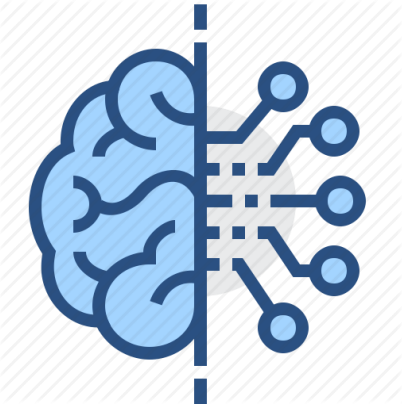
Thinking Machine - 3



How to Read This Book

This book is not about teaching, it's more about exploring. Author is noob trying to learn and know how computer works. Also, he wants to make projects to impress his crush on her birthday.

All the topics are divided into 3 major parts:

1. Why (Why is it important for author to read this new topic).
2. How (How will this topic help author in making effective project).
3. What (What is in the topic to learn).

Complete this book, take free course at:

https://electrovertlabs.com/courses/getting-started-with-arduino/

Give quiz and download Completion certificate. Passing marks 75%

Special Thanks to:

Authors of blogs written on different topics.

By

(This book is compiled by Electroverts lab)

Before starting, author wants to convey some message:

- It's totally okay if you don't know anything and completely new to Computers and Arduino.
  The fact that you are reading this makes you enter the group of people who wants to explore possibilities of the new world where everyone gets chance to do something meaningful and productive to change the world for good.

- Author is complete noob (new comer) in this world of computers. He don't want to teach anybody anything. He is just sharing whatever he is exploring and trying to make it simple.

- This book is not written for exam point of view, instead here author wants to explain how to think.

- Practice 5 Why concept. To dig deep in any topic, keep asking why, why 5 times and you'll know root cause.

- There is no point of memorizing anything, consider this book as a story of Computers.

- If you feel something is important, repeat it one more time.

- **Learn the art of using google, it will help you a lot to clear your concepts and get answers of different questions going in your mind.**
- **Learn the art of using google, it will help you a lot to clear your concepts and get answers of different questions going in your mind.**

- Try not to skip theory part as knowing what you are using (computer) will give you visibility on what exactly is happening when you write codes or use Instagram.

- Try to maintain consistency throughout, learning how to write code is like playing video games. You'll easily get bored and feel weak in beginning and with time you'll be able to think logically on your own and deploy strategies to conquer.

Contents

Contents iii

Getting Started 12

Required Software 17

Mac OS X 17

Windows XP and Later 19

Ubuntu Linux 9.04 and Later 20

Let's Plug Things 22

What is a microcontroller? 22

Inside a Microcontroller: Essential Components 22

Design of Microcontroller CPU 23

Microcontroller RAM 24

Use of Flash Memory in Microcontrollers 24

What is EEPROM in Microcontrollers? 24

Serial Bus Interface 24

Microcontroller I/O Ports 24

The Arduino Board 25

What exactly is Arduino? 25

Arduino - Board Description (Detailed) 29

Types of Arduino Boards 31

Arduino UNO 31

Arduino Mega 32

Arduino Esplora 32

Arduino Nano 32

Arduino MKR ZERO 33

Arduino Yún 33

Lilypad Arduino 33

ESP32 34

Nodemcu 34

WIOT Board 34

Current & Voltage limitations for Arduino UNO 35

Power 35

Power and Aux I/O Port 35

Input and Outputs 36
Using Analog Pins as Digital Pins 36
Taking a Look around the IDE 37
The Command Area 38
Menu Items 38
Creating Your First Sketch in the IDE 39
Comments 39
The Setup Function 40
Controlling the Hardware 40
The Loop Function 41
Verifying Your Sketch 42
Uploading and Running Your Sketch 43
Modifying Your Sketch 44
Arduino Programming 45
Structure 45
1. setup() 45
2. loop() 45
3. Functions 46
4. {} curly braces 46
5. ; semicolon 47
6. /*… */ block comments 47
7. // line comments 47
variables 48
1. variables 48
2. variable declaration 48
3. variable scope 49
datatypes 49
1. byte 49
2. int 50
3. long 50
4. float 50
5. arrays 50
arithmetic 51
1. arithmetic 51

2. compound assignments 52
3. comparison operators 52
4. logical operators 52
constants 53
1. constants 53
2. true/false 53
3. high/low 53
4. input/output 53
flow control 54
1. if 54
2. if… else 54
3. for 55
4. while 56
5. do… while 56
digital i/o 57
1. pinMode(pin, mode) 57
2. digitalRead(pin) 57
3. digitalWrite(pin, value) 57
analog i/o 58
1. analogRead(pin) 58
2. analogWrite(pin, value) 58
time 59
1. delay(ms) 59
2. millis() 59
math 59
1. min(x, y) 59
2. max(x, y) 59
random 60
1. randomSeed(seed) 60
2. random(max), random(min, max) 60
serial 60
1. Serial.begin(rate) 60
2. Serial.println(data) 61
digital output 61

digital input 62
high current output 63
pwm output 64
potentiometer input 65
variable resistor input 65
servo output 66
Let's get started 68
Planning Your Projects 68
About Electricity 69
Current 69
Voltage 69
Power 69
Electronic Components 69
The Resistor 70
Resistance 70
Reading Resistance Values 70
Chip Resistors 71
Multimeters 72
The Light-Emitting Diode (LED) 72
The Solderless Breadboard 74
Getting Started with Arduino projects 76
Project #1: LED Flasher 76
The Algorithm 76
The Hardware 76
The Schematic 76
The Sketch 77
Running the sketch 77
Code Overview 77
Project #2: Creating a Blinking LED Wave 81
The Algorithm 81
The Hardware 81
The Schematic 81
The Sketch 82

Running the Sketch 83
Using Variables 83
Project #3: Repeating with for Loops 84
Varying LED Brightness with Pulse-Width Modulation 86
Project #4: Demonstrating PWM 87
Hardware 87
The Schematic 87
The Sketch 87
Code Overview 88
Project #5: Traffic Lights 89
The Algorithm 89
Hardware 89
The Schematic 90
The Sketch 90
Code Overview 91
Project #6 LED Chase Effect 91
The Algorithm 91
Hardware 91
The Schematic 92
The Sketch 92
Code Overview 93
How analog pins work & how we take analog input? 96
What is the ADC? 96
Project #7 Using potentiometer to control brightness 97
Hardware 97
The Schematic 98
The Sketch 98
Code Overview 99
Project #8 Interactive LED Chase Effect 100
Hardware 100
The Schematic 100
The Sketch 101
Code Overview 101
Displaying Data from the Arduino in the Serial Monitor 102

Starting the Serial Monitor 102
Sending Text to the Serial Monitor 103
Displaying the Contents of Variables 103
Project #9 Using serial monitor to check potentiometer input 104
Hardware 104
The Sketch 104
The Schematic 105
Serial Monitor output 105
Sending Data from the Serial Monitor to the Arduino 106
Project #10 Multiplying a Number by 2 106
The Sketch 106
Output 107
Project #11 LED chase effect with speed controlled by Serial Monitor 108
Hardware 108
The Schematic 108
The Sketch 108
Code Overview 110
Output 110
RGB LED 111
How do RGB LEDs work? 111
How to create different colors? 111
Mixing colors 111
Common Anode and Common Cathode RGB LEDs 112
RGB LED Pins 113
Project #12 Using RBG LED to get different colours 113
Hardware 113
The Schematic 114
The Sketch 114
Code Overview 115
Project #13 Using random function to generate random colour through RBG LED 116
Random Function 116
WHY ARE RANDOM NUMBERS WITH ARDUINO ALL THE SAME? 116
ONE SOLUTION TO THE RANDOM PROBLEM 117

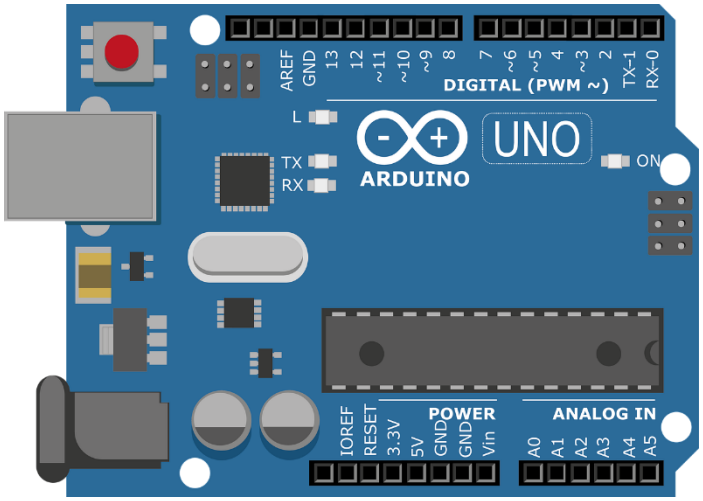
Hardware 117
The Schematic 118
The Sketch 118
Code Overview 119
Project #14 Using Serial monitor for obtaining different colour combinations 120
Hardware 120
Schematic 120
The Sketch 121
Output 122
Project #15 Controlling RBG colour by 3 potentiometer 124
Hardware 124
The Schematic 124
The Sketch 125
Code Overview 125
Project #16 Using TONE() function with Arduino 126
A QUICK INTRO TO PIEZO SPEAKERS (AKA PIEZO BUZZERS) 126
Hardware 126
The Schematic 126
The Sketch 127
Project #17 Piezo Sounder Alarm 131
Hardware 131
The Schematic 131
The Sketch 132
Code Overview 132
Project #18 LDR – The Light Sensor 134
What is an LDR? 134
Working Principle of LDR 134
Construction of an LDR 135
Hardware 136
The Schematic 137
The Sketch 137
Output 138
Project #19 Light automation using LDR 139
Hardware 139

The Schematic 139
The Sketch 140
Output 141
Project #20 Varying intensity of LED as outside light intensity 142
Hardware 142
The Schematic 142
The Sketch 143
Output 143
Project #21 Ultrasonic Distance sensor 144
What is Ultrasonic Sensor & how it works? 144
Hardware 147
The Schematic 147
The Sketch 148
Output 149
pulseIn() function 149
Project #22 Parking Alarm using Ultrasonic Sensor 151
Hardware 151
The Schematic 152
The Sketch 152
Output 154
Project #23 Water level detector 154
Hardware 154
The Schematic 155
The Sketch 155
Code Overview 157
Output 158
Project #24 TMP36 Temperature sensor 159
Hardware 159
About TMP36 Temperature sensor 159
How to measure Temperature 159
The Schematic 160
The Sketch 160
Output 161
Project #25 Temperature alarm 162

Hardware 162
The Schematic 163
The Sketch.....163
Output 165

Getting Started

Have you ever looked at some gadget and wondered how it really worked? Maybe it was a remote control boat, the system that controls an elevator, a vending machine, or an electronic toy? Or have you wanted to create your own robot or electronic signals for a model railroad, or perhaps you'd like to capture and analyze weather data over time? Where and how do you start? The Arduino board can help you find some of the answers to the mysteries of electronics in a hands-on way. The original creation of Massimo Banzi and David Cuartielles, the Arduino system offers an inexpensive way to build interactive projects, such as remote-controlled robots, GPS tracking systems, and electronic games. The Arduino project has grown exponentially since its introduction in 2005. It's now a thriving industry, **supported by a community of people united with the common bond of creating something new**. You'll find both individuals and groups, ranging from interest groups and clubs to local hackerspaces and educational institutions, all interested in toying with the Arduino.



To get a sense of the variety of Arduino projects in the wild, you can simply search the Internet. You'll find a list of groups offering introductory programs and courses with like-minded, creative people.

The Possibilities Are Endless

A quick scan through this course will show you that you can use the Arduino to do something as simple as blinking a small light, or even something more complicated, such as interacting with a cellular phone—and many different things in between. For example, have a look at Philip Lindsay's device, shown in Figure below. It can receive text messages from cellular phones and display them on a large sign for use in dance halls. This device uses an Arduino board and a cellular phone shield to receive text messages from other phones. The text message is sent to a pair of large, inexpensive dotmatrix displays for everyone to see.

SMS (short message service) text marquee

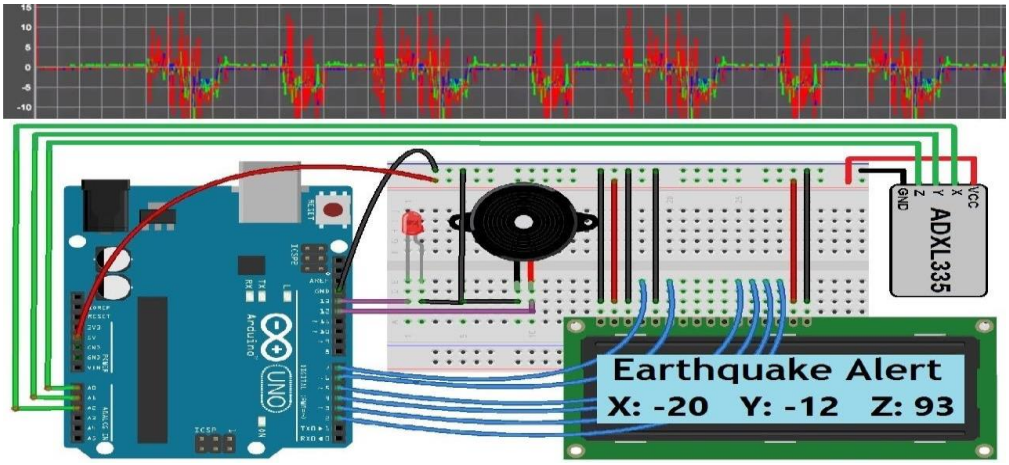
How about creating a unique marriage proposal? Tyler Cooper wanted an original way to propose to his girlfriend, so he built what he calls a "reverse geocache box"—a small box that contained an engagement ring, as shown in Figure below. When the box was taken to a certain area (measured by the internal GPS), it unlocked to reveal a romantic message and the ring. You can easily reproduce this device using an Arduino board, a GPS receiver, and an LCD module, with a small servo motor that acts as a latch to keep the box closed until it's in the correct location. The code required to create this is quite simple—something you could create in a few hours. The most time-consuming part is choosing the appropriate box in which to enclose the system.

Here's another example. Kurt Schulz was interested in monitoring the battery charge level of his moped. However, after realizing how simple it is to work with Arduino, his project morphed into what he calls the "Scooterputer": a complete moped management system. The Scooterputer can measure the battery voltage, plus it can display the speed, distance traveled, tilt angle, temperature, time, date, GPS position, and more. It also contains a cellular phone shield that can be controlled remotely, allowing remote tracking of the moped and engine shutdown in case it's stolen. The entire system can be controlled with a small touchscreen, shown in below. Each feature can be considered a simple building block, and anyone could create a similar system in a couple of weekends.

The Scooterputer display (courtesy of Kurt Schulz)

Earthquake detector:

Normally, you depend on the government officials to announce/inform about the earthquake stats (or the warning for it).
But with Arduino boards, you can simply build a basic earthquake detector and have transparent results for yourself without depending on the authorities. Click on the button below to know about the relevant details to help make it.



Build a Smoke Detection System Using the MQ-2 Gas Sensor

This will definitely sit on top of the best Arduino projects you can take on this year. The project is not only fancy regarding presentation but also solves a real-life problem in an exceptionally well laid out manner. Your project will consist of a system that detects smoke and other inflammatory gases by utilizing the MQ-2 gas sensor.

It will feature a buzzer that gets turned on whenever a certain level of smoke is detected by the system. An LED indicator will also be featured that will turn red when the system senses such gases and remain green when the environment is safe.

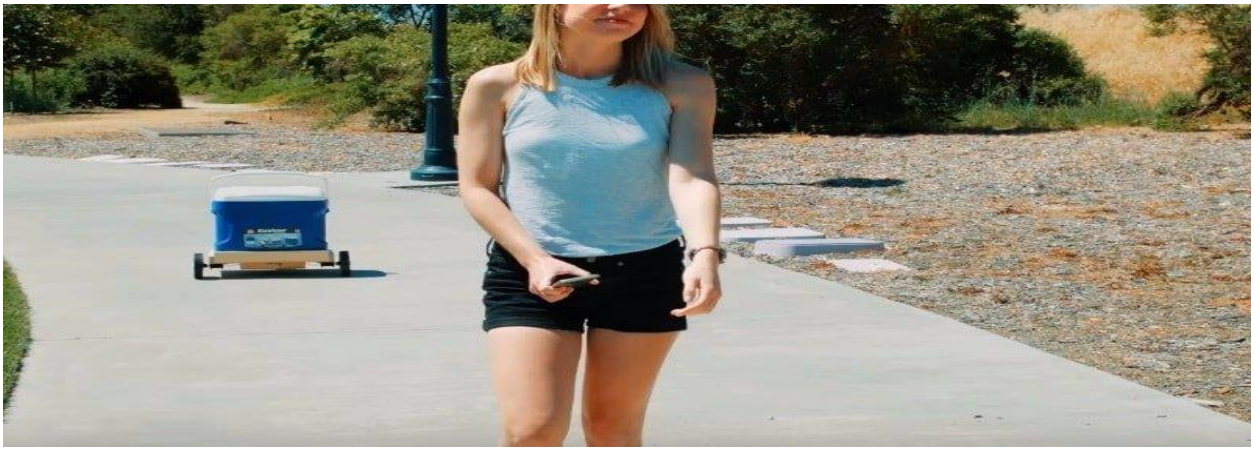
Build a Gesture Control with Arduino

One of the best Arduino projects regarding satisfaction; this is something even seasoned developers will sweat to build. The project requires you control an Arduino project by gesturing your hand. Let's imagine you're controlling the robot car project you built earlier by merely moving your hands rather than using an external controller. Sounds eclectic, right? You need tools such as an accelerometer, a gyroscope and a magnetometer to effectively build such a project.

Build an Autonomous "Follow Me" Cooler

A fun yet rewarding project to build for seasoned developers; this is one of the best Arduino projects that come handy in real-life scenarios. In this project, you will build an autonomous cooler that will follow you wherever you go. Sounds fun, right?

You can find an unused cooler from pawn shops or even buy a low-end version yourself. Create a wooden base which will include the wheels for transporting the cooler. This project will make sure your drinks stay cool and right behind wherever you go!



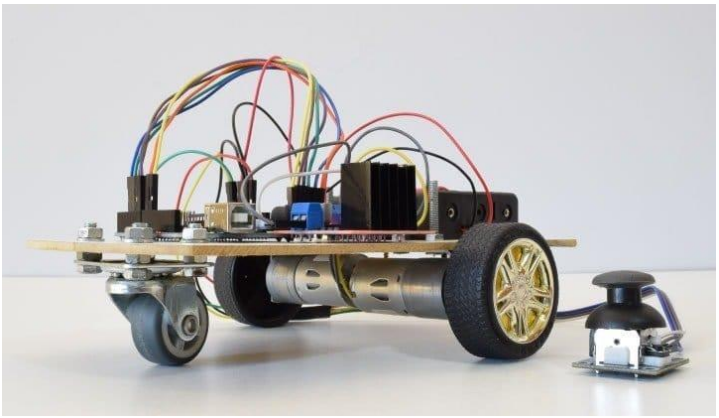
Build a Quadruped Using Arduino

This is a great Arduino project for people with interest in the field of robotics. This biologically inspired Quadruped is similar in outlook to a spider and requires you to carefully construct the outer materials.

Build a Robot Car

Nothing beats the combination of DC motors and Arduino when it comes to building awe-inspiring electronics projects. So, one of the best Arduino projects that you can take on this year is building a robot car from scratch.

You can only use household materials or may opt with 3D printing your car's components if aiming for a visually stunning robot. This project is not only fascinating in visual but also rewards you with a deep understanding of advanced motor control.



Required Software

You should be able to program your Arduino with just about any computer using a piece of software called an integrated development environment (IDE) for Arduino. To run this software, your computer should have one of the following operating systems installed:

- Mac OS X or higher
- Windows XP 32- or 64-bit, or higher
- Linux 32- or 64-bit (Ubuntu or similar)

Now is a good time to download and install the Arduino IDE, so jump to the heading that matches your operating system and follow the instructions. Make sure you have or buy the matching USB cable for your Arduino from the supplier as well. Even if you don't have your Arduino board yet, you can still download and explore the IDE. Because the IDE version number can change quite rapidly, the number in this course may not match the current version, but the instructions should still work.

Mac OS X

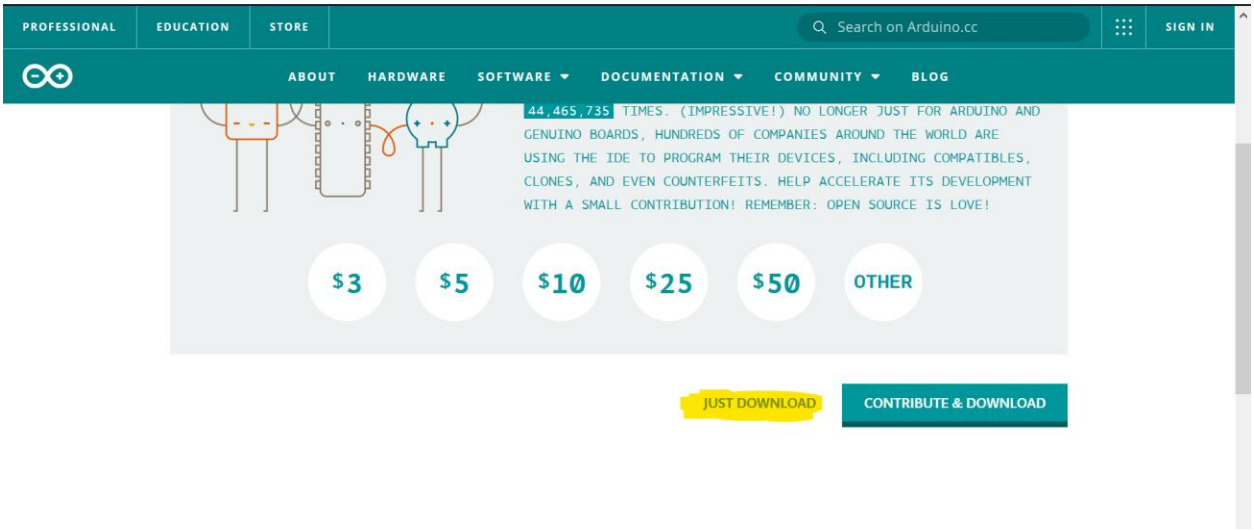
In this section, you'll find instructions for downloading and configuring the Arduino IDE in Mac OS X.

Installing the IDE

To install the IDE on your Mac, follow these instructions:

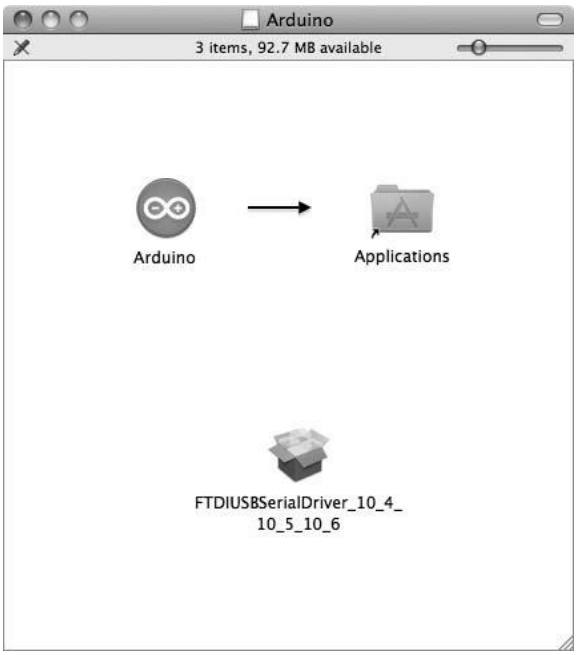
1. Using a web browser such as Safari, visit the software download page located at http://arduino.cc/en/Main/Software/

Click on the highlighted hyperlinked text and then chick on download in next page.

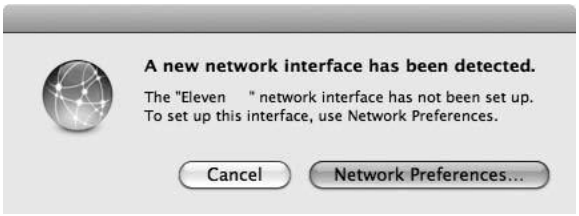


2. Once it's finished downloading, double-click the file to start the installation process. You will then be presented with the window shown below.

3. Drag the Arduino icon over the Applications folder and release the mouse button. A temporary status window will appear as the file is copied.



4. Now connect your Arduino to your Mac with the USB cable. After a moment, the dialog shown below will appear.



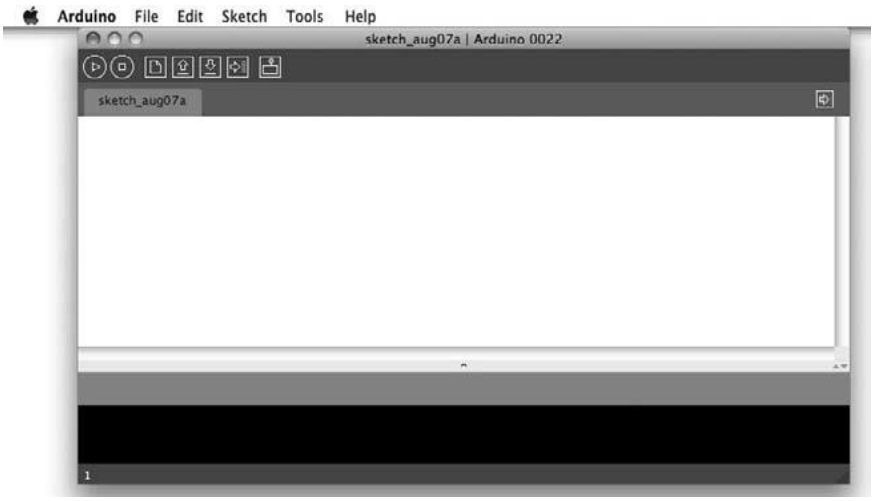
5. Click **Network Preferences**..., and then click **Apply** in the Network box. You can ignore the "not configured" status message.

Setting up the IDE

Once you have downloaded the IDE, use the following instructions to open and configure the IDE:

1. Open the Applications folder in Finder and double-click the Arduino icon.

2. A window may appear warning you about opening a web app. If it does, click Open to continue. You will then be presented with the IDE, as shown in Figure below.

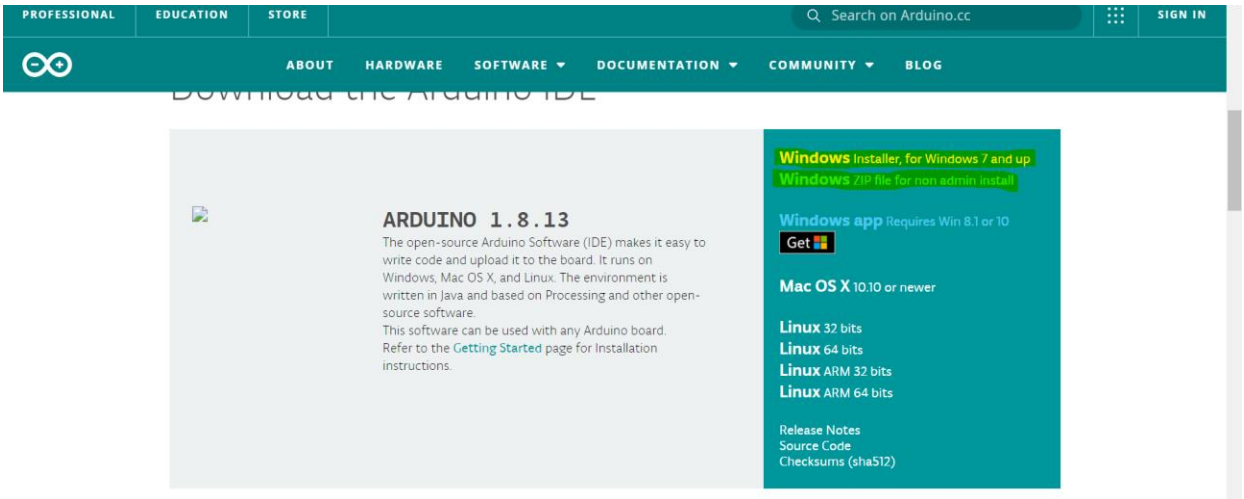


Windows XP and Later

In this section, you'll find instructions for downloading the IDE, installing drivers, and configuring the IDE in Windows.

To install the Arduino IDE for Windows, follow these instructions:

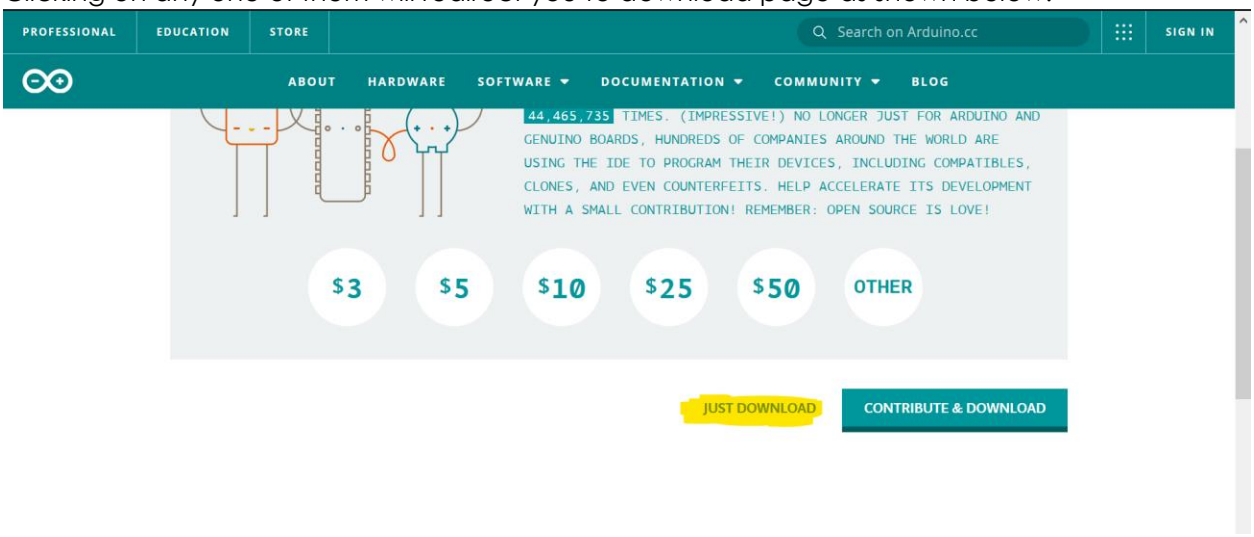
1. Using a web browser such as Firefox, visit the software download page located at http://arduino.cc/en/Main/Software/



There are 3 methods to install Arduino IDE in Windows.

1. Windows installer (Highlighted in yellow), it will download .exe file which you can install in your windows pc.
2. Windows zip file (Highlighted in green), it downloads zip file which you have to unzip and then use it as IDE launcher.
3. Window app (Highlighted in blue), it will redirect you to Microsoft store from where you can download the application on your pc.

Clicking on any one of them will redirect you to download page as shown below.



Ubuntu Linux 9.04 and Later

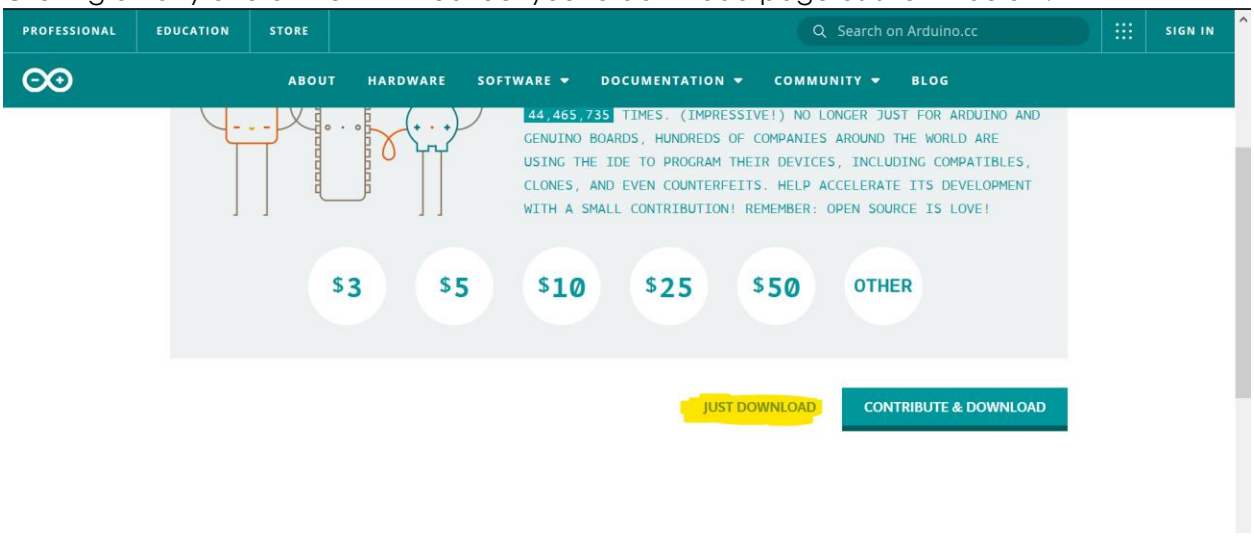
If you are running Ubuntu Linux, here are instructions for downloading and setting up the Arduino IDE.

Installing the IDE

Use the following instructions to install the IDE:

1. Using a web browser such as Firefox, visit the software download page located at http://arduino.cc/en/Main/Software/

Clicking on any one of them will redirect you to download page as shown below.



Download the file, run it and we are ready with next step.

Let's Plug Things

Arduino is all about plugging things. We are going to do that in a couple of minutes after we have learned a bit more about microcontrollers in general and especially the big and amazing Arduino family. This chapter is going to teach you how to be totally ready to code, wire, and test things with your new hardware friend. Yes, this will happen soon, very soon; now let's dive in!

What is a microcontroller?

A microcontroller is an integrated circuit (IC) containing all main parts of a typical computer, which are as follows:

- Processor
- Memories
- Peripherals
- Inputs and outputs

The processor is the brain, the part where all decisions are taken and which can calculate. Memories are often both spaces where both the core inner-self program and the user elements are running (generally called Read Only Memory (ROM) and Random Access Memory (RAM)). I define peripherals by the self-peripherals contained in a global board; these are very different types of integrated circuits with a main purpose: to support the processor and to extend its capabilities.

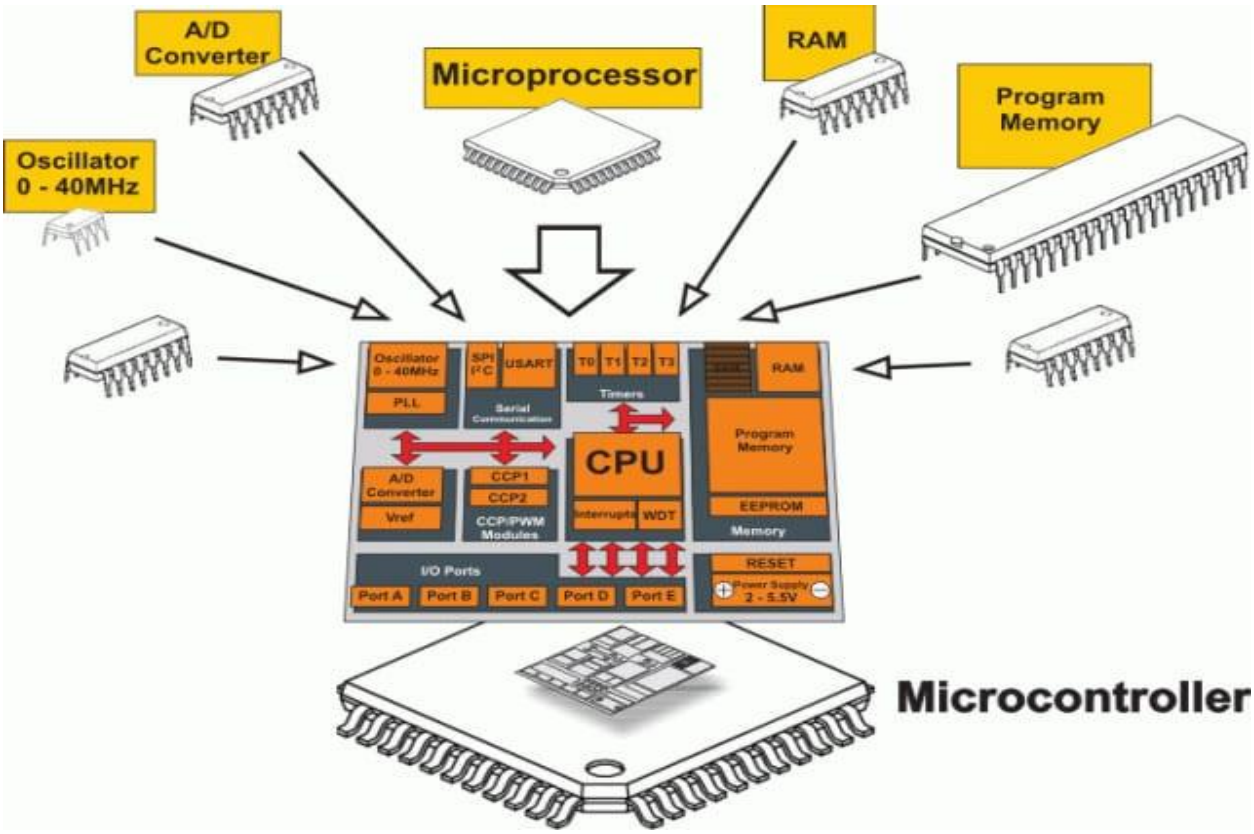
Inputs and outputs are the ways of communication between the world (around the microcontroller) and the microcontroller itself.

The very first single-chip processor was built and proposed by Intel Corporation in 1971 under the name Intel 4004. It was a 4-bit central processing unit (CPU). Since the 70s, things have evolved a lot and we have a lot of processors around us. Look around, you'll see your phone, your computer, and your screen. Processors or microprocessors drive almost everything. Compared to microprocessors, microcontrollers provide a way to reduce power consumption, size, and cost. Indeed, microprocessors, even if they are faster than processors embedded in microcontrollers, require a lot of peripherals to be able to work. The high-level of integration provided by a microcontroller makes it the friend of embedded systems that are car engine controller, remote controller of your TV, desktop equipment including your nice printer, home appliances, games of children, mobile phones, and I could continue…

Inside a Microcontroller: Essential Components

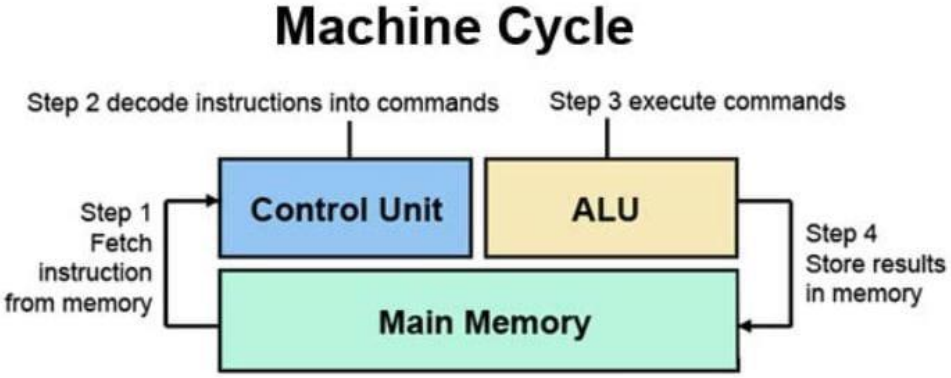
A microcontroller can be seen as a small computer, and this is because of the essential components inside of it; the Central Processing Unit (CPU), the Random-Access Memory (RAM), the Flash Memory, the Serial Bus Interface, the Input/Output Ports (I/O Ports), and in many cases, the Electrical Erasable Programmable Read-Only Memory (EEPROM). Figure below shows a great

diagram of the main parts and also other parts in the microcontroller. Let's dive into each of these components and see how they work inside the microcontroller.



Design of Microcontroller CPU

The CPU, sometimes called a processor or microprocessor, controls all of the instructions/data flow that it receives. You can think of it as the brains of the system, processing all the data input it receives and executes the required instructions. Its two main components are the Arithmetic Logic Unit (ALU), which performs arithmetic and logical operations, and the Control Unit (CU), which handles all of the processor's instruction executions. Figure below shows a usual "machine cycle" that the CPU goes through.



Microcontroller RAM

RAM is a component that temporarily stores data, and can be accessed quickly. It provides quick read-and-write access to the storage device. This differs from most other memories as it takes longer for data to be extracted since the data isn't readily available. You can see it as RAM having access to the surface of data – easily reachable – but anything that dives deeper will require a different type of memory. RAM improves total system performance because it allows the microcontroller to work with more information at the same time. Since RAM is temporary data, its content is always erased when the microcontroller is shut down.

Use of Flash Memory in Microcontrollers

Flash Memory is a type of non-volatile memory that, unlike RAM, retains its data for an extended period, even if the microcontroller is turned off. This keeps the saved program that you might have uploaded to the microcontroller. Flash Memory writes to a "block" or "sector" at a time, so if you need to just re-write one byte, Flash Memory will need to re-write the whole block that the byte is in, which can wear out quicker.

What is EEPROM in Microcontrollers?

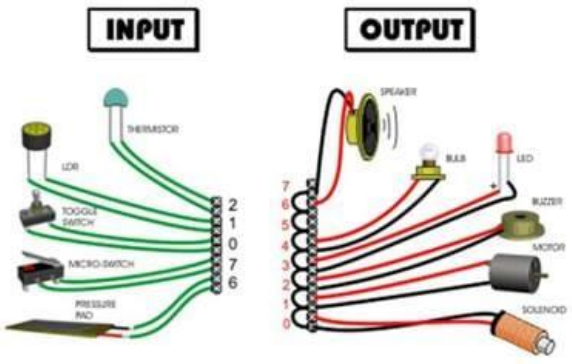
EEPROM is like Flash Memory, being a non-volatile memory and retaining its data even after shutdown. The difference is that, while Flash Memory re-writes a "block" of bytes, EEPROM can re-write any specific byte at any time. This extends the life of EEPROM compared to Flash Memory, but also means that it is more expensive.

Serial Bus Interface

A Serial Bus Interface is the serial communication in the microcontroller, sending data one bit at a time. With microcontroller boards, it connects ICs with signal traces on a printed circuit board (PCB). For ICs, they use serial bus to transfer data to reduce the number of pins in a package making them more cost effective. Examples of serial buses in ICs are SPIs or I2Cs.

Microcontroller I/O Ports

I/O ports are what the microcontroller uses to connect to real-world applications. Inputs receive changes in the real-world, from temperature sensing, to motion sensing, to push buttons, and much more. The input then goes to the CPU and decides what to do with that information. When it's time to do a certain command based on a certain value from the input, it sends a signal to the output ports, where it can range from a simple LED light going off, to running a motor for a certain part, to many more. Figure below shows some common input and output components.

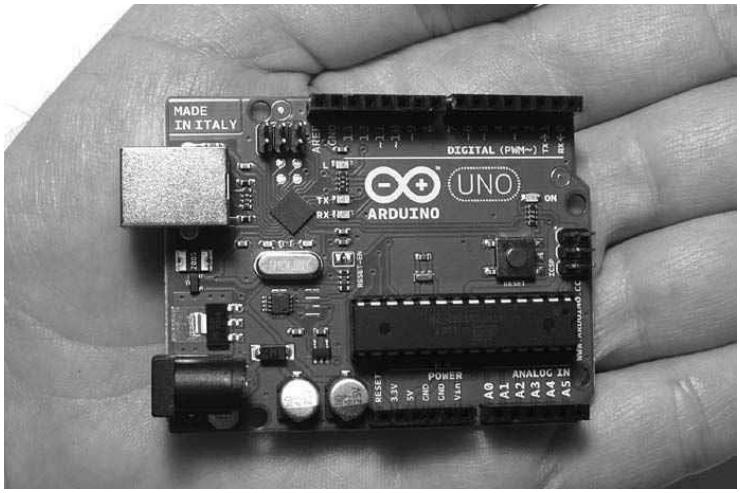


The Arduino Board

What exactly is Arduino?

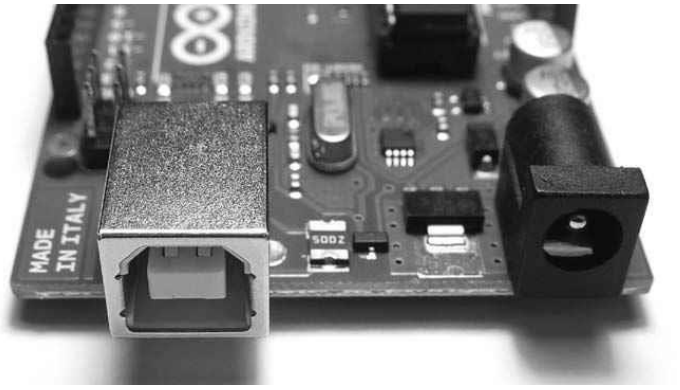
According to the Arduino website (http://www.arduino.cc/), it is an open-source electronics prototyping platform based on flexible, easy-to-use hardware and software. It's intended for artists, designers, hobbyists, and anyone interested in creating interactive objects or environments.

In simple terms, the Arduino is a tiny computer system that can be programmed with your instructions to interact with various forms of input and output. The current Arduino board model, the Uno, is quite small in size compared to the average human hand.



Although it might not look like much to the new observer, the Arduino system allows you to create devices that can interact with the world around you. By using an almost unlimited range of input and output devices, sensors, indicators, displays, motors, and more, you can program the exact interactions required to create a functional device. For example, artists have created installations with patterns of blinking lights that respond to the movements of passers-by, high school students have built autonomous robots that can detect an open flame and extinguish it, and geographers have designed systems that monitor temperature and humidity and transmit this data back to their offices via text message. In fact, you'll find an almost infinite number of examples with a quick search on the Internet. Now let's move on and explore our Arduino Uno hardware (in other words, the "physical part") in more detail and see what we have. Don't worry too much about understanding what you see here, because all these things will be discussed in greater detail in later chapters.

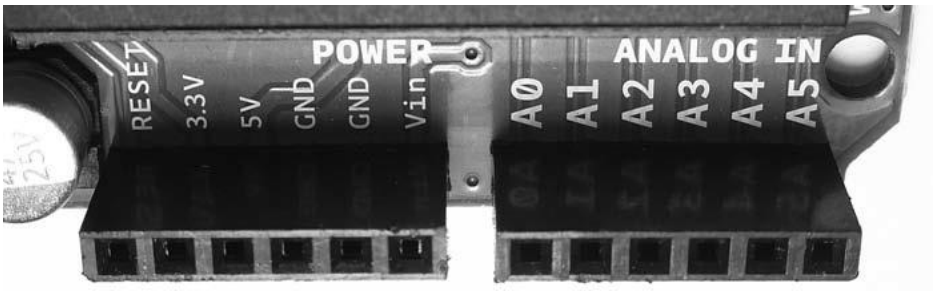
Let's take a quick tour of the Uno. Starting at the left side of the board, you'll see two connectors, as shown in figure below.



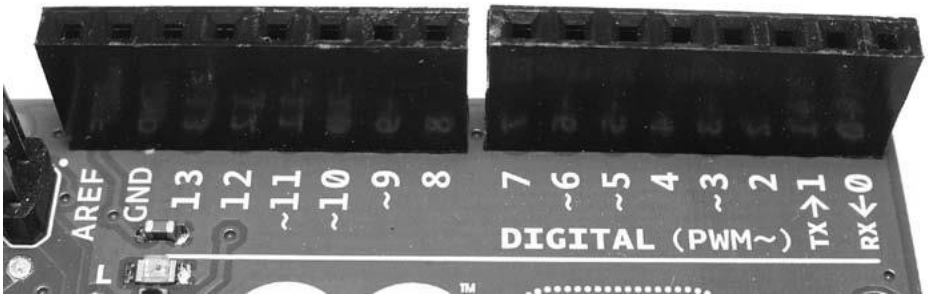
On the far left is the Universal Serial Bus (USB) connector. This connects the board to your computer for three reasons: to supply power to the board, to upload your instructions to the Arduino, and to send data to and receive it from a computer. On the right is the power connector. Through this connector, you can power the Arduino with a standard mains power adapter.

At the lower middle is the heart of the board: the microcontroller.

The microcontroller is the "brains" of the Arduino. It is a tiny computer that contains a processor to execute instructions, includes various types of memory to hold data and instructions from our sketches, and provides various avenues of sending and receiving data. Just below the microcontroller are two rows of small sockets, as shown below.



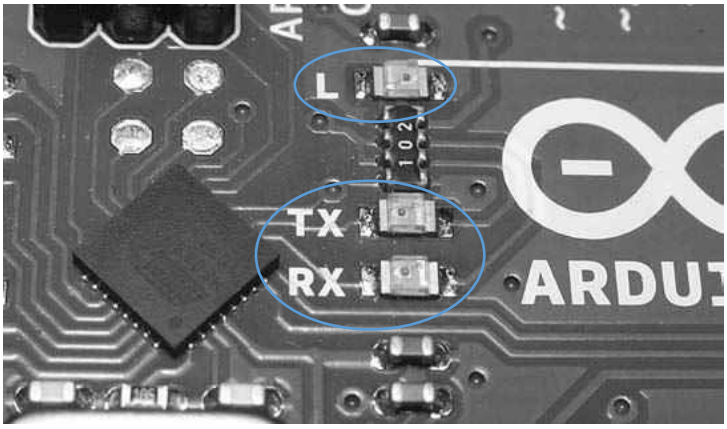
The first row offers power connections and the ability to use an external RESET button. The second row offers six analog inputs that are used to measure electrical signals that vary in voltage. Furthermore, pins A4 and A5 can also be used for sending data to and receiving it from other devices. Along the top of the board are two more rows of sockets, as shown in Figure below.



Sockets (or pins) numbered 0 to 13 are digital input/output (I/O) pins. They can either detect whether or not an electrical signal is present or generate a signal on command. Pins 0 and 1 are also known as the serial port, which is used to send and receive data to other devices, such as a computer via the USB connector circuitry.

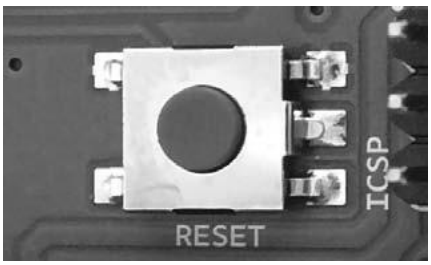
The pins labeled with a tilde (~) can also generate a varying electrical signal (PWM pins), which can be useful for such things as creating lighting effects & intensity or controlling speed of electric motors.

Next are some very useful devices called light-emitting diodes (LEDs); these very tiny devices light up when a current passes through them. The Arduino board has four LEDs: one on the far right labeled ON, which indicates when the board has power, and three in another group, as shown in Figure below.



The LEDs labeled TX and RX light up when data is being transmitted or received between the Arduino and attached devices via the serial port and USB. The L LED is for your own use (it is connected to the digital I/O pin number 13). The little black square part to the left of the LEDs is a tiny microcontroller that controls the USB interface that allows your Arduino to send data to and receive it from a computer, but you don't generally have to concern yourself with it.

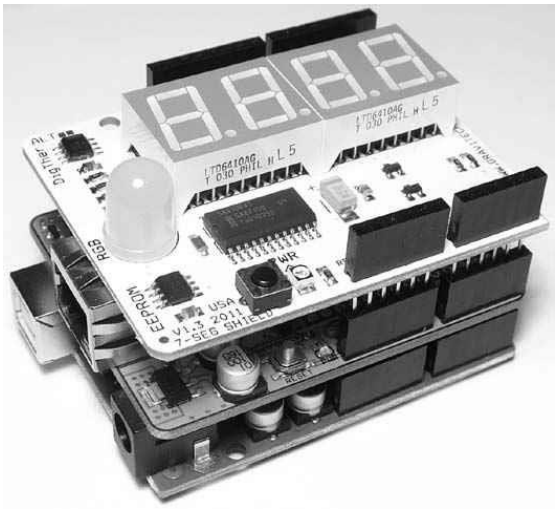
And, finally, the RESET button is shown in Figure below.



As with a normal computer, sometimes things can go wrong with the Arduino, and when all else fails, you might need to reset the system and restart your Arduino. This simple RESET button on the board (Figure above) is used to restart the system to resolve these problems.

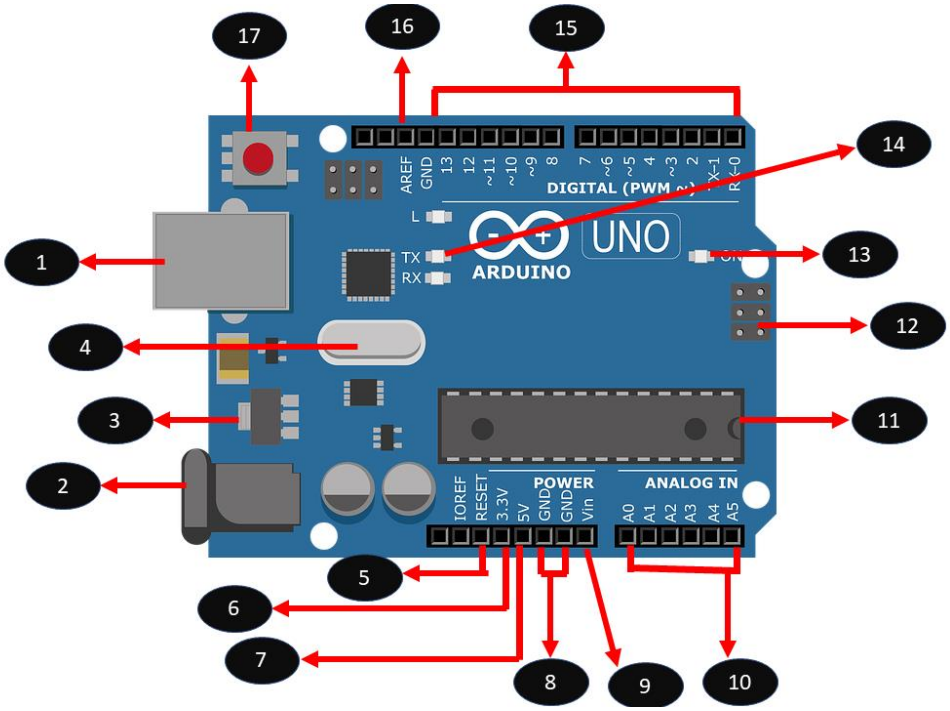
One of the great advantages of the Arduino system is its ease of expandability—that is, it's easy to add more hardware functions. The two rows of sockets along each side of the Arduino allow the connection of a shield, another circuit board with pins that allow it to plug into the Arduino. For example, the shield shown in Figure below contains an Ethernet interface that allows the Arduino to communicate over networks and the Internet, with plenty of space for custom circuitry.

Notice how the Ethernet shield also has rows of sockets. These enable you to insert one or more shields on top. For example, Figure below shows that another shield with a large numeric display, temperature sensor, extra data storage space, and a large LED has been inserted. Note that you do need to remember which shield uses which individual inputs and outputs to ensure that "clashes" do not occur. You can also purchase completely blank shields that allow you to add your own circuitry.



The companion to the Arduino hardware is the software, a collection of instructions that tell the hardware what to do and how to do it. Two types of software can be used: The first is the integrated development environment (IDE), and the second is the Arduino sketch you create yourself. The IDE software is installed on your personal computer and is used to compose and send sketches to the Arduino board.

Arduino - Board Description (Detailed)



1	**Power USB** Arduino board can be powered by using the USB cable from your computer. All you need to do is connect the USB cable to the USB connection (1).
2	**Power (Barrel Jack)** Arduino boards can be powered directly from the AC mains power supply by connecting it to the Barrel Jack (2).
3	**Voltage Regulator** The function of the voltage regulator is to control the voltage given to the Arduino board and stabilize the DC voltages used by the processor and other elements.
4	**Crystal Oscillator** The crystal oscillator helps Arduino in dealing with time issues. How does Arduino calculate time? The answer is, by using the crystal oscillator. The number printed on top of the Arduino crystal is 16.000H9H. It tells us that the frequency is 16,000,000 Hertz or 16 MHz.
5,17	**Arduino Reset** You can reset your Arduino board, i.e., start your program from the beginning. You can reset the UNO board in two ways. First, by using the reset button (17) on the

	board. Second, you can connect an external reset button to the Arduino pin labelled RESET (5).
6,7, 8,9	**Pins (3.3, 5, GND, Vin)** • 3.3V (6) − Supply 3.3 output volt • 5V (7) − Supply 5 output volt • Most of the components used with Arduino board works fine with 3.3 volt and 5 volt. • GND (8)(Ground) − There are several GND pins on the Arduino, any of which can be used to ground your circuit. • Vin (9) − This pin also can be used to power the Arduino board from an external power source, like AC mains power supply.
10	**Analog pins** The Arduino UNO board has six analog input pins A0 through A5. These pins can read the signal from an analog sensor like the humidity sensor or temperature sensor and convert it into a digital value that can be read by the microprocessor.
11	**Main microcontroller** Each Arduino board has its own microcontroller (11). You can assume it as the brain of your board. The main IC (integrated circuit) on the Arduino is slightly different from board to board. The microcontrollers are usually of the ATMEL Company. You must know what IC your board has before loading up a new program from the Arduino IDE. This information is available on the top of the IC. For more details about the IC construction and functions, you can refer to the data sheet.
12	**ICSP pin** Mostly, ICSP (12) is an AVR, a tiny programming header for the Arduino consisting of MOSI, MISO, SCK, RESET, VCC, and GND. It is often referred to as an SPI (Serial Peripheral Interface), which could be considered as an "expansion" of the output. Actually, you are slaving the output device to the master of the SPI bus.
13	**Power LED indicator** This LED should light up when you plug your Arduino into a power source to indicate that your board is powered up correctly. If this light does not turn on, then there is something wrong with the connection.
14	**TX and RX LEDs** On your board, you will find two labels: TX (transmit) and RX (receive). They appear in two places on the Arduino UNO board. First, at the digital pins 0 and 1, to indicate the pins responsible for serial communication. Second, the TX and RX led (13). The TX led flashes with different speed while sending the serial data. The speed of flashing depends on the baud rate used by the board. RX flashes during the receiving process.

15	**Digital I/O** The Arduino UNO board has 14 digital I/O pins (15) (of which 6 provide PWM (Pulse Width Modulation) output. These pins can be configured to work as input digital pins to read logic values (0 or 1) or as digital output pins to drive different modules like LEDs, relays, etc. The pins labeled "~" can be used to generate PWM.
16	**AREF** AREF stands for Analog Reference. It is sometimes, used to set an external reference voltage (between 0 and 5 Volts) as the upper limit for the analog input pins.
17	The Arduino R3 adds SDA & SCL pins which are next to the AREF and in addition, there are two pins which are placed near the RESET pin. The first pin is IOREF, it will allow the shields to adapt to the voltage from the board.

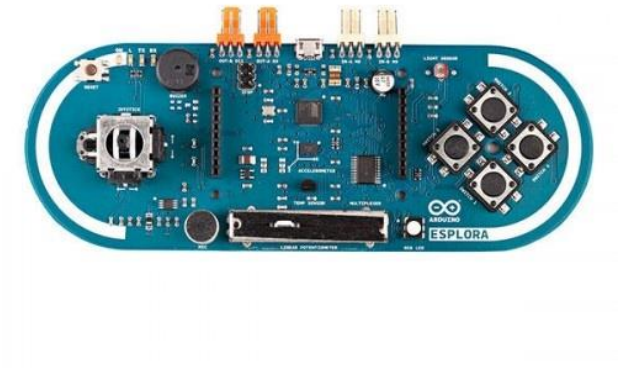
Types of Arduino Boards

There are plenty of Arduino development boards but we would see some of the most popular ones:

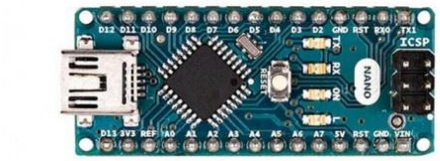
Arduino UNO

Arduino Mega

Arduino Esplora



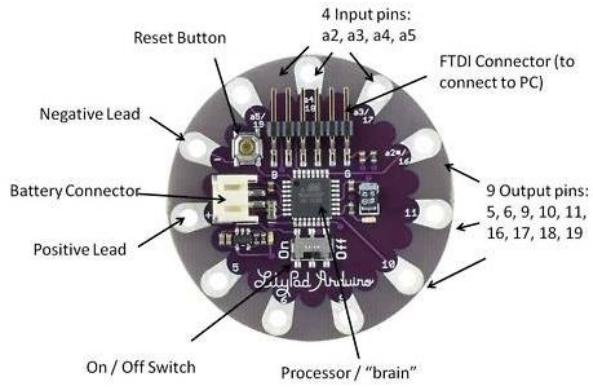
Arduino Nano



Arduino MKR ZERO

Arduino Yún

Lilypad Arduino

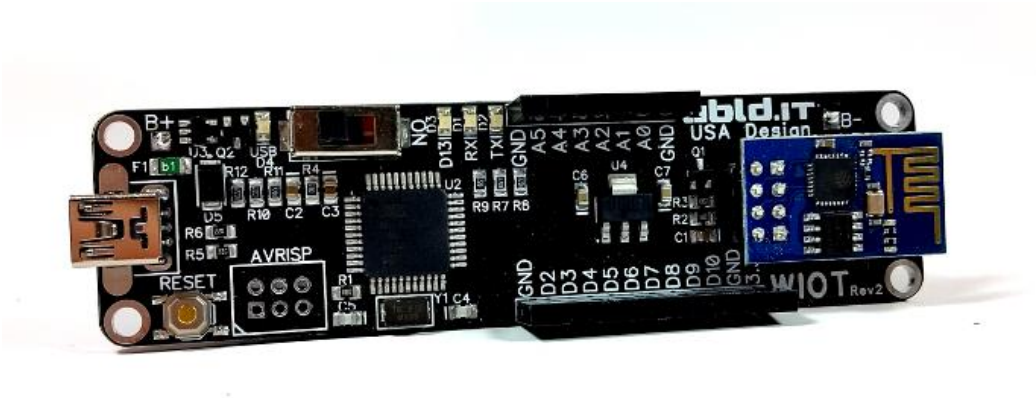


Many other microcontrollers support Arduino software for programming such as:

ESP32

Nodemcu

WIOT Board



Current & Voltage limitations for Arduino UNO

Power

You can power the Arduino board via the USB connector or via the DC power jack. The power jack is 2.1mm center powered.

You can use between 6V and 20V DC to power the board. It is recommended that you should not go below 7V to allow for the voltage drop across the power regulator. If you go too low then the regulator output might drop below 5V and this can cause issues with the boards operation.

It is also recommended that you do not go above 12V. The power regulator may over heat and cause damage to the board.

Power and Aux I/O Port

The pins are used as follows:

- **5V**: This is a regulated output from the on board voltage regulator. This power will come from either the USB or DC input jack. This is fed into the on board 5V voltage regulator. The output from the regulator is connected to this pin. You use this pin to provide 5V to power components connected to the Arduino board. The maximum current draw is approx 400mA on USB and higher if using the DC power jack.

- **3.3V**: This is a regulated output from the on board voltage regulator. The output from the 3.3V regulator is connected to this pin. You use this pin to provide 3.3V to power components connected to the Arduino board. The maximum current draw is 50mA

- You can power the board by connecting a regulated 5V source to the 5V pin or 3.3V to the 3.3V Pin. The power will go directly into the ATMega328 micro controller. The on board power regulators are bypassed. If something goes wrong here then you could very easily damage the ATMega328 chip. Arduino advice against powering the board this way.

- **GND**: Board ground as fed from the ground pins on the DC input jack and the USB connector. Use this ground for components connected to the Arduino board.

- **VIN**: This pin is connected to the input side of the on board voltage regulators. Whatever input DC is supplied to the board by the DC input jack will also appear on the VIN pin. You could also connect power to the board using this pin instead of the USB or DC input jack. Because it is connected to the input side of the voltage regulators the regulated 5V and 3.3V Dc will be supplied to the board.

Input and Outputs

There are 16 digital pins on the Arduino board. They can be used as inputs or outputs. They operate at **5V** and have a maximum current draw of **40mA**. They have an internal pull up resistor that is disabled by default. The pullup resistors are betwen 2 - 50kOhms and can be enabled via software.

We can control the digital I/O pins using the pinmode(), digitalWrite() and digitalRead functions.

Some of the digital I/O pins have additional functions:

- **Serial**: Pin 0 (RX) and 1 (TX). These pins are used to transmit and receive serial TTL (5V) data. These pins are also connected to the Atmega16u2 USB to Serial TTL chip on the Arduino board.
- **PWM**: Pins 3, 5, 6, 9, 10 and 11. The pins can provide a PWM (Pulse Width Modulated) 8 bit output. We use the analogWrite() function with a value between and 0 and 255 to control the duty cycle of the output.
- **LED**: There is a LED connected to Pin 13. When the output on pin 13 is high the LED will be turned on. The LED will be turned off when the output is low.

The Arduino Uno has 6 analog inputs that are labeled A0 through to A5. Each of these Analog pins have 10 bits of resolution which translates from 0 to 1024 different values. By default they measure from ground to 5 volts.

Using Analog Pins as Digital Pins

We can configure the Analog I/O pins to function the same as Digital pins. The Analog to Digital pin mappings are as follow:

A0 => Digital Pin 14

A1 => Digital Pin 15

A2 => Digital Pin 16

A3 => Digital Pin 17

A4 => Digital Pin 18

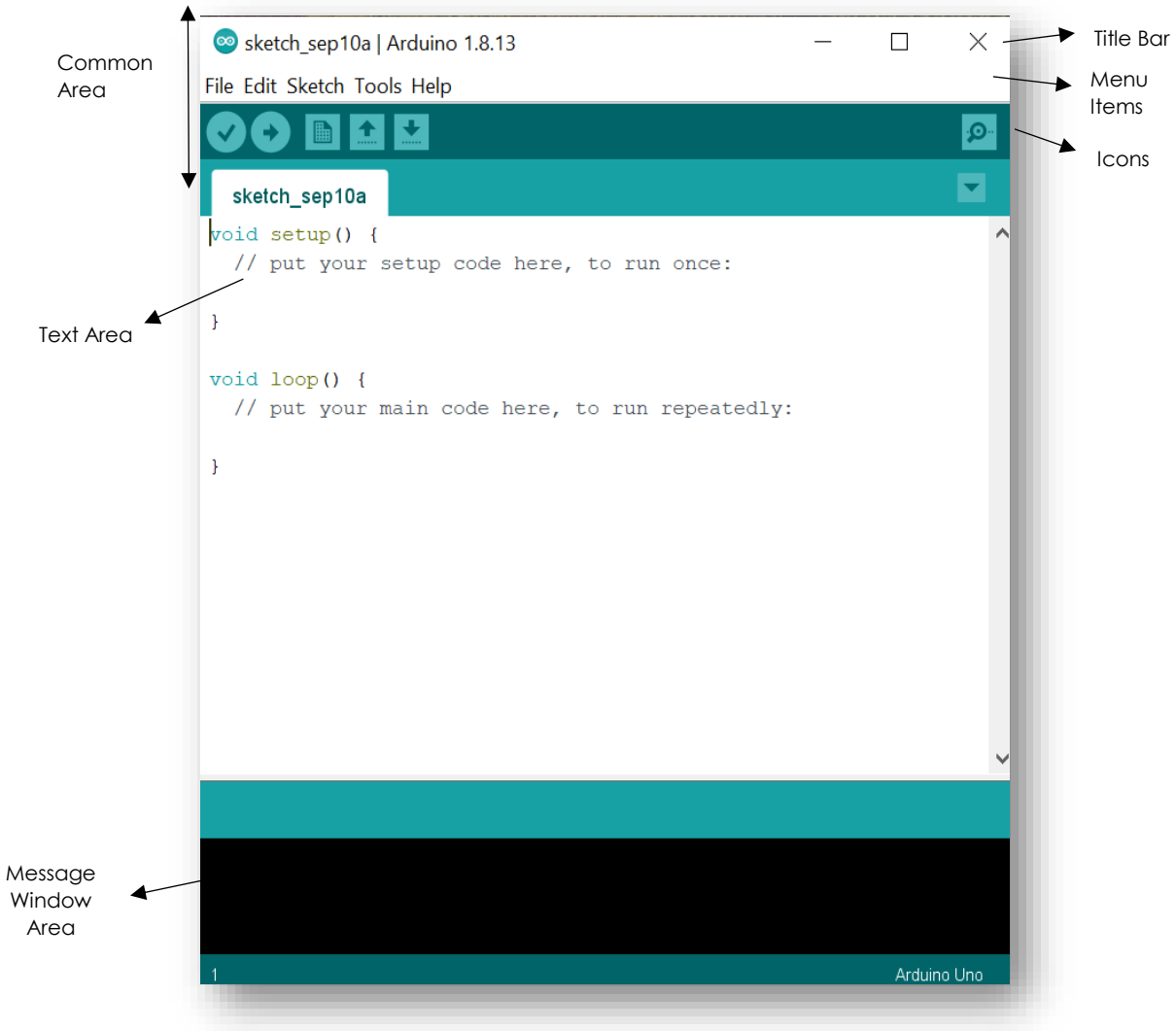
A5 => Digital Pin 19

We can now use the pinmode command to define the pin as an INPUT or OUTPUT. So for pin AO we would use 14 as the pin value. To write to the pin we would use digitalWrite with the appropriate digital pin value as shown in the list above.

Taking a Look around the IDE

Since we already downloaded Arduino IDE, open it and a box will appear like shown in figure below.

The Arduino IDE resembles a simple word processor. The IDE is divided into three main areas: the command area, the text area, and the message window area.



The Command Area

The command area is shown at the top of Figure above and includes the title bar, menu items, and icons. The title bar displays the sketch's filename (sketch_sep10a), as well as the version of the IDE (Arduino 1.8.13). Below this is a series of menu items (File, Edit, Sketch, Tools, and Help) and icons, as described next.

Menu Items

As with any word processor or text editor, you can click one of the menu items to display its various options.

File Contains options to save, load, and print sketches; a thorough set of example sketches to open; as well as the Preferences submenu

Edit Contains the usual copy, paste, and search functions common to any word processor

Sketch Contains the function to verify your sketch before uploading to a board, and some sketch folder and import options

Tools Contains a variety of functions as well as the commands to select the Arduino board type and USB port

Help Contains links to various topics of interest and the version of the IDE

The Icons

Below the menu toolbar are six icons. Mouse over each icon to display its name. The icons, from left to right, are as follows:

Verify Click this to check that the Arduino sketch is valid and doesn't contain any programming mistakes.

Upload Click this to verify and then upload your sketch to the Arduino board.

New Click this to open a new blank sketch in a new window.

Open Click this to open a saved sketch.

Save Click this to save the open sketch. If the sketch doesn't have a name, you will be prompted to create one.

Serial Monitor Click this to open a new window for use in sending and receiving data between your Arduino and the IDE.

The Text Area

The text area is shown in the middle of Figure above; this is where you'll create your sketches. The name of the current sketch is displayed in the tab at the upper left of the text area. (The default

name is the current date.) You'll enter the contents of your sketch here as you would in any text editor.

The Message Window Area

The message window area is shown at the bottom of Figure above. Messages from the IDE appear in the black area. The messages you see will vary and will include messages about verifying sketches, status updates, and so on.

At the bottom right of the message area, you should see the name of your Arduino board type as well as its connected USB port—Arduino Uno in this case, port is not showing because I haven't connected Arduino yet.

Creating Your First Sketch in the IDE

An Arduino sketch is a set of instructions that you create to accomplish a particular task; in other words, a sketch is a program. In this section you'll create and upload a simple sketch that will cause the Arduino's LED (shown in Figure above) to blink repeatedly, by turning it on and then off for 1 second intervals.

Note: Don't worry too much about the specific commands in the sketch we're creating here. The goal is to show you how easy it is to get the Arduino to do something.

To begin, connect your Arduino to the computer with the USB cable. Then open the IDE, choose **Tools** >> **Serial Port** & **Tools** >> **Board** >> **Select your board**, and make sure the USB port is selected. This ensures that the Arduino board is properly connected.

Comments

First, enter a comment as a reminder of what your sketch will be used for. A comment is a note of any length in a sketch, written for the user's benefit. Comments in sketches are useful for adding notes to yourself or others, for entering instructions, or for noting miscellaneous details. When programming your Arduino (creating sketches), it's a good idea to add comments regarding your intentions; these comments can prove useful later when you're revisiting a sketch. To add a comment on a single line, enter two forward slashes and then the comment, like this:

```
// Blink LED sketch by Electrovert Labs.
```

Comments here works same as in C programming we saw in previous course.

The two forward slashes tell the IDE to ignore the text that follows when verifying a sketch. (As mentioned earlier, when you verify a sketch, you're asking the IDE to check that everything is written properly with no errors or simply compiling your code.) To enter a comment that spans two or more lines, enter the characters /* on a line before the comment, and then end the comment with the characters */ on the following line, like this (Same as C programming):

```
/*
```

Author: printf("Electrovert Labs");

Arduino Blink LED Sketch

by Electrovert Labs

*/

As with the two forward slashes that precede a single line comment, the /* and */ tell the IDE to ignore the text that they bracket. Enter a comment describing your Arduino sketch using one of these methods, and then save your sketch by choosing **File** >> **Save As**. Enter a short name for your sketch (such as Blinking), and then click **OK**. The default filename extension for Arduino sketches is **.ino**, and the IDE should add this automatically. The name for your sketch should be, in this case, **Blinking.ino**, and you should be able to see it in your Sketchbook.

The Setup Function

The next stage in creating any sketch is to add the **void setup()** function (void is return type here, this means that setup function is not going to return anything, refer to C programming course for knowing functions in detail). This function contains a set of instructions for the Arduino to execute once only, each time it is reset or turned on. To create the setup function, add the following lines to your sketch, after the comments:

```
void setup()
{
}
```

Controlling the Hardware

Our program will blink the user LED on the Arduino (L written beside it). The user LED is connected to the Arduino's digital pin 13. A digital pin can either detect an electrical signal or generate one on command. In this project, we'll generate an electrical signal that will light the LED. This may seem a little complicated, but you'll learn more about digital pins in future chapters. For now, just continue with creating the sketch. Enter the following into your sketch between the braces in setup function.

```
pinMode(13, OUTPUT); // set digital pin 13 to output
```

The number 13 in the listing represents the digital pin you're addressing. You're setting this pin to OUTPUT, which means it will generate (output) an electrical signal. If you wanted it to detect an incoming electrical signal, then you would use INPUT instead. Notice that the function pinMode() ends with a semicolon (;). Every function in your Arduino sketches will end with a semicolon (same as C programming). Save your sketch again to make sure that you don't lose any of your work.

The Loop Function

Remember that our goal is to make the LED blink repeatedly. To do this, we'll create a loop function to tell the Arduino to execute an instruction over and over until the power is shut off or someone presses the RESET button. Enter the code shown in boldface after the void **setup()** section in the following listing to create an empty loop function. Be sure to end this new section with another brace (}), and then save your sketch again.

```
/*
Arduino Blink LED Sketch
by Electrovert Labs */

void setup()
{
pinMode(13, OUTPUT); // set digital pin 13 to output
}
void loop()
{
// place your main loop code here:
}
```

Warning *The Arduino IDE does not automatically save sketches, so save your work frequently!*

Next, enter the actual functions into void **loop()** for the Arduino to execute. Enter the following between the loop function's braces, and then click Verify to make sure that you've entered everything correctly:

```
digitalWrite(13, HIGH); // turn on digital pin 13
delay(1000); // pause for one second
digitalWrite(13, LOW); // turn off digital pin 13
delay(1000); // pause for one second
```

Let's take this all apart. The **digitalWrite()** function controls the voltage that is output from a digital pin: in this case, pin 13 to the LED.

By setting the second parameter of this function to HIGH, a "high" digital voltage is output; then current will flow from the pin and the LED will turn on. (If you were to set this parameter to LOW, then the current flowing through the LED would stop.) With the LED turned on, the light pauses for 1 second with **delay**(1000). The **delay**() function causes the sketch to do nothing for a period of time— in this case, 1,000 milliseconds, or 1 second.

Next, we turn off the voltage to the LED with digitalWrite(13, LOW);. Finally, we pause again for 1 second while the LED is off, with **delay**(1000);.

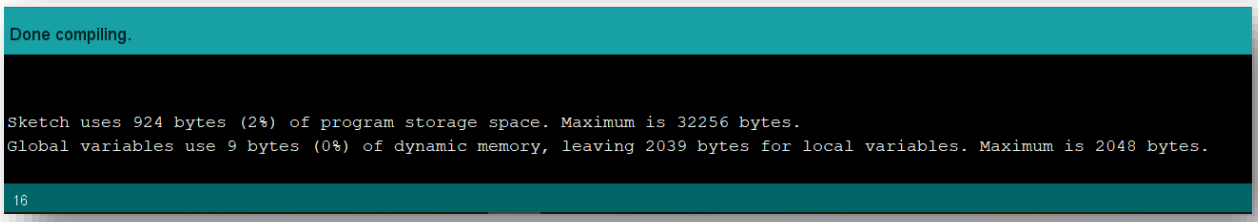
The completed sketch should look like this:

```
/*
Arduino Blink LED Sketch
by Electrovert Labs
*/
void setup()
{
pinMode(13, OUTPUT); // set digital pin 13 to output
}
void loop()
{
digitalWrite(13, HIGH); // turn on digital pin 13
delay(1000); // pause for one second
digitalWrite(13, LOW); // turn off digital pin 13
delay(1000); // pause for one second
}
```

Before you do anything further, save your sketch!

Verifying Your Sketch

When you verify your sketch, you ensure that it has been written correctly in a way that the Arduino can understand. To verify your complete sketch, click Verify in the IDE and wait a moment. Once the sketch has been verified, a note should appear in the message window, as shown in Figure below.



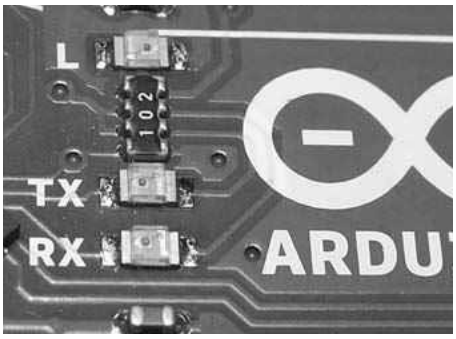
This "Done compiling" message tells you that the sketch is okay to upload to your Arduino. It also shows how much memory it will use (1,076 bytes in this case) of the total available on the Arduino (32,256 bytes). But what if your sketch isn't okay? Say, for example, you forgot to add a semicolon at the end of the second delay(1000) function. If something is broken in your sketch, then when you click Verify, the message window should display a verification error message similar to the one shown in Figure below.

```
delay(1000); // pause for one second
digitalWrite(13, LOW); // turn off digital pin 13
delay(1000) // pause for one second
}

expected ';' before '}' token

                                    Arduino\sketch_sep10a\sketch_sep10a.ino: In function 'void loop()':
sketch_sep10a:26:1: error: expected ';' before '}' token
 }
 ^
exit status 1
expected ';' before '}' token

26
```

The message tells you that the error occurs in the void loop function, lists the line number of the sketch where the IDE thinks the error is located (26, or line 26 of your current sketch), and displays the error itself (the missing semicolon, error: expected ';' before '}' token). Furthermore, the IDE should also highlight in yellow the location of the error or a spot just after it. This helps you easily locate and rectify the mistake.

Uploading and Running Your Sketch

Once you're satisfied that your sketch has been entered correctly, save it, ensure that your Arduino board is connected, and click Upload in the IDE. The IDE may verify your sketch again and then upload it to your Arduino board. During this process, the TX/RX LEDs on your board (shown in Figure below) should blink, indicating that information is traveling between the Arduino and your computer. Now for the moment of truth: Your Arduino should start running the sketch. If you've done everything correctly, then the LED (L) should blink on and off once every second! Congratulations. You now know the basics of how to enter, verify, and upload an Arduino sketch.



Modifying Your Sketch

After running your sketch, you may want to change how it operates, by, for example, adjusting the on or off delay time for the LED. Because the IDE is a lot like a word processor, you can open your saved sketch, adjust the values, and then save your sketch again and upload it to the Arduino. For example, to increase the rate of blinking, change both delay functions to make the LEDs blink for one-quarter of a second by adjusting the delay to 250 like this:

```
delay(250); // pause for one-quarter of one second
```

Then upload the sketch again. The LED should now blink faster, for one-quarter of a second each time.

Arduino Programming

Have a look at different topics covered in Arduino Programming topic. Come back to understand different topics when used in projects.

Structure

The basic structure of the Arduino programming language is fairly simple and runs in at least two parts. These two required parts, or functions, enclose blocks of statements.

```
void setup()
{
statements;
}
void loop()
{
statements;
}
```

Where **setup()** is the preparation, **loop()** is the execution. Both functions are required for the program to work.

The setup function should follow the declaration of any variables at the very beginning of the program. It is the first function to run in the program, is run only once, and is used to set pinMode or initialize serial communication.

The loop function follows next and includes the code to be executed continuously – reading inputs, triggering outputs, etc. This function is the core of all Arduino programs and does the bulk of the work.

1. setup()

The setup() function is called once when your program starts. Use it to initialize pin modes, or begin serial. It must be included in a program even if there are no statements to run.

```
void setup()
{
pinMode(pin, OUTPUT); // sets the 'pin' as output
}
```

2. loop()

After calling the setup() function, the loop() function does precisely what its name suggests, and loops consecutively, allowing the program to change, respond, and control the Arduino board.

```
void loop()
{
digitalWrite(pin, HIGH); // turns 'pin' on
```

```
delay(1000); // pauses for one second
digitalWrite(pin, LOW); // turns 'pin' off
delay(1000); // pauses for one second
}
```

3. Functions

A function is a block of code that has a name and a block of statements that are executed when the function is called. The functions void setup() and void loop() have already been discussed and other built-in functions will be discussed later.

Custom functions can be written to perform repetitive tasks and reduce clutter in a program. Functions are declared by first declaring the function type. This is the type of value to be returned by the function such as 'int' for an integer type function. If no value is to be returned the function type would be void. After type, declare the name given to the function and in parenthesis any parameters being passed to the function.

type functionName(parameters)

{

statements;

}

The following integer type function delayVal() is used to set a delay value in a program by reading the value of a potentiometer. It first declares a local variable v, sets v to the value of the potentiometer which gives a number between 0-1023, then divides that value by 4 for a final value between 0-255, and finally returns that value back to the main program.

```
int delayVal()
{
int v; // create temporary variable 'v'
v = analogRead(pot); // read potentiometer value
v /= 4; // converts 0-1023 to 0-255
return v; // return final value
}
```

4. {} curly braces

Curly braces (also referred to as just "braces" or "curly brackets") define the beginning and end of function blocks and statement blocks such as the void loop() function and the for and if statements.

type function()

{

statements;

}

An opening curly brace { must always be followed by a closing curly brace }. This is often referred to as the braces being balanced. Unbalanced braces can often lead to cryptic, impenetrable compiler errors that can sometimes be hard to track down in a large program.

The Arduino environment includes a convenient feature to check the balance of curly braces. Just select a brace, or even click the insertion point immediately following a brace, and its logical companion will be highlighted.

5. ; semicolon

A semicolon must be used to end a statement and separate elements of the program. A semicolon is also used to separate elements in a for loop.

int x = 13; // declares variable 'x' as the integer 13

Note: Forgetting to end a line in a semicolon will result in a compiler error. The error text may be obvious, and refer to a missing semicolon, or it may not. If an impenetrable or seemingly illogical compiler error comes up, one of the first things to check is a missing semicolon, near the line where the compiler complained.

6. /*... */ block comments

Block comments, or multi-line comments, are areas of text ignored by the program and are used for large text descriptions of code or comments that help others understand parts of the program. They begin with /* and end with */ and can span multiple lines.

/* this is an enclosed block comment

don't forget the closing comment -

they have to be balanced!

*/

Because comments are ignored by the program and take no memory space they should be used generously and can also be used to "comment out" blocks of code for debugging purposes.

Note: While it is possible to enclose single line comments within a block comment, enclosing a second block comment is not allowed.

7. // line comments

Single line comments begin with // and end with the next line of code. Like block comments, they are ignored by the program and take no memory space.

// this is a single line comment

Single line comments are often used after a valid statement to provide more information about what the statement accomplishes or to provide a future reminder.

variables

1. variables

A variable is a way of naming and storing a numerical value for later use by the program. As their namesake suggests, variables are numbers that can be continually changed as opposed to constants whose value never changes. A variable needs to be declared and optionally assigned to the value needing to be stored. The following code declares a variable called inputVariable and then assigns it the value obtained on analog input pin 2:

int inputVariable = 0; // declares a variable and

// assigns value of 0

inputVariable = analogRead(2); // set variable to value of

// analog pin 2

'inputVariable' is the variable itself. The first line declares that it will contain an int, short for integer. The second line sets the variable to the value at analog pin 2. This makes the value of pin 2 accessible elsewhere in the code.

Once a variable has been assigned, or re-assigned, you can test its value to see if it meets certain conditions, or you can use its value directly. As an example to illustrate three useful operations with variables, the following code tests whether the inputVariable is less than 100, if true it assigns the value 100 to inputVariable, and then sets a delay based on inputVariable which is now a minimum of 100:

```
if (inputVariable < 100) // tests variable if less than 100
{
inputVariable = 100; // if true assigns value of 100
}
delay(inputVariable); // uses variable as delay
```

Note: Variables should be given descriptive names, to make the code more readable. Variable names like tiltSensor or pushButton help the programmer and anyone else reading the code to understand what the variable represents. Variable names like var or value, on the other hand, do little to make the code readable and are only used here as examples. A variable can be named any word that is not already one of the keywords in the Arduino language.

2. variable declaration

All variables have to be declared before they can be used. Declaring a variable means defining its value type, as in int, long, float, etc., setting a specified name, and optionally assigning an initial value. This only needs to be done once in a program but the value can be changed at any time using arithmetic and various assignments.

The following example declares that inputVariable is an int, or integer type, and that its initial value equals zero. This is called a simple assignment.

int inputVariable = 0;

A variable can be declared in a number of locations throughout the program and where this definition takes place determines what parts of the program can use the variable.

3. variable scope

A variable can be declared at the beginning of the program before void setup(), locally inside of functions, and sometimes within a statement block such as for loops. Where the variable is declared determines the variable scope, or the ability of certain parts of a program to make use of the variable.

A global variable is one that can be seen and used by every function and statement in a program. This variable is declared at the beginning of the program, before the setup() function.

A local variable is one that is defined inside a function or as part of a for loop. It is only visible and can only be used inside the function in which it was declared. It is therefore possible to have two or more variables of the same name in different parts of the same program that contain different values. Ensuring that only one function has access to its variables simplifies the program and reduces the potential for programming errors.

The following example shows how to declare a few different types of variables and demonstrates each variable's visibility:

```
int value; // 'value' is visible
// to any function
void setup()
{
// no setup needed
}
void loop()
{
for (int i=0; i<20;) // 'i' is only visible
{ // inside the for-loop
i++;
}
float f; // 'f' is only visible
} // inside loop
```

datatypes

1. byte

Byte stores an 8-bit numerical value without decimal points. They have a range of 0-255.

byte someVariable = 180; // declares 'some Variable as a byte type

2. int

store a 16-bit value with a range of 32,767 to -32,768.

int someVariable = 1500; // declares 'someVariable' as an integer type

Note: Integer variables will roll over if forced past their maximum or minimum values by an assignment or comparison. For example, if x = 32767 and a subsequent statement adds 1 to x, x = x + 1 or x++, x will then rollover and equal -32,768.

3. long

Extended size datatype for long integers, without decimal points, stored in a 32-bit value with a range of 2,147,483,647 to -2,147,483,648.

long someVariable = 90000; // declares 'someVariable' as a long type

4. float

A datatype for floating-point numbers, or numbers that have a decimal point. Floating-point numbers have greater resolution than integers and are stored as a 32-bit value with a range of 3.4028235E+38 to -3.4028235E+38.

float someVariable = 3.14; // declares 'someVariable' as a floating-point type

Note: Floating-point numbers are not exact, and may yield strange results when compared. Floating point math is also much slower than integer math in performing calculations, so should be avoided if possible.

5. arrays

An array is a collection of values that are accessed with an index number. Any value in the array may be called upon by calling the name of the array and the index number of the value. Arrays are zero indexed, with the first value in the array beginning at index number 0. An array needs to be declared and optionally assigned values before they can be used.

int myArray[] = {value0, value1, value2...}

Likewise it is possible to declare an array by declaring the array type and size and later assign values to an index position:

int myArray[5]; // declares integer array w/ 6 positions

myArray[3] = 10; // assigns the 4th index the value 10

To retrieve a value from an array, assign a variable to the array and index position:

x = myArray[3]; // x now equals 10

Arrays are often used in for loops, where the increment counter is also used as the index position for each array value. The following example uses an array to flicker an LED. Using a for loop, the counter begins at 0, writes the value contained at index position 0 in the array flicker[], in this case 180, to the PWM pin 10, pauses for 200ms, then moves to the next index position.

```
int ledPin = 10; // LED on pin 10
byte flicker[] = {180, 30, 255, 200, 10, 90, 150, 60};
// above array of 8
void setup() // different values
{
pinMode(ledPin, OUTPUT); // sets OUTPUT pin
}
void loop()
{
for(int i=0; i<7; i++) // loop equals number
{ // of values in array
analogWrite(ledPin, flicker[i]); // write index value
delay(200); // pause 200ms
}
}
```

arithmetic

1. arithmetic

Arithmetic operators include addition, subtraction, multiplication, and division. They return the sum, difference, product, or quotient (respectively) of two operands.

y = y + 3;
x = x - 7;
i = j * 6;
r = r / 5;

The operation is conducted using the data type of the operands, so, for example, 9 / 4 results in 2 instead of 2.25 since 9 and 4 are integers and are incapable of using decimal points. This also means that the operation can overflow if the result is larger than what can be stored in the data type.

If the operands are of different types, the larger type is used for the calculation. For example, if one of the numbers (operands) are of the type float and the other of type integer, floating point math will be used for the calculation.

Choose variable sizes that are large enough to hold the largest results from your calculations. Know at what point your variable will rollover and also what happens in the other direction e.g. (0 - 1) OR (0 - - 32768). For math that requires fractions, use float variables, but be aware of their drawbacks: large size and slow computation speeds.

Note: Use the cast operator e.g. (int)myFloat to convert one variable type to another on the fly. For example, i = (int)3.6 will set i equal to 3.

2. compound assignments

Compound assignments combine an arithmetic operation with a variable assignment. These are commonly found in for loops as described later. The most common compound assignments include:

```
x ++ // same as x = x + 1, or increments x by +1
x -- // same as x = x - 1, or decrements x by -1
x += y // same as x = x + y, or increments x by +y
x -= y // same as x = x - y, or decrements x by -y
x *= y // same as x = x * y, or multiplies x by y
x /= y // same as x = x / y, or divides x by y
```

Note: For example, x *= 3 would triple the old value of x and re-assign the resulting value to x.

3. comparison operators

Comparisons of one variable or constant against another are often used in if statements to test if a specified condition is true. In the examples found on the following pages, ?? is used to indicate any of the following conditions:

```
x == y // x is equal to y
x != y // x is not equal to y
x < y // x is less than y
x > y // x is greater than y
x <= y // x is less than or equal to y
x >= y // x is greater than or equal to y
```

4. logical operators

Logical operators are usually a way to compare two expressions and return a TRUE or FALSE depending on the operator. There are three logical operators, AND, OR, and NOT, that are often used in if statements:

```
Logical AND:
if (x > 0 && x < 5) // true only if both
// expressions are true
Logical OR:
if (x > 0 || y > 0) // true if either
// expression is true
```

```
Logical NOT:
if (!x > 0) // true only if
// expression is false
```

constants

1. constants

The Arduino language has a few predefined values, which are called constants. They are used to make the programs easier to read. Constants are classified in groups.

2. true/false

These are Boolean constants that define logic levels. FALSE is easily defined as 0 (zero) while TRUE is often defined as 1, but can also be anything else except zero. So in a Boolean sense, -1, 2, and -200 are all also defined as TRUE.

```
if (b == TRUE);
{
doSomething;
}
```

3. high/low

These constants define pin levels as HIGH or LOW and are used when reading or writing to digital pins. HIGH is defined as logic level 1, ON, or 5 volts while LOW is logic level 0, OFF, or 0 volts.

digitalWrite(13, HIGH);

4. input/output

Constants used with the pinMode() function to define the mode of a digital pin as either INPUT or OUTPUT.

pinMode(13, OUTPUT);

flow control

1. if

if statements test whether a certain condition has been reached, such as an analog value being above a certain number, and executes any statements inside the brackets if the statement is true. If false the program skips over the statement. The format for an if test is:

if (someVariable ?? value)

{

doSomething;

}

The above example compares someVariable to another value, which can be either a variable or constant. If the comparison, or condition in parentheses is true, the statements inside the brackets are run. If not, the program skips over them and continues on after the brackets.

Note: Beware of accidentally using '=', as in if(x=10), while technically valid, defines the variable x to the value of 10 and is as a result always true. Instead use '==', as in if(x==10), which only tests whether x happens to equal the value 10 or not. Think of '=' as "equals" opposed to '==' being "is equal to".

2. if… else

if… else allows for 'either-or' decisions to be made. For example, if you wanted to test a digital input, and do one thing if the input went HIGH or instead do another thing if the input was LOW, you would write that this way:

if (inputPin == HIGH)

{

doThingA;

}

else

{

doThingB;

}

else can also precede another if test, so that multiple, mutually exclusive tests can be run at the same time. It is even possible to have an unlimited number of these else branches. Remember though, only one set of statements will be run depending on the condition tests:

```
if (inputPin < 500)
{
```

```
doThingA;
}
else if (inputPin >= 1000)
{
doThingB;
}
else
{
doThingC;
}
```

Note: An if statement simply tests whether the condition inside the parenthesis is true or false. This statement can be any valid C statement as in the first example, if (inputPin == HIGH). In this example, the if statement only checks to see if indeed the specified input is at logic level high, or +5v.

3. for

The for statement is used to repeat a block of statements enclosed in curly braces a specified number of times. An increment counter is often used to increment and terminate the loop. There are three parts, separated by semicolons (;), to the for loop header:

for (initialization; condition; expression)

{

doSomething;

}

The initialization of a local variable, or increment counter, happens first and only once. Each time through the loop, the following condition is tested. If the condition remains true, the following statements and expression are executed and the condition is tested again. When the condition becomes false, the loop ends.

The following example starts the integer i at 0, tests to see if i is still less than 20 and if true, increments i by 1 and executes the enclosed statements:

```
for (int i=0; i<20; i++) // declares i, tests if less
{ // than 20, increments i by 1
digitalWrite(13, HIGH); // turns pin 13 on
delay(250); // pauses for 1/4 second
digitalWrite(13, LOW); // turns pin 13 off
delay(250); // pauses for 1/4 second
}
```

Note: The C for loop is much more flexible than for loops found in some other computer languages, including BASIC. Any or all of the three header elements may be omitted, although the semicolons are required. Also the statements for initialization, condition, and expression can be any valid C statements with unrelated variables. These types of unusual for statements may provide solutions to some rare programming problems.

4. while

while loops will loop continuously, and infinitely, until the expression inside the parenthesis becomes false. Something must change the tested variable, or the while loop will never exit. This could be in your code, such as an incremented variable, or an external condition, such as testing a sensor.

while (someVariable ?? value)

{

doSomething;

}

The following example tests whether 'someVariable' is less than 200 and if true executes the statements inside the brackets and will continue looping until 'someVariable' is no longer less than 200.

```
while (someVariable < 200) // tests if less than 200
{
doSomething; // executes enclosed statements
someVariable++; // increments variable by 1
}
```

5. do... while

The do loop is a bottom driven loop that works in the same manner as the while loop, with the exception that the condition is tested at the end of the loop, so the do loop will always run at least once.

do

{

doSomething;

} while (someVariable ?? value);

The following example assigns readSensors() to the variable 'x', pauses for 50 milliseconds, then loops indefinitely until 'x' is no longer less than 100:

```
do
{
x = readSensors(); // assigns the value of
// readSensors() to x
delay (50); // pauses 50 milliseconds
} while (x < 100); // loops if x is less than 100
```

digital i/o

1. pinMode(pin, mode)

Used in void setup() to configure a specified pin to behave either as an INPUT or an OUTPUT.

pinMode(pin, OUTPUT); // sets 'pin' to output

Arduino digital pins default to inputs, so they don't need to be explicitly declared as inputs with pinMode(). Pins configured as INPUT are said to be in a high-impedance state.

There are also convenient 20KΩ pullup resistors built into the Atmega chip that can be accessed from software. These built-in pullup resistors are accessed in the following manner:

pinMode(pin, INPUT); // set 'pin' to input

digitalWrite(pin, HIGH); // turn on pullup resistors

Pullup resistors would normally be used for connecting inputs like switches. Notice in the above example it does not convert pin to an output, it is merely a method for activating the internal pull-ups.

Pins configured as OUTPUT are said to be in a low-impedance state and can provide 40 mA (milliamps) of current to other devices/circuits. This is enough current to brightly light up an LED (don't forget the series resistor), but not enough current to run most relays, solenoids, or motors.

Short circuits on Arduino pins and excessive current can damage or destroy the output pin, or damage the entire Atmega chip. It is often a good idea to connect an OUTPUT pin to an external device in series with a 470Ω or 1KΩ resistor.

2. digitalRead(pin)

Reads the value from a specified digital pin with the result either HIGH or LOW. The pin can be specified as either a variable or constant (0-13).

value = digitalRead(Pin); // sets 'value' equal to the input pin

3. digitalWrite(pin, value)

Outputs either logic level HIGH or LOW at (turns on or off) a specified digital pin. The pin can be specified as either a variable or constant (0-13).

digitalWrite(pin, HIGH); // sets 'pin' to high

The following example reads a pushbutton connected to a digital input and turns on an LED connected to a digital output when the button has been pressed:

```
int led = 13; // connect LED to pin 13
int pin = 7; // connect pushbutton to pin 7
int value = 0; // variable to store the read value
```

```
void setup()
{
pinMode(led, OUTPUT); // sets pin 13 as output
pinMode(pin, INPUT); // sets pin 7 as input
}
void loop()
{
value = digitalRead(pin); // sets 'value' equal to
// the input pin
digitalWrite(led, value); // sets 'led' to the
} // button's value
```

analog i/o

1. analogRead(pin)

Reads the value from a specified analog pin with a 10-bit resolution. This function only works on the analog in pins (0-5). The resulting integer values range from 0 to 1023.

value = analogRead(pin); // sets 'value' equal to 'pin'

Note: Analog pins unlike digital ones, do not need to be first declared as INPUT nor OUTPUT.

2. analogWrite(pin, value)

Writes a pseudo-analog value using hardware enabled pulse width modulation (PWM) to an output pin marked PWM. On newer Arduinos with the ATmega168 chip, this function works on pins 3, 5, 6, 9, 10, and 11. Older Arduinos with an ATmega8 only support pins 9, 10, and 11. The value can be specified as a variable or constant with a value from 0-255.

analogWrite(pin, value); // writes 'value' to analog 'pin'

A value of 0 generates a steady 0 volts output at the specified pin; a value of 255 generates a steady 5 volts output at the specified pin. For values in between 0 and 255, the pin rapidly alternates between 0 and 5 volts - the higher the value, the more often the pin is HIGH (5 volts). For example, a value of 64 will be 0 volts three-quarters of the time, and 5 volts one quarter of the time; a value of 128 will be at 0 half the time and 255 half the time; and a value of 192 will be 0 volts one quarter of the time and 5 volts three-quarters of the time.

Because this is a hardware function, the pin will generate a steady wave after a call to analogWrite in the background until the next call to analogWrite (or a call to digitalRead or digitalWrite on the same pin).

Note: Analog pins unlike digital ones, do not need to be first declared as INPUT nor OUTPUT.

The following example reads an analog value from an analog input pin, converts the value by dividing by 4, and outputs a PWM signal on a PWM pin:

```
int led = 10; // LED with 220 resistor on pin 10
int pin = 0; // potentiometer on analog pin 0
int value; // value for reading
void setup(){} // no setup needed
void loop()
{
value = analogRead(pin); // sets 'value' equal to 'pin'
value /= 4; // converts 0-1023 to 0-255
analogWrite(led, value); // outputs PWM signal to led
}
```

time

1. delay(ms)

Pauses a program for the amount of time as specified in milliseconds, where 1000 equals 1 second.

delay(1000); // waits for one second

2. millis()

Returns the number of milliseconds since the Arduino board began running the current program as an unsigned long value.

value = millis(); // sets 'value' equal to millis()

Note: This number will overflow (reset back to zero), after approximately 9 hours.

math

1. min(x, y)

Calculates the minimum of two numbers of any data type and returns the smaller number.

value = min(value, 100); // sets 'value' to the smaller of

// 'value' or 100, ensuring that it never gets above 100.

2. max(x, y)

Calculates the maximum of two numbers of any data type and returns the larger number.

value = max(value, 100); // sets 'value' to the larger of

// 'value' or 100, ensuring that it is at least 100.

random

1. randomSeed(seed)

Sets a value, or seed, as the starting point for the random() function.

randomSeed(value); // sets 'value' as the random seed

Because the Arduino is unable to create a truly random number, randomSeed allows you to place a variable, constant, or other function into the random function, which helps to generate more random "random" numbers. There are a variety of different seeds, or functions, that can be used in this function including millis() or even analogRead() to read electrical noise through an analog pin.

2. random(max), random(min, max)

The random function allows you to return pseudo-random numbers within a range specified by min and max values.

value = random(100, 200); // sets 'value' to a random

// number between 100-200

Note: Use this after using the randomSeed() function.

The following example creates a random value between 0-255 and outputs a PWM signal on a PWM pin equal to the random value:

```
int randNumber; // variable to store the random value
int led = 10; // LED with 220 resistor on pin 10
void setup() {} // no setup needed
void loop()
{
randomSeed(millis()); // sets millis() as seed
randNumber = random(255); // random number from 0-255
analogWrite(led, randNumber); // outputs PWM signal
delay(500); // pauses for half a second
}
```

serial

1. Serial.begin(rate)

Opens serial port and sets the baud rate for serial data transmission. The typical baud rate for communicating with the computer is 9600 although other speeds are supported.

```
void setup()
```

```
{
Serial.begin(9600); // opens serial port
} // sets data rate to 9600 bps
```

Note: When using serial communication, digital pins 0 (RX) and 1 (TX) cannot be used at the same time.

2. Serial.println(data)

This command takes the same form as Serial.print(), but is easier for reading data on the Serial Monitor.

Serial.println(analogValue); // sends the value of 'analogValue'

Note: For more information on the various permutations of the Serial.println() and Serial.print() functions please refer to the Arduino website.

The following simple example takes a reading from analog pin0 and sends this data to the computer every 1 second.

```
void setup()
{
Serial.begin(9600); // sets serial to 9600bps
}
void loop()
{
Serial.println(analogRead(0)); // sends analog value
delay(1000); // pauses for 1 second
}
```

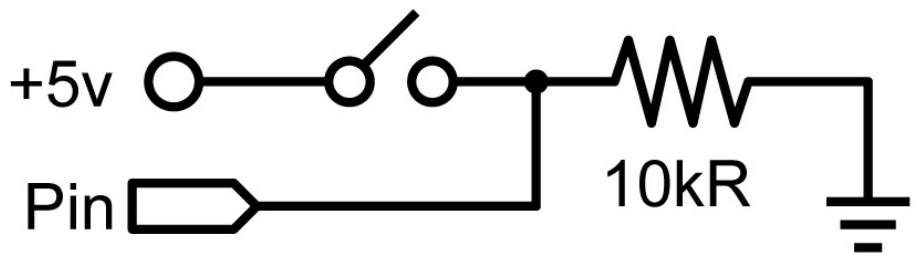
Additional Content

digital output

This is the basic 'hello world' program used to simply turn something on or off. In this example, an LED is connected to pin13, and is blinked every second. The resistor may be omitted on this pin since the Arduino has one built in.

```
int ledPin = 13; // LED on digital pin 13
void setup() // run once
{
pinMode(ledPin, OUTPUT); // sets pin 13 as output
}
void loop() // run over and over again
{
digitalWrite(ledPin, HIGH); // turns the LED on
delay(1000); // pauses for 1 second
digitalWrite(ledPin, LOW); // turns the LED off
delay(1000); // pauses for 1 second
}
```

digital input

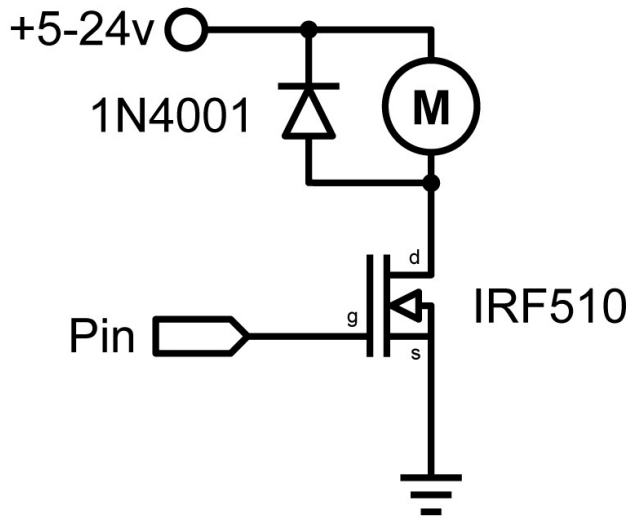


This is the simplest form of input with only two possible states: on or off. This example reads a simple switch or pushbutton connected to pin2. When the switch is closed the input pin will read HIGH and turn on an LED.

```
int ledPin = 13; // output pin for the LED
int inPin = 2; // input pin (for a switch)
void setup()
{
pinMode(ledPin, OUTPUT); // declare LED as output
pinMode(inPin, INPUT); // declare switch as input
}
void loop()
{
if (digitalRead(inPin) == HIGH) // check if input is HIGH
{
digitalWrite(ledPin, HIGH); // turns the LED on
delay(1000); // pause for 1 second
```

```
digitalWrite(ledPin, LOW); // turns the LED off
delay(1000); // pause for 1 second
}
}
```

high current output



Sometimes it is necessary to control more than 40ma from the Arduino. In this case a MOSFET or transistor could be used to switch higher current loads. The following example quickly turns on and off the MOSFET 5 times every second.

Note: The schematic shows a motor and protection diode but other non-inductive loads could be used without the diode.

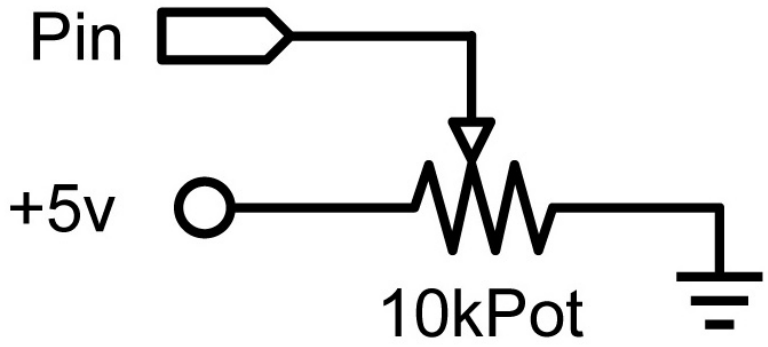
```
int outPin = 5; // output pin for the MOSFET
void setup()
{
pinMode(outPin, OUTPUT); // sets pin5 as output
}
void loop()
{
for (int i=0; i<=5; i++) // loops 5 times
{
digitalWrite(outPin, HIGH); // turns MOSFET on
delay(250); // pauses 1/4 second
digitalWrite(outPin, LOW); // turns MOSFET off
delay(250); // pauses 1/4 second
}
delay(1000); // pauses 1 second
}
```

pwm output

Pulsewidth Modulation (PWM) is a way to fake an analog output by pulsing the output. This could be used to dim and brighten an LED or later to control a servo motor. The following example slowly brightens and dims an LED using for loops.

```
int ledPin = 9; // PWM pin for the LED
void setup(){} // no setup needed
void loop()
{
for (int i=0; i<=255; i++) // ascending value for i
{
analogWrite(ledPin, i); // sets brightess level to i
delay(100); // pauses for 100ms
}
for (int i=255; i>=0; i--) // descending value for i
{
analogWrite(ledPin, i); // sets brightess level to i
delay(100); // pauses for 100ms
}
}
```

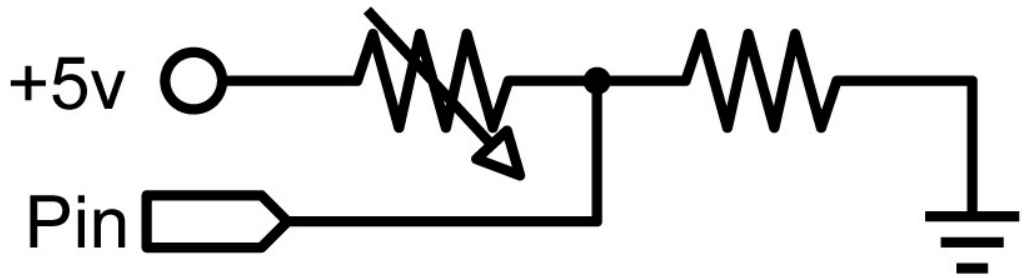
potentiometer input



Using a potentiometer and one of the Arduino's analog-to-digital conversion (ADC) pins it is possible to read analog values from 0-1024. The following example uses a potentiometer to control an LED's rate of blinking.

```
int potPin = 0; // input pin for the potentiometer
int ledPin = 13; // output pin for the LED
void setup()
{
pinMode(ledPin, OUTPUT); // declare ledPin as OUTPUT
}
void loop()
{
digitalWrite(ledPin, HIGH); // turns ledPin on
delay(analogRead(potPin)); // pause program
digitalWrite(ledPin, LOW); // turns ledPin off
delay(analogRead(potPin)); // pause program
}
```

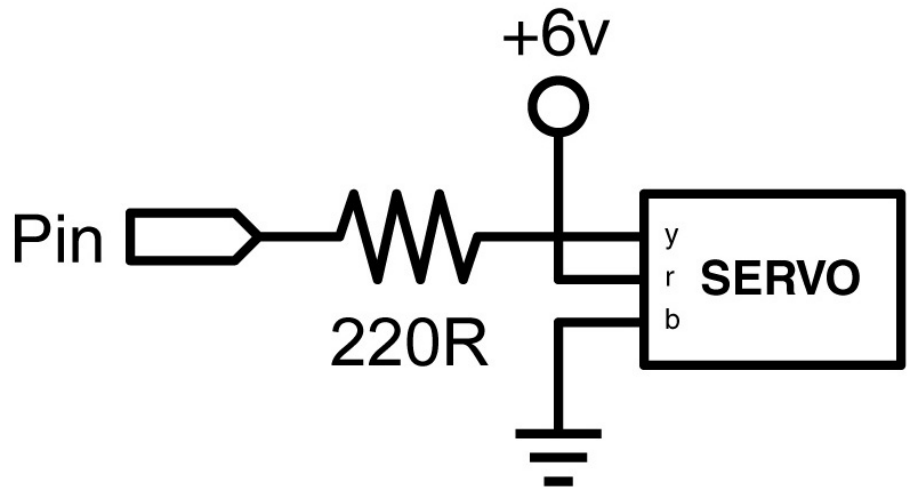
variable resistor input



Variable resistors include CdS light sensors, thermistors, flex sensors, and so on. This example makes use of a function to read the analog value and set a delay time. This controls the speed at which an LED brightens and dims.

```
int ledPin = 9; // PWM pin for the LED
int analogPin = 0; // variable resistor on analog pin 0
void setup(){} // no setup needed
void loop()
{
for (int i=0; i<=255; i++) // ascending value for i
{
analogWrite(ledPin, i); // sets brightess level to i
delay(delayVal()); // gets time value and pauses
}
for (int i=255; i>=0; i--) // descending value for i
{
analogWrite(ledPin, i); // sets brightess level to i
delay(delayVal()); // gets time value and pauses
}
}
int delayVal()
{
int v; // create temporary variable
v = analogRead(analogPin); // read analog value
v /= 8; // convert 0-1024 to 0-128
return v; // returns final value
}
```

servo output



Hobby servos are a type of self-contained motor that can move in a 180° arc. All that is needed is a pulse sent every 20ms. This example uses a servoPulse function to move the servo from 10° - 170° and back again.

```
int servoPin = 2; // servo connected to digital pin 2
int myAngle; // angle of the servo roughly 0-180
int pulseWidth; // servoPulse function variable
void setup()
{
pinMode(servoPin, OUTPUT); // sets pin 2 as output
}
void servoPulse(int servoPin, int myAngle)
{
pulseWidth = (myAngle * 10) + 600; // determines delay
digitalWrite(servoPin, HIGH); // set servo high
delayMicroseconds(pulseWidth); // microsecond pause
digitalWrite(servoPin, LOW); // set servo low
}
void loop()
{
// servo starts at 10 deg and rotates to 170 deg
for (myAngle=10; myAngle<=170; myAngle++)
{
servoPulse(servoPin, myAngle); // send pin and angle
delay(20); // refresh cycle
}
// servo starts at 170 deg and rotates to 10 deg
for (myAngle=170; myAngle>=10; myAngle--)
{
servoPulse(servoPin, myAngle); // send pin and angle
delay(20); // refresh cycle
}
}
```

Let's get started

Now you'll begin to bring your Arduino to life. As you will see, there is more to working with Arduino than just the board itself. You'll learn how to plan projects in order to make your ideas a reality and then move on to a quick primer on electricity. Electricity is the driving force behind everything we do in this book, and it's important to have a solid understanding of the basics in order to create your own projects.

You'll also take a look at the components that help bring real projects to life. Finally, you'll examine some new functions that are the building blocks for your Arduino sketches.

Planning Your Projects

When starting your first few projects, you might be tempted to write your sketch immediately after you've come up with a new idea. But before you start writing, a few basic preparatory steps are in order. After all, your Arduino board isn't a mind-reader; it needs precise instructions, and even if these instructions can be executed by the Arduino, the results may not be what you expected if you overlooked even a minor detail.

Whether you are creating a project that simply blinks a light or an automated model railway signal, a detailed plan is the foundation of success. When designing your Arduino projects, follow these basic steps:

1. Define your objective. Determine what you want to achieve.

2. Write your algorithm. An algorithm is a set of instructions that describes how to accomplish your project. Your algorithm will list the steps necessary for you to achieve your project's objective.

3. Select your hardware. Determine how it will connect to the Arduino.

4. Write your sketch. Create your initial program that tells the Arduino what to do.

5. Wire it up. Connect your hardware, circuitry, and other items to the Arduino board.

6. Test and debug. Does it work? During this stage, you identify errors and find their causes, whether in the sketch, hardware, or algorithm.

The more time you spend planning your project, the easier time you'll have during the testing and debugging stage.

NOTE: Even well-planned projects sometimes fall prey to feature creep. Feature creep occurs when people think up new functionality that they want to add to a project and then try to force new elements into an existing design. When you need to change a design, don't try to "slot in" or modify it with 11th-hour additions. Instead, start fresh by redefining your objective.

About Electricity

Let's spend a bit of time discussing electricity, since you'll soon be building electronic circuits with your Arduino projects. In simple terms, electricity is a form of energy that we can harness and convert into heat, light, movement, and power. Electricity has three main properties that will be important to us as we build projects: current, voltage, and power.

Current

The flow of electrical energy through a circuit is called the current. Electrical current flows through a circuit from the positive side of a power source, such as a battery, to the negative side of the power source. This is known as direct current (DC). For the purposes of this book, we will not deal with AC (alternating current). In some circuits, the negative side is called ground (GND). Current is measured in amperes or "amps" (A). Small amounts of current are measured in milliamps (mA), where 1,000 milliamps equal 1 amp.

Voltage

Voltage is a measure of the difference in potential energy between a circuit's positive and negative ends. This is measured in volts (V). The greater the voltage, the faster the current moves through a circuit.

Power

Power is a measurement of the rate at which an electrical device converts energy from one form to another. Power is measured in watts (W). For example, a 100 W light bulb is much brighter than a 60 W bulb because the higher-wattage bulb converts more electrical energy into light.

A simple mathematical relationship exists among voltage, current, and power:

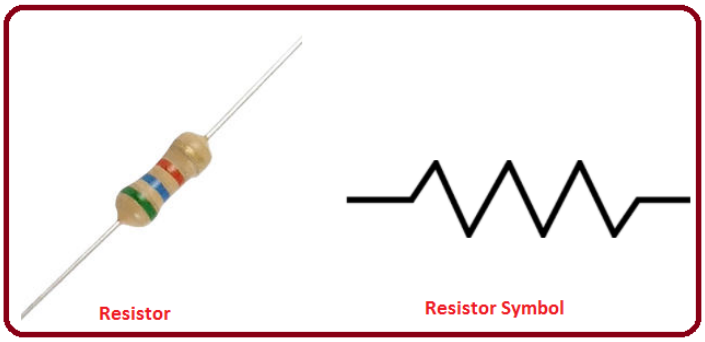
Power (W) = Voltage (V) × Current (A)

Electronic Components

Now that you know a little bit about the basics of electricity, let's look at how it interacts with various electronic components and devices. Electronic components are the various parts that control electric current in a circuit to make our designs a reality. Just as the various parts of a car work together to provide fuel, power, and mobility to allow us to drive, electronic components work together to control and harness electricity to help us create useful devices. Throughout this book, I'll explain specialized components as we use them. The following sections describe some of the fundamental components.

The Resistor

Various components, such as the Arduino's LED, require only a small amount of current to function—usually around 10 mA. When the LED receives excess current, it converts the excess to heat—too much of which can kill an LED. To reduce the flow of current to components such as LEDs, we can add a resistor between the voltage source and the component. Current flows freely along normal copper wire, but when it encounters a resistor, its movement is slowed. Some current is converted into a small amount of heat energy, which is proportional to the value of the resistor. Figure below shows an example of commonly used resistors.



Resistance

The level of resistance can be either fixed or variable. Resistance is measured in ohms (W) and can range from zero to thousands of ohms (kiloohms, or kΩ) to millions of ohms (megaohms, or M Ω).

Reading Resistance Values

Resistors are very small, so their resistance value usually cannot be printed on the components themselves. Although you can test resistance with a multimeter, you can also read resistance directly from a physical resistor, even without numbers. One common way to show the component's resistance is with a series of color-coded bands, read from left to right, as follows:

First band represents the first digit of the resistance

Second band represents the second digit of the resistance

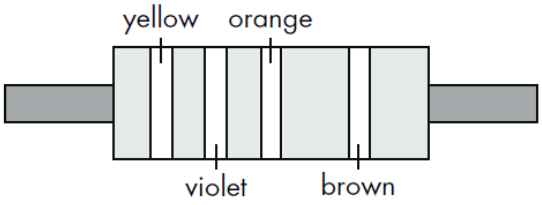
Third band represents the multiplier (for four-band resistors) or the third digit (for five-band resistors)

Fourth band represents the multiplier for five-band resistors Fifth band shows the tolerance (accuracy)

Table below lists the colors of resistors and their corresponding values. The fifth band represents a resistor's tolerance. This is a measure of the accuracy of the resistor. Because it is difficult to manufacture resistors with exact values, you select a margin of error as a percentage when buying a resistor. A brown band indicates 1 percent, gold indicates 5 percent, and silver indicates 10 percent tolerance.

Color	Ohms
Black	0
Brown	1
Red	2
Orange	3
Yellow	4
Green	5
Blue	6
Violet	7
Gray	8
White	9

Figure below shows a resistor diagram. The yellow, violet, and orange resistance bands are read as 4, 7, and 3, respectively, as listed in Table above. These values translate to 47,000 W, more commonly read as 47 kW.



Chip Resistors

Surface-mount chip resistors display a printed number and letter code, as shown in Figure below, instead of color stripes. The first two digits represent a single number, and the third digit represents the number of zeros to follow that number. For example, the resistor in Figure below has a value of 10,000 W, or 10 kW.

NOTE If you see a number and letter code on small chip resistors (such as 01C), google EIA-96 code calculator for lookup tables on that more involved code system.

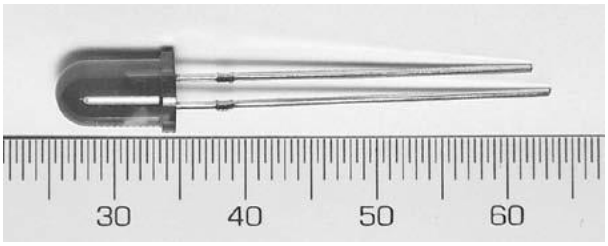
Multimeters

A multimeter is an incredibly useful and relatively inexpensive piece of test equipment that can measure voltage, resistance, current, and more. For example, Figure below shows a multimeter measuring a resistor.

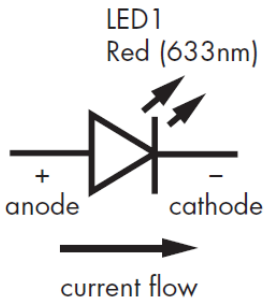
If you are colorblind, a multimeter is essential. As with other good tools, purchase your multimeter from a reputable retailer instead of fishing about on the Internet for the cheapest one you can find.

The Light-Emitting Diode (LED)

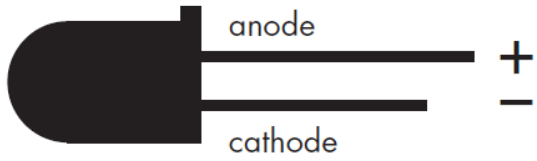
The LED is a very common, infinitely useful component that converts electrical current into light. LEDs come in various shapes, sizes, and colors. Figure below shows a common LED.



Connecting LEDs in a circuit takes some care, because they are polarized; this means that current can enter and leave the LED in one direction only. The current enters via the anode (positive) side and leaves via the cathode (negative) side, as shown in Figure below. Any attempt to make too much current flow through an LED in the opposite direction will break the component.



Thankfully, LEDs are designed so that you can tell which end is which. The leg on the anode side is longer, and the rim at the base of the LED is flat on the cathode side, as shown in Figure below.



When adding LEDs to a project, you need to consider the operating voltage and current. For example, common red LEDs require around 1.7 V and 5 to 20 mA of current. This presents a slight problem for us, because the Arduino outputs a set 5 V and a much higher current. Luckily, we can use a current-limiting resistor to reduce the current flow into an LED. But which value resistor do we use? That's where Ohm's Law comes in.

To calculate the required current-limiting resistor for an LED, use this formula:

R = (Vs − Vf) ÷ I, where,

Vs is the supply voltage (Arduino outputs 5 V).

Vf is the LED forward voltage drop (say, 1.7 V)

I is the current required for the LED (10 mA). (The value of I must be in amps, so 10 mA converts to 0.01 A.)

Now let's use this for our LEDs—with a value of 5 V for Vs , 1.7 V for Vf , and 0.01 A for I. Substituting these values into the formula gives a value for R of 330 Ω. However, the LEDs will happily light up when fed current less than 10 mA. It's good practice to use lower currents when possible to protect sensitive electronics, so we'll use 560 Ω, 1/4-watt resistors with our LEDs, which allow around 6 mA of current to flow.

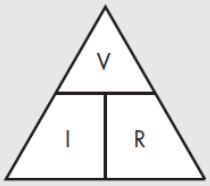
NOTE When in doubt, always choose a slightly higher value resistor, because it's better to have a dim LED than a dead one!

THE OHM'S LAW TRIANGLE

Ohm's Law states that the relationship between current, resistance, and voltage is as follows:

voltage (V) = current (I) × resistance (R)

If you know two of the quantities, then you can calculate the third. A popular way to remember Ohm's Law is with a triangle, as shown in Figure 3-9.



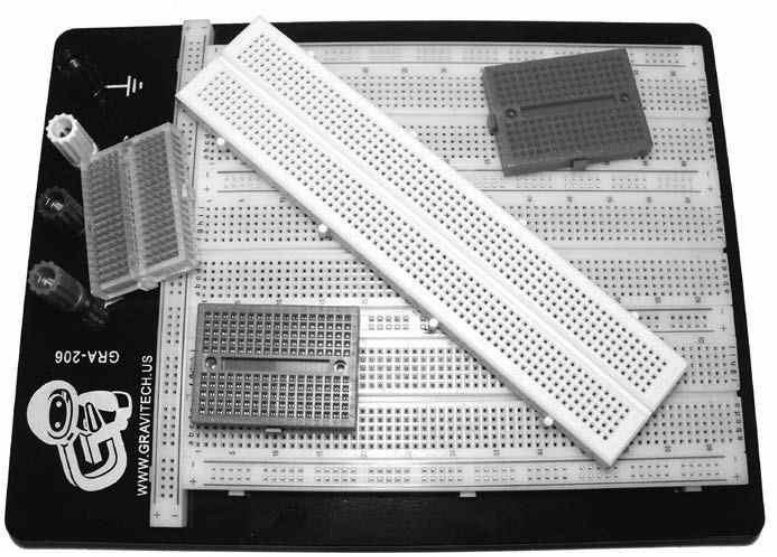
$V = I \times R$
$I = V \div R$
$R = V \div I$

Figure 3-9: The Ohm's Law triangle

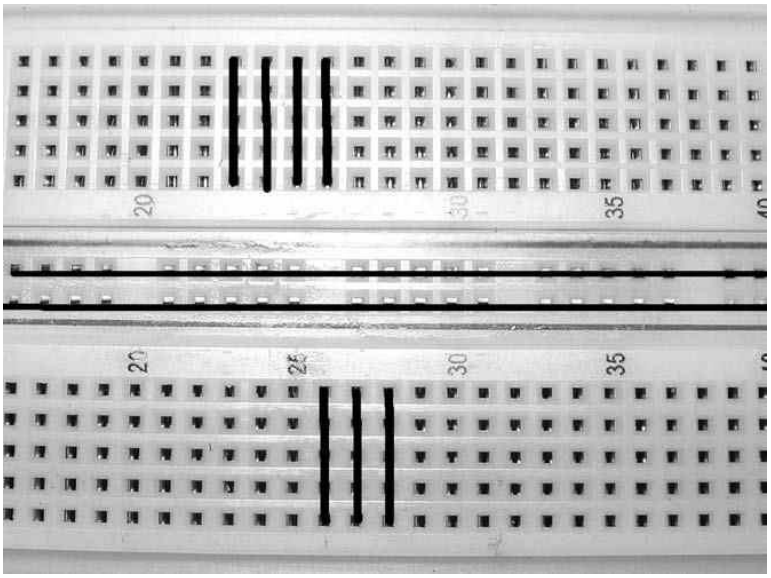
The Ohm's Law triangle diagram is a convenient tool for calculating voltage, current, or resistance when two of the three values are known. For example, if you need to calculate resistance, put your finger over *R*, which leaves you with voltage divided by current; to calculate voltage, cover *V*, which leaves you with current times resistance.

The Solderless Breadboard

Our ever-changing circuits will need a base—something to hold them together and build upon. A great tool for this purpose is a solderless breadboard. The breadboard is a plastic base with rows of electrically connected sockets (just don't cut bread on them). They come in many sizes, shapes, and colors, as shown in Figure below.



The key to using a breadboard is knowing how the sockets are connected—whether in short columns or in long rows along the edge or in the center. The connections vary by board. For example, in the breadboard shown in figure below, columns of five holes are connected vertically but isolated horizontally.



If you place two wires in one vertical row, then they will be electrically connected. By the same token, the long rows in the center between the horizontal lines are connected horizontally. Because you'll often need to connect a circuit to the supply voltage and ground, these long horizontal lines of holes are ideal for the supply voltage and ground. When you're building more complex circuits, a breadboard will get crowded and you won't always be able to place components exactly where you want. It's easy to solve this problem using short connecting wires.

Getting Started with Arduino projects

Project #1: LED Flasher

Let's start with blinking of LED.

The Algorithm

1. Turn on LED 1.
2. Wait for a second.
3. Turn off LED 1.
4. Repeat indefinitely.

The Hardware

- Arduino UNO
- Breadboard
- Arduino wire
- 2 * LED
- Jumper Wires
- 220 Ω resistor. (You can use any resistor between 100 ohm to 1k ohm)

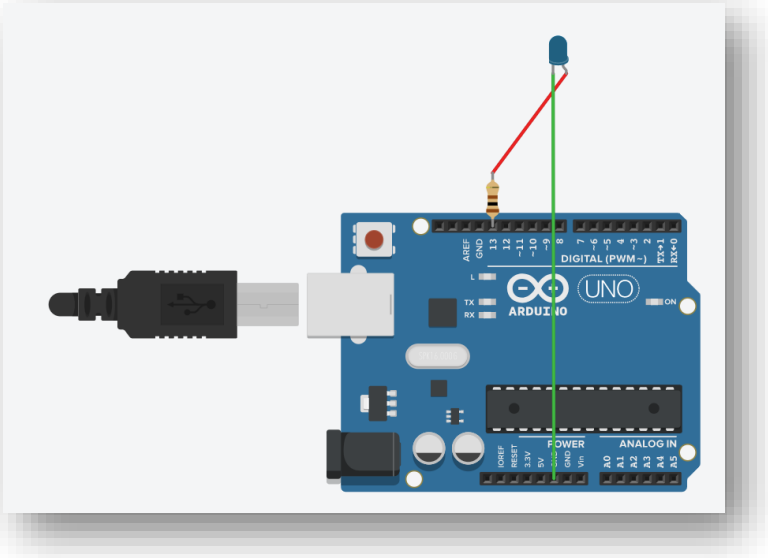
The Schematic

Here we are using Arduino UNO.

Digital pin number 13 is connected to 220 Ω resistor and then to anode pin of LED.

Cathode (-ve) pin of LED is connected to ground pin of Arduino.

That completes our circuit.



The Sketch

```
// Project 1 - LED Flasher
int ledPin = 13;
void setup() {
pinMode(ledPin, OUTPUT);
}
void loop() {
digitalWrite(ledPin, HIGH);
delay(1000);
digitalWrite(ledPin, LOW);
delay(1000);
}
```

Running the sketch

After verifying & uploading our code in Arduino board, we can see LED lighting up for 1 second and switching off for 1 second.

We can also see LED L i.e. inbuilt LED in Arduino board doing same thing as external LED, that's because digital pin number 13 is connected to LED L on the board.

You can experiment and change the pin number and delay timings for experimentation.

Code Overview

Let's take a look at the code for this project.

```
// Project 1 - LED Flasher
```

This is simply a comment in your code and is ignored by the compiler (the part of the IDE that turns your code into instructions the Arduino can understand before uploading it). Everything following // (double slashes) on a line is ignored by the compiler. This allows you to add notes to yourself and others who may read the code explaining how the code works.

Comments are essential in your code to help you understand what is going on and how your code works. Later on as your projects get more complex and your code expands into hundreds or maybe thousands of lines, comments will be vital in making it easy for you to see how it works. You may come up with an amazing piece of code, but if you go back and look at that code days, weeks or months later, you may forget how it all works. Comments will help you understand it easily. Also, if your code is meant to be seen by other people—and as the whole ethos of the Arduino, and indeed the whole Open Source community is to share code and schematics—I hope when you start making your own cool stuff with the Arduino, you will be willing to share it with the world.

Comments will enable others to understand what is going on in your code. You can also put comments into a block by using the /* and */ delimiters, for example:

/* All of the text within

the slash and the asterisks
is a comment and will be
ignored by the compiler */

The IDE will automatically turn the color of any commented text to grey.

The next line of the program is

```
int ledPin = 13;
```

This is what is known as a variable. A variable is a place to store data. Imagine a variable as a small box where you can keep things. A variable is called a variable because you can change its contents. Later on, we will carry out mathematical calculations on variables to make our program do more advanced things. In this case, you are setting up a variable of type int or integer. An integer is a number within the range of -32,768 to 32,767. Next, you have assigned that integer the name of ledPin and have given it a value of 13. We didn't have to call it ledPin; we could have called it anything we wanted to. But, as we want our variable name to be descriptive, we call it ledPin to show that the use of this variable is to set which pin on the Arduino we are going to use to connect our LED. In this case, we are using digital pin 13. At the end of this statement is a semicolon. This is a symbol to tell the compiler that this statement is now complete.

Although we can call our variables anything we want, every variable name in C must start with a letter; the rest of the name can consist of letters, numbers, and underscore characters. C recognizes upper and lower case characters as being different. Finally, you cannot use any of C's keywords such as main, while, switch, etc. as variable names. Keywords are constants, variables, and function names that are defined as part of the Arduino language. Don't use a variable name that is the same as a keyword. All keywords within the sketch will appear in red. So, you have set up an area in memory to store a number of type integer and have stored in that area the number 13. Next we have our setup() function

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

An Arduino sketch must have a setup() and loop() function, otherwise it will not work. The setup() function runs once and once only at the start of the program and is where you will issue general instructions to prepare the program before the main loop runs, such as setting up pin modes, setting serial baud rates, etc. Basically, a function is a block of code assembled into one convenient block. For example, if we created our own function to carry out a whole series of complicated mathematics that had many lines of code, we could run that code as many times as we liked simply by calling the function name instead of writing out the code again each time. Later on, we will go into functions in more detail when we start to create our own. In the case of our program, the setup() function only has one statement to carry out. The function starts with

void setup()

Here, we are telling the compiler that our function is called setup, that it returns no data (void) and that we pass no parameters to it (empty parenthesis). If our function returned an integer value and we also had integer values to pass to it (e.g. for the function to process), then it would look something like this

int myFunc(int x, int y)

In this case, we have created a function (or a block of code) called myFunc. This function has been passed two integers called x and y Once the function has finished, it will then return an integer value to the point after our function was called in the program (hence int before the function name).

All of the code within the function is contained within the curly braces. A { symbol starts the block of code and a } symbol ends the block. Anything in between those two symbols is code that belongs to the function. We will go into greater detail about functions later on in Project 4 in this chapter, so don't worry about them for now. All you need to know is that in this program, we have two functions, and the first function is called setup; its purpose is to setup anything necessary for our program to work before the main program loop runs.

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

Our setup function only has one statement and that is pinMode. Here we are telling the Arduino that we want to set the mode of one of our digital pins to be output mode, rather than input. Within the parenthesis, we put the pin number and the mode (OUTPUT or INPUT). Our pin number is ledPin, which has been previously set to the value 10 in our program. Therefore, this statement is simply telling the Arduino that the digital pin 10 is to be set to OUTPUT mode. As the setup() function runs only once, we now move onto the main function loop.

```
void loop() {
digitalWrite(ledPin, HIGH);
delay(1000);
digitalWrite(ledPin, LOW);
delay(1000);
}
```

The loop() function is the main program function and is called continuously as long as our Arduino is turned on. Every statement within the loop() function (within the curly braces) is carried out, one by one, step by step, until the bottom of the function is reached; then, the Arduino starts the loop again at the top of the function, and so on forever, or until you turn the Arduino off or press the Reset switch. In this project, we want the LED to turn on, stay on for one second, turn off and remain off for one second, and then repeat. Therefore, the commands to tell the Arduino to do that are contained within the loop() function, as we wish them to repeat over and over. The first statement is

```
digitalWrite(ledPin, HIGH);
```

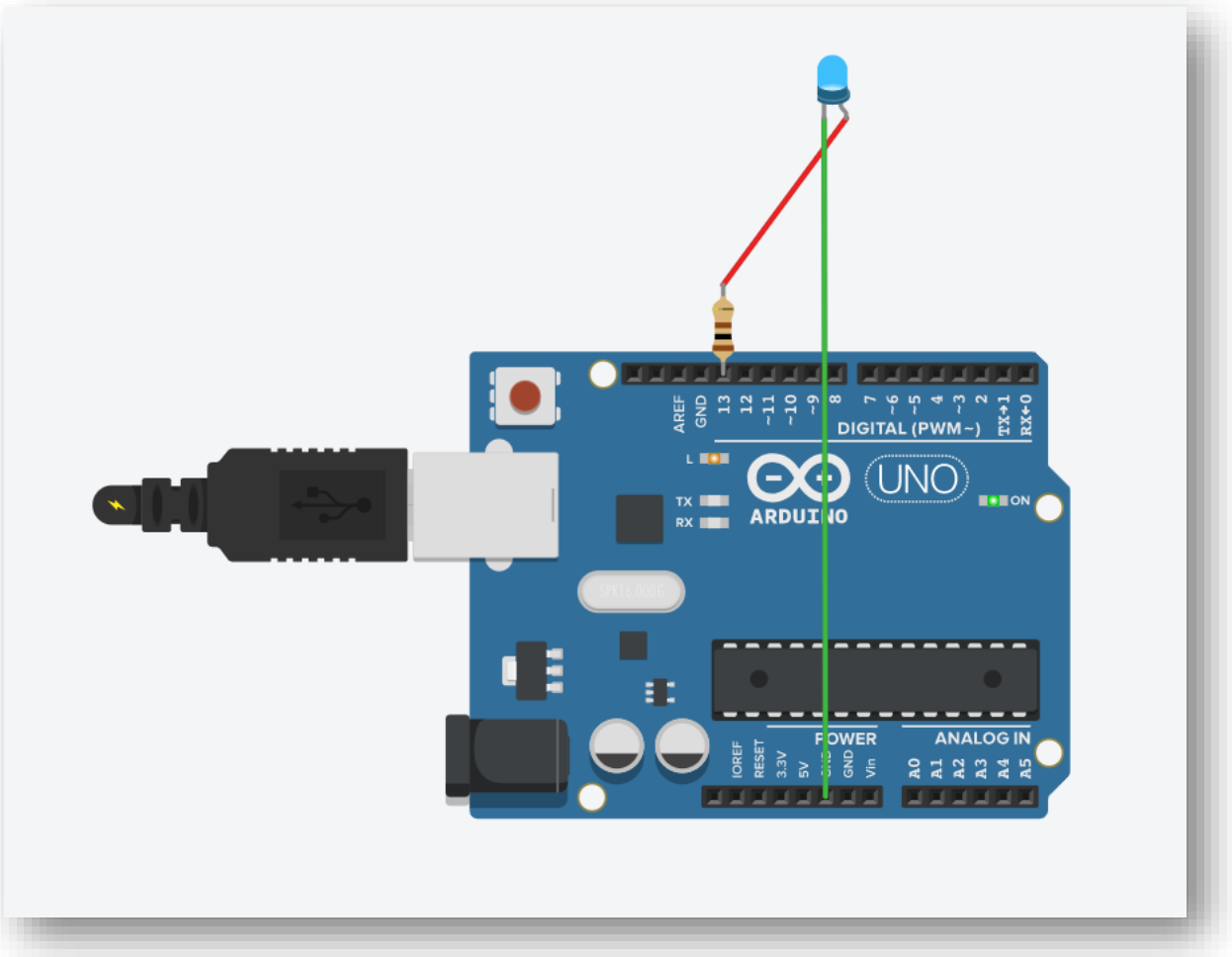
This writes a HIGH or a LOW value to the digital pin within the statement (in this case ledPin, which is digital pin 10). When you set a digital pin to HIGH, you are sending out 5 volts to that pin. When you set it to LOW, the pin becomes 0 volts, or ground. This statement therefore sends out 5v to digital pin 10 and turns the LED on. After that is

```
delay(1000);
```

This statement simply tells the Arduino to wait for 1,000 milliseconds (there are 1,000 milliseconds in a second) before carrying out the next statement which is

```
digitalWrite(ledPin, LOW);
```

This will turn off the power going to digital pin 10, and therefore turn the LED off. There is then another delay statement for another 1,000 milliseconds and then the function ends. However, as this is our main loop() function, the function will now start again at the beginning. By following the program structure step by step again, we can see that it is very simple.



Project #2: Creating a Blinking LED Wave

Let's put some LEDs and resistors to work. In this project, we'll use five LEDs to creating a kind of wavelike light pattern.

The Algorithm

Here's our algorithm for this project:

1. Turn on LED 1.
2. Wait half a second.
3. Turn off LED 1.
4. Turn on LED 2.
5. Wait half a second.
6. Turn off LED 2.
7. Continue until LED 5 is turned on, at which point the process reverses from LED 5 to 1.
8. Repeat indefinitely.

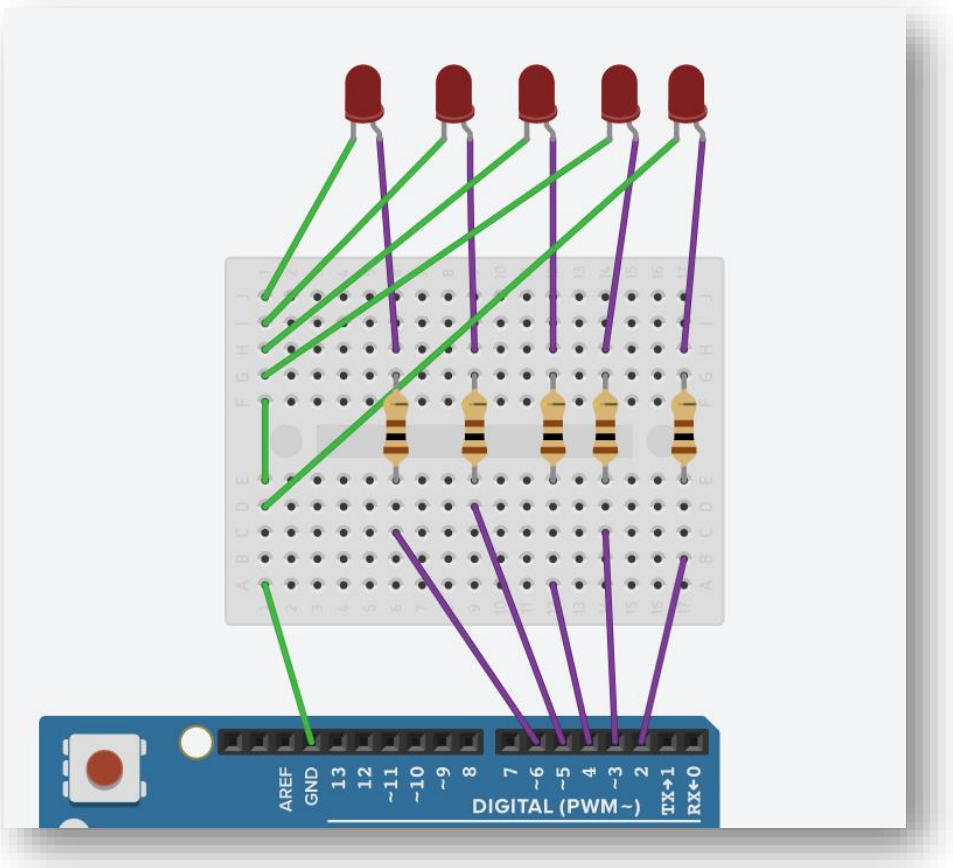
The Hardware

Here's what you'll need to create this project:

- Arduino UNO and connecting wire
- Five LEDs (any colour)
- Five resistors (you can use any between 100 to 1k ohm)
- One breadboard
- Jumper wires

The Schematic

Now let's build the circuit. Circuit layout can be described in several ways. For the first few projects in this book, we'll use physical layout diagrams similar to the one shown in Figure below. By comparing the wiring diagram to the functions in the sketch, you can begin to make sense of the circuit. For example, when we use digitalWrite(2, HIGH), a high voltage of 5 V flows from digital pin 2, through the current-limiting resistor, through the LED via the anode and then the cathode, and finally back to the Arduino's GND socket to complete the circuit. Then, when we use digitalWrite(2, LOW), the current stops and the LED turns off.



The Sketch

Now for our sketch. Enter this code into the IDE:

```
// Project 2 - Creating a Blinking LED Wave
u void setup()
{
pinMode(2, OUTPUT); // LED 1 control pin is set up as an output
pinMode(3, OUTPUT); // same for LED 2 to LED 5
pinMode(4, OUTPUT);
pinMode(5, OUTPUT);
pinMode(6, OUTPUT);
}
void loop()
{
digitalWrite(2, HIGH); // Turn LED 1 on
delay(500); // wait half a second
digitalWrite(2, LOW); // Turn LED 1 off
digitalWrite(3, HIGH); // and repeat for LED 2 to 5
delay(500);
digitalWrite(3, LOW);
digitalWrite(4, HIGH);
```

```
delay(500);
digitalWrite(4, LOW);
digitalWrite(5, HIGH);
delay(500);
digitalWrite(5, LOW);
digitalWrite(6, HIGH);
delay(500);
digitalWrite(6, LOW);
digitalWrite(5, HIGH);
delay(500);
digitalWrite(5, LOW);
digitalWrite(4, HIGH);
delay(500);
digitalWrite(4, LOW);
digitalWrite(3, HIGH);
delay(500);
digitalWrite(3, LOW);
// the loop() will now loop around and start from the top again
}
```

In void setup(), the digital I/O pins are set to outputs, because we want them to send current to the LEDs on demand. We specify when to turn on each LED using the digitalWrite() function in the void loop() section of the sketch.

Running the Sketch

Now connect your Arduino and upload the sketch. After a second or two, the LEDs should blink from left to right and then back again. Success is a wonderful thing—embrace it! If nothing happens, however, then immediately remove the USB cable from the Arduino and check that you typed the sketch correctly. If you find an error, fix it and upload your sketch again. If your sketch matches exactly and the LEDs still don't blink, check your wiring on the breadboard. You now know how to make an LED blink with your Arduino, but this sketch is somewhat inefficient. For example, if you wanted to modify this sketch to make the LEDs cycle more quickly, you would need to alter each delay(500).

There is a better way.

Using Variables

In computer programs, we use variables to store data. For example, in the sketch for Project 2, we used the function delay(500) to keep the LEDs turned on.

The problem with the sketch as written is that it's not very flexible. If we want to make a change to the delay time, then we have to change each entry manually. To address this problem, we'll

create a variable to represent the value for the delay() function. Enter the following line in the Project 2 sketch above the void setup() function and just after the initial comment:

int d = 250;

This assigns the number 250 to a variable called d. Next, change every 500 in the sketch to a d. Now when the sketch runs, the Arduino will use the value in d for the delay() functions. When you upload the sketch after making these changes, the LEDs will turn on and off at a much faster rate, as the delay value is much smaller at the 250 value. int indicates that the variable contains an integer—a whole number between −32,768 and 32,767. Simply put, any integer value has no fraction or decimal places. Now, to alter the delay, simply change the variable declaration at the start of the sketch. For example, entering 100 for the delay would speed things up even more:

int d = 100;

Experiment with the sketch, perhaps altering the delays and the sequence of HIGH and LOW. Have some fun with it. Don't disassemble the circuit yet, though; we'll continue to use it with more projects in this chapter.

Project #3: Repeating with for Loops

When designing a sketch, you'll often repeat the same function. You could simply copy and paste the function to duplicate it in a sketch, but that's inefficient and a waste of your Arduino's program memory. Instead, you can use for loops. The benefit of using a for loop is that you can determine how many times the code inside the loop will repeat. To see how a for loop works, enter the following code as a new sketch:

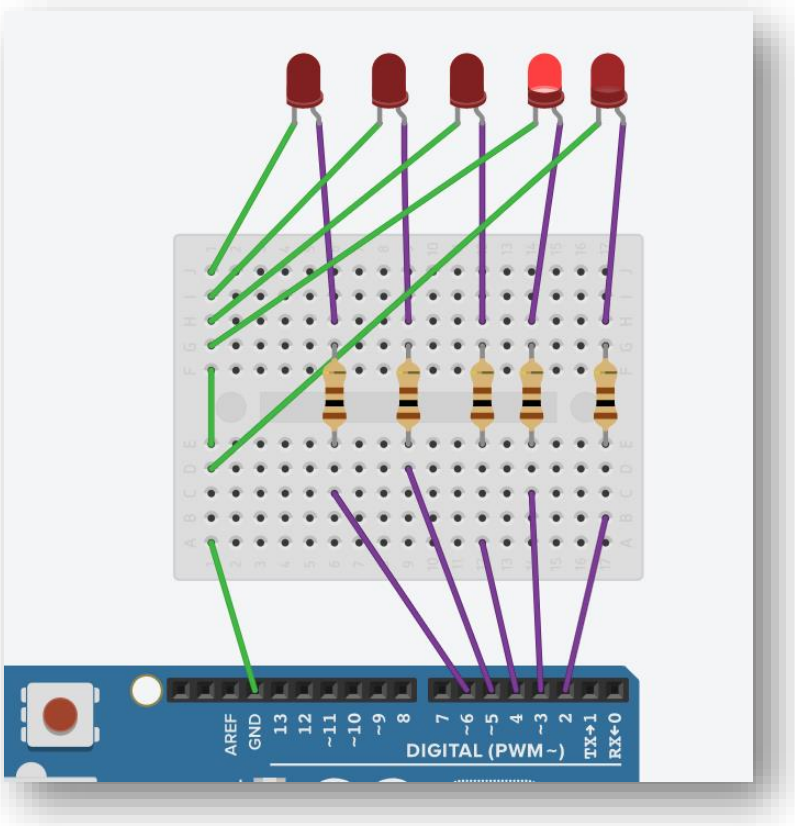
```
// Project 3 - Repeating with for Loops
int d = 100;
void setup()
{
pinMode(2, OUTPUT);
pinMode(3, OUTPUT);
pinMode(4, OUTPUT);
pinMode(5, OUTPUT);
pinMode(6, OUTPUT);
}
void loop() {

for ( int a = 2; a < 7 ; a++ )
{
digitalWrite(a, HIGH);
delay(d);
digitalWrite(a, LOW);
delay(d);
}
}
```

The for loop will repeat the code within the curly brackets beneath it as long as some condition is true. Here, we have used a new integer variable, a, which starts with the value 2. Every time the code is executed, the a++ will add 1 to the value of a. The loop will continue in this fashion while the value of a is less than 7 (the condition). Once it is equal to or greater than 7, the Arduino moves on and continues with whatever code comes after the for loop.

The number of loops that a for loop executes can also be set by counting down from a higher number to a lower number. To demonstrate this, add the following loop to the Project 3 sketch after the first for loop:

```
for ( int a = 5 ; a > 1 ; a-- )
{
digitalWrite(a, HIGH);
delay(d);
digitalWrite(a, LOW);
delay(d);
}
```

Here, the for loop sets the value of a equal to 5 and then subtracts 1 after every loop due to the a--. The loop continues in this manner while the value for a is greater than 1 (a > 1) and finishes once the value of a falls to 1 or less than 1. We have now re-created Project 1 using less code. Upload the sketch and see for yourself!

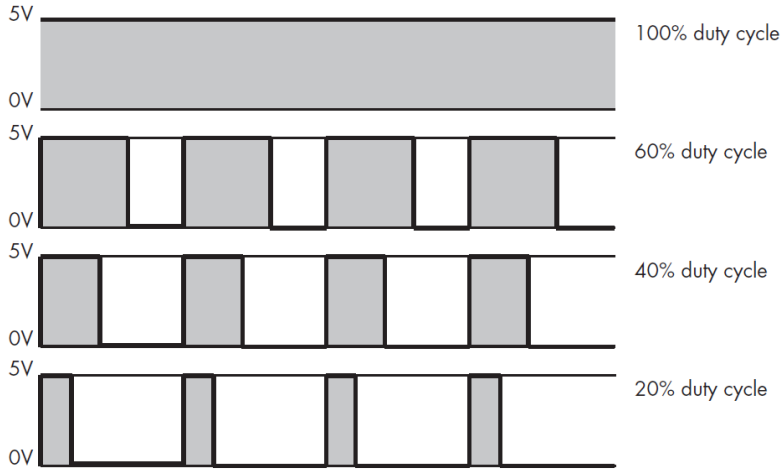


Varying LED Brightness with Pulse-Width Modulation

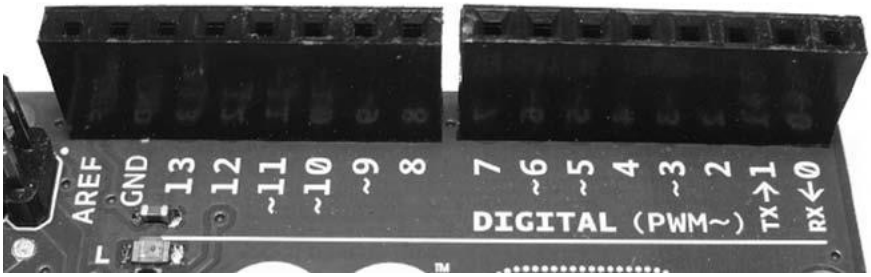
Rather than just turning LEDs on and off rapidly using digitalWrite(), we can define the level of brightness of an LED by adjusting the amount of time between each LED's on and off states using pulse-width modulation (PWM). PWM can be used to create the illusion of an LED being on at different levels of brightness by turning the LED on and off rapidly, at around 500 cycles per second. The brightness we perceive is determined by the amount of time the digital output pin is on versus the amount of time it is off—that is, every time the LED is lit or unlit. Because our eyes can't see flickers faster than 50 cycles per second, the LED appears to have a constant brightness.

The greater the duty cycle (the longer the pin is on compared to off in each cycle), the greater the perceived brightness of the LED connected to the digital output pin.

Figure below shows various PWM duty cycles. The filled-in gray areas represent the amount of time that the light is on. As you can see, the amount of time per cycle that the light is on increases with the duty cycle.



Only digital pins 3, 5, 6, 9, 10, and 11 on a regular Arduino board can be used for PWM. They are marked on the Arduino board with a tilde (~), as shown in Figure below.



To create a PWM signal, we use the function analogWrite(x, y), where x is the digital pin and y is a value for the duty cycle, between 0 and 255, where 0 indicates a 0 percent duty cycle and 255 indicates 100 percent duty cycle.

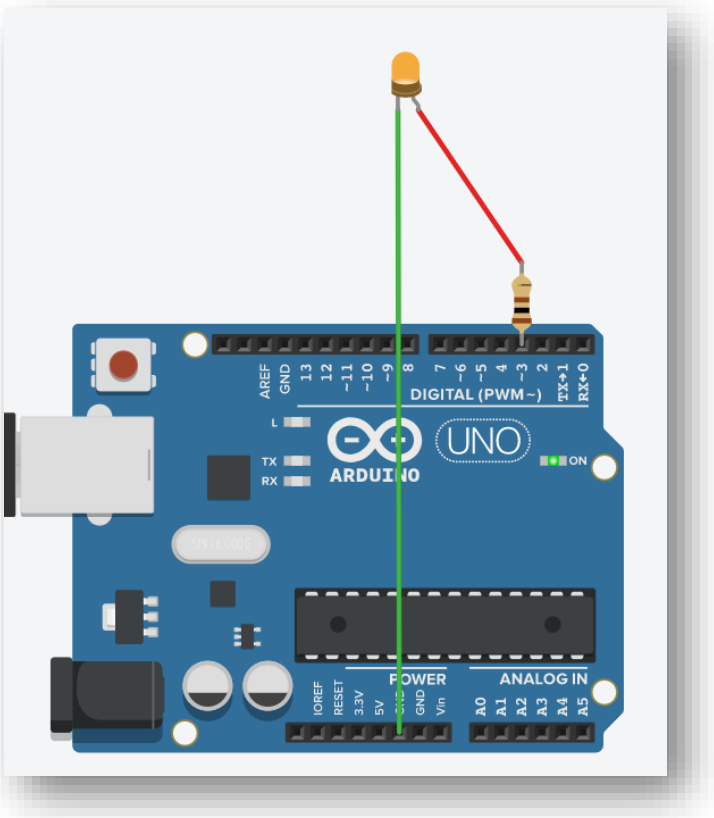
Project #4: Demonstrating PWM

In this project we are going to use PWM pins to generate fading effect in LED.

Hardware

Same as project 1.

The Schematic



The Sketch

```
// Project 4 - Demonstrating PWM
int d = 5;
void setup()
{
pinMode(3, OUTPUT); // LED control pin is 3, a PWM capable pin
}
```

```
void loop() {

for ( int a = 0 ; a < 256 ; a++ )
 {
analogWrite(3, a);
delay(d);
}

for ( int a = 255 ; a >= 0 ; a-- )
{
analogWrite(3, a);
delay(d);
}

delay(200);
}
```

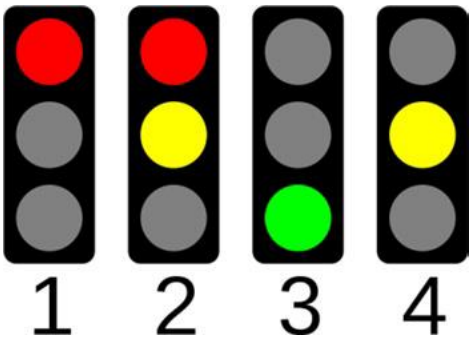
The LED on digital pin 3 will exhibit a "breathing effect" as the duty cycle increases and decreases. In other words, the LED will turn on, increasing in brightness until fully lit, and then reverse.

Code Overview

Project #5: Traffic Lights

We are now going to create a set of traffic lights based on the system that will change from green to red, via amber, and back again, after a set length of time using the four-state system.

The Algorithm

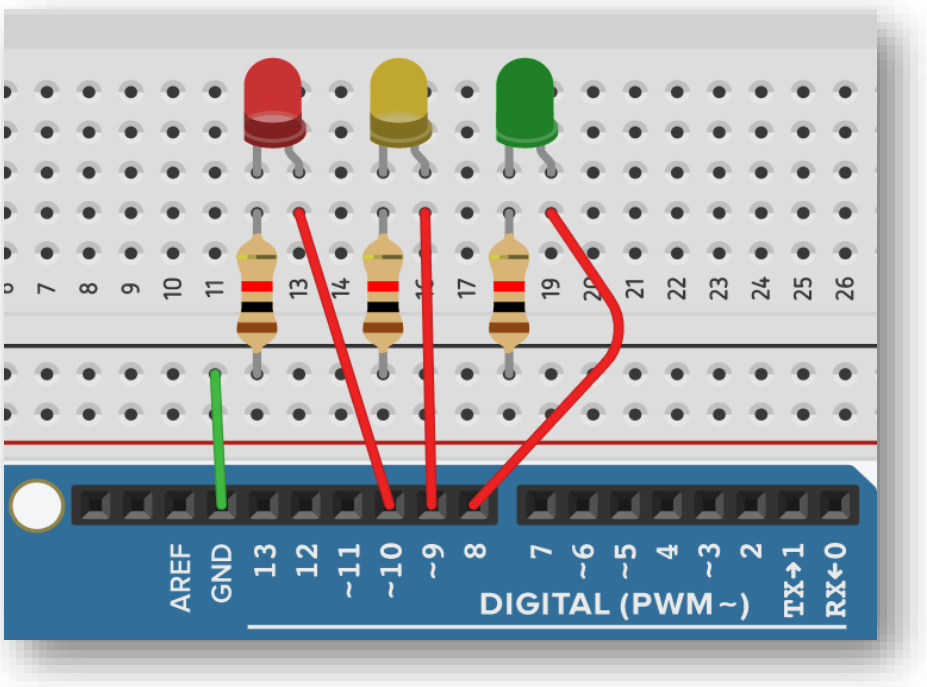


1. We will first keep red pin high for 5 second.
2. Turn Yellow and Red pin high for next 2 seconds.
3. Turn off Red and Yellow and Turn Green pin high for 5 seconds.
4. Then turn Yellow pin high for 2 seconds after turning Green pin to low.
5. Repeat.

Hardware

- Arduino UNO and connecting wire
- Breadboard
- Jumper Wires
- Red, Green and Yellow LED
- 3 * 220 Ω resistors

The Schematic



Here Anode pins of LED are directly connected with digital pins of Arduino and Cathode pins of LED are connected to ground through resistors.

This completes the circuit and the LED will light up once we pass current through digital pins.

The Sketch

```
// Project 5 - Traffic Lights
int ledDelay = 5000; // delay in between changes
int redPin = 10;
int yellowPin = 9;
int greenPin = 8;
void setup() {
pinMode(redPin, OUTPUT);
pinMode(yellowPin, OUTPUT);
pinMode(greenPin, OUTPUT);
}
void loop() {
digitalWrite(redPin, HIGH); // turn the red light on
delay(ledDelay); // wait 5 seconds
digitalWrite(yellowPin, HIGH); // turn on yellow
delay(2000); // wait 2 seconds
digitalWrite(greenPin, HIGH); // turn green on
```

```
digitalWrite(redPin, LOW); // turn red off
digitalWrite(yellowPin, LOW); // turn yellow off
delay(ledDelay); // wait ledDelay milliseconds
digitalWrite(yellowPin, HIGH); // turn yellow on
digitalWrite(greenPin, LOW); // turn green off
delay(2000); // wait 2 seconds
digitalWrite(yellowPin, LOW); // turn yellow off
// now our loop repeats
}
```

Code Overview

Project #6 LED Chase Effect

We are now going to use a string of LEDs (10 in total) to make an LED chase effect.

The Algorithm

If a LED is on for more than 65 milliseconds then current from current LED will be shifted to next LED.

Check if time difference between last LED change and current time is greater than 65 seconds.

If yes, turn off all LEDs and turn current LED pin number HIGH.

Then keep on increasing current LED pin number by 1 till it reaches 9.

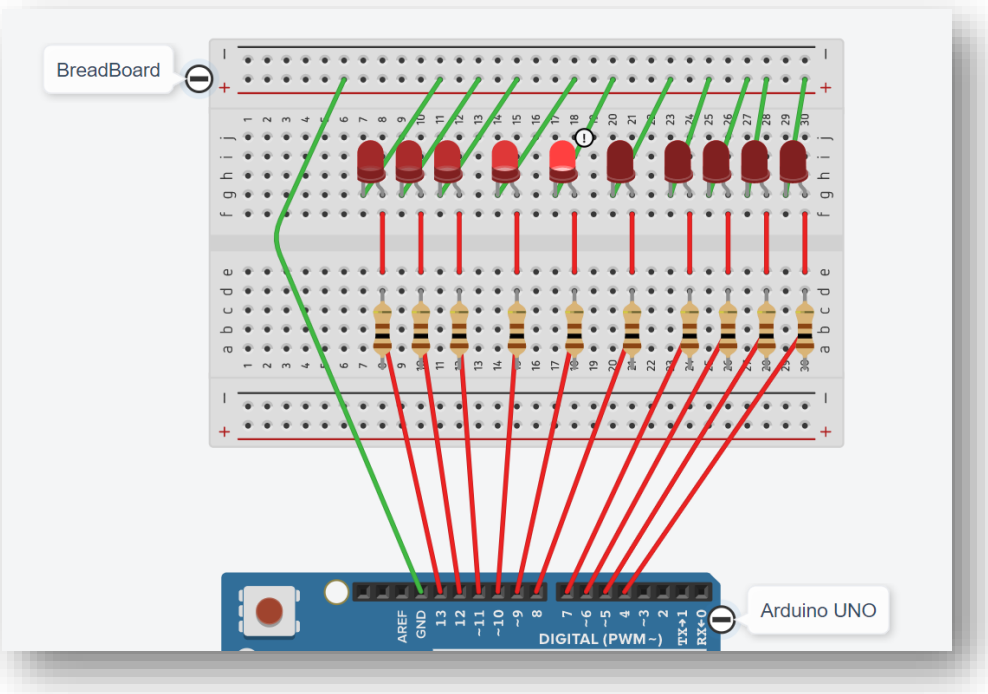
After value of current LED reaches 9, change direction of increment of LED to -1 from 1.

Again if current LED value reaches 0 then change value of direction of increment to 1 from 0.

Hardware

- Arduino UNO
- Arduino Wire
- 10 x 5mm LEDs
- 10 x Current-Limiting Resistors (from 100 ohm to 1k ohm, you can take any)

The Schematic



The Sketch

```
// Project 5 - LED Chase Effect
byte ledPin[] = {4, 5, 6, 7, 8, 9, 10, 11, 12, 13}; // Create array for LED pins
int ledDelay = 65; // delay between changes
int direction = 1;
int currentLED = 0;
unsigned long changeTime;
void setup() {
for (int x=0; x<10; x++) { // set all pins to output
pinMode(ledPin[x], OUTPUT);
}
changeTime = millis();
}
void loop() {
if ((millis() - changeTime) > ledDelay) { // if it has been ledDelay ms since last ch
ange
changeLED();
changeTime = millis();
}
}

void changeLED() {
```

```
for (int x=0; x<10; x++) { // turn off all LED's
digitalWrite(ledPin[x], LOW);
}
digitalWrite(ledPin[currentLED], HIGH); // turn on the current LED
currentLED += direction; // increment by the direction value
// change direction if we reach the end
if (currentLED == 9) {direction = -1;}
if (currentLED == 0) {direction = 1;}
}
```

Code Overview

Our very first line in this sketch is

```
byte ledPin[] = {4, 5, 6, 7, 8, 9, 10, 11, 12, 13};
```

which declares the ledPin variable to be an array (collection) of elements of data type byte. All the elements share the same name (in this case, ledPin), but can be selected individually by an index number, much like apartments in an apartment building. The [] after the variable name in the declaration tells the compiler this is an array variable rather than a simple (non-indexed) variable. We have then initialized the array with 10 values, which are the digital pin numbers 4 through to 13. To access an element of the array, we follow the array name with the index number of that element between square brackets. The index number between the [] doesn't have to be a constant—it can be a variable or expression. Arrays are zero indexed, which simply means that the index of the first element is zero, rather than one. So in our 10-element array, the index numbers are 0 to 9. In this case, the third element (ledPin[2]) has the value of 6, and the seventh element (ledPin[6]) has a value of 10.

You have to tell the compiler the size of the array if you do not initialize it with data first in the declaration. In our sketch we did not explicitly choose a size, as the compiler is able to count the values we have assigned to the array to work out that the size is 10 elements. If we had declared the array, but not initialized it with values at the same time, we would need to declare a size, for example we could have done this:

```
byte ledPin[10];
```

and then loaded data into the elements later on. To retrieve a value from the array, we would do somethinglike this:

```
x = ledpin[5];
```

In this example, x would now hold a value of 8. To get back to your program, we have started off by declaring andinitializing an array with 10 values that are the digital pin numbers used for the outputs to our 10 LEDs.

In our main loop, we check that at least ledDelay milliseconds have passed since the last change of LEDs, and ifso, it passes control to our function. The reason we are only going to pass control to

the changeLED() function in thisway, rather than using delay() commands, is to allow other code, if needed, to run in the main program loop (as longas that code takes less than ledDelay to run.

The function we created is

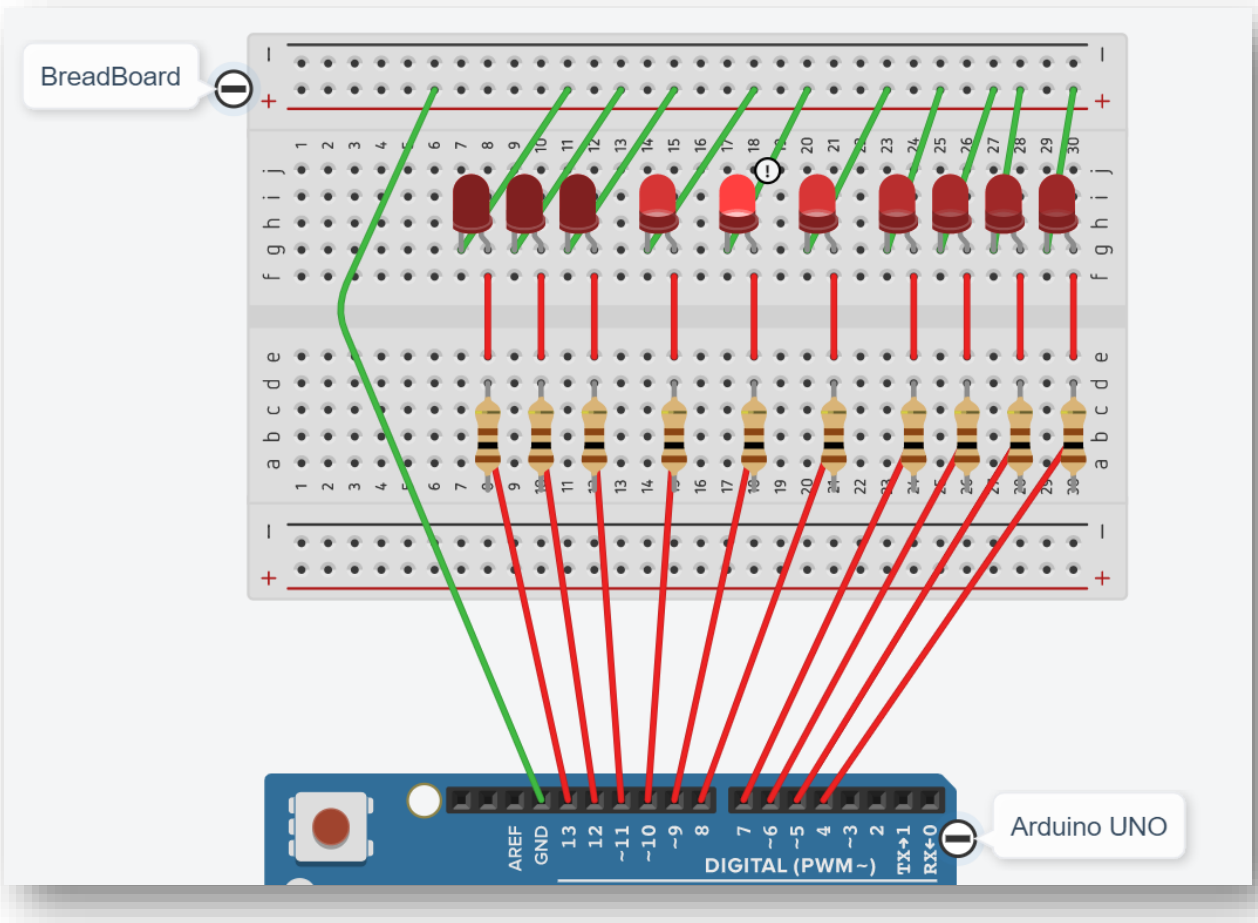
```
void changeLED() {
// turn off all LED's
for (int x=0; x<10; x++) {
digitalWrite(ledPin[x], LOW);
}
// turn on the current LED
digitalWrite(ledPin[currentLED], HIGH);
// increment by the direction value
currentLED += direction;
// change direction if we reach the end
if (currentLED == 9) {direction = -1;}
if (currentLED == 0) {direction = 1;}
}
```

and the job of this function is to turn all LEDs off and then turn on the current LED (this is done so fast you will not see it happening), which is stored in the variable current LED.

This variable then has direction added to it. As direction can only be either a 1 or a −1, then the number will either increase (+1) or decrease by one (current LED +(−1)).We then have an if statement to see if we have reached the end of the row of LEDs, and if so, we then reverse the direction variable.

By changing the value of led Delay, you can make the LED ping back and forth at different speeds. Try different values to see what happens. You have to stop the program and manually change the value of led Delay, then upload the amended code to see any changes.

However, wouldn't it be nice to be able to adjust the speed while the program is running? Yes, it would, so let's do exactly that in the next project by introducing a way to interact with the program and adjust the speed using a potentiometer.


BreadBoard
Arduino UNO
AREF
GND
13
12
~11
~10
~9
8
7
~6
~5
4
~3
2
TX→1
RX←0
DIGITAL (PWM~)

How analog pins work & how we take analog input?

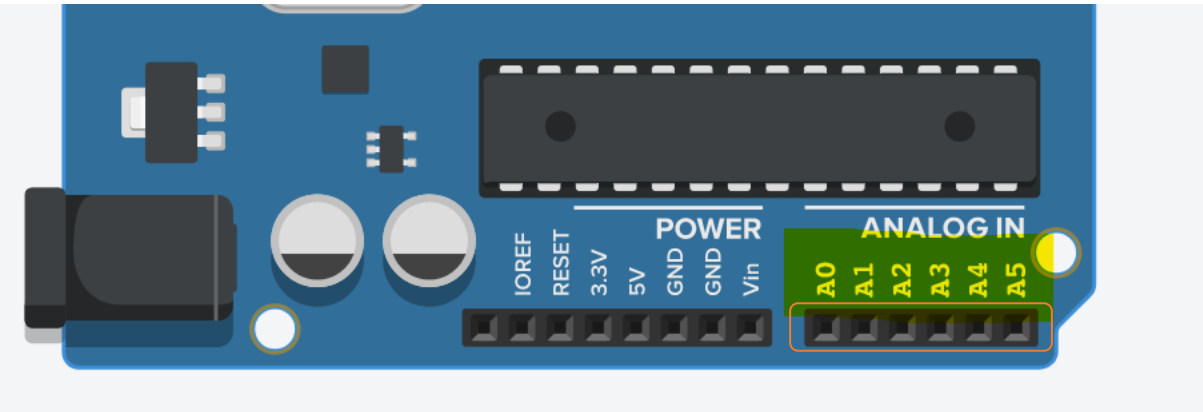
Microcontrollers are capable of detecting binary signals: is the button pressed or not? These are digital signals. When a microcontroller is powered from five volts, it understands zero volts (0V) as a binary 0 and a five volts (5V) as a binary 1.

The world however is not so simple and likes to use shades of gray. What if the signal is 2.72V? Is that a zero or a one? We often need to measure signals that vary; these are called analog signals. A 5V analog sensor may output 0.01V or 4.99V or anything in between.

Luckily, nearly all microcontrollers have a device built into them that allows us to convert these voltages into values that we can use in a program to make a decision.

What is the ADC?

An Analog to Digital Converter (ADC) is a very useful feature that converts an analog voltage on a pin to a digital number. By converting from the analog world to the digital world, we can begin to use electronics to interface to the analog world around us.



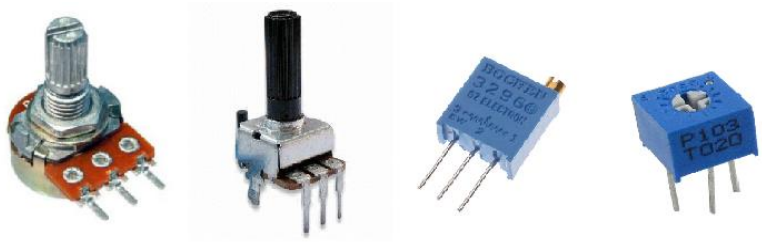
Not every pin on a microcontroller has the ability to do analog to digital conversions. On the Arduino board, these pins have an 'A' in front of their label (A0 through A5) to indicate these pins can read analog voltages.

ADCs can vary greatly between microcontrollers. The ADC on the Arduino is a 10-bit ADC meaning it has the ability to detect 1,024 (2^10) discrete analog levels. Some microcontrollers have 8-bit ADCs (2^8 = 256 discrete levels) and some have 16-bit ADCs (2^16 = 65,536 discrete levels).

In our case ADC maps (0 – 5) volts to (0 to 1023). Where 0 volt being 0 & 5 volt being 1023 reading.

Project #7 Using potentiometer to control brightness

Using potentiometer to set brightness of the LED.

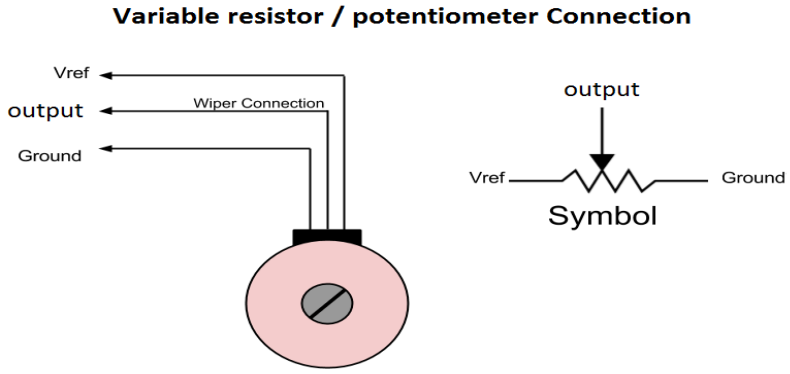


Potentiometers are variable resistors and they function to alter their resistance via a knob or dial. You have probably used one before by adjusting the volume on your stereo or using a light dimmer.

Potentiometers have a range of resistance. They can be attuned from zero ohms to whatever maximum resistance that is specific to it. For example, a potentiometer of 10 kΩ can be adjusted from 0 Ω to its maximum of 10 kΩ.

In this project we will learn how to use a potentiometer with and without Arduino board to fade an LED.

We will also learn how to use analogRead() and map() functions.

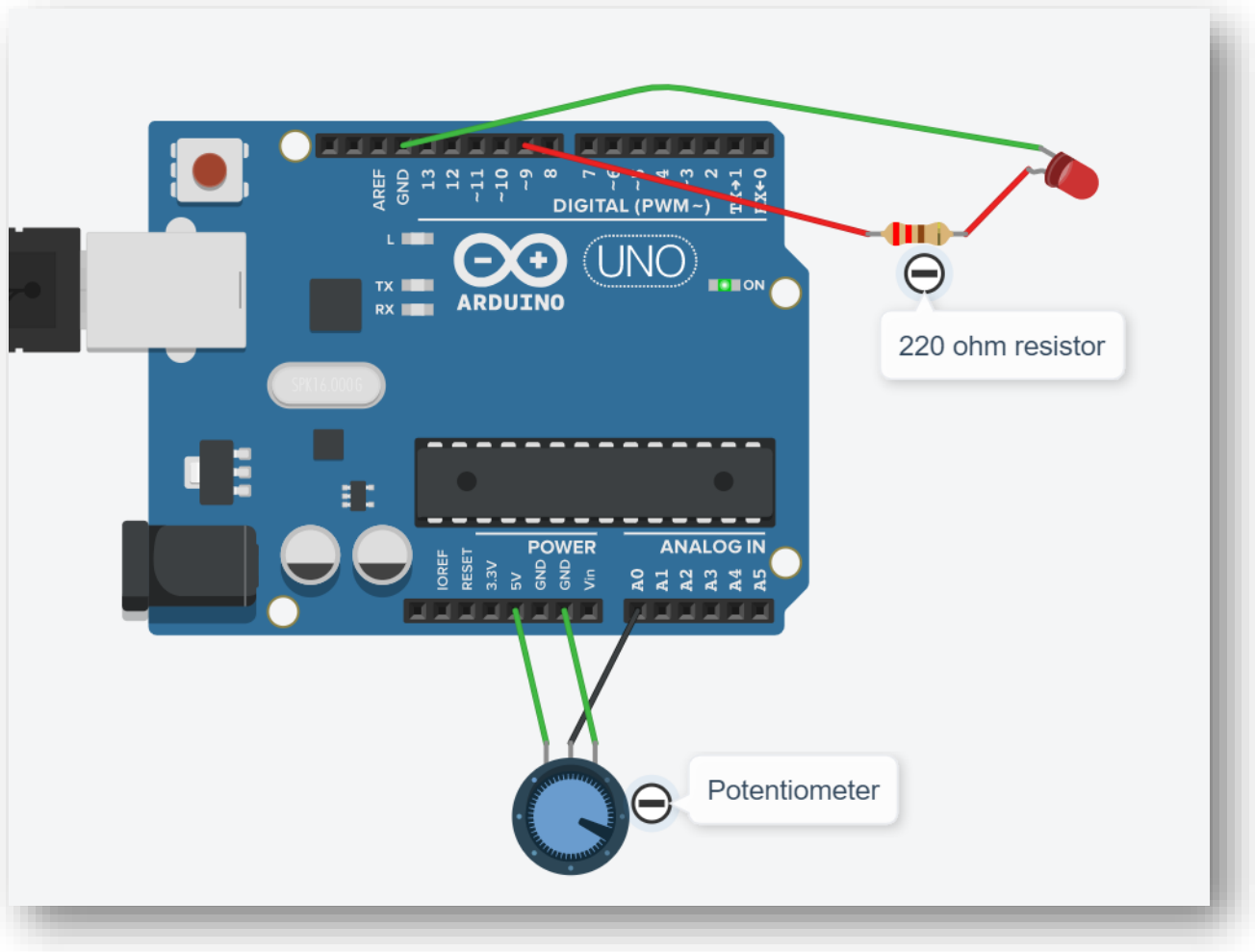


All potentiometers have three pins. The outer pins are used for connecting power source (Vref and gnd). The middle pin (output) give us the variable of resistance value.

Hardware

- Arduino Uno and connecting wire
- LED
- Potentiometer
- Resistor (Choose any between 100Ω to 1kΩ)

The Schematic



The Sketch

```
//Constants:
const int ledPin = 9;  //pin 9 has PWM function
const int potPin = A0; //pin A0 to read analog input

//Variables:
int value; //save analog value


void setup(){
  //Input or output
  pinMode(ledPin, OUTPUT);
  pinMode(potPin, INPUT); //Optional

}
```

```
void loop(){

  value = analogRead(potPin); //Read and save analog value from potentiometer
  value = map(value, 0, 1023, 0, 255); //Map value 0-1023 to 0-255 (PWM)
  analogWrite(ledPin, value);  //Send PWM value to led
}
```

Here we are taking input from potentiometer, a potentiometer sends value from 0 to 1023 when turned from one side to other.

Then we are using **map** function to change value of input from **value** variable from (0 – 1023) to (0 – 255).

By turning the shaft of the potentiometer, we change the amount of resistance on either side of the wiper which is connected to the center pin of the potentiometer. This changes the relative "closeness" of that pin to 5 volts and ground, giving us a different analog input. When the shaft is turned all the way in one direction, there are 0 volts going to the pin, and we read 0. When the shaft is turned all the way in the other direction, there are 5 volts going to the pin and we read 1023. In between, analogRead() returns a number between 0 and 1023 that is proportional to the amount of voltage being applied to the pin.

Code Overview

- Read analog value from potentiometer middle pin
  -> value = analogRead(potPin)
- Map analog values 0-1023 to pwm values 0-255
  -> value = map(value, 0, 1023, 0, 255);

  To know more about map() function
- Send pwm value to led

-> analogWrite(ledPin, value);

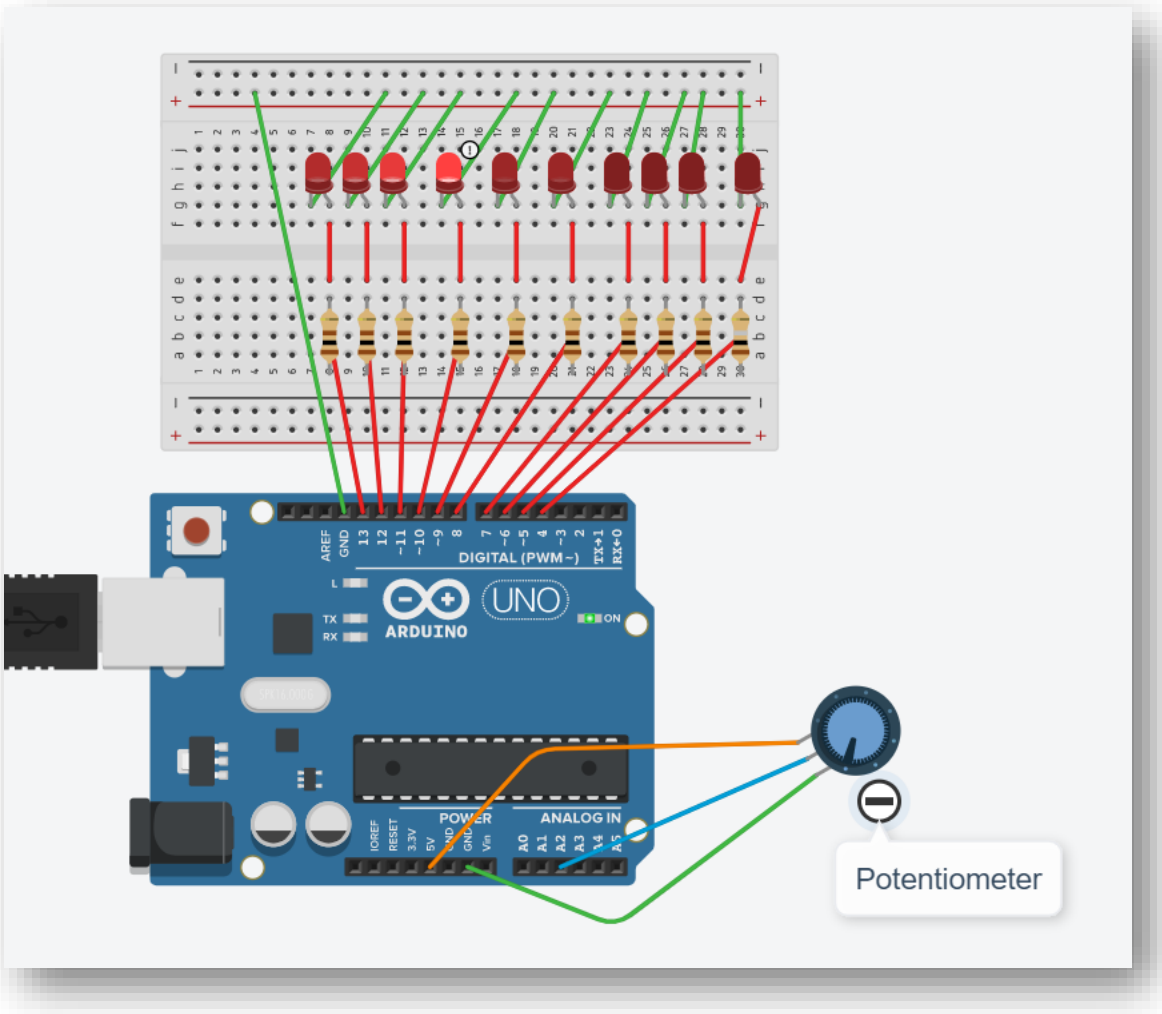
Project #8 Interactive LED Chase Effect

Leave your circuit board as it was in Project 6. We are going to add a potentiometer to this circuit which will allow you to change the speed of the lights while the code is running.

Hardware

Same as project #6 + we will need one 4.7 KΩ or 10 KΩ potentiometer.

The Schematic



The Sketch

```
byte ledPin[] = {4, 5, 6, 7, 8, 9, 10, 11, 12, 13}; // Create array for LED pins
int ledDelay, value; // delay between changes
int direction = 1;
int currentLED = 0;
unsigned long changeTime;
int potPin = 2; // select the input pin for the potentiometer

void setup() {
for (int x=0; x<10; x++) { // set all pins to output
pinMode(ledPin[x], OUTPUT);
}
changeTime = millis();
}
void loop() {
value = analogRead(potPin); // read the value from the pot
ledDelay = map(value, 0, 1023, 0, 200); //value from potentiometer will be from 0 to
1023, we are mapping
// that from 0 to 200 here

if ((millis() - changeTime) > ledDelay) { // if it has been ledDelay ms since last ch
ange
changeLED();
changeTime = millis();
}
}
void changeLED() {
for (int x=0; x<10; x++) { // turn off all LED's
digitalWrite(ledPin[x], LOW);
}
digitalWrite(ledPin[currentLED], HIGH); // turn on the current LED
currentLED += direction; // increment by the direction value
// change direction if we reach the end
if (currentLED == 9) {direction = -1;}
if (currentLED == 0) {direction = 1;}
}
```

Code Overview

Code will be very similar to project 6 code only difference is that there we have fixed that ledDelay time to 65 milliseconds (static), but here we are changing that duration with potentiometer input (dynamic).

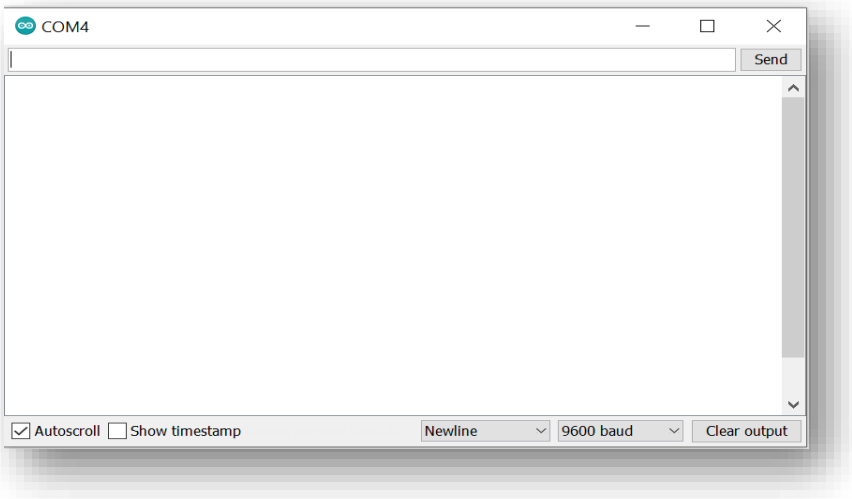
Displaying Data from the Arduino in the Serial Monitor

So far, we have sent sketches to the Arduino and used the LEDs to show us Output. Blinking LEDs make it easy to get feedback from the Arduino, but blinking lights can tell us only so much. In this section you'll learn how to use the Arduino's cable connection and the IDE's Serial Monitor window to display data from the Arduino and send data to the Arduino from the computer keyboard.

The Serial Monitor

To open the Serial Monitor, start the IDE and click the Serial Monitor icon button on the tool bar, shown in Figure. 

The Serial Monitor should open and look similar to Figure below



As you can see in Figure above, the Serial Monitor displays an input field at the top, consisting of a single row and a Send button, and an output window below it, where data from the Arduino is displayed. When the Autoscroll box is checked, the most recent output is displayed, and once the screen is full, older data rolls off the screen as newer output is received. If you uncheck Autoscroll, you can manually examine the data using a vertical scroll bar.

Starting the Serial Monitor

Before we can use the Serial Monitor, we need to activate it by adding this function to our sketch in void setup():

```
Serial.begin(9600);
```

The value 9600 is the speed at which the data will travel between the computer and the Arduino, also known as baud. This value must match the speed setting at the bottom right of the Serial Monitor, as shown in Figure above.

Sending Text to the Serial Monitor

To send text to the Serial Monitor to be displayed in the output window, you can use **Serial.print**:

```
Serial.print("Arduino for Everyone!");
```

This sends the text between the quotation marks to the Serial Monitor's output window. You can also use Serial.println to display text and then force any following text to start on the next line:

```
Serial.println("Arduino for Everyone!");
```

Displaying the Contents of Variables

You can also display the contents of variables on the Serial Monitor. For example, this would display the contents of the variable results:

```
Serial.println(results);
```

If the variable is a float, the display will default to two decimal places. You can specify the number of decimal places used as a number between 0 and 6 by entering a second parameter after the variable name. For example, to display the float variable results to four decimal places, you would enter the following:

```
Serial.print(results,4);
```

Project #9 Using serial monitor to check potentiometer input

In this project we will basically check input received by the potentiometer in project 8 and see the mapped value on our computer screen.

Hardware

Same as project 8.

The Sketch

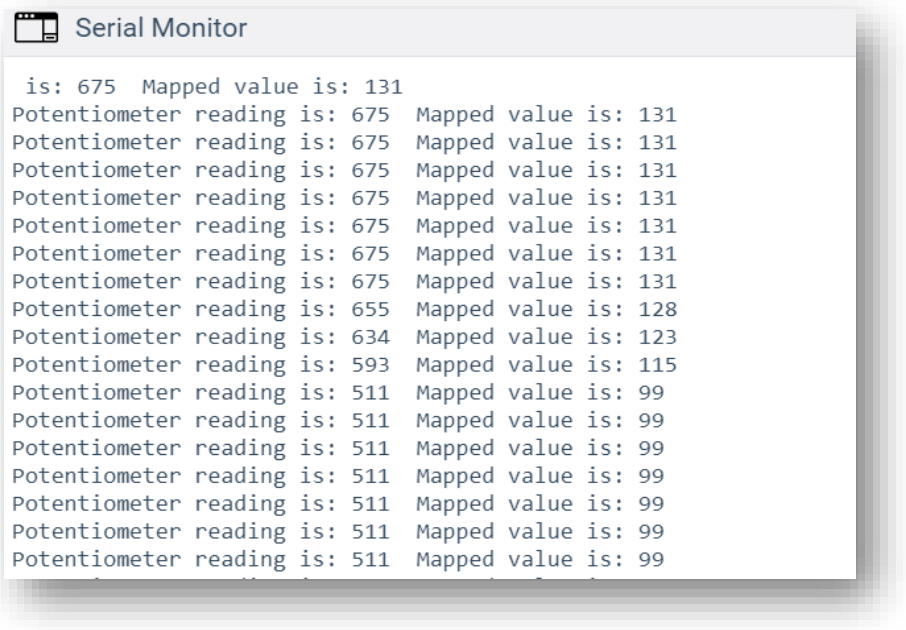
```
byte ledPin[] = {4, 5, 6, 7, 8, 9, 10, 11, 12, 13}; // Create array for LED pins
int ledDelay, value; // delay between changes
int direction = 1;
int currentLED = 0;
unsigned long changeTime;
int potPin = 2; // select the input pin for the potentiometer

void setup() {
for (int x=0; x<10; x++) { // set all pins to output
pinMode(ledPin[x], OUTPUT);
Serial.begin(9600);
}
changeTime = millis();
}
void loop() {
value = analogRead(potPin); // read the value from the potentiometer
Serial.print("Potentiometer reading is: "); // we are printing on serial monitor scre
en
Serial.print(value); // value printed on serial monitor screen
Serial.print("  ");  // Sapce printed in serial monitor screen
ledDelay = map(value, 0, 1023, 0, 200);
Serial.print("Mapped value is: ");
Serial.println(ledDelay); // Serial.println will take cursor to new line after execut
ion
if ((millis() - changeTime) > ledDelay) { // if it has been ledDelay ms since last ch
ange
changeLED();
changeTime = millis();
}
}
void changeLED() {
for (int x=0; x<10; x++) { // turn off all LED's
digitalWrite(ledPin[x], LOW);
}
```

```
digitalWrite(ledPin[currentLED], HIGH); // turn on the current LED
currentLED += direction; // increment by the direction value
// change direction if we reach the end
if (currentLED == 9) {direction = -1;}
if (currentLED == 0) {direction = 1;}
}
```

The Schematic

Same as project 8

Serial Monitor output



Sending Data from the Serial Monitor to the Arduino

Serial is used for communication between the Arduino board and a computer or other devices. All Arduino boards have at least one serial port (also known as a UART or USART): Serial. It communicates on digital pins 0 (RX) and 1 (TX) as well as with the computer via USB. Thus, if you use these functions, you cannot also use pins 0 and 1 for digital input or output.

To send data from the Serial Monitor to the Arduino, we need the Arduino to listen to the serial buffer—the part of the Arduino that receives data from the outside world via the serial pins (digital 0 and 1) that are also connected to the USB circuit and cable to your computer. The serial buffer holds incoming data from the Serial Monitor's input window.

If you want to read data from serial monitor use following command:

```
Serial.parseInt() // for reading integer values
Serial.parseFloat() // for reading float values
Serial.readString() // for reading string values
```

To know more about Serial class and methods click here.

Project #10 Multiplying a Number by 2

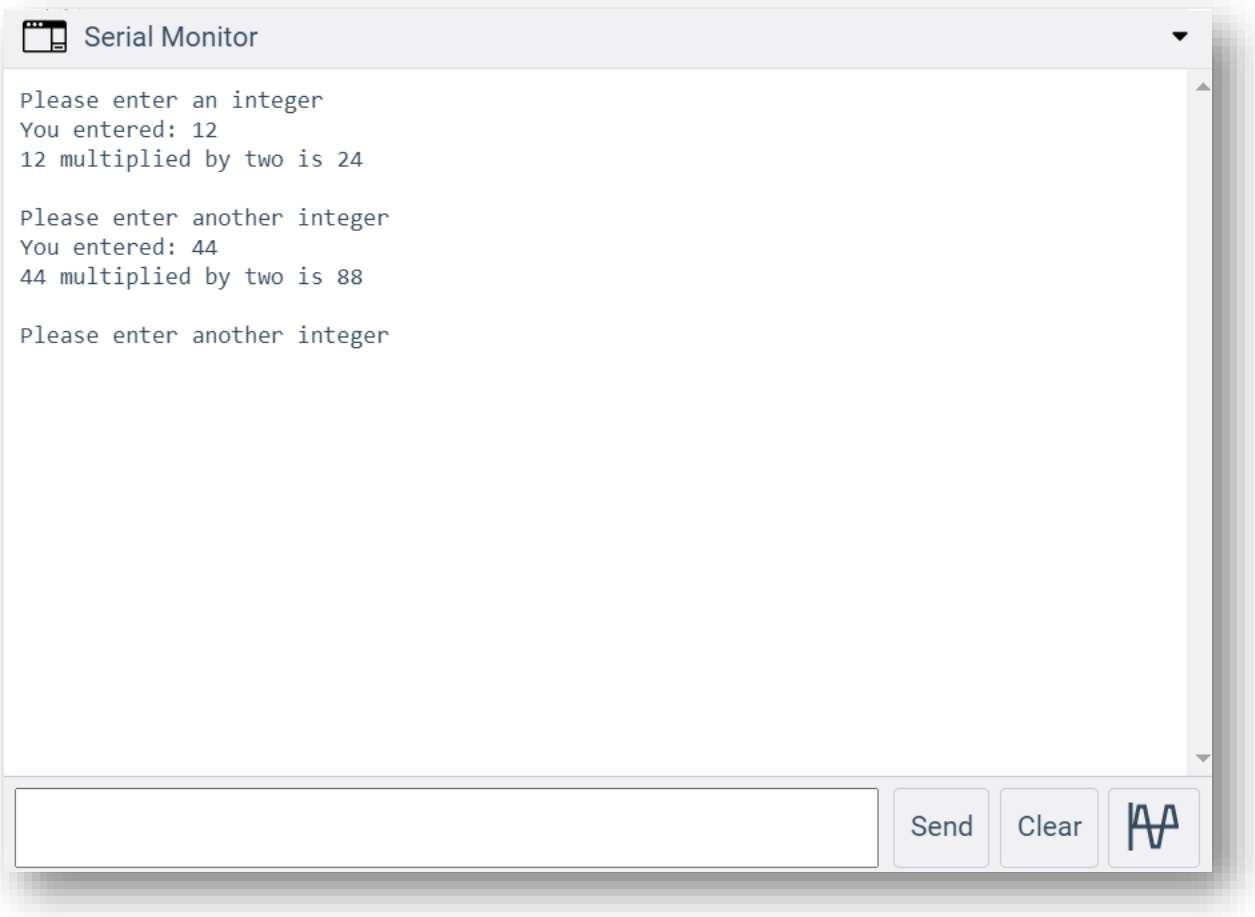
In this project we are going to take input from user via serial monitor and multiply that number by 2 and then print it on serial monitor screen.

The Sketch

```
int number;
void setup()
{
Serial.begin(9600); //We are setting up braud rate to 9600. It means that serial port
 // can handle 9600 bits/ second of data transfer.
Serial.println("Please enter an integer");
}
void loop()
{
number = 0; // zero the incoming number ready for a new read
Serial.flush(); // clear any "junk" out of the serial buffer before waiting
while (Serial.available() == 0)
{
// do nothing until something enters the serial buffer
}
while (Serial.available() > 0) // if there is something enetered by user
{
```

```
number = Serial.parseInt();
// read the number in the serial buffer,

 }
// Show me the number!
Serial.print("You entered: ");
Serial.println(number);
Serial.print(number);
Serial.print(" multiplied by two is ");
number = number * 2;
Serial.println(number);

Serial.println("\nPlease enter another integer");
}
```

Output



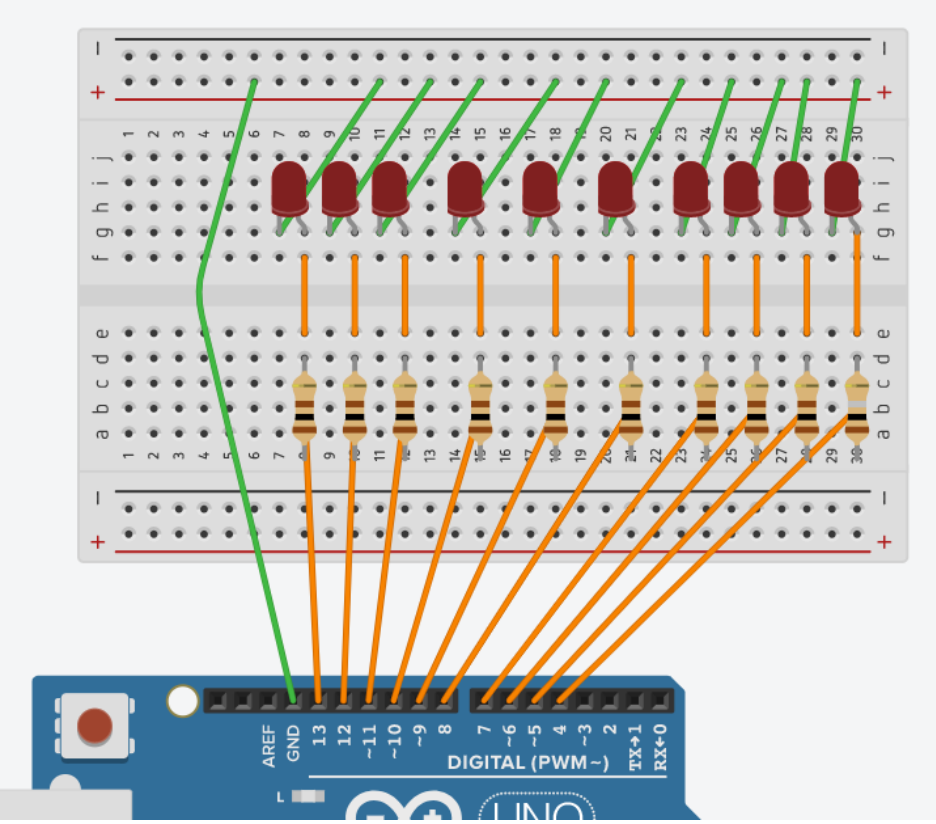
Project #11 LED chase effect with speed controlled by Serial Monitor

In this project we are going to use same circuit as we used in Project #6. Instead of fixing LED delay period, we will provide input by serial monitor and change the speed in real time.

Hardware

Same as project #6.

The Schematic



The Sketch

```
byte ledPin[] = {4, 5, 6, 7, 8, 9, 10, 11, 12, 13}; // Create array for LED pins
int ledDelay = 1000; // delay between changes
int direction = 1;
int currentLED = 0;
```

```
unsigned long changeTime;
void setup() {
    for (int x=0; x<10; x++) { // set all pins to output
        pinMode(ledPin[x], OUTPUT);
    }
    changeTime = millis();
    Serial.begin(9600); // setting data transfer rate to 9600 bits per second
}

void loop() {

Serial.flush(); // clear any "junk" out of the serial buffer before waiting


while (Serial.available() > 0) // if there is some input from user saved in buffer
// Serial.available() will be > 0
{
  ledDelay = Serial.parseInt(); // we are now reading the input from serial monitor

}

if ((millis() - changeTime) > ledDelay) { // if it has been ledDelay ms since last ch
ange
    changeLED();
    changeTime = millis();
    Serial.println(ledDelay);  }

}

void changeLED() {
for (int x=0; x<10; x++) { // turn off all LED's
digitalWrite(ledPin[x], LOW);
}
digitalWrite(ledPin[currentLED], HIGH); // turn on the current LED
currentLED += direction; // increment by the direction value
// change direction if we reach the end
if (currentLED == 9) {direction = -1;}
if (currentLED == 0) {direction = 1;}
}
```

Code Overview

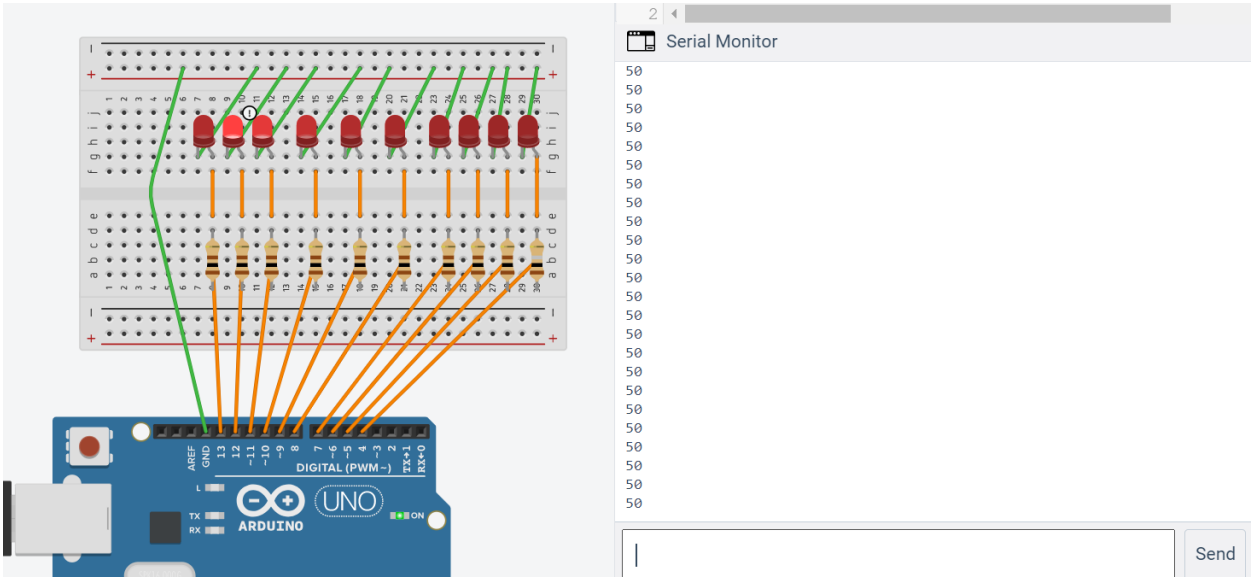
Everything will be same as code overview of project #6 except here we are providing LED Delay time through serial port and changing speed of chase effect in real time.

```
while (Serial.available() > 0) // if there is some input from user saved in buffer
// Serial.available() will be > 0
{
  ledDelay = Serial.parseInt(); // we are now reading the input from serial monitor

}
```

In this part of code, we are checking if there is something available in Serial buffer and if found true, we are reading that integer value and storing it in ledDelay variable.

By doing this, we can change the value of ledDelay after each loop completion thus changing speed of the chase effect.

Output



RGB LED

How do RGB LEDs work?

With an RGB LED you can produce almost any color. How is this possible with just one single LED?

An RGB LED is a combination of 3 LEDs in just one package:

1x Red LED

1x Green LED

1x Blue LED

You can produce almost any color by combining those three colors. An RGB LED is shown in the following figure:

How to create different colors?

With an RGB LED you can, of course, produce red, green, and blue light, and by configuring the intensity of each LED, you can produce other colors as well.

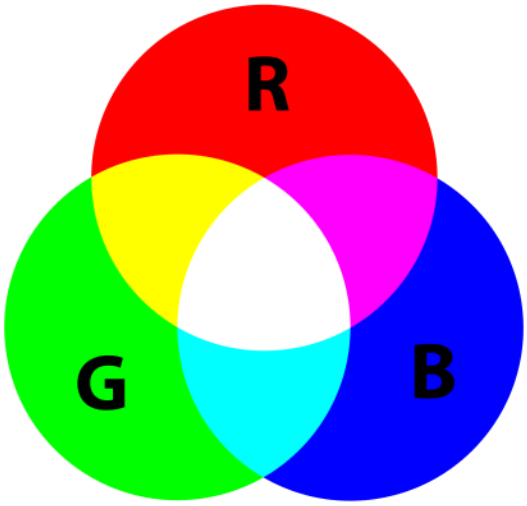
For example, to produce purely blue light, you'd set the blue LED to the highest intensity and the green and red LEDs to the lowest intensity. For a white light, you'd set all three LEDs to the highest intensity.

Mixing colors

To produce other colors, you can combine the three colors in different intensities. To adjust the intensity of each LED you can use a PWM signal.

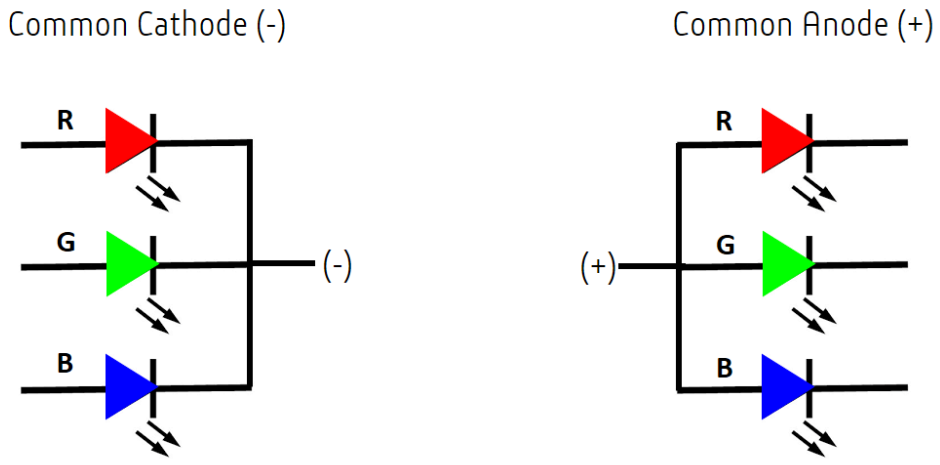
Because the LEDs are very close to each other, our eyes see the result of the combination of colors, rather than the three colors individually.

To have an idea on how to combine the colors, take a look at the following chart. This is the simplest color mixing chart, but gives you an idea how it works and how to produce different colors.



Common Anode and Common Cathode RGB LEDs

There are two kinds of RGB LEDs: common anode LED and common cathode LED. The figure below illustrates a common anode and a common cathode LED.

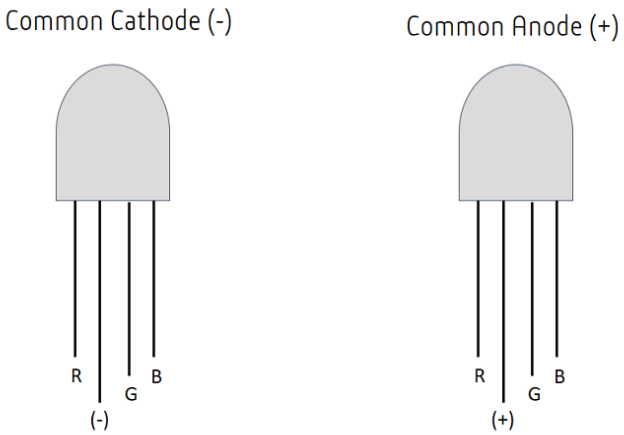


In a common cathode RGB LED, all three LEDs share a negative connection (cathode). In a common anode RGB LED, the three LEDs share a positive connection (anode).

This results in an LED that has 4 pins, one for each LED, and one common cathode or one common anode.

RGB LED Pins

RGB LEDs have four leads—one for each LED and another for the common anode or cathode. You can identify each lead by its length, as shown in the following figure.



With the LED facing you so the anode or cathode (the longest lead) is second from the left, the leads should be in the following order: red, anode or cathode, green, and blue.

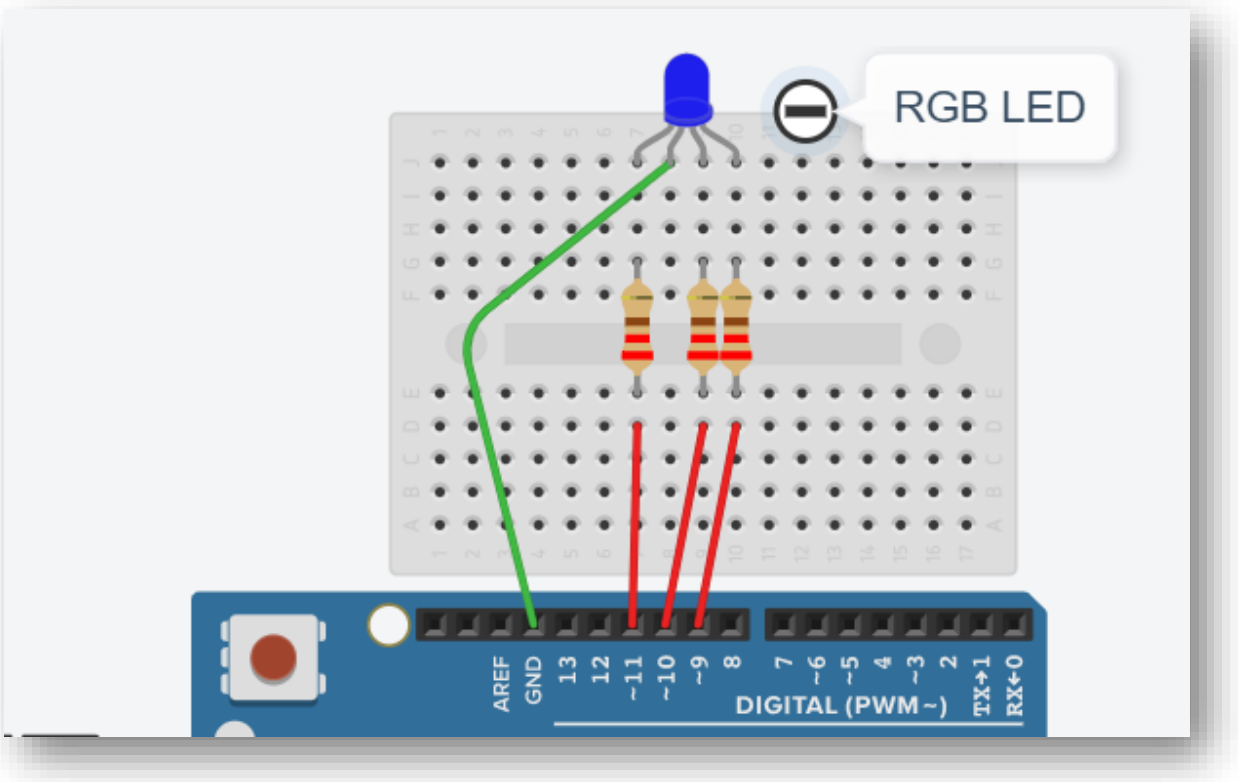
Project #12 Using RBG LED to get different colours

In this project we are going to use RGB LED to get different colour combinations. By changing intensity of RED, BLUE & GREEN lights, we can obtain a range of colours.

Hardware

- Arduino UNO
- Resistor from 150 ohm to 1k ohm
- RGB LED
- Breadboard

The Schematic



The Sketch

```
int red_light_pin= 11; //We are connecting red light pin in RBG LED to digital pin nu
mber 11
int green_light_pin = 10; // Green light pin to digital pin number 10
int blue_light_pin = 9; // Blue light pin to digital pin number 9
void setup() {
  pinMode(red_light_pin, OUTPUT); //Setting pin 9, 10 & 11 as output.
  pinMode(green_light_pin, OUTPUT);
  pinMode(blue_light_pin, OUTPUT);
}
void loop() {

// RGB_color is user defined function in which we are taking 3 ardument as input
// i.e. Intensity of red light, green light and blue light. Intensity goes from 0 to
255
// Where 0 being least and 255 being max
  RGB_color(255, 0, 0); // Red.. So here we are saying to analog write red pin value
to 255
  // and other pins to 0. So, that will cause the RBG LED to give out only red light
  delay(1000);
```

```
  RGB_color(0, 0, 255); // Green
  delay(1000); // here we are giving a delay of 1 second before colour change
  RGB_color(0, 255, 0); // Blue
  delay(1000);
  RGB_color(255, 125, 255); // Raspberry
  delay(1000);
  RGB_color(0, 255, 255); // Cyan
  delay(1000);
  RGB_color(255, 255, 0); // Magenta
  delay(1000);
  RGB_color(255, 0, 255); // Yellow
  delay(1000);
  RGB_color(255, 255, 255); // White
  delay(1000);
}
void RGB_color(int red_light_value, int green_light_value, int blue_light_value)
 { // user defined function where we are taking intensity of red, green & blue pins a
s input
 // and then analogwriting each pin with provided intensity to find different colour
combinations
  analogWrite(red_light_pin, red_light_value);
  analogWrite(green_light_pin, green_light_value);
  analogWrite(blue_light_pin, blue_light_value);
}
```

Code Overview

Project #13 Using random function to generate random colour through RBG LED

In this project, we are going to use random() function to generate random colours from RBG LED.

Random Function

```
random()
```

Description

The random function generates pseudo-random numbers.

Syntax

random(max)

random(min, max)

Parameters

min: lower bound of the random value, inclusive (optional).

max: upper bound of the random value, exclusive.

Returns

A random number between min and max-1. Data type: long.

Creating truly random numbers in Arduino is harder than you might think. The closest we can get in Arduino, and just about anywhere else, is using pseudo random numbers. That is, numbers that mimic randomness, but in fact do have a pattern if analyzed for a long enough period.

WHY ARE RANDOM NUMBERS WITH ARDUINO ALL THE SAME?

The most important thing to understand when using the **random()** function with Arduino is that it will generate the exact same list of pseudo random numbers every time.

The easy way to overcome this is using the **randomSeed()** function.

This function takes a value (an integer for example), and uses the number to alter the random list generated by the random() function. The number you pass to the randomSeed() function is called a 'seed'.

But again

randomSeed(42);

and then random() will create exact same sequence every time.

Or

randomSeed(10);

random(); will generate exactly same random number every time.

To counter this problem we have to make sure that we put random inputs in randomSeed() function. So it will then generate random values.

ONE SOLUTION TO THE RANDOM PROBLEM

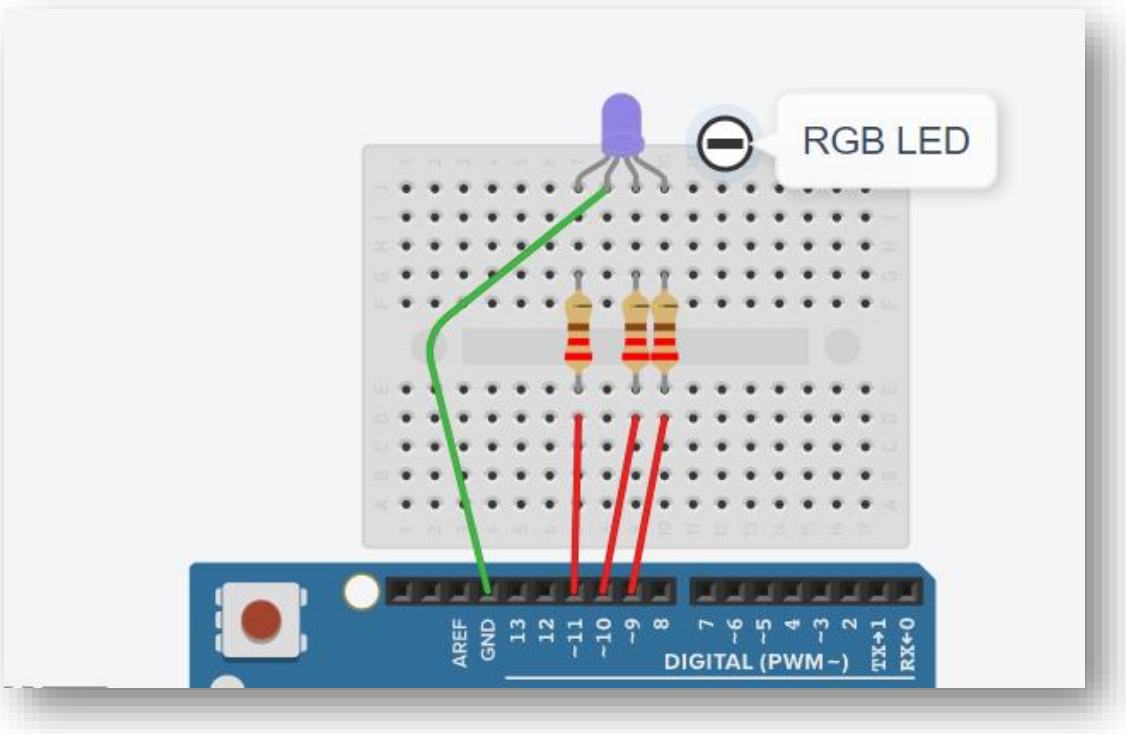
Use the analogRead() function to read a value from an unused analog pin. Since an unused pin that has no reference voltage attached to it is basically 'floating', it will return a "noise" value. This noise value can seed the randomSeed() function to produce differing sequences of random numbers every time the sketch is run.

Check out sketch below to understand random number generation properly.

Hardware

Same as project #12.

The Schematic



The Sketch

```
int red_light_pin= 11; // Same as project 12
int green_light_pin = 10;
int blue_light_pin = 9;
int red, green, blue;
void setup() {
  pinMode(red_light_pin, OUTPUT);
  pinMode(green_light_pin, OUTPUT);
  pinMode(blue_light_pin, OUTPUT);
  randomSeed(analogRead(A0)); // we are using noise value of unused analog pin as see
d
  // noise value will be random, so that will generate random numbers for us everytim
e
}

void loop() {
    RGB_color(random(0,255), random(0,255), random(0,255)); // we are sending random
intensities of
    // red, green and blue in RGB LED. This will generate random colours everytime.
```

```
    delay(50); // I am taking delay of 50 milliseconds, you can change this vakue and
 experiment with it.
}

void RGB_color(int red_light_value, int green_light_value, int blue_light_value)
 {
  analogWrite(red_light_pin, red_light_value);
  analogWrite(green_light_pin, green_light_value);
  analogWrite(blue_light_pin, blue_light_value);
}
```

Code Overview

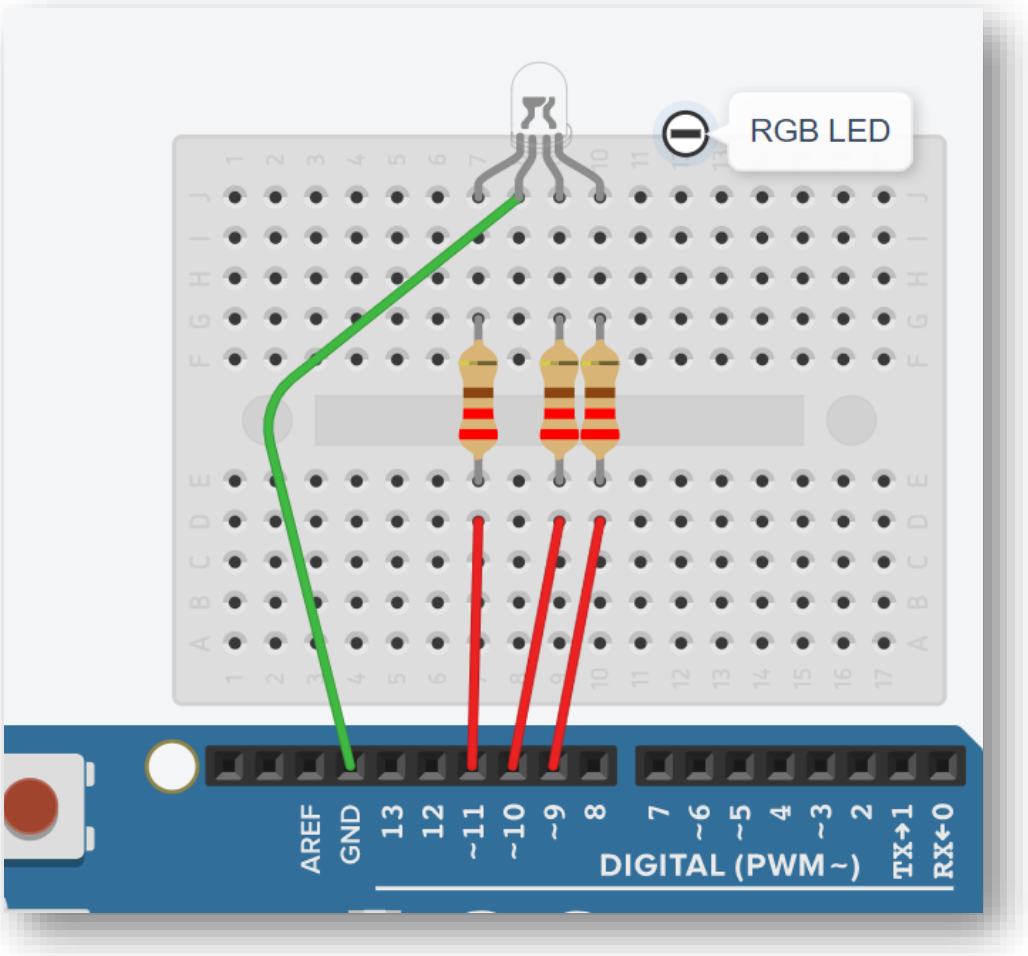
Project #14 Using Serial monitor for obtaining different colour combinations

In this project we are going to give input intensities for different colours through serial monitor to obtain colour combinations in RBG LED.

Hardware

Same as project #12.

Schematic



The Sketch

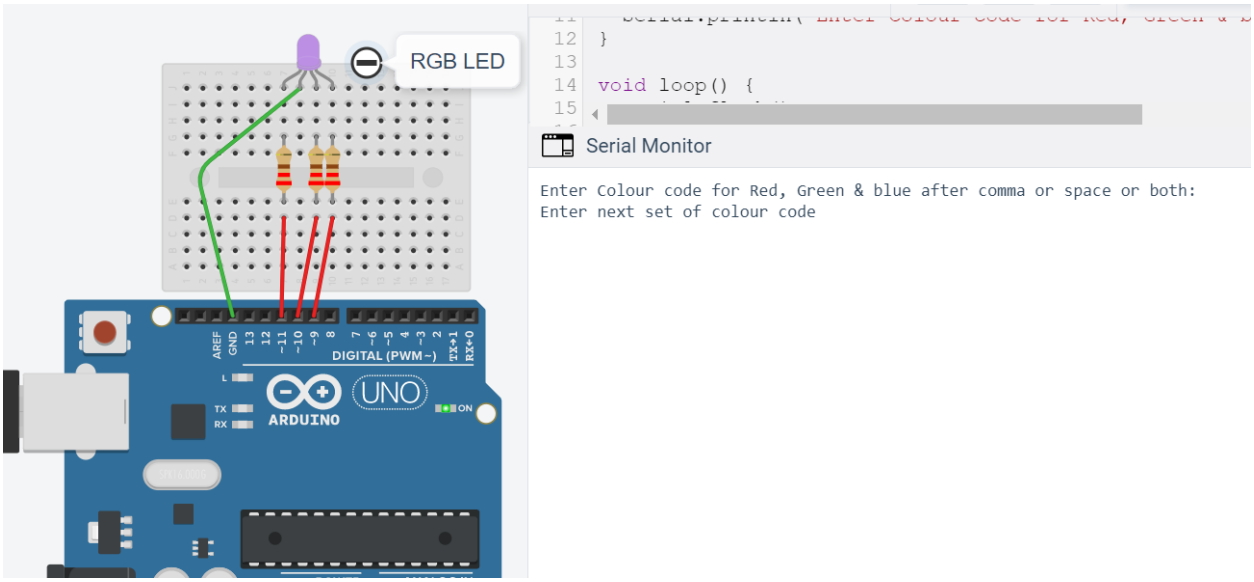
```
int red_light_pin= 11;
int green_light_pin = 10;
int blue_light_pin = 9;
int red, green, blue;
void setup() {
  pinMode(red_light_pin, OUTPUT);
  pinMode(green_light_pin, OUTPUT);
  pinMode(blue_light_pin, OUTPUT);
  Serial.begin(9600); // Starting serial communication at 9600 bits per second rate.
  delay(3000);
  Serial.println("Enter Colour code for Red, Green & blue after comma or space or bot
h: ");
  // Printing this after 3 seconds of giving power to arduino

}

void loop() {
  Serial.flush(); // clearing any junk out of serial buffer for a new read

  while (Serial.available() == 0){ //
    // do nothing
    // it will check for user's input, if not given this loop will keep on iterating
without doing anything
    //RGB_color(red, blue, green);
  }
    while (Serial.available() > 0){ // once the input is provided by user, control wi
ll be moved to this part arduino
    // here we are reading input given by user seperated by comma or space into red,
green and blue variable
      red = Serial.parseInt(); // here we are reading 1st integer in buffer and stori
ng it in red variable
      green = Serial.parseInt(); // then we are reading 2nd integer in buffer provide
d by user and storing it in variable named green
      blue = Serial.parseInt(); // the last will be saved in variable named blue
      // if user will provide any more input after 3rd one... again Serial.available(
) will be true and that input will be saved in
      // variable named red replacing old assignment

      Serial.println("Enter next set of colour code "); // after taking the input, we
 are printing for next set of color code.
```

```
      // untill provided, Serial.available() will be false and control won't come in
this while loop.
    }

  RGB_color(red, blue, green); // after getting inputs from user, send the intensity
values to user defined function and light-up the
  // RBG as wanted by user


}

void RGB_color(int red_light_value, int green_light_value, int blue_light_value)
 {
  analogWrite(red_light_pin, red_light_value);
  analogWrite(green_light_pin, green_light_value);
  analogWrite(blue_light_pin, blue_light_value);
}
```

Output

It will first ask for set of 3 numbers separated by comma, space or both.

After providing and pressing on send, it will send the input in serial buffer where Arduino will read it and change intensities of the LED's.


RGB LED
}
void loop() {
Serial Monitor
Enter Colour code for Red, Green & blue after comma or space or both:
Enter next set of colour code
DIGITAL (PWM~)
ARDUINO
UNO

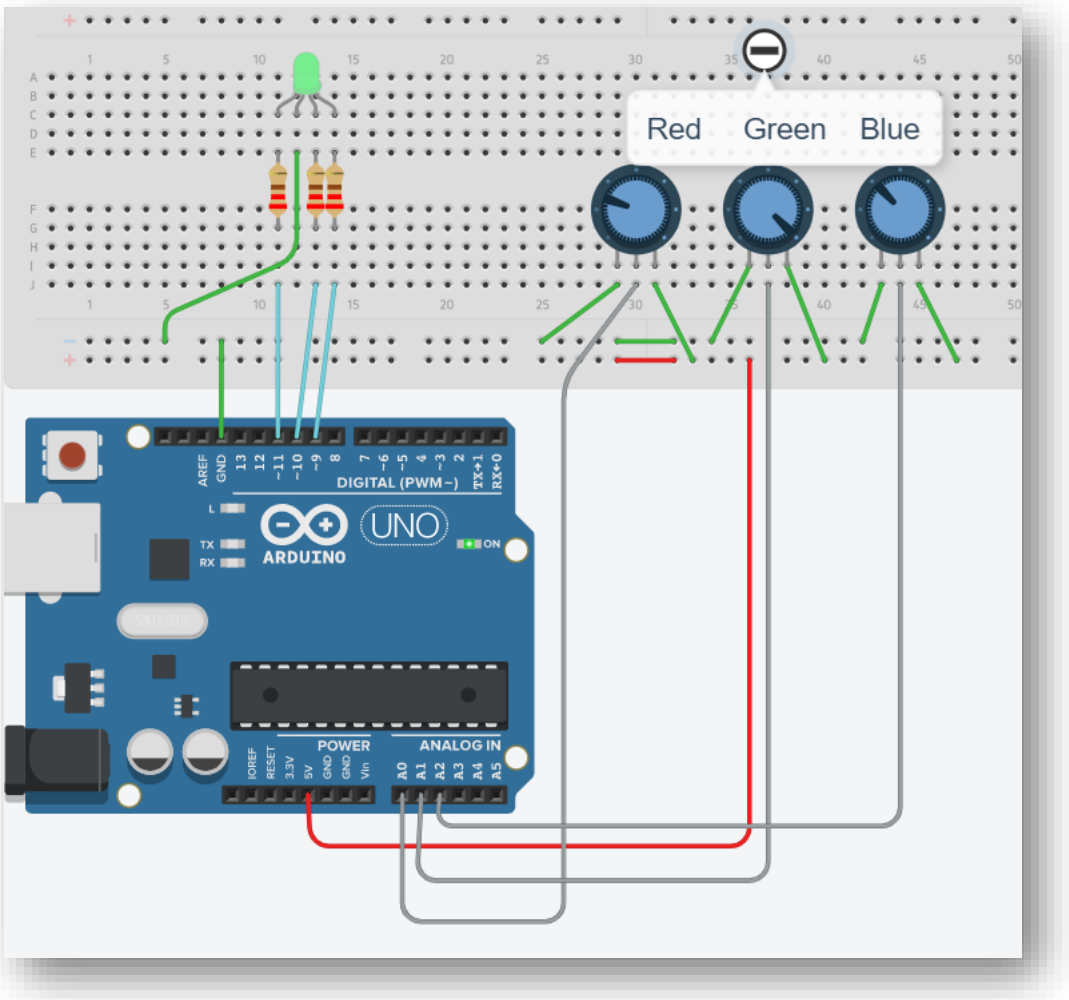
Project #15 Controlling RBG colour by 3 potentiometer

The project we'll build uses three potentiometers to control the light intensity of each pin (LED) of the RGB LED to produce any color you want.

Hardware

- Arduino UNO
- RGB LED common anode or RGB LED common cathode
- 3 x Potentiometer
- Breadboard
- 3 × 220 Ω Resistor
- Jumper wires

The Schematic



The Sketch

```
int redPin = 11;     // Red RGB pin -> D11
int greenPin = 9;   // Green RGB pin -> D9
int bluePin = 10;    // Blue RGB pin -> D10

int potRed = A0;     // Potentiometer controls Red pin -> A0
int potGreen = A1;   // Potentiometer controls Green pin -> A1
int potBlue = A2;    // Potentiometer controls Blue pin -> A2

void setup() {
  pinMode(redPin,OUTPUT);
  pinMode(bluePin,OUTPUT);
  pinMode(greenPin, OUTPUT);

  pinMode(potRed, INPUT);
  pinMode(potGreen, INPUT);
  pinMode(potBlue, INPUT);
}

void loop() {
  // Reads the current position of the potentiometer and converts
  // to a value between 0 and 255 to control the according RGB pin with PWM
  analogWrite(redPin, map(analogRead(potRed), 0,1023, 0, 255));
  analogWrite(greenPin,map(analogRead(potGreen), 0,1023, 0, 255));
  analogWrite(bluePin, map(analogRead(potBlue), 0,1023, 0, 255));

  delay(10);
}
```

Code Overview

Project #16 Using TONE() function with Arduino

Do you need to make some noise with Arduino? Maybe a simple tone for an alarm, maybe a beep to alert you when a specific input threshold is met, or maybe to play the Super Mario Brothers soundtrack to entertain your juvenile mind.

Whatever your audible need, you will likely find the easiest, quickest and possibly the cheapest way to make some noise is using the tone() function and piezo speaker with your Arduino.

A QUICK INTRO TO PIEZO SPEAKERS (AKA PIEZO BUZZERS)

Ahh, noise….

Birds make it, kids make it – it can be music to our ears or pure torture.

We are going to use a piezo buzzer to make some noise with Arduino.

A piezo buzzer is pretty sweet. It's not like a regular speaker that you might think of. It uses a material that's piezoelectric, it actually changes shape when you apply electricity to it. By adhering a piezo-electric disc to a thin metal plate, and then applying electricity, we can bend the metal back and forth, which in turn creates noise.

The faster you bend the material, the higher the pitch of the noise that's produced. This rate is called frequency. Again, the higher the frequency, the higher the pitch of the noise we hear.

So basically, by shocking the plate over and over really fast, we can make noise.

Hardware

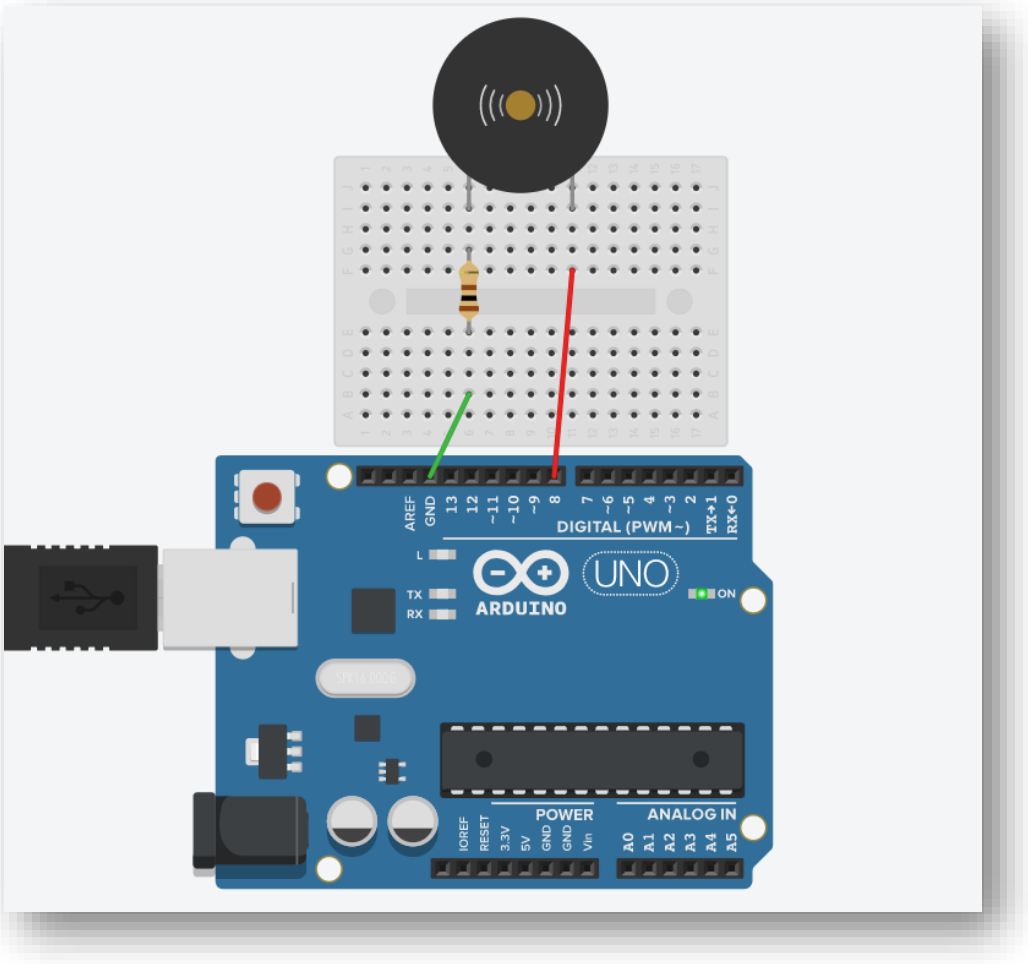
- Arduino board
- Solderless breadboard
- Jumper wires
- 100 ohm resistor
- Piezo speaker (aka piezo buzzer)

The Schematic

It's painfully easy to set up a simple piezo speaker circuit with an Arduino.

1. Place the piezo buzzer into the breadboard, so that the two leads are on two separate rows.
2. Using jumper wires, connect the positive lead to Arduino digital pin 8. The case of the buzzer may have a positive sign (+) on it to indicate the positive lead (if not, then the red wire usually indicates the positive lead).
3. Connect the other lead to the 100 ohm resistor, and then to ground.

That's it.



The Sketch

The tone() function works with two arguments, but can take up to three arguments. Let's address the two required items first:

```
tone( pin number, frequency in hertz);
```

The pin number that you will use on the Arduino.

The frequency specified in hertz. Hertz are cycles per second.

The frequency is an unsigned integer and can take a value up to 65,535 – but if you are trying to make tones for the human ear, then values between 2,000 and 5,000 are where our ears are most tuned.

Here is a simple sketch demonstrating the tone() function:

```
//A sketch to demonstrate the tone() function

//Specify digital pin on the Arduino that the positive lead of piezo buzzer is attach
ed.
int piezoPin = 8;

void setup() {

}//close setup

void loop() {

  /*Tone needs 2 arguments, but can take three
    1) Pin#
    2) Frequency - this is in hertz (cycles per second) which determines the pitch of
 the noise made
    3) Duration - how long teh tone plays
  */
  tone(piezoPin, 1000, 500);

  //tone(piezoPin, 1000, 500);
  //delay(1000);

}
```

As an experiment, try changing the second argument in tone() to 100, 1000, 10000, 650000 and listen to the effect it has on the audio signal.

You will notice that the higher the number, the higher the pitch that is created.

HOW TO SEPARATE THE NOISE – AKA MAKE A BEAT

What if we want to add some space between the noise, so we can get a beat?

We can try adding a delay(1000) after the tone(), but if you test this out, you will find it doesn't get you anywhere.

This is because the tone() function uses one of the built in timers on the Arduino's micro-contoller. tone() works independently of the delay() function. You can start a tone and do other stuff – while the tone is playing in the background.

But, back to that question, how can we separate the noise a little?

Let's talk about that third parameter we can pass to the tone() function – it is the duration of the tone in milliseconds.

```
tone( pin number, frequency in hertz, duration in milliseconds);
```

If you want to generate distinct beats, and you want to do this with the delay() function, then you need to keep in mind what we just said, that the tone() function uses one of the built in timers on the Arduino board.

Therefore, if you use 500 milliseconds as the third argument in tone(), and follow that by a delay of 1000 milliseconds, you will only be creating a "quiet time" of 500 milliseconds.

```
//This code only generates a delay of 500 milliseconds between the tone

  tone(8, 2000, 500);

  delay(1000);
```

```
//This code generates a delay of 1000 milliseconds between the tone

  tone( 8, 2000, 500);

  delay(1500);
```

Instead of the time being added together – as in 500 millisecond of noise, and then 1000 milliseconds of delay – the delay and the tone start at about the exact same time, so we get 500 millisecond of tone separated by 500 milliseconds of quiet.

500 milliseconds of noise and 1000 milliseconds of quiet in 2nd case.

It's a little quirky to wrap your mind around, but if you play around with it a little, you'll pick up on the intricacies.

Final code for beep sound will be:

```
int piezoPin = 8;

void setup() {

}//close setup

void loop() {

  tone(piezoPin, 1000, 500);
  delay(1000);

}
```

We can also write the above code as:

```
int piezoPin = 8;

void setup() {

}//close setup

void loop() {

  tone(piezoPin, 1000);
  delay(500);

  noTone(piezoPin); /* noTone function is turn off the sound, we only need to provide
  pin number here */
  delay(500)

}
```

THE LIMITS YOU SHOULD KNOW WHEN USING TONE()

Like everything else in the world, the tone() function has some limitations of which you should be aware. Let's touch on them here:

1. You can't use tone() while also using analogWrite() on pins 3 or 11. If you do – you get some whacky results – neither will work like you expect. That's because the tone() function uses the same built in timer that analogWrite() does for pins 3 and 11. It's worth trying just hear the weird noises.

2. You cannot generate a tone lower than 31 HZ. You can pass values 31 and less to the tone() function, but it doesn't mean you will get a good representation of it.

3. The tone() function cannot be used by two separate pins at the same time. Let's say you have two separate piezo speakers, each on a different pin. You can't have them both play at the same time. One has to be on, and then the other. Furthermore, before you can have the other pin use the tone() function, you must call the noTone() function and "turn off" the tone from the previous pin.

Additional Links

1. Super Mario tone
2. tone() in the Arduino reference

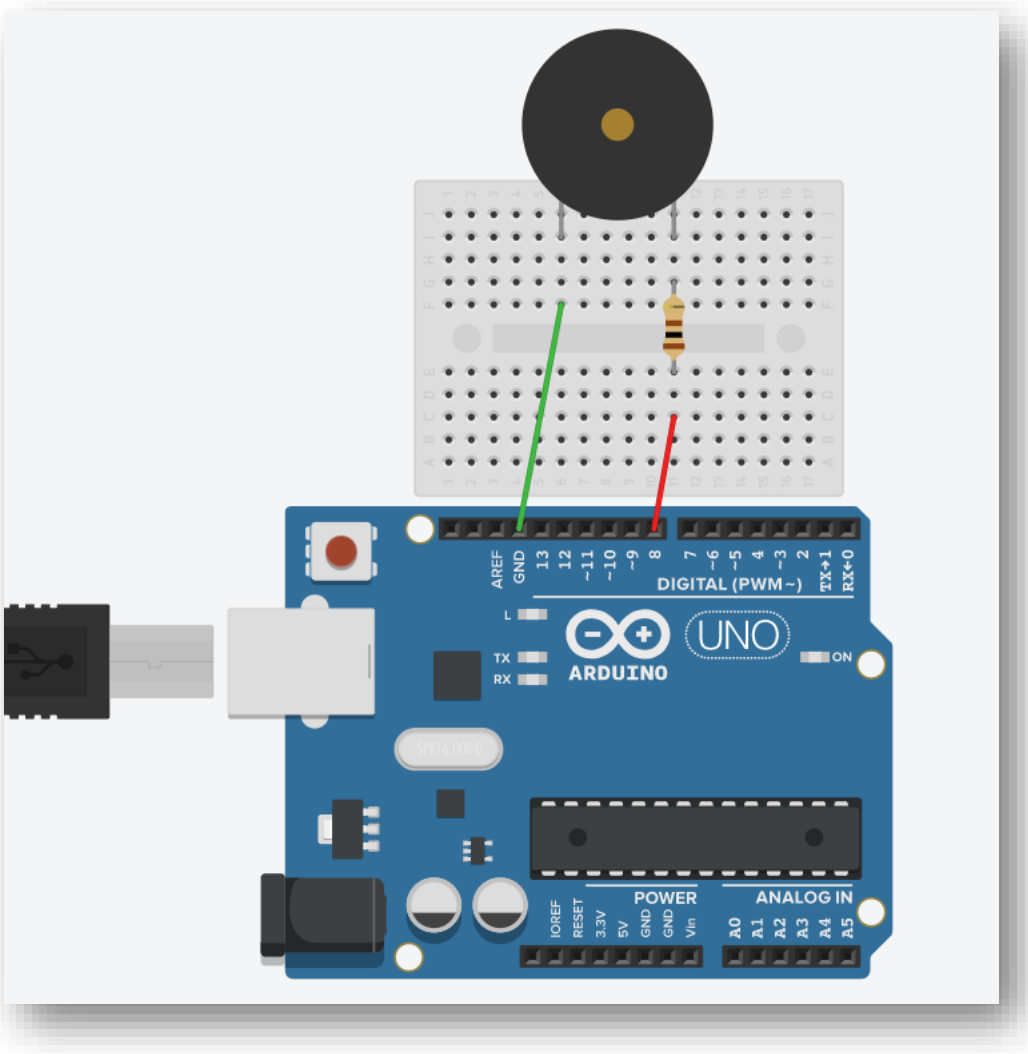
Project #17 Piezo Sounder Alarm

By connecting a piezo sounder to a digital output pin, we are going to create a wailing alarm sound.

Hardware

Same as project #16.

The Schematic



The Sketch

```
float sinVal;
int toneVal;
void setup() {
pinMode(8, OUTPUT);
}

void loop() {
for (int x=0; x<180; x++) {
// convert degrees to radians then obtain sin value
sinVal = (sin(x*(3.1412/180)));
// generate a frequency from the sin value
toneVal = 2000+(int(sinVal*1000));
tone(8, toneVal);
delay(2);
}
}
```

Code Overview

First we set up two variables

```
float sinVal;
int toneVal;
```

The sinVal float variable will hold the sine value that will cause the tone to rise and fall.

The toneVal variable will be used to take the value in sinVal and convert it to a frequency we require.

In the setup function, we now set digital pin 8 to an output.

```
void setup() {
pinMode(8, OUTPUT);
}
```

Then we move onto the main loop. We set a for loop to run from 0 to 179 to ensure that the sine value does not go into the negative.

```
for (int x=0; x<180; x++) {
```

We convert the value of x into radians:

```
sinVal = (sin(x*(3.1412/180)));
```

Then that value is converted into a frequency suitable for the alarm sound.

```
toneVal = 2000+(int(sinVal*1000));
```

We take 2,000 and add the sinVal multiplied by 1,000. This gives us a good range of frequencies for the rising and falling tone to go between as the sine wave rises and falls.

Next, we use the tone() command to generate the frequency at the piezo sounder.

```
tone(8, toneVal);
```

The tone() command requires either two or three parameters, thus:

```
tone(pin, frequency)
tone(pin, frequency, duration)
```

The pin is the digital pin being used to output to the piezo and the frequency is the frequency of the tone in hertz. There is also the optional duration parameter in milliseconds for the length of the tone. If no duration is specified, the tone will keep on playing until you play a different tone or you use the noTone(pin) command to cease the tone generation on the specified pin.

Finally, we run a delay of 2 milliseconds in-between the frequency changes to ensure the sine wave rises and falls at the speed we require.

```
delay(2);
```

You may be wondering why we did not put the 2 milliseconds into the duration parameter of the tone()

command, like this:

```
tone(8, toneVal, 2);
```

This is because our for loop is so short that it will change the frequency in less than 2 milliseconds anyway, thus rendering the duration parameter useless. Therefore, a delay of 2 milliseconds is put in after the tone is generated to ensure it plays for at least 2 milliseconds before the for loop repeats and the tone changes again.

You could use this alarm generation principle later on when you learn how to connect sensors to your Arduino.

Then, you could activate an alarm when a sensor threshold has been reached, for example, if someone gets too close to an ultrasonic detector or if a temperature gets too high.

If you change the values of 2,000 and 1,000 in the toneVal calculation and the length of the delay, you can generate different alarm sounds. Experiment a little, and see what sounds you can make.

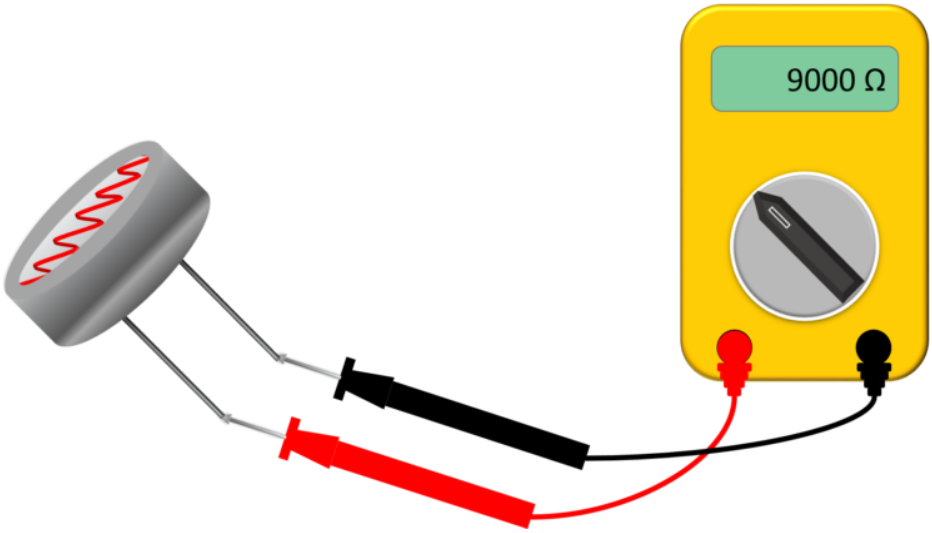
Project #18 LDR – The Light Sensor

What is an LDR?

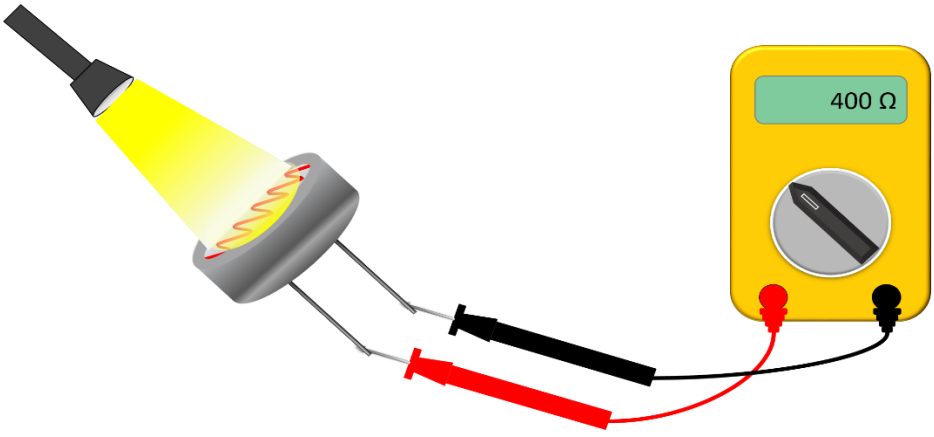
An LDR is a photoelectric device. LDR stands for Light Dependent Resistor also known popularly as photo-resistor.

Working Principle of LDR

The semiconductor material used to make an LDR is such that in the absence of light, the resistance of material increases as shown in the figure below.



The resistance decreases as light strikes on it, as shown in the below figure. Providing power supply, allows the current to pass through the circuit.



You may have question in your mind that why there is a change in the resistance due to the absence or presence of light. The answer is – in a normal state, semiconductors have few free electrons. When light strikes on it, the free electrons gain enough energy to move through the material. Hence, the resistance of a semiconductor material varies.

Reading the above lines, you may be thinking that from where this energy comes. How simply light is giving energy?

Well, the answer to your above queries is 'photo effect'. Albert Einstein discovered that light striking on material surface is not only in form of waves. There must be tiny particles accompanying the waves. For this discovery in the year 1905, Einstein awarded noble prize. These particles are now popularly known as 'Photons'.

The energy of a single photon is given by,

$E=hf$

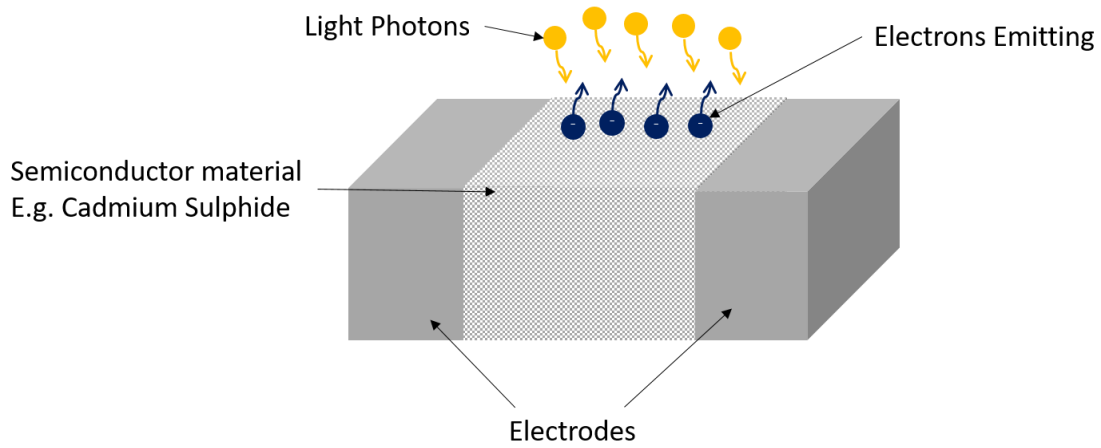
Where,

E = energy of photon

f = frequency of light

h = Planck's constant

When a photon strikes on the material surface part of its energy is used to release an electron from material surface and remaining is used to impart kinetic energy to the electron.

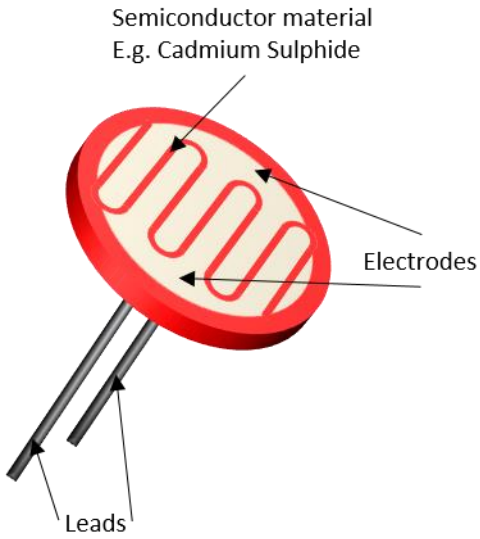


Construction of an LDR

The energy of a photon is very small. Hence, we must have a material that can work with such a small amount of energy. Few materials have such properties.

Two of them are Cadmium sulfide and Cadmium selenide. Both this material's resistance changes as light are incident on it. Most commonly known material to make an LDR is Cadmium Sulfide.

Highly purified Cadmium sulfide in powder form mixed with a binding agent. Further, it is compressed and heated. The mixture becomes like a lump. It is then sandwiched between two electrodes in serpentine shape or zigzag shape as shown in the below figure. This disc is then either encapsulated in transparent resin or encased in glass in order to protect cadmium sulfide from contamination by the atmosphere.



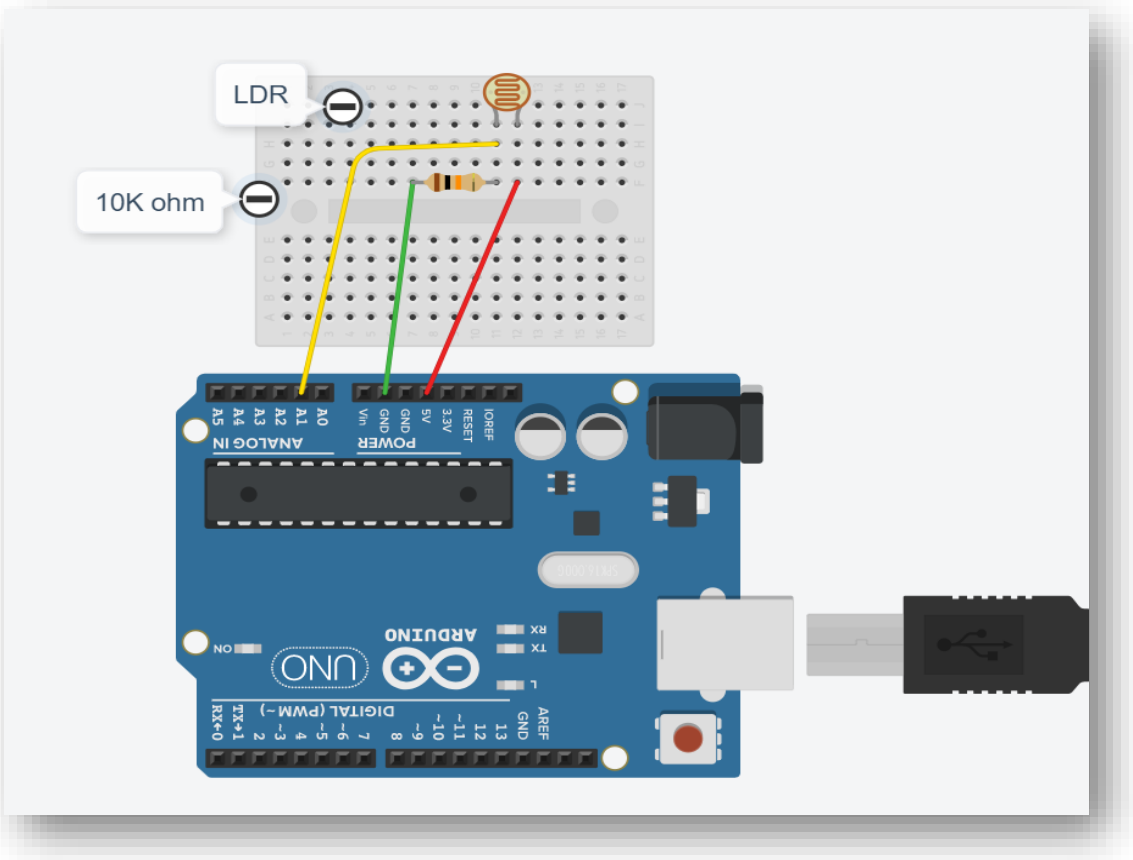
Hardware

Arduino UNO
Breadboard
Jumper Wires
Resistor

In the project below let's try to analyze and understand how to connect LDR and how to get input data from it.

Guess how we take input in volts and convert them into digital numbers in order to use them in our codes.
Well we got you all covered, click here to navigate.

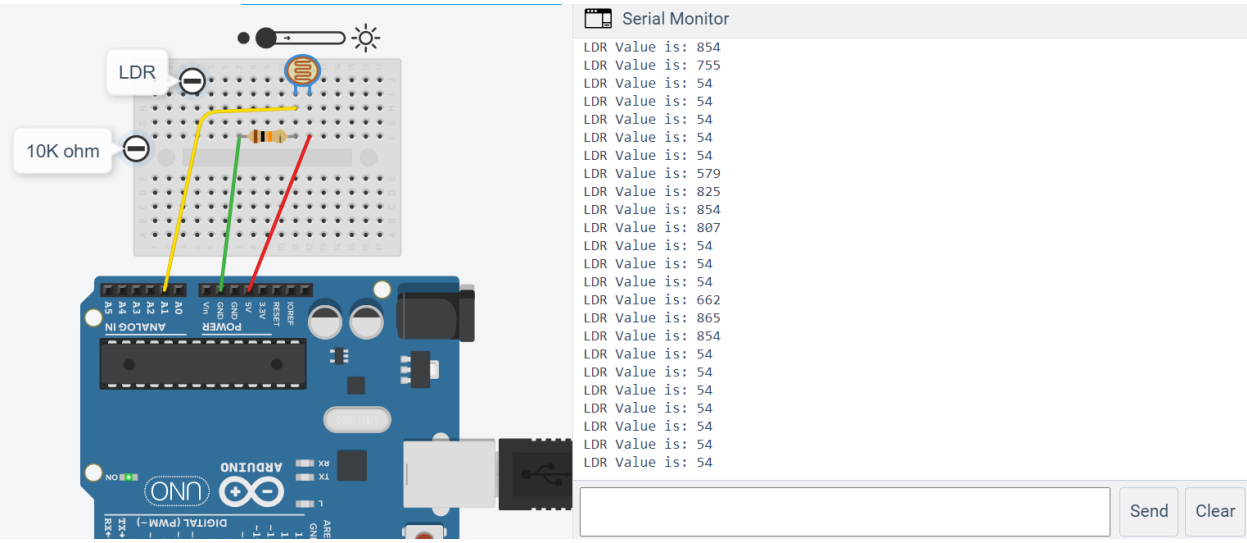
The Schematic



The Sketch

```
const int LDR = A1; // we are fixing our LDR pin for taking analog input
int input_val = 0;

void setup()
{
  Serial.begin(9600); // starting serial monitor at 9600 bits per second
  pinMode(LDR, INPUT); // optional
}

void loop()
{
  input_val = analogRead(LDR); /* reading the LDR input and converting from
(0-5)volts to (0-1023) through ADC */
  Serial.print("LDR Value is: ");
  Serial.println(input_val);
  delay(100);
}
```

Output



When we will expose the LDR to light, resistance will decrease and that will result in higher reading from the analog pins. Opposite will happen in darkness and reading will come down drastically as we reduce brightness.

Note: If during connection, you put positive in place of negative and vice versa, the input reading will be opposite to current one i.e. more when dark and less when bright.

Don't worry, you can adjust your codes to work with this connection too.

We will see how in coming project.

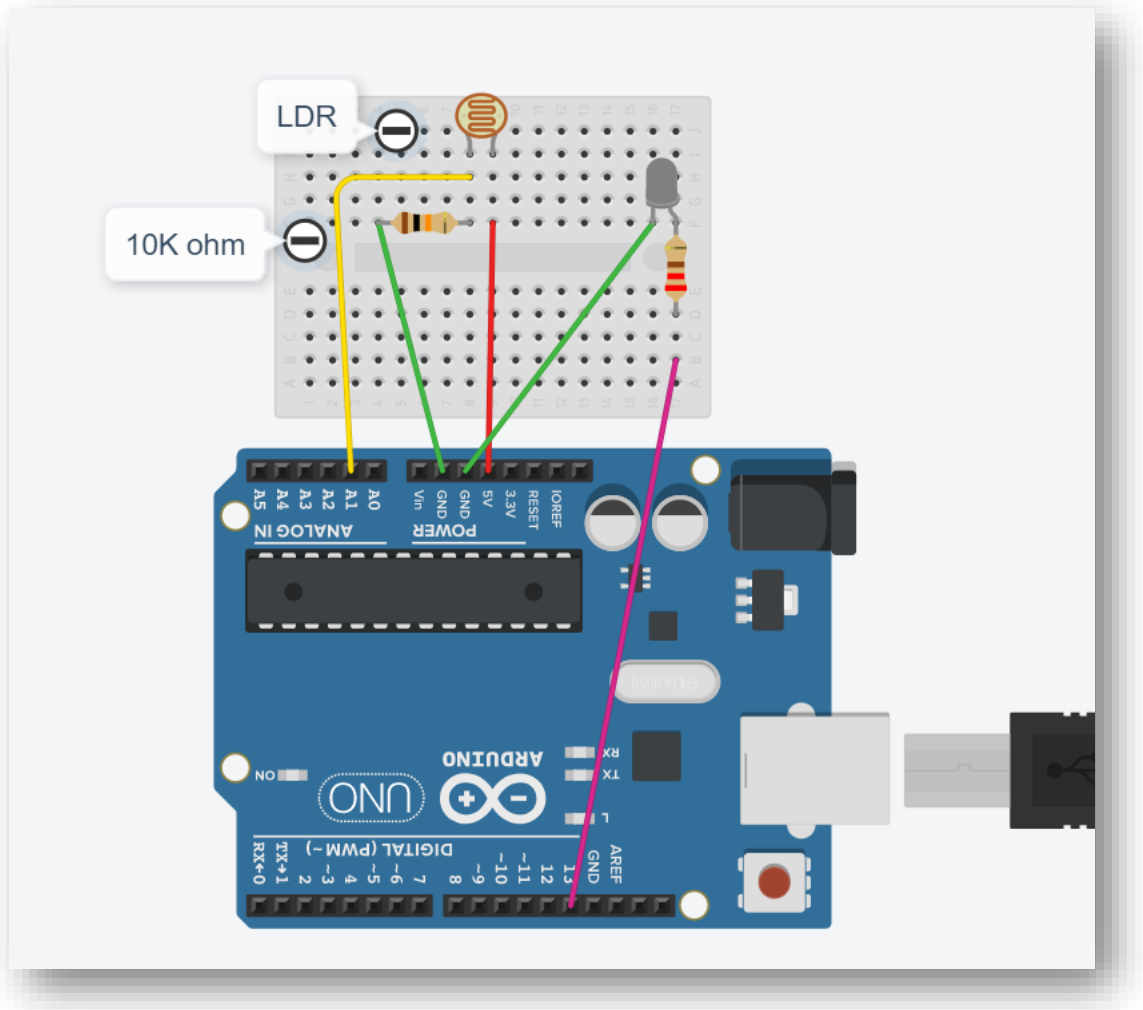
Project #19 Light automation using LDR

In this project we are going to use LDR to detect light coming in our room and then turn LED ON/OFF according to light's intensity.

Hardware

Arduino UNO
LDR
10K ohm & 220 ohm resistor
Breadboard
LED

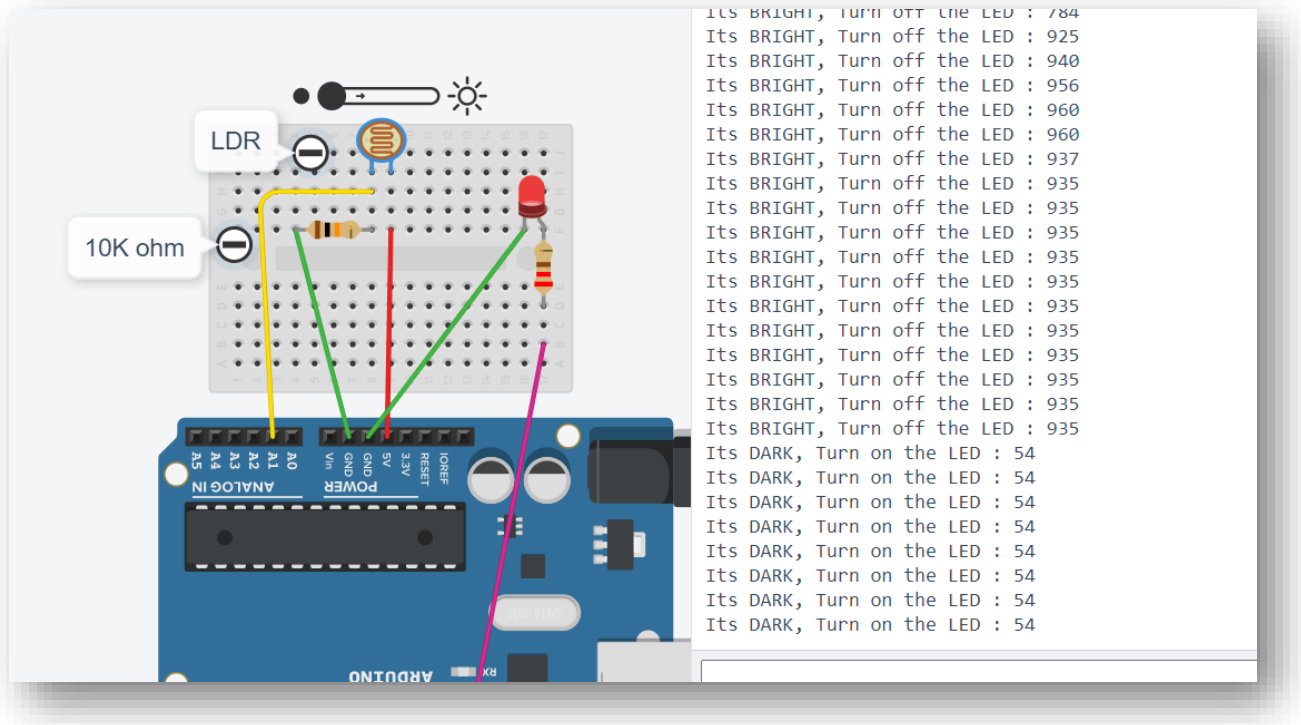
The Schematic



The Sketch

```
const int ledPin = 13;
const int ldrPin = A1;

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

void loop() {
int ldrStatus = analogRead(ldrPin); // we are reading input from LDR

if (ldrStatus <= 900) { // if the value is less then our fixed value, its dark outsid
e then
// Take fixed value after expermenting in your room, check which value should be opti
mal for your requirements

digitalWrite(ledPin, HIGH);
Serial.print("Its DARK, Turn on the LED : ");
Serial.println(ldrStatus);
}

else (ldrStatus > 900) {
digitalWrite(ledPin, LOW);
Serial.print("Its BRIGHT, Turn off the LED : ");
Serial.println(ldrStatus);
}
}
```

Output


LDR
10K ohm
ANALOG IN
POWER
ARDUINO
Its BRIGHT, Turn off the LED : 784
Its BRIGHT, Turn off the LED : 925
Its BRIGHT, Turn off the LED : 940
Its BRIGHT, Turn off the LED : 956
Its BRIGHT, Turn off the LED : 960
Its BRIGHT, Turn off the LED : 960
Its BRIGHT, Turn off the LED : 937
Its BRIGHT, Turn off the LED : 935
Its BRIGHT, Turn off the LED : 935
Its BRIGHT, Turn off the LED : 935
Its BRIGHT, Turn off the LED : 935
Its BRIGHT, Turn off the LED : 935
Its BRIGHT, Turn off the LED : 935
Its BRIGHT, Turn off the LED : 935
Its BRIGHT, Turn off the LED : 935
Its BRIGHT, Turn off the LED : 935
Its BRIGHT, Turn off the LED : 935
Its DARK, Turn on the LED : 54
Its DARK, Turn on the LED : 54
Its DARK, Turn on the LED : 54
Its DARK, Turn on the LED : 54
Its DARK, Turn on the LED : 54
Its DARK, Turn on the LED : 54
Its DARK, Turn on the LED : 54
Its DARK, Turn on the LED : 54

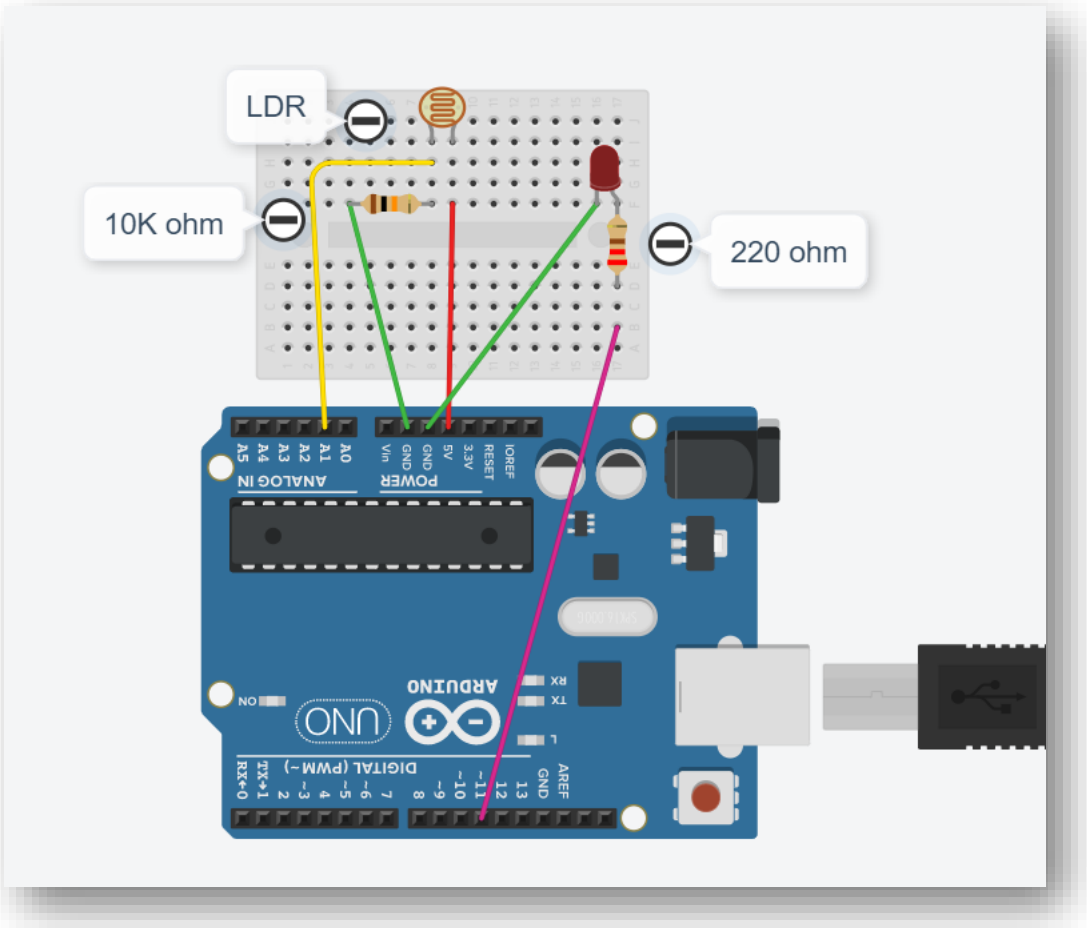
Project #20 Varying intensity of LED as outside light intensity

In this project we are going to vary intensity of room light on LDR which will eventually change intensity of LED. When room light will be more, LED will be dim and decrease in room light, LED's intensity will increase uniformly.

Hardware

Same as Project #19.

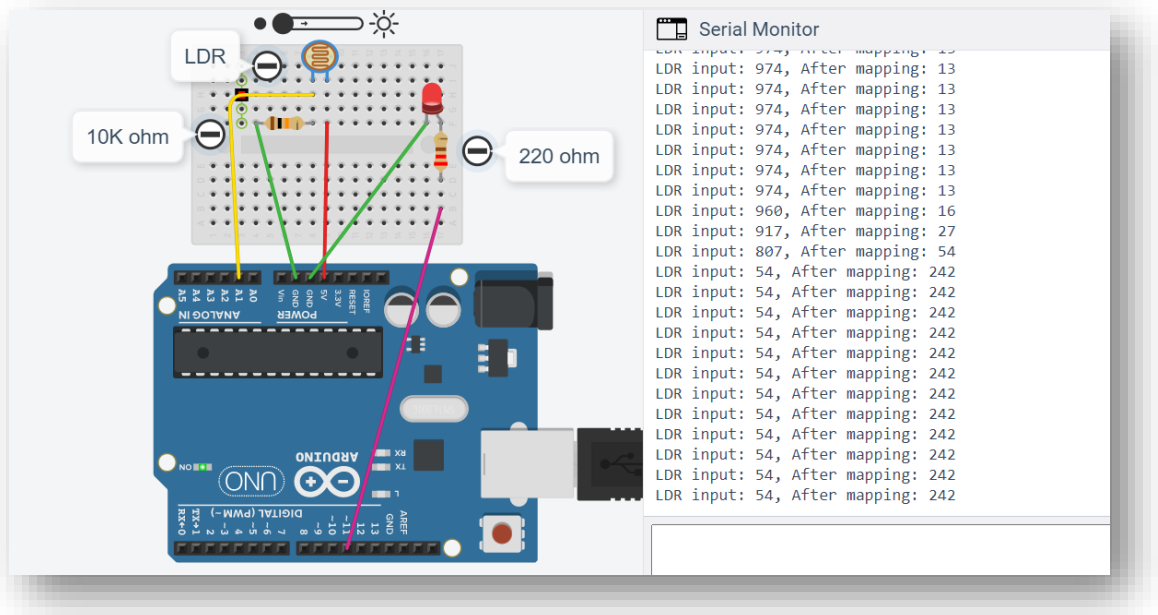
The Schematic



The Sketch

```
const int ledPin = 11;
const int ldrPin = A1;
int ldrStatus, LED_intensity; // one for LDR input and other for LED intensity

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

void loop() {
ldrStatus = analogRead(ldrPin); // we are reading LDR pin

LED_intensity = map(ldrStatus, 0, 1023, 255, 0); // mapping from (0-
1023) to (255 - 0) uniformly

Serial.print("LDR input: ");
Serial.print(ldrStatus);
Serial.print(", After mapping: ");
Serial.println(LED_intensity);

analogWrite(ledPin, LED_intensity);
delay(100);
}
```

Output



Project #21 Ultrasonic Distance sensor

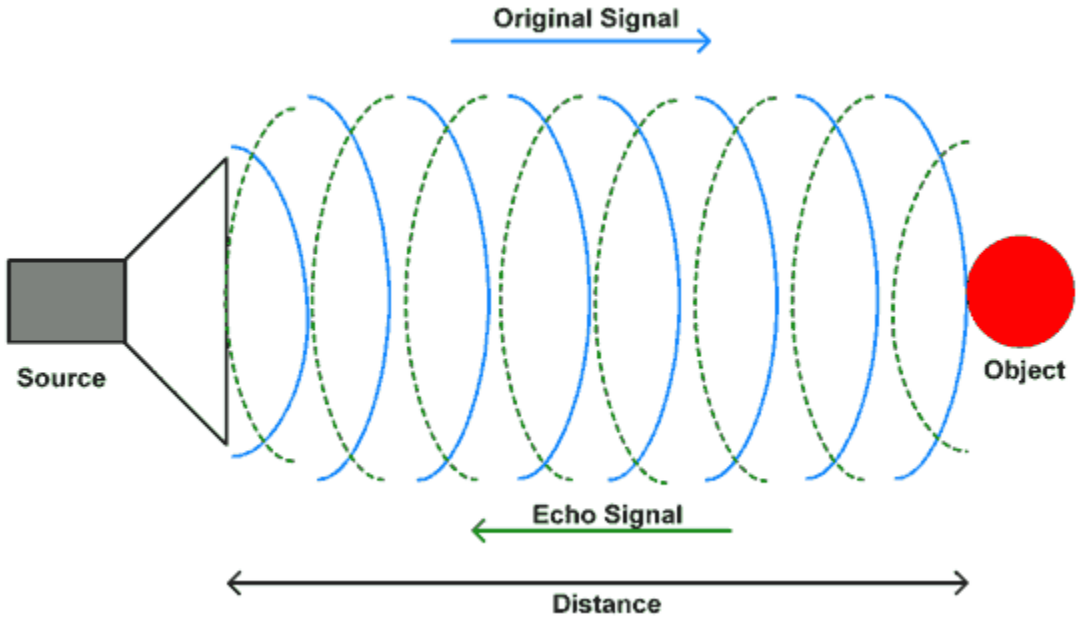
In this project we are going to understand output given by Ultrasonic distance sensor and understand possibilities where we can use this sensor in real life use.

What is Ultrasonic Sensor & how it works?

The ultrasonic sensor works on the principle of SONAR and RADAR system which is used to determine the distance to an object.

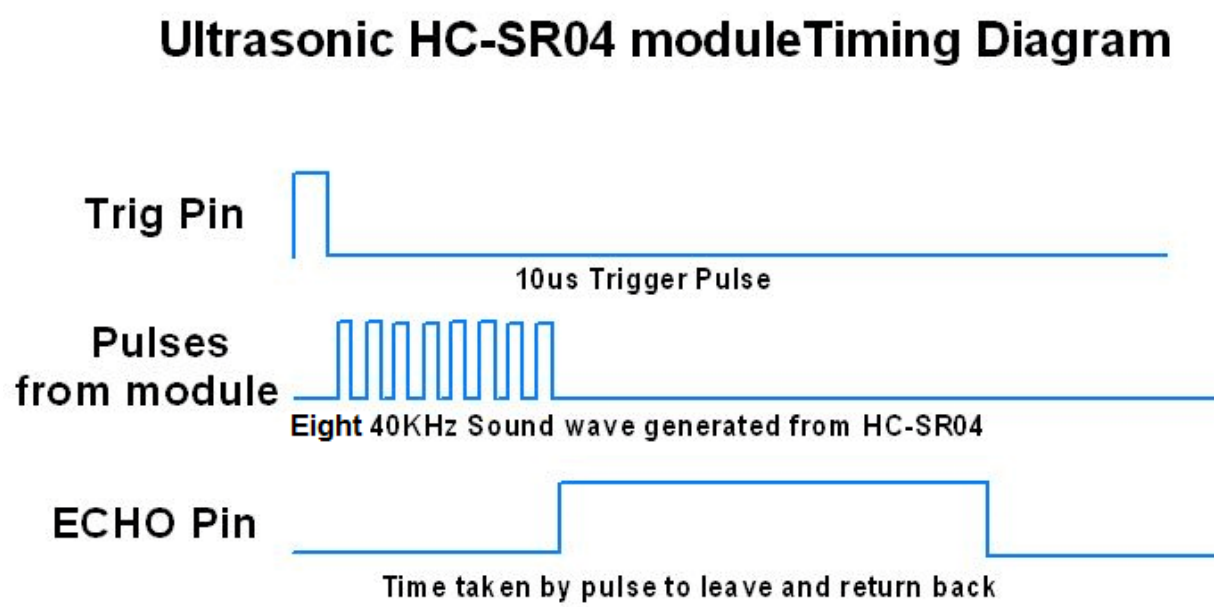
It all starts, when a pulse of at least 10 µS (10 microseconds) in duration is applied to the Trigger pin. In response to that the sensor transmits a sonic burst of eight pulses at 40 KHz. This 8-pulse pattern makes the "ultrasonic signature" from the device unique, allowing the receiver to differentiate the transmitted pattern from the ambient ultrasonic noise.

The eight ultrasonic pulses travel through the air away from the transmitter. Meanwhile the Echo pin goes HIGH to start forming the beginning of the echo-back signal.

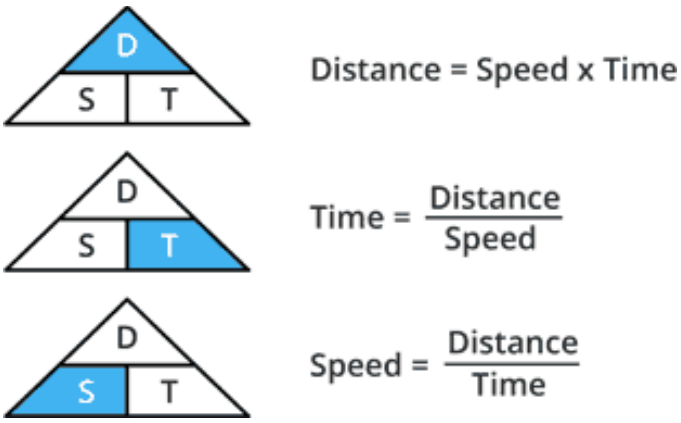


In case, If those pulses are not reflected back then the Echo signal will timeout after 38 mS (38 milliseconds) and return low. Thus a 38 mS pulse indicates no obstruction within the range of the sensor.

If those pulses are reflected back the Echo pin goes low as soon as the signal is received. This produces a pulse whose width varies between 150 µS to 25 mS, depending upon the time it took for the signal to be received.



The width of the received pulse is then used to calculate the distance to the reflected object. This can be worked out using simple distance-speed-time equation, we learned in High school. In case you forgot, an easy way to remember the distance, speed and time equations is to put the letters into a triangle.



Let's take an example to make it clearer. Suppose we have an object in front of the sensor at an unknown distance and we received a pulse of width 500 µS on the Echo pin. Now let's calculate how far the object from the sensor is. We will use the below equation.

Distance = Speed x Time

Here, we have the value of Time i.e. 500 µs and we know the speed. What speed do we have? The speed of sound, of course! Its 340 m/s. We have to convert the speed of sound into cm/µs in order to calculate the distance. A quick Google search for "speed of sound in centimeters per

microsecond" will say that it is 0.034 cm/µs. You could do the math, but searching it is easier. Anyway, with that information, we can calculate the distance!

Distance = 0.034 cm/µs x 500 µs

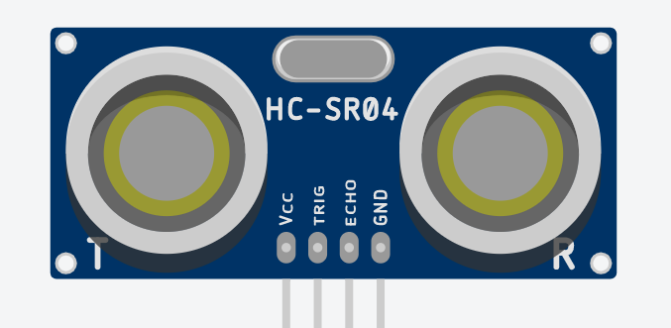
But this is not done! Remember that the pulse indicates the time it took for the signal to be sent out and reflected back so to get the distance so, you'll need to divide your result in half.

Distance = (0.034 cm/µs x 500 µs) / 2

Distance = 8.5 cm

So, now we know that the object is 8.5 centimeters away from the sensor.

HC-SR-04 Ultrasonic Module



HC-SR-04 has an ultrasonic transmitter, receiver and control circuit.

In ultrasonic module HCSR04, we have to give trigger pulse, so that it will generate ultrasound of frequency 40 kHz. After generating ultrasound i.e. 8 pulses of 40 kHz, it makes echo pin high. Echo pin remains high until it does not get the echo sound back. So the width of echo pin will be the time for sound to travel to the object and return back. Once we get the time we can calculate distance, as we know the speed of sound.

HC-SR04 can measure up to range from 2 cm - 400 cm.

HC-SR04 Pin Description

VCC >> +5 V supply

TRIG >> Trigger input of sensor. Microcontroller applies 10 us trigger pulse to the HC-SR04 ultrasonic module.

ECHO >> Echo output of sensor. Microcontroller reads/monitors this pin to detect the obstacle or to find the distance.

GND >> Ground

Author: printf("Electrovert Labs");

Hardware

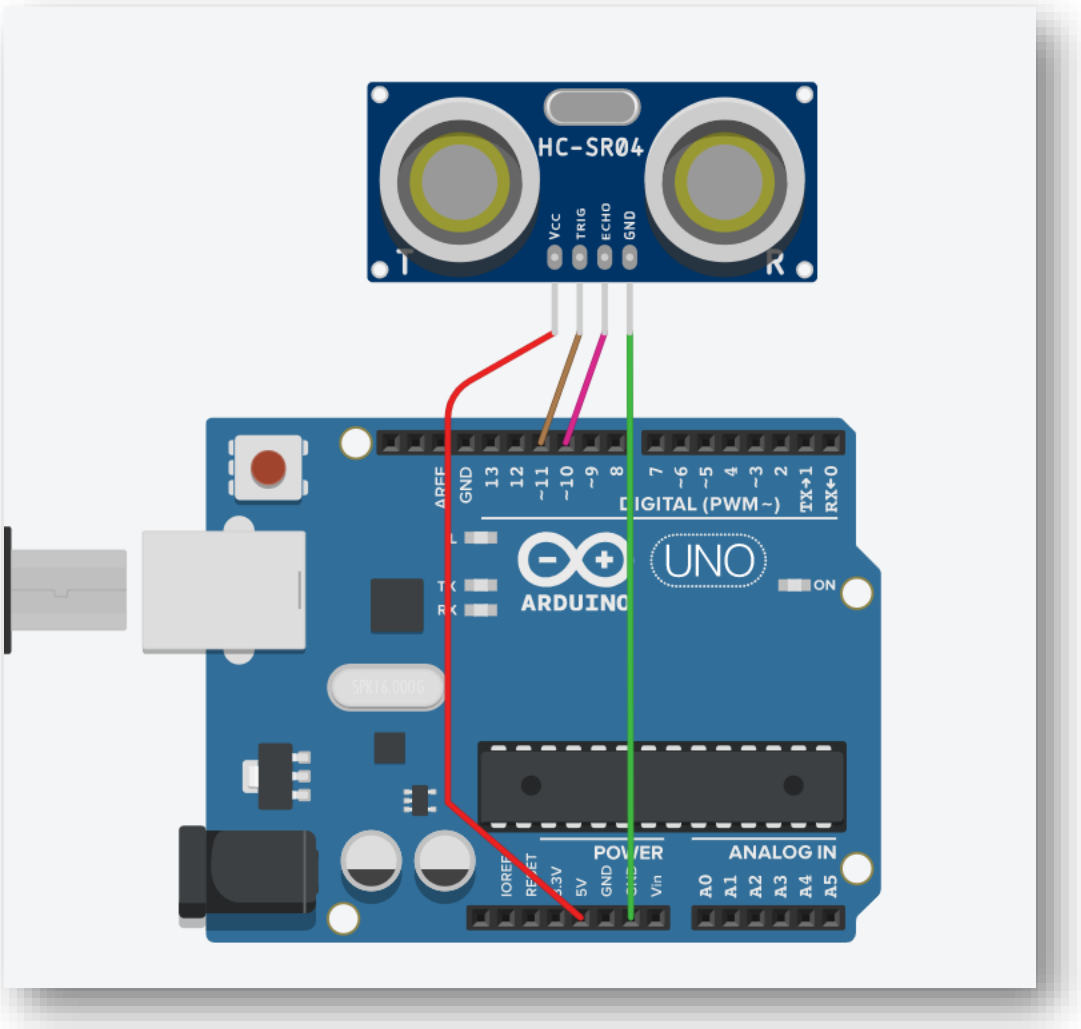
Arduino UNO

Breadboard

Jumper Wires

Ultrasonic Sensor (HC-SR04)

The Schematic

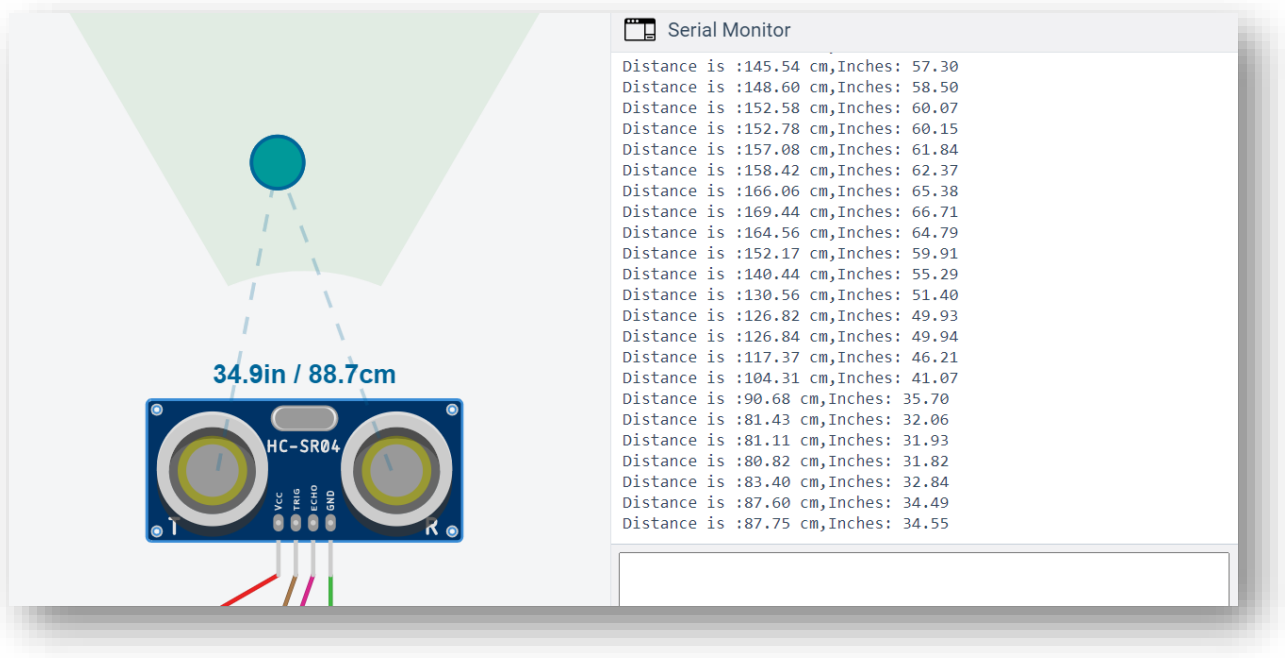


The Sketch

```
#define trigPin 11 //by defining trigPin, we cannot change its value later
#define echoPin 10 // it will be constant throughout

void setup() {
Serial.begin (9600);
pinMode(trigPin, OUTPUT); /* setting trigger pin to output so that it can send signal
s */
pinMode(echoPin, INPUT); // setting echo pin as input so that it can receive signals
}

float measure_distance(int trig, int echo); /* declaring the user-
defined function that we are
Going to use later in this program (This step is not necessary in Compiler based lang
uages) */


void loop(){

float dist_cm = measure_distance(trigPin, echoPin); // calling out user-
defined function

Serial.print("Distance is :");
Serial.print(dist_cm);
Serial.print(" cm,");
Serial.print("Inches: ");
Serial.println(0.393701 * dist_cm); // converting cm to inch

delay(200);
}


float measure_distance(int trig, int echo) // defining user defied function
{
  float distance, duration;

    digitalWrite(trig, LOW); // keeping triger pin to low for 2 milliseconds
    delayMicroseconds(2);
    digitalWrite(trig, HIGH); /* making it high for 10 milliseconds, so that
     it send send out batch of 8 pulses */
    delayMicroseconds(10);

    digitalWrite(trig, LOW); // then finally setting trigger pin to low


    duration = pulseIn(echo, HIGH); /* after pulse being sent, turning on echo
    pin and measuring time taken for the signal to bounce back */
```

```
    distance = (duration * 0.034/2); /* distance = time x speed in cm/microsecond
    distance is divided by because measured duration is for 2 * distance i.e.
    one for going out and other for bouncing back */

    return distance; // returning value of distance in centimeters
}
```

Output



pulseIn() function

Description

Reads a pulse (either HIGH or LOW) on a pin. For example, if value is HIGH, pulseIn() waits for the pin to go from LOW to HIGH, starts timing, then waits for the pin to go LOW and stops timing. Returns the length of the pulse in microseconds or gives up and returns 0 if no complete pulse was received within the timeout.

The timing of this function has been determined empirically and will probably show errors in longer pulses. Works on pulses from 10 microseconds to 3 minutes in length.

Syntax

pulseIn(pin, value, timeout)

Parameters

pin: the number of the pin on which you want to read the pulse. (int)

value: type of pulse to read: either HIGH or LOW. (int)

timeout (optional): the number of microseconds to wait for the pulse to start; default is one second (unsigned long)

Returns

The length of the pulse (in microseconds) or 0 if no pulse started before the timeout (unsigned long).

Project #22 Parking Alarm using Ultrasonic Sensor

In this project we are going to make parking alarm system that we use in our vehicle, simple concept is that it will start beeping when some obstacle comes in front and speed of beeps goes high as the obstacle comes nearer.

Hardware

Arduino UNO

Ultrasonic Sensor

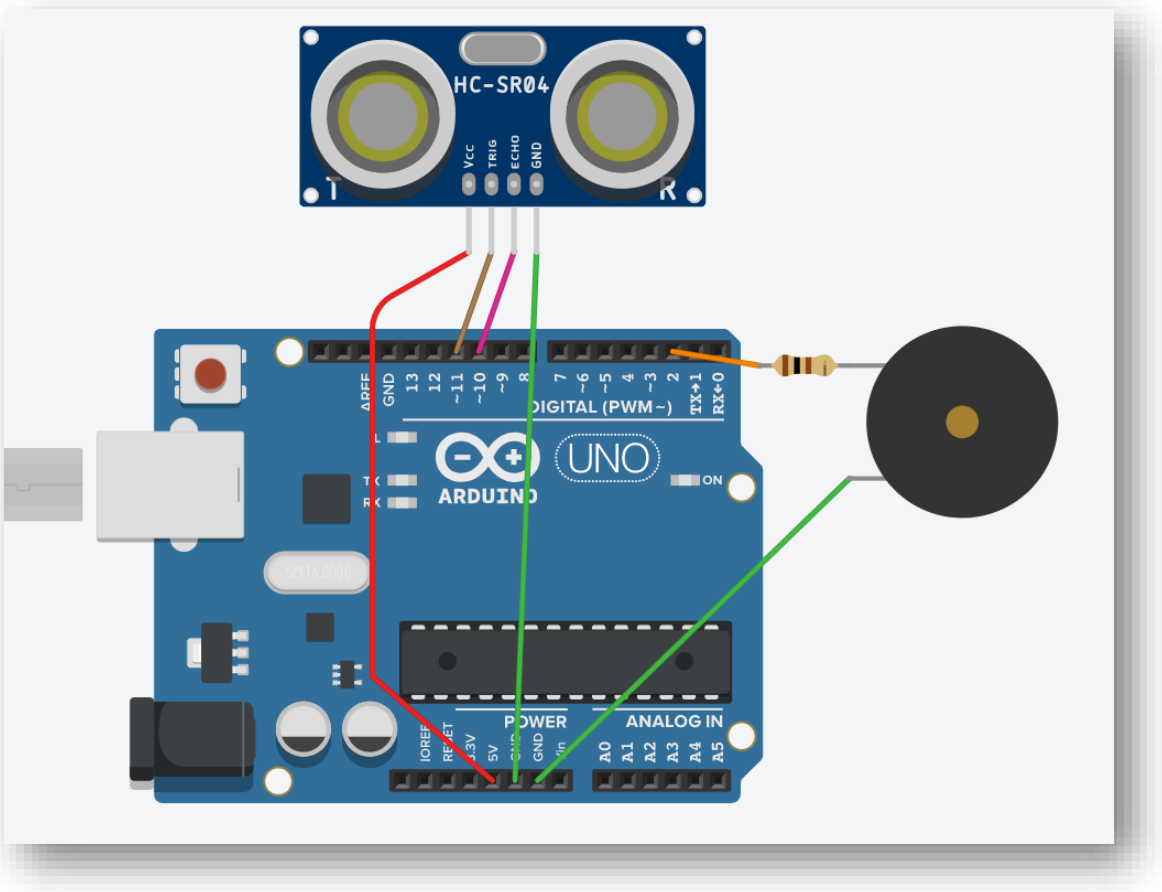
5v Buzzer

Breadboard

Jumper wires

100 ohm resistor

The Schematic

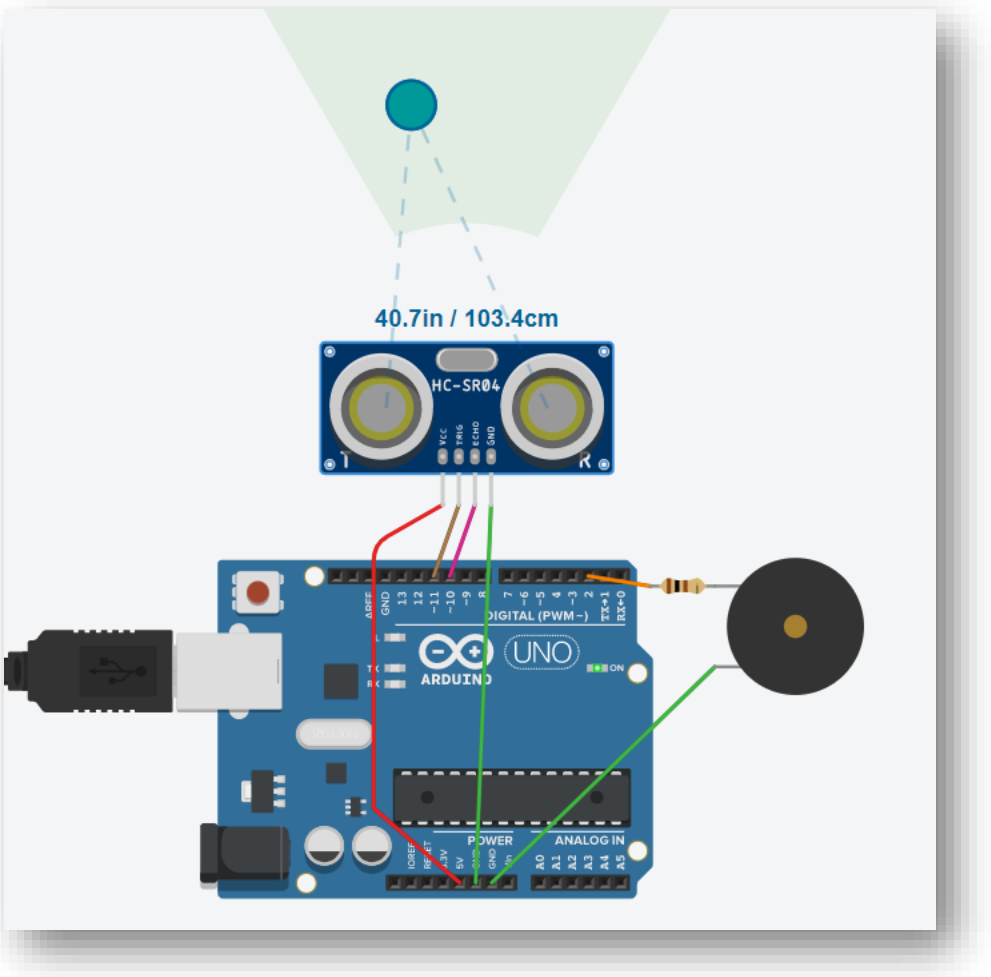


The Sketch

```
#define trigPin 11 //by defining trigPin, we cannot change its value later
#define echoPin 10 // it will be constant throughout
int buzzer = 2;

void setup() {

Serial.begin (9600);

pinMode(trigPin, OUTPUT); // setting trigger pin to output so that it can send signal
s
pinMode(echoPin, INPUT); // setting echo pin as input so that it can receive signals
pinMode(buzzer, OUTPUT); // setting buzzer pin as output pin
}

//float measure_distance(int trig, int echo);
/* declaring the user-defined function that we are
```

```
going to use later in this program (This step is not necessary in Compiler based lang
uages) */

void loop(){

float dist_cm = measure_distance(trigPin,echoPin); // calling out user-
defined function

Serial.print("Distance is :");
Serial.print(dist_cm);
Serial.print(" cm");

  if (dist_cm <= 200){ // if the distance will be less then 2 meters
    tone(buzzer, 1000); // buzzer will start at frequency 1000 hz
    delay(dist_cm); // with delay of the distance

    noTone(buzzer); // and turn of with dilay of distance
    delay(dist_cm);

    /* so if some object will come closer, distance will be reduced, reducing delay t
oo
    so, we will hear beep frequency fast with reduction in distance */
  }
}


float measure_distance(int trig, int echo)
{
    float distance, duration;

    digitalWrite(trig, LOW);
    delayMicroseconds(2);
    digitalWrite(trig, HIGH);
    delayMicroseconds(10);
    digitalWrite(trig, LOW);


    duration = pulseIn(echo, HIGH);
    distance = (duration * 0.034/2);
    return distance;
}
```

Output



Project #23 Water level detector

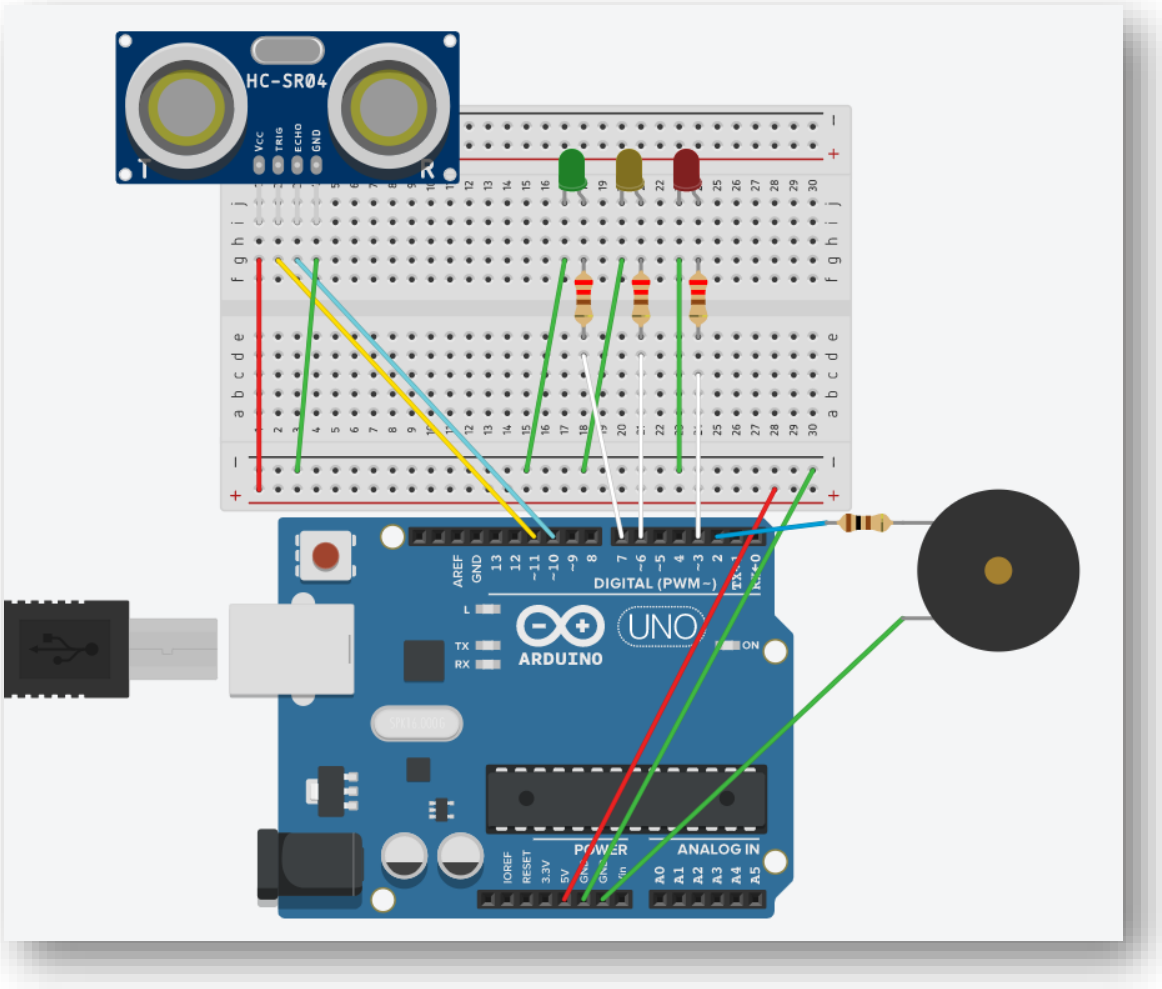
In this project we are going to measure water level in tank, use buzzer & LED as an alarm to convey status of water in the tank.

Hardware

Arduino UNO
Breadboard
Jumper Wires
Red, Yellow & Green LED
3 x 220 ohm resistor,

Ultrasonic Distance sensor
5v Buzzer

The Schematic



The Sketch

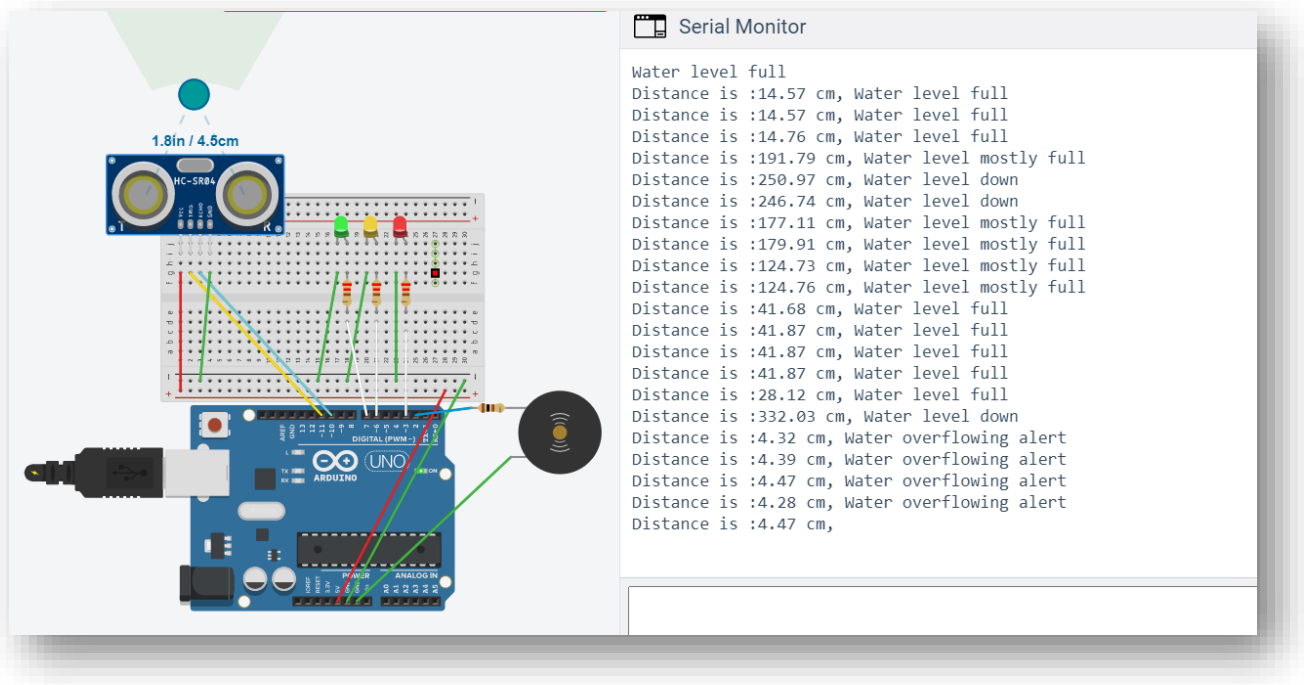
```
#define trigPin 11
#define echoPin 10
int buzzer = 2;
int led[3] = {3, 6, 7}; /* storing 3, 6 & 7 are digital pins
in index 0, 1 & 2 of array led */

void setup() {

Serial.begin (9600);
```

```
pinMode(trigPin, OUTPUT);
pinMode(echoPin, INPUT);
pinMode(buzzer, OUTPUT);
for (int i = 0; i<3; i++){
    pinMode(led[i], OUTPUT);
    digitalWrite(led[i], LOW);
  }
}


void loop(){

float dist_cm = measure_distance(trigPin,echoPin);

Serial.print("Distance is :");
Serial.print(dist_cm);
Serial.print(" cm, ");

  if (dist_cm > 200.0){
    digitalWrite(led[0], HIGH);
    digitalWrite(led[1], LOW);
    digitalWrite(led[2], LOW);
    tone(buzzer, 1000);
    Serial.println("Water level down");
  }

  else if ((dist_cm > 50.0) && (dist_cm <= 200.0)){
    digitalWrite(led[0], LOW);
    digitalWrite(led[1], HIGH);
    digitalWrite(led[2], LOW);
    noTone(buzzer);
    Serial.println("Water level mostly full");
  }

  else if ((dist_cm > 10.0) && (dist_cm <= 50.0)){
    digitalWrite(led[0], LOW);
    digitalWrite(led[1], LOW);
    digitalWrite(led[2], HIGH);
    noTone(buzzer);
    Serial.println("Water level full");
  }

  else
  {
    for (int j = 0; j<3; j++){
      digitalWrite(led[j], HIGH); }
```

```
    siren();
    Serial.println("Water overflowing alert");
  }

}


float measure_distance(int trig, int echo)
{
    float distance, duration;

    digitalWrite(trig, LOW);
    delayMicroseconds(2);
    digitalWrite(trig, HIGH);
    delayMicroseconds(10);
    digitalWrite(trig, LOW);


    duration = pulseIn(echo, HIGH);
    distance = (duration * 0.034/2);
    return distance;
}

void siren(void){
   for (int x=0; x<180; x++) {
        // convert degrees to radians then obtain sin value
        float sinVal = (sin(x*(3.1412/180)));
        // generate a frequency from the sin value
        int toneVal = 2000+(int(sinVal*1000));
        tone(buzzer, toneVal);
        delay(4);
    }
  }
```

Code Overview

Output


1.8in / 4.5cm
HC-SR04
Serial Monitor
Water level full
Distance is :14.57 cm, Water level full
Distance is :14.57 cm, Water level full
Distance is :14.76 cm, Water level full
Distance is :191.79 cm, Water level mostly full
Distance is :250.97 cm, Water level down
Distance is :246.74 cm, Water level down
Distance is :177.11 cm, Water level mostly full
Distance is :179.91 cm, Water level mostly full
Distance is :124.73 cm, Water level mostly full
Distance is :124.76 cm, Water level mostly full
Distance is :41.68 cm, Water level full
Distance is :41.87 cm, Water level full
Distance is :41.87 cm, Water level full
Distance is :41.87 cm, Water level full
Distance is :28.12 cm, Water level full
Distance is :332.03 cm, Water level down
Distance is :4.32 cm, Water overflowing alert
Distance is :4.39 cm, Water overflowing alert
Distance is :4.47 cm, Water overflowing alert
Distance is :4.28 cm, Water overflowing alert
Distance is :4.47 cm,

Project #24 TMP36 Temperature sensor

In this project we are we are going to use TMP36 temperature sensor to measure surrounding temperature and analyze input given by the sensor.

Hardware

- Arduino UNO
- Arduino Wire
- Breadboard
- Jumper wires
- TMP36 sensor

About TMP36 Temperature sensor

The TMP36 series are precision integrated-circuit temperature devices with an output voltage linearly proportional to the Centigrade temperature. TMP36 is three terminal linear temperature sensor. It can measure temperature from -40 degree Celsius to +125 degree Celsius. The voltage output of the TMP36 increases 10mV per degree Celsius rise in temperature. TMP36 can be operated from a 5V supply and the stand by current is less than 60uA.

How to measure Temperature

The TMP36 temperature sensor produces an analog voltage directly proportional to temperature with an output of 1 millivolt per 0.1°C (10 mV per degree). The sketch will convert the analogRead value into voltage and ***subtract the offset to be able to get the temperature***. (Offset is 500 mv)

If you're using a 5V Arduino, and connecting the sensor directly into an Analog pin, you can use these formulas to turn the 10-bit analog reading into a temperature:

Voltage at pin = (reading from sensor) * 5 / 1024

This formula converts the number 0-1024 from the Sensor into 0-5 Volts.

If you're using a 3.3V Arduino, you'll want to use this:

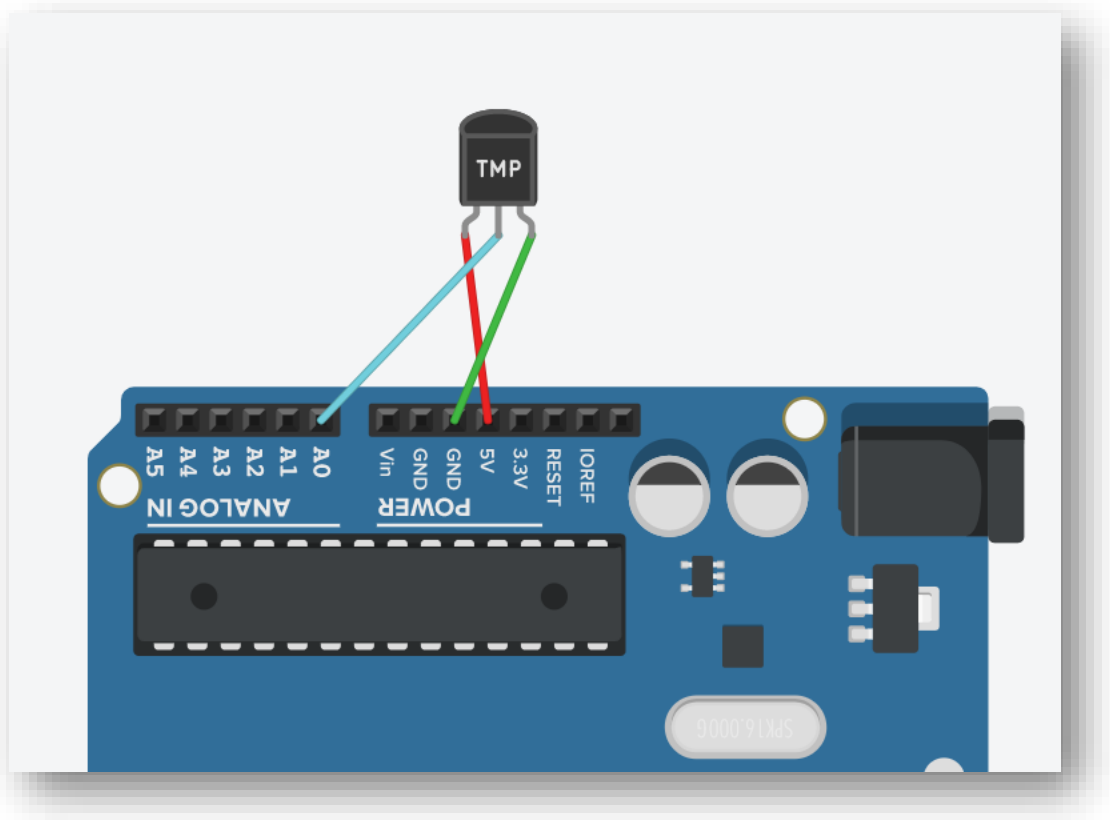
Voltage at pin = (reading from sensor) * 3.3 / 1024

This formula converts the number 0-1024 from the sensor into 0-5V.

Then, to convert volts into degrees we will need the following formule:

Temperature = [(analog voltage in V) – 0.5] * 100

The Schematic

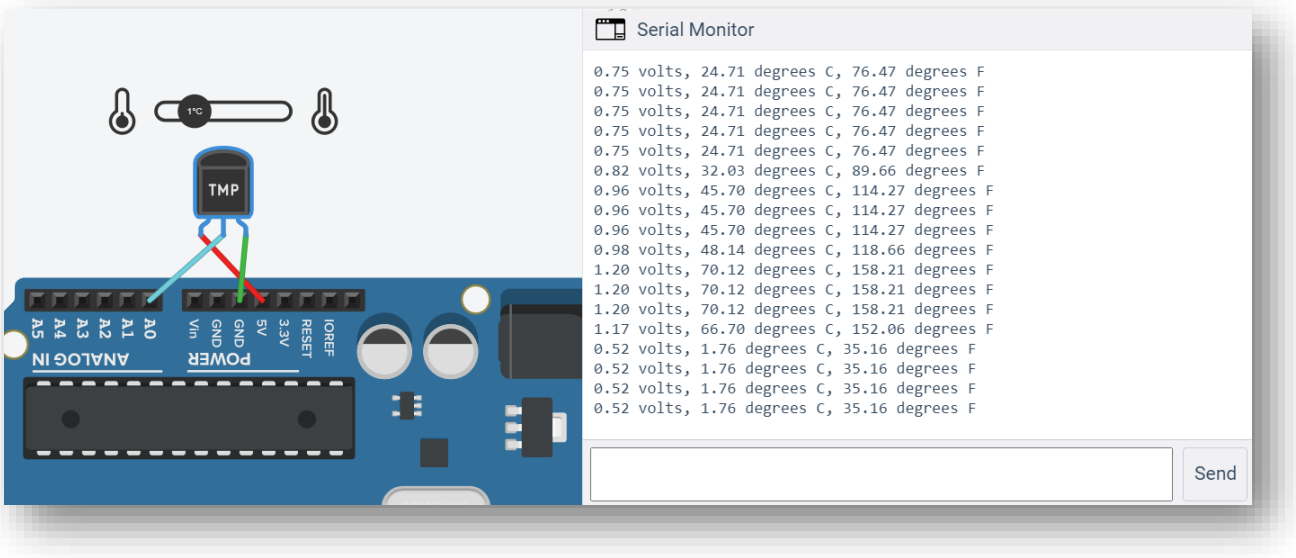


The Sketch

```
int sensorPin = A0;

void setup() {
  Serial.begin(9600); // Start Serial Communication
  pinMode(sensorPin, INPUT); // Optional step for taking input
}

void loop() {
  // read the analog signal from the sensor
  int reading = analogRead(sensorPin);

  /* Convert the reading into voltage, reading by ADC will be from 0 to 1024.
     For 5 volts, multiply by 5 and divide by 1024 */
  float voltage = reading * 5.0/1024.0;


  // print the reading to the Serial Monitor
  Serial.print(voltage); Serial.print(" volts, ");
```

```
  // Convert the 10 mV per degree with an offset of 500mV to degrees
  float temperatureC = (voltage - 0.5) * 100;

  // Print the Temperature
  Serial.print(temperatureC); Serial.print(" degrees C, ");

  // Convert the temperature into Fahrenheit
  float temperatureF = (temperatureC * 9.0 / 5.0) + 32.0;

  // print Fahrenheit to the Serial Monitor
  Serial.print(temperatureF); Serial.println(" degrees F");

  delay(500); // delay for 500 milliseconds.
}
```

Output



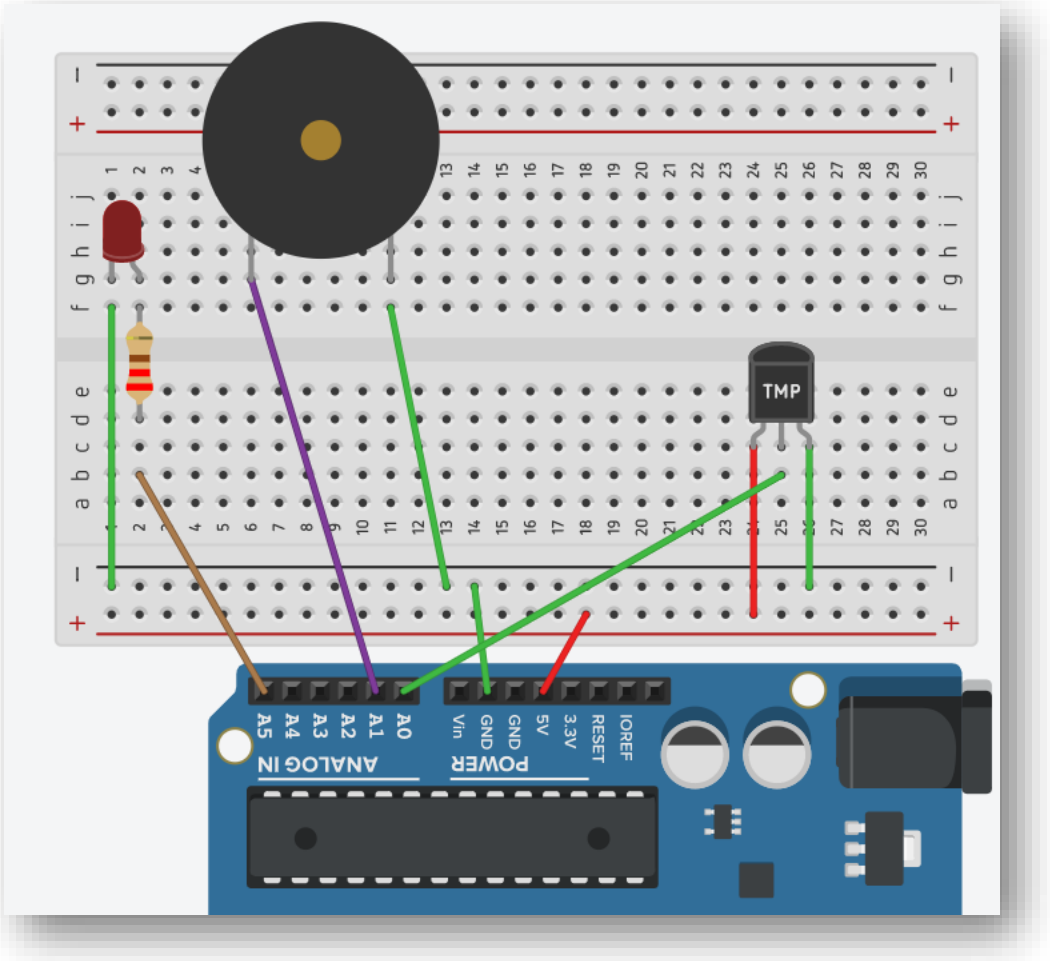
Project #25 Temperature alarm

In this project we are going to use temperature sensor to sense surrounding temperature and start alarm, Red LED when it reaches critical value. I have used >=45 & <= 0 degree Celsius as critical value, you can change them as per requirement.

Hardware

- Arduino UNO
- Arduino Wire
- Breadboard
- Jumper wires
- TMP36 sensor
- 5V Buzzer
- Red LED
- Any resistor between 200 to 500 ohm

The Schematic



The Sketch

```
int sensorPin = A0;
int buzzer = 15; //I am using analog pin as digital here
// Count goes from 14-19 for A0-A5

int led = 19;

void setup() {
  Serial.begin(9600); // Start Serial Communication
  pinMode(sensorPin, INPUT); // Optional step
  pinMode(buzzer, OUTPUT); pinMode(led, OUTPUT);
}

void loop() {
  float temperature = temp();
```

```
  if (temperature >=45 || temperature <=0 ){ /* if temp goes above 44 degress OR be b
elow 1 degree */
    digitalWrite(led, HIGH); // we will make LED HIGH
    siren(); // and will start siren noise
  }

  else // if temp in not in alarming range, turn off buzzer and LED
  {
    noTone(buzzer);
    digitalWrite(led, LOW);
  }

}

float temp (void){
    // read the analog signal from the sensor
  int reading = analogRead(sensorPin);

  /* Convert the reading into voltage, reading by ADC will be from 0 to 1024.
     For 5 volts, multiply by 5 and divide by 1024 */
  float voltage = reading * 5.0/1024.0;


  // print the reading to the Serial Monitor
  Serial.print(voltage); Serial.print(" volts, ");

  // Convert the 10 mV per degree with an offset of 500mV to degrees
  float temperatureC = (voltage - 0.5) * 100;

  // Print the Temperature
  Serial.print(temperatureC); Serial.print(" degrees C, ");

  // Convert the temperature into Fahrenheit
  float temperatureF = (temperatureC * 9.0 / 5.0) + 32.0;

  // print Fahrenheit to the Serial Monitor
  Serial.print(temperatureF); Serial.println(" degrees F");

  return temperatureC;
}

void siren(void){
   for (int x=0; x<180; x++) {
    // convert degrees to radians then obtain sin value
    float sinVal = (sin(x*(3.1412/180)));
    // generate a frequency from the sin value
    int toneVal = 2000+(int(sinVal*1000));
    tone(buzzer, toneVal);
```

```
    delay(4);
  }
  }
```

Output