I don’t know how I can replicate.
But ... I can explain a few simple tricks that make the process so much easier!

```
/*
Don't forget to check out my YouTube channel:
https://www.youtube.com/channel/UCKp8cQWkiGGfAV5FM_Q0mxA
*/
void setup()
  pinMode(2, OUTPUT);
  pinMode(3, OUTPUT);
  pinMode(4, OUTPUT);
  pinMode(5, OUTPUT);
  pinMode(6, OUTPUT);
  pinMode(7, OUTPUT);
  pinMode(8, OUTPUT);
  pinMode(9, OUTPUT);
  pinMode(10, OUTPUT);
  pinMode(11, OUTPUT) ;
  pinMode(12, OUTPUT);
```

```
  pinMode(13, OUTPUT);
}

void allLayer() {
  digitalWrite(11, HIGH);
  digitalWrite(12, HIGH);
  digitalWrite(13, HIGH);
}
void noLayer() {
  digitalWrite(11, LOW);
  digitalWrite(12, LOW);
  digitalWrite(13, LOW);
  }
void rotate() {
  allLayer();
  digitalWrite(5, HIGH);
  digitalWrite(6, HIGH);
  digitalWrite(7, HIGH);
  delay(100);
  empty();
  digitalWrite(2, HIGH);
  digitalWrite(6, HIGH);
  digitalWrite(10, HIGH);
  delay(100);
  empty() ;
  digitalWrite(3, HIGH);
  digitalWrite(6, HIGH);
  digitalWrite(9, HIGH);
```

```
  delay(100);
  empty();
  digitalWrite(4, HIGH);
  digitalWrite(6, HIGH);
  digitalWrite(8, HIGH);
  delay(100);
  empty();
  noLayer();
  }
void fill() {
  digitalWrite(2, HIGH);
  digitalWrite(3, HIGH);
  digitalWrite(4, HIGH);
  digitalWrite(5, HIGH);
  digitalWrite(6, HIGH);
  digitalWrite(7, HIGH);
  digitalWrite(8, HIGH);
  digitalWrite(9, HIGH);
  digitalWrite(10, HIGH);
  }
void swirl() {
  digitalWrite(2, HIGH) ;
  delay(50);
  digitalWrite(3, HIGH);
  delay(50);
  digitalWrite(4, HIGH);
  delay(50);
```

```
  digitalWrite(7, HIGH);
  delay(50);
  digitalWrite(10, HIGH);
  delay(50);
  digitalWrite(9, HIGH);
  delay(50);
  digitalWrite(8, HIGH);
  delay(50);
  digitalWrite(5, HIGH);
  delay(50);
  digitalWrite(6, HIGH);
  delay(50);
  }

void myName() {
  //Y
  digitalWrite(11, HIGH);
  digitalWrite(2, HIGH);
  digitalWrite(4, HIGH);
  digitalWrite(6, HIGH);
  digitalWrite(9, HIGH) ;
  delay(100);
  noLayer();
  digitalWrite(12, HIGH);
  delay(100);
  noLayer();
  digitalWrite(13, HIGH);
  delay(200);
```

```
noLayer();
digitalWrite(12, HIGH);
delay(100);
noLayer();
digitalWrite(11, HIGH);
delay(100);
noLayer();
empty();

//O
digitalWrite(11, HIGH);
digitalWrite(2, HIGH);
digitalWrite(3, HIGH);
digitalWrite(4, HIGH);
digitalWrite(5, HIGH);
digitalWrite(7, HIGH);
digitalWrite(8, HIGH);
digitalWrite(9, HIGH);
digitalWrite(10, HIGH) ;
delay(100);
noLayer();
digitalWrite(12, HIGH);
delay(100);
noLayer();
digitalWrite(13, HIGH);
delay(200);
noLayer();
digitalWrite(12, HIGH);
```

```
delay(100);
noLayer();
digitalWrite(11, HIGH);
delay(100);
noLayer();
empty();

//U
digitalWrite(11, HIGH);
digitalWrite(2, HIGH);
digitalWrite(4, HIGH);
digitalWrite(5, HIGH);
digitalWrite(7, HIGH);
digitalWrite(8, HIGH);
digitalWrite(9, HIGH);
digitalWrite(10, HIGH);
delay(100) ;
noLayer();
digitalWrite(12, HIGH);
delay(100);
noLayer();
digitalWrite(13, HIGH);
delay(200);
noLayer();
digitalWrite(12, HIGH);
delay(100);
noLayer();
digitalWrite(11, HIGH);
```

```
delay(100);
noLayer();
empty();

//R
digitalWrite(11, HIGH);
digitalWrite(2, HIGH);
digitalWrite(3, HIGH);
digitalWrite(4, HIGH);
digitalWrite(5, HIGH);
digitalWrite(8, HIGH);
delay(100);
noLayer();
digitalWrite(12, HIGH);
delay(100) ;
noLayer();
digitalWrite(13, HIGH);
delay(200);
noLayer();
digitalWrite(12, HIGH);
delay(100);
noLayer();
digitalWrite(11, HIGH);
delay(100);
noLayer();
empty();

//I
```

```
  digitalWrite(11, HIGH);
  digitalWrite(3, HIGH);
  digitalWrite(6, HIGH);
  digitalWrite(9, HIGH);
  delay(100);
  noLayer();
  digitalWrite(12, HIGH);
  delay(100);
  noLayer();
  digitalWrite(13, HIGH);
  delay(200);
  noLayer();
  digitalWrite(12, HIGH) ;
  delay(100);
  noLayer();
  digitalWrite(11, HIGH);
  delay(100);
  noLayer();
  empty();
  }

void empty() {
  digitalWrite(2, LOW);
  digitalWrite(3, LOW);
  digitalWrite(4, LOW);
  digitalWrite(5, LOW);
  digitalWrite(6, LOW);
  digitalWrite(7, LOW);
```

```
  digitalWrite(8, LOW);
  digitalWrite(9, LOW);
  digitalWrite(10, LOW);
  }
void createX() {
  digitalWrite(2, HIGH);
  digitalWrite(4, HIGH);
  digitalWrite(6, HIGH);
  digitalWrite(8, HIGH);
  digitalWrite(10, HIGH) ;
  delay(1000);
  digitalWrite(2, LOW);
  digitalWrite(4, LOW);
  digitalWrite(6, LOW);
  digitalWrite(8, LOW);
  digitalWrite(10, LOW);
  delay(200);
   digitalWrite(2, HIGH);
  digitalWrite(4, HIGH);
  digitalWrite(6, HIGH);
  digitalWrite(8, HIGH);
  digitalWrite(10, HIGH);
  delay(200);
   digitalWrite(2, LOW);
  digitalWrite(4, LOW);
  digitalWrite(6, LOW);
  digitalWrite(8, LOW);
```

```
  digitalWrite(10, LOW);
  delay(200);
digitalWrite(2, HIGH);
  digitalWrite(4, HIGH);
  digitalWrite(6, HIGH);
  digitalWrite(8, HIGH);
  digitalWrite(10, HIGH);
  delay(200);
   digitalWrite(2, LOW) ;
  digitalWrite(4, LOW);
  digitalWrite(6, LOW);
  digitalWrite(8, LOW);
  digitalWrite(10, LOW);
  delay(200);
   digitalWrite(2, HIGH);
  digitalWrite(4, HIGH);
  digitalWrite(6, HIGH);
  digitalWrite(8, HIGH);
  digitalWrite(10, HIGH);
  delay(200);
  digitalWrite(2, LOW);
  digitalWrite(4, LOW);
  digitalWrite(6, LOW);
  digitalWrite(8, LOW);
  digitalWrite(10, LOW);
  delay(200);
   digitalWrite(2, HIGH);
```

```
  digitalWrite(4, HIGH);
  digitalWrite(6, HIGH);
  digitalWrite(8, HIGH);
  digitalWrite(10, HIGH);
  delay(200);
  digitalWrite(2, LOW);
  digitalWrite(4, LOW);
  digitalWrite(6, LOW) ;
  digitalWrite(8, LOW);
  digitalWrite(10, LOW);
  delay(500);
}

void loop() {

//FILL - UP AND DOWN

digitalWrite(13, HIGH);
fill();
delay(200);
empty();
noLayer();
digitalWrite(12, HIGH);
fill();
delay(100);
empty();
noLayer();
digitalWrite(11, HIGH);
fill();
```

```
delay(200);
empty();
noLayer();
digitalWrite(12, HIGH);
fill();
delay(100);
empty();
noLayer();
digitalWrite(13, HIGH);
fill();
delay(200);
empty();
noLayer();
digitalWrite(12, HIGH);
fill();
delay(100);
empty();
noLayer();
digitalWrite(11, HIGH);
fill();
delay(200);
empty();
noLayer();
digitalWrite(12, HIGH);
fill();
delay(100);
empty();
```

```
noLayer();
digitalWrite(13, HIGH);
fill();
delay(200);
empty();
noLayer();
digitalWrite(12, HIGH);
fill();
delay(100);
empty();
noLayer();
digitalWrite(11, HIGH);
fill();
delay(200);
empty();
noLayer();
digitalWrite(12, HIGH);
fill();
delay(100);
empty();
noLayer();
digitalWrite(13, HIGH);
fill();
delay(200);
empty();
noLayer();
delay(1000);
```

```
//SWIRL - 1 LAYER EACH TIME

digitalWrite(13, HIGH);
swirl();
delay(10);
empty();
noLayer();
digitalWrite(12, HIGH);
swirl();
delay(10);
empty();
noLayer();
digitalWrite(11, HIGH);
swirl();
delay(10);
delay(400);
empty();
noLayer();

//SWIRL - ALL LAYERS AT ONCE

allLayer();
swirl();
delay(400);
empty();
noLayer();
delay(1000);

//NAME COMES UP
myName();
```

```
delay(1000);

//ROTATING PANE - ALL LAYERS
rotate();
rotate();
rotate();
rotate();
rotate();
rotate();
rotate();
rotate();
delay(1000);
}
```

First we will clarify how we want to use our Arduino pins. We can do this by pasting the following code into our useless "setup":

```
pinMode (2, OUTPUT);
pinMode (3, OUTPUT);
pinMode (4, OUTPUT);
pinMode (5, OUTPUT);
pinMode (6, OUTPUT);
pinMode (7, OUTPUT);
pinMode (8, OUTPUT);
pinMode (9, OUTPUT);
pinMode (10, OUTPUT);
pinMode (11, OUTPUT);
pinMode (12, OUTPUT);
pinMode (13, OUTPUT);
```

Anchors 2-10 should control the LEDs on each layer. Pin 11 control is the lowest layer, pin 12 controls the middle and pin 13 controls the top layer.

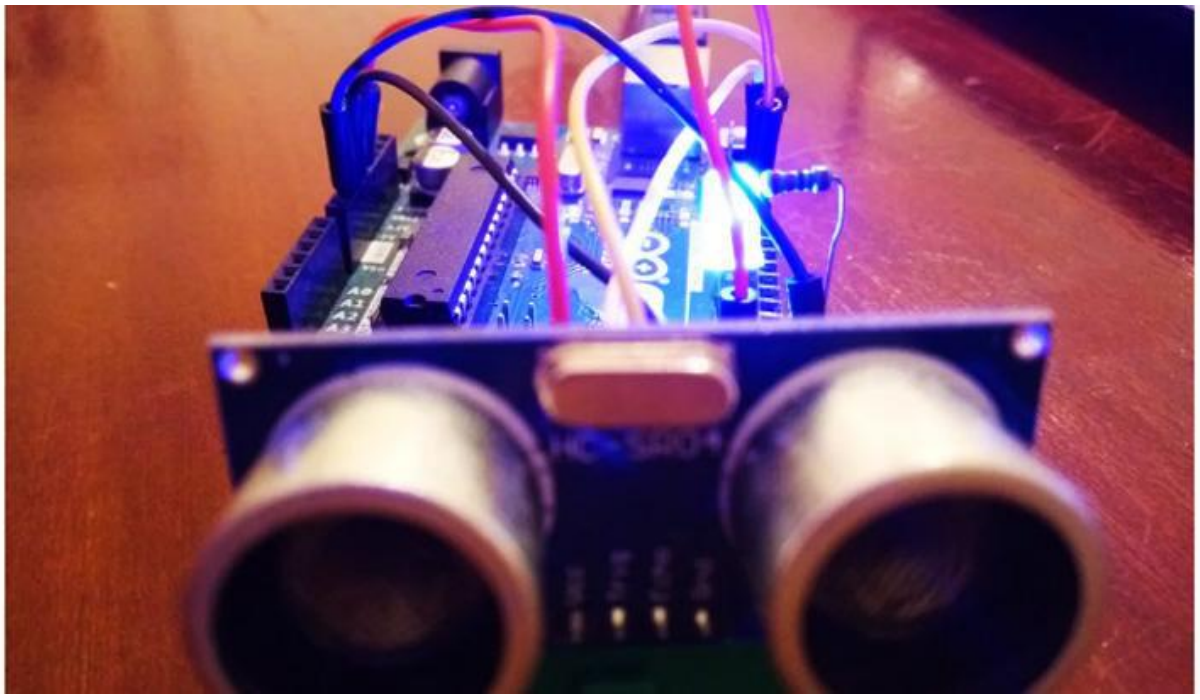
To connect your Arduino to the subject you must use the following pinout (depending on the image provided):
Left to right> D10 - D9 - D8 - D7 - D6 - D5 - D4 - D3 - D2 - GND - D11 - D12 - D13

It helps to create a void to make all the layers work at the same time. You can do this easily using the following void:

```
Void allLayer () {
digitalWrite (11, HIGH);
digitalWrite (12, HIGH);
digitalWrite (13, HIGH);
}
```

To disable all layers simultaneously you can use the same format. All you have to do is change the void name and change the maximum value to LOW.
You can also use this structure to make all the LEDs at once.

Project 07 - Ultrasonic Sensor (HC-SR04)



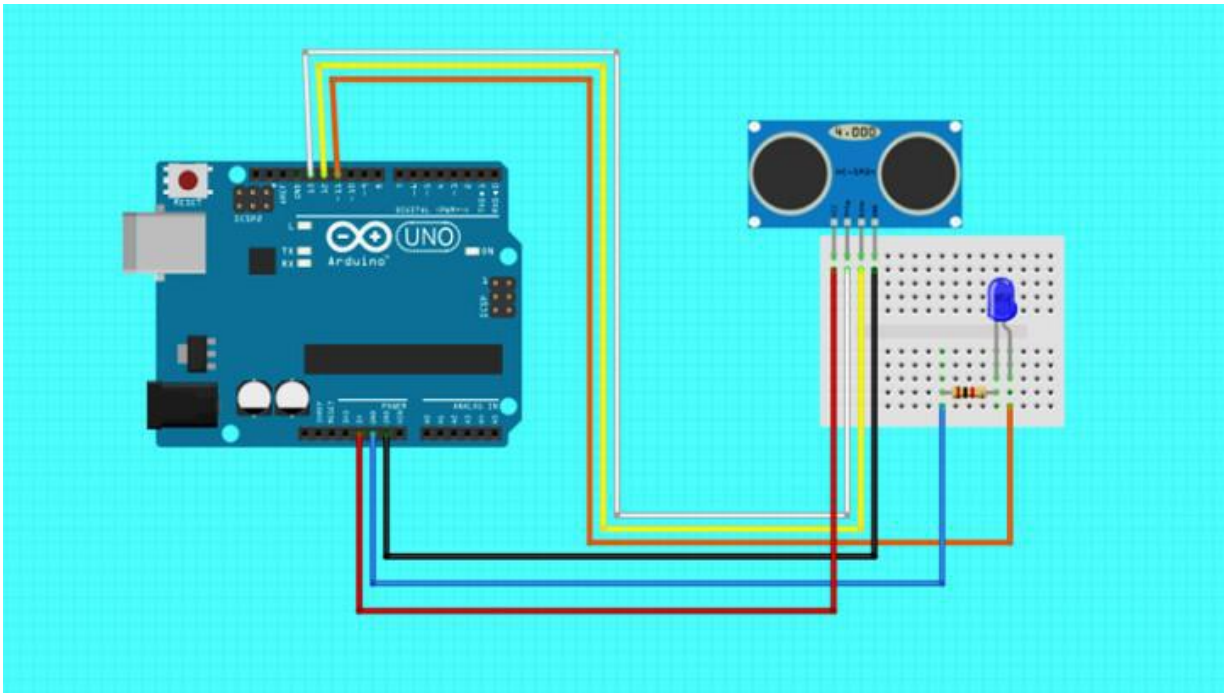
Ultrasonic module from HC - SR04 provides unbalanced measuring function of 2cm - 400cm, accuracy ranging from up to 3mm. Modules include ultrasonic transmission equipment, receiver and control circuit.

This is a simple example of using an ultrasonic sensor (HC-SR04) in Arduino where we will open a range of distance guidance and print the distance from the object to the serial monitor.

Step 1: All Required Parts

1. Arduino Uno
2. Ultrasonic Sensor (HC-SR04).
3. Mini-BreadBoard
4. 1 kohm Resistor.
5. Jumpers.
6. Blue LED.

Step 2: Connect the parts



The connection of the parts is very simple is necessary to follow the following pictures.

Step 3: Write Your Code

```
#define trigPin 13
#define echoPin 12
#define led 11

void setup()
{ Serial.begin (9600);
pinMode(trigPin, OUTPUT);
pinMode(echoPin, INPUT);
pinMode(led, OUTPUT);
}

void loop()
{ long duration, distance;
```

```
digitalWrite(trigPin, LOW);
delayMicroseconds(2);
digitalWrite(trigPin, HIGH);
delayMicroseconds(10);
digitalWrite(trigPin, LOW);
duration = pulseIn(echoPin, HIGH);
distance = (duration/2) / 29.1;
if (distance < 10)
{ digitalWrite(led,HIGH);
}
else {
digitalWrite(led,LOW);
}
Serial.print(distance);
Serial.println(" cm");
delay(500);
}
```

Step 4: Upload and run.

You can change the range value by adding 10 to the position if (range <= 10) is the value you want.

Project 08 - How to Use an RGB LED

The RGB LED has 4 pins, one for each color (Red, Green, Blue) and a standard cathode. It has color-coded diodes that separate the trees that can be combined to create all kinds of color! Any color is possible depending on how bright each diode is.

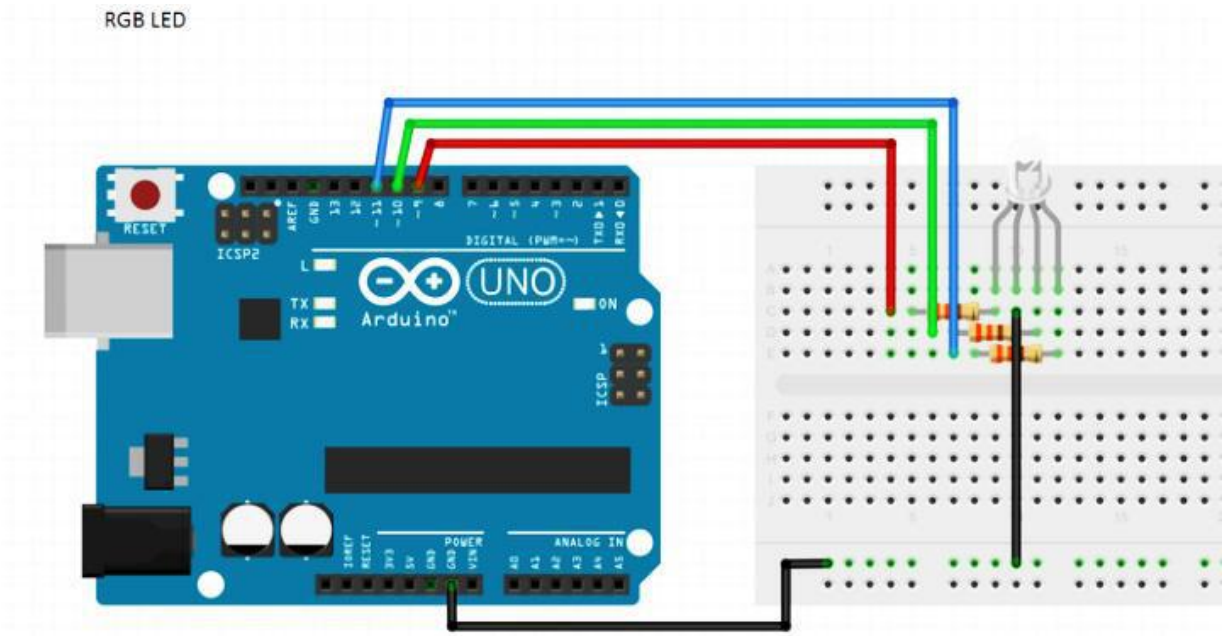
In this tutorial, you will learn how to use ArGBino RGB LED and create a unique color combination.

Step 1: What You Will Need

For this tutorial you will need:

Arduino uno
Bread board
RGB LED
3x 330Ohm independent

Step 2: Circuit



The connection is very simple, see the picture above for the circuit board scheme.

Step 3: Code

/*

SparkFun Inventor's Kit

Example sketch 03

RGB LE D

Make an RGB LED display a rainbow of colors!

Hardware connections:

An RGB LED is actually three LEDs (red, green, and blue) in one package. When you run them at different brightnesses, the red, green and blue mix to form new colors.

Starting at the flattened edge of the flange on the LED, the pins are ordered RED, COMMON, GREEN, BLUE.

Connect RED to a 330 Ohm resistor. Connect the other end
of the resistor to Arduino digital pin 9.

Connect COMMON pin to GND.

Connect GREEN to a 330 Ohm resistor. Connect the other end
of the resistor to Arduino digital pin 10.

Connect BLUE to a 330 Ohm resistor. Connect the other end
of the resistor to Arduino digital pin 11.

This sketch was written by SparkFun Electronics,
with lots of help from the Arduino community.
Visit http://learn.sparkfun.com/products/2 for SIK information.
Visit http://www.arduino.cc to learn about the Arduino.

Version 2.0 6/2012 MDG

*/

// First we'll define the pins by name to make the sketch
// easier to follow.

// Here's a new trick: putting the word "const" in front of a
// variable indicates that this is a "constant" value that will
// never change. (You don't have to do this, but if you do, the
// Arduino will give you a friendly warning if you accidentally
// try to change the value, so it's considered good form.)

const int RED_PIN = 9;
const int GREEN_PIN = 10;
const int BLUE_PIN = 11;

```
// This variable controls how fast we loop through the colors.
// (Try changing this to make the fading faster or slower.)

int DISPLAY_TIME = 100;  // In milliseconds


void setup()
{
  // Here we'll configure the Arduino pins we're using to
  // drive the LED to be outputs:

  pinMode(RED_PIN, OUTPUT);
  pinMode(GREEN_PIN, OUTPUT);
  pinMode(BLUE_PIN, OUTPUT);
}


void loop()
{
  // In this sketch, we'll start writing our own functions.
  // This makes the sketch easier to follow by dividing up
  // the sketch into sections, and not having everything in
  // setup() or loop().

  // We'll show you two ways to run the RGB LED.

  // The first way is to turn the individual LEDs (red, blue,
  // and green) on and off in various combinations. This gives you
  // a total of eight colors (if you count "black" as a color).
  // We've written a function called mainColors() that steps
  // through all eight of these colors. We're only "calling" the
```

```
  // function here (telling it to run). The actual function code
  // is further down in the sketch .

  mainColors();

  // The above function turns the individual LEDs full-on and
  // full-off. If you want to generate more than eight colors,
  // you can do so by varying the brightness of the individual
  // LEDs between full-on and full-off.

  // The analogWrite() function lets us do this. This function
  // lets you dim a LED from full-off to full-on over 255 steps.

  // We've written a function called showSpectrum() that smoothly
  // steps through all the colors. Again we're just calling it
  // here; the actual code is further down in this sketch.

  showSpectrum();
}

// Here's the mainColors() function we've written.

// This function displays the eight "main" colors that the RGB LED
// can produce. If you'd like to use one of these colors in your
// own sketch, you cancopy and paste that section into your code.

void mainColors()
{
  // Off (all LEDs off):

  digitalWrite(RED_PIN, LOW);
  digitalWrite(GREEN_PIN, LOW);
```

```
digitalWrite(BLUE_PIN, LOW);

delay(1000);

// Red (turn just the red LED on):

digitalWrite(RED_PIN, HIGH);
digitalWrite(GREEN_PIN, LOW);
digitalWrite(BLUE_PIN, LOW);

delay(1000);

// Green (turn just the green LED on):

digitalWrite(RED_PIN, LOW);
digitalWrite(GREEN_PIN, HIGH);
digitalWrite(BLUE_PIN, LOW);

delay(1000);

// Blue (turn just the blue LED on) :

digitalWrite(RED_PIN, LOW);
digitalWrite(GREEN_PIN, LOW);
digitalWrite(BLUE_PIN, HIGH);

delay(1000);

// Yellow (turn red and green on):

digitalWrite(RED_PIN, HIGH);
digitalWrite(GREEN_PIN, HIGH);
digitalWrite(BLUE_PIN, LOW);

delay(1000);
```

```
  // Cyan (turn green and blue on):

  digitalWrite(RED_PIN, LOW);
  digitalWrite(GREEN_PIN, HIGH);
  digitalWrite(BLUE_PIN, HIGH);

  delay(1000);

  // Purple (turn red and blue on):

  digitalWrite(RED_PIN, HIGH);
  digitalWrite(GREEN_PIN, LOW) ;
  digitalWrite(BLUE_PIN, HIGH);

  delay(1000);

  // White (turn all the LEDs on):

  digitalWrite(RED_PIN, HIGH);
  digitalWrite(GREEN_PIN, HIGH);
  digitalWrite(BLUE_PIN, HIGH);

  delay(1000);
}

// Below are two more functions we've written,
// showSpectrum() and showRGB().

// showRGB() displays a single color on the RGB LED.
// You call showRGB() with the number of a color you want
// to display.

// showSpectrum() steps through all the colors of the RGB LED,
```

```
// displaying a rainbow. showSpectrum() actually calls showRGB()
// over and over to do this.

// We'll often break tasks down into individual functions like
// this, which makes your sketches easier to follow, and once
// you have a handy function, you can reuse it in your other
// programs.


// showSpectrum()

// This function steps through all the colors of the RGB LED.
// It does this by stepping a variable from 0 to 768 (the total
// number of colors), and repeatedly calling showRGB() to display
// the individual colors.

// In this function, we're using a "for() loop" to step a variable
// from one value to another, and perform a set of instructions
// for each step. For() loops are a very handy way to get numbers
// to count up or down.

// Every for() loop has three statements separated by semicolons:

//   1. Something to do before starting

//   2. A test to perform; as long as it's true,
//      it will keep looping

//   3. Something to do after each loop (usually
//      increase a variable)

// For the for() loop below, these are the three statements:
```

```
//   1. x = 0;     Before starting, make x = 0.

//   2. x < 768;   While x is less than 768, run the
//                 following code.

//   3. x++        Putting "++" after a variable means
//                 "add one to it". (You can also use "x = x + 1")

// Every time you go through the loop, the statements following
// the loop (those within the brackets) will run.

// And when the test in statement 2 is finally false, the sketch
// will continue.


void showSpectrum()
{
  int x;  // define an integer variable called "x"

  // Now we'll use a for() loop to make x count from 0 to 767
  // (Note that there's no semicolon after this line!
  // That's because the for() loop will repeat the next
  // "statement", which in this case is everything within
  // the following brackets {} )

  for (x = 0; x < 768; x++)

  // Each time we loop (with a new value of x), do the following:

  {
    showRGB(x);  // Call RGBspectrum() with our new x
    delay(10);   // Delay for 10 ms (1/100th of a second)
  }
```

```
}

// showRGB()

// This function translates a number between 0 and 767 into a
// specific color on the RGB LED. If you have this number count
// through the whole range (0 to 767), the LED will smoothly
// change color through the entire spectrum.

// The "base" numbers are:
// 0   = pure red
// 255 = pure green
// 511 = pure blue
// 767 = pure red (again)

// Numbers between the above colors will create blends. For
// example, 640 is midway between 512 (pure blue) and 767
// (pure red). It will give you a 50/50 mix of blue and red,
// resulting in purple.

// If you count up from 0 to 767 and pass that number to this
// function, the LED will smoothly fade between all the colors.
// (Because it starts and ends on pure red, you can start over
// at 0 without any break in the spectrum).

void showRGB(int color)
{
  int redIntensity;
  int greenIntensity;
```

```
int blueIntensity;

// Here we'll use an "if / else" statement to determine which
// of the three (R,G,B) zones x falls into. Each of these zones
// spans 255 because analogWrite() wants a number from 0 to 255.

// In each of these zones, we'll calculate the brightness
// for each of the red, green, and blue LEDs within the RGB LED.

if (color <= 255)          // zone 1
{
  redIntensity = 255 - color;    // red goes from on to off
  greenIntensity = color;        // green goes from off to o n
  blueIntensity = 0;             // blue is always off
}
else if (color <= 511)     // zone 2
{
  redIntensity = 0;                    // red is always off
  greenIntensity = 255 - (color - 256); // green on to off
  blueIntensity = (color - 256);        // blue off to on
}
else // color >= 512       // zone 3
{
  redIntensity = (color - 512);         // red off to on
  greenIntensity = 0;                   // green is always off
  blueIntensity = 255 - (color - 512);  // blue on to off
}

// Now that the brightness values have been set, command the LED
// to those values
```

Here's the code, embedded using bender code!

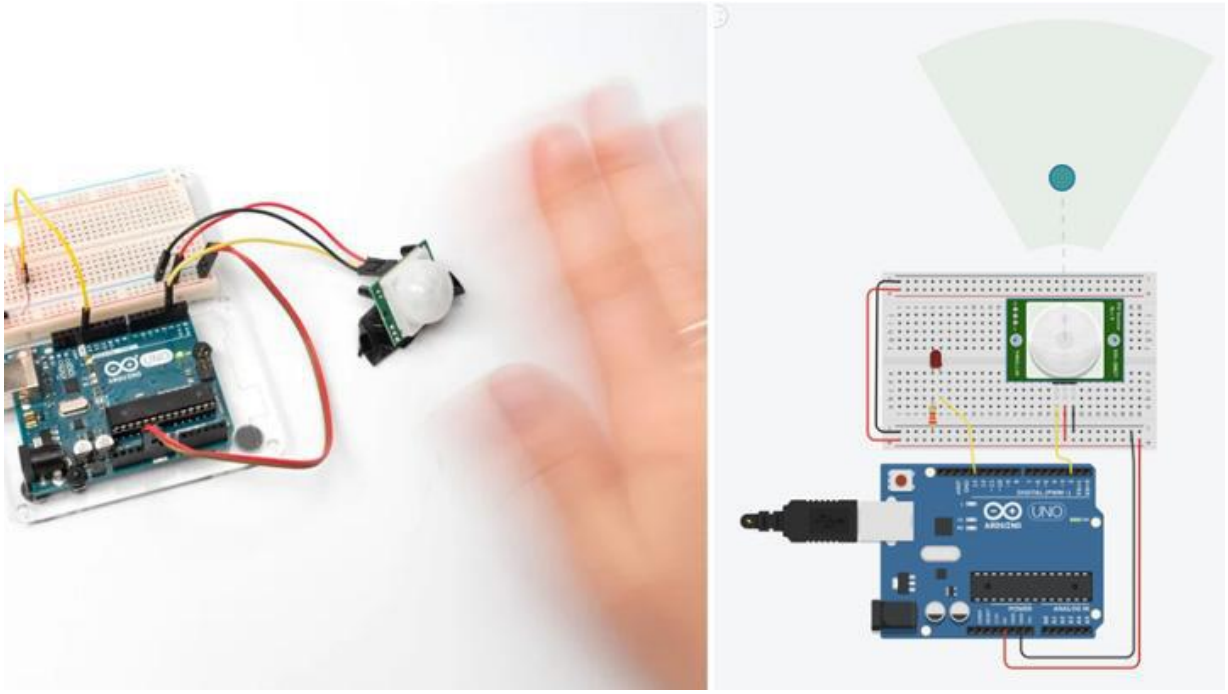
We will use code from SparkFun Inventor's Kit - SIK Guide, Example sketch 03. This is an excellent diagram to learn how RGB LED works. The code is self-explanatory, and ideas do a better job than I explain how it works.

Try downloading the plugin bender plugin and click on the Run on Arduino button to edit your Arduino board with this drawing. And of course, plan your Arduino with RGB LED drawing!

Step 4: Well done

You have successfully completed one Arduino "How to" tutorial and learned how to use RGB LED with Arduino.

Project 09 - PIR Motion Sensor



Let's learn to feel the movement in the PIR sensory chamber and Arduino's digital input. We will connect the circuit using a breadboard and use a simple Arduino code to control a single LED. We'll use

Tinkercad Circuits to mimic a circuit so you can follow without elements, and show you how to build a visual circuit too.

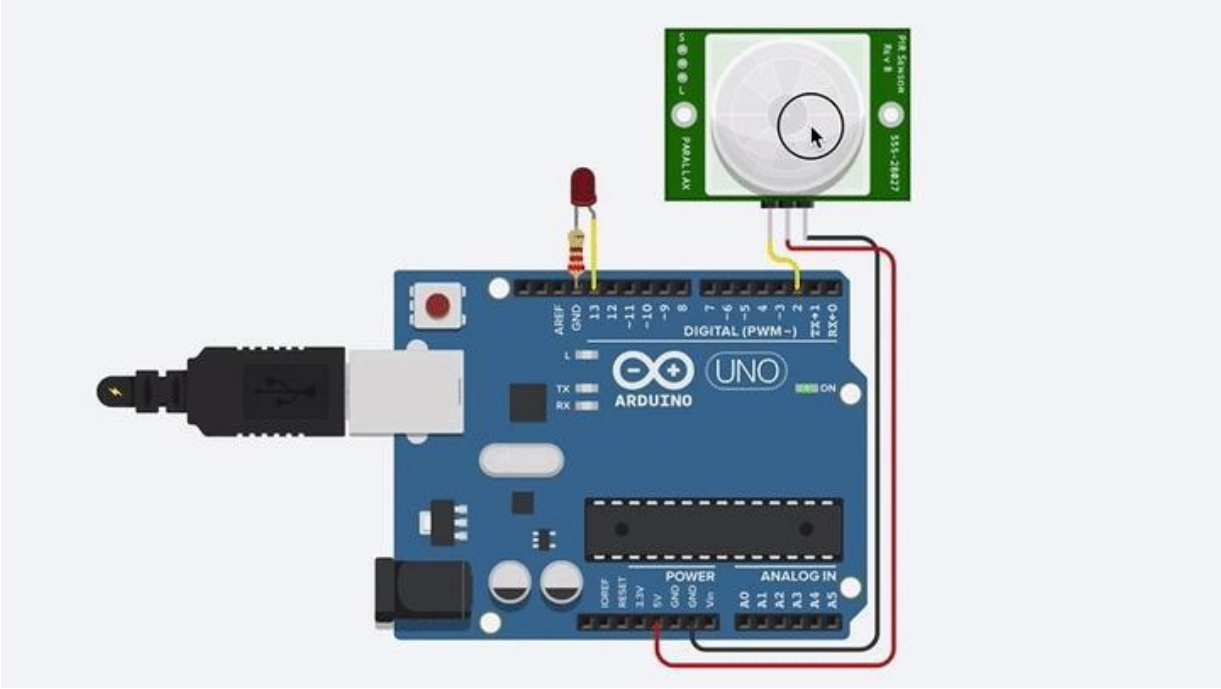
So far you've learned to read Arduino's digital input button, so we'll build on those skills in this tutorial. Although the motion sensor may seem complicated with a dedicated circuit board, it is designed to send a HIGH or LOW signal in the same way as a pushbutton.

The PIR stands for Passive InfraRed, which describes intermediate technology - it absorbs light infrared light levels (unlike an infrared camera that can emit even infrared light to capture its light). The white dome is a lens that expands the field of vision of the IR detector. The sensor automatically detects the LOW signal, reads the amount of infrared light coming in, and triggers the HIGH signal at a certain time when the light levels change, indicating movement. It can tell you if there is any movement in the scene, but it can't find the distance — so you can look at the type of analog input sensor called the ultrasonic rangefinder.

To optionally select a virtual circuit, connect your Arduino Uno board, USB cable, solderless breadboard, LED, resistor (any value from 100-1K), PIR motion sensor, and cables of bread board.

You can follow up almost using Tinkercad Circuits. You can watch this tutorial within Tinkercad (free sign-in required)! Examine the regional sample and create your own right next to it. Tinkercad Circuits is a free browser-based program that allows you to create and emulate circuits. It is ideal for learning, teaching and prototyping.

Step 1: Build a Circuit



Check the circuit sample here in the embedded circuit below by starting the simulation and clicking the circular motion sensor. This will activate the highlighted area in front of the sensor with the "object" circle inside. You may need to expand the view when the circle is closed on the screen. Click and drag the "object" circle before the sensor to represent the movement. The LED will turn on for a moment when motion is detected.

The free cable type for this region is shown above. If necessary, take a moment to refresh your breadboard experience. You can download the new Tinkercad Circuits window and build your own version of this circuit next to the sample.

Identify the PIR movement sensor, LED, resistor, and Arduino connected cables.

Drag the Arduino Uno and bread board from the object panel to the performance plane.

Connect the power of the breadboard (+) and the rail (-) rail to the Arduino 5V with the ground (GND), respectively, by clicking to create wires.

Increase the strength of the power rail and their proper buses on the opposite side of the bread board by forming a red fence between both electric buses and a black fence between both lower buses.

Connect the LED to different loops of bread so that the cathode (negative, short leg) connects to one leg of the resistor (any from 100-1K ohms is fine). The resistor can enter any position because the separate resistors, unlike the LEDs, have to be connected in a certain way in order to work.

Connect the other leg to the ground resistance.

Install the LED anode (good leg, long leg) on Arduino pin 13.

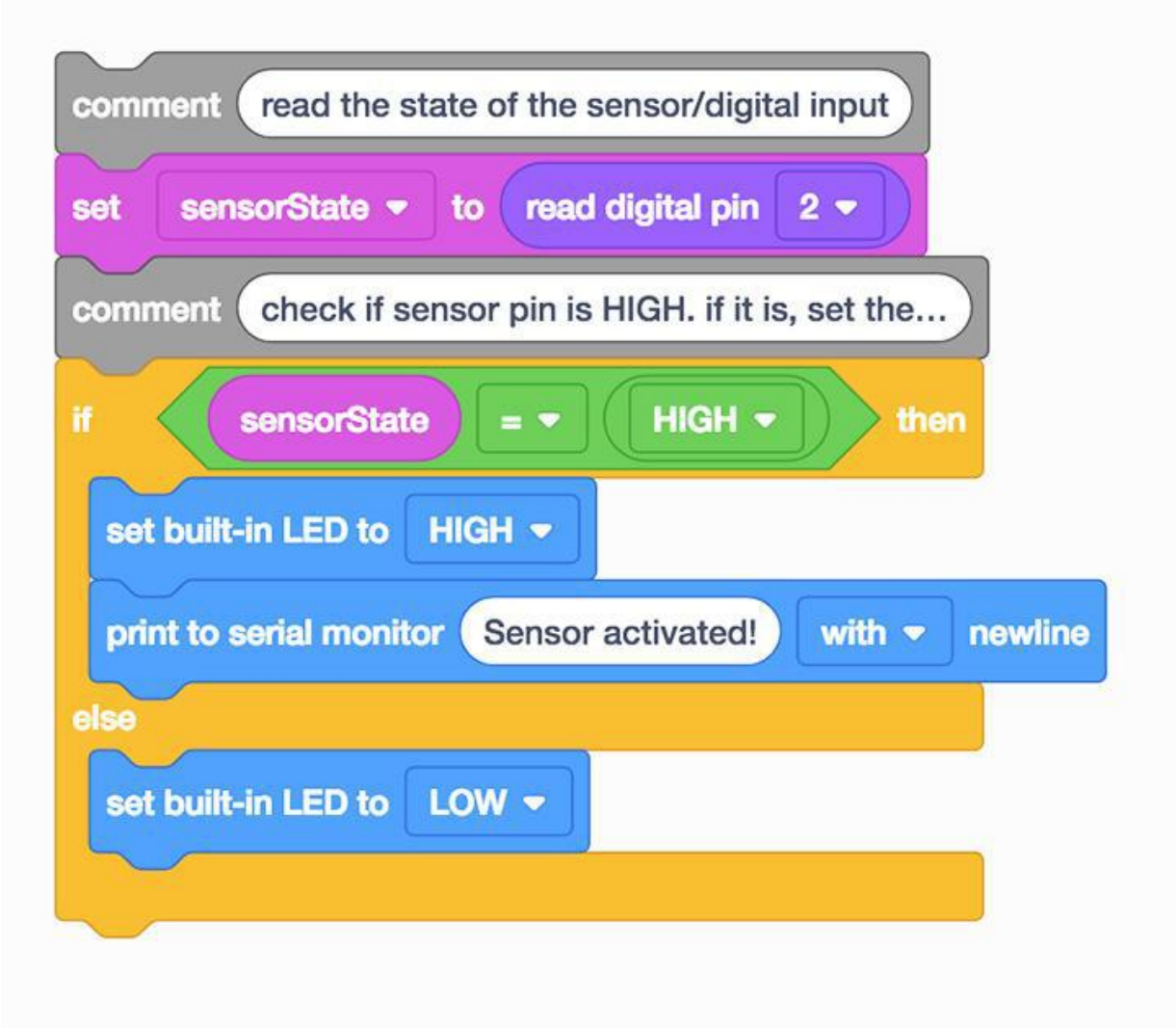
Drag the PIR movement sensor from the component panel to your bread board, so its legs connect in three different lines.

Click to create a call that connects the right leg to power.

Connect the center leg to the ground.

Create a wire that connects the left leg to the Arduino analog pin A0.

Step 2: Blocked Code



Let's use the Blocks coding interface to listen to the PIR movement sensor, and then make the decision to turn on the LED depending on the sensor state: activated or not activated.

Click the "Code" button to open the code editor.

Click the Variables section in the code editor. Create a new variable called sensorState.

Drag the "set" block.

We will maintain the status of our PIR motion sensor in our flexible sensorState. Click the Input Input section, drag outside the "read digital PIN" block, and place it in the "set" block behind the word "to".

Since our sensor is connected to Arduino on Pin 2, change the drop down of the "read digital PIN" block to 2. Now your blocks should read "set sensorState to read digital pin 2" which stores digital pin sensor reading in our State variable sensor!

Click the Control section and remove the "if any" block

Set it up to test whether the sensorState equals HIGH using the Math comparison block. Drag the Mat comparison block in your statement to see if our flexible sensorState is equal to HIGH.

We want to turn on our LED when the sensor is working - otherwise, we want our LED to be turned off. Under the Output block section, find the block “set built-in LED up and down”. Try adding two of these blocks to our if statement so that the LED will only light up when the sensor is working. built-in built-in LED should be HIGH when sensitivity is high (otherwise, the built-in LED should be LOW.

Step 3: Explain the PIR Motion Sensor Arduino code

When the code editor is open, you can click the drop-down menu on the left and select "Blocks + Text" to display the Arduino code made of code blocks. Follow along as we examine the code in detail.

```
int sensorState = 0;
```

Before setting (), we create a variable to maintain the current sensor state. It is called int because it is a whole number, or another whole number (although we will be using values 0 and 1, LOW and HIGH).

```
void setup()
{
  pinMode(2, INPUT);
  pinMode(13, OUTPUT);
  Serial.begin(9600);
}
```

Within setup, anchors are configured using the pinMode () function. Pin 2 is set as input, so we can "listen" to the sensor's electrical

condition. Pin 13 is configured as an LED control switch. To be able to send messages, Arduino opens a new serial communication channel with Serial.begin (), which captures the baud rate (any connection speed), this time at 9600 bits per second.

```
void loop()
{
  // read the state of the sensor/digital input
  sensorState = digitalRead(2);
```

Anything behind the collection of slashes // comments, which we humans should learn, and not included in the program where Arduino works. In a large loop, a function called digitalRead (); checks pin 2 status (which will be 5V aka HIGH or ground aka LOW), and saves that status to the variable sensorState we have made up.

```
// check if sensor pin is HIGH. if it is, set the
  // LED on.
  if (sensorState == HIGH) {
    digitalWrite(13, HIGH);
    Serial.println("Sensor activated!");
  } else {
    digitalWrite(13, LOW);
  }
  delay(10); // Delay a little bit to improve simulation performance
}
```

Below the two lines of comment statement is a test to see if the sensorState is HIGH (== is a comparison operator, not to be confused with =, which is a share operator). When the condition is met, the built-in LED is METAL (turned on). If not, the code contained within the rest {is used instead: the built-in LED is set to LOW (off). If the statements can be alone, or with one or more statements.

Step 4: Setting up the PIR Motion Sensor

If you are building a virtual circuit, you will need to make a small set with your PIR motion sensor.

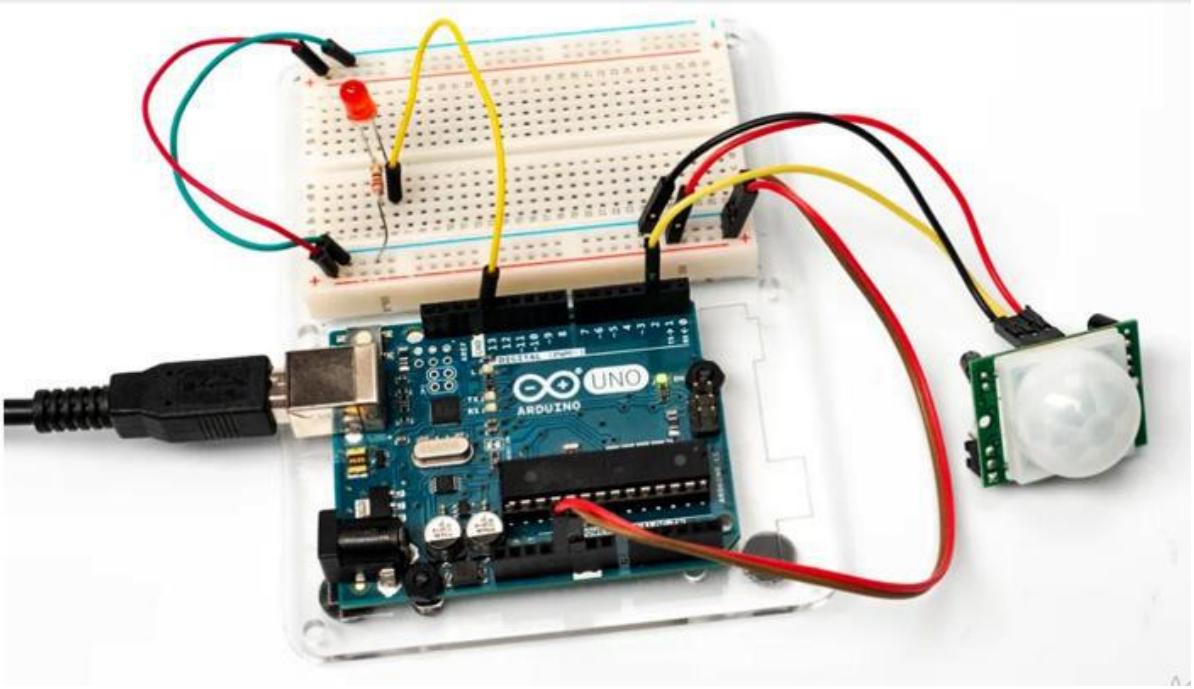
Find a line of three topics on the circuit board. They will be in the middle of one edge and labeled GND, OUT, and + 5v (or similar).

Connect the plug connector to the three wires, with a black wire lined with GND.

Double check the connector sitting firmly.

Alternatively, you can connect three male to male wires to the head pins.

Step 5: Create a Physical Arduino Circuit (Optional)



To customize your Arduino Uno, you will need to install free software (or a web editor plugin) and open it.

Connect the Arduino Uno circuit by inserting parts and wires to match the connections shown here in the Tinkercad Circuits. To get deeper into working with your Arduino Uno board, check out the free section of Arduino Instructables.

Copy the code from the Tinkercad Circuits code window and paste it into a blank diagram in your Arduino software, or click the download button (down arrow) and open the file using Arduino. You can also find this example in Arduino software by navigating to File -> Examples -> 02. Digital -> Button (with a different but different name).

Connect your USB cable and select your board and hole in the software tools menu.

Download the code and watch your LED light as you go before the sensor!

Step 6: Adjustment of PIR Motion Sensor

Some PIR motion sensors come with two adjustment potentiometers that change the sensitivity and length of the opening signal. The PIR movement sensor here at Tinkercad Circuits does not mimic this adjustment.

Optionally use a small screwdriver to adjust the sensitivity and dial time on the side of the circuit board of your PIR motion sensor. Try to see the effect on regional behavior.

Step 7: Next, Try

Now that you have learned to detect the PIR sensor signal and use it when the status quo statements are ready, you are ready to get used to entering more codes and building your sensor on the completed project.

Can you replace the LED with a servo motor, and write a program to rotate the servo when the sensor is triggered?

Try the 3D printing side of Tinkercad to create an electric enclosed space by turning on your PIR motion sensor.

Try replacing your PIR motion sensor with other digital inputs such as a push button or tilt button.

Learn how to monitor your Arduino digital and analog installations using a computer using Serial Monitor.

You can also learn about many electronic skills with free command categories in Arduino, Basic Electronics, LEDs and lighting, 3D printing, and more.

Project 10 - DIY Arduino Obstacle Avoiding Car at Home

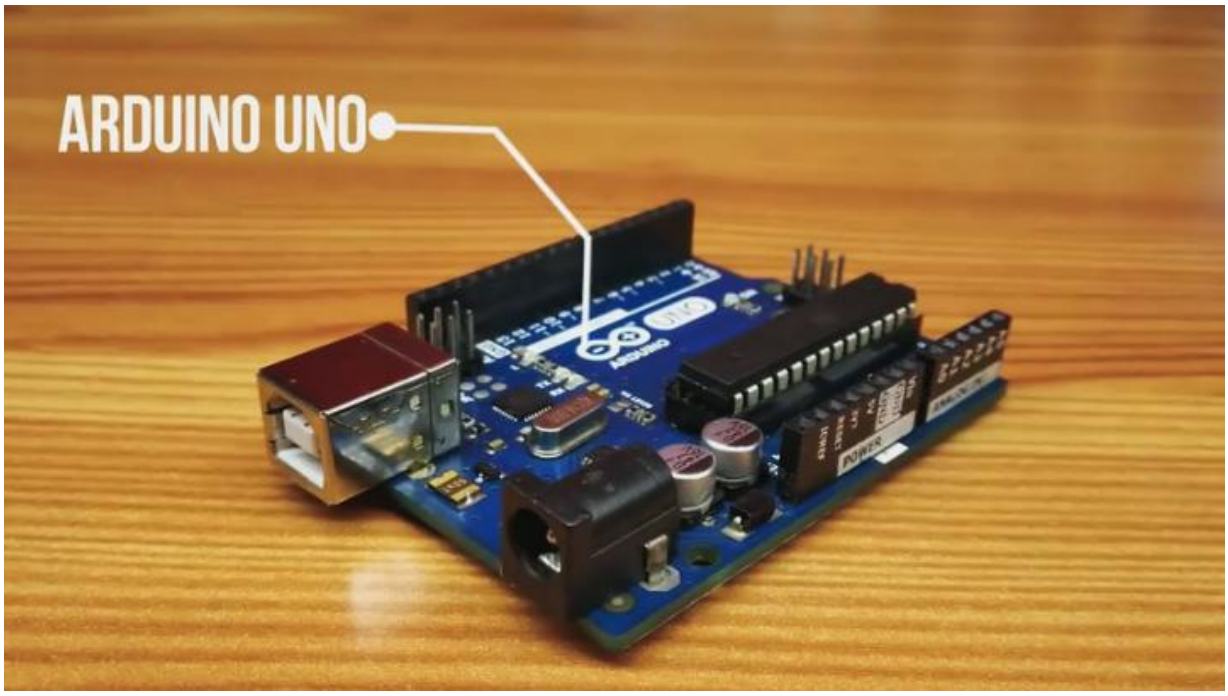
In this article I will show you how to make an Arduino Barrier Avoidance Car at home.

Here is the Component List

Arduino Uno

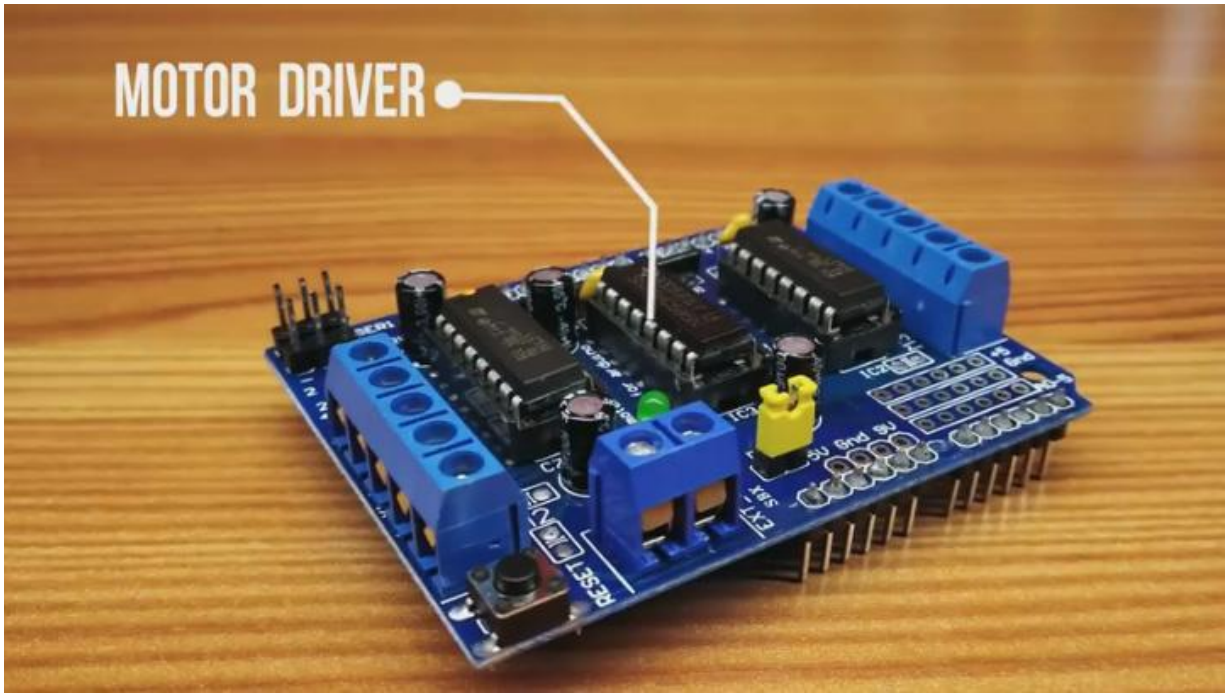
Driver's Shield
Wheels (4x)
TT Gear Motor (4x)
Servo Motor
Ultrasonic Sensor
18650 Li-on Battery (2x)
18650 Battery Holder
Jumpper and Female Phone
Acrylic Sheet
DC Power Switch

Step 1: Building materials

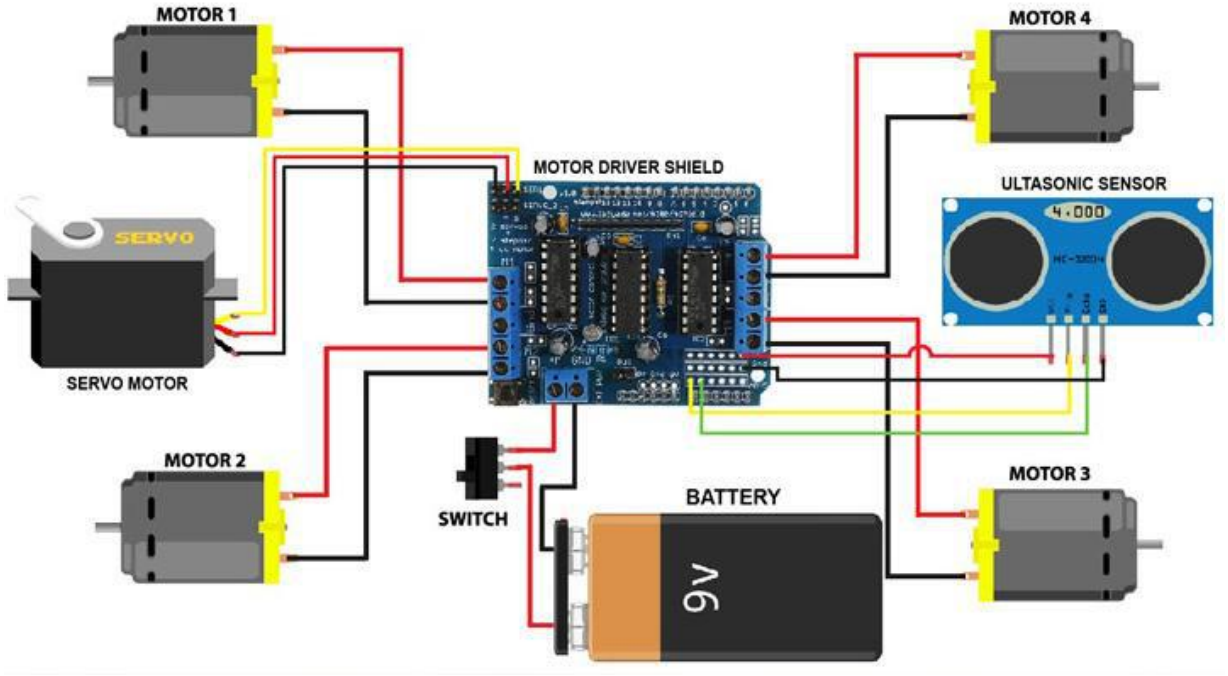


Here are all the things you need.

Step 2: Paste All Items In Place



Step 3: Connect the cables (Circuit diagram)



Step 4: Program Time

```
//ARDUINO OBSTACLE AVOIDING CAR//
// Before uploading the code you have to install the necessary library//
//AFMotor Library https://learn.adafruit.com/adafruit-motor-shield/library-install //
//NewPing Library https://github.com/livetronic/Arduino-NewPing//
//Servo Library https://github.com/arduino-libraries/Servo.git //
// To Install the libraries go to sketch >> Include Library >> Add .ZIP File >> Select the Downloaded ZIP files From the Above links //

#include <AFMotor.h>
#include <NewPing.h>
#include <Servo.h>

#define TRIG_PIN A0
#define ECHO_PIN A1
#define MAX_DISTANCE 200
#define MAX_SPEED 190 // sets speed of DC  motors
#define MAX_SPEED_OFFSET 20

NewPing sonar(TRIG_PIN, ECHO_PIN, MAX_DISTANCE);

AF_DCMotor motor1(1, MOTOR12_1KHZ);
AF_DCMotor motor2(2, MOTOR12_1KHZ);
AF_DCMotor motor3(3, MOTOR34_1KHZ);
AF_DCMotor motor4(4, MOTOR34_1KHZ);
Servo myservo;

boolean goesForward=false;
int distance = 100;
int speedSet = 0;
```

```
void setup() {

  myservo.attach(10);
  myservo.write(115);
  delay(2000);
  distance = readPing() ;
  delay(100);
  distance = readPing();
  delay(100);
  distance = readPing();
  delay(100);
  distance = readPing();
  delay(100);
}

void loop() {
int distanceR = 0;
int distanceL =  0;
delay(40);

if(distance<=15)
{
  moveStop();
  delay(100);
  moveBackward();
  delay(300);
  moveStop();
  delay(200);
  distanceR = lookRight();
```

```
  delay(200);
  distanceL = lookLeft();
  delay(200) ;

  if(distanceR>=distanceL)
  {
    turnRight();
    moveStop();
  }else
  {
    turnLeft();
    moveStop();
  }
}else
{
  moveForward();
}
distance = readPing();
}

int lookRight()
{
    myservo.write(50);
    delay(500);
    int distance = readPing();
    delay(100);
    myservo.write(115);
    return distance;
```

```
}

int lookLeft()
{
    myservo.write(170);
    delay(500);
    int distance = readPing();
    delay(100);
    myservo.write(115);
    return distance;
    delay(100);
}

int readPing() {
  delay(70);
  int cm = sonar.ping_cm();
  if(cm==0)
  {
    cm = 250;
  }
  return cm;
}

void moveStop() {
  motor1.run(RELEASE);
  motor2.run(RELEASE);
  motor3.run(RELEASE);
  motor4.run(RELEASE) ;
  }
```

```
void moveForward() {

if(!goesForward)
  {
    goesForward=true;
    motor1.run(FORWARD);
    motor2.run(FORWARD);
    motor3.run(FORWARD);
    motor4.run(FORWARD);
   for (speedSet = 0; speedSet < MAX_SPEED; speedSet +=2) // slowly bring the speed up to avoid loading down the batteries too quickly
   {
    motor1.setSpeed(speedSet);
    motor2.setSpeed(speedSet);
    motor3.setSpeed(speedSet);
    motor4.setSpeed(speedSet);
    delay(5);
   }
  }
}

void moveBackward() {
    goesForward=false;
    motor1.run(BACKWARD);
    motor2.run(BACKWARD);
    motor3.run(BACKWARD);
    motor4.run(BACKWARD);
  for (speedSet = 0; speedSet < MAX_SPEED; speedSet +=2) // slowly bring the speed up to avoid loading down the batteries too quickly
```

```
  {
    motor1.setSpeed(speedSet);
    motor2.setSpeed(speedSet);
    motor3.setSpeed(speedSet);
    motor4.setSpeed(speedSet);
    delay(5);
  }
}

void turnRight() {
  motor1.run(FORWARD);
  motor2.run(FORWARD);
  motor3.run(BACKWARD);
  motor4.run(BACKWARD);
  delay(500);
  motor1.run(FORWARD);
  motor2.run(FORWARD);
  motor3.run(FORWARD);
  motor4.run(FORWARD);
}

void turnLeft() {
  motor1.run(BACKWARD);
  motor2.run(BACKWARD);
  motor3.run(FORWARD);
  motor4.run(FORWARD);
  delay(500);
  motor1.run(FORWARD);
```

```
  motor2.run(FORWARD);
  motor3.run(FORWARD);
  motor4.run(FORWARD);
}
```

Enter the code provided on your robot ...

Step 5: Testing the Robot ...