

Robotics & Programming

Academy: Cybernetics Robo Academy, Bangladesh

Class : Bravo (Class 8 - 10) & Charlie (Class 11 - 12)

Total Class : 16 , Course Hour : 32 Hours(2 Hours/Class)

Admission : +88017 61 500020

Written By :

Syed Razwanul Haque (Nabil) ,
CTO, Cybernetics Robo Academy, Bangladesh
CEO, CRUX (www.cruxbd.com)

github.com/Nabilphysics
www.nabilbd.com

Class 1:

- 1.1 - Introduction to Programming and Robotics
- 1.2 - Basic Math(int, float, double , basic algebra, algebraic addition, subtraction, multiplication, divide, etc)
- 1.3 - Relation between programming and mathematics
- 1.4 - Brief Discussion About Arduino, Microcontroller & Robotics
- 1.5 - Why should we use arduino
- 1.6 - Discuss about our plan
- 1.7 - Install Codeblocks & Arduino IDE

College Level:

- 1.8 - Comparison Between various microcontroller
- 1.9 - 8 bit, 16 bit, 32 bit Microcontroller

Class 2:

- 2.1 - Practical Introduction to Arduino IDE
- 2.2 - Voltage
- 2.3 - Current
- 2.4 - Resistance & Resistor
- 2.5 - Power Supply / Adapter / Voltage Regulator / Arduino 5V
- 2.6 - Breadboard
- 2.7 - LED Connection & On/Off
- 2.8 - Switch / Pushbutton
- 2.9 - Series & Parallel Connection
- 2.10 - Variable in C & Its Type

College Level:

- 2.11 - Resistor Color Code (See 2.4.10 to 2.4.13)
- 2.12 - Resistor Power Rating (See 2.4.14)
- 2.13 - Parts Datasheet
- 2.14 - Install Electronics Plus App for Calculation & Datasheet

Class 3:

- 3.1 - Identifier & Keyword in C
- 3.2 - Input & Output in C
- 3.3 - Addition , Subtraction, Multiplication , Division in C
- 3.4 - Analog and Digital
- 3.5 - Linear Voltage Regulator
- 3.6 - Ohm's Law & Voltage Divider
- 3.7 - Capacitor
- 3.8 - Pull Up & Pull Down Resistor
- 3.9 - Arduino Simple Code Structure
- 3.10 - Single & Multiple LED Blink
- 3.11 - Digital Read
- 3.12 - Basic Serial Communication

College Level:

3.13 - Button Debounce

Class 4:

- 4.1 - Local & Global Variable
- 4.2 - Arduino - Control Statements
- 4.3 - Basic Multimeter Operation
- 4.4 - Voltage Reading with Multimeter
- 4.5 - Variable Resistor & Analog Read (ADC)

College Level :

4.6 - Voltage Divider Details

Class 5:

- 5.1 - Discuss about previous class
- 5.2 - Loop
- 5.3 - Code - Even number & Odd Number
- 5.4 - LED blinking using loop
- 5.5 - Serial Plotter

College Level :

5.6 - ADC Resolution

Class 6:

- 6.1 - Multiple LED control based on if else
- 6.2 - LED control with Serial Read
- 6.3 - LED on off using Processing
- 6.4 - Toggle
- 6.5 - Nested loop
- 6.6 - Increment, Decrement using for loop

College Level:

- 6.7 - EEPROM
- 6.8 - Pyramid print with star

Class 7:

- 7.1 - Array in C
- 7.2 - Array in Arduino
- 7.3 - C Operators
- 7.4 - Analogwrite
- 7.5 - LED Fade
- 7.6 - Servo Motor Control
- 7.7 - Battery - Voltage , Discharge Rate, Series, Parallel, Internal Resistance

College Level:

- 7.8 - Basic Interrupt
- 7.9 - VS Code for Arduino

7.10 - AnalogWrite Resolution

Class 8:

- 8.1 - Variable Data Types and Size Details
- 8.2 - Project : Bluetooth Control Robot Car (Part 1)
 - 8.2.1 - Various motor types
 - 8.2.2 - Parameter for choosing a motor
 - 8.2.3 - DC Motor Control Mechanism
 - 8.2.4 - DC Motor Driver IC
 - 8.2.5 - DC Motor Control with Serial

College Level:

- 6 Motor Drive using 3 Motor Driver

Class 9:

- 9.1 - Function in Arduino
- 9.2 - Project : Bluetooth Control Robot Car (Part 2)
 - 9.2.1 - Bluetooth Module (HC-05)
 - 9.2.2 - HC-05 connection with Arduino
 - 9.2.3 - Chat between bluetooth and arduino serial monitor

College Level:

- 9.3 - Arduino Preprocessor
- 9.4 - Arduino RAM & ROM

Class 10 :

- 10.1 - Project : Bluetooth Control Robot Car (Part 3)
 - 10.1.1 - Connect All Together
 - 10.1.2 - Upload Code
 - 10.1.3 - Download App, Connect & Test
 - 10.1.4 - Tools: Glue Gun, PVC, Soldering Iron
 - 10.1.5 - Assemble All & Drive

Class 11:

- 11.1 - Ohm's Law & Power Details
- 11.2 - Project : Sonar Sensor Integration and Show data in Serial & LCD (Part 1)
 - College Level:**

- 11.3 - Interface Laser Distance Measuring Sensor

Class 12:

- Project : Sonar Sensor Integration and Show data in Serial & LCD (Part 2)
 - 12.1 - LCD Interfacing with Arduino
 - 12.2 - LCD Shield Interfacing with Arduino
 - 12.3 - Show sonar sensor data in LCD
 - College Level**
 - 12.4 - I2C Protocol , I2C LCD

Class 13:

- 13.1 - Project :Make a LFR(Line Follower Robot) (Part 1) - Introduction
- 13.2 - Basic LFR Mechanism
- 13.3 - Line Detection & Control Mechanism
- 13.4 - LFR Circuit & Sensor Array Interfacing
- 13.5 - Sensor Data Show in Serial

Class 14:

- 14.1 - LFR Part-2 : Make a complete LFR Robot
- 14.2 - Guide to Improvement

College Level :

- 14.3 - DIY Sensor circuit with TCRT-5000

Class 15:

- Project: Temperature Sensor Interfacing and Show Temperature
- 15.1 - LM35 Temperature Sensor Interfacing
 - 15.2 - Waterproof NTC Thermistor Interfacing

College Level:

- 15.3 - Basic Oscilloscope Operation

Class 16:

- 16.1 - Introduction to RFID
- 16.2 - Introduction to Fingerprint Sensor
- 16.3 - Do it yourself / Homework

Parts Needed for This Course

- 1) Arduino Mega / Arduino Uno
- 2) Bread Board
- 3) Multimeter
- 4) Jumper Wire (M-M, M-F, F-F)
- 5) Resistor (220 Ohms, 1K, 10K)
- 6) Push Button
- 7) LED
- 8) Potentiometer (10K)
- 9) Servo Motor (SG-90)
- 10) 18650 Battery Case
- 11) 18650 Battery
- 12) 18650 Battery Charger
- 13) L298N Motor Driver
- 14) HC-05 Bluetooth Module
- 15) Tools - Glue Gun
- 16) Tools - Soldering Iron

- 17) Tools - Screwdriver Set
- 18) DC Gear Motor with Wheel (Yellow)
- 19) Ball Caster
- 20) PVC Board for Robot
- 21) LCD Shield
- 22) HC-SR04 Ultrasonic Sonar Sensor
- 23) Line Follower Sensor Array
- 24) TCRT-5000 Sensor
- 25) LM35 Temperature Sensor
- 26) vl53l0x Time of Flight Distance Sensor
- 27) NTC Thermistor
- 28) Oscilloscope (Optional)

Class Details

Class 1:

- 1.1 - Introduction to Programming and Robotics
- 1.2 - Basic Math(int, float, double, basic algebra, algebraic addition, subtraction, multiplication, divide, etc)
- 1.3 - Relation between programming and mathematics
- 1.4 - Relation between programming and robotics
- 1.5 - Brief Discussion About Microcontroller & Arduino
- 1.6 - Why should we use arduino
- 1.7 - Discuss our course plan
- 1.8 - Install Arduino IDE & Codeblocks

College Level:

- 1.9 - Comparison Between various microcontroller
- 1.10 - 8 bit, 16 bit, 32 bit Microcontroller

1.1 - Introduction to Programming and Robotics

Word robot was coined by a Czech novelist Karel Capek in a 1920 play titled Rossum's Universal Robots (RUR). Robot in Czech is a word for workers or servants. The first programmable robot is designed by George Devol, who coins the term Universal Automation. He later shortens this to Unimation, which becomes the name of the first robot company (1962). According to the robot institute of America, A robot is a reprogrammable, multifunctional manipulator designed to move material, parts, tools or specialized devices through variable programmed motions for the performance of a variety of tasks. We use various programming languages to make a robot. The robot contains a computer or microcontroller to run its instructions or algorithms.

Typical knowledgebase for the design and operation of robotics systems

- Dynamic system modeling and analysis
- Feedback control
- Sensors and signal conditioning
- Actuators (muscles) and power electronics
- Hardware/computer interfacing
- Computer programming

Various Robot :

(reference: <http://engineering.nyu.edu/mechatronics/smart/pdf/Intro2Robotics.pdf>)

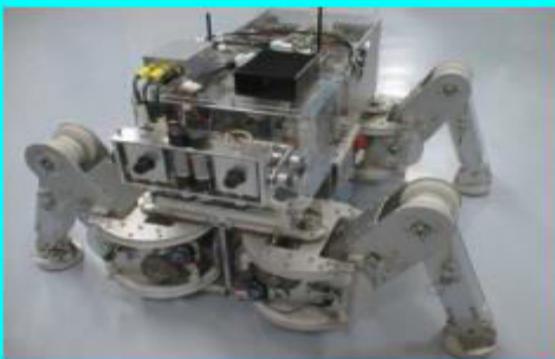
Types of Robots: I

Manipulator



Types of Robots: II

Legged Robot



Wheeled Robot



<http://engineering.nyu.edu/mechatronics/smart/pdf/Intro2Robotics.pdf>

Types of Robots: III

Autonomous Underwater Vehicle



Unmanned Aerial Vehicle



<http://engineering.nyu.edu/mechatronics/smart/pdf/Intro2Robotics.pdf>

Use of Robot :

Robot Uses: I



Jobs that are dangerous
for humans

Decontaminating Robot

Cleaning the main circulating pump
housing in the nuclear power plant

<http://engineering.nyu.edu/mechatronics/smart/pdf/Intro2Robotics.pdf>

Robot Uses: II



Welding Robot

Repetitive jobs that are boring, stressful, or labor-intensive for humans

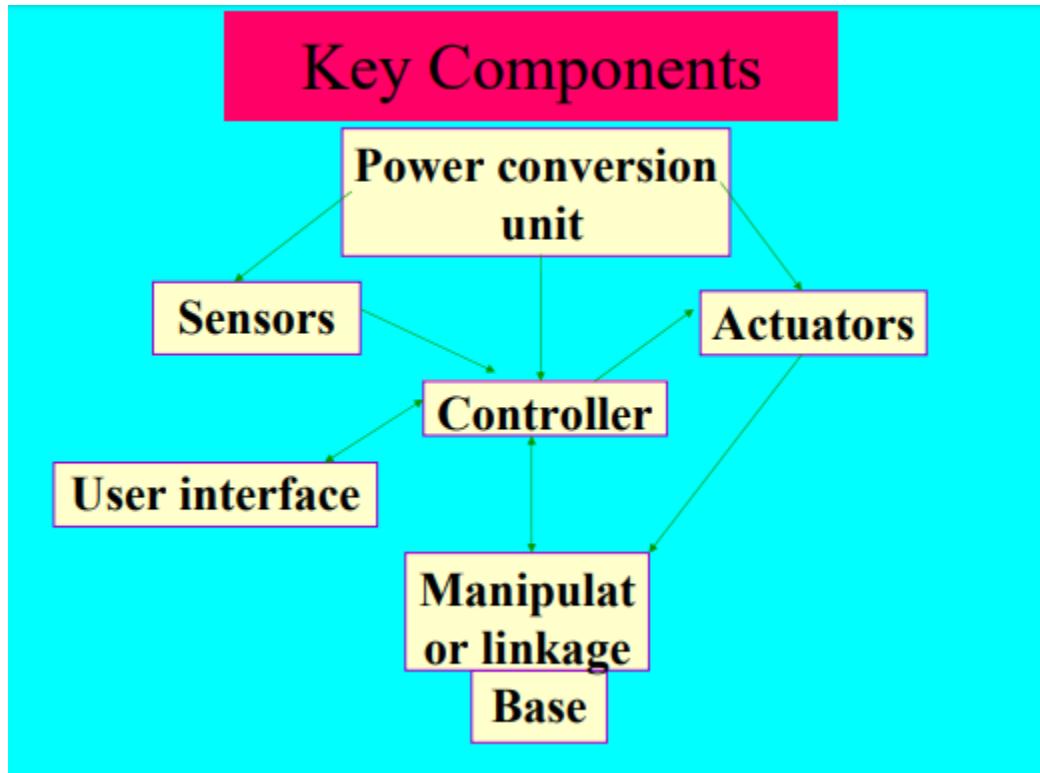
Robot Uses: III



The SCRUBMATE Robot

Menial tasks that human don't want to do

Key Components of Robotics :



Various Sensors used in Robotics:

Sensors

- Human senses: sight, sound, touch, taste, and smell provide us vital information to function and survive
- Robot sensors: measure robot configuration/condition and its environment and send such information to robot controller as electronic signals (e.g., arm position, presence of toxic gas)
- Robots often need information that is beyond 5 human senses (e.g., ability to: see in the dark, detect tiny amounts of invisible radiation, measure movement that is too small or fast for the human eye to see)



Accelerometer
Using Piezoelectric Effect



Flexiforce Sensor

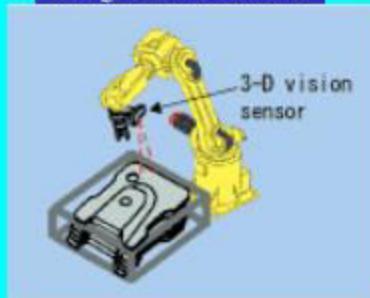
Vision Sensors

Vision Sensor: e.g., to pick bins, perform inspection, etc.

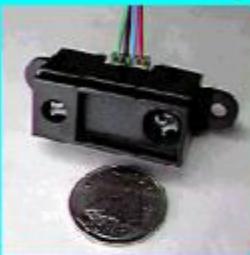
Part-Picking: Robot can handle work pieces that are randomly piled by using 3-D vision sensor. Since alignment operation, a special parts feeder, and an alignment pallete are not required, an automatic system can be constructed at low cost.



In-Sight Vision Sensors



Proximity Sensors



Infrared Ranging Sensor

Devantech SRF04



UltraSonic Ranger

Example



KOALA ROBOT

- 6 ultrasonic sonar transducers to explore wide, open areas
- Obstacle detection over a wide range from 15cm to 3m
- 16 built-in infrared proximity sensors (range 5-20cm)
- Infrared sensors act as a “virtual bumper” and allow for negotiating tight spaces

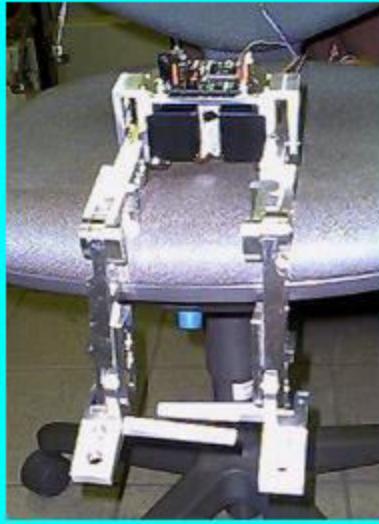
Tilt Sensors

Tilt sensors: e.g., to balance a robot



Tilt Sensor

Example



Planar Bipedal Robot

Actuators/Muscles: I

- Common robotic actuators utilize combinations of different electro-mechanical devices
 - Synchronous motor
 - Stepper motor
 - AC servo motor
 - Brushless DC servo motor
 - Brushed DC servo motor



<http://www.ab.com/motion/servo/fseries.html>

1.2 - Basic Math(int, float, double , basic algebra, algebraic addition, subtraction, multiplication, division, etc)

The teacher will explain in the classroom.

Video : <https://www.youtube.com/watch?v=2SpuBqvNjHI>

1.3 - Relation between programming and mathematics

The teacher will explain in the classroom.

link : <https://softwareengineering.stackexchange.com/questions/136987/what-does-mathematics-have-to-do-with-programming>

1.4 - Brief Discussion About Microcontroller & Arduino

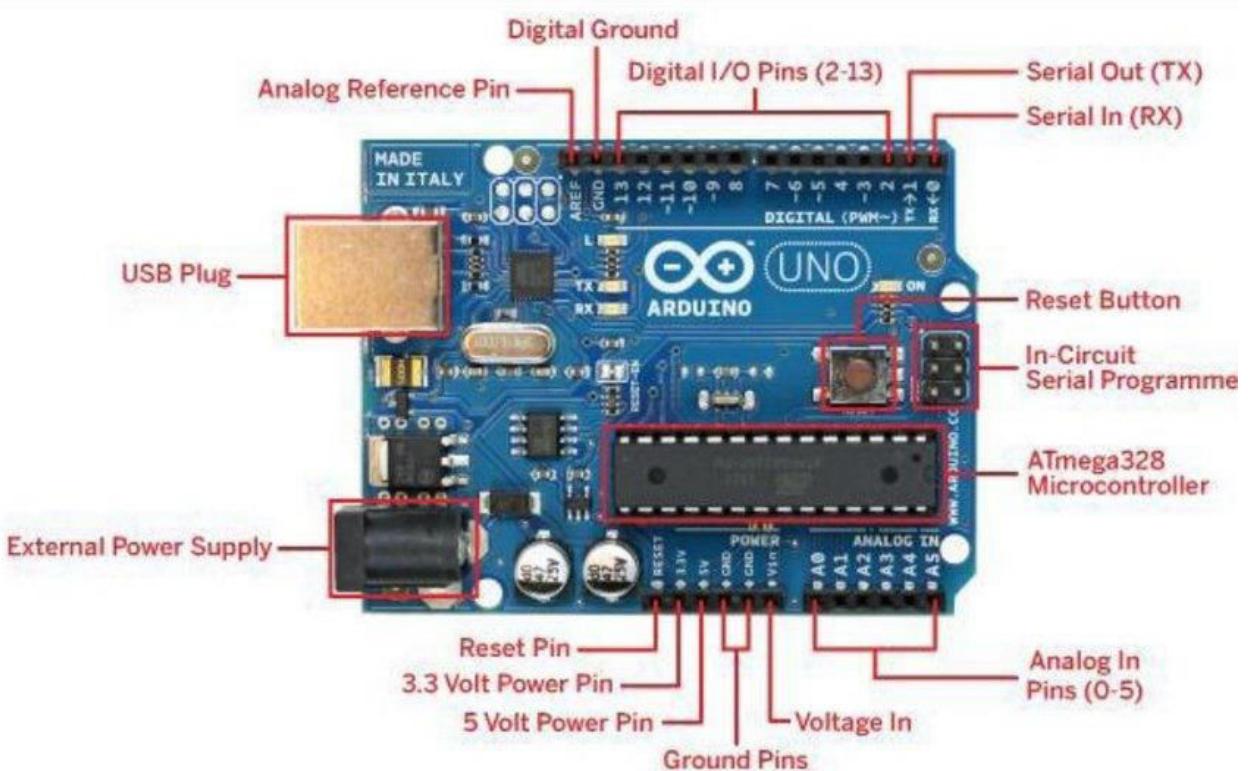
1.4.1 - Microcontroller

A **microcontroller** (**MCU** for *microcontroller unit*, or **UC** for μ -*controller*) is a small computer on a single integrated circuit. In modern terminology, it is similar to, but less sophisticated than, a system on a chip (SoC); an SoC may include a microcontroller as one of its components. A microcontroller contains one or more CPUs (processor cores) along with memory and programmable input/output peripherals. Program memory in the form of ferroelectric RAM, NOR flash or OTP ROM is also often included on chip, as well as a small amount of RAM.

Microcontrollers are designed for embedded applications, in contrast to the microprocessors used in personal computers or other general purpose applications consisting of various discrete chips. A micro-controller can be comparable to a little stand alone computer; it is an extremely powerful device, which is able of executing a series of pre-programmed tasks and interacting with extra hardware devices. Being packed in a tiny integrated circuit (IC) whose size and weight is regularly negligible, it is becoming the perfect controller for as robots or any machines required some type of intelligent automation. A single microcontroller can be enough to manage a small mobile robot, an automatic washer machine or a security system. Several microcontrollers contains a memory to store the program to be executed, and a lot of input/output lines that can be used to act jointly with other devices, like reading the state of a sensor or controlling a motor.

[reference: <https://en.wikipedia.org/wiki/Microcontroller>

<https://www.elprocus.com/difference-between-avr-arm-8051-and-pic-microcontroller/>]



1.4.1.1 - Microcontroller features

A microcontroller's processor will vary by application. Options range from the simple 4-bit, 8-bit or 16-bit processors to more complex 32-bit or 64-bit processors. In terms of memory, microcontrollers can use random access memory (RAM), flash memory, EPROM or EEPROM. Generally, microcontrollers are designed to be readily usable without additional computing components because they are designed with sufficient onboard memory as well as offering pins for general I/O operations, so they can directly interface with sensors and other components.

Microcontroller architecture can be based on the Harvard architecture or von Neumann architecture, both offering different methods of exchanging data between the processor and memory. With a Harvard architecture, the data bus and instruction are separate, allowing for simultaneous transfers. With a Von Neumann architecture, one bus is used for both data and instructions.

[reference: <https://internetofthingsagenda.techtarget.com/definition/microcontroller>]

1.4.1.2 - Microcontrollers vs. microprocessors

The distinction between microcontrollers and microprocessors has gotten less clear as chip density and complexity have become relatively cheap to manufacture and microcontrollers have thus integrated more "general computer" types of functionality. On the whole, though, microcontrollers can be said to function usefully on their own, with direct connection to sensors and actuators, where microprocessors are designed to maximize compute power on the chip, with internal bus connections (rather than direct I/O) to supporting hardware such as RAM and serial ports. Simply put, coffee makers use microcontrollers; desktop computers use microprocessors.

1.4.4.3 - Arduino

Video :

Arduino Introduction [<https://www.youtube.com/watch?v=bY03PJihDMw>]

Arduino Introduction [<https://www.youtube.com/watch?v=CqrQmQqpHXc>]

What's an Arduino [https://www.youtube.com/watch?v=_h1m6R9YW8c]

Arduino is an open-source electronics platform based on easy-to-use hardware and software.

Arduino boards are able to read inputs - light on a sensor, a finger on a button, or a Twitter message - and turn it into an output - activating a motor, turning on an LED, publishing something online. You can tell your board what to do by sending a set of instructions to the microcontroller on the board. To do so you use the Arduino programming language (based on Wiring), and the Arduino Software (IDE), based on Processing.

Over the years Arduino has been the brain of thousands of projects, from everyday objects to complex scientific instruments. A worldwide community of makers - students, hobbyists, artists, programmers, and professionals - has gathered around this open-source platform, their contributions have added up to an incredible amount of accessible knowledge that can be of great help to novices and experts alike.

Arduino was born at the Ivrea Interaction Design Institute as an easy tool for fast prototyping, aimed at students without a background in electronics and programming. As soon as it reached a wider community, the Arduino board started changing to adapt to new needs and challenges, differentiating its offer from simple 8-bit boards to products for IoT applications, wearable, 3D printing, and embedded environments. All Arduino boards are completely open-source, empowering users to build them independently and eventually adapt them to their particular needs. The software, too, is open-source, and it is growing through the contributions of users worldwide.

[reference - <https://www.arduino.cc/en/guide/introduction>]

1.5 - Why should we use arduino

Thanks to its simple and accessible user experience, Arduino has been used in thousands of different projects and applications. The Arduino software is easy-to-use for beginners, yet flexible enough for advanced users. It runs on Mac, Windows, and Linux. Teachers and students use it to build low cost scientific instruments, to prove chemistry and physics principles, or to get started with programming and robotics. Designers and architects build interactive prototypes, musicians and artists use it for installations and to experiment with new musical instruments. Makers, of course, use it to build many of the projects exhibited at the Maker Faire, for example. Arduino is a key tool to learn new things. Anyone - children, hobbyists, artists,

programmers - can start tinkering just following the step by step instructions of a kit, or sharing ideas online with other members of the Arduino community.

There are many other microcontrollers and microcontroller platforms available for physical computing. Parallax Basic Stamp, Netmedia's BX-24, Phidgets, MIT's Handyboard, and many others offer similar functionality. All of these tools take the messy details of microcontroller programming and wrap it up in an easy-to-use package. Arduino also simplifies the process of working with microcontrollers, but it offers some advantage for teachers, students, and interested amateurs over other systems:

Inexpensive - Arduino boards are relatively inexpensive compared to other microcontroller platforms. The least expensive version of the Arduino module can be assembled by hand, and even the pre-assembled Arduino modules cost less than \$50

Cross-platform - The Arduino Software (IDE) runs on Windows, Macintosh OSX, and Linux operating systems. Most microcontroller systems are limited to Windows.

Simple, clear programming environment - The Arduino Software (IDE) is easy-to-use for beginners, yet flexible enough for advanced users to take advantage of as well. For teachers, it's conveniently based on the Processing programming environment, so students learning to program in that environment will be familiar with how the Arduino IDE works.

Open source and extensible software - The Arduino software is published as open source tools, available for extension by experienced programmers. The language can be expanded through C++ libraries, and people wanting to understand the technical details can make the leap from Arduino to the AVR C programming language on which it's based. Similarly, you can add AVR-C code directly into your Arduino programs if you want to.

Open source and extensible hardware - The plans of the Arduino boards are published under a Creative Commons license, so experienced circuit designers can make their own version of the module, extending it and improving it. Even relatively

inexperienced users can build the breadboard version of the module in order to understand how it works and save money.

1.6 - Discuss about our course plan

Teacher will discuss in the classroom/lab. Basically, we will try to make this course very enjoyable as well as we will try to learn practically and theoretically.

1.7 - Install Arduino IDE & Codeblocks

1.7.1 - Arduino IDE Installation

Go to this link <https://www.arduino.cc/en/Main/Software> and download arduino software based on your operating system and install it. You may also use arduino web editor.

1.7.2 - Codeblocks IDE Installation

Go to this link <https://www.wikihow.com/Download,-Install,-and-Use-Code::Blocks> and follow instructions. Make sure you have installed Codeblocks with MinGW

College Level:

1.8 - Comparison Between various microcontroller

1.8.1 - 8051 Microcontroller

8051 microcontroller is an 8-bit family of microcontroller is developed by the Intel in the year 1981. This is one of the popular families of microcontroller are being used all across the world. This microcontroller was moreover referred as “system on a chip” since it has 128 bytes of RAM, 4Kbytes of a ROM, 2 Timers, 1 Serial port, and 4 ports on a single chip. The CPU can also work for 8bits of data at a time since 8051 is an 8-bit processor. In case the data is bigger than 8 bits, then it has to be broken into parts so that the CPU can process easily. Most

manufacturers contain put 4Kbytes of ROM even though the number of ROM can be exceeded up to 64 K bytes.

1.8.2 - PIC Microcontroller

Peripheral Interface Controller (PIC) is microcontroller developed by a Microchip, PIC microcontroller is fast and simple to implement program when we contrast other microcontrollers like 8051. The ease of programming and simple to interfacing with other peripherals PIC become successful microcontroller. We know that microcontroller is an integrated chip which is consists of RAM, ROM, CPU, TIMER and COUNTERS. The PIC is a microcontroller which as well consists of RAM, ROM, CPU, timer, counter, ADC (analog to digital converters), DAC (digital to analog converter). PIC Microcontroller also support the protocols like CAN, SPI, UART for an interfacing with additional peripherals. PIC mostly used to modify Harvard architecture and also supports RISC (Reduced Instruction Set Computer) by the above requirement RISC and Harvard we can simply that PIC is faster than the 8051 based controllers which is prepared up of Von-Newman architecture.

1.8.3 - AVR Microcontroller

AVR microcontroller was developed in the year of 1996 by Atmel Corporation. The structural design of AVR was developed by the Alf-Egil Bogen and Vegard Wollan. AVR derives its name from its developers and stands for Alf-Egil Bogen Vegard Wollan RISC microcontroller, also known as Advanced Virtual RISC. The AT90S8515 was the initial microcontroller which was based on the AVR architecture, though the first microcontroller to hit the commercial market was AT90S1200 in the year 1997.

AVR Microcontrollers are Available in three Categories

TinyAVR:- Less memory, small size, appropriate just for simpler applications

MegaAVR:- These are the mainly popular ones having a good quantity of memory (up to 256 KB), higher number of inbuilt peripherals and appropriate for modest to complex applications.

XmegaAVR:- Used in commercial for complex applications, which need large program memory and high speed.

1.8.4 - ARM Processor

An ARM processor is also one of a family of CPUs based on the RISC (reduced instruction set computer) architecture developed by Advanced RISC Machines (ARM).

An ARM makes at 32-bit and 64-bit RISC multi-core processors. RISC processors are designed to perform a smaller number of types of computer instructions so that they can operate at a higher speed, performing extra millions of instructions per second (MIPS). By stripping out unnecessary instructions and optimizing pathways, RISC processors give an outstanding performance at a part of the power demand of CISC (complex instruction set computing) procedure.

ARM processors are widely used in customer electronic devices such as smartphones, tablets, multimedia players and other mobile devices, such as wearables. Because of their reduced to the instruction set, they need fewer transistors, which enable a smaller die size of the integrated circuitry (IC). The ARM processors, smaller size reduced the difficulty and lower power expenditure makes them suitable for increasingly miniaturized devices.

1.8.5 - 8051, PIC, AVR, ARM Comparison Chart

Main Difference between AVR, ARM, 8051 and PIC Microcontrollers

	8051	PIC	AVR	ARM
Bus width	8-bit for standard core	8/16/32-bit	8/32-bit	32-bit mostly also available in 64-bit
Communication Protocols	UART, USART, SPI, I2C	PIC, UART, USART, LIN, CAN, Ethernet, SPI, I2S	UART, USART, SPI, I2C, (special purpose AVR support CAN, USB, Ethernet)	UART, USART, LIN, I2C, SPI, CAN, USB, Ethernet, I2S, DSP, SAI (serial audio interface), IrDA
Speed	12 Clock/instruction cycle	4 Clock/instruction cycle	1 clock/ instruction cycle	1 clock/ instruction cycle
Memory	ROM, SRAM, FLASH	SRAM, FLASH	Flash, SRAM, EEPROM	Flash, SDRAM, EEPROM
ISA	CLSC	Some feature of RISC	RISC	RISC
Memory Architecture	Von Neumann architecture	Harvard architecture	Modified	Modified Harvard architecture
Power Consumption	Average	Low	Low	Low
Families	8051 variants	PIC16, PIC17, PIC18, PIC24, PIC32	Tiny, Atmega, Xmega, special purpose AVR	ARMv4, 5, 6, 7 and series
Community	Vast	Very Good	Very Good	Vast
Manufacturer	NXP, Atmel, Silicon Labs, Dallas, Cypress, Infineon, etc.	Microchip Average	Atmel	Apple, Nvidia, Qualcomm, Samsung Electronics, and TI etc.
Cost (as compared to features provide)	Very Low	Average	Average	Low
Other Feature	Known for its Standard	Cheap	Cheap, effective	High speed operation Vast
Popular Microcontrollers	AT89C51, P89v51, etc.	PIC18fXX8, PIC16f88X, PIC32MXX	Atmega8, 16, 32, Arduino Community	LPC2148, ARM Cortex-M0 to ARM Cortex-M7, etc.

1.9 - 8 bit, 16 bit, 32-bit Microcontroller

8-bit, 16-bit, 32-bit or 64-bit means the size of each instruction the CPU executes. Since all code will be compiled down to these instructions it means the final size will also increase. That's one. If your CPU can process n-bits in x cycles, then it would need 2x cycles to process 2n-bits. So you might wanna define your data types accordingly. A common scenario is the size of an 'int' and the capability to process floating points.

A 16-bit CPU will be 'slower' than a 32-bit one, simply because it can process fewer data at any given time. Of course, there's clock speed and other things.

Cross-compilers. 32-bit MCUs definitely have more cross-compilers than the lower ones.

Finally, the address range. How much memory you can interface. Lower bit CPUs can address a lesser amount of memory.

Basically, an 8-bit device will require a greater number of bus accesses and more instructions to perform 16 or 32bit arithmetic operations so it may be slower by more than its basic clock speed for that reason.

While not related to architecture width, 16 and 8-bit devices are unlikely to incorporate an FPU or MMU, or even cache memory whereas these are more common in 32bit devices.

Class 2:

2.1 - Practical Introduction to Arduino IDE

2.2 - Voltage

2.3 - Current

2.4 - Resistance & Resistor

2.5 - Power Supply / Adapter / Voltage Regulator / Arduino 5V

2.6 - Breadboard

2.7 - LED Connection & On/Off

- 2.8 - Switch / Pushbutton
- 2.9 - Series & Parallel Connection
- 2.10 - Variable in C & Its Type

College Level:

- 2.11 - Resistor Color Code (See 2.4.10 to 2.4.13)**
- 2.12 - Resistor Power Rating (See 2.4.14)**
- 2.13 - Parts Datasheet**
- 2.14 - Install Electronics Plus App for Calculation & Datasheet**

2.1 - Practical Introduction to Arduino IDE

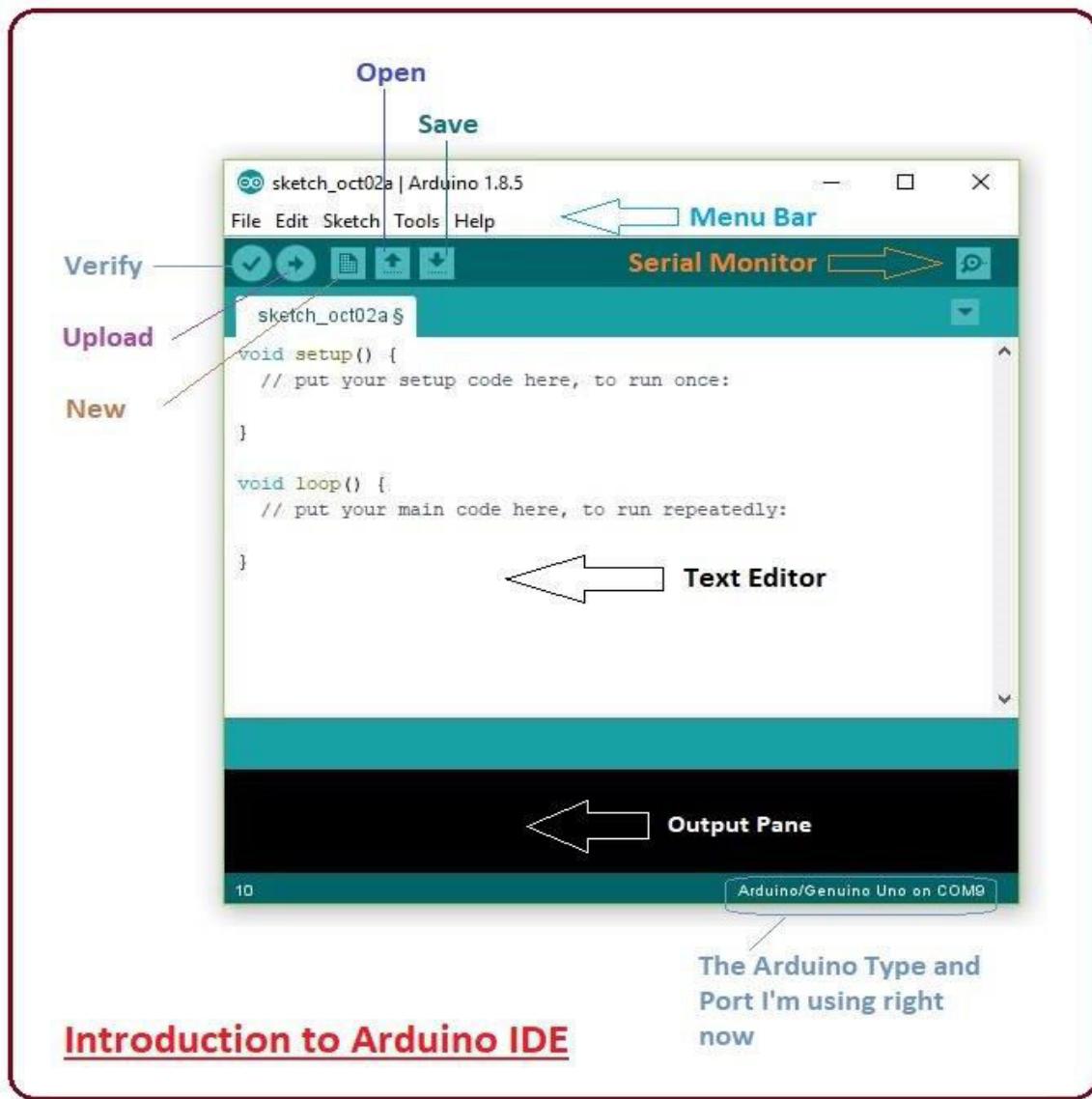
[reference - <https://www.theengineeringprojects.com/2018/10/introduction-to-arduino-ide.html>]
 video : **Introduction to Arduino IDE** [https://www.youtube.com/watch?v=nbD_V4QtNvY]

- 1) Arduino IDE is an open source software that is mainly used for writing and compiling the code into the Arduino Module.
- 2) It is an official Arduino software, making code compilation too easy that even a common person with no prior technical knowledge can get their feet wet with the learning process.
- 3) It is easily available for operating systems like MAC, Windows, Linux and runs on the Java Platform that comes with inbuilt functions and commands that play a vital role for debugging, editing and compiling the code in the environment.
- 4) A range of Arduino modules available including Arduino Uno, Arduino Mega, Arduino Leonardo, Arduino Micro and many more.
- 5) Each of them contains a microcontroller on the board that is actually programmed and accepts the information in the form of code.
- 6) The main code, also known as a sketch, created on the IDE platform will ultimately generate a Hex File which is then transferred and uploaded in the controller on the board.
- 7) The IDE environment mainly contains two basic parts: Editor and Compiler where former is used for writing the required code and later is used for compiling and uploading the code into the given Arduino Module.
- 8) This environment supports both C and C++ languages.

The IDE environment is mainly distributed into three sections

1. Menu Bar
2. Text Editor
3. Output Pane

As you download and open the IDE software, it will appear like an image below.

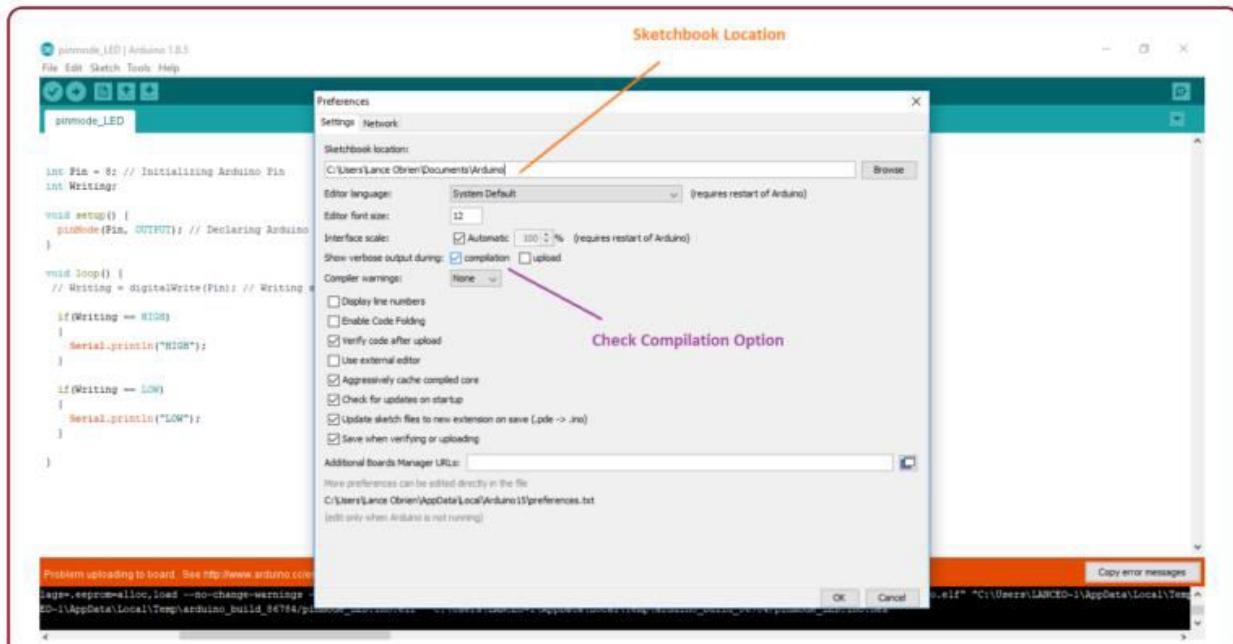


The bar appearing on the top is called **Menu Bar** that comes with five different options as follow

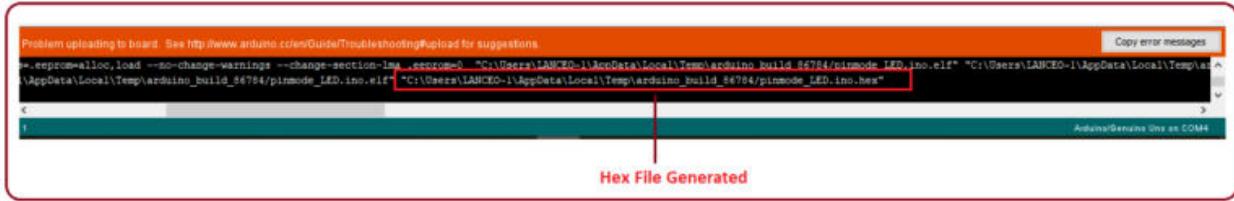
- 1) **File** – You can open a new window for writing the code or open an existing one. Following table shows the number of further subdivisions the file option is categorized into.

File	
New	This is used to open new text editor window to write your code
Open	Used for opening the existing written code
Open Recent	The option reserved for opening recently closed program
Sketchbook	It stores the list of codes you have written for your project
Examples	Default examples already stored in the IDE software
Close	Used for closing the main screen window of recent tab. If two tabs are open, it will ask you again as you aim to close the second tab
Save	It is used for saving the recent program
Save as	It will allow you to save the recent program in your desired folder
Page setup	Page setup is used for modifying the page with portrait and landscape options. Some default page options are already given from which you can select the page you intend to work on
Print	It is used for printing purpose and will send the command to the printer
Preferences	It is page with number of preferences you aim to setup for your text editor page
Quit	It will quit the whole software all at once

As you go to the preference section and check the compilation section, the Output Pane will show the code compilation as you click the upload button.



And at the end of compilation, it will show you the hex file it has generated for the recent sketch that will send to the Arduino Board for the specific task you aim to achieve.



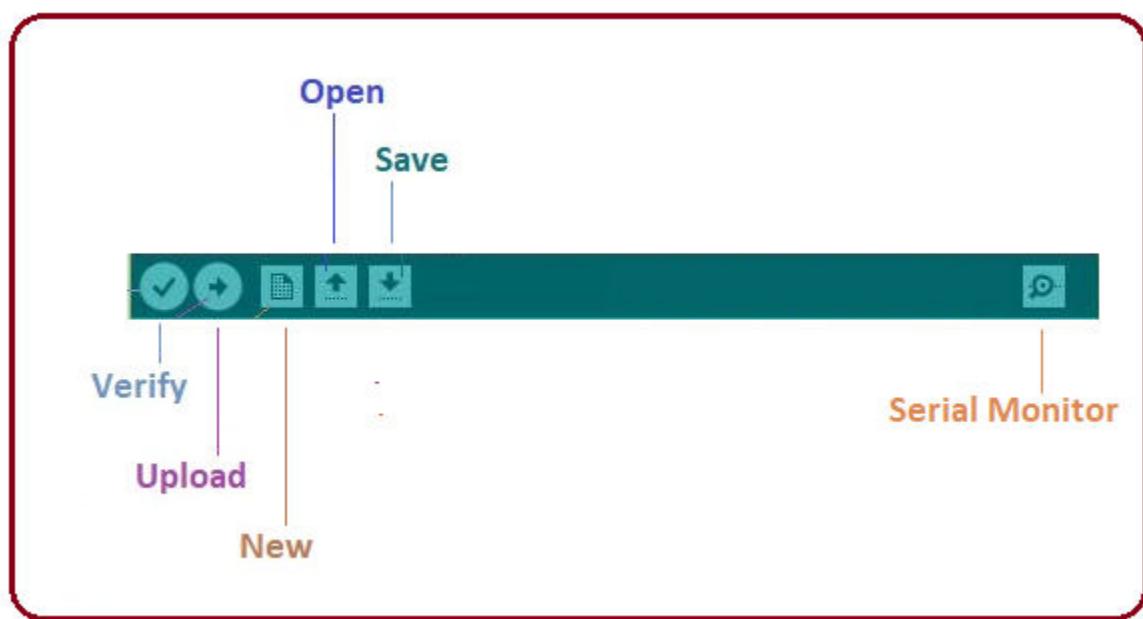
Edit – Used for copying and pasting the code with further modification for font

Sketch – For compiling and programming

Tools – Mainly used for testing projects. The Programmer section in this panel is used for burning a bootloader to the new microcontroller.

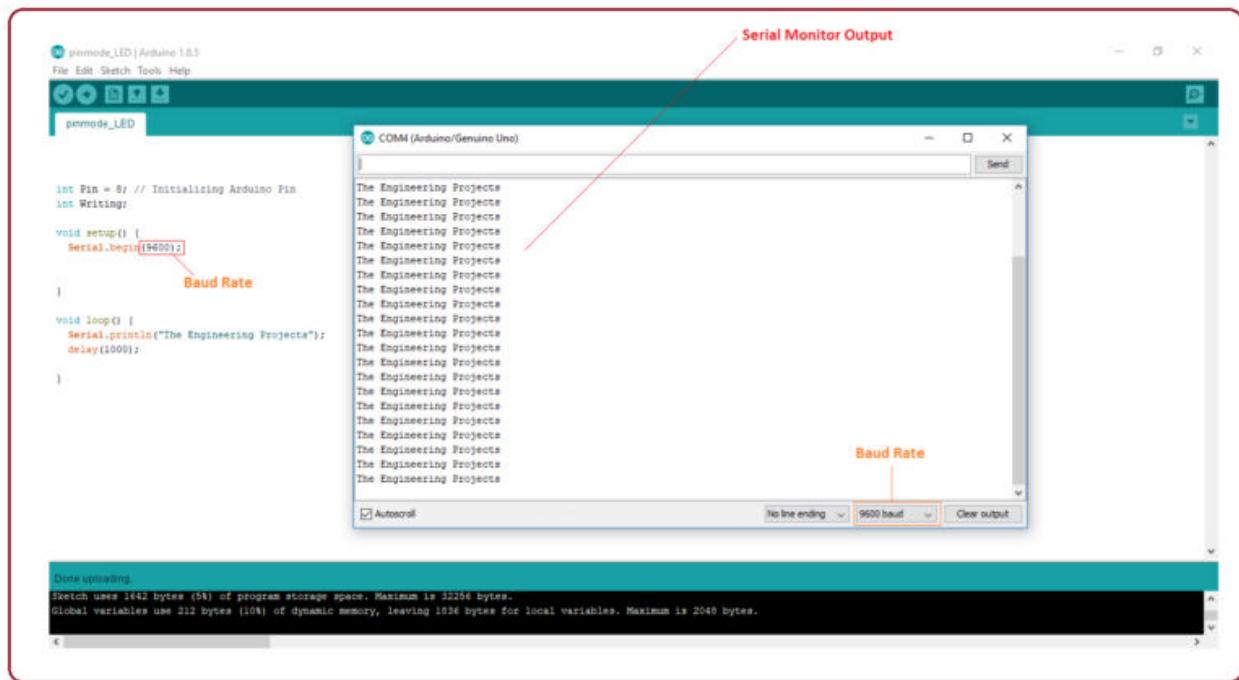
Help – In case you are feeling skeptical about software, complete help is available from getting started to troubleshooting.

The Six Buttons appearing under the Menu tab are connected with the running program as follow.



- > The check mark appearing in the circular button is used to verify the code. Click this once you have written your code.
- > The arrow key will upload and transfer the required code to the Arduino board.
- > The dotted paper is used for creating a new file.
- > The upward arrow is reserved for opening an existing Arduino project.

- > The downward arrow is used to save the current running code.
- > The button appearing on the top right corner is a **Serial Monitor** – A separate pop-up window that acts as an independent terminal and plays a vital role for sending and receiving the Serial Data. You can also go to the Tools panel and select Serial Monitor, or pressing Ctrl+Shift+M all at once will open it instantly. The Serial Monitor will actually help to debug the written Sketches where you can get a hold of how your program is operating. Your Arduino Module should be connected to your computer by USB cable in order to activate the Serial Monitor.
- > You need to select the baud rate of the Arduino Board you are using right now. For my Arduino Uno Baud Rate is 9600, as you write the following code and click the Serial Monitor, the output will show as the image below.



The main screen below is known as a simple text editor used for writing the required code.

```

int Pin = 8; // Initializing Arduino Pin
int Writing;

void setup() {
    pinMode(Pin, OUTPUT); // Declaring Arduino Pin as an Output
}

void loop() {
    Writing = digitalWrite(Pin); // Writing status of Arduino digital Pin

    if(Writing == HIGH)
    {
        Serial.println("HIGH");
    }

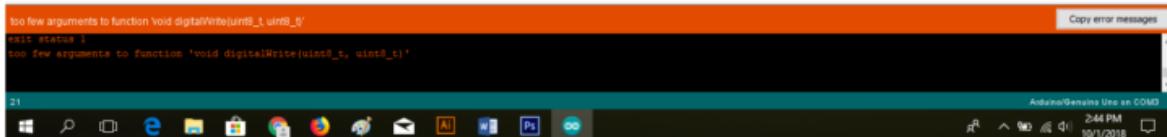
    if(Writing == LOW)
    {
        Serial.println("LOW");
    }
}

```



Text Editor

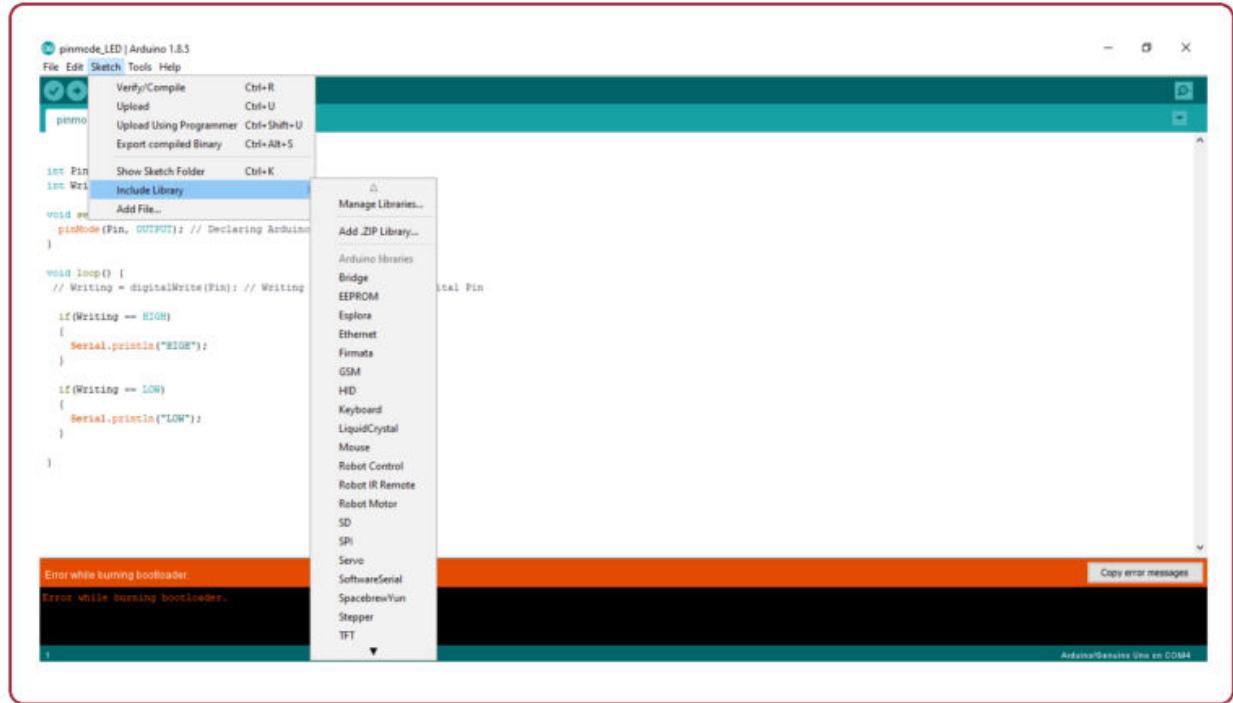
The bottom of the main screen is described as an Output Pane that mainly highlights the compilation status of the running code: the memory used by the code, and errors occurred in the program. You need to fix those errors before you intend to upload the hex file into your Arduino Module.



Output Window

Libraries :

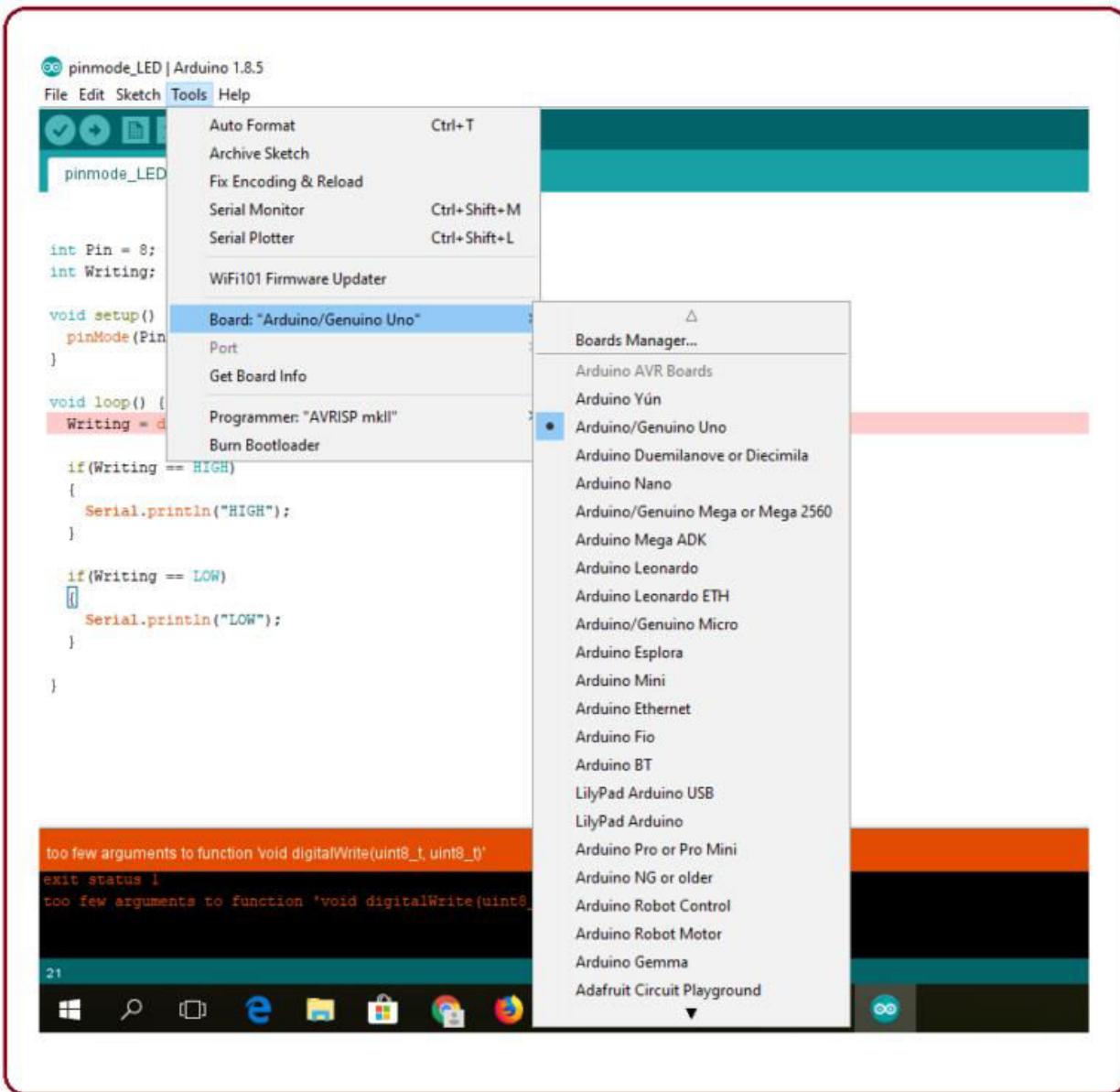
Libraries are very useful for adding the extra functionality into the Arduino Module. There is a list of libraries you can add by clicking the Sketch button in the menu bar and going to Include Library.



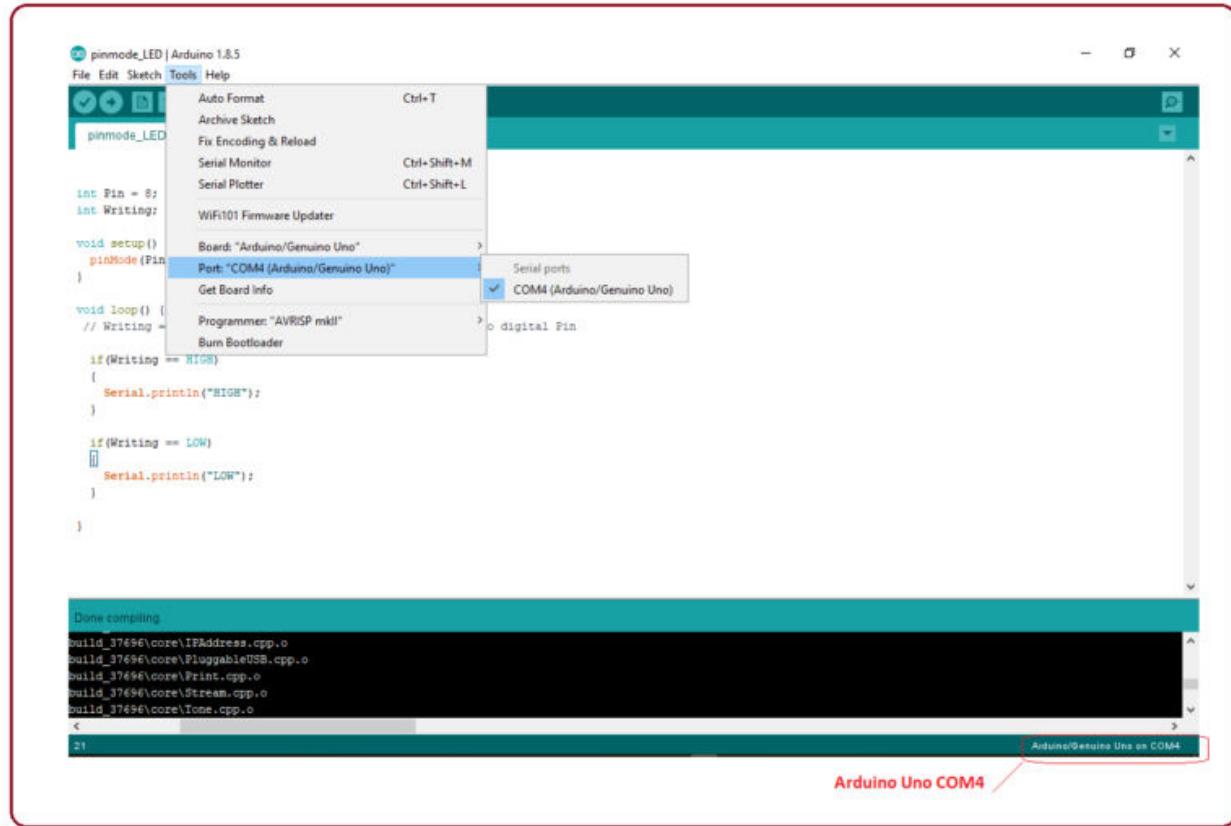
Most of the libraries are preinstalled and come with the Arduino software. However, you can also download them from the external sources.

How to Select the Board :

In order to upload the sketch, you need to select the relevant board you are using and the ports for that operating system. As you click the Tools on the Menu, it will open like the figure below. Just go to the “Board” section and select the board you aim to work on. Similarly, COM1, COM2, COM4, COM5, COM7 or higher are reserved for the serial and USB board. You can look for the USB serial device in the ports section of the Windows Device Manager.

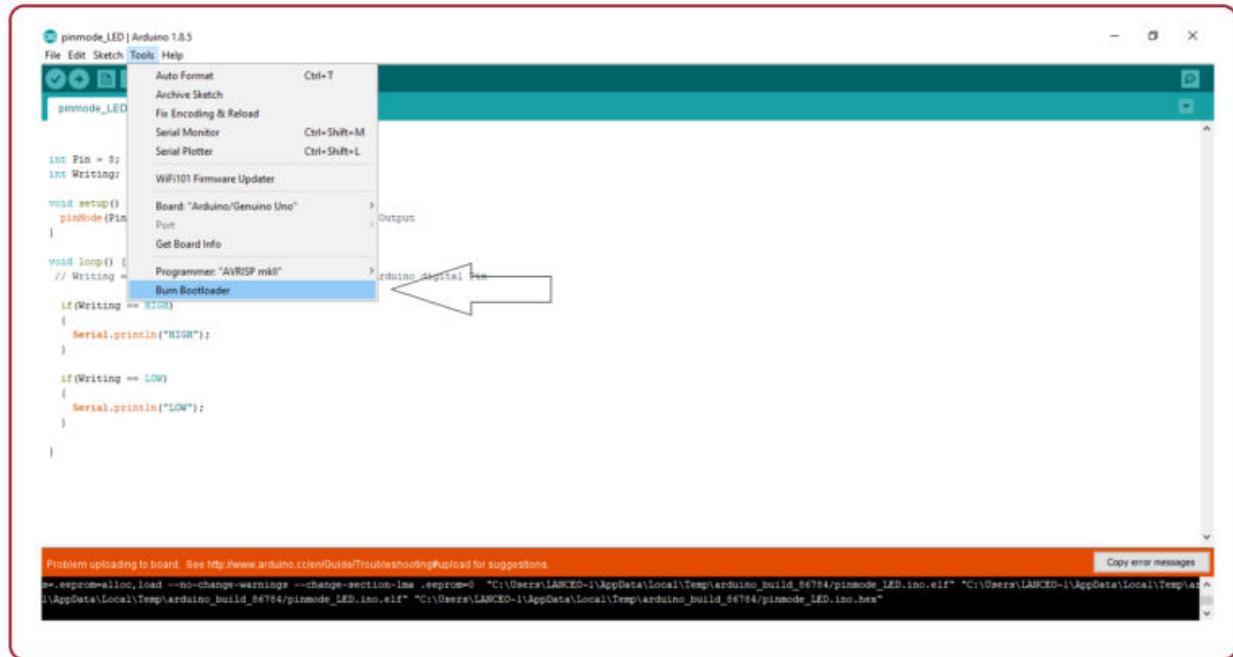


Following figure shows the COM4 that I have used for my project, indicating the Arduino Uno with COM4 port at the right bottom corner of the screen.



Bootloader :

As you go to the Tools section, you will find a bootloader at the end. It is very helpful to burn the code directly into the controller, setting you free from buying the external burner to burn the required code. When you buy the new Arduino Module, the bootloader is already installed inside the controller. However, if you intend to buy a controller and put in the Arduino module, you need to burn the bootloader again inside the controller by going to the Tools section and selecting the burn bootloader.



Video : How to Upload Arduino Bootloader

[<https://www.youtube.com/watch?v=VX0P8B3pk7g>]

How to install bootloader onto arduino with USBasp

[<https://www.youtube.com/watch?v=Rz6Guj6qY1o>]

2.2 - Voltage

[reference - <https://www.fluke.com/en/learn/best-practices/measurement-basics/electricity/what-is-voltage>

<https://learn.sparkfun.com/tutorials/voltage-current-resistance-and-ohms-law/all>

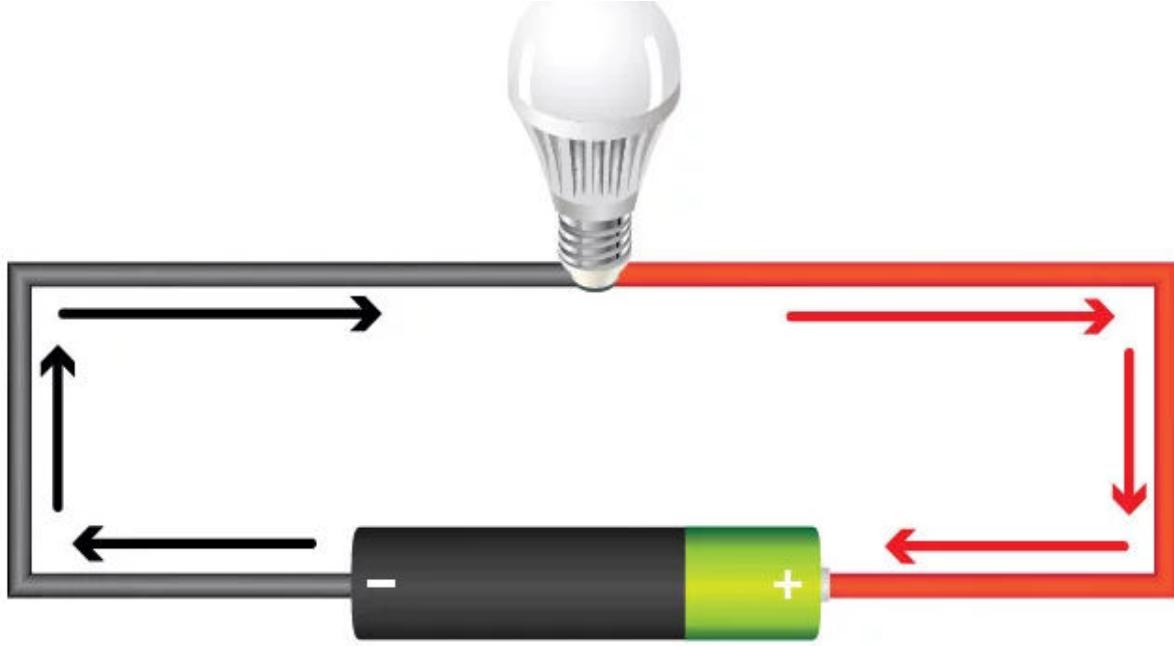
Video : What is Voltage [<https://www.youtube.com/watch?v=z8qfhFXjsrw>]

Voltage is the pressure from an electrical circuit's power source that pushes charged electrons (current) through a conducting loop, enabling them to do work such as illuminating a light.

In brief, voltage = pressure, and it is measured in volts (V). The term recognizes Italian physicist Alessandro Volta (1745-1827), inventor of the voltaic pile—the forerunner of today's household battery.

In electricity's early days, voltage was known as electromotive force (emf). This is why in equations such as Ohm's Law, voltage is represented by the symbol E.

Example of voltage in a simple direct current (dc) circuit:

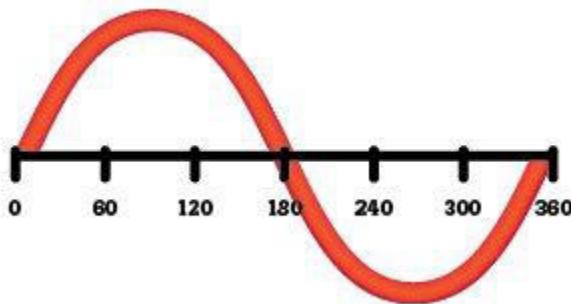


1. In this dc circuit, the switch is closed (turned ON).
2. Voltage in the power source—the "potential difference" between the battery's two poles—is activated, creating pressure that forces electrons to flow as current out the battery's negative terminal.
3. Current reaches the light, causing it to glow.
4. Current returns to the power source.

Voltage is either **alternating current (ac) voltage** or **direct current (dc) voltage**. Ways they differ:

Alternating current voltage (represented on a digital multimeter by \tilde{V}):

- Flows in evenly undulating waves, as shown below:



- Reverses direction at regular intervals.
- Commonly produced by utilities via **generators**, where mechanical energy—rotating motion powered by flowing water, steam, wind or heat—is converted to electrical energy.
- More common than dc voltage. Utilities deliver ac voltage to homes and businesses where the majority of devices use ac voltage.
- Primary voltage supplies vary by nation. In the United States, for example, it's 120 volts.

- Some household devices, such as TVs and computers, utilize dc voltage power. They use rectifiers (such as that chunky block in a laptop computer's cord) to convert ac voltage and current to dc.



Generators convert rotating motion into electricity. The rotary motion is commonly caused by flowing water (hydroelectric power) or steam from water heated by gas, oil, coal or nuclear power.

Direct current voltage (represented on a digital multimeter by \overline{V} and \overline{mV}):

- Travels in a straight line, and in one direction only.
- Commonly produced by sources of stored energy such as **batteries**.
- Sources of dc voltage have positive and negative terminals. Terminals establish polarity in a circuit, and polarity can be used to determine if a circuit is dc or ac.
- Commonly used in battery-powered portable equipment (autos, flashlights, cameras).

2.3 - Current

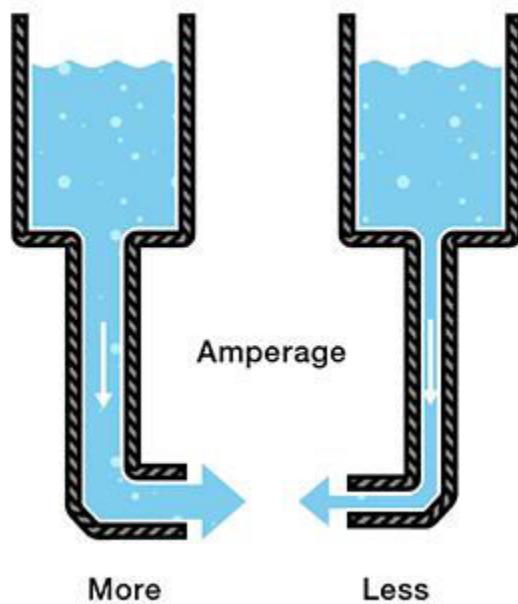
[reference : <https://learn.sparkfun.com/tutorials/voltage-current-resistance-and-ohms-law/all>]

Video : What is Current [<https://www.youtube.com/watch?v=kYwNj9uauJ4>]

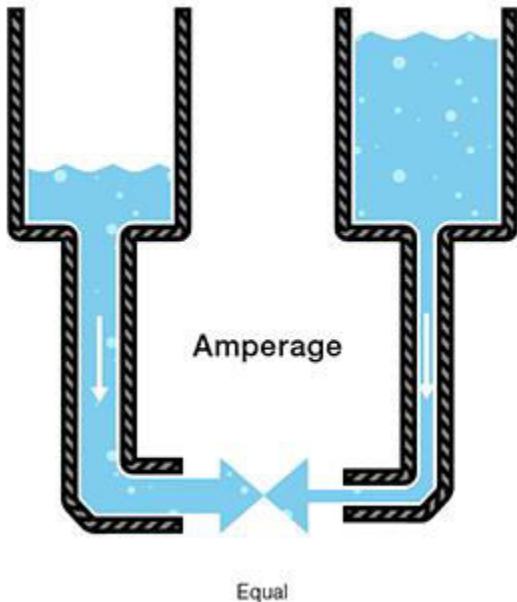
We can think of the amount of water flowing through the hose from the tank as current. The higher the pressure, the higher the flow, and vice-versa. With water, we would measure the volume of the water flowing through the hose over a certain period of time. With electricity, we measure the amount of charge flowing through the circuit over a period of time. Current is measured in Amperes (usually just referred to as "Amps"). An ampere is defined as

6.241×10^{18} electrons (1 Coulomb) per second passing through a point in a circuit. Amps are represented in equations by the letter "I".

Let's say now that we have two tanks, each with a hose coming from the bottom. Each tank has the exact same amount of water, but the hose on one tank is narrower than the hose on the other.



We measure the same amount of pressure at the end of either hose, but when the water begins to flow, the flow rate of the water in the tank with the narrower hose will be less than the flow rate of the water in the tank with the wider hose. In electrical terms, the current through the narrower hose is less than the current through the wider hose. If we want the flow to be the same through both hoses, we have to increase the amount of water (charge) in the tank with the narrower hose.



This increases the pressure (voltage) at the end of the narrower hose, pushing more water through the tank. This is analogous to an increase in voltage that causes an increase in current.

Now we're starting to see the relationship between voltage and current. But there is a third factor to be considered here: the width of the hose. In this analogy, the width of the hose is the resistance. This means we need to add another term to our model:

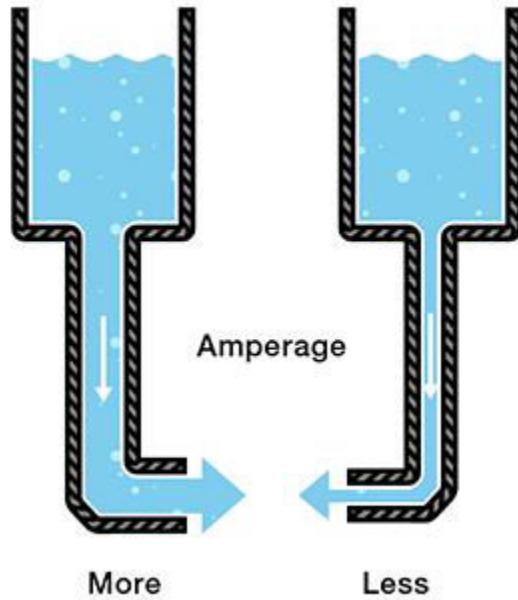
- Water = Charge (measured in Coulombs)
- Pressure = Voltage (measured in Volts)
- Flow = Current (measured in Amperes, or "Amps" for short)
- **Hose Width = Resistance**

2.4 - Resistance & Resistor

[reference - <https://learn.sparkfun.com/tutorials/voltage-current-resistance-and-ohms-law/all>]

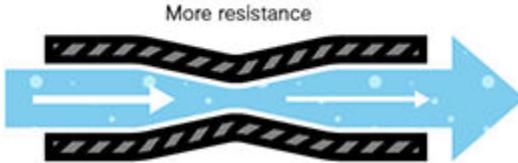
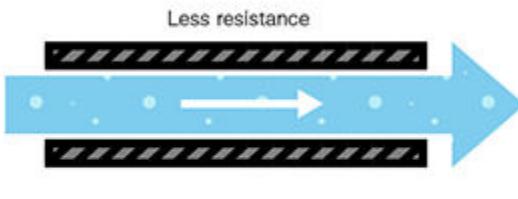
2.4.1 - Resistance

Consider again our two water tanks, one with a narrow pipe and one with a wide pipe.



It stands to reason that we can't fit as much volume through a narrow pipe than a wider one at the same pressure. This is resistance. The narrow pipe "resists" the flow of water through it even though the water is at the same pressure as the tank with the wider pipe.

Resistance



In electrical terms, this is represented by two circuits with equal voltages and different resistances. The circuit with the higher resistance will allow less charge to flow, meaning the circuit with higher resistance has less current flowing through it.

This brings us back to Georg Ohm. Ohm defines the unit of resistance of "1 Ohm" as the resistance between two points in a conductor where the application of 1 volt will push 1 ampere, or 6.241×10^{18} electrons. This value is usually represented in schematics with the greek letter " Ω ", which is called omega, and pronounced "ohm".

2.4.2 - Resistor

[reference : <https://learn.sparkfun.com/tutorials/resistors/all>]

2.4.2 - Resistor Basics

Resistors are electronic components which have a specific, never-changing electrical resistance. The resistor's resistance limits the flow of electrons through a circuit.

They are passive components, meaning they only consume power (and can't generate it). Resistors are usually added to circuits where they complement active components like op-amps, microcontrollers, and other integrated circuits. Commonly resistors are used to limit current, divide voltages, and pull-up I/O lines.

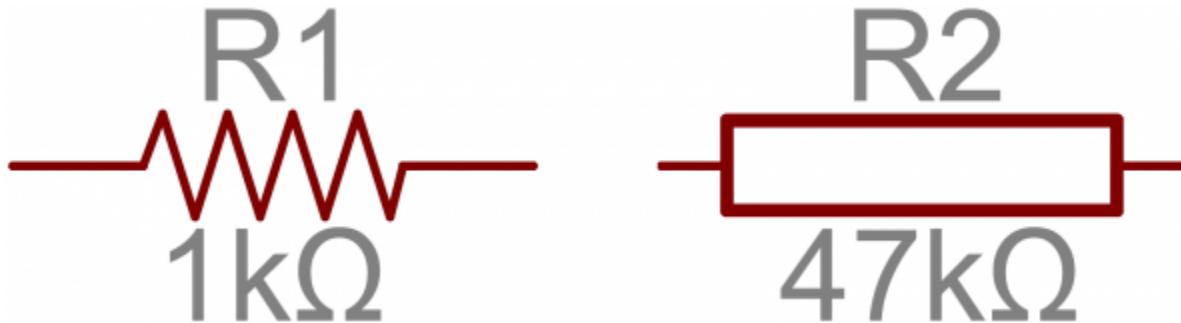
2.4.3 - Resistor units

The electrical resistance of a resistor is measured in **ohms**. The symbol for an ohm is the greek capital-omega: Ω . The (somewhat roundabout) definition of 1Ω is the resistance between two points where 1 volt (1V) of applied potential energy will push 1 ampere (1A) of current.

As SI units go, larger or smaller values of ohms can be matched with a prefix like kilo-, mega-, or giga-, to make large values easier to read. It's very common to see resistors in the kilohm ($k\Omega$) and megohm ($M\Omega$) range (much less common to see miliohm ($m\Omega$) resistors). For example, a $4,700\Omega$ resistor is equivalent to a $4.7k\Omega$ resistor, and a $5,600,000\Omega$ resistor can be written as $5,600k\Omega$ or (more commonly as) $5.6M\Omega$.

2.4.4 - Schematic symbol

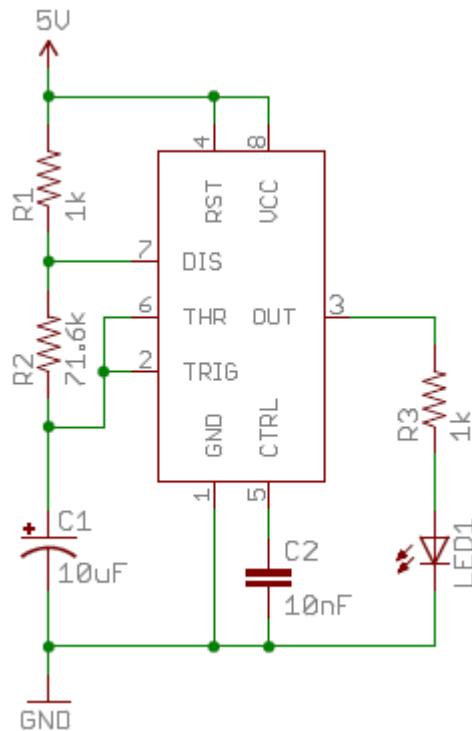
All resistors have **two terminals**, one connection on each end of the resistor. When modeled on a schematic, a resistor will show up as one of these two symbols:



Two common resistor schematic symbols. R1 is an American-style $1k\Omega$ resistor, and R2 is an international-style $47k\Omega$ resistor.

The terminals of the resistor are each of the lines extending from the squiggle (or rectangle). Those are what connect to the rest of the circuit.

The resistor circuit symbols are usually enhanced with both a resistance value and a name. The value, displayed in ohms, is obviously critical for both evaluating and actually constructing the circuit. The name of the resistor is usually an *R* preceding a number. Each resistor in a circuit should have a unique name/number. For example, here's a few resistors in action on a 555 timer circuit:



In this circuit, resistors play a key role in setting the frequency of the 555 timer's output. Another resistor (R3) limits the current through an LED.

2.4.5 - Types of Resistors

Resistors come in a variety of shapes and sizes. They might be through-hole or surface-mount. They might be a standard, static resistor, a pack of resistors, or a special variable resistor.

2.4.6 - Termination and mounting

Resistors will come in one of two termination-types: through-hole or surface-mount. These types of resistors are usually abbreviated as either PTH (plated through-hole) or SMD/SMT (surface-mount technology or device).

Through-hole resistors come with long, pliable leads which can be stuck into a [breadboard](#) or hand-soldered into a prototyping board or [printed circuit board \(PCB\)](#). These resistors are usually more useful in breadboarding, prototyping, or in any case where you'd rather not solder tiny, little 0.6mm-long SMD resistors. The long leads usually require trimming, and these resistors are bound to take up much more space than their surface-mount counterparts.

The most common through-hole resistors come in an axial package. The size of an axial resistor is relative to its power rating. A common $\frac{1}{2}W$ resistor measures about 9.2mm across, while a smaller $\frac{1}{4}W$ resistor is about 6.3mm long.



A half-watt ($\frac{1}{2}W$) resistor (above) sized up to a quarter-watt ($\frac{1}{4}W$).

Surface-mount resistors are usually tiny black rectangles, terminated on either side with even smaller, shiny, silver, conductive edges. These resistors are intended to sit on top of PCBs, where they're soldered onto mating landing pads. Because these resistors are so small, they're usually set into place by a [robot](#), and sent through an oven where solder melts and holds them in place.

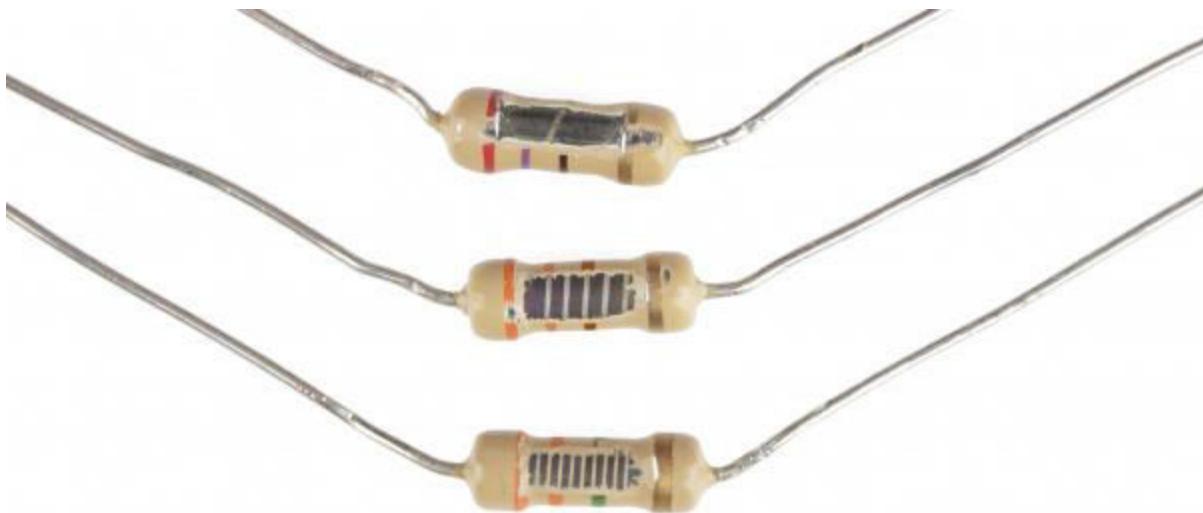


A tiny 0603 330Ω resistor hovering over shiny George Washington's nose on top of a [U.S. quarter]([http://en.wikipedia.org/wiki/Quarter_\(United_States_coin\)](http://en.wikipedia.org/wiki/Quarter_(United_States_coin))).

SMD resistors come in standardized sizes; usually either 0805 (0.8mm long by 0.5mm wide), 0603, or 0402. They're great for mass circuit-board-production, or in designs where space is a precious commodity. They take a steady, precise hand to manually solder, though!

2.4.7 - Resistor composition

Resistors can be constructed out of a variety of materials. Most common, modern resistors are made out of either a **carbon, metal, or metal-oxide film**. In these resistors, a thin film of conductive (though still resistive) material is wrapped in a helix around and covered by an insulating material. Most of the standard, no-frills, through-hole resistors will come in a carbon-film or metal-film composition.



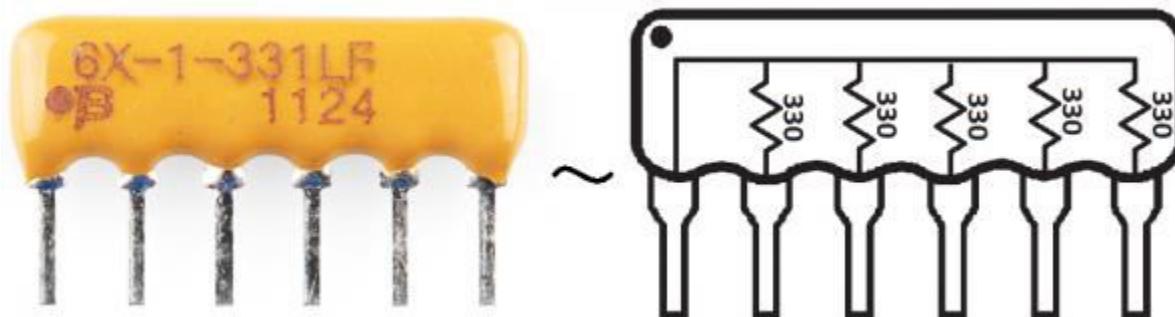
Peek inside the guts of a few carbon-film resistors. Resistance values from top to bottom: 27Ω, 330Ω and a 3.3MΩ. Inside the resistor, a carbon film is wrapped around an insulator. More wraps means a higher resistance. Pretty neat!

Other through-hole resistors might be wirewound or made of super-thin metallic foil. These resistors are usually more expensive, higher-end components specifically chosen for their unique characteristics like a higher power-rating, or maximum temperature range.

Surface-mount resistors are usually either **thick or thin-film** variety. Thick-film is usually cheaper but less precise than thin. In both resistor types, a small film of resistive metal alloy is sandwiched between a ceramic base and glass/epoxy coating, and then connected to the terminating conductive edges.

2.4.8 - Special resistor packages

There are a variety of other, special-purpose resistors out there. Resistors may come in pre-wired packs of five-or-so **resistor arrays**. Resistors in these arrays may share a common pin, or be set up as voltage dividers.



An array of five 330Ω resistors, all tied together at one end.

Resistors don't have to be static either. Variable resistors, known as **rheostats**, are resistors which can be adjusted between a specific range of values. Similar to the rheostat is the **potentiometer**. Pots connect two resistors internally, in series, and adjust a center tap between

them creating an adjustable [voltage divider](#). These variable resistors are often used for inputs, like volume knobs, which need to be adjustable.



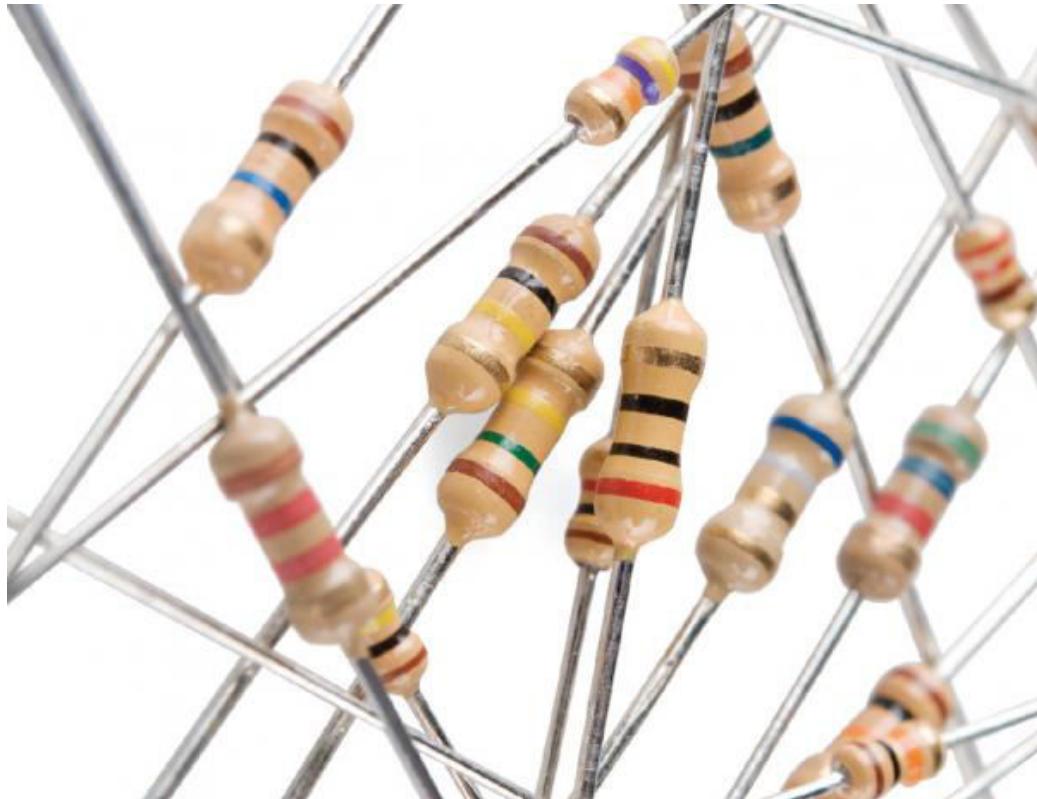
A smattering of potentiometers. From top-left, clockwise: [a standard 10k trimpot](#), [2-axis joystick](#), [softpot](#), [slide pot](#), [classic right-angle](#), and a [breadboard friendly 10k trimpot](#).

2.4.9 - Decoding Resistor Markings

Though they may not display their value outright, most resistors are marked to show what their resistance is. PTH resistors use a color-coding system (which really adds some flair to circuits), and SMD resistors have their own value-marking system.

2.4.10 - Decoding the Color Bands (College Level)

Through-hole, axial resistors usually use the color-band system to display their value. Most of these resistors will have four bands of color circling the resistor, though you will also find five band and six band resistors.



2.4.10.1 - Four Band Resistors

In the standard four band resistors, the first two bands indicate the **two most-significant digits** of the resistor's value. The third band is a weight value, which **multiplies** the two significant digits by a power of ten.

The final band indicates the **tolerance** of the resistor. The tolerance explains how much more or less the *actual* resistance of the resistor can be compared to what its nominal value is. No resistor is made to perfection, and different manufacturing processes will result in better or worse tolerances. For example, a $1\text{k}\Omega$ resistor with 5% tolerance could actually be anywhere between $0.95\text{k}\Omega$ and $1.05\text{k}\Omega$.

How do you tell which band is first and last? The last, tolerance band is often clearly separated from the value bands, and usually it'll either be silver or gold.

2.4.10.2 - Five and Six Band Resistors

Five band resistors have a third significant digit band between the first two bands and the **multiplier band**. Five band resistors also have a wider range of tolerances available.

Six band resistors are basically five band resistors with an additional band at the end that indicates the temperature coefficient. This indicates the expected change in resistor value as

the temperature changes in degrees Celsius. Generally these temperature coefficient values are extremely small, in the ppm range.

2.4.11 - Decoding Resistor Color Bands (College Level)

When decoding the resistor color bands, consult a resistor color code table like the one below. For the first two bands, find that color's corresponding digit value. The $4.7\text{k}\Omega$ resistor shown here has color bands of **yellow** and **violet** to begin - which have digit values of 4 and 7 (47). The third band of the $4.7\text{k}\Omega$ is **red**, which indicates that the 47 should be multiplied by 102 (or 100). 47 times 100 is 4,700!



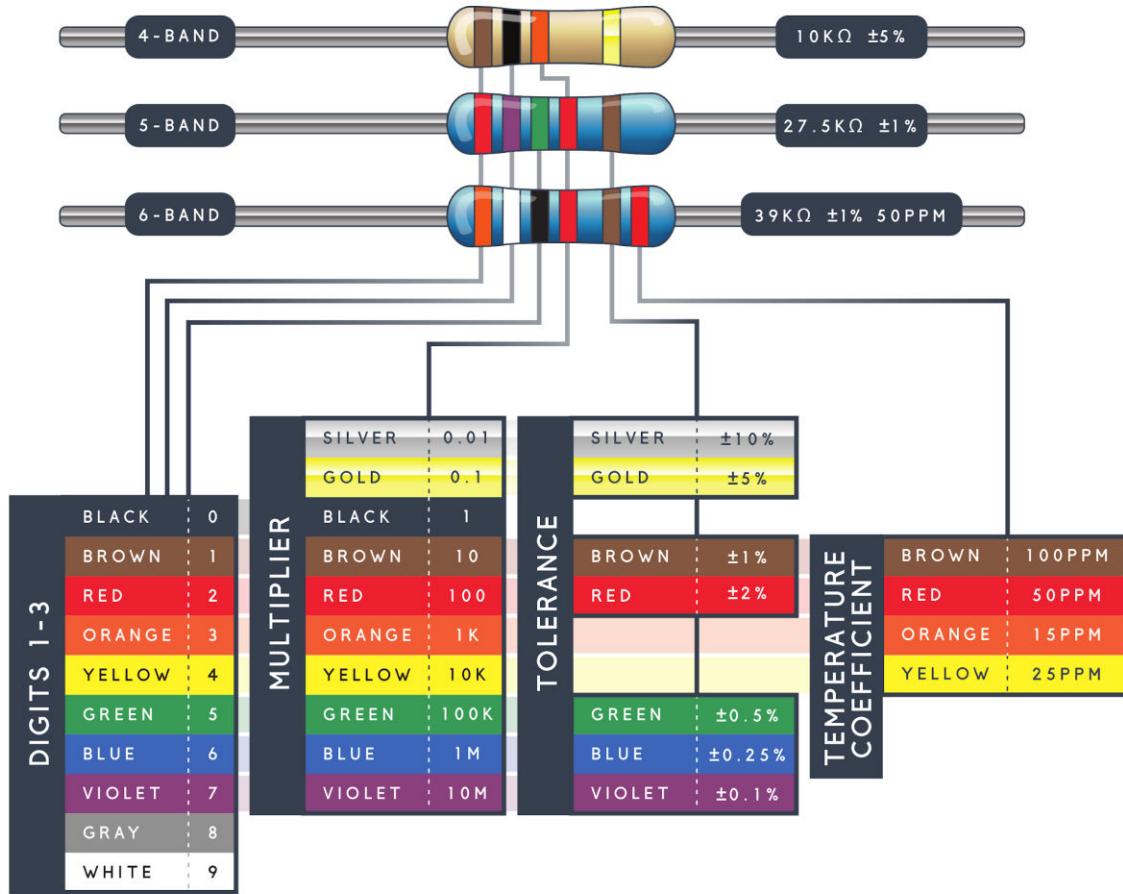
4.7kΩ resistor with four color bands

If you're trying to commit the color band code to memory, a mnemonic device might help. There are a handful of (sometimes unsavory) mnemonics out there to help remember the resistor color code. A good one, which spells out the difference between *black* and *brown* is:

"Big brown rabbits often yield great big vocal groans when gingerly snapped."

Or, if you remember "ROY G. BIV", subtract the *indigo* (poor indigo, no one remembers indigo), and add black and brown to the front and gray and white to the back of the classic rainbow color-order.

2.4.12 - Resistor Color Code Table (College Level)



2.4.13 - Decoding Surface-Mount Markings (College Level)

SMD resistors, like those in 0603 or 0805 packages, have their own way of displaying their value. There are a few common marking methods you'll see on these resistors. They'll usually have three to four characters -- numbers or letters -- printed on top of the case.

If the three characters you're seeing are *all numbers*, you're probably looking at an E24 marked resistor. These markings actually share some similarity with the color-band system used on the PTH resistors. The first two numbers represent the first two most-significant digits of the value, the last number represents a magnitude



In the above example picture, resistors are marked 104, 105, 205, 751, and 754. The resistor marked with 104 should be 100kΩ (10x104), 105 would be 1MΩ (10x105), and 205 is 2MΩ (20x105). 751 is 750Ω (75x101), and 754 is 750kΩ (75x104).

Another common coding system is E96, and it's the most cryptic of the bunch. E96 resistors will be marked with three characters -- two numbers at the beginning and a letter at the end. The two numbers tell you the first three digits of the value, by corresponding to one of the not-so-obvious values on this lookup table.

| CodeValue |
|-----------|-----------|-----------|-----------|-----------|-----------|-----------|
| 01 | 100 | 17 | 147 | 33 | 215 | 49 |
| 02 | 102 | 18 | 150 | 34 | 221 | 50 |
| 03 | 105 | 19 | 154 | 35 | 226 | 51 |
| 04 | 107 | 20 | 158 | 36 | 232 | 52 |
| 05 | 110 | 21 | 162 | 37 | 237 | 53 |
| 06 | 113 | 22 | 165 | 38 | 243 | 54 |
| 07 | 115 | 23 | 169 | 39 | 249 | 55 |
| 08 | 118 | 24 | 174 | 40 | 255 | 56 |
| 09 | 121 | 25 | 178 | 41 | 261 | 57 |
| 10 | 124 | 26 | 182 | 42 | 267 | 58 |
| 11 | 127 | 27 | 187 | 43 | 274 | 59 |
| 12 | 130 | 28 | 191 | 44 | 280 | 60 |
| 13 | 133 | 29 | 196 | 45 | 287 | 61 |
| 14 | 137 | 30 | 200 | 46 | 294 | 62 |
| 15 | 140 | 31 | 205 | 47 | 301 | 63 |
| 16 | 143 | 32 | 210 | 48 | 309 | 64 |

The letter at the end represents a multiplier, matching up to something on this table:

LetterMultiplier	LetterMultiplier	LetterMultiplier			
Z	0.001	A	1	D	1000
Y or R	0.01	B or H	10	E	10000
X or S	0.1	C	100	F	100000



So a 01C resistor is our good friend, 10kΩ (100x100), 01B is 1kΩ (100x10), and 01D is 100kΩ. Those are easy, other codes may not be. 85A from the picture above is 750Ω (750x1) and 30C is actually 20kΩ.

2.4.14 - Power Rating (College Level)

The power rating of a resistor is one of the more hidden values. Nevertheless it can be important, and it's a topic that'll come up when selecting a resistor type.

Power is the rate at which energy is transformed into something else. It's calculated by multiplying the voltage difference across two points by the current running between them, and is measured in units of a watt (W). Light bulbs, for example, power electricity into light. But a resistor can only turn electrical energy running through it into **heat**. Heat isn't usually a nice playmate with electronics; too much heat leads to smoke, sparks, and fire!

Every resistor has a specific maximum power rating. In order to keep the resistor from heating up too much, it's important to make sure the power across a resistor is kept under its maximum rating. The power rating of a resistor is measured in watts, and it's usually somewhere between $\frac{1}{8}W$ (0.125W) and 1W. Resistors with power ratings of more than 1W are usually referred to as power resistors, and are used specifically for their power dissipating abilities.

2.4.14.1 - Finding a resistor's power rating (College Level)

A resistor's power rating can usually be deduced by observing its package size. Standard through-hole resistors usually come with $\frac{1}{4}W$ or $\frac{1}{2}W$ ratings. More special purpose, power resistors might actually list their power rating on the resistor.



These power resistors can handle a lot more power before they blow. From top-right to bottom-left there are examples of 25W, 5W and 3W resistors, with values of 2Ω, 3Ω 0.1Ω and 22kΩ. Smaller power-resistors are often used to sense current.

The power ratings of surface mount resistors can usually be judged by their size as well. Both 0402 and 0603-size resistors are usually rated for 1/16W, and 0805's can take 1/10W.

2.4.14.2 - Measuring power across a resistor (College Level)

Power is usually calculated by multiplying voltage and current ($P = IV$). But, by applying Ohm's law, we can also use the resistance value in calculating power. If we know the current running through a resistor, we can calculate the power as:

$$P = I^2 \cdot R$$

Or, if we know the voltage across a resistor, the power can be calculated as:

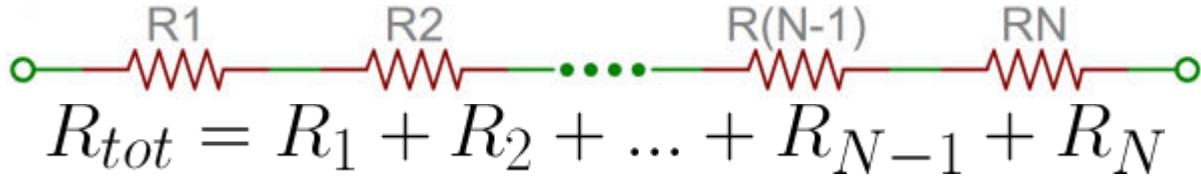
$$P = \frac{V^2}{R}$$

2.4.15 - Series and Parallel Resistors

Resistors are paired together all the time in electronics, usually in either a series or parallel circuit. When resistors are combined in series or parallel, they create a **total resistance**, which can be calculated using one of two equations. Knowing how resistor values combine comes in handy if you need to create a specific resistor value.

2.4.15.1 - Series resistors

When connected in series resistor values simply add up.

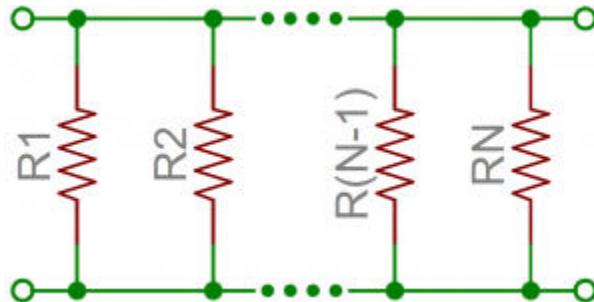


N resistors in series. The total resistance is the sum of all series resistors.

So, for example, if you just *have to have* a $12.33\text{k}\Omega$ resistor, seek out some of the more common resistor values of $12\text{k}\Omega$ and 330Ω , and butt them up together in series.

2.4.15.2 - Parallel resistors

Finding the resistance of resistors in parallel isn't quite so easy. The total resistance of N resistors in parallel is the inverse of the sum of all inverse resistances. This equation might make more sense than that last sentence:



$$\frac{1}{R_{tot}} = \frac{1}{R_1} + \frac{1}{R_2} + \dots + \frac{1}{R_{N-1}} + \frac{1}{R_N}$$

N resistors in parallel. To find the total resistance, invert each resistance value, add them up, and then invert that.

(The inverse of resistance is actually called **conductance**, so put more succinctly: the conductance of parallel resistors is the sum of each of their conductances).

As a special case of this equation: if you have **just two** resistors in parallel, their total resistance can be calculated with this slightly-less-inverted equation:

$$R_{tot} = \frac{R_1 \cdot R_2}{R_1 + R_2}$$

As an even *more special* case of that equation, if you have two parallel resistors of **equal value** the total resistance is half of their value. For example, if two 10kΩ resistors are in parallel, their total resistance is 5kΩ.

A shorthand way of saying two resistors are in parallel is by using the parallel operator: **||**. For example, if R1 is in parallel with R2, the conceptual equation could be written as R1||R2. Much cleaner, and hides all those nasty fractions!

2.5 - Power Supply / Adapter / Voltage Regulator / Arduino 5V

[reference - <https://playground.arduino.cc/Learning/WhatAdapter/>

<https://www.technobYTE.org/2016/07/power-up-the-arduino-uno/>

<https://www.instructables.com/id/Power-Supply-For-Arduino-power-and-breadboard/>

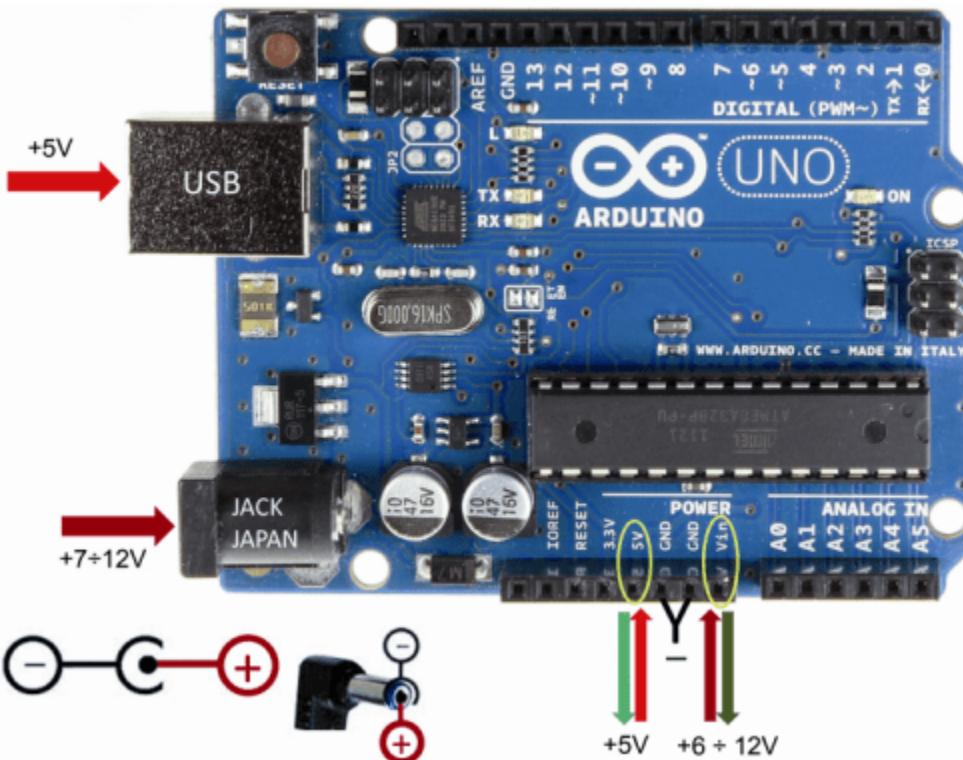
<https://randomnerdtutorials.com/arduino-5-ways-to-power-up-your-arduino/>

Video : Arduino 101 : Four ways to switch on an Arduino Uno

[\[https://www.youtube.com/watch?time_continue=50&v=YnyeCix9Whs\]](https://www.youtube.com/watch?time_continue=50&v=YnyeCix9Whs)

In every arduino there is a microcontroller . Most of the microcontrollers used in arduino run with 5V or some cases 3.3V(e.g. arduino due) . If we apply more than 5V microcontroller will be damaged. Now question is how we can run arduino with adapter or power supply. Its simple arduino has a adapter port where we can apply 9V to 12V . Onboard voltage regulator convert it to 5V . If we use barebone microcontroller we must need a voltage converter for desired voltage. We can use lab power supply or a adapter . We can make voltage regulator using many conventional IC(integrated circuit) . 7805, LM2576 etc can be used for voltage level conversion.

Let's say we have a 12V battery . Now if we want to use this battery with microcontroller we have to step it down to 5V using voltage regulator.



in this picture we can see arduino has a power jack where we can apply 7V to 12V . Arduino has 5V output which we can use to operate our external circuit or device(remember if current consumption of external circuit is law otherwise we will need external power source with enough current rating according to external device which we are going to operate). Vin pin is connected with DC jack so don't assume it's a 5V supply rather its usually 0.7V lower then input jack.

You can use low cost buck converter module for powering arduino or barebone microcontroller.

video : LM2596 Buck Module [<https://www.youtube.com/watch?v=liLFq0xM7xE>]



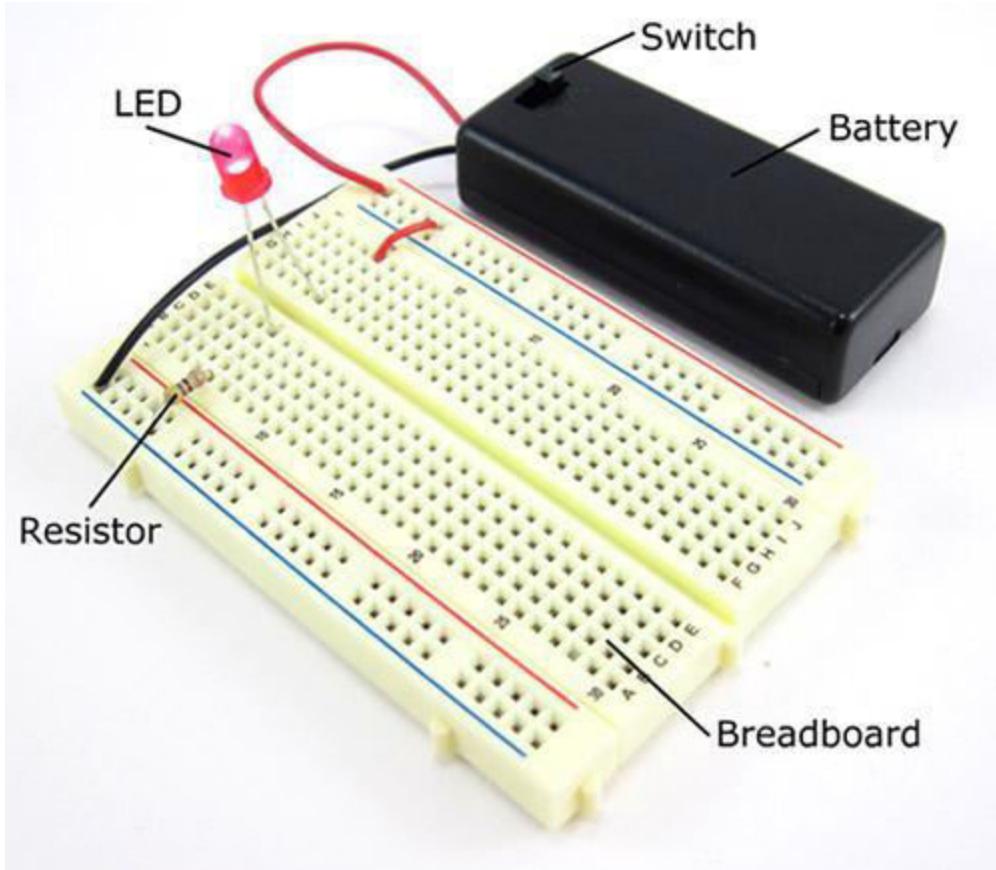
2.6 - Breadboard

[reference -<https://www.sciencebuddies.org/science-fair-projects/references/how-to-use-a-breadboard>]

video : How to use a breadboard

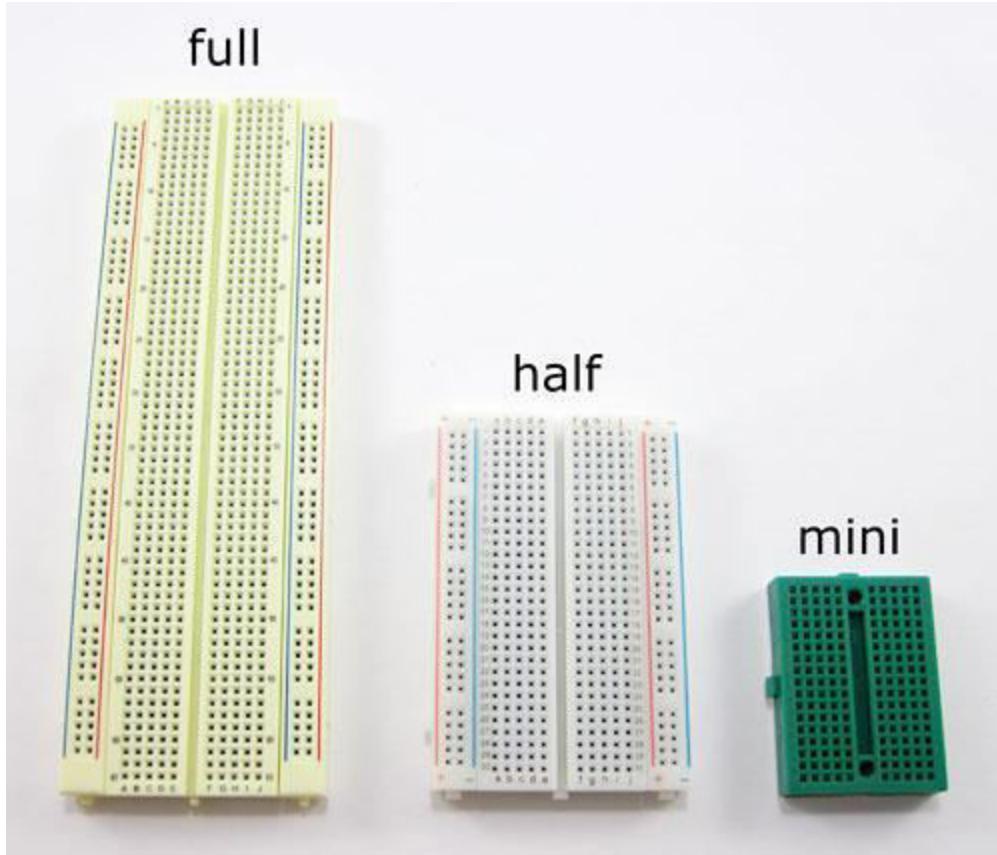
[https://www.youtube.com/watch?time_continue=198&v=6WReFkfrUIk]

A breadboard is a rectangular plastic board with a bunch of tiny holes in it. These holes let you easily insert electronic components to **prototype** (meaning to build and test an early version of) an electronic circuit, like this one with a battery, switch, resistor, and an LED (light-emitting diode).

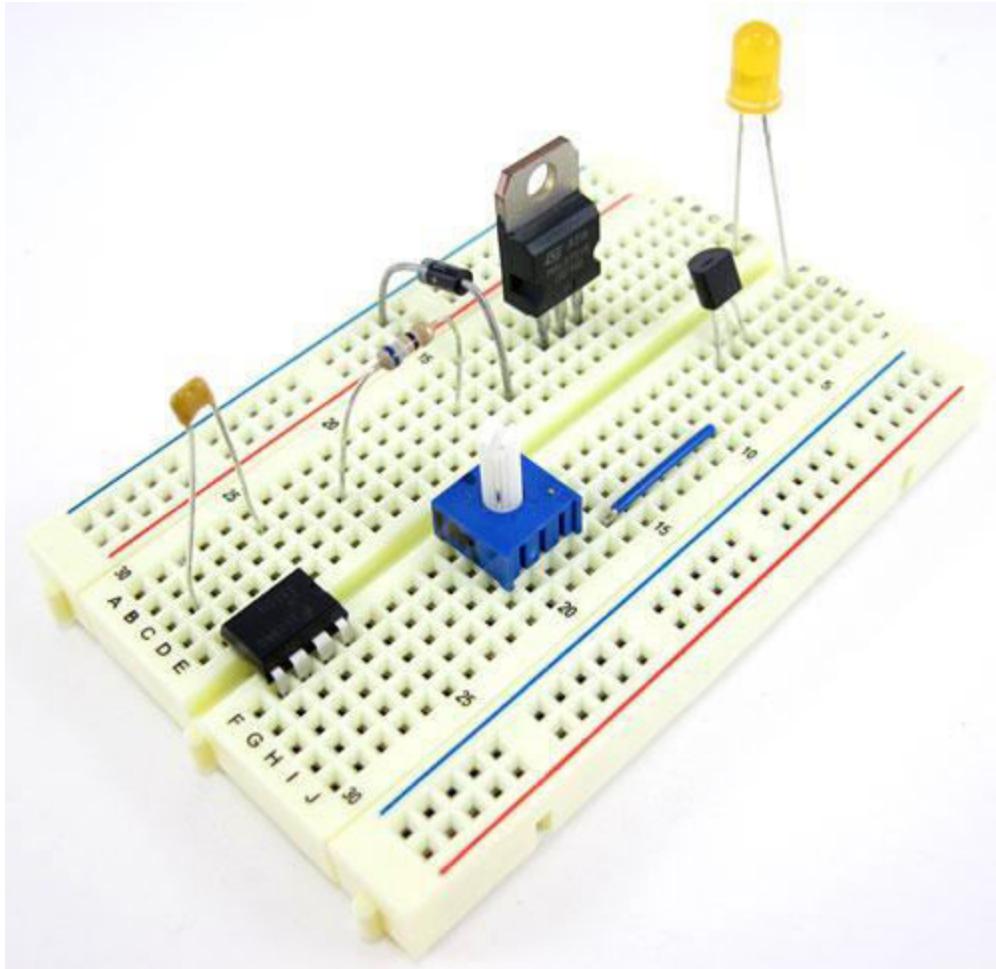


The connections are not permanent, so it is easy to *remove* a component if you make a mistake, or just start over and do a new project. This makes breadboards great for beginners who are new to electronics. You can use breadboards to make all sorts of fun electronics projects, from different types of robots or an electronic drum set, to an electronic rain detector to help conserve water in a garden, just to name a few.

Modern breadboards are made from plastic, and come in all shapes, sizes, and even different colors. While larger and smaller sizes are available, the most common sizes you will probably see are "full-size," "half-size," and "mini" breadboards. Most breadboards also come with tabs and notches on the sides that allow you to snap multiple boards together. However, a single half-sized breadboard is sufficient for many beginner-level projects.



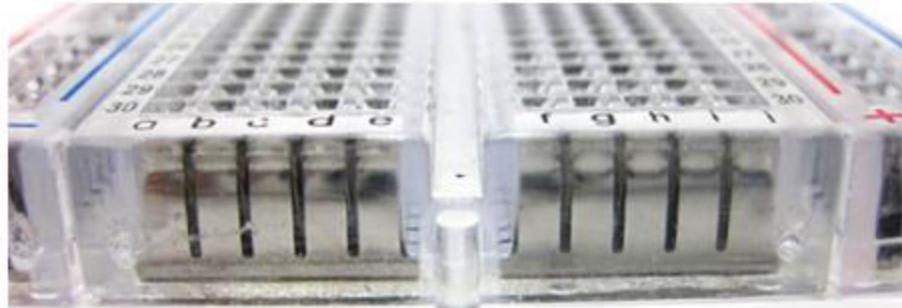
Breadboards are designed so you can push these leads into the holes. They will be held in place snugly enough that they will not fall out (even if you turn the breadboard upside-down), but lightly enough that you can easily pull on them to remove them.



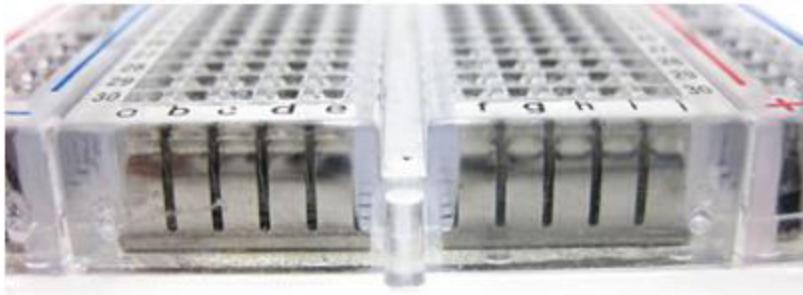
The leads can fit into the breadboard because the *inside* of a breadboard is made up of rows of tiny metal clips. This is what the clips look like when they are removed from a breadboard.



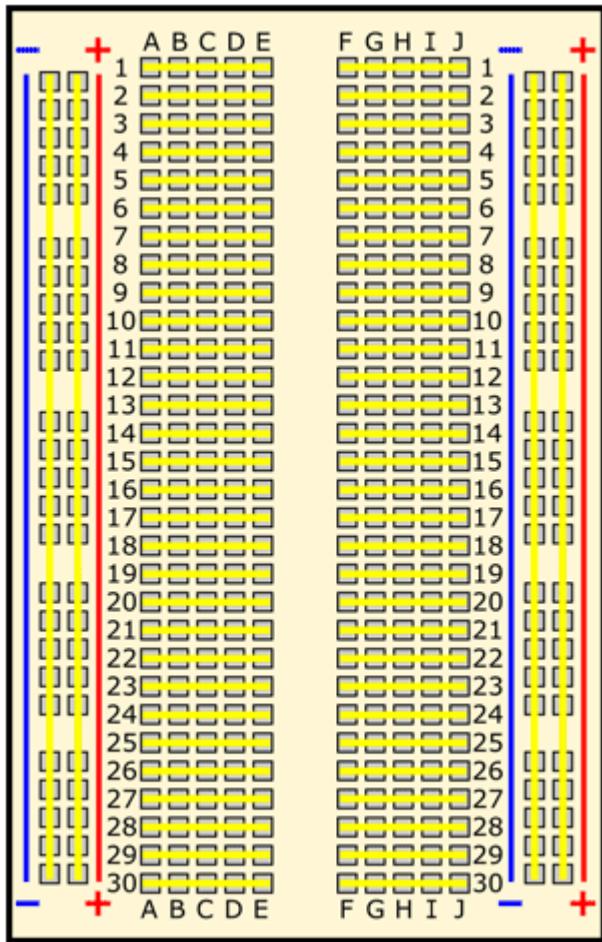
are actually made of transparent plastic, so you can see the clips inside.



Most breadboards have a backing layer that prevents the metal clips from falling out. The backing is typically a layer of sticky, double-sided tape covered by a protective layer of paper. If you want to permanently "stick" the breadboard to something (for example, a robot), you just need to peel off the paper layer to expose the sticky tape underneath. In this picture, the breadboard on the right has had its backing removed completely (so you can see all the metal clips). The breadboard on the left still has its sticky backing, with one corner of the paper layer peeled up.



Remember that the inside of the breadboard is made up of sets of five metal clips. This means that each set of five holes forming a half-row (columns A–E or columns F–J) is electrically connected. For example, that means hole A1 is electrically connected to holes B1, C1, D1, and E1. It is *not* connected to hole A2, because that hole is in a different row, with a separate set of metal clips. It is also *not* connected to holes F1, G1, H1, I1, or J1, because they are on the other "half" of the breadboard—the clips are not connected across the gap in the middle .Unlike all the main breadboard rows, which are connected in sets of five holes, the buses typically run the entire length of the breadboard (but there are some exceptions). This image shows which holes are electrically connected in a typical half-sized breadboard, highlighted in yellow lines.

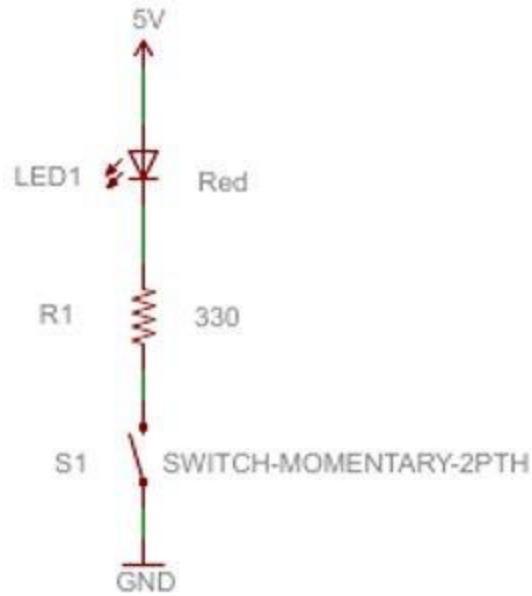


2.7 - LED Connection & On/Off

[reference - <https://learn.sparkfun.com/tutorials/how-to-use-a-breadboard/all>
<https://www.sciencebuddies.org/science-fair-projects/references/how-to-use-a-breadboard>]

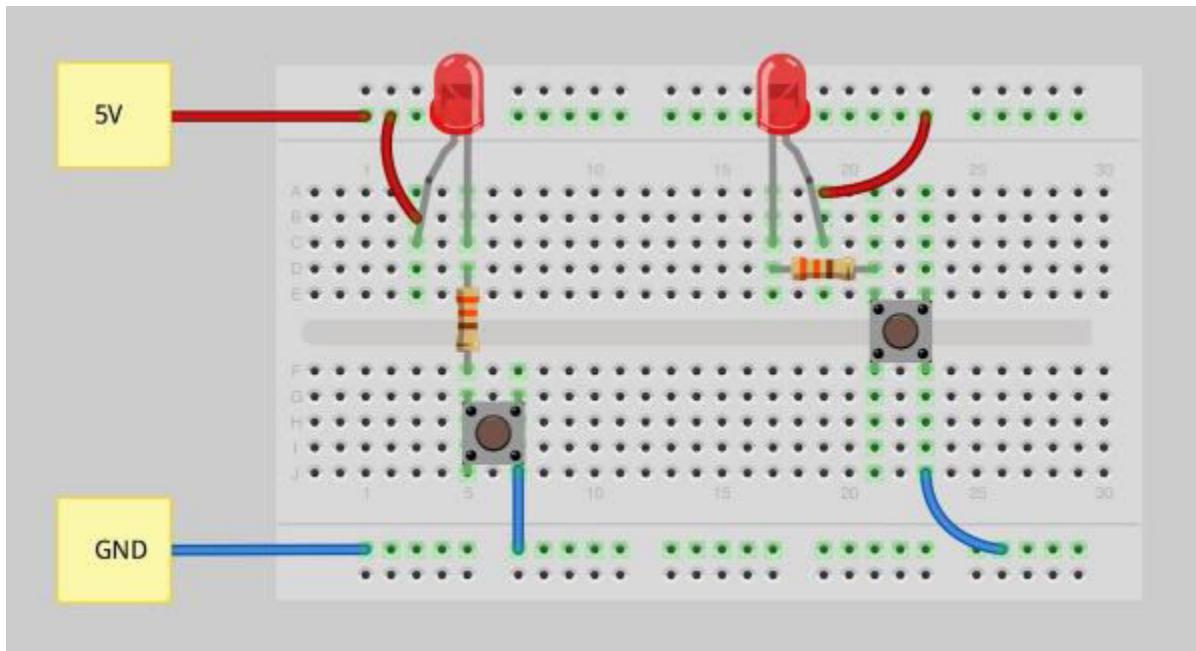
Schematics are universal pictograms that allow people all over the world to understand and build electronics. Every electronic component has a very unique schematic symbol. These symbols are then assembled into circuits using a variety of programs. You could also draw them out by hand. If you want to dive deeper in the world of electronics and circuit building, learning to read schematics is a very important step in doing so.

Here we have a schematic for the above circuit. Power (assuming the switch is flipped to the 5V side) is represented by the arrow at the top. It then goes to the LED (the triangle and line with arrows emitting out of it). The LED is then connected to the resistor (the squiggly line). That is connected to the button (the latch-looking symbol). Last the button is connect to ground (the horizontal line at the bottom). Every LED has certain current threshold so we must have to limits its current flow. Resistor limits the current flowing through LED. If we don't use resistor LED might have been burnt .

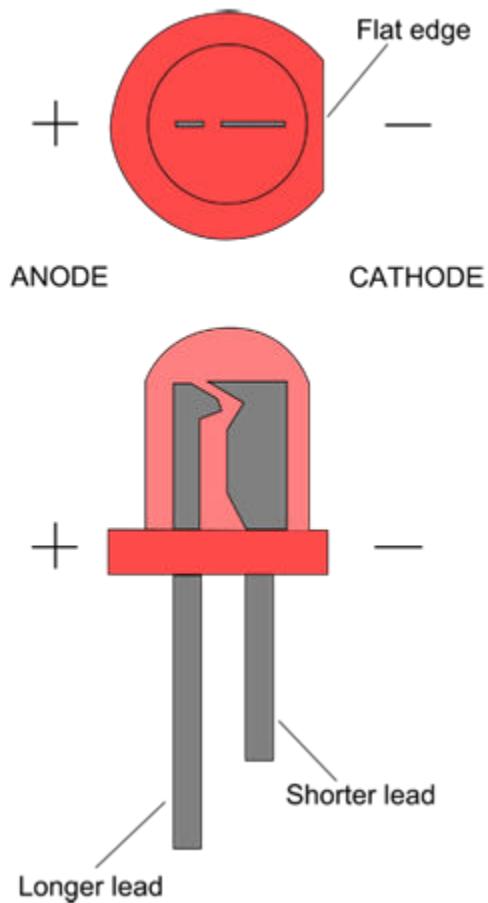


Schematics allow people from different nationalities and languages to build and collaborate on circuits designed by anyone. As mentioned, you can *build* a circuit in many different ways, but, as this schematic shows, there are certain connections that must be made.

Now to on and off a led we need to mount LED, button, resistor , battery in a breadboard.



LEDs have a positive side (called the anode) and a negative side (called the cathode). The metal lead for the anode is longer than the lead for the cathode. The cathode side also usually has a flat edge on the plastic part of the LED.

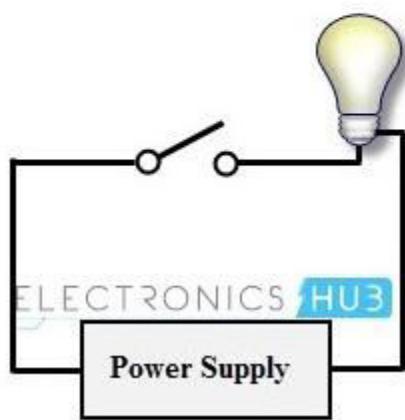
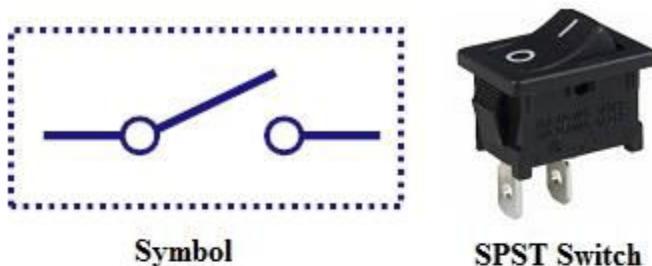


2.8 - Various switch used in Electronics

[reference - <https://www.electronicshub.org-switches/>
<https://www.allaboutcircuits.com/textbook/digital/chpt-4/switch-types/>
https://www.electronics-notes.com/articles/electronic_components/switches-relays/electronics-switches.php]

2.8.1 - Single Pole Single Throw Switch (SPST)

This is the basic ON and OFF switch consisting of one input contact and one output contact. It switches a single circuit and it can either make (ON) or break (OFF) the load. The contacts of SPST can be either normally open or normally closed configurations .



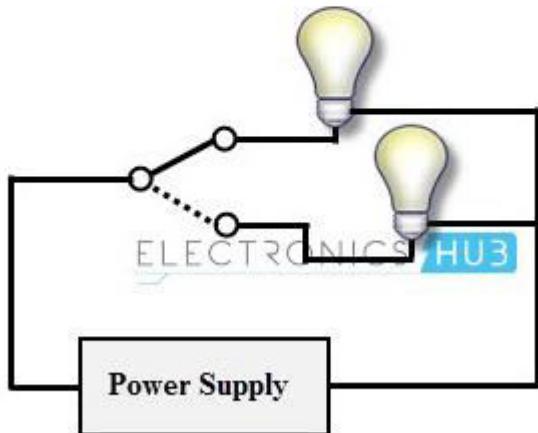
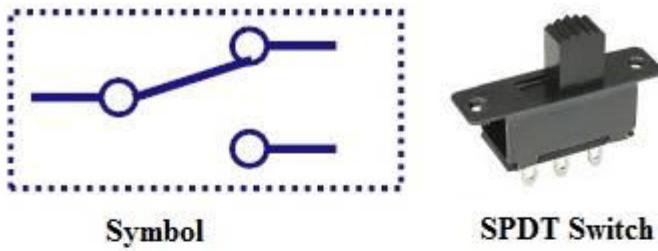
2.8.2 - Single Pole Double Throw Switch (SPDT)

This switch has three terminals, one is input contact and remaining two are output contacts.

This means it consists two ON positions and one OFF position.

In most of the circuits, these switches are used as changeover to connect the input between two choices of outputs.

The contact which is connected to the input by default is referred as normally closed contact and contact which will be connected during ON operation is a normally open contact.



SPDT Switch Circuit

2.8.3 -Double Pole Single Throw Switch (DPST)

This switch consists of four terminals, two input contacts and two output contacts.

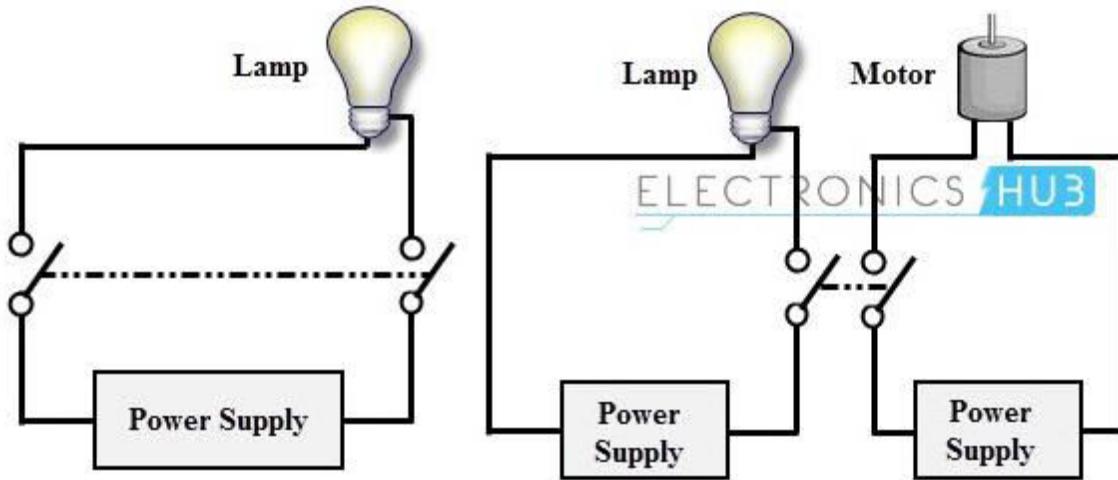
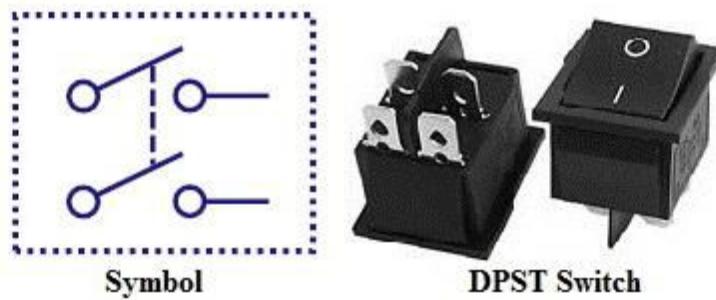
It behaves like a two separate SPST configurations, operating at the same time.

It has only one ON position, but it can actuate the two contacts simultaneously, such that each input contact will be connected to its corresponding output contact.

In OFF position both switches are at open state.

This type of switches is used for controlling two different circuits at a time.

Also, the contacts of this switch may be either normally open or normally closed configurations.



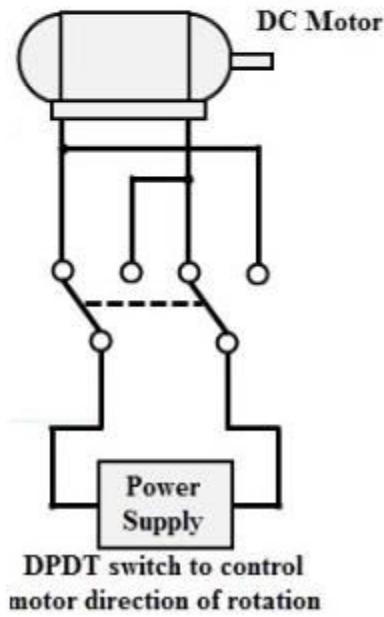
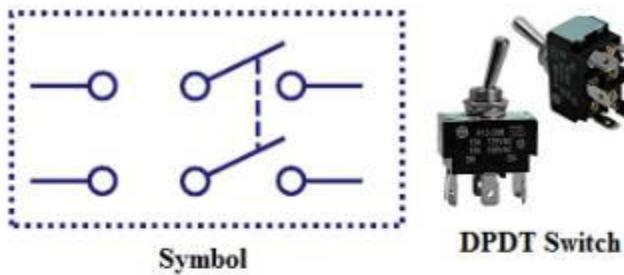
2.8.4 - Double Pole Double Throw Switch (DPDT)

This is a dual ON/OFF switch consisting of two ON positions.

It has six terminals, two are input contacts and remaining four are the output contacts.

It behaves like a two separate SPDT configuration, operating at the same time.

Two input contacts are connected to the one set of output contacts in one position and in another position, input contacts are connected to the other set of output contacts.



2.8.5 - Push Button Switch

It is a momentary contact switch that makes or breaks connection as long as pressure is applied (or when the button is pushed).

Generally, this pressure is supplied by a button pressed by someone's finger.

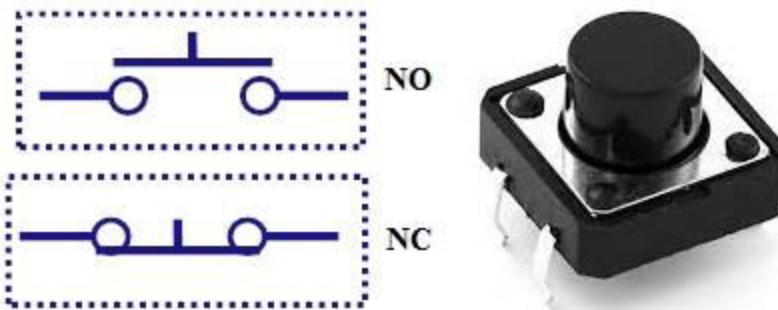
This button returns its normal position, once the pressure is removed.

The internal spring mechanism operates these two states (pressed and released) of a push button.

It consists of stationary and movable contacts, of which stationary contacts are connected in series with the circuit to be switched while movable contacts are attached with a push button.

Push buttons are majorly classified into normally open, normally closed and double acting push buttons as shown in the above figure.

Double acting push buttons are generally used for controlling two electrical circuits.



2.8.6 - Toggle Switch

A toggle switch is manually actuated (or pushed up or down) by a mechanical handle, lever or rocking mechanism. These are commonly used as light control switches.

Most of these switches come with two or more lever positions which are in the versions of SPDT, SPST, DPST and DPDT switch. These are used for switching high currents (as high as 10 A) and can also be used for switching small currents.

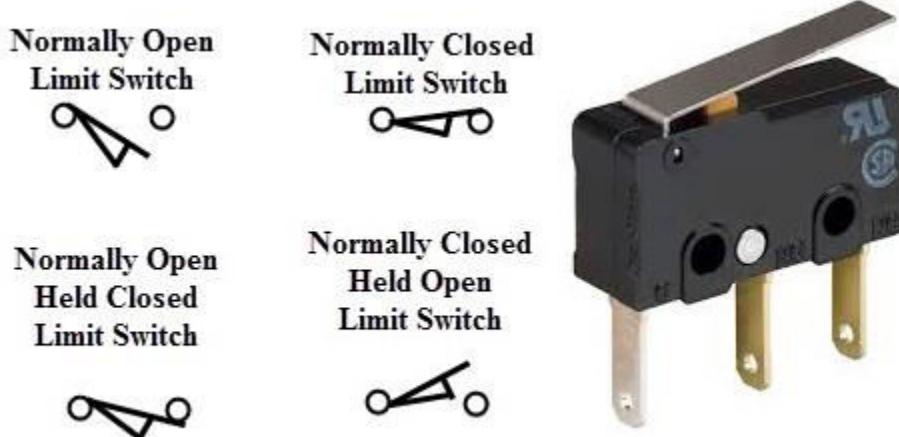
These are available in different ratings, sizes and styles and are used for different type of applications. The ON condition can be any of their level positions, however, by convention the downward is the closed or ON position.

2.8.7 - Limit Switch

The control schemes of a limit switch are shown in above figure , in which four varieties of limit switches are presented.

Some switches are operated by the presence of an object or by the absence of objects or by the motion of machine instead of human hand operation. These switches are called as limit switches.

These switches consist of a bumper type of arm actuated by an object. When this bumper arm is actuated, it causes the switch contacts to change position.



Limit Switch

2.8.8 - Float Switches

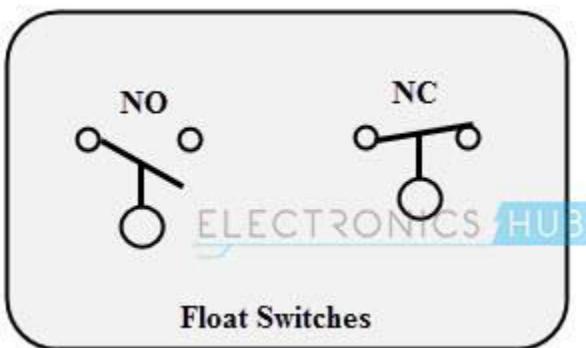
Float switches are mainly used for controlling DC and AC motor pumps according to the liquid or water in a tank or sump.

This switch is operated when the float (or floating object) moves downward or upward based on water level in a tank.

This float movement of rod or chain assembly and counterweight causes to open or close electrical contacts. Another form of float switch is the mercury bulb type switch that does not consist of any float rod or chain arrangement.

This bulb consist of mercury contacts such that when the liquid level rises or falls, the state of contacts also changes.

The ball float switch symbol is shown in the above figure. These float switches can be normally open or normally closed type.



2.8.9 - Flow Switches

These are mainly used to detect the movement of liquid or air flow through a pipe or duct. The air flow switch (or a micro switch) is constructed by a snap-action.

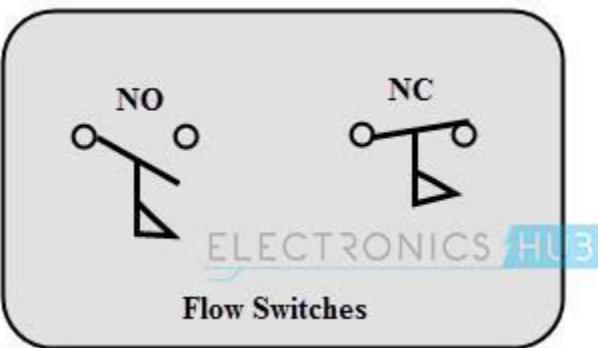
This micro switch is attached to a metal arm .To this metal arm, a thin plastic or metal piece is connected.

When a large amount of air passes through the metal or plastic piece, it causes the movement of metal arm and thus operates the contacts of the switch.

Liquid flow switches are designed with a paddle that inserted across the flow of liquid in a pipe. When liquid flows through the pipe, force exerted against the paddle changes the position of the contacts.

The above figure shows the switch symbol used for both air flow and liquid flow. The flag symbol on the switch indicates the paddle which senses the flow or movement of liquid.

These switches again normally open or normally closed type configurations.



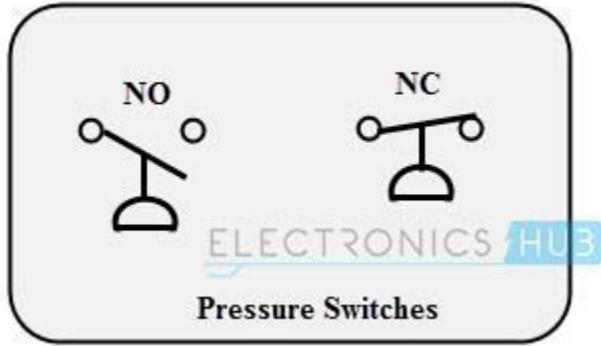
2.8.10 - Pressure Switches

These switches are commonly used in industrial applications in order to sense the pressure of hydraulic systems and pneumatic devices.

Depends on the range of pressure to be measured, these pressure switches are classified into diaphragm operated pressure switch, metal bellow type pressure switch and piston type pressure switch.

In all these types, pressure detection element operates a set of contacts (which can be either double pole or single pole contacts).

This switch symbol consist a half-circle connected to a line in which flat part indicates a diaphragm. These switches may be either normally open or normally closed type configurations.



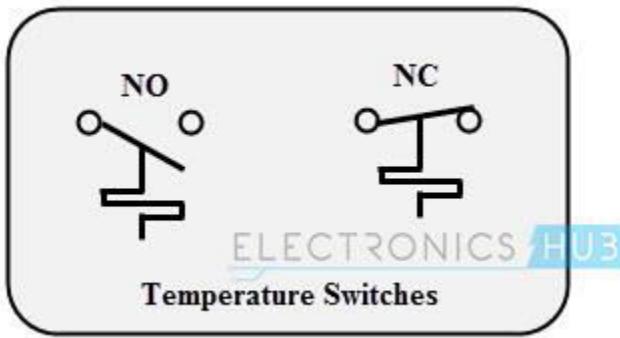
2.8.11 - Temperature Switches

The most common heat sensing element is the bimetallic strip that operates on the principle of thermal expansion.

The bimetallic strips are made with two dissimilar metals (that are having different thermal expansion rates) and are bonded with each other.

The switch contacts are operated when the temperature causes the strip to bend or wrap. Another method of operating the temperature switch is to use mercury glass tube.

When the bulb is heated, mercury in the tube will expand and then generates pressure to operate the contacts.



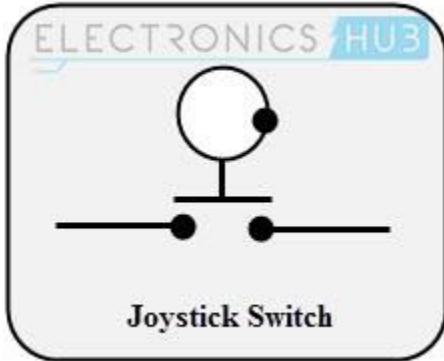
2.8.12 - Joystick Switch

Joystick switches are manually actuated control devices used mainly in portable control equipments.

It consists of a lever which moves freely in more than one axis of motion.

Depending on the movement of the lever pushed, one or more switch contacts are actuated. These are ideally suited for lowering, raising and triggering movements to the left and right.

These are used for building machinery, cable controls and cranes. The symbol for the joystick is shown below.



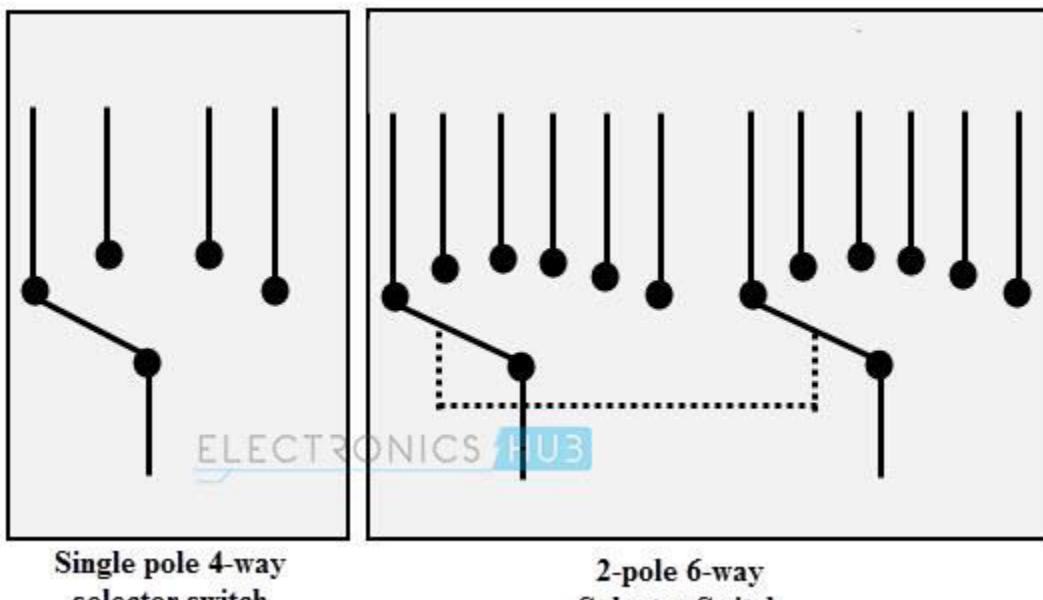
2.8.13 - Rotary Switches

These are used for connecting one line to one of many lines.

Examples of these switches are range selectors in electrical metering equipment, channel selectors in communication devices and band selectors in multi-band radios.

It consists of one or more moving contacts (knob) and more than one stationary contact.

These switches are come with different arrangement of contacts such as single pole 12-way, 3-pole 4-way, 2-pole 6-way and 4-pole 3-way.



2.9 - Series & Parallel Circuit

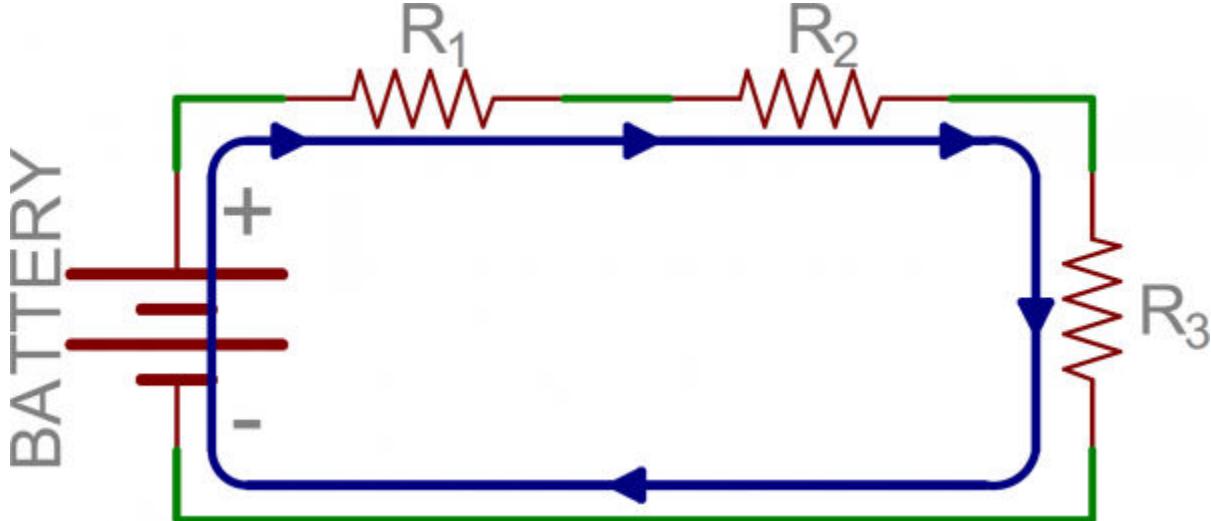
[reference - <https://learn.sparkfun.com/tutorials/series-and-parallel-circuits/all>
]

Video : Series and Parallel Circuit

https://www.youtube.com/watch?time_continue=225&v=8lMO7VAYeKY

2.9.1 - Series Circuit Defined

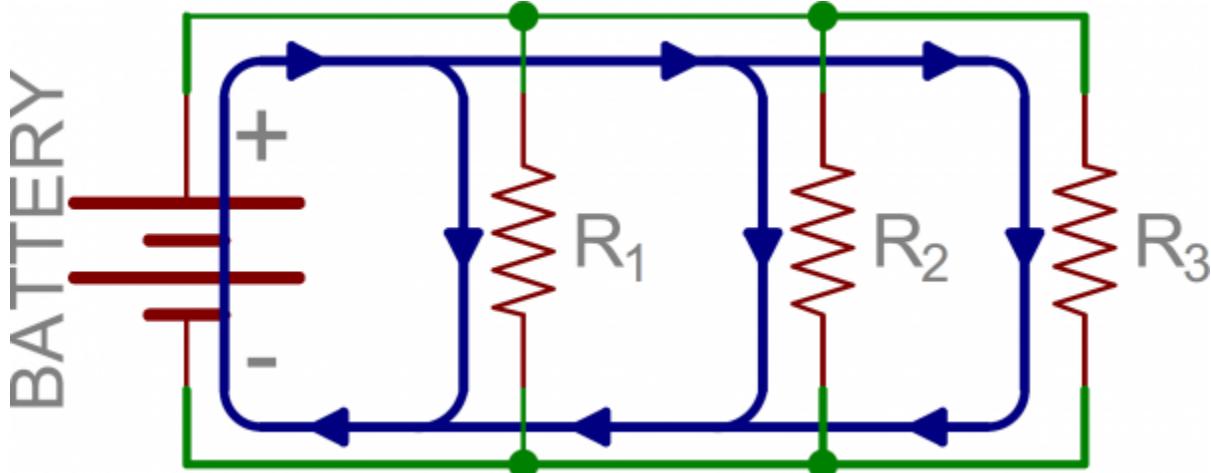
Two components are in series if they share a common node and if the same current flows through them. Here's an example circuit with three series resistors:



There's only one way for the current to flow in the above circuit. Starting from the positive terminal of the battery, current flow will first encounter R_1 . From there the current will flow straight to R_2 , then to R_3 , and finally back to the negative terminal of the battery. Note that there is only one path for current to follow. These components are in series.

2.9.2 - Parallel Circuits Defined

If components share two common nodes, they are in parallel. Here's an example schematic of three resistors in parallel with a battery:

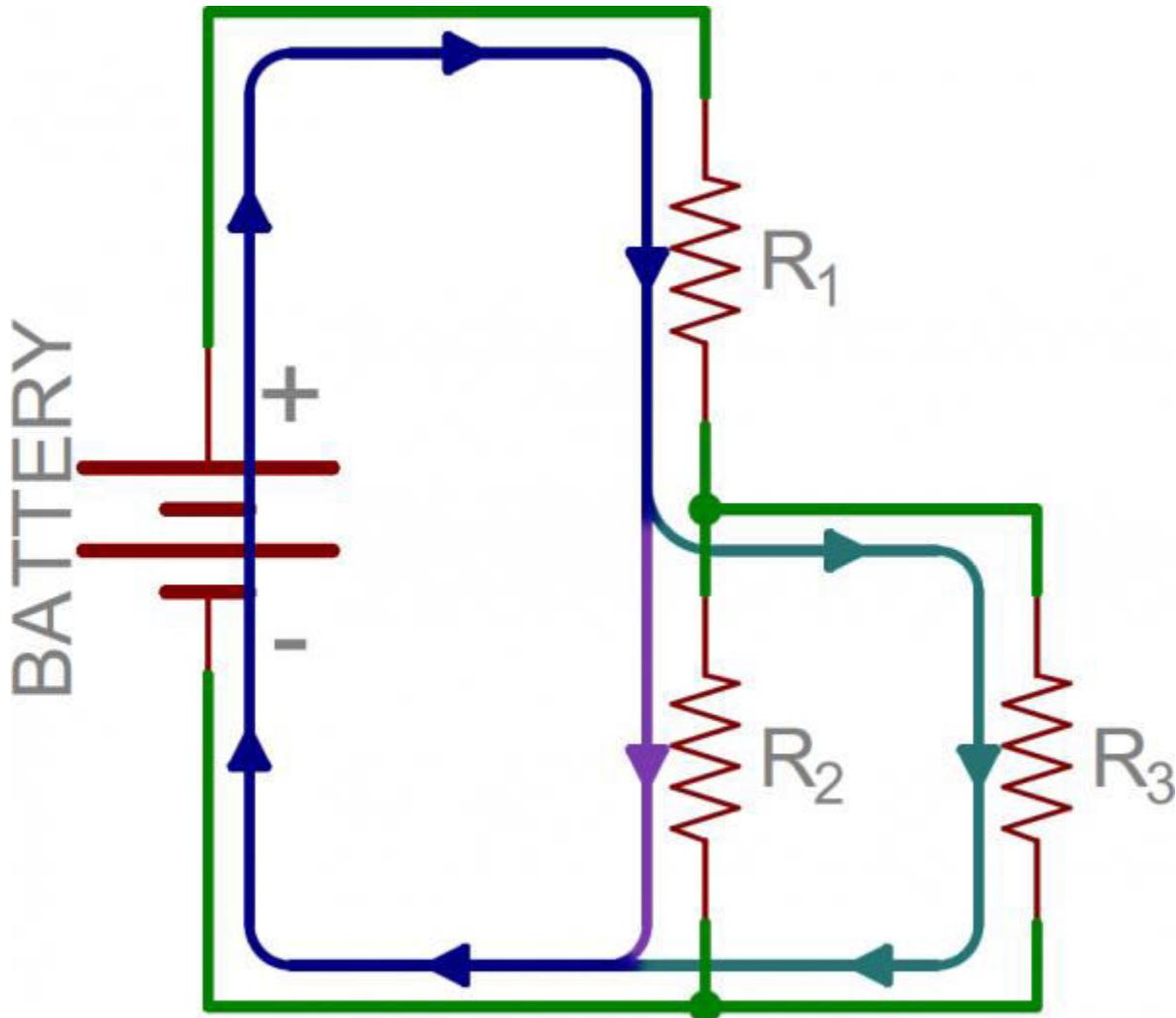


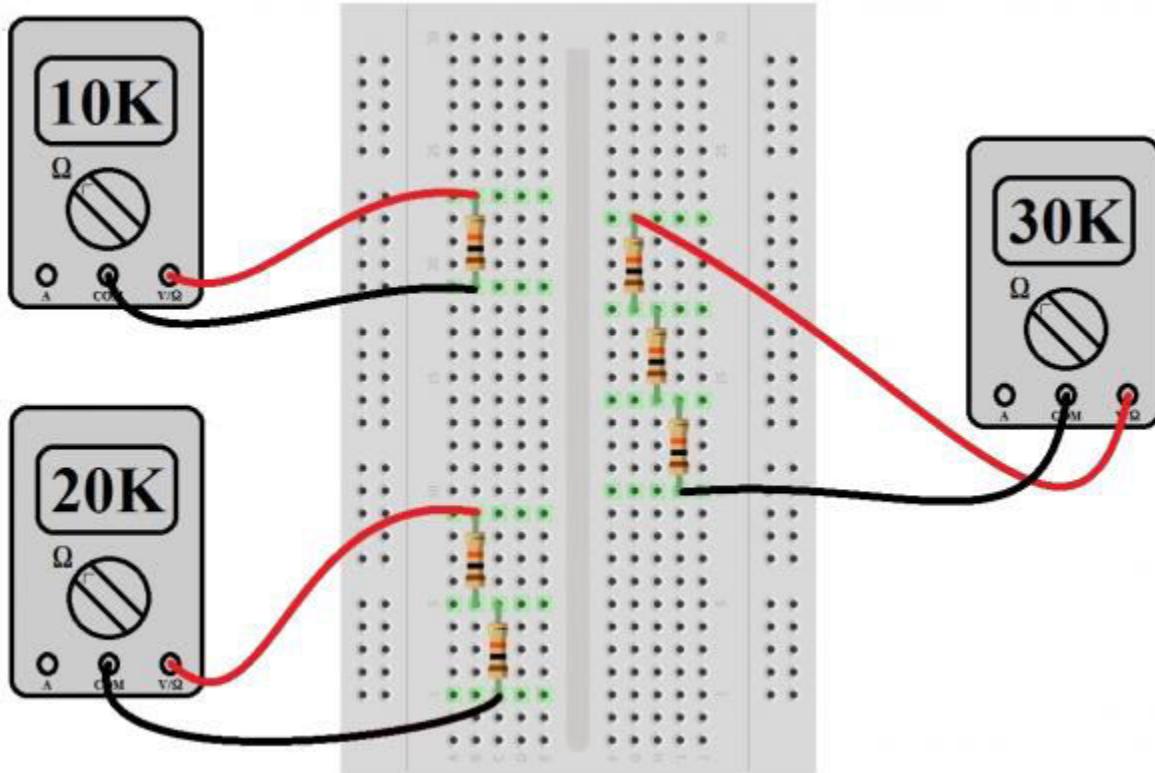
From the positive battery terminal, current flows to R_1 ... and R_2 , and R_3 . The node that connects the battery to R_1 is also connected to the other resistors. The other ends of these resistors are similarly tied together, and then tied back to the negative terminal of the battery. There are three distinct paths that current can take before returning to the battery, and the associated resistors are said to be in parallel.

Where series components all have equal currents running through them, parallel components all have the same voltage drop across them -- series:current::parallel:voltage

2.9.3 - Series and Parallel Circuits Working Together

In this example, R₂ and R₃ are in parallel with each other, and R₁ is in series with the parallel combination of R₂ and R₃





2.10 - Variable Types in C

[reference - <https://www.studytonight.com/c/variables-in-c.php>
<https://fresh2refresh.com/c-programming/c-data-types/>]

When we want to store any information(data) on our computer/laptop, we store it in the computer's memory space. Instead of remembering the complex address of that memory space where we have stored our data, our operating system provides us with an option to create folders, name them, so that it becomes easier for us to find it and access it.

Similarly, in C language, when we want to use some data value in our program, we can store it in a memory space and name the memory space so that it becomes easier to access it.

The naming of an address is known as variable. Variable is the name of memory location. Unlike constant, variables are changeable, we can change value of a variable during execution of a program. A programmer can choose a meaningful variable name. Example : average, height, age, total etc.

Type	Storage size	Value range
char	1 byte	-128 to 127 or 0 to 255
unsigned char	1 byte	0 to 255
signed char	1 byte	-128 to 127
int	2 bytes	-32,768 to 32,767
unsigned int	2 bytes	0 to 65,535
short	2 bytes	-32,768 to 32,767
long	4 bytes	-2,147,483,648 to 2,147,483,647
unsigned long	4 bytes	0 to 4,294,967,295
float	4 bytes	1.2E-38 to 3.4E+38 (6 decimal places)
double	8 bytes	2.3E-308 to 1.7E+308 (15 decimal places)
Long double	10 bytes	3.4E-4932 to 1.1E+4932 (19 decimal places)

```
#include <stdio.h>
#include <limits.h>
int main()
{
    int a;
    char b;
    float c;
    double d;
    printf("Storage size for int data type:%d \n",sizeof(a));
    printf("Storage size for char data type:%d \n",sizeof(b));
    printf("Storage size for float data type:%d \n",sizeof(c));
    printf("Storage size for double data type:%d\n",sizeof(d));
    return 0;
}
```

College Level:

2.11 - Resistor Color Code (See 2.4.10 to 2.4.13)

2.12 - Resistor Power Rating (See 2.4.14)

2.13 - Parts Datasheet

A datasheet, data sheet, or spec sheet is a document that summarizes the performance and other technical characteristics of a product, machine, component (e.g., an electronic

component), material, a subsystem (e.g., a power supply) or software in sufficient detail that allows design engineer to understand the role of the component in the overall system. Typically, a datasheet is created by the manufacturer and begins with an introductory page describing the rest of the document, followed by listings of specific characteristics, with further information on the connectivity of the devices.

for example you can see datasheet of a simple voltage regulator IC 7805 .

<https://www.sparkfun.com/datasheets/Components/LM7805.pdf>

2.14 - Install Electronics Plus App for Calculation & Datasheet

Download "Electronics Plus" App from Google Play Store from this link

<https://play.google.com/store/apps/details?id=com.electronics.crux.electronicsFree>

Electronics Plus FREE is a compact app which is designed by CRUX where you will find Electronics, Electrical, Drone/RC Plane calculators, Aeronautics, Thousands of Datasheets collections, Component Pinouts in one App. It's a must needed app for all Electronics and Electrical enthusiastic. We will be adding more calculator's and features. You can request more calculators and features in contact option.

This free version contains ad. If you want a Ad free version with improved feature please download "Electronics Plus PRO".

Class 3:

- 3.1 - Identifier & Keyword in C
- 3.2 - Input & Output in C
- 3.3 - Addition , Subtraction, Multiplication , Division in C
- 3.4 - Analog and Digital
- 3.5 - Linear Voltage Regulator
- 3.6 - Ohm's Law & Voltage Divider
- 3.7 - Capacitor
- 3.8 - Pull Up & Pull Down Resistor
- 3.9 - Arduino Simple Code Structure
- 3.10 - Single & Multiple LED Blink
- 3.11 - Digital Read
- 3.12 - Serial Communication
- College Level:**
- 3.13 - Button Debounce

3.1 - Keyword & Identifier in C

Video : C Programming Tutorial for Beginner
[\[https://www.youtube.com/watch?v=KJqsSFOSQv0\]](https://www.youtube.com/watch?v=KJqsSFOSQv0)

3.1.1 - Keyword

All the words used in C program which have fixed predefined meaning and whose meanings can't be changed by the users are termed as Keywords. There are fixed number of keywords in C programming language and every keyword serves as a building block for program statements. So you have to write keywords exactly like below.

32 ANSI C Keywords			
auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
const	extern	return	union
char	float	short	unsigned
continue	for	signed	volatile
default	goto	sizeof	void
do	if	static	while

3.1.2 - Identifier

In C language identifiers are the names given to variables, constants, functions and user-defined data. These identifier are defined against a set of rules.

Rules for an Identifier :

- 1) An Identifier can only have alphanumeric characters (a-z , A-Z , 0-9) and underscore(_).
- 2) The first character of an identifier can only contain alphabet(a-z , A-Z) or underscore (_).
- 3) Identifiers are also case sensitive in C. For example name and Name are two different identifier in C.
- 4) Keywords are not allowed to be used as Identifiers.
- 5) No special characters, such as semicolon, period, white spaces, slash or comma are permitted to be used in or as Identifier.

Example :

```
int pinNumber;
float balance;
```

```
char c;
```

Here pinNumber , balance & c are identifier and int , float & char are keyword.

3.2 - Input & Output in C

[reference - <https://www.programiz.com/c-programming/c-input-output>
<https://www.geeksforgeeks.org/basic-input-and-output-in-c/>]

C language has standard libraries that allow input and output in a program. The stdio.h or standard input output library in C that has methods for input and output.

scanf()

The scanf() method, in C, reads the value from the console as per the type specified.

Syntax:

```
scanf("%X", &variableOfType);
```

where %X is the format specifier in C. It is a way to tell the compiler what type of data is in a variable and **& is the address operator in C**, which tells the compiler to change the real value of this variable, stored at this address in the memory.

printf()

The printf() method, in C, prints the value passed as the parameter to it, on the console screen.

Syntax:

```
printf("%X", variableOfType);
```

where %X is the format specifier in C. It is a way to tell the compiler what type of data is in a variable

and

& is the address operator in C, which tells the compiler to change the real value of this variable, stored at this address in the memory.

How to take input and output of basic types in C?

The Syntax for input and output for these are:

Integer:

```
Input: scanf("%d", &intVariable);
Output: printf("%d", intVariable);
```

Float:

```
Input: scanf("%f", &floatVariable);
Output: printf("%f", floatVariable);
```

Character:

```
Input: scanf("%c", &charVariable);
Output: printf("%c", charVariable);
```

Example 1 :

This code will allow user to give two input then it will calculate output

```
#include <stdio.h>
int a;
int b;

int main()
{
    printf("Please Enter A=");
    scanf("%d",&a);
    printf("Please Enter B=");
    scanf("%d",&b);
    printf("Sum is = %d\n",a+b);
    return 0;
}
```

Example 2 :

Various types of output in C

```
#include <stdio.h>

int a = 5;
float f = 2.5;
char c = 'A';
char z[100] = "I am learning C programming language.";

int main()
{
    //This is a comment
```

```

//%d for integer, %f for float , %c for char , %s for string
printf("integer a = %d\n",a);// \n for newline
printf("float f = %f\n",f);
printf("Char c = %c\n",c);
printf("This is string : %s", z); // %s is format specifier

return 0;
}

```

3.3 - Addition , Subtraction, Multiplication , Division in C

This example will demonstrate all of those process.

```

#include <stdio.h>
int a;
int b;
int addition,subtraction,multiplication,division,remainder;

int main()
{
    printf("Please Enter A=");
    scanf("%d",&a);
    printf("Please Enter B=");
    scanf("%d",&b);

    addition = a + b;
    subtraction = a - b;
    multiplication = a * b;
    division = a / b;
    remainder = a % b;

    printf("addition = %d\n",addition);
    printf("subtraction = %d\n",subtraction);
    printf("multiplication = %d\n",multiplication);
    printf("division = %d\n",division);
    printf("remainder = %d\n",remainder);

    return 0;
}

```

3.4 - Analog and Digital

[reference - <https://learn.sparkfun.com/tutorials/analog-vs-digital/all>

<https://techdifferences.com/difference-between-analog-and-digital-signal.html>]

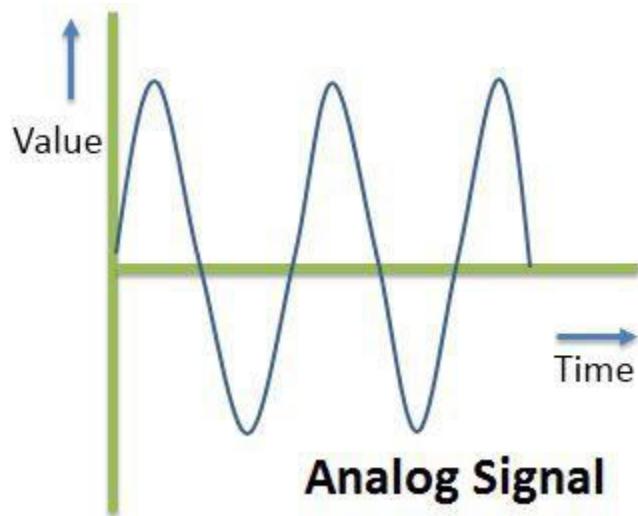
Video : Analog vs Digital [<https://www.youtube.com/watch?v=btqAUdbj85E>]

Analog and Digital Signals [<https://www.youtube.com/watch?v=Z3rsO912e3I>]

Difference between Analog and Digital [<https://youtu.be/WxJKXGugfh8>]

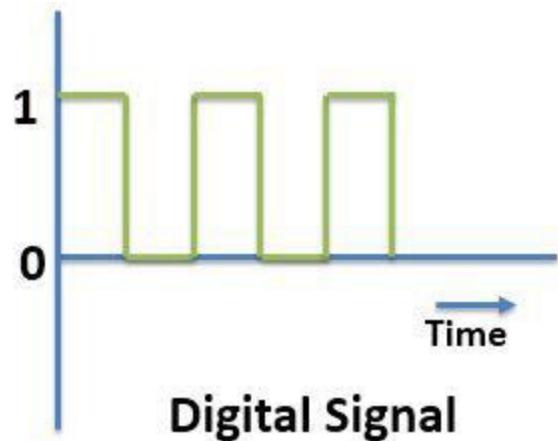
3.4.1 - Analog Signal

Analog signal is a kind of continuous waveform that changes over time. An analog signal is further classified into simple and composite signals. A simple analog signal is a sine wave that cannot be decomposed further. On the other hand, a composite analog signal can be further decomposed into multiple sine waves. An analog signal is described using amplitude, period or frequency and phase. Amplitude marks the maximum height of the signal. Frequency marks the rate at which signal is changing. Phase marks the position of the wave with respect to time zero. An analog signal is not immune to noise hence, it faces distortion and decrease the quality of transmission. The range of value in an analog signal is not fixed.



3.4.2 - Digital Signal

Digital signals also carry information like analog signals but is somewhat different from analog signals. Digital signal is noncontinuous, discrete time signal. Digital signal carries information or data in the binary form i.e. a digital signal represents information in the form of bits. Digital signal can be further decomposed into simple sine waves that are called harmonics. Each simple wave has different amplitude, frequency and phase. Digital signal is described with bit rate and bit interval. Bit interval describes the time required for sending a single bit. On the other hand, bit rate describes the frequency of bit interval. A digital signal is more immune to the noise; hence, it hardly faces any distortion. Digital signals are easier to transmit and are more reliable when compared to analog signals. Digital signal has a finite range of values. The digital signal consists of 0s and 1s.



3.4.3 - Analog and Digital Signal Comparison

BASIS FOR COMPARISON	ANALOG SIGNAL	DIGITAL SIGNAL
Basic	An analog signal is a continuous wave that changes over a time period.	A digital signal is a discrete wave that carries information in binary form.
Representation	An analog signal is represented by a sine wave.	A digital signal is represented by square waves.
Description	An analog signal is described by the amplitude, period or frequency, and phase.	A digital signal is described by bit rate and bit intervals.
Range	Analog signal has no fixed range.	Digital signal has a finite numbers i.e. 0 and 1.
Distortion	An analog signal is more prone to distortion.	A digital signal is less prone to distortion.
Transmit	An analog signal transmit data in the form of a wave.	A digital signal carries data in the binary form i.e. 0 nad 1.
Example	The human voice is the best example of an analog signal.	Signals used for transmission in a computer are the digital signal.

3.5 - Linear Voltage Regulator

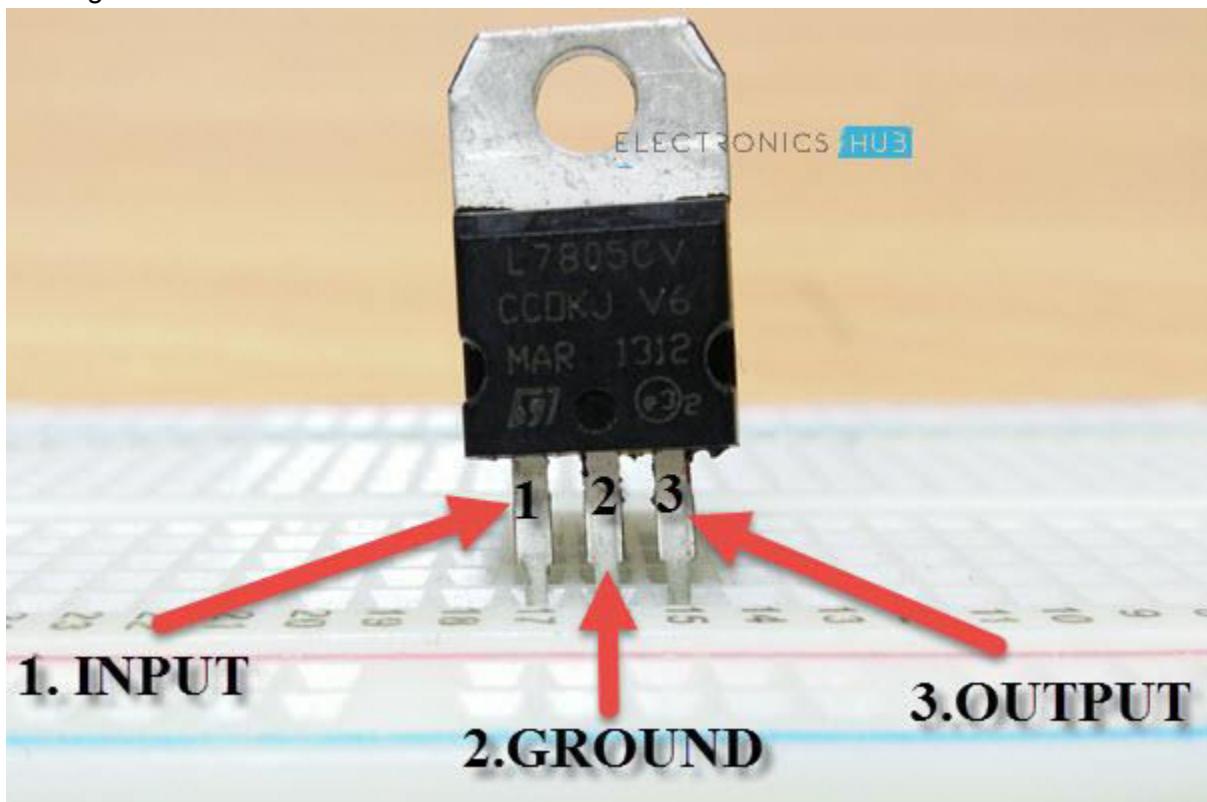
[reference - <https://predictabledesigns.com/linear-and-switching-voltage-regulators-introduction/>] Linear regulators can be thought of as variable resistance devices, where the internal resistance is varied in order to maintain a constant output voltage. In reality, the variable resistance is provided by means of a transistor controlled by an amplifier feedback loop.

Linear regulators normally consist of a minimal of three pins – an input input, an output pin, and a ground pin. External capacitors are placed on the input and output terminals to provide filtering and to improve the transient response to sudden load changes. The output capacitor is also required for stability of the voltage regulator's feedback loop.

3.5.1 - LM7805

[reference - <https://www.electronicshub.org/understanding-7805-ic-voltage-regulator/>]

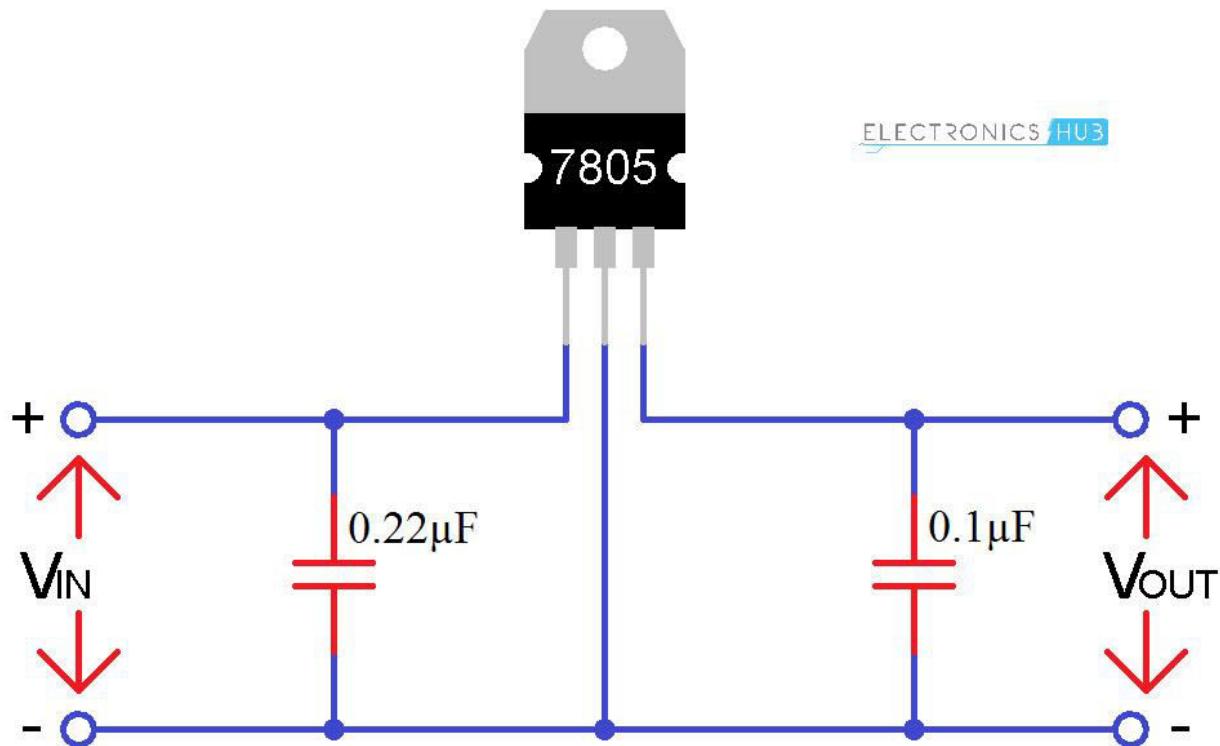
7805 is a three terminal linear voltage regulator IC with a fixed output voltage of 5V which is useful in a wide range of applications. Currently, the 7805 Voltage Regulator IC is manufactured by Texas Instruments, ON Semiconductor, STMicroelectronics, Diodes incorporated, Infineon Technologies, etc. 7805 is a three terminal device with the three pins being 1. INPUT, 2. GROUND and 3. OUTPUT. The following image shows the pins on a typical 7805 IC in To-220 Package.



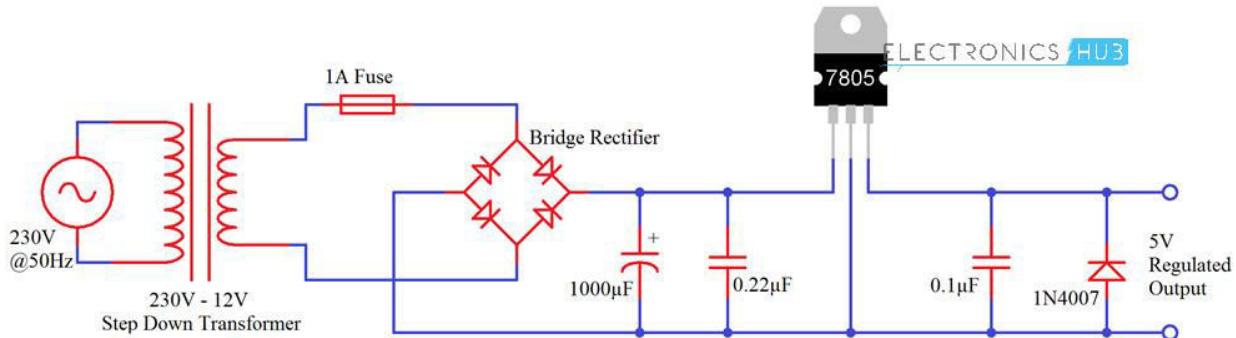
The pin description of the 7805 is described in the following table:

PIN NO.	PIN	DESCRIPTION
1	INPUT	Pin 1 is the INPUT Pin. A positive unregulated voltage is given as input to this pin.
2	GROUND	Pin 2 is the GROUND Pin. It is common to both Input and Output.
3	OUTPUT	Pin 3 is the OUTPUT Pin. The output regulated 5V is taken at this pin of the IC.

Basic Circuit :



The following image shows the circuit diagram of producing a regulated 5V from AC Mains supply.

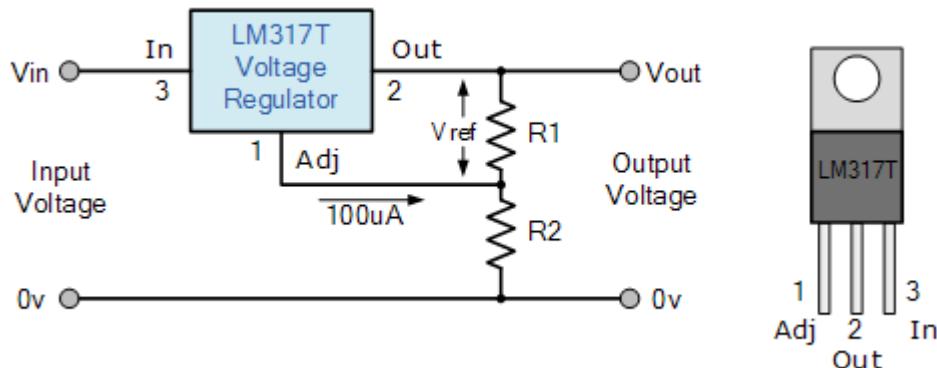


3.5.2 - LM317

[reference - <https://www.electronics-tutorials.ws/blog/variable-voltage-power-supply.html>]

The LM317T is an adjustable 3-terminal positive voltage regulator capable of supplying different DC voltage outputs other than the fixed voltage power supply of +5 or +12 volts, or as a variable output voltage from a few volts up to some maximum value all with currents of about 1.5 amperes.

With the aid of a small bit of additional circuitry added to the output of the PSU we can have a bench power supply capable of a range of fixed or variable voltages either positive or negative in nature. In fact this is more simple than you may think as the transformer, rectification and smoothing has already been done by the PSU beforehand all we need to do is connect our additional circuit to the +12 volt yellow wire output. But firstly, let's consider a fixed voltage output.



The voltage across the feedback resistor R₂ is a constant 1.25V reference voltage, V_{ref} produced between the “output” and “adjustment” terminal. The adjustment terminal current is a constant current of 100µA. Since the reference voltage across resistor R₁ is constant, a constant current i will flow through the other resistor R₂, resulting in an output voltage of:

$$V_{out} = 1.25 \left(1 + \frac{R_2}{R_1} \right)$$

Then whatever current flows through resistor R1 also flows through resistor R2 (ignoring the very small adjustment terminal current), with the sum of the voltage drops across R1 and R2 being equal to the output voltage, V_{out}. Obviously the input voltage, V_{in} must be at least 2.5 volts greater than the required output voltage to power the regulator.

3.6 - Ohm's Law & Voltage Divider

3.6.1 - Ohm's law

[reference - https://www.ducksters.com/science/physics/ohms_law.php]

One of the most important and basic laws of electrical circuits is Ohm's law which states that the current passing through a conductor is proportional to the voltage over the resistance.

Equation

Ohm's law may sound a bit confusing when written in words, but it can be described by the simple formula:

$$I = \frac{V}{R}$$

where I = current in amps, V = voltage in volts, and R = resistance in ohms

This same formula can be also be written in order to calculate for the voltage or the resistance:

$$I = \frac{V}{R} \quad \text{or} \quad V = IR \quad \text{or} \quad R = \frac{V}{I}$$

Ohm's law describes the way current flows through a resistance when a different electric potential (voltage) is applied at each end of the resistance. One way to think of this is as water flowing through a pipe. The voltage is the water pressure, the current is the amount of water flowing through the pipe, and the resistance is the size of the pipe. More water will flow through the pipe (current) the more pressure is applied (voltage) and the bigger the pipe is (lower the resistance).

Interesting Facts about Ohm's Law

It is generally applied only to direct current (DC) circuits, not alternating current (AC) circuits. In AC circuits, because the current is constantly changing, other factors such as capacitance and inductance must be taken into account.

The concept behind Ohm's law was first explained by German Physicist Georg Ohm who the law is also named after.

The tool for measuring volts in an electric circuit is called a voltmeter. An ohmmeter is used for measuring resistance. A multimeter can measure several functions including voltage, current, resistance, and temperature.

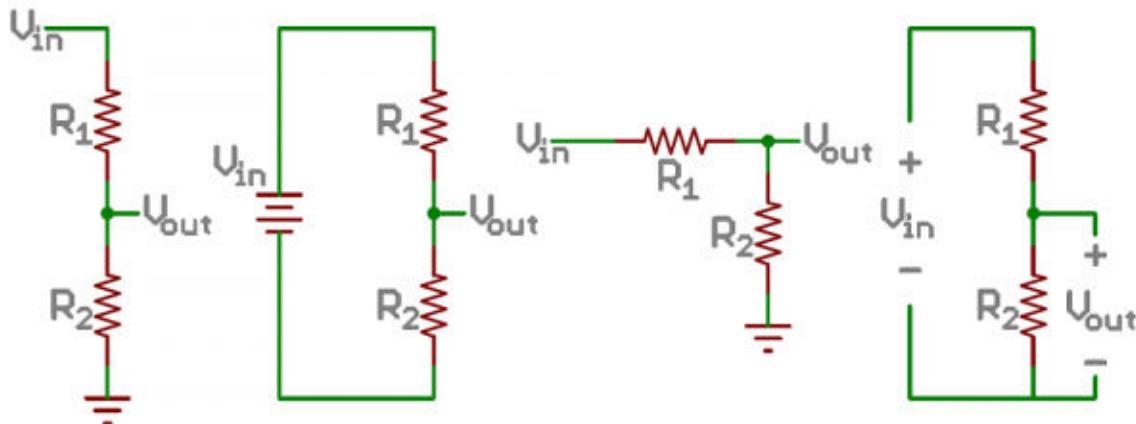
3.6.2 - Voltage Divider

[reference - <https://learn.sparkfun.com/tutorials/voltage-dividers/all>]

In electronics, a voltage divider (also known as a potential divider) is a passive linear circuit that produces an output voltage (V_{out}) that is a fraction of its input voltage (V_{in}). Voltage division is the result of distributing the input voltage among the components of the divider. A simple example of a voltage divider is two resistors connected in series, with the input voltage applied across the resistor pair and the output voltage emerging from the connection between them.

The Circuit

A voltage divider involves applying a voltage source across a series of two resistors. You may



Examples of voltage divider schematics. Shorthand, longhand, resistors at same/different angles, etc.

see it drawn a few different ways, but they should always essentially be the same circuit.

The Equation

The voltage divider equation assumes that you know three values of the above circuit: the input voltage (V_{in}), and both resistor values (R_1 and R_2). Given those values, we can use this equation to find the output voltage (V_{out}):

$$V_{out} = V_{in} \cdot \frac{R_2}{R_1 + R_2}$$

3.7 - Capacitor

[reference -

https://www.ducksters.com/science/physics/resistors_capacitors_and_inductors.php

A capacitor represents the amount of capacitance in a circuit. The capacitance is the ability of a component to store an electrical charge. You can think of it as the "capacity" to store a charge. The capacitance is defined by the equation

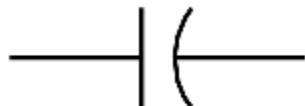
$$C = q/V$$

where q is the charge in coulombs and V is the voltage.

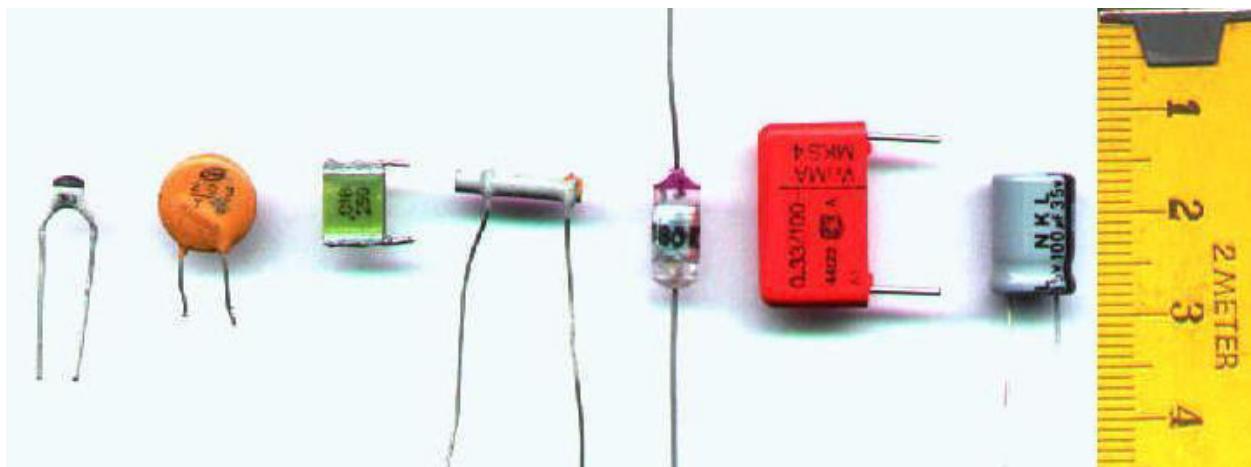
In a DC circuit, a capacitor becomes an open circuit blocking any DC current from passing the capacitor. Only AC current will pass through a capacitor.

Capacitance is measured in Farads.

The symbol for capacitance is two parallel lines. Sometimes one of the lines is curved as shown below. The letter "C" is used in equations.



Capacitor Symbol



Capacitor materials. From left: multilayer ceramic, ceramic disc, multilayer polyester film, tubular ceramic, polystyrene, metalized polyester film, aluminum electrolytic. Major scale divisions are in centimetres.

3.8 - Pull Up & Pull Down Resistor

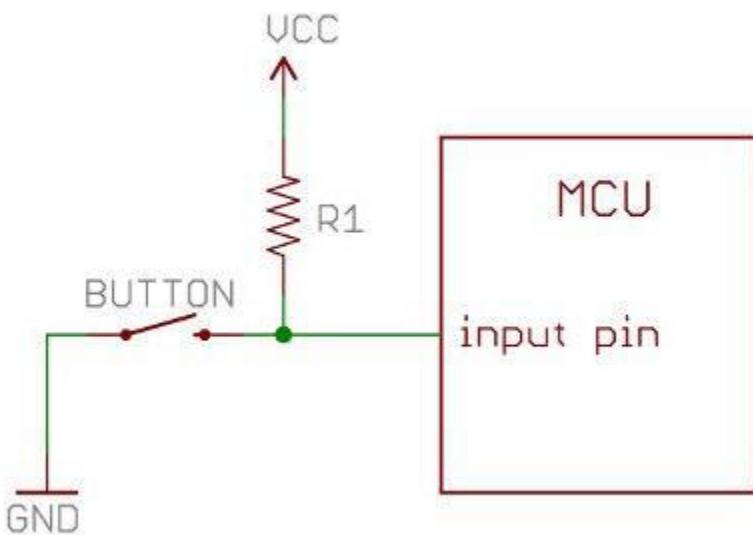
[reference - <https://learn.sparkfun.com/tutorials/pull-up-resistors/all>]

3.8.1 - Pull Up Resistor

Let's say you have an MCU with one pin configured as an input. If there is nothing connected to the pin and your program reads the state of the pin, will it be high (pulled to VCC) or low (pulled to ground)? It is difficult to tell. This phenomena is referred to as floating. To prevent this unknown state, a pull-up or pull-down resistor will ensure that the pin is in either a high or low state, while also using a low amount of current.

For simplicity, we will focus on pull-ups since they are more common than pull-downs. They operate using the same concepts, except the pull-up resistor is connected to the high voltage (this is usually 3.3V or 5V and is often referred to as VCC) and the pull-down resistor is connected to ground.

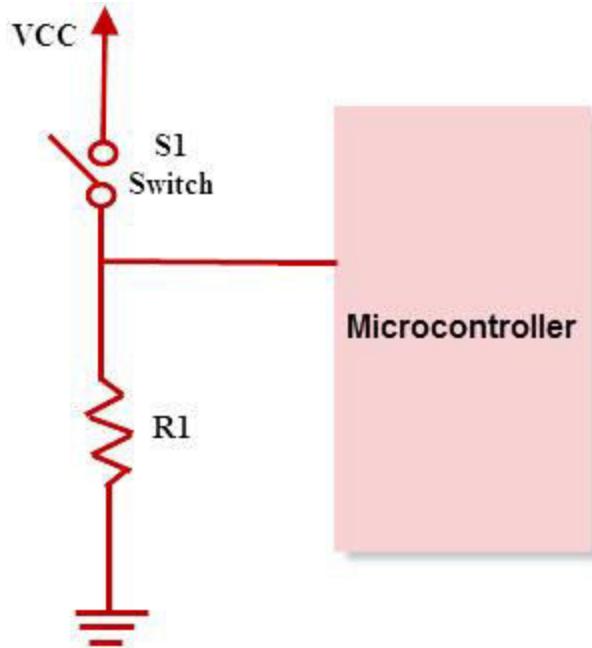
Pull-ups are often used with buttons and switches.



With a pull-up resistor, the input pin will read a high state when the button is not pressed. In other words, a small amount of current is flowing between VCC and the input pin (not to ground), thus the input pin reads close to VCC. When the button is pressed, it connects the input pin directly to ground. The current flows through the resistor to ground, thus the input pin reads a low state. Keep in mind, if the resistor wasn't there, your button would connect VCC to ground, which is very bad and is also known as a short.

3.8.2 - Pull Down Resistor

As pull up resistors, Pull-down resistors also work in the same way. But, they pull the pin to a low value. Pull-down resistors are connected between a particular pin on a microcontroller and the ground terminal. An example of a pull down resistor is a digital circuit shown in the figure below. A switch is connected between the VCC and the microcontroller pin. When the switch is closed in the circuit, the input of the microcontroller is logic 1, but when the switch is open in a circuit, the pull down resistor pulls down the input voltage to the ground (logic 0 or logic low value). The pull down resistor should have a higher resistance than the impedance of the logic circuit.



3.9 - Arduino Code Simple Structure

video : Arduino Control Flow [<https://youtu.be/QpPGGuGbCA>]

Setup() Function

- A sketch does not have a main() function
- Every sketch has a **setup()** function
 - Executed once when Arduino is powered up
 - Used for initialization operations
 - Returns no value, takes no arguments

```
void setup() {
```

```
    ...
```

```
}
```

Loop () Function

- Every sketch has a **loop()** function
 - Executed iteratively as long as the Arduino is powered up
 - **loop()** starts executing after **setup()** has finished
 - **loop()** is the main program control flow
 - Returns no value, takes no arguments
- ```
void loop() {
```

...

```
}
```

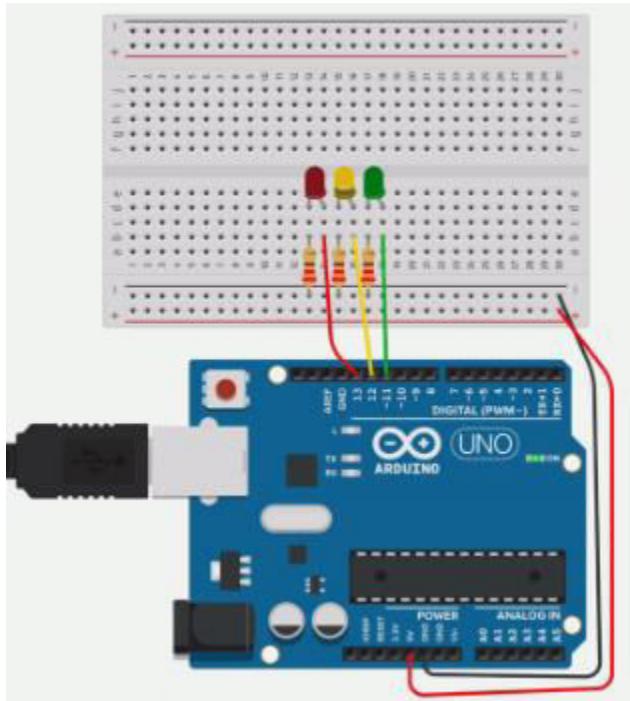
## 3.10 - Single & Multiple LED Blink

We have to connect LED with series resistor . `digitalWrite(ledPin , HIGH)` will ON LED and `digitalWrite(ledPin , LOW)` will OFF LED. Arduino has a built in LED in digital pin 13.

### **Example : Single LED Blink**

```
int ledPin = 13;
void setup() {
 pinMode(ledPin, OUTPUT);
}
void loop() {
 digitalWrite(ledPin, HIGH); // turn the LED on (HIGH is the voltage level)
 delay(1000); // wait for a second
 digitalWrite(ledPin, LOW); // turn the LED off by making the voltage LOW
 delay(1000); // wait for a second
}
```

### Example : Multiple LED Blink



```

int ledPin1 = 11;
int ledPin2 = 12;
int ledPin3 = 13;
void setup() {
 // initialize digital pin LED_BUILTIN as an output.
 pinMode(ledPin1, OUTPUT);
 pinMode(ledPin2, OUTPUT);
 pinMode(ledPin3, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {
 digitalWrite(ledPin1, HIGH); // turn the LED on (HIGH is the voltage
 level)
 digitalWrite(ledPin2, HIGH);
 digitalWrite(ledPin3, HIGH);
 delay(1000); // wait for a second
 digitalWrite(ledPin1, LOW); // turn the LED off by making the voltage
 LOW
 digitalWrite(ledPin2, LOW);
 digitalWrite(ledPin3, LOW);
 delay(1000); // wait for a second
}

```

## 3.11 - Digital Read

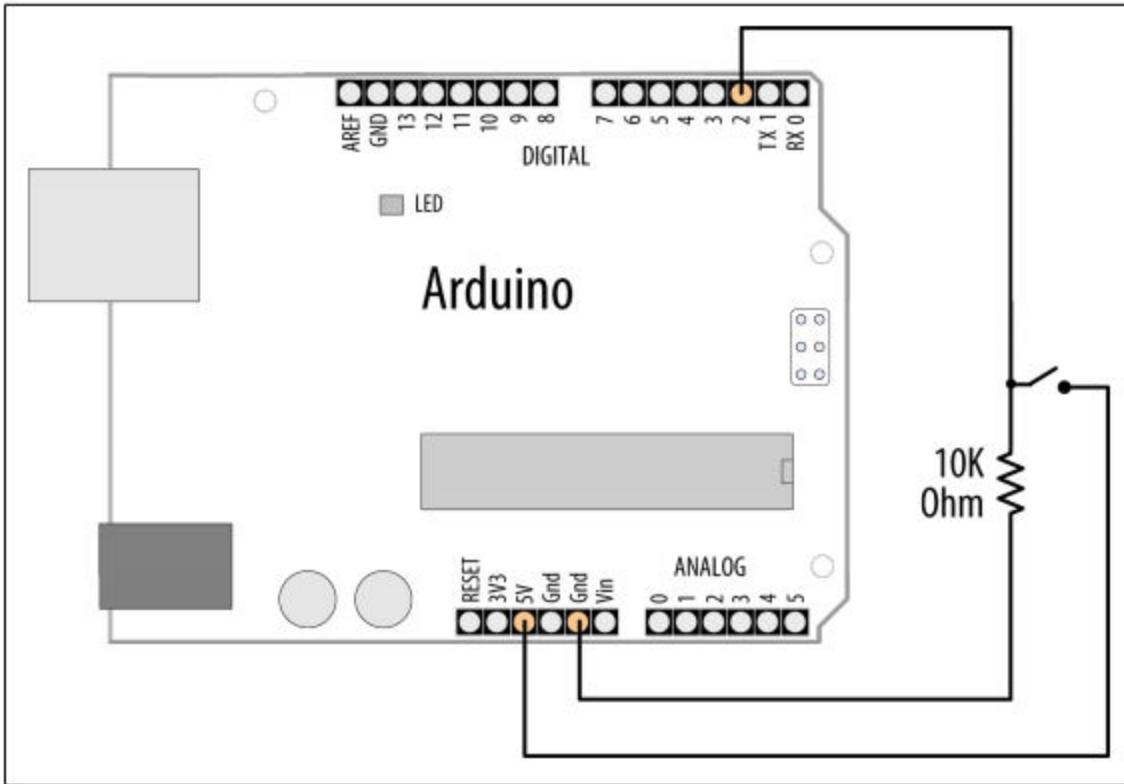
Steps :

- 1) We declare ledPin & inputPin variable and assign pin 13 & 2
- 2) setup() function will setup ledPin as output & inputPin as input
- 3) In loop() section digitalRead(inputPin) will read voltage level of pin 2 and store the value in val variable. val will be HIGH or LOW depending on Voltage level of pin 2
- 4) if condition will set output pin 13 as HIGH or LOW using digitalWrite(ledPin , HIGH) or digitalWrite(ledPin , LOW)

### 3.11.1 - Digital Read with External Pull Down Resistor

In this section with digitalRead() we will read the input state of a pin of an arduino.

```
/*
Pushbutton sketch
a switch connected to pin 2 lights the LED on pin 13
*/
int ledPin = 13; // choose the pin for the LED
int inputPin = 2; // choose the input pin (for a pushbutton)
void setup() {
 pinMode(ledPin, OUTPUT); // declare LED as output
 pinMode(inputPin, INPUT); // declare pushbutton as input
}
void loop(){
 int val = digitalRead(inputPin); // read input value
 if (val == HIGH) // check if the input is HIGH
 {
 digitalWrite(ledPin, HIGH); // turn LED on if switch is pressed
 }
 else
 {
 digitalWrite(ledPin, LOW); // turn LED off
 }
}
```



*Switch connected using pull-down resistor*

The `digitalRead` function monitors the voltage on the input pin (`inputPin`), and it returns a value of HIGH if the voltage is high (5 volts) and LOW if the voltage is low (0 volts).

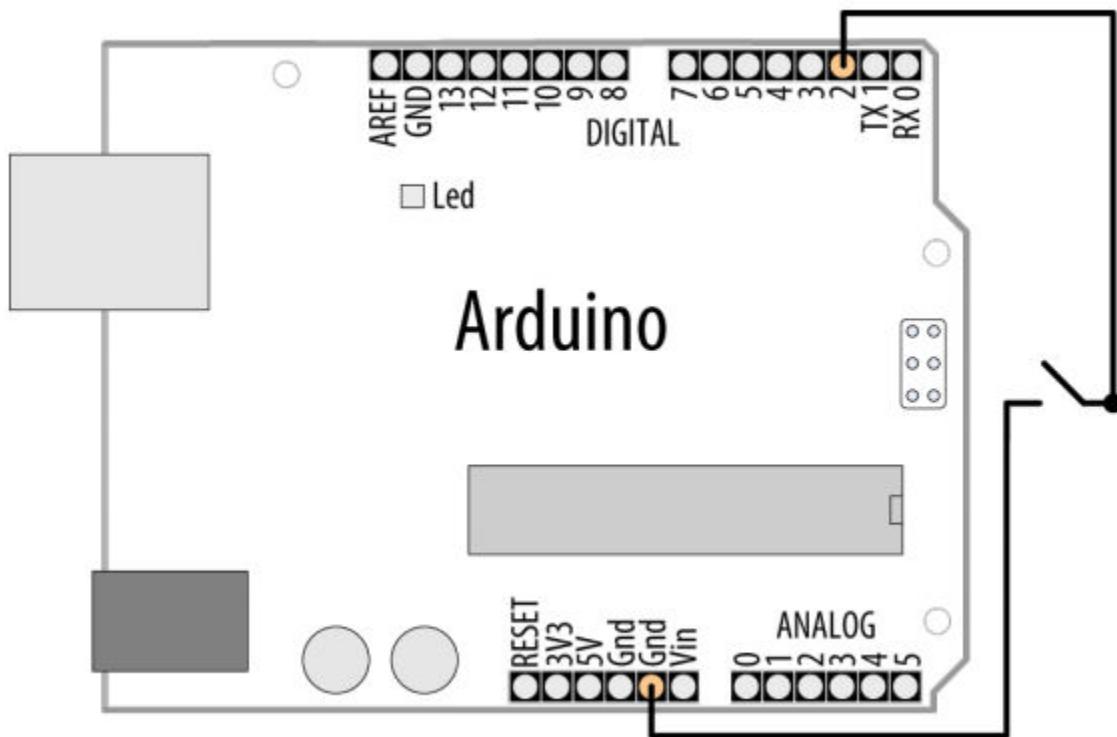
Actually, any voltage that is greater than 2.5 volts (half of the voltage powering the chip) is considered HIGH and less than this is treated as LOW. If the pin is left unconnected (known as floating), the value returned from `digitalRead` is indeterminate (it may be HIGH or LOW, and it cannot be reliably used). The resistor shown in Figure ensures that the voltage on the pin will be low when the switch is not pressed, because the resistor “pulls down” the voltage to ground. When the switch is pushed, a connection is made between the pin and +5 volts, so the value on the pin interpreted by digital Read changes from LOW to HIGH.

### 3.11.2 - Digital Read with Software Input Pullup

Since pull-up resistors are so commonly needed, many MCUs, like the ATmega328 microcontroller on the Arduino platform, have internal pull-ups that can be enabled and disabled. To enable internal pull-ups on an Arduino, you can use the following line of code in your `setup()` function:

```
pinMode(2, INPUT_PULLUP); // Enable internal pull-up resistor on pin 5
```

in this condition by default digital pin 2 will be HIGH so we have to connect push button between pin 2 and GND . So whenever we will press push button, the output of digitalRead() will be LOW.



```

int pushButton = 2;
int led1 = 13;

void setup() {
 pinMode(pushButton , INPUT_PULLUP);
 pinMode(led1 , OUTPUT);
 Serial.begin(9600);
}

void loop() {
 int x = digitalRead(pushButton);
 if(x == LOW){// LOW means pressed (because INPUT_PULLUP is used)
 digitalWrite(led1 , HIGH);
 }
 else{
 digitalWrite(led1, LOW);
 }
 Serial.println(digitalRead(pushButton));
}

```

We will learn about Serial Communication in next class but for now open serial monitor and check switch condition in serial monitor.

### 3.12 - Basic Serial Communication

[reference - <https://www.ladyada.net/learn/arduino/lesson4.html>]

Serial communications provide an easy and flexible way for your Arduino board to interact with your computer and other devices. We have to know little bit about arduino library. Arduino library is a bunch of code which help us to do various functionality with Arduino. To enable serial communication we need serial library which include automatically when we write `Serial.begin()` in void `setup()` section. For other library we may have to write `#include <libraryName.h>`

We have to select speed of serial communication between arduino and computer or other device with `Serial.begin(9600)` . Here 9600 means 9600 bit per second . We will learn details about serial communication later for now we will see a code.

**Code :**

```
int A = 5 ;
void setup () {
Serial.begin(9600);
}
void loop () {
Serial.println("We are learning Arduino Serial Communication");
Serial.print("Value of A = ");
Serial.println(A);
}
```

Now Open Serial Monitor from Tools > Serial Monitor and see output.

### College Level:

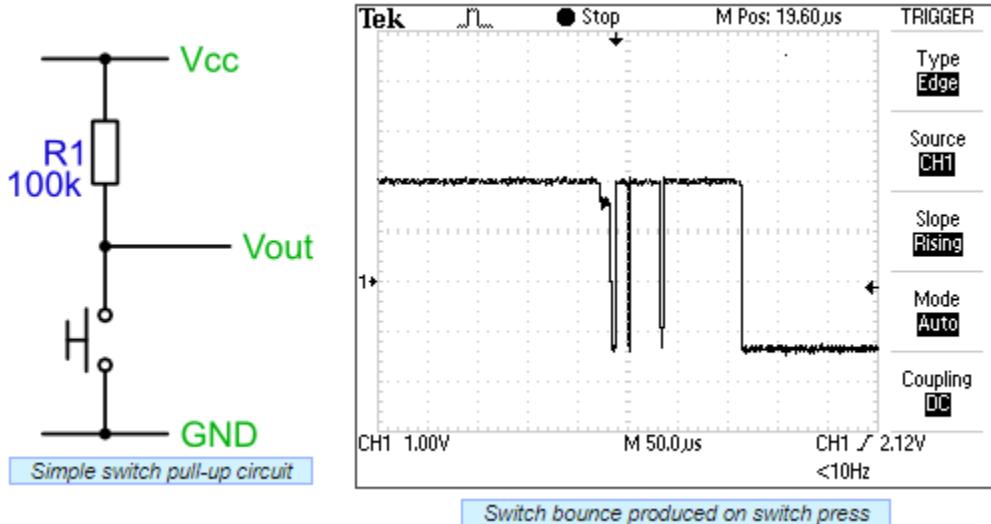
### 3.13 - Button Debounce

[reference - <http://www.labbookpages.co.uk/electronics/debounce.html>]

The left-hand image below shows a simple push switch with a pull-up resistor. The right hand image shows the trace at the output terminal, `Vout`, when the switch is pressed. As can be seen, pressing the switch does not provide a clean edge. If this signal was used as an input to a digital counter, for example, you'd get multiple counts rather than the expected single count.

Note that the same can also occur on the release for a switch.

The problem is that the contacts within the switch don't make contact cleanly, but actually slightly 'bounce'. The bounce is quite slow, so you can recreate the trace, and the problem quite easily



### Code :

```
// constants won't change. They're used here to set pin numbers:
const int buttonPin = 2; // the number of the pushbutton pin
const int ledPin = 13; // the number of the LED pin

// Variables will change:
int ledState = HIGH; // the current state of the output pin
int buttonState; // the current reading from the input pin
int lastButtonState = LOW; // the previous reading from the input pin

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

void setup() {
 pinMode(buttonPin, INPUT_PULLUP);
 pinMode(ledPin, OUTPUT);

 // set initial LED state
 digitalWrite(ledPin, ledState);
```

```
}
```

```
void loop() {
 // read the state of the switch into a local variable:
 int reading = digitalRead(buttonPin);

 // check to see if you just pressed the button
 // (i.e. the input went from LOW to HIGH), and you've waited long enough
 // since the last press to ignore any noise:

 // If the switch changed, due to noise or pressing:
 if (reading != lastButtonState) {
 // reset the debouncing timer
 lastDebounceTime = millis();
 }

 if ((millis() - lastDebounceTime) > debounceDelay) {
 // whatever the reading is at, it's been there for longer than the
 debounce
 // delay, so take it as the actual current state:

 // if the button state has changed:
 if (reading != buttonState) {
 buttonState = reading;

 // only toggle the LED if the new button state is LOW
 if (buttonState == LOW) {
 ledState = !ledState;
 }
 }
 }
 // set the LED:
 digitalWrite(ledPin, ledState);
 // save the reading. Next time through the loop, it'll be the
 lastButtonState:
 lastButtonState = reading;
}
```

## Class 4:

- 4.1 - Local & Global Variable
- 4.2 - Arduino - Control Statements
- 4.3 - Basic Multimeter Operation
- 4.4 - Variable Resistor & Analog Read (ADC)
- College Level :**
- 4.5 - Voltage Divider Details

### 4.1 - Local & Global Variable

[reference - <http://www.toptechboy.com/arduino/lesson-33-understanding-local-and-global-variables-in-arduino/>]

**Video : Understanding Arduino Local and Global Variables**

[\[https://youtu.be/8mbyebZwHFc\]](https://youtu.be/8mbyebZwHFc)

It's important to understand size of data type in arduino for efficient coding.

| Numeric types | Bytes | Range                           | Use                                                                                    |
|---------------|-------|---------------------------------|----------------------------------------------------------------------------------------|
| int           | 2     | -32768 to 32767                 | Represents positive and negative integer values.                                       |
| unsigned int  | 2     | 0 to 65535                      | Represents only positive values; otherwise, similar to int.                            |
| long          | 4     | -2147483648 to 2147483647       | Represents a very large range of positive and negative values.                         |
| unsigned long | 4     | 4294967295                      | Represents a very large range of positive values.                                      |
| float         | 4     | 3.4028235E+38 to -3.4028235E+38 | Represents numbers with fractions; use to approximate real-world measurements.         |
| double        | 4     | Same as float                   | In Arduino, double is just another name for float.                                     |
| boolean       | 1     | false (0) or true (1)           | Represents true and false values.                                                      |
| char          | 1     | -128 to 127                     | Represents a single character. Can also represent a signed value between -128 and 127. |
| byte          | 1     | 0 to 255                        | Similar to char, but for unsigned values.                                              |

In Arduino, if a variable is declared at the top of the program, before the void setup, all parts of the program can use that variable. Hence, it is called a Global variable. On the other hand, if the variable is declared between a set of curly brackets, the variable is only recognized within that scope . . . that is, it will only be recognized and can only be used between that set of curly brackets.

For example, if a variable is declared in the void setup, it will not be recognized and can not be used in the void loop, because the void loop is within its own set of curly brackets.

Similarly, if there are two for loops inside the void loop, each for loop has its own set of curly brackets. If a variable is declared inside the first for loop, it will not be recognized inside the other for loop, and will not be recognized in the other parts of the void loop.

This might sound like a hassle, but using local variables really helps you stay out of trouble. The best way to do functions is to use local variable, and inside each function, the variables are declared that are needed by that function.

#### **global and local variable example code :**

```
int ledPin = 13; // this is a GLOBAL variable
void setup() {
 pinMode(ledPin, OUTPUT);
}

void loop() {
 int delayTime = 1000; // This is a LOCAL Variable
 digitalWrite(ledPin, HIGH);
 delay(delayTime);
 digitalWrite(ledPin, LOW);
 delay(delayTime);
}
```

Here, ledPin is a global variable and delayTime is a local variable.

## **4.2 - Arduino - Control Statements**

[reference - [https://www.tutorialspoint.com/arduino/arduino\\_control\\_statements.htm](https://www.tutorialspoint.com/arduino/arduino_control_statements.htm)  
<https://www.javatpoint.com/c-if-else>]

Control Statements are elements in Source Code that control the flow of program execution.  
 Some Control Statements are :

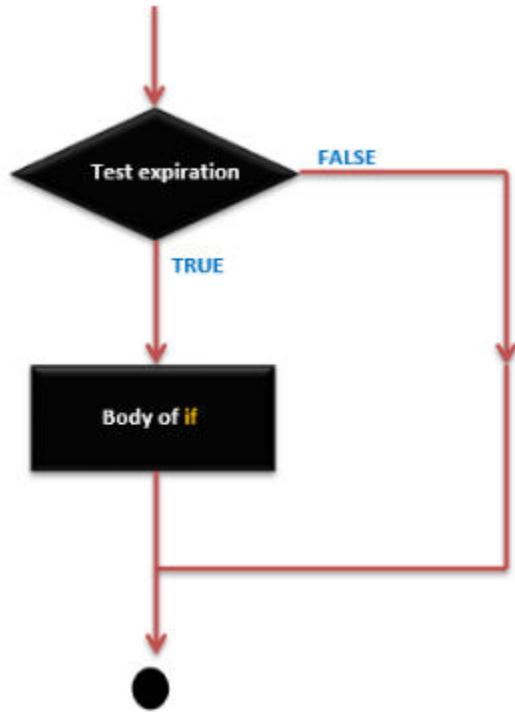
| S.NO. | Control Statement & Description                                                                                                                                                                                                |
|-------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 1     | <b>If statement</b><br>It takes an expression in parenthesis and a statement or block of statements. If the expression is true then the statement or block of statements gets executed otherwise these statements are skipped. |
| 2     | <b>If ...else statement</b><br>An <b>if</b> statement can be followed by an optional <b>else</b> statement, which executes when the expression is false.                                                                       |
| 3     | <b>If...else if ...else statement</b><br>The <b>if</b> statement can be followed by an optional <b>else if...else</b> statement, which is very useful to test various conditions using single <b>if...else if</b> statement.   |
| 4     | <b>switch case statement</b><br>Similar to the if statements, <b>switch...case</b> controls the flow of programs by allowing the programmers to specify different codes that should be executed in various conditions.         |
| 5     | <b>Conditional Operator ? :</b><br>The conditional operator <b>? :</b> is the only ternary operator in C.                                                                                                                      |

#### 4.2.1 - if statement

It takes an expression in parenthesis and a statement or block of statements. If the expression is true then the statement or block of statements gets executed otherwise these statements are skipped.

```
if (expression) {
 Block of statements;
}
```

if Statement – Execution Sequence:



Example Code :

```

/* Global variable definition */
int A = 5 ;
int B = 9 ;

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

void loop () {
/* check the condition */
if (A > B){ /* if condition is true then execute the following
statement*/
 Serial.println("A is greater than B");
}
if (A < B){
 Serial.println("A is smaller than B");
}
}

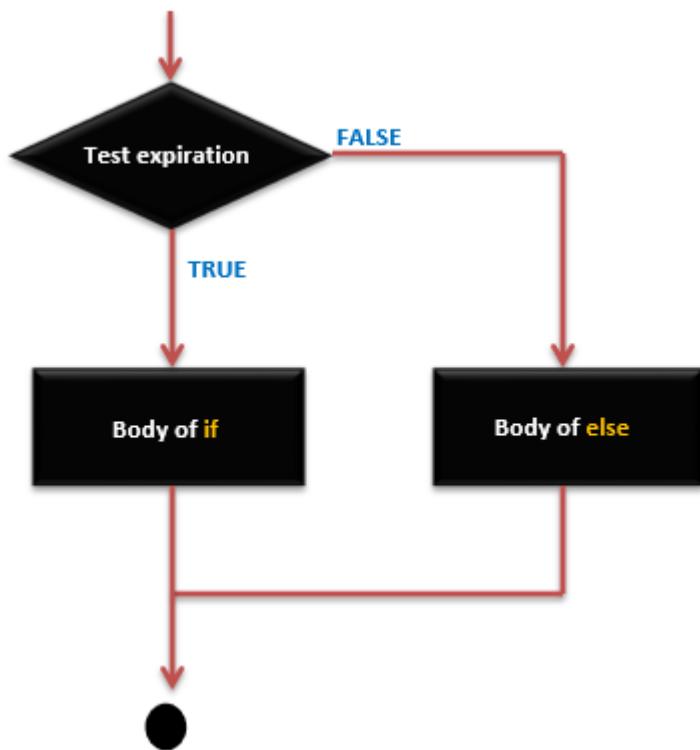
```

#### 4.2.2 - if else statement

An if statement can be followed by an optional else statement, which executes when the expression is false.

if else Statement – Execution Sequence:

```
if (expression) {
 Block of statements;
}
else {
 Block of statements;
}
```



Example code for C:

```
#include<stdio.h>
int main(){
int number=0;
printf("Enter a number:");
scanf("%d",&number);
if(number%2==0){
printf("%d is even number",number);
}
return 0;
}
```

Example code for Arduino :

```
/* Global variable definition */
int A = 5 ;
int B = 9 ;

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

void loop () {
/* check the condition */
if (A > B){ /* if condition is true then execute the following
statement*/
 Serial.println("A is greater then B");
}
else {
 Serial.println("A is smaller then B");
}
}
```

#### 4.2.3 - if else if statement

**Video : Changing LED Blink Rate on Arduino using if-else-if**

[<https://youtu.be/EqDSe6iX0oc>]

The if statement can be followed by an optional else if...else statement, which is very useful to test various conditions using single if...else if statement.

When using if...else if...else statements, keep in mind –

- An if can have zero or one else statement and it must come after any else if's.
- An if can have zero to many else if statements and they must come before the else.
- Once an else if succeeds, none of the remaining else if or else statements will be tested.

if else if Statement – Execution Sequence:

```
if (expression_1) {
 Block of statements;
}

else if(expression_2) {
 Block of statements;
}
```

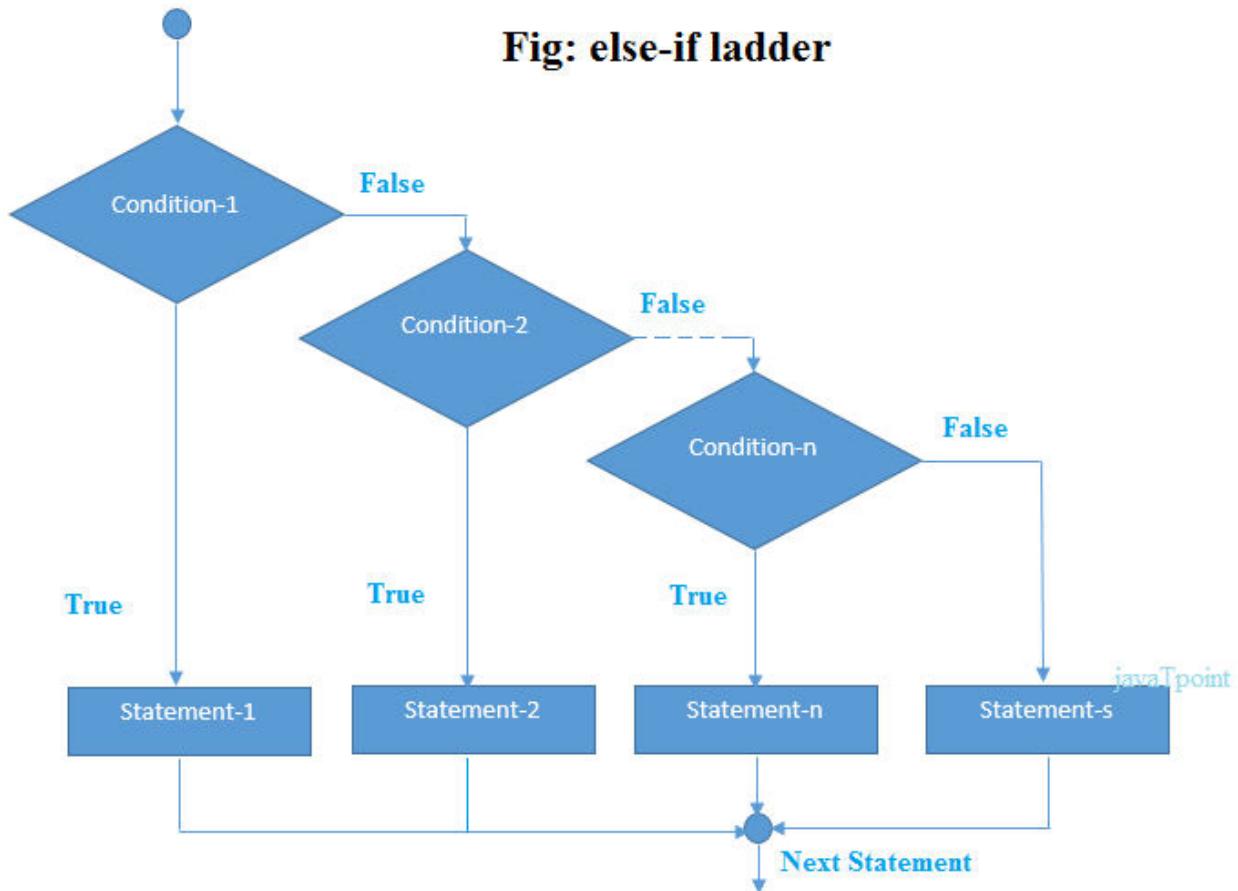
```

.
.
.

else {
 Block of statements;
}

```

**Fig: else-if ladder**



#### Example Code for C:

```

#include <stdio.h>
int main()
{
 int marks;
 printf("Enter your marks?");
 scanf("%d",&marks);
 if(marks > 85 && marks <= 100)
 {
 printf("Congrats ! you scored grade A ...");
 }
}

```

```

}
else if (marks > 60 && marks <= 85)
{
 printf("You scored grade B + ...");
}
else if (marks > 40 && marks <= 60)
{
 printf("You scored grade B ...");
}
else if (marks > 30 && marks <= 40)
{
 printf("You scored grade C ...");
}
else
{
 printf("Sorry you are fail ...");
}
}
}

```

### **Arduino Example Code :**

```

void setup() {
 Serial.begin(9600);
 pinMode(13, OUTPUT); // LED on pin 13 of UNO
}

char rx_byte = 0;

void loop() {
 if (Serial.available() > 0) { // is a character available?
 rx_byte = Serial.read();
 }
 if (rx_byte == 'a') {
 digitalWrite(13, HIGH);
 delay(500);
 digitalWrite(13, LOW);
 delay(500);
 }
 else if (rx_byte == 'b') {
 digitalWrite(13, HIGH);
 delay(200);
 digitalWrite(13, LOW);
 delay(200);
 }
}

```

```

 }
}
```

#### 4.2.4 - Switch Statement

Switch case statements are a substitute for long if statements that compare a variable to several integral values

- The switch statement is a multiway branch statement. It provides an easy way to dispatch execution to different parts of code based on the value of the expression.
- Switch is a control statement that allows a value to change control of execution.

#### Syntax of Switch Statement :

```
switch (n)
```

```
{
 case 1: // code to be executed if n = 1;
 break;
 case 2: // code to be executed if n = 2;
 break;
 default: // code to be executed if n doesn't match any cases
}
```

#### Rules for switch statement in C language :

- 1) The switch expression must be of an integer or character type.
- 2) The case value must be an integer or character constant.
- 3) The case value can be used only inside the switch statement.
- 4) The break statement in switch case is not must. It is optional. If there is no break statement found in the case, all the cases will be executed present after the matched case. It is known as fall through the state of C switch statement.

Let's try to understand it by the examples. We are assuming that there are following variables.

```
int x,y,z;
char a,b;
float f;
```

| Valid Switch      | Invalid Switch | Valid Case    | Invalid Case |
|-------------------|----------------|---------------|--------------|
| switch(x)         | switch(f)      | case 3;       | case 2.5;    |
| switch(x>y)       | switch(x+2.5)  | case 'a';     | case x;      |
| switch(a+b-2)     |                | case 1+2;     | case x+2;    |
| switch(func(x,y)) |                | case 'x'>'y'; | case 1,2,3;  |

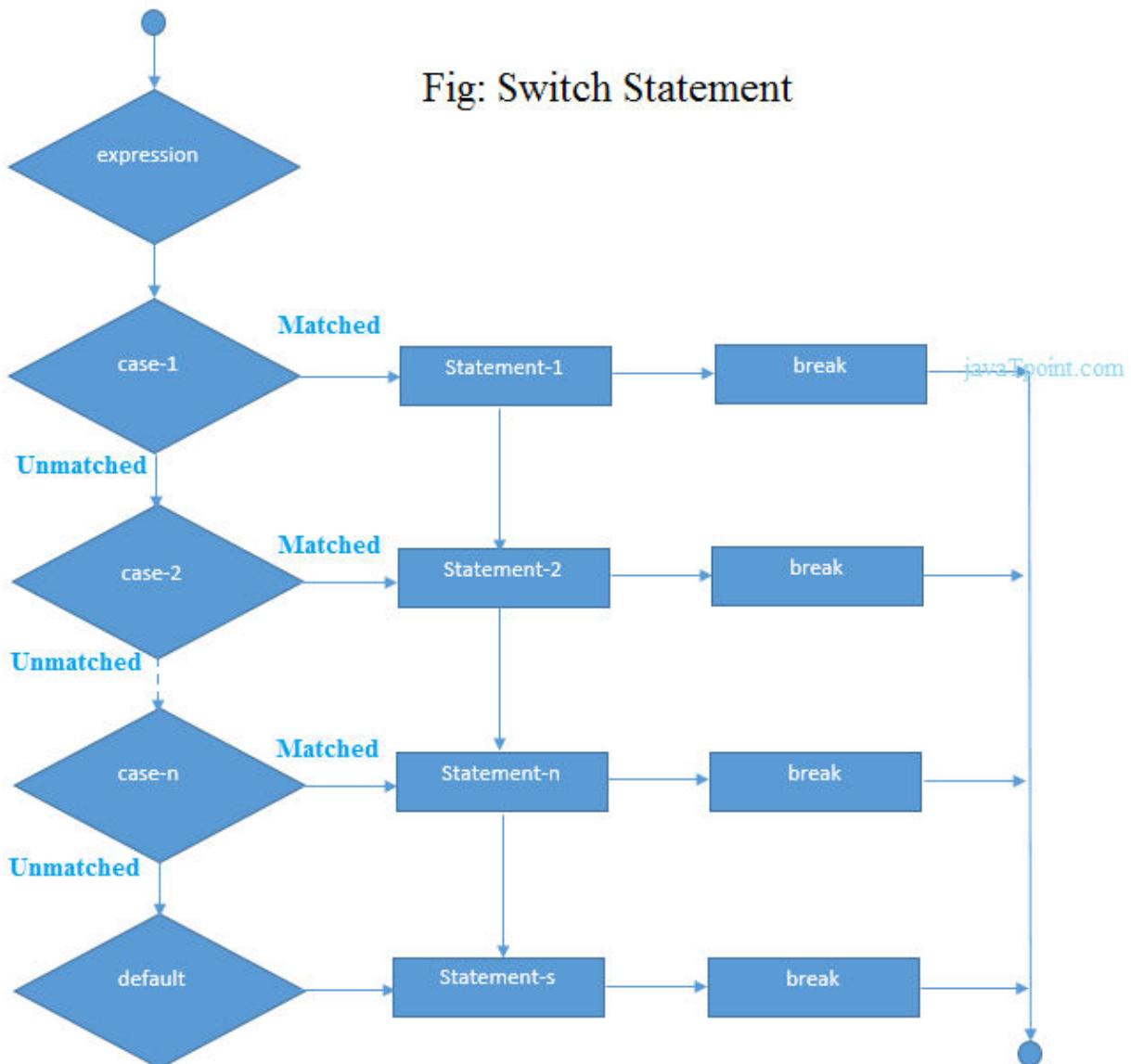
**Flowchart :**

Fig: Switch Statement

**Example Code in C :**

```
// Following is a simple program to demonstrate
// syntax of switch.
```

```
#include <stdio.h>
int main()
{
 int x = 2;
 switch (x)
 {
 case 1: printf("Choice is 1");
 break;
 case 2: printf("Choice is 2");
 break;
 case 3: printf("Choice is 3");
 break;
 default: printf("Choice other than 1, 2 and 3");
 break;
 }
 return 0;
}
```

**Example Code in Arduino :** (We will explain analogRead(A0) in later)

```
const int sensorMin = 0; // sensor minimum, discovered through
experiment
const int sensorMax = 600; // sensor maximum, discovered through
experiment

void setup() {
 // initialize serial communication:
 Serial.begin(9600);
}

void loop() {
 // read the sensor:
 int sensorReading = analogRead(A0);
 // map the sensor range to a range of four options:
 int range = map(sensorReading, sensorMin, sensorMax, 0, 3);

 // do something different depending on the range value:
 switch (range) {
 case 0: // your hand is on the sensor
 Serial.println("dark");
 break;
 case 1: // your hand is close to the sensor
 Serial.println("dim");
```

```

 break;
case 2: // your hand is a few inches from the sensor
 Serial.println("medium");
 break;
case 3: // your hand is nowhere near the sensor
 Serial.println("bright");
 break;
}
delay(1); // delay in between reads for stability
}

```

#### 4.2.5 - Conditional Operator

The conditional operator ? : is the only ternary operator in C.

conditional operator Syntax :

```
expression1 ? expression2 : expression3
```

Expression1 is evaluated first. If its value is true, then expression2 is evaluated and expression3 is ignored. If expression1 is evaluated as false, then expression3 evaluates and expression2 is ignored. The result will be a value of either expression2 or expression3 depending upon which of them evaluates as True.

Conditional operator associates from right to left.

#### Conditional Operator Rules :

expression1 must be a scalar expression; expression2 and expression3 must obey one of the following rules:

- Both expressions have to be of arithmetic type. expression2 and expression3 are subject to usual arithmetic conversions, which determines the resulting type.
- Both expressions have to be of compatible struct or union types. The resulting type is a structure or union type of expression2 and expression3.
- Both expressions have to be of void type. The resulting type is void.
- Both expressions have to be of type pointer to qualified or unqualified versions of compatible types. The resulting type is a pointer to a type qualified with all type qualifiers of the types pointed to by both expressions.
- One expression is a pointer, and the other is a null pointer constant. The resulting type is a pointer to a type qualified with all type qualifiers of the types pointed to by both expressions.
- One expression is a pointer to an object or incomplete type, and the other is a pointer to a qualified or unqualified version of void. The resulting type is that of the non-pointer-to-void expression.

#### Example Code in C :

```
#include <stdio.h>
```

```

int main()
{
 int x=1, y ;
 y = (x ==1 ? 2 : 0) ;
 printf("x value is %d\n", x);
 printf("y value is %d", y);
}

```

#### **Example Code in Arduino :**

```

int val1, val2, result;

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

 // change the values of val1 and val2 to see what the
 // conditional expression does
 val1 = 2;
 val2 = 5;
 // if val1 is bigger than val2, return val1
 // else if val1 is less than val2, return val2
 result = (val1 > val2) ? val1 : val2;

 // show result in serial monitor window
 Serial.print("The bigger number is: ");
 Serial.println(result);
}

void loop() {
}

```

## 4.3 - Basic Multimeter Operation

[reference - <https://learn.sparkfun.com/tutorials/how-to-use-a-multimeter/all>  
<https://randomnerdtutorials.com/how-to-use-a-multimeter/>]

This section is mostly addressed for beginners who are starting out in electronics and have no idea how to use a multimeter and how it can be useful. We'll explore the most common features on a multimeter and how to measure current, voltage, resistance and how to check continuity.

### 4.3.1 - Parts of a Multimeter



A multimeter is composed by four essential sections:

- **Display:** this is where the measurements are displayed
- **Selection knob:** this selects what you want to measure
- **Ports:** this is where you plug in the probes



- **Probes:** a multimeter comes with two probes. Generally one is red and the other is black.



Note: There isn't any difference between the red and the black probes, just the color.

So, assuming the convention:

- the black probe is always connected to the COM port.
- the red probe is connected to one of the other ports depending on what you want to measure.

#### Ports :

The “**COM**” or “–“ port is where the black probe should be connected. The COM probe is conventionally black.



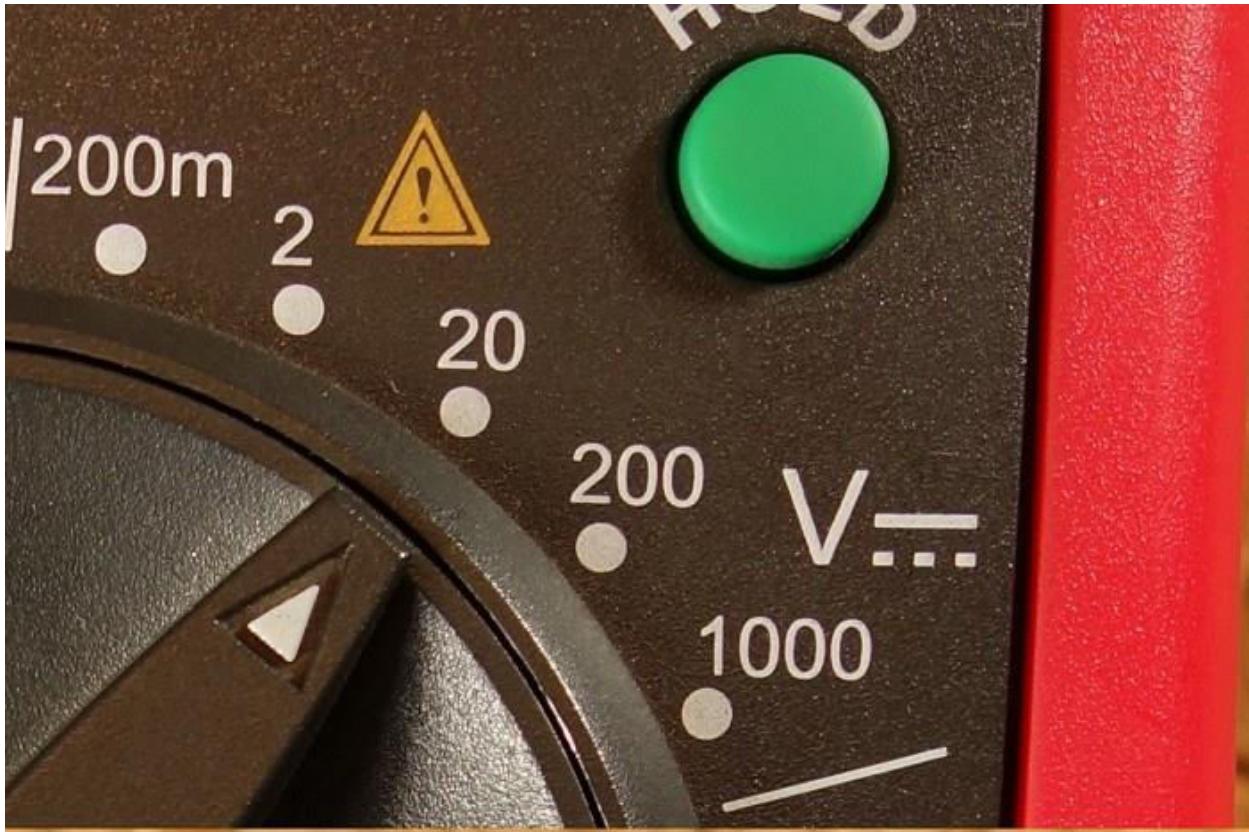
- 10A is used when measuring large currents, greater than 200mA

- $\mu\text{AmA}$  is used to measure current
- $\text{V}\Omega$  allows you to measure voltage and resistance and test continuity

**These ports can vary depending on the multimeter you're using.**

#### 4.3.2 -Measuring Voltage

You can measure DC voltage or AC voltage. The V with a straight line means DC voltage.



The V with the wavy line means AC voltage.



#### To measure voltage:

- 1) Set the mode to V with a wavy line if you're measuring AC voltage or to the V with a straight line if you're measuring DC voltage.
- 2) Make sure the red probe is connected to the port with a V next to it.
- 3) Connect the red probe to the positive side of your component, which is where the current is coming from.
- 4) Connect the COM probe to the other side of your component.
- 5) Read the value on the display.

**Tip:** to measure voltage you have to connect your multimeter in parallel with the component you want to measure the voltage. Placing the multimeter in parallel is placing each probe along with the leads of the component you want to measure the voltage.

#### Example: measuring a battery's voltage

In this example we're going to measure the voltage of a 1.5V battery. You know that you'll have approximately 1.5V. So, you should select a range with the selection knob that can read the 1.5V. So you should select 2V in the case of this multimeter. If you have an autorange multimeter, you don't have to worry about the range you need to select.



What if you didn't know what was the value of the voltage? If you need to measure the voltage of something, and you don't know the range in which the value will fall under, you need to try several ranges.

If the range you've selected is lower than the real value, on the display you'll read 1 as shown in the picture below. The 1 means that the voltage is higher than the range you've selected.



If you select a higher range, most part of the times you'll be able to read the value of the voltage, but with less accuracy.



**What happens if you switch the red and the black probe?**

The reading on the multimeter has the same value, but it's negative.



### **Example: measuring voltage in a circuit**

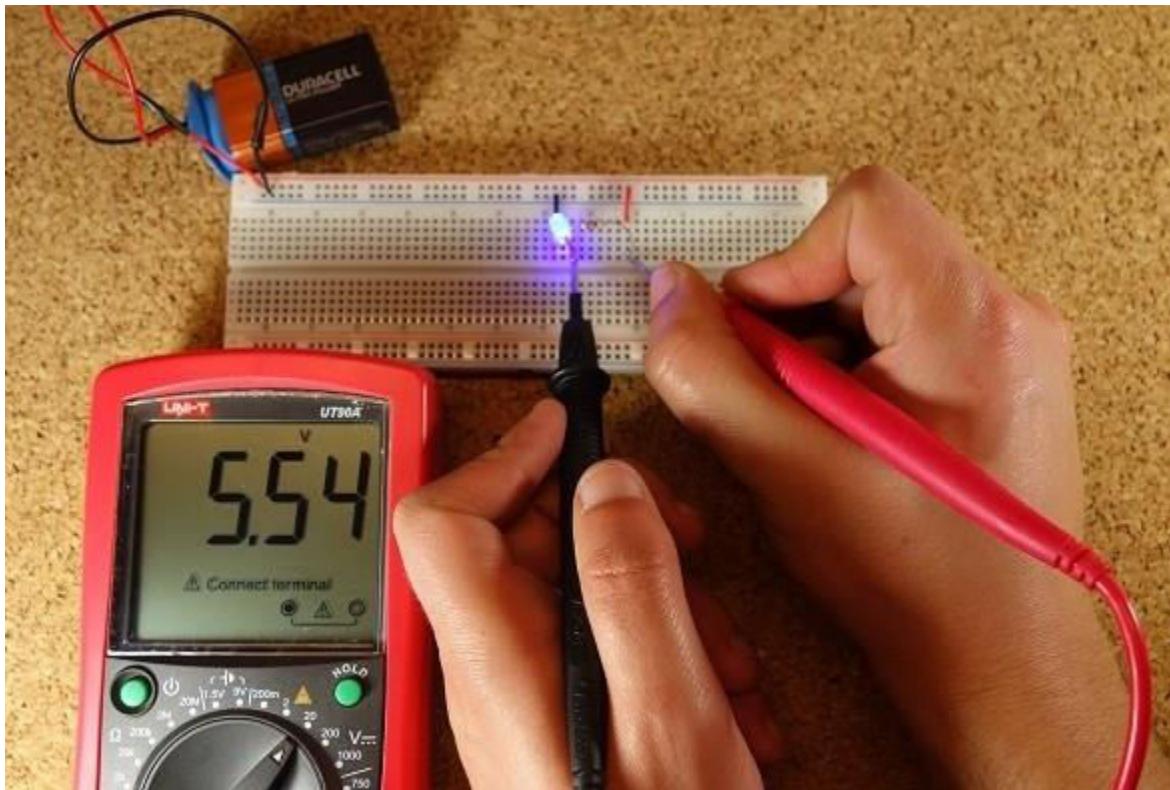
In this example we'll show you how to measure the voltage drop across a resistor in a simple circuit. This example circuit lights up an LED.

**TIP:** two components in parallel share voltage, so you should connect the multimeter probes in parallel with the component you want to measure the voltage.

To wire the circuit you need to connect an LED to 9V battery through a 470 Ohm resistor.

#### **To measure the voltage drop across the resistor:**

- 1) You just have to place the red probe in one lead of the resistor and the black probe on the other lead of the resistor.
- 2) The red probe should be connected to the part that the current is coming from.
- 3) Also, don't forget to make sure the probes are plugged in the right ports.

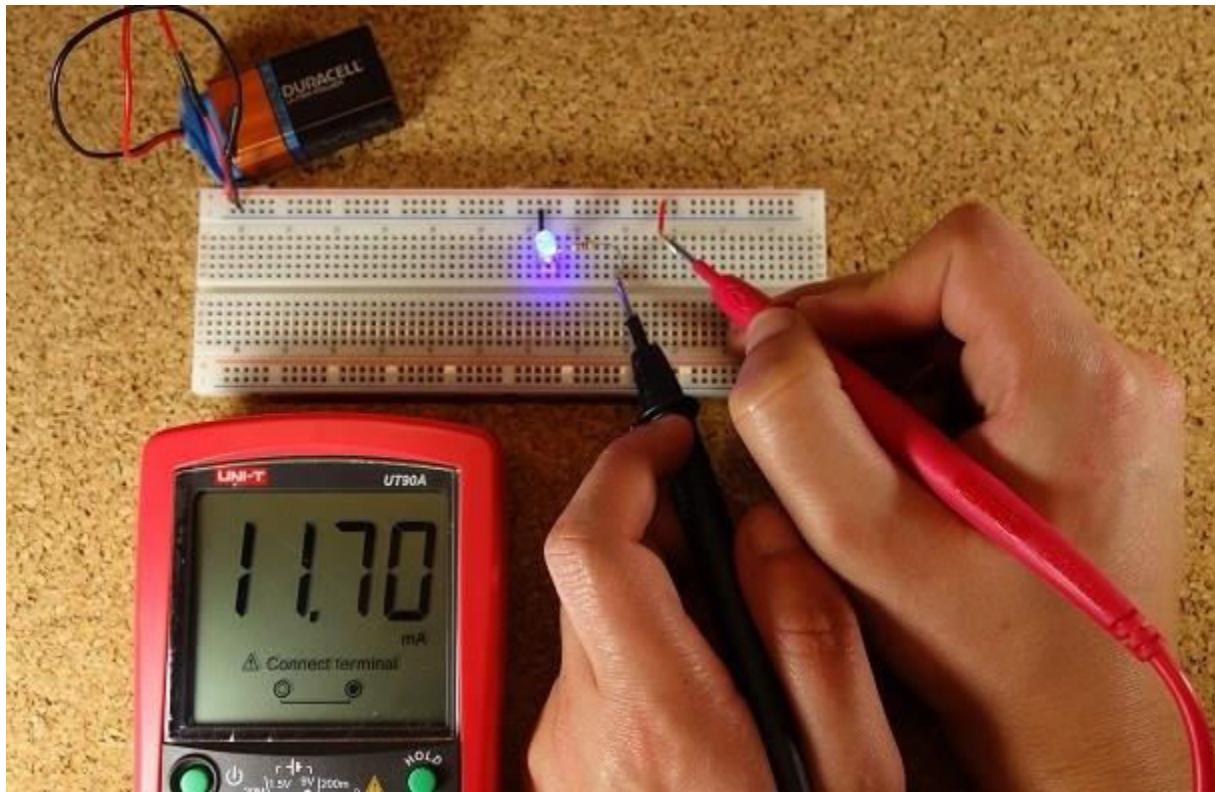
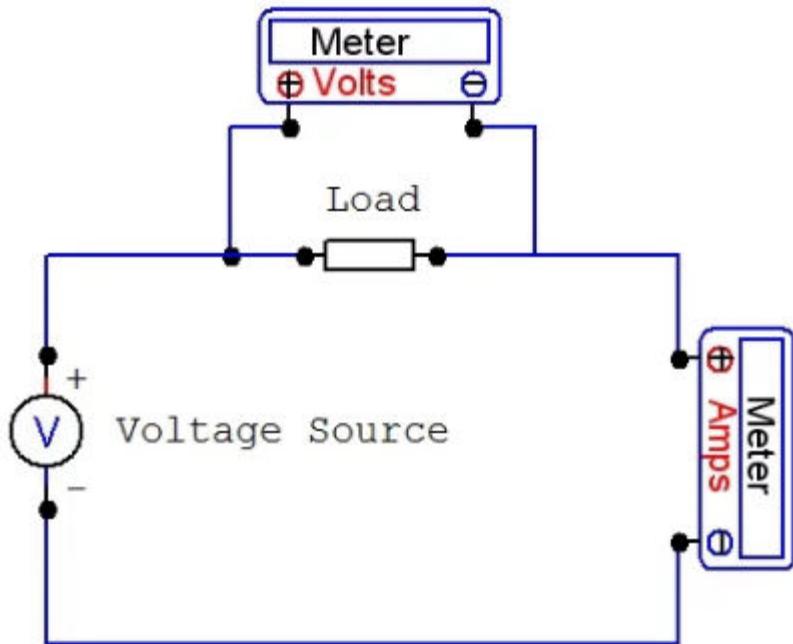


#### 4.3.3 - Measuring Current

To measure current you need to bear in mind that components in series share a current. So, you need to connect your multimeter in series with your circuit.

**TIP:** to place the multimeter in series, you need to place the red probe on the lead of a component and the black probe on the next component lead. The multimeter acts as if it was a wire in your circuit. If you disconnect the multimeter, your circuit won't work.

Before measuring the current, be sure that you've plugged in the red probe in the right port, in this case  $\mu\text{AmA}$ . In the example below, the same circuit of the previous example is used. The multimeter is part of the circuit.



#### 4.3.4 - Measuring Resistance

Plug the red probe into the right port and turn the selection knob to the resistance section. Then, connect the probes to the resistor leads. The way you connect the leads doesn't matter, the result is the same.



As you can see, the 470 $\Omega$  resistor, only has 461 $\Omega$ .

#### 4.3.5 - Checking Continuity

Most multimeters provide a feature that allows you to test the continuity of your circuit. This allows you to easily detect bugs such as faulty wires. It also helps you check if two points of the circuit are connected.

To use this functionality select the mode that looks like a speaker.



### How does continuity work?

If there is very low resistance between two points, which is less than a few ohms, the two points are electrically connected and you'll hear a continuous sound.

If the sound isn't continuous or if you don't hear any sound at all, it means that what you're testing has a faulty connection or isn't connected at all.

**WARNING:** To test continuity you should turn off the system! Turn off the power supply!

Touch the two probes together and, as they are connected, you'll hear a continuous sound. To test the continuity of a wire, you just need to connect each probe to the wire tips.



## 4.4- Read Analog Voltage (ADC)

[reference - <https://learn.sparkfun.com/tutorials/analog-to-digital-conversion/all>  
<https://programmingelectronics.com/tutorial-21-analog-input-old-version/>  
<https://www.arduino.cc/en/tutorial/AnalogInput>]

### 4.4.1 - The Analog World

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 inbetween. Luckily, nearly all microcontrollers have a device built

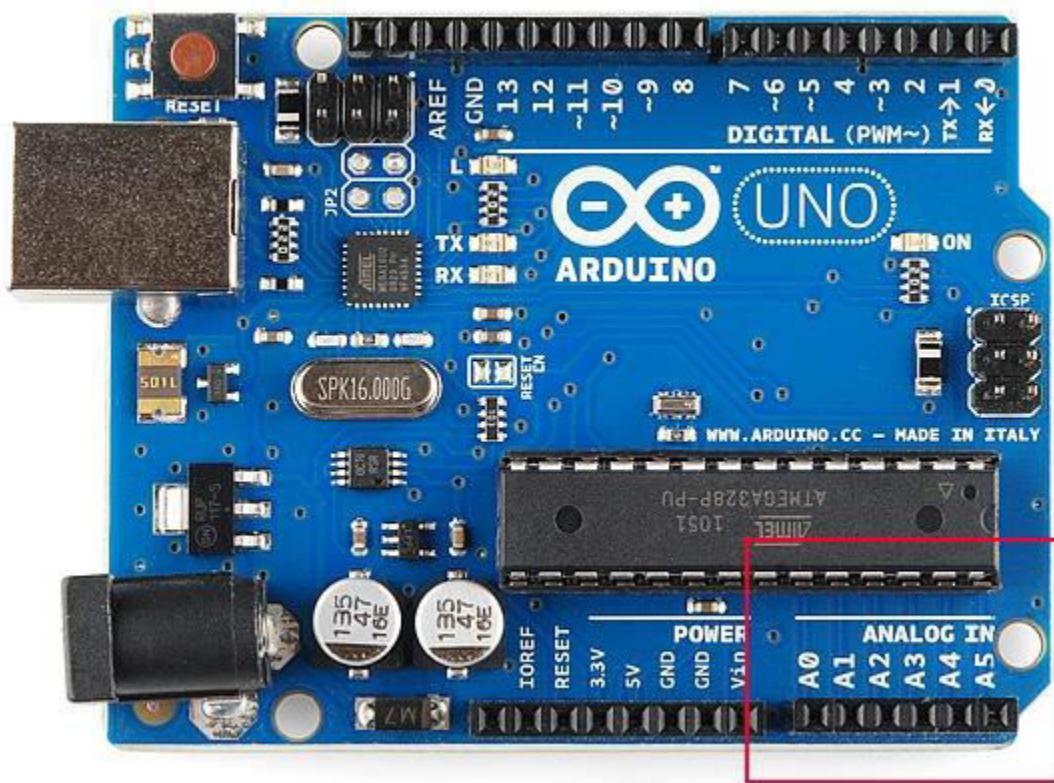
into them that allows us to convert these voltages into values that we can use in a program to make a decision.

#### 4.4.2 - 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 microcontroller. 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).

The way an ADC works is fairly complex. There are a few different ways to achieve this feat (see Wikipedia for a list), but one of the most common technique uses the analog voltage to charge up an internal capacitor and then measure the time it takes to discharge across an internal resistor. The microcontroller monitors the number of clock cycles that pass before the capacitor is discharged. This number of cycles is the number that is returned once the ADC is complete.



#### 4.4.3 - ADC value to voltage

The ADC reports a ratiometric value. This means that the ADC assumes 5V is 1023 and anything less than 5V will be a ratio between 5V and 1023.

$$\frac{\text{Resolution of the ADC}}{\text{System Voltage}} = \frac{\text{ADC Reading}}{\text{Analog Voltage Measured}}$$

Analog to digital conversions are dependant on the system voltage. Because we predominantly use the 10-bit ADC of the Arduino on a 5V system, we can simplify this equation slightly:

$$\frac{1023}{5} = \frac{\text{ADC Reading}}{\text{Analog Voltage Measured}}$$

If your system is 3.3V, you simply change 5V out with 3.3V in the equation. If your system is 3.3V and your ADC is reporting 512, what is the voltage measured? It is approximately 1.65V.

If the analog voltage is 2.12V what will the ADC report as a value?

$$\frac{1023}{5.00V} = \frac{x}{2.12V}$$

Rearrange things a bit and we get:

$$\frac{1023}{5.00V} * 2.12V = x$$

$$x = 434$$

Ahah! The ADC should report 434. On an Arduino UNO, for example, this yields a resolution between readings of: 5 volts / 1024 units or, 0.0049 volts (4.9 mV) per unit.

#### **Example Code for ADC read :**

```
int analogPin = A0; // potentiometer wiper (middle terminal) connected to
analog pin A0
 // outside leads to ground and +5V
int val = 0; // variable to store the value read

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

void loop() {
 val = analogRead(analogPin); // read the input pin
 Serial.println(val); // debug value
```

```
}
```

**Example Code for Voltage Reading :** Arduino IDE > File > Examples > ReadAnalogVoltage

```
void setup() {
 // initialize serial communication at 9600 bits per second:
 Serial.begin(9600);
}

// the loop routine runs over and over again forever:
void loop() {
 // read the input on analog pin 0:
 int sensorValue = analogRead(A0);
 // Convert the analog reading (which goes from 0 - 1023) to a voltage (0
 - 5V):
 float voltage = sensorValue * (5.0 / 1023.0);
 // print out the value you read:
 Serial.println(voltage);
}
```

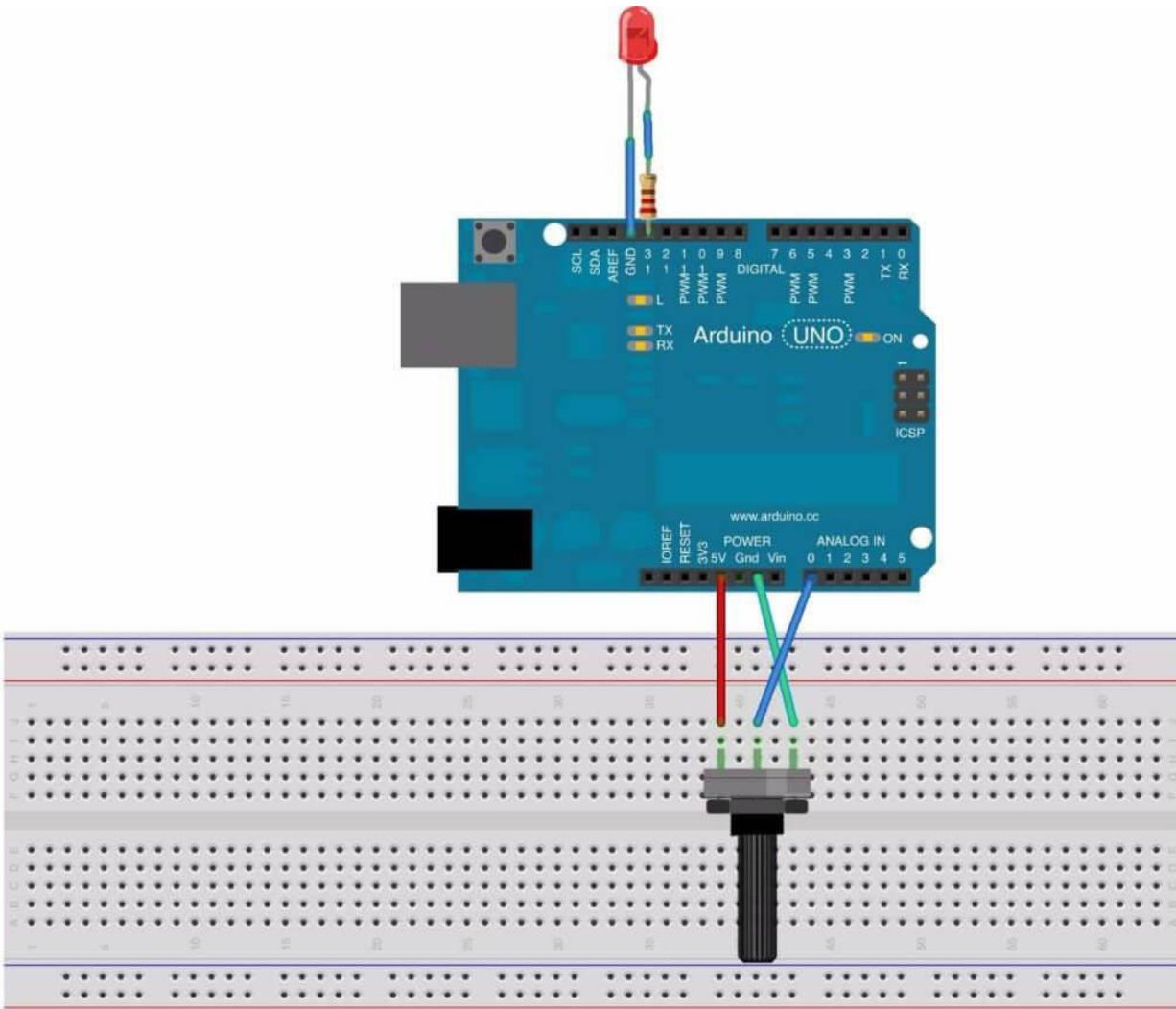
**Example Code for LED Blinking Delay control with ADC:**

```
int sensorPin = A0; // select the input pin for the potentiometer
int ledPin = 13; // select the pin for the LED
int sensorValue = 0; // variable to store the value coming from the sensor

void setup() {
 // declare the ledPin as an OUTPUT:
 pinMode(ledPin, OUTPUT);
}

void loop() {
 // read the value from the sensor:
 sensorValue = analogRead(sensorPin);
 // turn the ledPin on
 digitalWrite(ledPin, HIGH);
 // stop the program for <sensorValue> milliseconds:
 delay(sensorValue);
 // turn the ledPin off:
 digitalWrite(ledPin, LOW);
 // stop the program for for <sensorValue> milliseconds:
 delay(sensorValue);
```

}

**Circuit :****College Level :****4.5 - Voltage Divider Details**

[reference - <https://www.allaboutcircuits.com/tools/voltage-divider-calculator/>]

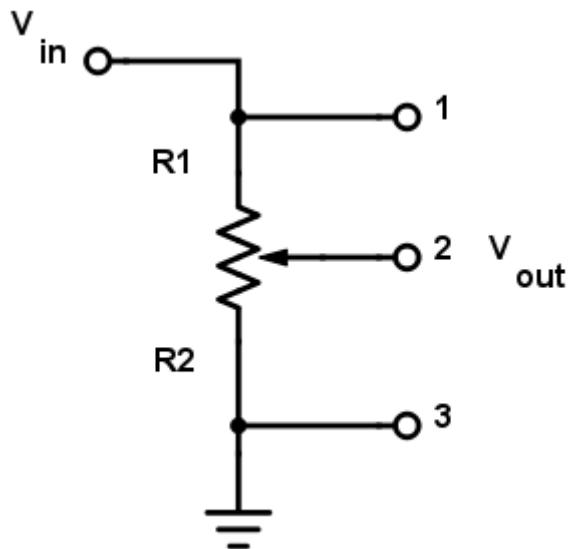
**Applications**

Since voltage dividers are fairly common, they can be found in a number of applications. Below are just some of the places where this circuit is found.

**4.5.1 - Potentiometers**



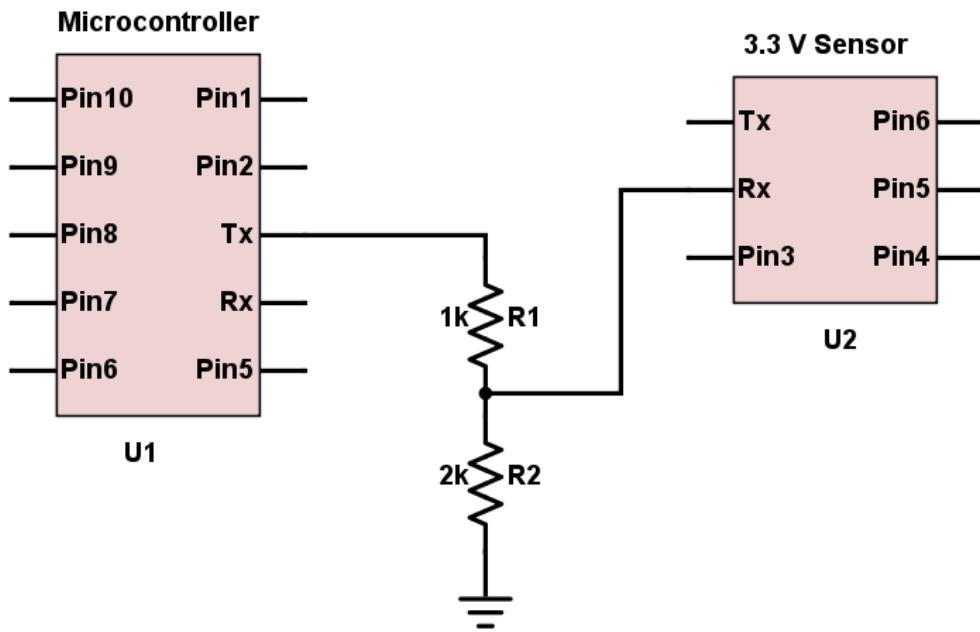
Perhaps the most common voltage divider circuit is the one involving a potentiometer, which is a variable resistor. A potentiometer's schematic diagram is shown below:



A "pot" usually has three external pins: two are the ends of the resistor and one is connected to the wiper arm. The wiper cuts the resistor in two and moves it, adjusting the ratio between the top half and the bottom half of the resistor. Connect the two external pins to a voltage (input) and a reference (ground) with the middle (wiper pin) as your output pin and you have yourself a voltage divider.

#### 4.5.2 - Level Shifters

Another area where voltage dividers are useful is when a voltage needs to be leveled down. The most common scenario is when interfacing signals between a sensor and a microcontroller with two different voltage levels. Most microcontrollers operate at 5V while some sensors can only accept a maximum voltage of 3.3V. Naturally, you want to level the voltage from the microcontroller down to avoid damage to the sensor. An example circuit is shown below:



The circuit above shows a voltage divider circuit involving a  $2k\Omega$  and a  $1k\Omega$  resistor. If the voltage from the microcontroller is 5V, then the leveled-down voltage to the sensor is calculated as:

$$V_{out} = 5 * \frac{2k\Omega}{2k\Omega + 1k\Omega} = 3.33V$$

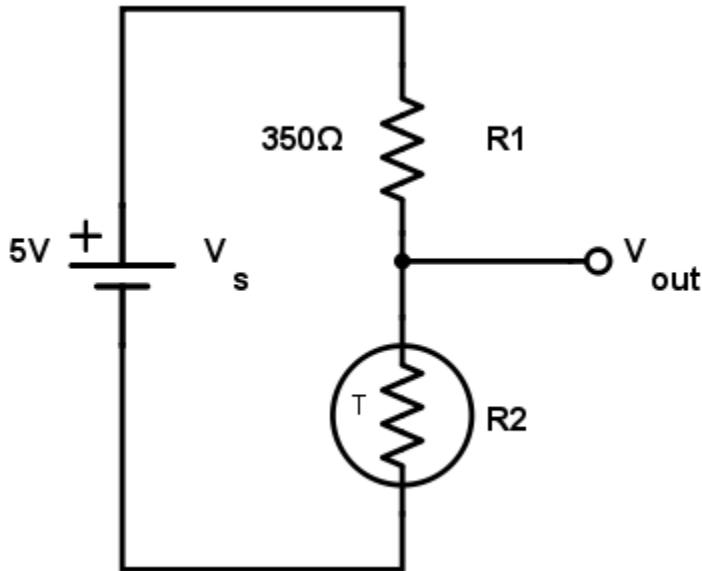
This voltage level is now safe for the sensor to handle. Take note that this circuit only works for leveling down voltages and not leveling up.

Below are some other resistor combinations used for leveling down commonly encountered voltages:

| Resistor Combination | Use         |
|----------------------|-------------|
| 4.7 kΩ and 6.8 kΩ    | 12V to 5V   |
| 4.7 kΩ and 3.9 kΩ    | 9V to 5V    |
| 3.6 kΩ and 9.1 kΩ    | 12V to 3.3V |
| 3.3 kΩ and 5.7 kΩ    | 9V to 3.3V  |

#### 4.5.3 - Resistive Sensor Reading

A lot of sensors are resistive devices and most microcontrollers read voltage, not resistance. Thus, a resistive sensor is usually connected in a voltage divider circuit with a resistor in order to interface with a microcontroller. An example setup is shown below:



A thermistor is a sensor whose resistance changes proportionally to temperature. Let us say that the thermistor has a room temperature resistance of 350Ω. The paired resistance is chosen to also be 350Ω.

When the thermistor is at room temperature, the output voltage is:

$$V_{out} = 5 * \frac{350\Omega}{350\Omega + 350\Omega} = 2.5V$$

When the temperature increases, the thermistor resistance changes to 350.03Ω, the output changes to:

$$V_{out} = 5 * \frac{350.03\Omega}{350\Omega + 350.03\Omega} = 2.636V$$

Such a small change in voltage is detectable by a microcontroller. If the thermistor transfer function is known, the equivalent temperature can now be calculated.

## Class 5:

- 5.1 - Discuss about previous class
- 5.2 - Loop
- 5.3 - Code - Even number & Odd Number
- 5.4 - LED blinking using loop
- 5.5 - Serial Plotter
- College Level :**
- 5.6 - ADC Resolution

### 5.1 - Discuss about previous class

Complete previous class if any topics was not finished.

### 5.2 - Loop

[reference - <https://www.programiz.com/c-programming/c-for-loop>  
<https://startingelectronics.org/software/arduino/learn-to-program-course/07-for-loop/>  
<https://beginnersbook.com/2014/01/c-for-loop/>]

Loops are used in programming to repeat a block of code until a specific condition is met. An increment counter is usually used to increment and terminate the loop. The for statement is useful for any repetitive operation, and is often used in combination with arrays to operate on collections of data/pins. There are three loops in C programming:

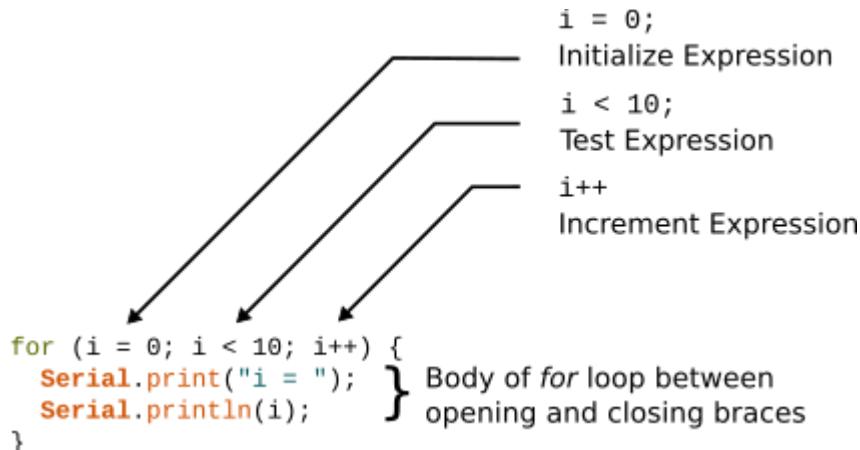
- 1) for loop
- 2) while loop
- 3) do...while loop

#### 5.2.1 - for loop

The syntax of for loop is:

```
for (initializationStatement; testExpression; updateStatement)
{
 // codes
}
```

#### How for loop works?



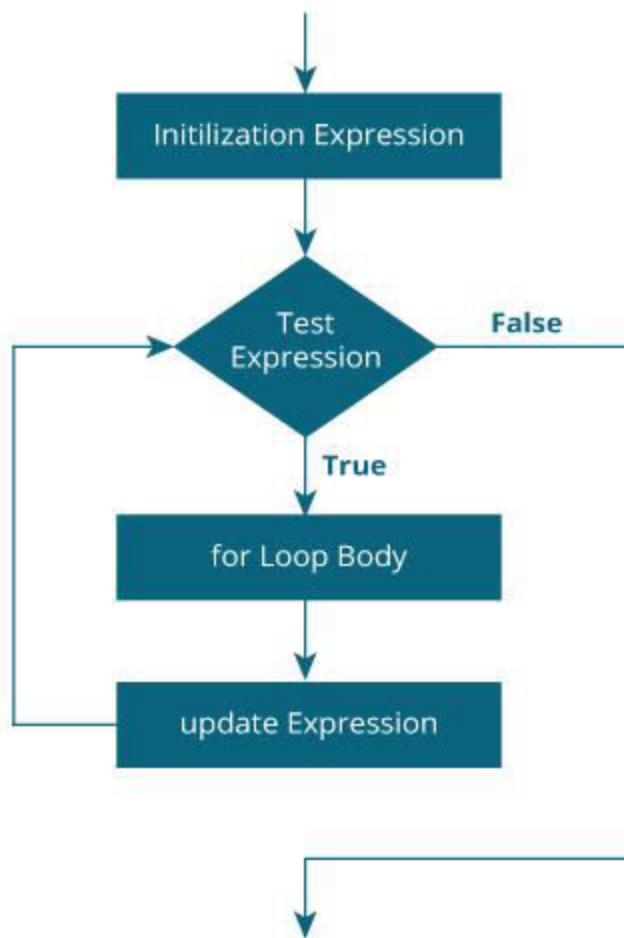
The initialization statement is executed only once.

Then, the test expression is evaluated. If the test expression is false (0), for loop is terminated. But if the test expression is true (nonzero), codes inside the body of for loop is executed and the update expression is updated.

This process repeats until the test expression is false.

The for loop is commonly used when the number of iterations is known.

### for loop Flowchart



### Example 1: for loop in C

```

// Program to calculate the sum of first n natural numbers
// Positive integers 1,2,3...n are known as natural numbers

#include <stdio.h>
int main()
{
 int num, count, sum = 0;

 printf("Enter a positive integer: ");
 scanf("%d", &num);

 // for loop terminates when n is less than count
 for(count = 1; count <= num; ++count)
 {
 sum += count;
 }
}

```

```

 printf("Sum = %d", sum);

 return 0;
}

```

output :  
Enter a positive integer: 10  
Sum = 55

### **Example 2: Nested For Loop in C**

```

#include <stdio.h>
int main()
{
 for (int i=0; i<2; i++)
 {
 for (int j=0; j<4; j++)
 {
 printf("%d, %d\n", i ,j);
 }
 }
 return 0;
}

```

Output :  
0, 0  
0, 1  
0, 2  
0, 3  
1, 0  
1, 1  
1, 2  
1, 3

In the above example we have a for loop inside another for loop, this is called nesting of loops.

### **Example 3: For Loop in Arduino**

**Video : For Loop in Arduino [<https://youtu.be/5a2im1PcVjc>]**

```

void setup() {
 int i;

 Serial.begin(9600);
}

```

```

for (i = 0; i < 10; i++) {
 Serial.print("i = ");
 Serial.println(i);
}
}

void loop() {
}

```

### 5.2.2 - while loop

[reference - <https://www.programiz.com/c-programming/c-do-while-loops>]

The syntax of a while loop is:

```

while (testExpression)
{
 //codes
}

```

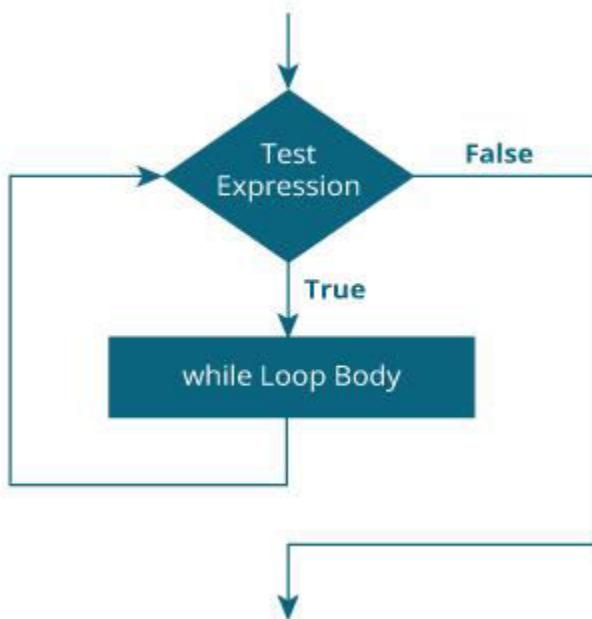
#### How while loop works?

The while loop evaluates the test expression.

If the test expression is true (nonzero), codes inside the body of while loop is executed. The test expression is evaluated again. The process goes on until the test expression is false.

When the test expression is false, the while loop is terminated.

#### Flowchart of while loop



**Example 1: While loop in C**

```
#include <stdio.h>
int main()
{
 int count=1;
 while (count <= 4)
 {
 printf("%d ", count);
 count++;
 }
 return 0;
}
```

**Output:**

1 2 3 4

**Example 2: While loop in Arduino**

```
void setup() {
 int i = 0;

 Serial.begin(9600);

 while (i < 10) {
 Serial.print("i = ");
 Serial.println(i);
 i++;
 }
}

void loop() {
```

**5.2.3 - do ... while loop**

The do..while loop is similar to the while loop with one important difference. The body of do...while loop is executed once, before checking the test expression. Hence, the do...while loop is executed at least once.

**do...while loop Syntax**

```
do
{
```

```
// codes
}
while (testExpression);
```

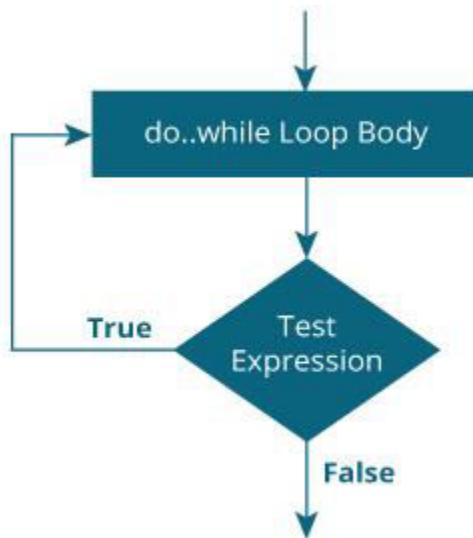
### How do...while loop works?

The code block (loop body) inside the braces is executed once.

Then, the test expression is evaluated. If the test expression is true, the loop body is executed again. This process goes on until the test expression is evaluated to 0 (false).

When the test expression is false (nonzero), the do...while loop is terminated.

### Flowchart of do...while Loop



### Example 1: do while in C

```
#include <stdio.h>

int main()
{
 int i = 1;
 do{
 printf("%d",i);
 i++;
 }
 while(i < 11);

 return 0;
}
```

**Output :** 12345678910

### Example 2: do while in C

```
#include<stdio.h>
int main()
{
 int i = 1; // declare and initialize i to 1

 do
 {
 // check whether i is multiple of 3 not or not
 if(i % 3 == 0)
 {
 printf("%d ", i); // print the value of i
 }
 i++; // increment i by 1
 }while(i < 100); // stop the loop when i becomes greater than 100

 // signal to operating system everything works fine
 return 0;
}
```

### Example 3: do while in Arduino

```
void setup() {
 int sum = 0;

 Serial.begin(9600);

 // count up to 25 in 5s
 do {
 sum = sum + 5;
 Serial.print("sum = ");
 Serial.println(sum);
 delay(500); // 500ms delay
 } while (sum < 25);
}

void loop() {
```

### 5.3 - Code - Even number & Odd Number

Example 1 : Using if else

```
#include <stdio.h>
int main()
{
 int number;

 printf("Enter an integer: ");
 scanf("%d", &number);

 // True if the number is perfectly divisible by 2
 if(number % 2 == 0)
 printf("%d is even.", number);
 else
 printf("%d is odd.", number);

 return 0;
}
```

**Example 2 : Using Conditional Operator**

```
#include <stdio.h>
int main()
{
 int number;

 printf("Enter an integer: ");
 scanf("%d", &number);

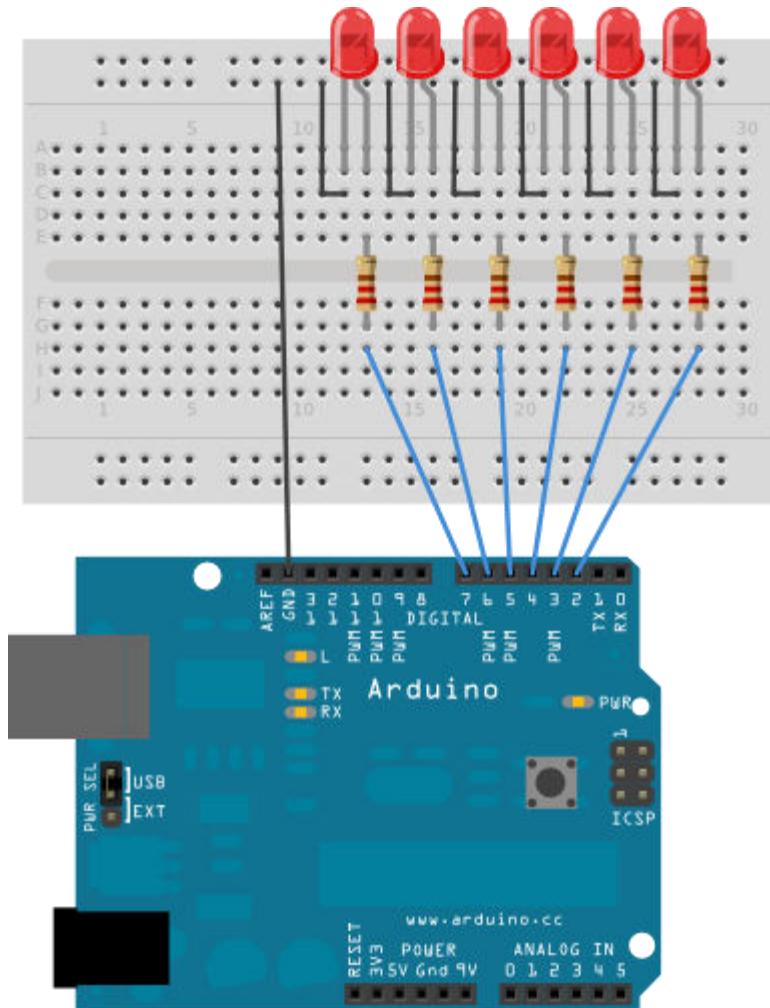
 (number % 2 == 0) ? printf("%d is even.", number) : printf("%d is
odd.", number);

 return 0;
}
```

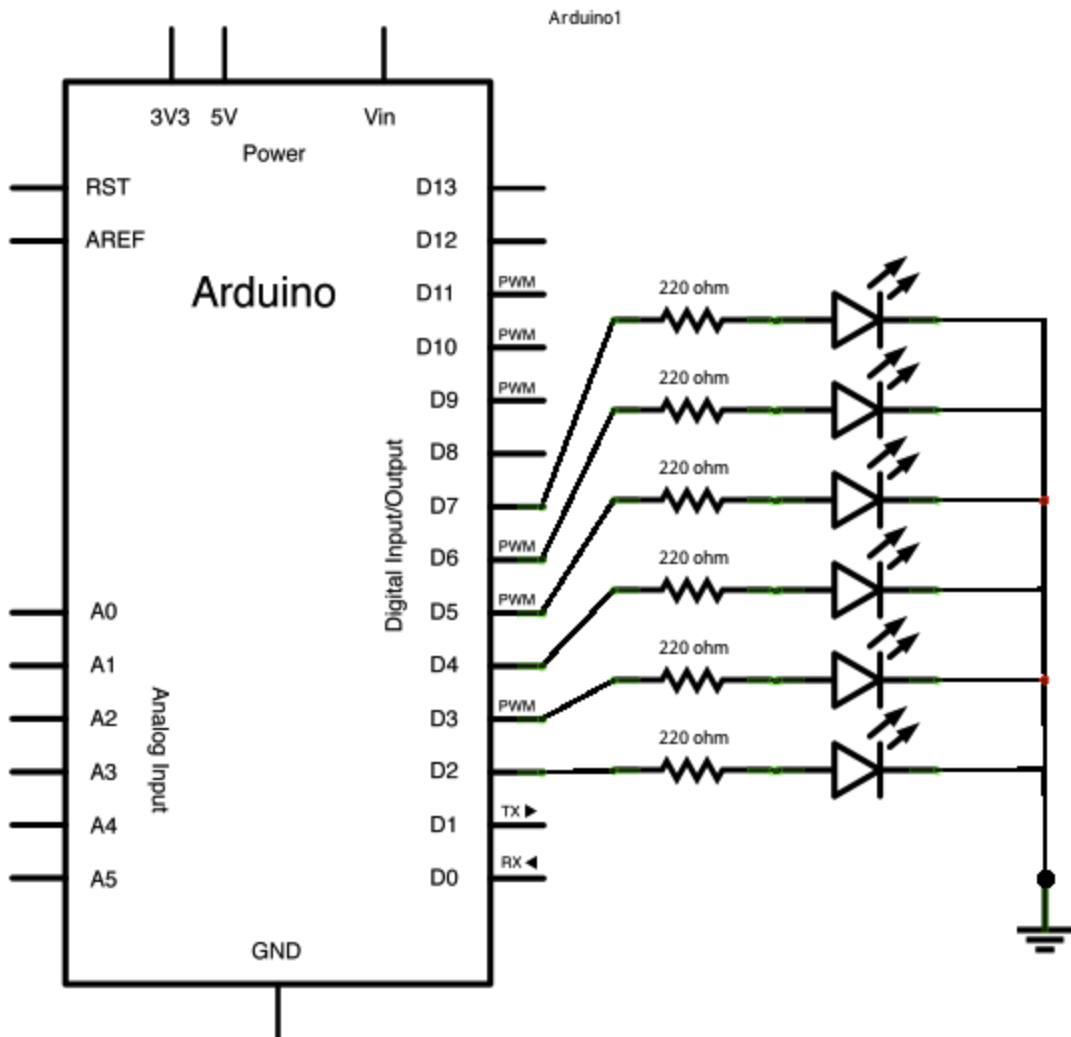
### 5.4 - LED blinking using loop

[reference - <https://www.arduino.cc/en/Tutorial/ForLoopIteration>]

**Circuit :**



**Schematic :**

**Code :**

```

int timer = 100; // The higher the number, the slower the timing.

void setup() {
 // use a for loop to initialize each pin as an output:
 for (int thisPin = 2; thisPin < 8; thisPin++) {
 pinMode(thisPin, OUTPUT);
 }
}

void loop() {
 // loop from the lowest pin to the highest:
 for (int thisPin = 2; thisPin < 8; thisPin++) {
 // turn the pin on:
 }
}

```

```

 digitalWrite(thisPin, HIGH);
 delay(timer);
 // turn the pin off:
 digitalWrite(thisPin, LOW);
}

// loop from the highest pin to the lowest:
for (int thisPin = 7; thisPin >= 2; thisPin--) {
 // turn the pin on:
 digitalWrite(thisPin, HIGH);
 delay(timer);
 // turn the pin off:
 digitalWrite(thisPin, LOW);
}
}
}

```

## 5.5 - Serial Plotter

[reference - <https://learn.adafruit.com/experimenters-guide-for-metro/circ08-using%20the%20arduino%20serial%20plotter>]

Arduino comes with a cool tool called the Serial Plotter. It can give you visualizations of variables in real-time. This is super useful for visualizing data, troubleshooting your code, and visualizing your variables as waveforms.

**Code:**

```

int sensorPin = A0; // select the input pin for the potentiometer
int ledPin = 13; // select the pin for the LED
int sensorValue = 0; // variable to store the value coming from the sensor

void setup() {
 // declare the ledPin as an OUTPUT:
 pinMode(ledPin, OUTPUT);
 // begin the serial monitor @ 9600 baud
 Serial.begin(9600);
}

void loop() {
 // read the value from the sensor:
 sensorValue = analogRead(sensorPin);

 Serial.println(sensorValue);
}

```

```
Serial.print(" ");
delay(20);
}
```

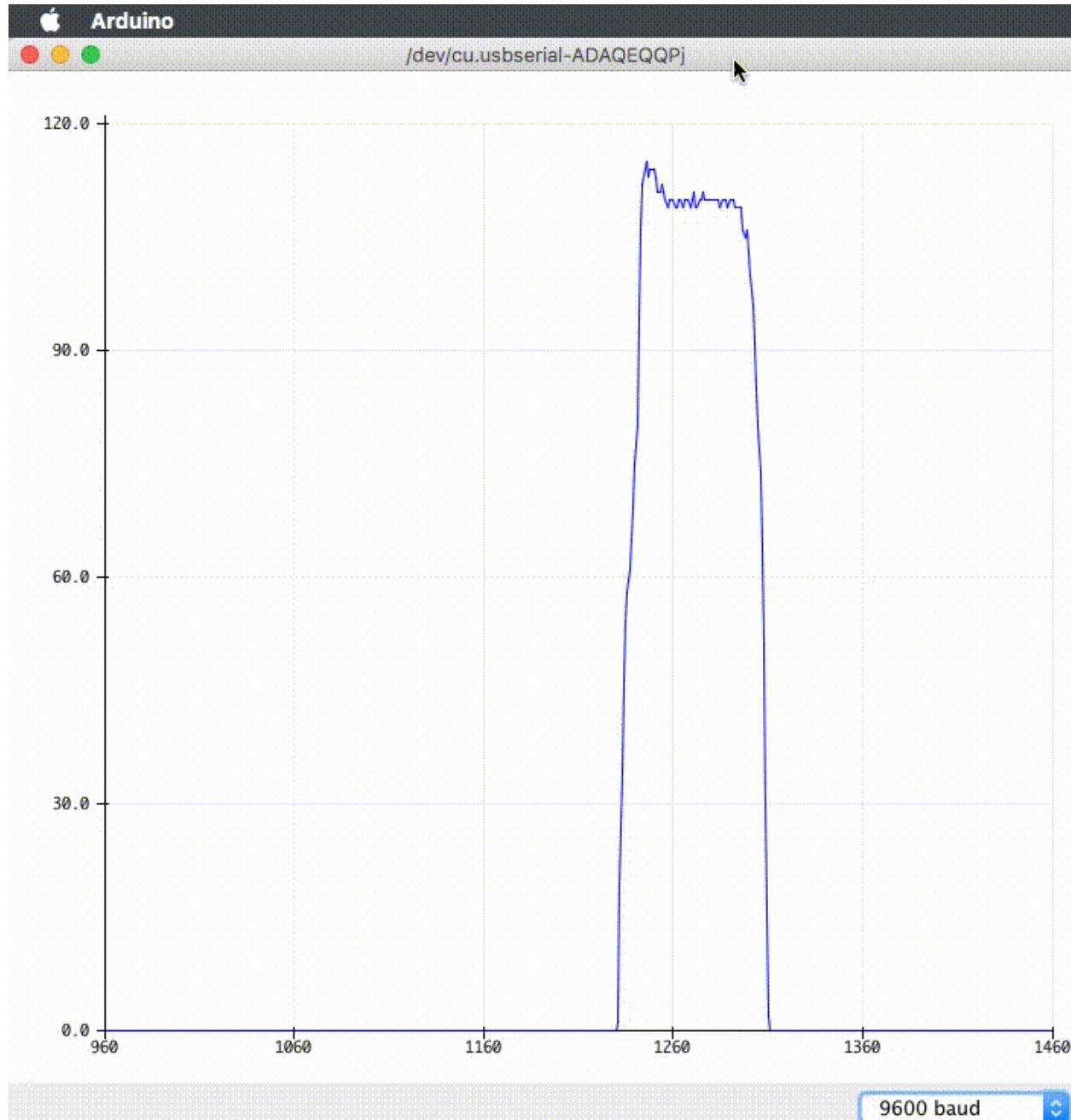
When you call Serial.println(value), the Serial Plotter will put that variable on the plot. The Y-Axis of the plot auto-resizes.

If you want to plot multiple variables, you'll need a Serial.println(value) call for each of the variables, separated by a Serial.print(" ") or Serial.print("\t"):

```
Serial.print(variable1);
Serial.print(" ");
Serial.println(variable2);
```

Let's try it out with the new code above. Compile and upload the program above, then navigate to Tools > Serial Plotter. The code uses a baud rate of 9600, make sure it's set in the serial monitor as 9600 too.

You should see something like this when you twist the trim potentiometer around:



College Level :

## 5.6 - ADC Resolution

[reference - <https://www.arduino.cc/en/Reference/AnalogReadResolution>]

The resolution of an ADC (A/D) converter is defined as the smallest change in the value of an input signal that changes the value of the digital output by one count. For example if you develop a machine which can slice a pizza in two parts vs a machine which can slice a pizza in four parts we can say second machine has greater resolution than the first machine.

`analogReadResolution()` is an extension of the Analog API for the Arduino Due, Arduino and Genuino Zero and MKR1000.

Sets the size (in bits) of the value returned by `analogRead()`. It defaults to 10 bits (returns values between 0-1023) for backward compatibility with AVR based boards.

The Due, Zero and MKR Family boards have 12-bit ADC capabilities that can be accessed by changing the resolution to 12. This will return values from `analogRead()` between 0 and 4095.

### Syntax

```
analogReadResolution(bits)
```

**bits:** determines the resolution (in bits) of the value returned by `analogRead()` function. You can set this 1 and 32. You can set resolutions higher than 12 but values returned by `analogRead()` will suffer approximation. See the note below for details.

### Example Code :

```
void setup() {
 // open a serial connection
 Serial.begin(9600);
}

void loop() {
 // read the input on A0 at default resolution (10 bits)
 // and send it out the serial connection
 analogReadResolution(10);
 Serial.print("ADC 10-bit (default) : ");
 Serial.print(analogRead(A0));

 // change the resolution to 12 bits and read A0
 analogReadResolution(12);
 Serial.print(", 12-bit : ");
 Serial.print(analogRead(A0));

 // change the resolution to 16 bits and read A0
 analogReadResolution(16);
 Serial.print(", 16-bit : ");
```

```
Serial.print(analogRead(A0));

// change the resolution to 8 bits and read A0
analogReadResolution(8);
Serial.print(", 8-bit : ");
Serial.println(analogRead(A0));

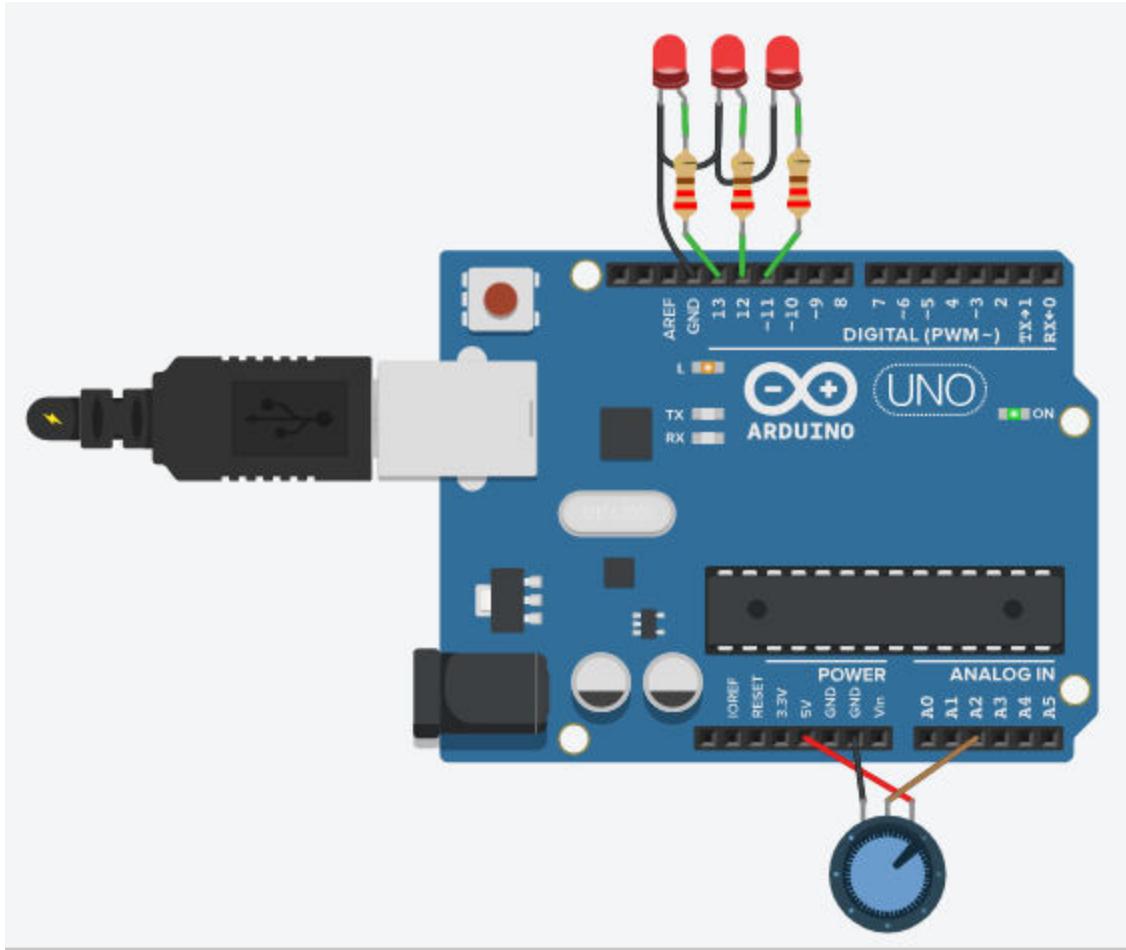
// a little delay to not hog serial monitor
delay(100);
}
```

## Class 6:

- 6.1 - Multiple LED control based on if (LED Bar)
  - 6.2 - LED control with Serial Read
  - 6.3 - LED on off using Processing
  - 6.4 - Toggle
  - 6.5 - Nested loop
  - 6.6 - Increment, Decrement using for loop
- College Level:**
- 6.7 - EEPROM
  - 6.8 - Pyramid print with start

### 6.1 - Multiple LED control based on if (LED Bar)

Now we will control LED using if condition and analogRead . Connect three LED with three pin of arduino with resistor .

**Code :**

```
//Connect three LED with three pin of arduino with resistor
//Connect a potentiometer with A2
int potPin = 2;
int ledPin1 = 11;
int ledPin2 = 12;
int ledPin3 = 13;
int val = 0; // variable to store the value coming from the sensor

void setup() {
 pinMode(ledPin1, OUTPUT);
 pinMode(ledPin2, OUTPUT);
 pinMode(ledPin3, OUTPUT);
 Serial.begin(9600);
}

void loop() {
```

```

val = analogRead(potPin); // read the value from the sensor
if(val < 75){
 digitalWrite(ledPin1, LOW);
 digitalWrite(ledPin2, LOW);
 digitalWrite(ledPin3, LOW);
}
if((val > 75)&&(val < 400)){
 digitalWrite(ledPin1, HIGH);
 digitalWrite(ledPin2, LOW);
 digitalWrite(ledPin3, LOW);
}
if((val > 400)&&(val < 800)){
 digitalWrite(ledPin1, HIGH);
 digitalWrite(ledPin2, HIGH);
 digitalWrite(ledPin3, LOW);
}
if(val > 800){
 digitalWrite(ledPin1, HIGH);
 digitalWrite(ledPin2, HIGH);
 digitalWrite(ledPin3, HIGH);
}
Serial.println(val);
}

```

## 6.2 - LED control with Serial Read

In this section we will control built in LED of arduino with serial. Upload this code and open arduino serial monitor. If you write **H** or **h** in serial monitor and press enter the LED will be ON and if you enter **L** or **l** LED will be OFF. You should expand this code to control many LED's.

```

int ledPin = 13;
char state;
void setup() {
 pinMode(ledPin, OUTPUT);
 Serial.begin(9600);
}
void loop() {
 if (Serial.available() > 0) {
 state = Serial.read();
 if (state == 'H' || state == 'h') {
 digitalWrite(ledPin, HIGH);
 }
 }
}

```

```

if (state == 'L' || state == 'l') {
 digitalWrite(ledPin, LOW);
}
Serial.println(state);
delay(50);
}

```

## 6.3 - LED on off using Processing

[reference - <https://processing.org/>]

Processing is a flexible software sketchbook and a language for learning how to code within the context of the visual arts. Since 2001, Processing has promoted software literacy within the visual arts and visual literacy within technology. There are tens of thousands of students, artists, designers, researchers, and hobbyists who use Processing for learning and prototyping.

### 6.3.1 - Download & Install

Download and Install processing software from this link <https://processing.org/download/>

### 6.3.2 - Processing Code

**Video : Processing and Arduino EP#1 buttons [<https://youtu.be/NHUB6PBxI6M>]**

Before running this code make sure you have downloaded **ControlP5** library in processing.

To download ControlP5 library go Sketch > Import Library > Add Library than search for ControlP5 and Install. One important thing is check serial port of arduino IDE and edit **port = new Serial(this, "COM10", 9600);** according to your serial port of arduino.

```

//https://youtu.be/NHUB6PBxI6M
import controlP5.*; //library
import processing.serial.*; //library
Serial port; //do not change
ControlP5 cp5; //create ControlP5 object
PFont font; //do not change

void setup() { //same as arduino program

 size(300, 300); //window size, (width, height)
 port = new Serial(this, "COM10", 9600); //connected arduino port
 cp5 = new ControlP5(this); //do not change
 font = createFont("Georgia Bold", 20); //font for buttons and title
}

```

```

cp5.addButton("ON") //name of button
 .setPosition(95, 50) //x and y upper left corner of button
 .setSize(120, 70) //(width, height)
 .setFont(font) //font
 .setColorBackground(color(#23F73D)) //background r,g,b
 .setColorForeground(color(#E8F723)) //mouse over color r,g,b
 .setColorLabel(color(0, 0, 0)) //text color r,g,b
 ;
}

cp5.addButton("OFF")
 .setPosition(95, 150)
 .setSize(120, 70)
 .setFont(font)
 .setColorBackground(color(255, 0, 0))
 .setColorForeground(color(0, 255, 0))
 .setColorLabel(color(0, 0, 0))
 ;
}

void draw() {

 background(#93FCFF); // background color of window (r, g, b)
 textAlign(CENTER);
 text("LED Controller",150,245);
 text("Cybernetics Robo Academy",150,265); // x , y position
}

void ON() {
 port.write(1);
}

void OFF() {
 port.write(2);
}

```

### 6.3.3 - Arduino Code

```

void setup() {
 pinMode(13, OUTPUT); //set pin as output , red led
 Serial.begin(9600); //start serial
}

```

```

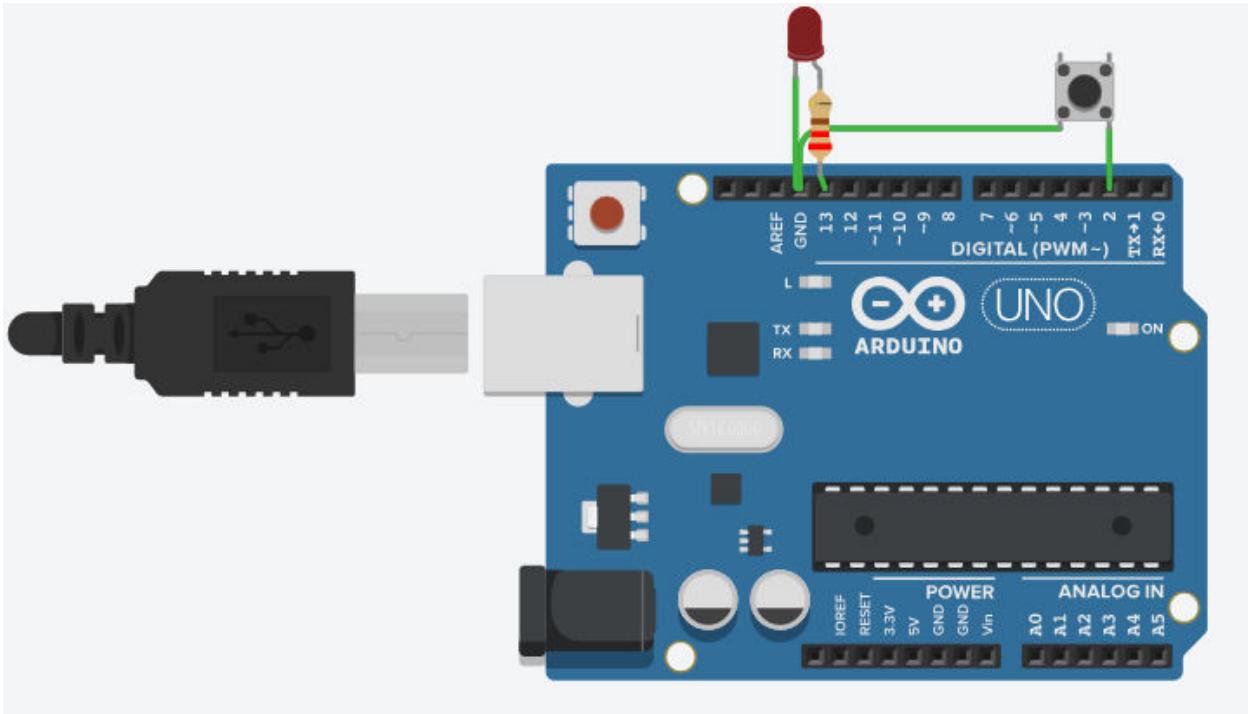
void loop() {

 if (Serial.available()) { //id data available
 int val = Serial.read();
 if (val == 1) { //if 1 received
 digitalWrite(13, HIGH); //turn on
 }
 if (val == 2) { //if 2 received
 digitalWrite(13, LOW); //turn off
 }
 }
}

```

## 6.4 - Toggle

In this section we will toggle a led that means if we press push button it will on and another time it will off.



### Arduino Code :

```

Simple toggle switch
Created by: P.Agiakatsikas

```

```

int button = 2;
int led = 13;
int status = false;

void setup() {
 pinMode(led, OUTPUT);
 pinMode(button, INPUT_PULLUP); // set the internal pull up resistor,
unpressed button is HIGH
 Serial.begin(9600);
}

void loop() {
 //a) if the button is not pressed the false status is reversed by !status
and the LED turns on
 //b) if the button is pressed the true status is reversed by !status and
the LED turns off

 if (digitalRead(button) == LOW) {
 Serial.print("status 1:");
 Serial.println(status);
 status = !status;
 digitalWrite(led, status);
 Serial.print("status 2:");
 Serial.println(status);
 }
 while (digitalRead(button) == LOW); //to prevent auto toggle
 delay(50); // Small delay
}

```

## 6.5 - Nested loop

Nested loop means loop inside loop.

**Nested loop syntax :**

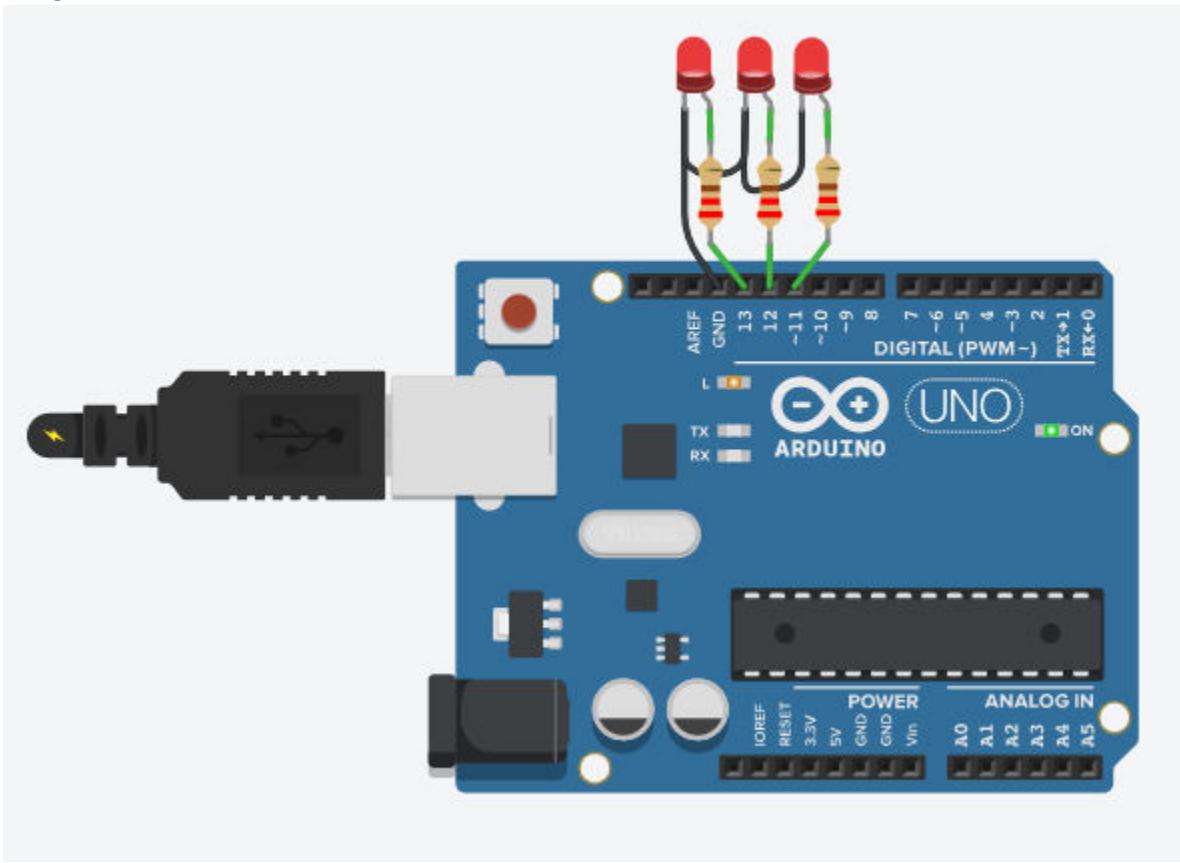
```

for (initialize ;control; increment or decrement) {
 // statement block
 for (initialize ;control; increment or decrement) {
 // statement block
 }
}

```

### 6.5.1 - Arduino LED Blinking using Nested for loop

Diagram :



### Arduino Code

```
//Syed Razwanul Haque Nabil
//www.cruxbd.com
//www.cyberneticsroboacademy.com
int ledPin1 = 11;
int ledPin2 = 12;
int ledPin3 = 13;

void setup() {
 pinMode(ledPin1, OUTPUT);
 pinMode(ledPin2, OUTPUT);
 pinMode(ledPin3, OUTPUT);
 Serial.begin(9600);
}

void loop() {
 for (int j = 0; j < 3; j++) {
 delay(2000);
 }
}
```

```

for (int i = 0; i < 5; i++) {
 digitalWrite(ledPin1, HIGH);
 digitalWrite(ledPin2, HIGH);
 digitalWrite(ledPin3, HIGH);
 delay(100);
 digitalWrite(ledPin1, LOW);
 digitalWrite(ledPin2, LOW);
 digitalWrite(ledPin3, LOW);
 delay(100);
 Serial.print("J= ");
 Serial.print(j);
 Serial.print(" i= ");
 Serial.println(i);
}
}
}

```

## 6.6 - Increment, Decrement using for loop

### Code : Increment and Decrement in C

```

#include<stdio.h>
int i,n; //Global Variable
int main()
{
 printf("Enter an integer number = ");
 scanf("%d",&n);
 printf("Increment\n");
 for(i=0;i<=n;i++){
 printf("i = %d\n",i);
 }
 printf("Decrement\n");
 for(i=n;i>=0;i--){
 printf("i = %d\n",i);
 }
 return 0;
}

```

## College Level:

### 6.7 - Break & Continue Statement

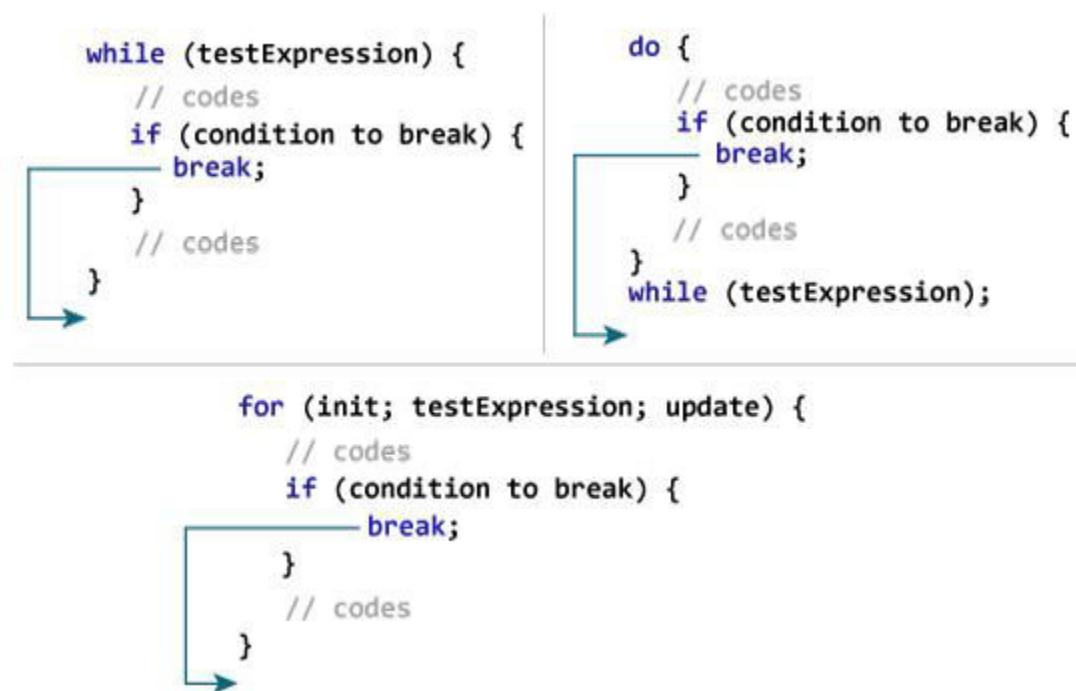
#### 6.7.1 - Break Statement

The break statement terminates the loop (for, while and do...while loop) immediately when it is encountered. Its syntax is:

```
break;
```

The break statement is almost always used with if...else statement inside the loop.

#### How does break statement works?



#### Example 1: break statement

```
// Program to calculate the sum of maximum of 10 numbers
// If negative number is entered, loop terminates and sum is displayed

#include <stdio.h>
int main()
{
 int i;
 double number, sum = 0.0;

 for(i=1; i <= 10; ++i)
 {
 if(number < 0)
 break;
 // codes
 sum += number;
 }
 // codes
}
```

```

printf("Enter a n%d: ",i);
scanf("%lf",&number);

// If user enters negative number, loop is terminated
if(number < 0.0)
{
 break;
}

sum += number; // sum = sum + number;
}

printf("Sum = %.2lf",sum);

return 0;
}

```

**Output :**

Enter a n1: 2.4

Enter a n2: 4.5

Enter a n3: 3.4

Enter a n4: -3

Sum = 10.30

This program calculates the sum of maximum of 10 numbers. Why maximum of 10 numbers?

It's because if the user enters negative number, the break statement is executed which terminates the for loop, and sum is displayed.

In C, break is also used with switch statement.

**6.7.2 - Continue Statement**

The continue statement skips statements after it inside the loop. Its syntax is:

```
continue;
```

The continue statement is almost always used with if...else statement.

**How does continue statement works?**

```

→ while (testExpression) {
 // codes
 if (testExpression) {
 continue;
 }
 // codes
}

do {
 // codes
 if (testExpression) {
 continue;
 }
 // codes
}
→ while (testExpression);

```

```

→ for (init; testExpression; update) {
 // codes
 if (testExpression) {
 continue;
 }
 // codes
}

```

### Example 1: Continue Statement C

This code will print 1 to 10 except 7 . 7 is skipped because of continue.

```

#include <stdio.h>
int main()
{
 int i;
 for(i=1;i<=10;i++){
 if(i==7){
 continue;
 }
 printf("%d\n",i);
 }
 return 0;
}

```

### Example 2: Continue Statement in Arduino

```

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

void loop() {
 for(int i=1;i<=7;i++){
 if(i==4){ // 4 will be skipped
 continue;
 }
 }
}

```

```

 Serial.println(i);
 delay(1000);
 }
}
```

## 6.8 - Pyramid print with star

[reference - <https://www.programiz.com/c-programming/examples/pyramid-pattern>]

### **Example 1: Program to print half pyramid using \***

```

*
* *
* * *
* * * *
* * * * *
```

### **Code : Example 1**

```

#include <stdio.h>
int main()
{
 int i, j, rows;

 printf("Enter number of rows: ");
 scanf("%d",&rows);

 for(i=1; i<=rows; ++i)
 {
 for(j=1; j<=i; ++j)
 {
 printf("* ");
 }
 printf("\n");
 }
 return 0;
}
```

### **Example 2: Program to print half pyramid a using numbers**

```

1
1 2
1 2 3
```

```
1 2 3 4
1 2 3 4 5
```

**Code : Example 2**

```
#include <stdio.h>
int main()
{
 int i, j, rows;

 printf("Enter number of rows: ");
 scanf("%d",&rows);

 for(i=1; i<=rows; ++i)
 {
 for(j=1; j<=i; ++j)
 {
 printf("%d ",j);
 }
 printf("\n");
 }
 return 0;
}
```

Follow this link(<https://www.programiz.com/c-programming/examples/pyramid-pattern>) for further example .

## Class 7:

- 7.1 - Array in C
- 7.2 - Array in Arduino
- 7.3 - C Operators
- 7.4 - Analogwrite
- 7.5 - LED Fade
- 7.6 - Servo Motor Control
- 7.7 - Battery - Voltage , Discharge Rate, Series, Parallel, Internal Resistance

**College Level:**

- 7.8 - Basic Interrupt
- 7.9 - VS Code for Arduino
- 7.10 - AnalogWrite Resolution

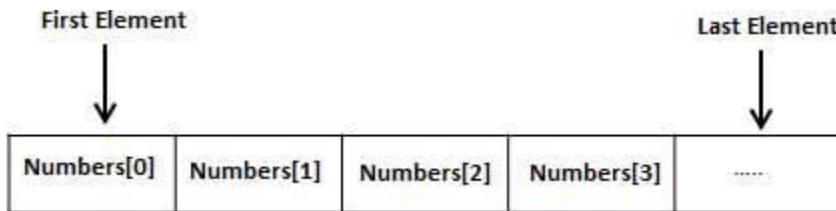
## 7.1 - Array

Arrays a kind of data structure that can store a fixed-size sequential collection of elements of the same type. An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type.

Instead of declaring individual variables, such as number0, number1, ..., and number99, you declare one array variable such as numbers and use numbers[0], numbers[1], and ..., numbers[99] to represent individual variables. A specific element in an array is accessed by an index.

All arrays consist of contiguous memory locations. The lowest address corresponds to the first element and the highest address to the last element.

[reference- [https://www.tutorialspoint.com/cprogramming/c\\_arrays.htm](https://www.tutorialspoint.com/cprogramming/c_arrays.htm)]



### 7.1.1 - Declaring Arrays

To declare an array in C, a programmer specifies the type of the elements and the number of elements required by an array as follows –

```
type arrayName [arraySize];
```

This is called a *single-dimensional* array. The arraySize must be an integer constant greater than zero and type can be any valid C data type. For example, to declare a 10-element array called balance of type double, use this statement –

```
double balance[10];
```

Here *balance* is a variable array which is sufficient to hold up to 10 double numbers.

### 7.1.2 - Initializing Arrays

You can initialize an array in C either one by one or using a single statement as follows –

```
double balance[5] = {1000.0, 2.0, 3.4, 7.0, 50.0};
```

The number of values between braces { } cannot be larger than the number of elements that we declare for the array between square brackets [ ].

If you omit the size of the array, an array just big enough to hold the initialization is created.

Therefore, if you write –

```
double balance[] = {1000.0, 2.0, 3.4, 7.0, 50.0};
```

You will create exactly the same array as you did in the previous example. Following is an example to assign a single element of the array –

```
balance[4] = 50.0;
```

The above statement assigns the 5<sup>th</sup> element in the array with a value of 50.0. All arrays have 0 as the index of their first element which is also called the base index and the last index of an array will be total size of the array minus 1. Shown below is the pictorial representation of the array we discussed above –

|         | 0      | 1   | 2   | 3   | 4    |
|---------|--------|-----|-----|-----|------|
| balance | 1000.0 | 2.0 | 3.4 | 7.0 | 50.0 |

### 7.1.3 - Accessing Array Elements

An element is accessed by indexing the array name. This is done by placing the index of the element within square brackets after the name of the array. For example –

```
double salary = balance[9];
```

The above statement will take the 10<sup>th</sup> element from the array and assign the value to salary variable.

**Example 1:**

```
#include <stdio.h>
int pins[5] = {2,7,9,78};
char names[5] = {'n','a','b','i','l'};
int i;
int main () {
 printf("%d\n",pins[3]);
for(i=0; i < 5; i++){
 printf("%c",names[i]);
}
return 0;
}
```

**Example 2 :** The following example Shows how to use all the three above mentioned concepts . declaration, assignment, and accessing arrays.

```
#include <stdio.h>

int main () {

 int n[10]; /* n is an array of 10 integers */
 int i,j;

 /* initialize elements of array n to 0 */
 for (i = 0; i < 10; i++) {
 n[i] = i + 100; /* set element at location i to i + 100 */
 }

 /* output each array element's value */
 for (j = 0; j < 10; j++) {
 printf("Element[%d] = %d\n", j, n[j]);
 }
}
```

```

 }

 return 0;
}

```

When the above code is compiled and executed, it produces the following result –

```

Element[0] = 100
Element[1] = 101
Element[2] = 102
Element[3] = 103
Element[4] = 104
Element[5] = 105
Element[6] = 106
Element[7] = 107
Element[8] = 108
Element[9] = 109

```

#### 7.1.4 - Arrays in Detail

Arrays are important to C and should need a lot more attention. The following important concepts related to array should be clear to a C programmer –

| Sr.No. | Concept & Description                                                                                                                                                  |
|--------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 1      | <p><b><u>Multi-dimensional arrays</u></b></p> <p>C supports multidimensional arrays. The simplest form of the multidimensional array is the two-dimensional array.</p> |
| 2      | <p><b><u>Passing arrays to functions</u></b></p> <p>You can pass to the function a pointer to an array by specifying the array's name without an index.</p>            |

|   |                                                                                                                     |
|---|---------------------------------------------------------------------------------------------------------------------|
| 3 | <b><u>Return array from a function</u></b>                                                                          |
|   | C allows a function to return an array.                                                                             |
| 4 | <b><u>Pointer to an array</u></b>                                                                                   |
|   | You can generate a pointer to the first element of an array by simply specifying the array name, without any index. |

## 7.2 - Array in Arduino

[reference - <https://www.arduino.cc/reference/en/language/variables/data-types/array/>]

[reference - <https://www.arduino.cc/en/Tutorial/Arrays>]

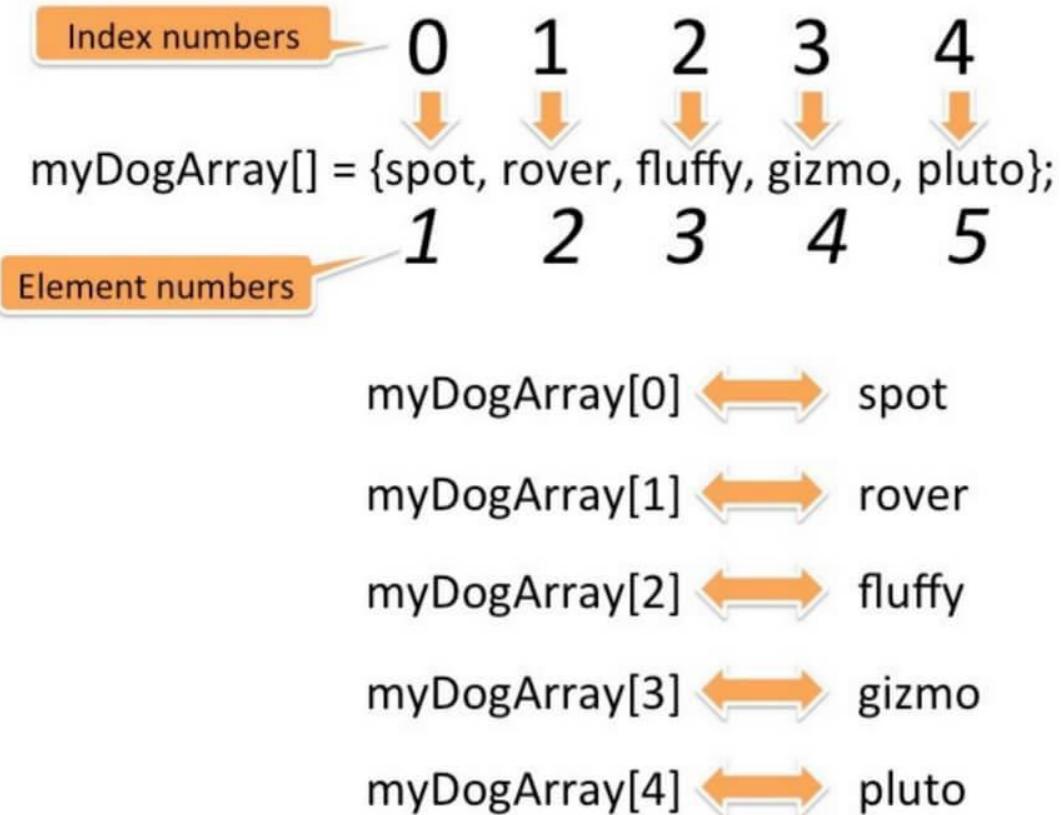
This variation on the For Loop Iteration example shows how to use an array. An array is a variable with multiple parts. If you think of a variable as a cup that holds values, you might think of an array as an ice cube tray. It's like a series of linked cups, all of which can hold the same maximum value.

The For Loop Iteration example shows you how to light up a series of LEDs attached to pins 2 through 7 of the Arduino or Genuino board, with certain limitations (the pins have to be numbered contiguously, and the LEDs have to be turned on in sequence).

This example shows you how you can turn on a sequence of pins whose numbers are neither continuous nor necessarily sequential. To do this is, you can put the pin numbers in an array and then use for loops to iterate over the array.

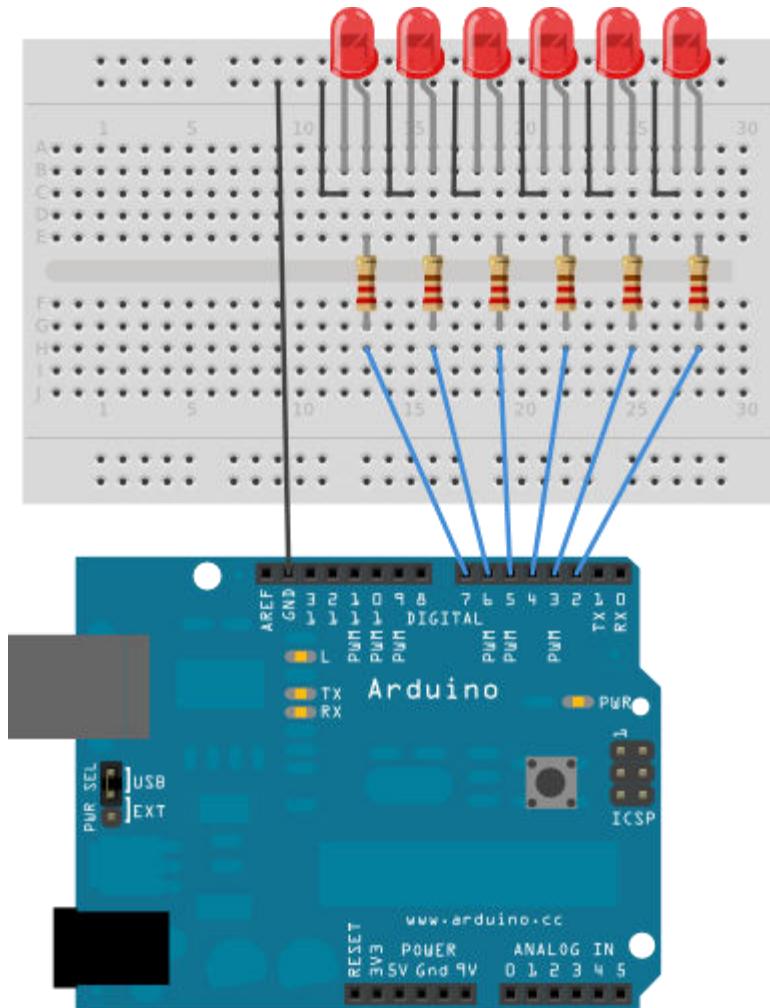
This example makes use of 6 LEDs connected to the pins 2 - 7 on the board using 220 ohm resistors, just like in the For Loop. However, here the order of the LEDs is determined by their order in the array, not by their physical order.

This technique of putting the pins in an array is very handy. You don't have to have the pins sequential to one another, or even in the same order. You can rearrange them in any order you want.

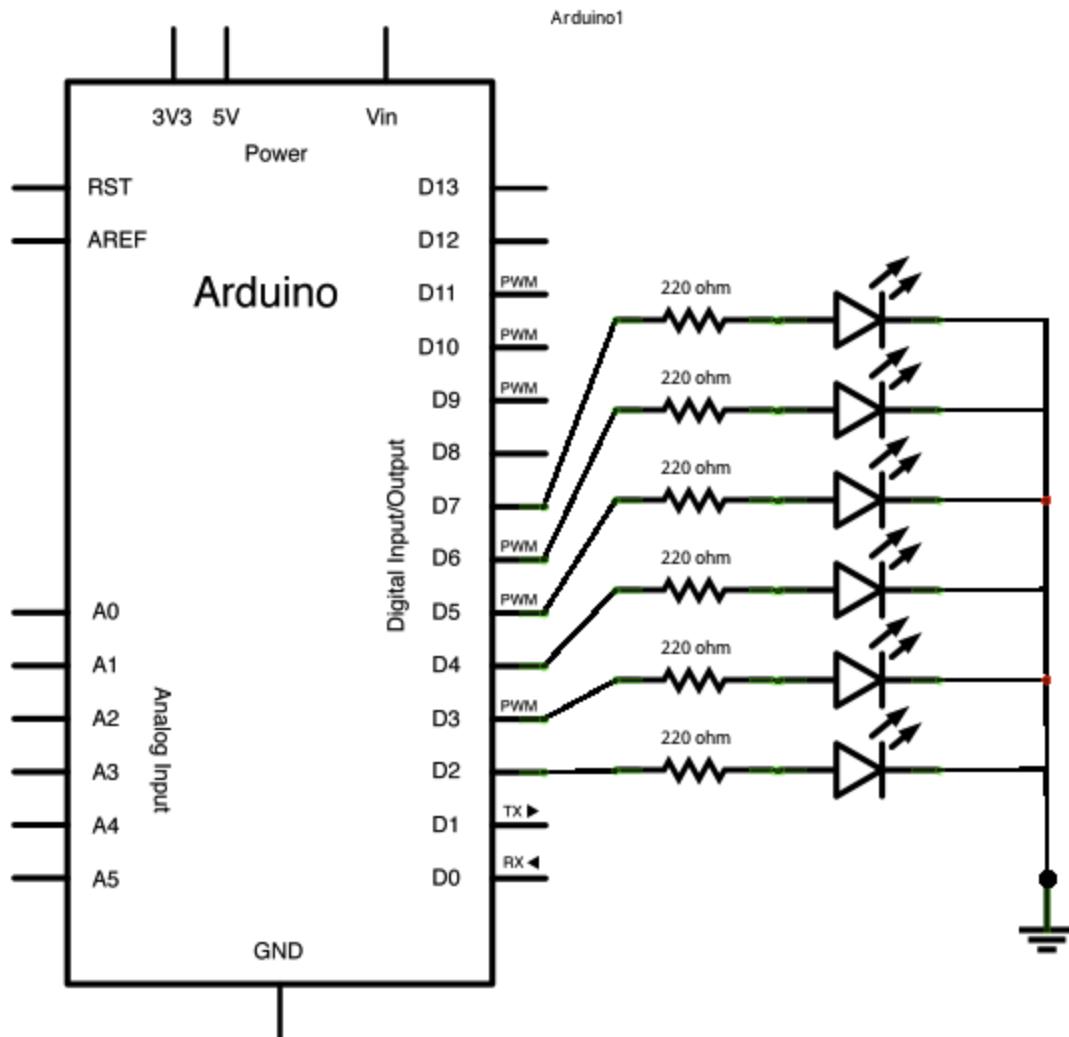


#### Circuit :

Connect six LEDs, with 220 ohm resistors in series, to digital pins 2-7 on your board.



**Schematic :**



### Code :

[reference - <https://www.arduino.cc/en/Tutorial/Arrays>]

[reference - <https://programmingelectronics.com/tutorial-13-how-to-use-arrays-with-arduino/>

```
/*
 Arrays
 Demonstrates the use of an array to hold pin numbers in order to iterate over
 the pins in a sequence. Lights multiple LEDs in sequence, then in reverse.
```

Unlike the For Loop tutorial, where the pins have to be contiguous, here the pins can be in any random order.

```
The circuit:
```

```
- LEDs from pins 2 through 7 to ground
```

```
created 2006
```

```
by David A. Mellis
```

```
modified 30 Aug 2011
```

```
by Tom Igoe
```

```
This example code is in the public domain.
```

```
http://www.arduino.cc/en/Tutorial/Array
```

```
*/
```

```
int timer = 100;
// The higher the number, the slower the timing.
int ledPins[] = {
 2, 7, 4, 6, 5, 3
}; // an array of pin numbers to which LEDs are attached
int pinCount = 6;
// the number of pins (i.e. the length of the array)
```

```
void setup() {
 // the array elements are numbered from 0 to (pinCount - 1).
 // use a for loop to initialize each pin as an output:
 for (int thisPin = 0; thisPin < pinCount; thisPin++) {
 pinMode(ledPins[thisPin], OUTPUT);
 }
}
```

```
void loop() {
 // loop from the lowest pin to the highest:
 for (int thisPin = 0; thisPin < pinCount; thisPin++) {
 // turn the pin on:
 digitalWrite(ledPins[thisPin], HIGH);
 delay(timer);
 // turn the pin off:
 digitalWrite(ledPins[thisPin], LOW);
 }
}
```

```
// loop from the highest pin to the lowest:
```

```
for (int thisPin = pinCount - 1; thisPin >= 0; thisPin--) {
 // turn the pin on:
 digitalWrite(ledPins[thisPin], HIGH);
```

```

 delay(timer);
 // turn the pin off:
 digitalWrite(ledPins[thisPin], LOW);
}
}

```

## 7.3 - C Operators

[reference - [https://www.tutorialspoint.com/cprogramming/c\\_operators.htm](https://www.tutorialspoint.com/cprogramming/c_operators.htm)]

An operator is a symbol that tells the compiler to perform specific mathematical or logical functions. C language is rich in built-in operators and provides the following types of operators

-

- Arithmetic Operators
- Relational Operators
- Logical Operators
- Bitwise Operators
- Assignment Operators
- Misc Operators

We will, in this chapter, look into the way each operator works.

### 7.3.1 - Arithmetic Operators

The following table shows all the arithmetic operators supported by the C language. Assume variable A holds 10 and variable B holds 20 then –

**Example :**

```

#include <stdio.h>
main() {
 int a = 21;
 int b = 10;
}

```

```

int c ;
c = a + b;
printf("Line 1 - Value of c is %d\n", c);
c = a - b;
printf("Line 2 - Value of c is %d\n", c);
c = a * b;
printf("Line 3 - Value of c is %d\n", c);
c = a / b;
printf("Line 4 - Value of c is %d\n", c);
c = a % b;
printf("Line 5 - Value of c is %d\n", c);
c = a++;
printf("Line 6 - Value of c is %d\n", c);
c = a--;
printf("Line 7 - Value of c is %d\n", c);
}

```

When you compile and execute the above program, it produces the following result –

Line 1 - Value of c is 31

Line 2 - Value of c is 11

Line 3 - Value of c is 210

Line 4 - Value of c is 2

Line 5 - Value of c is 1

Line 6 - Value of c is 21

Line 7 - Value of c is 22

| <b>Operator</b> | <b>Description</b>                       | <b>Example</b> |
|-----------------|------------------------------------------|----------------|
| +               | Adds two operands.                       | $A + B = 30$   |
| -               | Subtracts second operand from the first. | $A - B = -10$  |

|    |                                                              |               |
|----|--------------------------------------------------------------|---------------|
| *  | Multiples both operands.                                     | $A * B = 200$ |
| /  | Divides numerator by de-numerator.                           | $B / A = 2$   |
| %  | Modulus Operator and remainder of after an integer division. | $B \% A = 0$  |
| ++ | Increment operator increases the integer value by one.       | $A++ = 11$    |
| -- | Decrement operator decreases the integer value by one.       | $A-- = 9$     |

### 7.3.2 - Relational Operators

The following table shows all the relational operators supported by C. Assume variable A holds 10 and variable B holds 20 then –

#### Example :

```
#include <stdio.h>
main() {
 int a = 21;
 int b = 10;
 int c ;
 if(a == b) {
 printf("Line 1 - a is equal to b\n");
 } else {
 printf("Line 1 - a is not equal to b\n");
 }
 if (a < b) {
 printf("Line 2 - a is less than b\n");
 } else {
 printf("Line 2 - a is not less than b\n");
 }

 if (a > b) {
 printf("Line 3 - a is greater than b\n");
 }
}
```

```
 } else {
 printf("Line 3 - a is not greater than b\n");
}
/* Lets change value of a and b */
a = 5;
b = 20;
if (a <= b) {
 printf("Line 4 - a is either less than or equal to b\n");
}
if (b >= a) {
 printf("Line 5 - b is either greater than or equal to b\n");
}
}
```

When you compile and execute the above program, it produces the following result –

Line 1 - a is not equal to b

Line 2 - a is not less than b

Line 3 - a is greater than b

Line 4 - a is either less than or equal to b

Line 5 - b is either greater than or equal to b

| Operator           | Description                                                                                                                          | Example                 |
|--------------------|--------------------------------------------------------------------------------------------------------------------------------------|-------------------------|
| <code>==</code>    | Checks if the values of two operands are equal or not. If yes, then the condition becomes true.                                      | $(A == B)$ is not true. |
| <code>!=</code>    | Checks if the values of two operands are equal or not. If the values are not equal, then the condition becomes true.                 | $(A != B)$ is true.     |
| <code>&gt;</code>  | Checks if the value of left operand is greater than the value of right operand. If yes, then the condition becomes true.             | $(A > B)$ is not true.  |
| <code>&lt;</code>  | Checks if the value of left operand is less than the value of right operand. If yes, then the condition becomes true.                | $(A < B)$ is true.      |
| <code>&gt;=</code> | Checks if the value of left operand is greater than or equal to the value of right operand. If yes, then the condition becomes true. | $(A >= B)$ is not true. |
| <code>&lt;=</code> | Checks if the value of left operand is less than or equal to the value of right operand. If yes, then the condition becomes true.    | $(A <= B)$ is true.     |

### 7.3.3 - Logical Operators

Following table shows all the logical operators supported by C language. Assume variable A

holds 1 and variable B holds 0, then –

```
#include <stdio.h>

main() {
 int a = 5;
 int b = 20;
 int c ;
 if (a && b) {
 printf("Line 1 - Condition is true\n");
 }
 if (a || b) {
 printf("Line 2 - Condition is true\n");
 }
 /* lets change the value of a and b */
 a = 0;
 b = 10;
 if (a && b) {
 printf("Line 3 - Condition is true\n");
 } else {
 printf("Line 3 - Condition is not true\n");
 }
 if (!(a && b)) {
 printf("Line 4 - Condition is true\n");
 }
}
```

When you compile and execute the above program, it produces the following result –

Line 1 - Condition is true

Line 2 - Condition is true

Line 3 - Condition is not true

Line 4 - Condition is true

| Operator | Description                                                                                                                                                | Example            |
|----------|------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------|
| &&       | Called Logical AND operator. If both the operands are non-zero, then the condition becomes true.                                                           | (A && B) is false. |
|          | Called Logical OR Operator. If any of the two operands is non-zero, then the condition becomes true.                                                       | (A    B) is true.  |
| !        | Called Logical NOT Operator. It is used to reverse the logical state of its operand. If a condition is true, then Logical NOT operator will make it false. | !(A && B) is true. |

#### 7.3.4 - Bitwise Operators

Bitwise operator works on bits and perform bit-by-bit operation. The truth tables for &, |, and ^ is as follows –

| p | q | p & q | p   q | p ^ q |
|---|---|-------|-------|-------|
| 0 | 0 | 0     | 0     | 0     |
| 0 | 1 | 0     | 1     | 1     |
| 1 | 1 | 1     | 1     | 0     |

|   |   |   |   |   |
|---|---|---|---|---|
| 1 | 0 | 0 | 1 | 1 |
|---|---|---|---|---|

Assume A = 60 and B = 13 in binary format, they will be as follows –

A = 0011 1100

B = 0000 1101

-----

A&B = 0000 1100

A|B = 0011 1101

A^B = 0011 0001

~A = 1100 0011

The following table lists the bitwise operators supported by C. Assume variable 'A' holds 60

and variable 'B' holds 13, then –

### Example :

```
#include <stdio.h>
main() {
 unsigned int a = 60; /* 60 = 0011 1100 */
 unsigned int b = 13; /* 13 = 0000 1101 */
 int c = 0;
 c = a & b; /* 12 = 0000 1100 */
 printf("Line 1 - Value of c is %d\n", c);
 c = a | b; /* 61 = 0011 1101 */
 printf("Line 2 - Value of c is %d\n", c);
 c = a ^ b; /* 49 = 0011 0001 */
 printf("Line 3 - Value of c is %d\n", c);
 c = ~a; /* -61 = 1100 0011 */
 printf("Line 4 - Value of c is %d\n", c);
 c = a << 2; /* 240 = 1111 0000 */
 printf("Line 5 - Value of c is %d\n", c);
 c = a >> 2; /* 15 = 0000 1111 */
 printf("Line 6 - Value of c is %d\n", c);
}
```

When you compile and execute the above program, it produces the following result –

Line 1 - Value of c is 12

Line 2 - Value of c is 61

Line 3 - Value of c is 49

Line 4 - Value of c is -61

Line 5 - Value of c is 240

Line 6 - Value of c is 15

| <b>Operator</b> | <b>Description</b>                                                                                                       | <b>Example</b>                            |
|-----------------|--------------------------------------------------------------------------------------------------------------------------|-------------------------------------------|
| &               | Binary AND Operator copies a bit to the result if it exists in both operands.                                            | $(A \& B) = 12$ ,<br>i.e., 0000<br>1100   |
|                 | Binary OR Operator copies a bit if it exists in either operand.                                                          | $(A   B) = 61$ ,<br>i.e., 0011<br>1101    |
| ^               | Binary XOR Operator copies the bit if it is set in one operand but not both.                                             | $(A ^ B) = 49$ ,<br>i.e., 0011<br>0001    |
| ~               | Binary One's Complement Operator is unary and has the effect of 'flipping' bits.                                         | $(\sim A) = \sim(60)$ ,<br>i.e., -0111101 |
| <<              | Binary Left Shift Operator. The left operand's value is moved left by the number of bits specified by the right operand. | $A << 2 = 240$<br>i.e., 1111<br>0000      |

|    |                                                                                                                            |                                   |
|----|----------------------------------------------------------------------------------------------------------------------------|-----------------------------------|
| >> | Binary Right Shift Operator. The left operand's value is moved right by the number of bits specified by the right operand. | A >> 2 = 15<br>i.e., 0000<br>1111 |
|----|----------------------------------------------------------------------------------------------------------------------------|-----------------------------------|

### 7.3.5 -Assignment Operators

The following table lists the assignment operators supported by the C language –

| Operator | Description                                                                                                                         | Example                                       |
|----------|-------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------|
| =        | Simple assignment operator. Assigns values from right side operands to left side operand                                            | C = A + B will assign the value of A + B to C |
| +=       | Add AND assignment operator. It adds the right operand to the left operand and assigns the result to the left operand.              | C += A is equivalent to C = C + A             |
| -=       | Subtract AND assignment operator. It subtracts the right operand from the left operand and assigns the result to the left operand.  | C -= A is equivalent to C = C - A             |
| *=       | Multiply AND assignment operator. It multiplies the right operand with the left operand and assigns the result to the left operand. | C *= A is equivalent to C = C * A             |

|     |                                                                                                                                |                                   |
|-----|--------------------------------------------------------------------------------------------------------------------------------|-----------------------------------|
| /=  | Divide AND assignment operator. It divides the left operand with the right operand and assigns the result to the left operand. | C /= A is equivalent to C = C / A |
| %=  | Modulus AND assignment operator. It takes modulus using two operands and assigns the result to the left operand.               | C %= A is equivalent to C = C % A |
| <=> | Left shift AND assignment operator.                                                                                            | C <=> 2 is same as C = C << 2     |
| >=> | Right shift AND assignment operator.                                                                                           | C >=> 2 is same as C = C >> 2     |
| &=  | Bitwise AND assignment operator.                                                                                               | C &= 2 is same as C = C & 2       |
| ^=  | Bitwise exclusive OR and assignment operator.                                                                                  | C ^= 2 is same as C = C ^ 2       |
| =   | Bitwise inclusive OR and assignment operator.                                                                                  | C  = 2 is same as C = C   2       |

Misc Operators ↪ sizeof & ternary

Besides the operators discussed above, there are a few other important operators including sizeof and ?: supported by the C Language.

| Operator | Description                        | Example                                                 |
|----------|------------------------------------|---------------------------------------------------------|
| sizeof() | Returns the size of a variable.    | sizeof(a), where a is integer, will return 4.           |
| &        | Returns the address of a variable. | &a; returns the actual address of the variable.         |
| *        | Pointer to a variable.             | *a;                                                     |
| ? :      | Conditional Expression.            | If Condition is true ? then value X : otherwise value Y |

### 7.3.6 - Operators Precedence in C

Operator precedence determines the grouping of terms in an expression and decides how an expression is evaluated. Certain operators have higher precedence than others; for example, the multiplication operator has a higher precedence than the addition operator.

For example,  $x = 7 + 3 * 2$ ; here, x is assigned 13, not 20 because operator \* has a higher precedence than +, so it first gets multiplied with  $3*2$  and then adds into 7.

Here, operators with the highest precedence appear at the top of the table, those with the lowest appear at the bottom. Within an expression, higher precedence operators will be evaluated first.

#### Example :

```
#include <stdio.h>
main() {
```

```

int a = 20;
int b = 10;
int c = 15;
int d = 5;
int e;
 e = (a + b) * c / d; // (30 * 15) / 5
printf("Value of (a + b) * c / d is : %d\n", e);
e = ((a + b) * c) / d; // (30 * 15) / 5
printf("Value of ((a + b) * c) / d is : %d\n" , e);
e = (a + b) * (c / d); // (30) * (15/5)
printf("Value of (a + b) * (c / d) is : %d\n" , e);
e = a + (b * c) / d; // 20 + (150/5)
printf("Value of a + (b * c) / d is : %d\n" , e);
return 0;
}

```

When you compile and execute the above program, it produces the following result –

```

Value of (a + b) * c / d is : 90
Value of ((a + b) * c) / d is : 90
Value of (a + b) * (c / d) is : 90
Value of a + (b * c) / d is : 50

```

| Category       | Operator                       | Associativity |
|----------------|--------------------------------|---------------|
| Postfix        | () [] -> . ++ --               | Left to right |
| Unary          | + - ! ~ ++ -- (type)* & sizeof | Right to left |
| Multiplicative | * / %                          | Left to right |
| Additive       | + -                            | Left to right |
| Shift          | << >>                          | Left to right |

|             |                                   |               |
|-------------|-----------------------------------|---------------|
| Relational  | < <= > >=                         | Left to right |
| Equality    | == !=                             | Left to right |
| Bitwise AND | &                                 | Left to right |
| Bitwise XOR | ^                                 | Left to right |
| Bitwise OR  |                                   | Left to right |
| Logical AND | &&                                | Left to right |
| Logical OR  |                                   | Left to right |
| Conditional | ?:                                | Right to left |
| Assignment  | = += -= *= /= %= >>= <<= &= ^=  = | Right to left |
| Comma       | ,                                 | Left to right |

## 7.4 - Analogwrite/PWM

`analogWrite()`

[reference - <https://www.arduino.cc/reference/en/language/functions/analog-io/analogwrite/> ]

#### 7.4.1 - Description

Writes an analog value (PWM wave) to a pin. Can be used to light a LED at varying brightnesses or drive a motor at various speeds. After a call to `analogWrite()`, the pin will generate a steady square wave of the specified duty cycle until the next call to `analogWrite()` (or a call to `digitalRead()` or `digitalWrite()`) on the same pin. The frequency of the PWM signal on most pins is approximately 490 Hz. On the Uno and similar boards, pins 5 and 6 have a frequency of approximately 980 Hz.

On most Arduino boards (those with the ATmega168 or ATmega328P), this function works on pins 3, 5, 6, 9, 10, and 11. On the Arduino Mega, it works on pins 2 - 13 and 44 - 46. Older Arduino boards with an ATmega8 only support `analogWrite()` on pins 9, 10, and 11.

The Arduino DUE supports `analogWrite()` on pins 2 through 13, plus pins DAC0 and DAC1. Unlike the PWM pins, DAC0 and DAC1 are Digital to Analog converters, and act as true analog outputs.

You do not need to call `pinMode()` to set the pin as an output before calling `analogWrite()`. The `analogWrite` function has nothing to do with the analog pins or the `analogRead` function.

#### Syntax

```
analogWrite(pin, value)
```

#### Parameters

`pin`: the pin to write to. Allowed data types: int.

`value`: the duty cycle: between 0 (always off) and 255 (always on). Allowed data types: int

#### 7.4.2 -Example Code :

Sets the output to the LED proportional to the value read from the potentiometer.

```
int ledPin = 9; // LED connected to digital pin 9
int analogPin = 3; // potentiometer connected to analog pin 3
int val = 0; // variable to store the read value

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

void loop() {
 val = analogRead(analogPin); // read the input pin
 analogWrite(ledPin, val / 4); // analogRead values go from 0 to 1023,
```

```
analogWrite values from 0 to 255
}
```

#### 7.4.2 - Notes and Warnings

The PWM outputs generated on pins 5 and 6 will have higher-than-expected duty cycles. This is because of interactions with the millis() and delay() functions, which share the same internal timer used to generate those PWM outputs. This will be noticed mostly on low duty-cycle settings (e.g. 0 - 10) and may result in a value of 0 not fully turning off the output on pins 5 and 6.

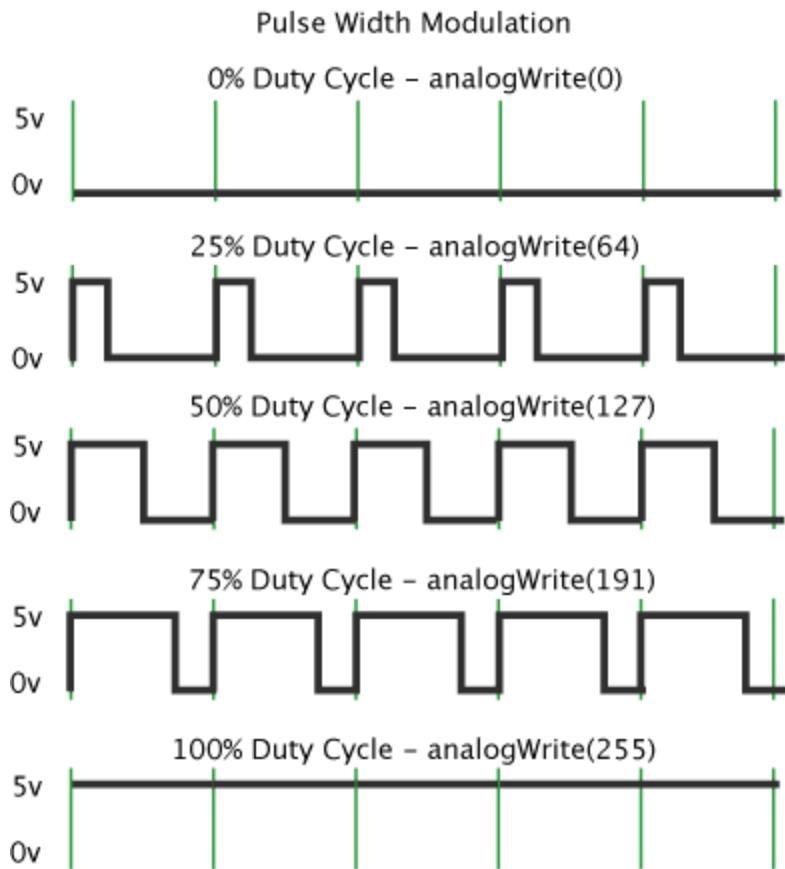
#### 7.4.3 - PWM

[reference - *Written by Timothy Hirzel*]

The Fading example demonstrates the use of analog output (PWM) to fade an LED. It is available in the File->Sketchbook->Examples->Analog menu of the Arduino software.

Pulse Width Modulation, or PWM, is a technique for getting analog results with digital means. Digital control is used to create a square wave, a signal switched between on and off. This on-off pattern can simulate voltages in between full on (5 Volts) and off (0 Volts) by changing the portion of the time the signal spends on versus the time that the signal spends off. The duration of "on time" is called the pulse width. To get varying analog values, you change, or modulate, that pulse width. If you repeat this on-off pattern fast enough with an LED for example, the result is as if the signal is a steady voltage between 0 and 5v controlling the brightness of the LED.

In the graphic below, the green lines represent a regular time period. This duration or period is the inverse of the PWM frequency. In other words, with Arduino's PWM frequency at about 500Hz, the green lines would measure 2 milliseconds each. A call to `analogWrite()` is on a scale of 0 - 255, such that `analogWrite(255)` requests a 100% duty cycle (always on), and `analogWrite(127)` is a 50% duty cycle (on half the time) for example.



Once you get this example running, grab your arduino and shake it back and forth. What you are doing here is essentially mapping time across the space. To our eyes, the movement blurs each LED blink into a line. As the LED fades in and out, those little lines will grow and shrink in length. Now you are seeing the pulse width.

## 7.5 - LED Fade

This example shows how to fade an LED on pin 9 using the `analogWrite()` function. The `analogWrite()` function uses PWM, so if you want to change the pin you're using, be sure to use another PWM capable pin. On most Arduino, the PWM pins are identified with a "~" sign, like ~3, ~5, ~6, ~9, ~10 and ~11.

```
int led = 9; // the PWM pin the LED is attached to
int brightness = 0; // how bright the LED is
int fadeAmount = 5; // how many points to fade the LED
void setup() {
pinMode(led, OUTPUT);
}
void loop() {
// set the brightness of pin 9:
```

```

analogWrite(led, brightness);

// change the brightness for next time through the loop:
brightness = brightness + fadeAmount;

// reverse the direction of the fading at the ends of the fade
if (brightness <= 0 || brightness >= 255) {
 fadeAmount = -fadeAmount;
}
// wait for 30 milliseconds to see the dimming effect
delay(30);
}

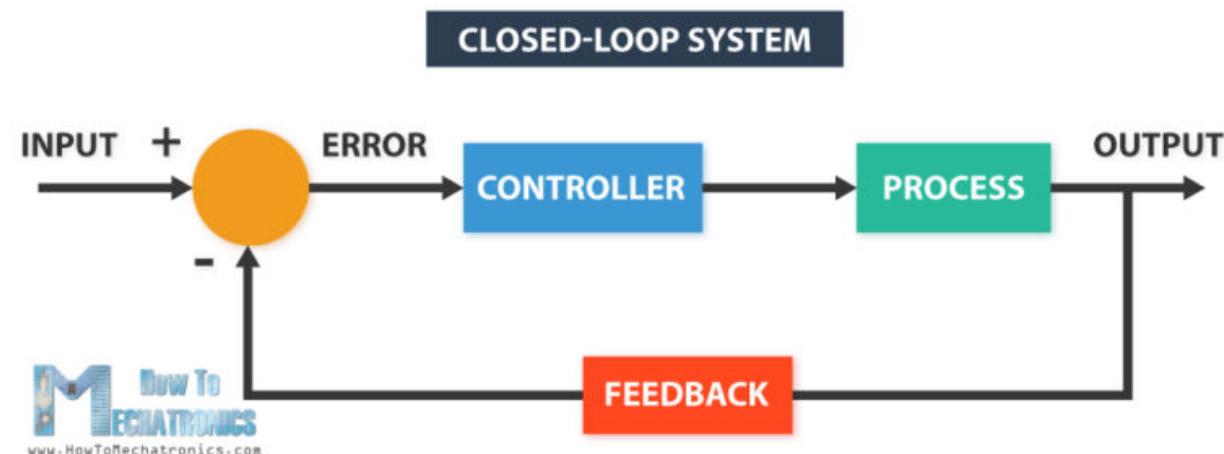
```

## 7.6 - Servo Motor Control

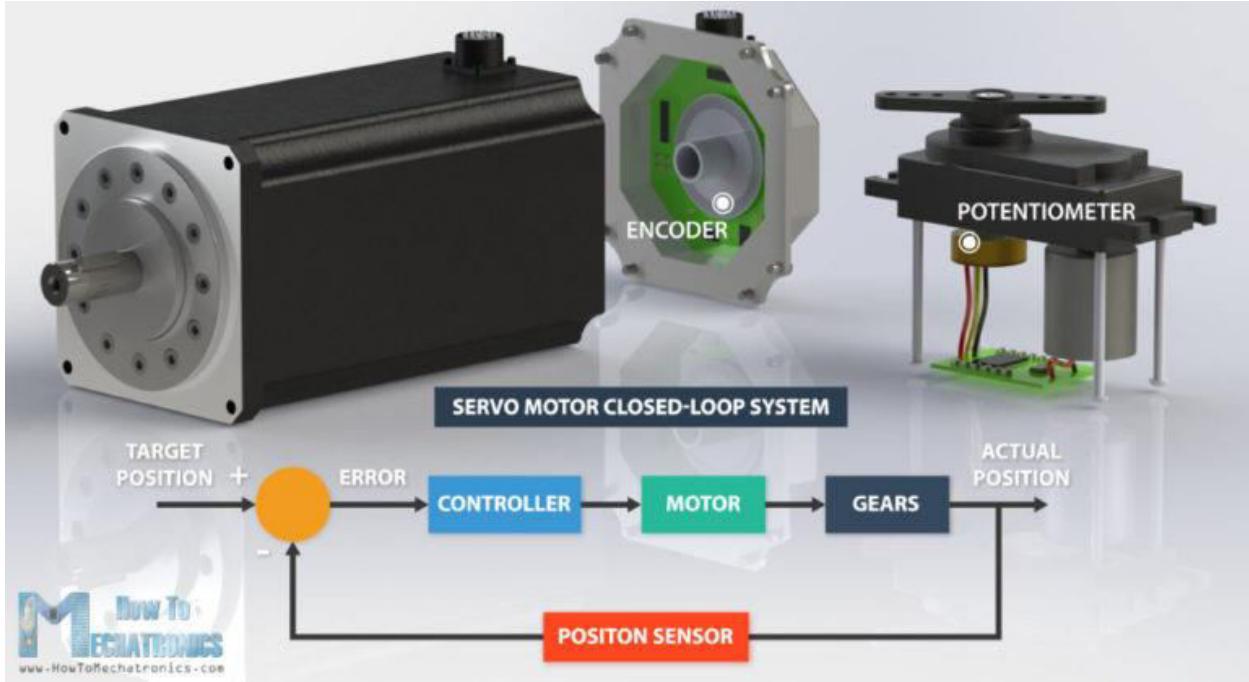
[reference - <https://howtomechatronics.com/how-it-works/how-servo-motors-work-how-to-control-servos-using-arduino/>]

### 7.6.1 - Overview

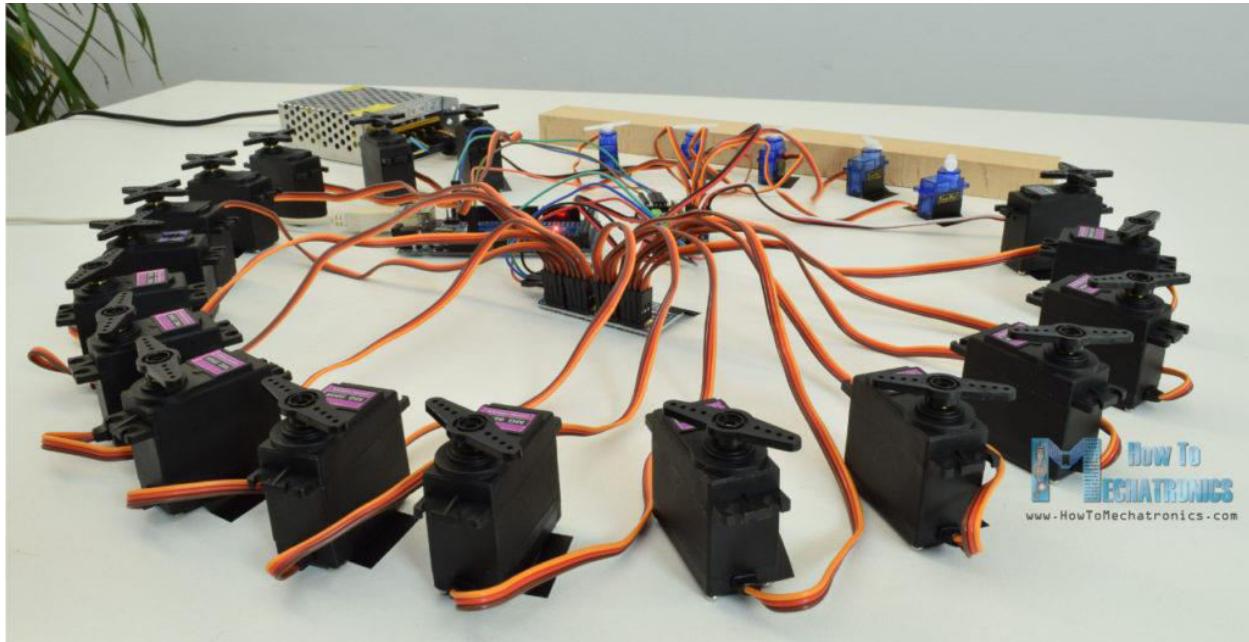
There are many types of servo motors and their main feature is the ability to precisely control the position of their shaft. A servo motor is a closed-loop system that uses position feedback to control its motion and final position.



In industrial type servo motors the position feedback sensor is usually a high precision encoder, while in the smaller RC or hobby servos the position sensor is usually a simple potentiometer. The actual position captured by these devices is fed back to the error detector where it is compared to the target position. Then according to the error the controller corrects the actual position of the motor to match with the target position.



In this tutorial we will take a detailed look at the hobby servo motors. We will explain how these servos work and how to control them using Arduino.

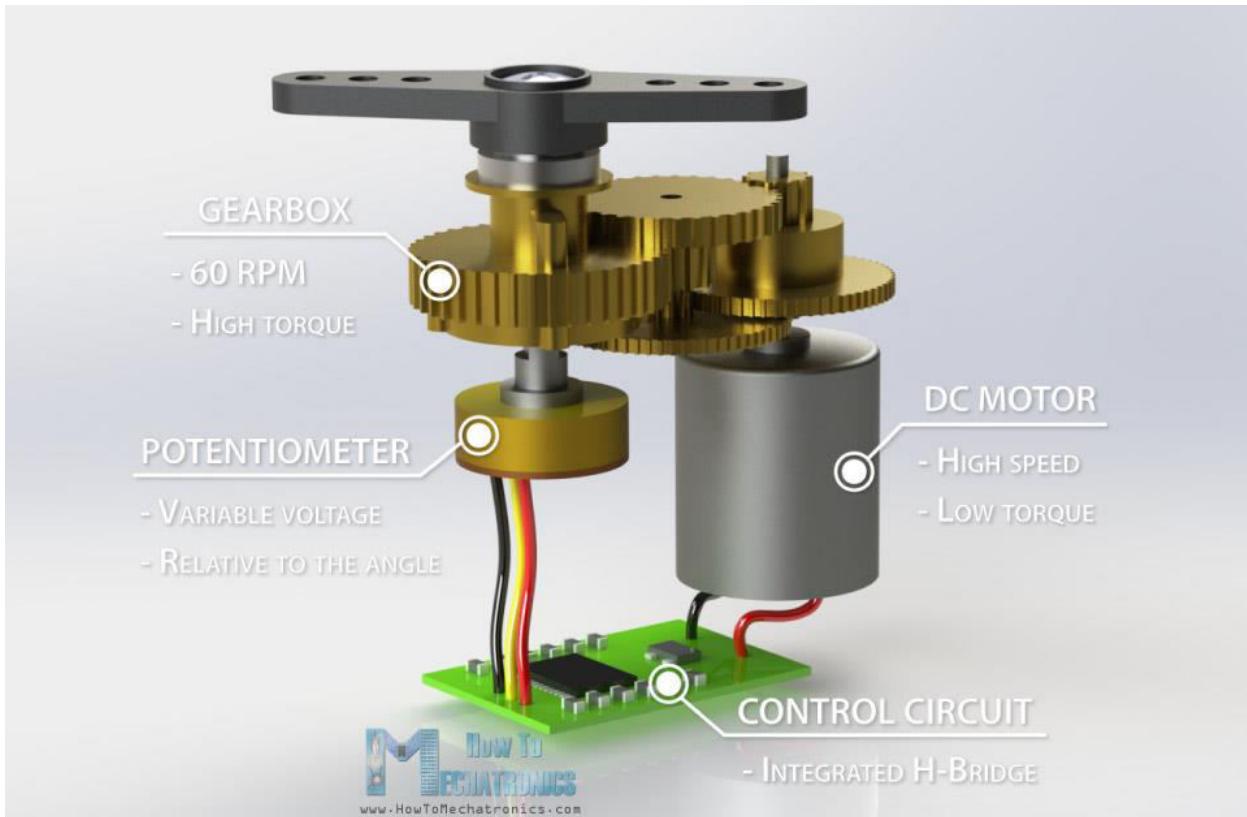


Hobby servos are small in size actuators used for controlling RC toys cars, boats, airplanes etc. They are also used by engineering students for prototyping in robotics, creating robotic arms, biologically inspired robots, humanoid robots and so on.



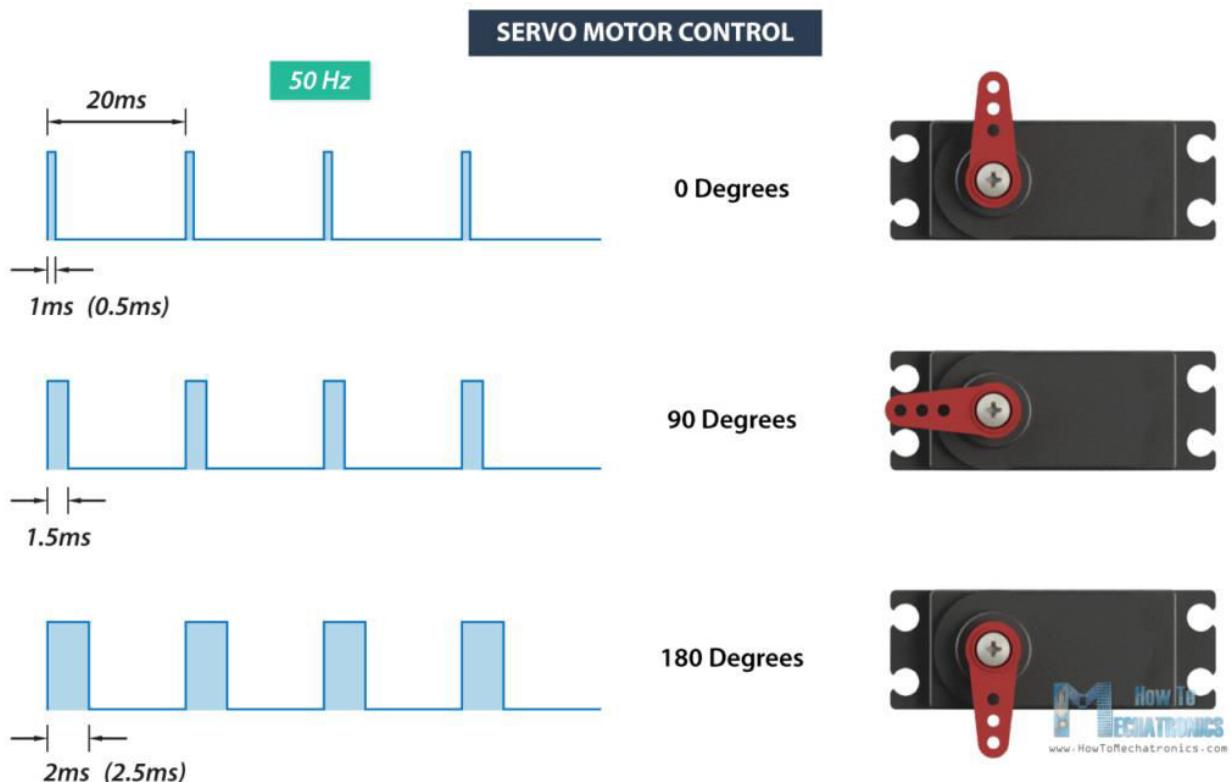
### 7.6.2 - How RC/ Hobby Servos Work

Inside a hobby servo there are four main components, a DC motor, a gearbox, a potentiometer and a control circuit. The DC motor is high speed and low torque but the gearbox reduces the speed to around 60 RPM and at the same time increases the torque.



The potentiometer is attached on the final gear or the output shaft, so as the motor rotates the potentiometer rotates as well, thus producing a voltage that is related to the absolute angle of the output shaft. In the control circuit, this potentiometer voltage is compared to the voltage coming from the signal line. If needed, the controller activates an integrated H-Bridge which enables the motor to rotate in either direction until the two signals reach a difference of zero.

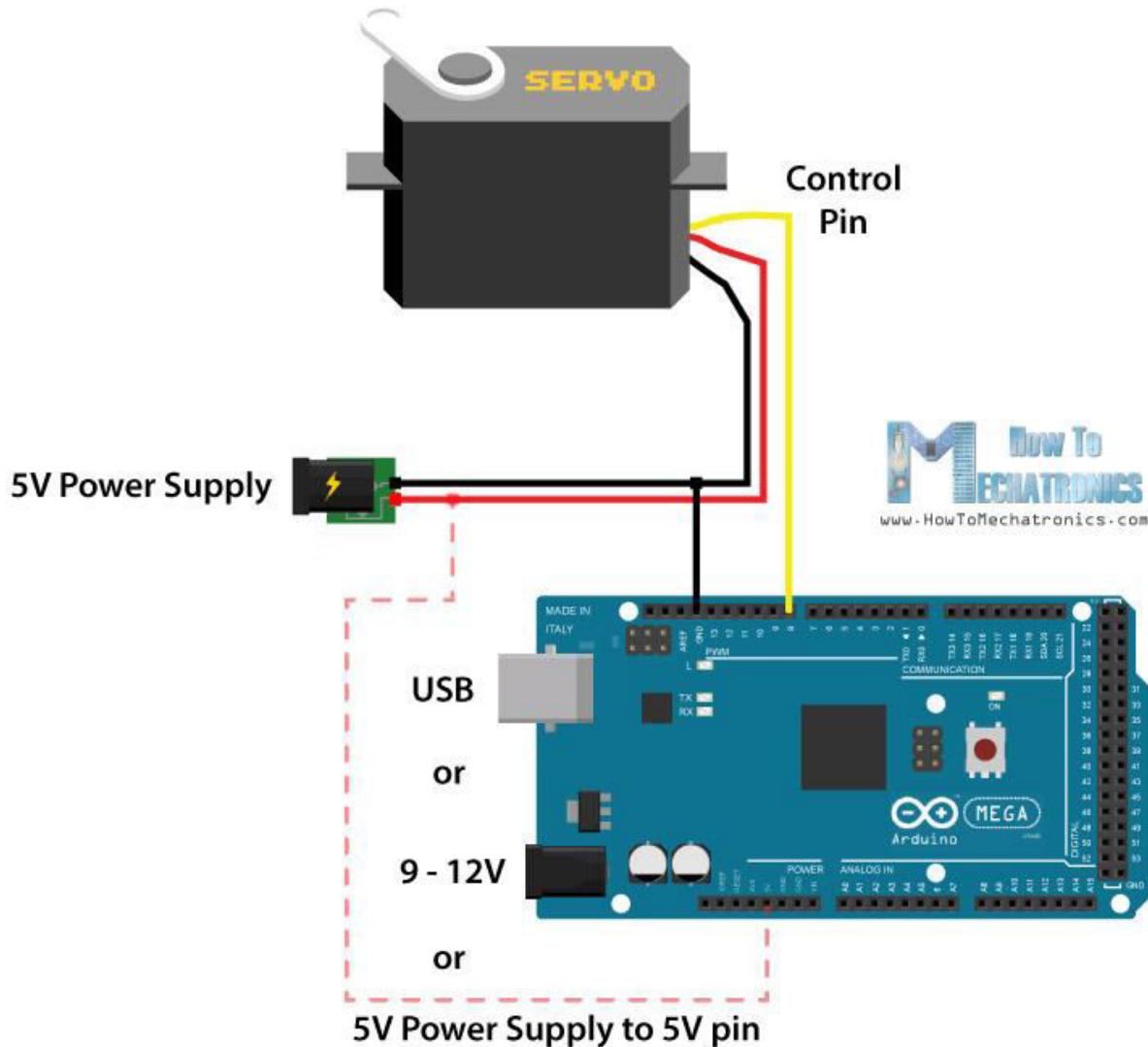
A servo motor is controlled by sending a series of pulses through the signal line. The frequency of the control signal should be 50Hz or a pulse should occur every 20ms. The width of pulse determines angular position of the servo and these type of servos can usually rotate 180 degrees (they have a physical limits of travel).



Generally pulses with 1ms duration correspond to 0 degrees position, 1.5ms duration to 90 degrees and 2ms to 180 degrees. Though the minimum and maximum duration of the pulses can sometimes vary with different brands and they can be 0.5ms for 0 degrees and 2.5ms for 180 degrees position.

### 7.6.3 - Arduino Servo Motors Control

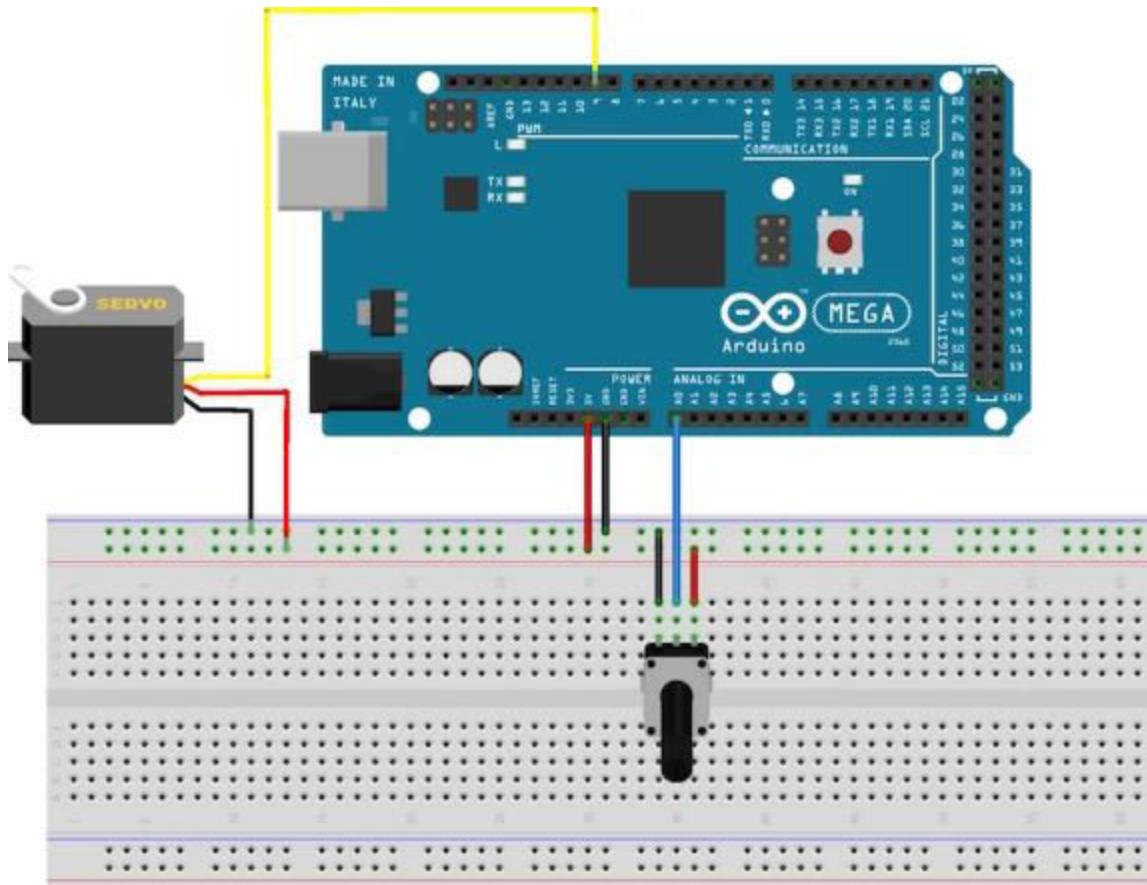
Let's put the above said to test and make a practical example of controlling a hobby servo using Arduino. I will use the MG996R which is a high-torque servo featuring metal gearing with stall torque of 10 kg-cm. The high torque comes at a price and that's the stall current of the servo which is 2.5A. The running current is from 500mA to 900mA and the operating voltage is from 4.8 to 7.2V.



We simply need to connect the control pin of the servo to any digital pin of the Arduino board, connect the Ground and the positive wires to the external 5V power supply, and also connect the Arduino ground to the servo ground.

#### 7.6.4 - Arduino Servo Motor Control Code

Connect a 10K potentiometer and a servo motor with arduino using this diagram.



### Code : Example > Servo > Knob

```
#include <Servo.h>

Servo myservo; // create servo object to control a servo

int potpin = 0; // analog pin used to connect the potentiometer
int val; // variable to read the value from the analog pin

void setup() {
 myservo.attach(9); // attaches the servo on pin 9 to the servo object
}

void loop() {
 val = analogRead(potpin); // reads the value of the
 // potentiometer (value between 0 and 1023)
 val = map(val, 0, 1023, 0, 180); // scale it to use it with the servo
 // (value between 0 and 180)
 myservo.write(val); // sets the servo position according
```

```

to the scaled value
 delay(15); // waits for the servo to get there
}

```

### Code : Example > Servo > Sweep

```

#include <Servo.h>

Servo myservo; // create servo object to control a servo
// twelve servo objects can be created on most boards

int pos = 0; // variable to store the servo position

void setup() {
 myservo.attach(9); // attaches the servo on pin 9 to the servo object
}

void loop() {
 for (pos = 0; pos <= 180; pos += 1) {
 // goes from 0 degrees to 180 degrees
 // in steps of 1 degree
 myservo.write(pos);
 // tell servo to go to position in variable 'pos'
 delay(15);
 // waits 15ms for the servo to reach the position
 }
 for (pos = 180; pos >= 0; pos -= 1) {
 // goes from 180 degrees to 0 degrees
 myservo.write(pos);
 // tell servo to go to position in variable 'pos'
 delay(15);
 // waits 15ms for the servo to reach the position
 }
}

```

### Servo Control With Serial Monitor >

```

#include <Servo.h>

Servo myservo; // create servo object to control a servo
// twelve servo objects can be created on most boards

```

```

int pos = 0; // variable to store the servo position
char state;
void setup() {
 myservo.attach(9); // attaches the servo on pin 9 to the servo object
 Serial.begin(9600);
}

void loop() {
 if (Serial.available() > 0) {
 state = Serial.read();
 if (state == 'O' || state == 'o') {
 myservo.write(40);
 delay(15);
 }
 if (state == 'L' || state == 'l') {
 myservo.write(140);
 delay(15);
 }
 }
}

```

## 7.7 - Battery - Battery Types ,Discharge Rate, Series, Parallel

[reference - <https://medium.com/husarion-blog/batteries-choose-the-right-power-source-for-your-robot-5417a3ec19ca>

<https://www.engineersgarage.com/article/choosing-battery-robots>

<https://www.batterysolutions.com/recycling-information/battery-types/>

<https://www.robotshop.com/community/tutorials/show/basics-how-do-i-choose-a-battery>

[https://batteryuniversity.com/learn/article/serial\\_and\\_parallel\\_battery\\_configurations](https://batteryuniversity.com/learn/article/serial_and_parallel_battery_configurations)

[http://web.mit.edu/evt/summary\\_battery\\_specifications.pdf](http://web.mit.edu/evt/summary_battery_specifications.pdf)

<https://www.sciencedirect.com/topics/engineering/battery-capacity>

<https://rogershobbycenter.com/lipoquide>

]

### 7.7.1 - Battery Types

#### 7.7.1.1 - Overview

There are various types of batteries available for robotics.

Batteries that are great for robotics are :

**Li-Ion** - lithium-ion battery

**Li-Poly** - lithium polymer batteries

**NiMH** - nickel metal hydride battery

In fact, Li-Poly batteries are the sub-group of the Li-Ion batteries. We can say that they are a special version of regular Li-Ion batteries. Why are they special? The difference is that during production, Li-Ion cells need to be pressed into a metal can (usually cylindrical) so that they remain in one piece. Li-Poly cells, which were introduced later, have different construction and can hold themselves up without the support of the external cylinder.

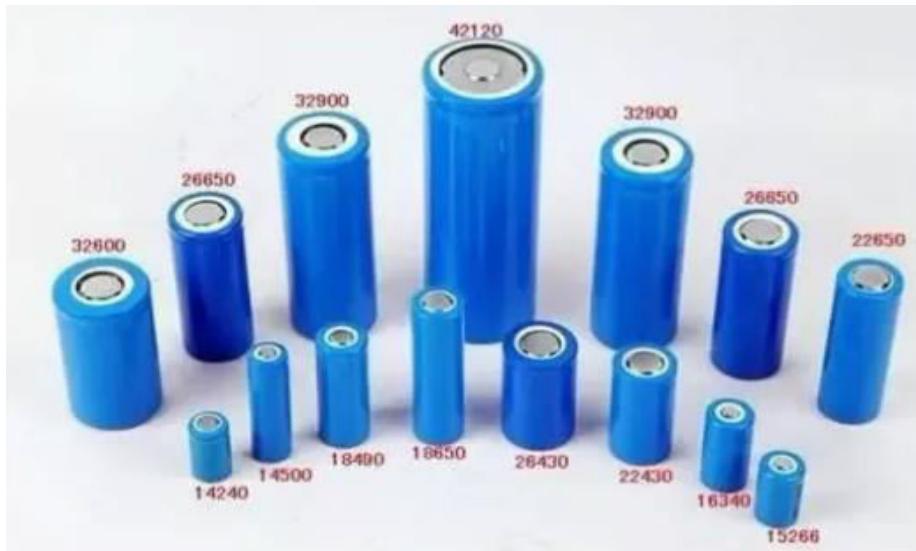
However, electrical parameters of Li-Ion and Li-Poly are almost identical.

The NiMH batteries are popular due to low internal resistance and good power-to-weight ratio. They are also much safer than Li-based cells. Explosions of such batteries are really rare (however I wouldn't recommend throwing them in the fire). The specific energy (energy-to-weight ratio) of NiMH is much worse than lithium cells.

### 7.7.1.1 - Battery Shapes

#### 7.7.1.1.1 - Cylindrical





The most popular form of any batteries is a cylindrical can. Used for Li-Ion, NiMH and very rarely for Li-Poly cells (they doesn't need a can as mentioned earlier). The cells on the photo above have an 18mm diameter and are 65mm long. That's the reason why they are called "18650" cells. So, If you find a 18650 cell, you can be almost certain that it's a Li-Ion cell.

You can see that the yellow one has a flat "+" terminal—it is prepared for welding cells in a group to form battery packs (like the one shown below). The blue, "consumer" cell has a raised "+" terminal, so it can be easily inserted and removed from a battery holder with a spring on the "-" terminal side. Consumer cells often have built-in protection circuits and therefore are about 2–4mm longer than regular 18650 cells.

Other popular dimensions for Li-Ions and NiMH cells are 14500 (14x50mm) widely known as AA. You can also find 18500, 26650, 16650, AAA, C, D and many other types.



#### 7.7.1.1.2 - Prismatic

This shape is used when the battery construction requires a metal can. The prismatic Li-Ion cells were commonly used in handheld devices (like cell-phones) when the Li-Poly batteries were not yet popular. Fun fact: Nokia 3310 used a NiMH battery in a prismatic shape.



### 7.7.1.1.3 - Pouch

Li-Poly batteries, do not need a metal enclosure. They are produced in form of thin, elastic slices, which are stacked and inserted into the pouches instead of being rolled up into the can. Pouch cells are cost-effective and are usually very thin therefore are a perfect fit when it comes to supplying smartphones, tablets and netbooks with power.



They  
stacked in blocks:

can also be



### 7.7.1.2 -

#### Example robots—what batteries to choose?

Robots are different, so is their appetite for power. Some example robot and recommended battery types -

#### **Mini-sumo robot -**

NiMH: 7x AAA battery pack—FDK HR-4U or similar

Li-Ion: 3x18650 battery pack: Tenergy 31012, 11.1V 2200mAh with protection circuit

Li-Poly: Hyperion G5 50C 2S 850mAh LiPo, Turnigy, Gens ace

#### **Quadcopter -**

NiMH: not this time—we need a lot of energy from a weight unit; Li- based cells are much better

Li-Ion: 12x18650 (4S3P) battery pack made of KeepPower IMR18650 2500mAh 3,7V 20A cells (20A continuous, 35A peak)

Li-Poly: Turnigy Graphene 5000mAh 4S 45C Lipo Pack w/XT90 (45C continuous, 90C peak)

#### **Small AGV -**

NiMH: not this time—energy is a priority; Li- based cells are much better

Li-Ion: 6x18650 (3S2P) battery pack made of INR18650–35E Samsung 3500mAh

Li-Poly: a 3S1P battery pack made of LP616594 4700mAh 3.7V

### **7.7.2 - Battery Capacity , Discharge Rate & Internal Resistance**

#### **7.7.2.1 - Battery Capacity**

The energy stored in a battery, called the battery capacity, is measured in either watt-hours (Wh), kilowatt-hours (kWh), or ampere-hours (Ahr). The most common measure of battery capacity is Ah, defined as the number of hours for which a battery can provide a current equal to the discharge rate at the nominal voltage of the battery. The unit of Ah is commonly used when working with battery systems as the battery voltage will vary throughout the charging or discharging cycle. The Wh capacity can be approximated from the Ahr capacity by multiplying the AH capacity by the nominal (or, if known, time average) battery voltage. A more accurate approach takes into account the variation of voltage by integrating the AH capacity x V(t) over the time of the charging cycle. For example, a 12 volt battery with a capacity of 500 Ah battery allows energy storage of approximately  $100 \text{ Ah} \times 12 \text{ V} = 1,200 \text{ Wh}$  or 1.2 KWh. However, because of the large impact from charging rates or temperatures, for practical or accurate analysis, additional information about the variation of battery capacity are also provided by battery manufacturers.

#### **7.7.2.2 - Battery Discharge Rate**

Voltage and Capacity had a direct impact on certain aspects of the vehicle, whether it's speed or run time. This makes them easy to understand. The Discharge Rating (I'll be referring to it as the C Rating from now on) is a bit harder to understand, and this has lead to it being the most over-hyped and misunderstood aspects of LiPo batteries.

*The C Rating is simply a measure of how fast the battery can be discharged safely and without harming the battery.* One of the things that makes it complicated is that it's not a stand-alone

number; it requires you to also know the capacity of the battery to ultimately figure out the safe amp draw (the "C" in C Rating actually stands for **Capacity**). Once you know the capacity, it's pretty much a plug-and-play math problem. Using the above battery, here's the way you find out the maximum safe continuous amp draw:

$$\mathbf{50C = 50 \times Capacity \text{ (in Amps)}}$$

**Calculating the C-Rating of our example battery:  $50 \times 5 = 250A$**

The resulting number is the maximum sustained load you can safely put on the battery. Going higher than that will result in, at best, the degradation of the battery at a faster than normal pace. At worst, it could burst into flames. So our example battery can handle a maximum continuous load of 250A.

Most batteries today have two C Ratings: a Continuous Rating (which we've been discussing), and a Burst Rating. The Burst rating works the same way, except it is only applicable in 10-second bursts, not continuously. For example, the Burst Rating would come into play when accelerating a vehicle, but not when at a steady speed on a straight-away. The Burst Rating is almost always higher than the Continuous Rating. Batteries are usually compared using the Continuous Rating, not the Burst Rating.

There is a lot of vitriolic comments on the Internet about what C Rating is best. Is it best to get the highest you can? Or should you get a C Rating that's just enough to cover your need? There isn't a simple answer. All I can give you is my take on the issue. When I set up a customer with a LiPo battery, I first find out what the maximum current his or her application will draw. Let's look at how that works.

Let's assume that our example customer is purchasing a Slash VXL R/C truck. That motor, according to Traxxas, has a maximum continuous current draw of 65A and a burst draw of 100A. Knowing that, I can safely say that a **2S 5000mAh 20C LiPo** will be sufficient, and will in fact have more power than we need. Remember, it has a maximum safe continuous discharge rating of 100A, more than enough to handle the 65A the Velineon motor will draw. Similarly, the Burst Rate of 150A easily covers the 100A the motor could draw.

However, the ratings on the motor aren't the whole picture. The way the truck is geared, the terrain the truck is driving on, the size of the tires, the weight of the truck... all of these things have an impact on the final draw on the battery. It's very possible that the final draw on the battery is higher than the maximum motor draw. So having that little bit of overhead is crucial, because you can't easily figure out a hard number that the truck will never go over.

For most applications, a 20C or 25C battery should be fine. But if you're driving a heavy truck, or you're geared up for racing, or you have a large motor for 3D flying applications, you should probably start around a 40C battery pack. But since there is no easy way to figure this out, I encourage you to talk to your local hobby shop to have them help determine which battery pack is right for your application.

### **7.7.2.1 - Internal Resistance of Battery (College Level)**

There is one very important rating we haven't talked about yet: Internal Resistance (or IR). Problem is, you won't find the IR rating anywhere on the battery. That's because the internal resistance of a battery changes over time, and sometimes because of the temperature. However, just because you can't read the rating on the battery doesn't mean it isn't important. In a way, the internal resistance is one of the most important ratings for a battery.

To understand why the IR is important, we have to understand what it is. In simple terms, **Internal Resistance** is a measure of the difficulty a battery has delivering its energy to your motor and speed control (or whatever else you have a battery hooked up to). The higher the number, the harder it is for the energy to reach its preferred destination. The energy that doesn't "go all the way" is lost as heat. So the internal resistance is kind of a measure of the efficiency of the battery.

**Internal Resistance is measured in milliohms ( $m\Omega$ ).**

**1,000 milliohms is equal to 1 Ohm ( $\Omega$ )**

Measuring the IR of your battery requires a special toolset. You either need a charger that will measure it for you or a tool that specifically measures internal resistance. Given that the only tool I have found for this (at least in the hobby world) is almost as expensive as a charger that does this for you, I'd go with a charger for this process. Some chargers measure each cell's IR separately, and some measure the entire battery pack as a whole. Since internal resistance is a cumulative effect, and the cells are wired in series, if you have a charger that does each cell independently, you need to add up the IR values of each cell, like this:

Suppose we have a 3S (3-cell) LiPo battery, and the measuring the cells independently yields these results.

**Cell 1: 3  $m\Omega$    Cell 2: 5  $m\Omega$    Cell 3: 4  $m\Omega$**

To find the total internal resistance for the battery pack, we would add up the values for the three cells.

$$3\Omega + 5\Omega + 4\Omega = 12 \text{ m}\Omega$$

For a charger that measures the pack as a whole, all you would see is the 12 mΩ - the rest would be done for you - behind the scenes, as it were. Either way, the goal is to have the IR for the entire pack.

The first reason internal resistance is important has to do with your battery's health. As a LiPo battery is used, a build up of Li<sub>2</sub>O forms on the inside terminals of the battery (we'll go more in depth on this later in the Discharging section). As that build up occurs, the IR goes up, making the battery less efficient. After many, many uses, the battery will simply wear out and be unable to hold on to any energy you put in during charging - most of it will be lost as heat. If you've ever seen a supposed fully charged battery discharge almost instantly, a high IR is probably to blame.

To understand how Internal Resistance works in R/C applications, first we have to understand Ohm's Law. It says that the current (Amps) through a conductor between two points is directly proportional to the difference in voltage across those two points. The modern formula is as follows: Amps = Volts / Resistance. In the formula, the resistance is measured in Ohms, not milliohms, so we'd have to convert our measurements. If we use our previous 3S LiPo, and plug it into the equation along with a 1A draw, we can find out how much our battery pack's voltage will drop as a result of the load. First, we have to change the equation to solve for volts, which would look like this:

$$\text{Amps} \times \text{Resistance} = \text{Volts}$$

So plugging in our numbers and solving the equation would look like this:

$$1\text{A} \times 0.012 \Omega = 0.012\text{V}$$

So our battery would experience a tiny drop in voltage when a 1A load is applied. Considering our 3S LiPo is around 12.6V when fully charged, that's not a big deal, right? Well, let's see what happens when we increase the load to 10A.

$$\mathbf{10A \times 0.012 \Omega = 0.120V}$$

Now we see that when we increased the load 10X, we also increased the voltage drop 10X. But neither of these examples are very "real world". Let's use the Slash VXL from the previous section and plug those numbers in. If you recall, our Velineon motor has a maximum continuous current rating of 65A. Let's assume we manage to hit that mark when driving and use that.

$$\mathbf{65A \times 0.012 \Omega = 0.780V}$$

Wow, more than 3/4 of a volt! That's around 6.2% of the total voltage of our battery pack. Pretty respectable, but it's still a reasonable drop in voltage.

"So, yeah, the voltage drops. But so what? What does that actually mean? How does it effect my R/C vehicle?" Well, let's continue on with our example to show you.

The Velineon motor our Slash VXL uses has a Kv rating of 3500. That means it spins 3,500 RPM per volt. On a fully charged 3S LiPo we'll see this (assuming no voltage drop):

$$\mathbf{12.6V \times 3500RPM = 44,100 RPM}$$

Now, assuming we can hit that 65A draw on our unloaded motor (which we can't in real life, but for the purposes of demonstration we can), here's the RPM on the same motor with our voltage drop from before:

$$\mathbf{11.82V \times 3500RPM = 41,370 RPM}$$

Difference of 2,730 RPM

See the drop in performance? That's the effect Ohm's Law has on our hobby. A lower internal resistance means your car or truck or airplane or boat or helicopter goes faster and has more power.

This begs the question: how low should it be? Unfortunately, there's no easy answer for this. It's all dependant on your use case and battery. What is great for one battery may be terrible for another. Based on my online research, combined with my own experience and findings, I would say, as a general rule, a per cell rating of between 0-6 mΩ is as good as it gets. Between 7 and

12 mΩ is reasonable. 12 to 20 mΩ is where you start to see the signs of aging on a battery, and beyond 20mΩ per cell, you'll want to start thinking about retiring the battery pack. But this is only a guide - there is no hard rule set here. And if your charger doesn't give you the per cell measurements, you'll have to divide your total count by the number of cells in your battery to get an approximate per cell rating.

### 7.7.2 - Battery Series & Parallel

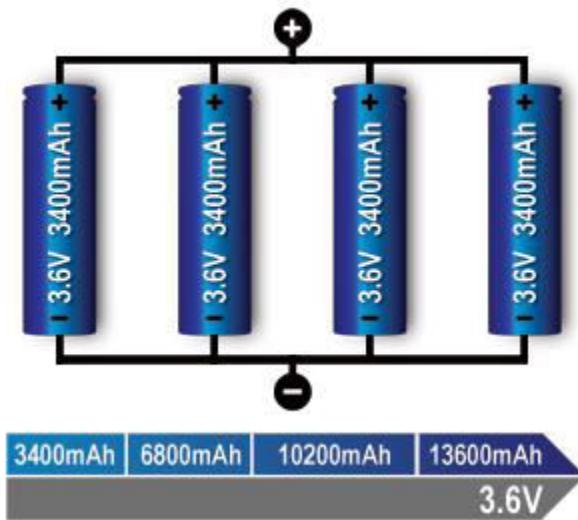
[https://batteryuniversity.com/learn/article/serial\\_and\\_parallel\\_battery\\_configurations](https://batteryuniversity.com/learn/article/serial_and_parallel_battery_configurations)

#### 7.7.2.1 - Battery Series



For Battery Series Connection > Voltage increases , Capacity remains same  
This connection is called 4S1P(4 Series 1 Parallel)

#### 7.7.2.2 - Battery Parallel



For Battery Parallel Connection > Capacity increases , Voltage remains same  
This connection is called 1S4P (1 Series 4 Parallel)

College Level:

## 7.8 - Basic Interrupt

[reference - <http://playground.arduino.cc/code/interrupts>

<https://www.arduino.cc/reference/en/language/functions/external-interrupts/attachinterrupt/>

<http://www.avr-tutorials.com/interrupts/about-avr-8-bit-microcontrollers-interrupts>

<http://gammon.com.au/interrupts>

]

### 7.8.1 - Overview

Interrupts are basically events that require immediate attention by the microcontroller. When an interrupt event occurs the microcontroller pause its current task and attend to the interrupt by executing an Interrupt Service Routine (ISR) at the end of the ISR the microcontroller returns to the task it had pause and continue its normal operations.

In order for the microcontroller to respond to an interrupt event the interrupt feature of the microcontroller must be enabled along with the specific interrupt. This is done by setting the Global Interrupt Enabled bit and the Interrupt Enable bit of the specific interrupt.

### 7.8.2 - Using Interrupts

Interrupts are useful for making things happen automatically in microcontroller programs, and can help solve timing problems. Good tasks for using an interrupt may include reading a rotary encoder, or monitoring user input.

If you wanted to insure that a program always caught the pulses from a rotary encoder, so that it never misses a pulse, it would make it very tricky to write a program to do anything else, because the program would need to constantly poll the sensor lines for the encoder, in order to catch pulses when they occurred. Other sensors have a similar interface dynamic too, such as trying to read a sound sensor that is trying to catch a click, or an infrared slot sensor (photo-interrupter) trying to catch a coin drop. In all of these situations, using an interrupt can free the microcontroller to get some other work done while not missing the input.

### 7.8.3 - About Interrupt Service Routines

ISRs are special kinds of functions that have some unique limitations most other functions do not have. An ISR cannot have any parameters, and they shouldn't return anything.

Generally, an ISR should be as short and fast as possible. If your sketch uses multiple ISRs, only one can run at a time, other interrupts will be executed after the current one finishes in an order that depends on the priority they have. millis() relies on interrupts to count, so it will never increment inside an ISR. Since delay() requires interrupts to work, it will not work if called inside an ISR. micros() works initially, but will start behaving erratically after 1-2 ms.

delayMicroseconds() does not use any counter, so it will work as normal.

Typically global variables are used to pass data between an ISR and the main program. To make sure variables shared between an ISR and the main program are updated correctly, declare them as volatile.

### 7.8.4 - Digital Pins With Interrupts

The first parameter to **attachInterrupt()** is an interrupt number. Normally you should use digitalPinToInterrupt(pin) to translate the actual digital pin to the specific interrupt number. For example, if you connect to pin 3, use **digitalPinToInterrupt(3)** as the first parameter to **attachInterrupt()**. Inside the attached function, delay() won't work and the value returned by

`millis()` will not increment. Serial data received while in the function may be lost. You should declare as volatile any variables that you modify within the attached function.

| BOARD                             | DIGITAL PINS USABLE FOR INTERRUPTS                                               |
|-----------------------------------|----------------------------------------------------------------------------------|
| Uno, Nano, Mini, other 328-based  | 2, 3                                                                             |
| Uno WiFi Rev.2                    | all digital pins                                                                 |
| Mega, Mega2560, MegaADK           | 2, 3, 18, 19, 20, 21                                                             |
| Micro, Leonardo, other 32u4-based | 0, 1, 2, 3, 7                                                                    |
| Zero                              | all digital pins, except 4                                                       |
| MKR Family boards                 | 0, 1, 4, 5, 6, 7, 8, 9, A1, A2                                                   |
| Due                               | all digital pins                                                                 |
| 101                               | all digital pins (Only pins 2, 5, 7, 8, 10, 11, 12, 13 work with <b>CHANGE</b> ) |

### 7.8.5 - Syntax

`attachInterrupt(digitalPinToInterrupt(pin), ISR, mode);` (recommended)

`attachInterrupt(interrupt, ISR, mode);` (not recommended)

`attachInterrupt(pin, ISR, mode);` (Not recommended. Arduino SAMD Boards, Uno WiFi Rev2, Due, 101 only)

### 7.8.6 - Parameters

**interrupt:** the number of the interrupt (int)

**pin:** the pin number

**ISR:** the ISR to call when the interrupt occurs; this function must take no parameters and return nothing. This function is sometimes referred to as an interrupt service routine.

**mode:** defines when the interrupt should be triggered. Four constants are predefined as valid values:

- **LOW** to trigger the interrupt whenever the pin is low,
- **CHANGE** to trigger the interrupt whenever the pin changes value
- **RISING** to trigger when the pin goes from low to high,
- **FALLING** for when the pin goes from high to low.

The Due, Zero and MKR1000 boards allows also:

- **HIGH** to trigger the interrupt whenever the pin is high.

reference - <https://www.youtube.com/watch?v=CRJUdf5TTQQ>

### 7.8.6 - Example Code

```
const byte ledPin = 13;
const byte interruptPin = 2;
volatile byte state = LOW;

void setup() {
 pinMode(ledPin, OUTPUT);
 pinMode(interruptPin, INPUT_PULLUP);
 attachInterrupt(digitalPinToInterrupt(interruptPin), blink, CHANGE);
}

void loop() {
 digitalWrite(ledPin, state);
}

void blink() {
 state = !state;
}
```

### 7.8.7 - Interrupt Numbers

Normally you should use **digitalPinToInterrupt(pin)**, rather than place an interrupt number directly into your sketch. The specific pins with interrupts, and their mapping to interrupt number varies on each type of board. Direct use of interrupt numbers may seem simple, but it can cause compatibility trouble when your sketch is run on a different board.

However, older sketches often have direct interrupt numbers. Often number 0 (for digital pin 2) or number 1 (for digital pin 3) were used. The table below shows the available interrupt pins on various boards.

Note that in the table below, the interrupt numbers refer to the number to be passed to **attachInterrupt()**. For historical reasons, this numbering does not always correspond directly to the interrupt numbering on the ATmega chip (e.g. int.0 corresponds to INT4 on the ATmega2560 chip).

| BOARD                                  | INT.0 | INT.1 | INT.2 | INT.3 | INT.4 | INT.5 |
|----------------------------------------|-------|-------|-------|-------|-------|-------|
| Uno, Ethernet                          | 2     | 3     |       |       |       |       |
| Mega2560                               | 2     | 3     | 21    | 20    | 19    | 18    |
| 32u4 based<br>(e.g Leonardo,<br>Micro) | 3     | 2     | 0     | 1     | 7     |       |

For Uno WiFiRev.2, Due, Zero, MKR Family and 101 boards the interrupt number = pin number.

### 7.8.8 - detachInterrupt()

Turns off the given interrupt.

#### Syntax

**detachInterrupt(digitalPinToInterrupt(pin))** (recommended)

**detachInterrupt(interrupt)** (not recommended)

**detachInterrupt(pin)** (Not recommended. Arduino SAMD Boards, Uno WiFi Rev2, Due, 101 only)

#### Parameters

**interrupt:** the number of the interrupt to disable

**pin:** the pin number of the interrupt to disable

## 7.9 - VS Code for Arduino

Follow the link:

<https://medium.com/home-wireless/use-visual-studio-code-for-arduino-2d0cf4c1760b>

<https://medium.com/home-wireless/use-visual-studio-code-for-arduino-2d0cf4c1760b>

## 7.10 - AnalogWrite Resolution

Follow the link :

<https://www.arduino.cc/reference/en/language/functions/zero-due-mkr-family/analogwriteresolution/>

**analogWriteResolution()** is an extension of the Analog API for the Arduino Due.

**analogWriteResolution()** sets the resolution of the **analogWrite()** function. It defaults to 8 bits (values between 0-255) for backward compatibility with AVR based boards.

The **Due** has the following hardware capabilities:

- 12 pins which default to 8-bit PWM, like the AVR-based boards. These can be changed to 12-bit resolution.
- 2 pins with 12-bit DAC (Digital-to-Analog Converter)

By setting the write resolution to 12, you can use **`analogWrite()`** with values between 0 and 4095 to exploit the full DAC resolution or to set the PWM signal without rolling over.

The **Zero** has the following hardware capabilities:

- 10 pins which default to 8-bit PWM, like the AVR-based boards. These can be changed to 12-bit resolution.
- 1 pin with 10-bit DAC (Digital-to-Analog Converter).

By setting the write resolution to 10, you can use **`analogWrite()`** with values between 0 and 1023 to exploit the full DAC resolution

The **MKR Family** of boards has the following hardware capabilities:

- 4 pins which default to 8-bit PWM, like the AVR-based boards. These can be changed from 8 (default) to 12-bit resolution.
- 1 pin with 10-bit DAC (Digital-to-Analog Converter)

By setting the write resolution to 12 bits, you can use **`analogWrite()`** with values between 0 and 4095 for PWM signals; set 10 bit on the DAC pin to exploit the full DAC resolution of 1024 values.

## 7.11 - Internal Resistance of Battery (see 7.7.2.1)

## Class 8:

- 8.1 - Variable Data Types and Size Details
- 8.2 - Project : Bluetooth Control Robot Car (Part 1)
  - 8.2.1 - Various motor types
  - 8.2.2 - Parameter for choosing a motor
  - 8.2.3 - DC Motor Control Mechanism
  - 8.2.4 - DC Motor Driver IC
  - 8.2.5 - DC Motor Control with Serial

### College Level:

6 Motor Drive using 3 Motor Driver

## 8.1 - Variable Data Types and Size Details

[reference - <https://learn.sparkfun.com/tutorials/data-types-in-arduino/all>  
<https://learn.adafruit.com/adafruit-motor-selection-guide/dc-motors>]

The Arduino environment is really just C++ with library support and built-in assumptions about the target environment to simplify the coding process. C++ defines a number of different data types; here we'll talk only about those used in Arduino with an emphasis on traps awaiting the unwary Arduino programmer.

Below is a list of the data types commonly seen in Arduino, with the memory size of each in parentheses after the type name. Note: **signed** variables allow both positive and negative numbers, while **unsigned** variables allow only positive values.

- **boolean** (8 bit) - simple logical true/false
- **byte** (8 bit) - unsigned number from 0-255
- **char** (8 bit) - signed number from -128 to 127. The compiler will attempt to interpret this data type as a character in some circumstances, which may yield unexpected results
- **unsigned char** (8 bit) - same as 'byte'; if this is what you're after, you should use 'byte' instead, for reasons of clarity
- **word** (16 bit) - unsigned number from 0-65535
- **unsigned int** (16 bit)- the same as 'word'. Use 'word' instead for clarity and brevity
- **int** (16 bit) - signed number from -32768 to 32767. This is most commonly what you see used for general purpose variables in Arduino example code provided with the IDE
- **unsigned long** (32 bit) - unsigned number from 0-4,294,967,295. The most common usage of this is to store the result of the millis() function, which returns the number of milliseconds the current code has been running
- **long** (32 bit) - signed number from -2,147,483,648 to 2,147,483,647
- **float** (32 bit) - signed number from -3.4028235E38 to 3.4028235E38. Floating point on the Arduino is not native; the compiler has to jump through hoops to make it work. If you can avoid it, you should. We'll touch on this later.

## 8.2 - Project : Bluetooth Control Robot Car (Part 1)

We will build a robot which will have two motor , a L298 motor driver shield and a HC-05 bluetooth module.

### 8.2.1 - Various motor types

[reference - <https://www.circuito.io/blog/arduino-motor-guide/>]

#### 8.2.1.1 - DC Motors

While it may be a bit outdated, the standard Direct Current (DC) brushed motor is as simple as they come. Easy to assemble and cheap to make, they are found filling practically every role where electric motors are called for.

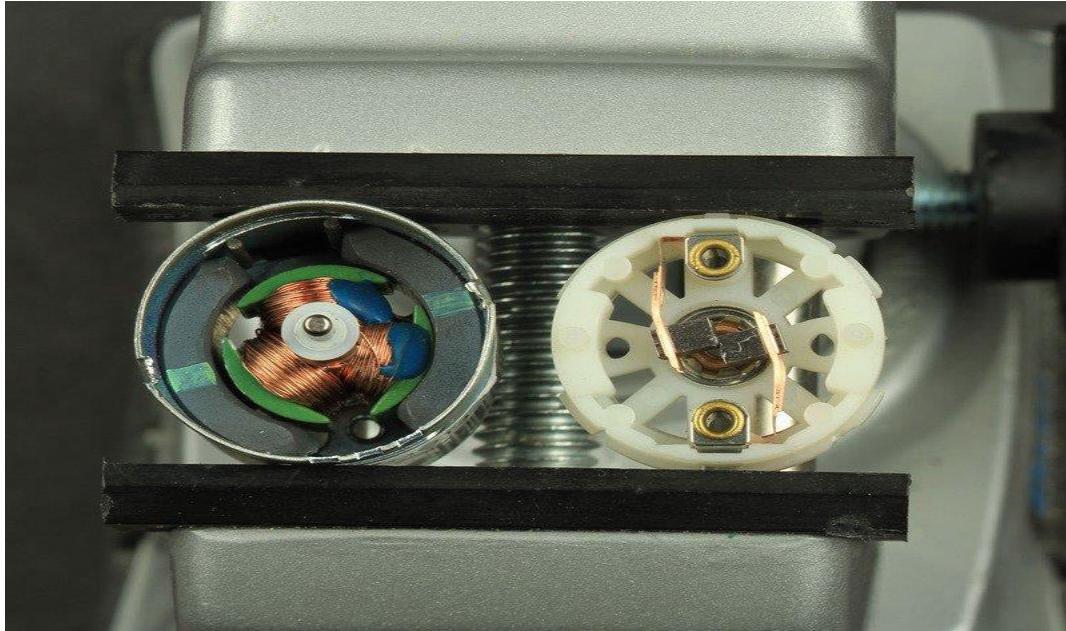
##### 8.2.1.1.1 - How DC Motor Works

The DC brushed motor consists of an odd numbered set of conductive windings arranged around a central axle to which the commutator is attached. The outer assembly contains two

magnets with opposite polarities. As the windings are energized they are charged and begin to get attracted towards the outer assembly resulting in rotation. This charge is altered as the brushes come in contact with the commutator and so allows the windings to continue to experience attraction towards the outer assembly, allowing the motor to rotate as long as power is provided. As you'd expect, simple motors are simple to control. In terms of power, increasing voltage or amperage can adjust the motor speed of rotation (RPMs) or Torque respectively. To reverse the direction of the motor, a simple polarity reversal of the motor contacts is all that is required.

To achieve the greater precision demanded in modern applications, DC brushed motors may be paired to a device called a wheel encoder/ Rotary encoder. These devices are able to read and sense the angular position of a motor's axle and later the microcontroller converts the output signal into digital information. This data can be used to determine location, motor speed and acceleration of the motor it is attached to.





**Pros:**

- High torque
- Simple assembly and easy to control
- Extremely cheap to manufacture

**Cons:**

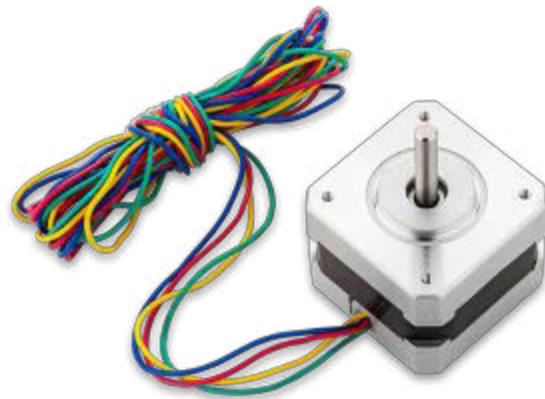
- Wear and tear on brushes reduces its lifespan
- Prolonged usage at higher RPM can cause brush heating and damage to the motor
- Small amounts of electromagnetic interference on radio frequencies
- demands more computation for closed loop control

### 8.2.1.2 - Stepper Motor

A variant of the regular DC motor, the stepper motor is the go-to choice when you need a motor capable of tightly controlled movements to provide precision beyond the capabilities of a standard motor. They are commonly used in the printing industry as well in simple robots such as factory robot arms configured for specific tasks.

#### 8.2.1.2.1 - How does this motor work

Unlike other motors which are intended to provide unlimited rotation when in use, stepper motors are designed to move a particular number of 'steps' at a particular speed when powered up. With each charge of the motor providing enough power for a single step. How small each 'step' indicates the overall accuracy capability of the motor.



#### **8.2.1.2.2 - Stepper motor control**

Unlike other motor types, the stepper motor is designed for use with more advanced electronics in order to take advantage of the fine motor control that it is capable of. For that reason, stepper motors require a driver circuit which connects it to a control system (such as an Arduino) capable of regulating power to the motor.

##### **Pros:**

- Extremely high precision control over the operation
- Simplicity of construction and reliability
- High torque - not all
- Open loop control

##### **Cons:**

- Highly specialized usage
- Requires advanced control system for proper operation

#### **8.2.1.3 - Brushless Motor**

An evolution of the brushed motor, brushless motors are quickly becoming the motor of choice for many hobbyists and enthusiasts owing to the immense potential and improved reliability over their predecessors. Like brushed motors, they provide a good deal of torque as well as being able to operate reliably at high RPM. This makes the brushless motor extremely versatile, very capable as an Arduino Motor and particularly popular with those building RC cars or drones.



#### **8.2.1.3.1 - How BLDC motor works**

The brushless motor uses an alternating current to create opposite charges between the windings and the magnets on the outer assembly. In this design only the outer assembly attached to the axle rotates; the benefit of having only a single moving part combined with removing the need for contact brushes is that brushless motors enjoy higher energy efficiency, longer operational lifetimes, smooth delivery of mechanical energy to the axle and low friction.

#### **8.2.1.3.2 - Brushless motor control**

While harder to control than a simple brushed motor, the use of modern technology has made them much more simple to control via computers. [Hall effect sensors](#), which can detect changes to magnetic fields and translate that into digital information are often used with brushless motors to monitor and control their output, just like wheel encoders are used with brushed motors.

##### **Pros:**

- Long lifespan
- Single moving part provides high reliability
- Low friction construction advantageous for extended operation at high speed
- Energy efficiency

##### **Cons:**

- Requires specialized control systems
- For optimal motor usage, requires the use of a gearbox regulate power delivery

#### **8.2.2 - Parameter for choosing a motor**

Choosing the right motor for your project will often depend on the type of project you're attempting to build and what performance indicators will determine the ideal motor for you. There are three main parameters that are co-dependent, that we need to take into consideration when choosing the right motor for our project

##### **8.2.2.1 - Torque**

While we won't be delving into serious math and physics in this article, an understanding of your motor's torque rating and how it influences your decisions when picking out your projects motor is important, in order to make sure you select a motor that meets your needs or expectations adequately.

Simply put, a motor's Torque rating is the amount of rotational force that your motor is capable of exerting on a load. Torque is determined by a simple formula:

**Torque = Distance from axis of rotation X Force**

For an example of how torque affects your creations, we can look at another field where torque is important, the automotive sector. When ascending hills or steep roads, an engine with a high torque rating is preferable as it is able to apply greater force to the load (the vehicle) allowing it to travel up the hill even at low speeds or a standing start.

A high torque rating is important for maintaining high rotational motor speed such as in a drone. A high torque motor can change RPM values quickly, an ability which should translate into smoother more responsive performance from an electric vehicle using such a motor. Good options for those looking for high torque motors are a standard DC brushed motors which, while cheap, has a shorter lifespan and requires more maintenance or a more expensive but more reliable and energy efficient DC brushless motor.

Keeping your motors torque rating in line with your creation's needs is important to ensure long-term operation as a torque rating that is too low makes your creation inoperable, while a torque rating that's too high could cause mechanical stress during operation. While the risk of catastrophic failure due to excessively high torque ratings is low when it comes to DIY electronics, it can still cause parts to wear out faster than they should and shorten the operational lifespan of your new invention.

#### **8.2.2.2 - Voltage - Velocity/RPM**

In this context, Velocity is the speed at which the electric motor rotates. This angular rotational speed is measured in revolutions per minute or RPM and along with torque are the two main factors that influence motor performance. In slightly oversimplified terms, where a high torque motor is good at moving heavier loads at a slow speed, a high RPM drive system will allow for higher speeds while reducing your overall load carrying capacity.

When adding an Arduino motor to the project you need to take into consideration your [power supply](#)- wall or batteries and make sure that the motor voltage characteristics are met in order to get the most out of it.

For advanced builders, a gearbox, similar to those used in regular cars allows motors to function well in both capacities as the need arises. As those of you who drive will already know, lower gears give you plenty of torque to start moving but you need to quickly change up to 2nd and 3rd gear to maintain acceleration. And if you've ever tried moving from a standing start in a high gear, say 4th or 5th on a manual transmission, you'll know how a high RPM without sufficient torque or inertial momentum is going to get you nowhere quickly.

In general, a brushless motor is a great option for those who want a reasonable torque along with reliability and energy efficiency even at higher RPMs but are not interested in dealing with the weight and complexity of a gearbox.

### **8.2.2.3 - Current/Ampereage**

When choosing an electric motor, it's important to select one that is able to provide the power you need while still keeping its power draw within acceptable limits. Wattage (electric power) is Volts X Amps. While the voltage is linked to increasing a motor's RPMs, likewise a higher amp rating is needed to increase torque. While having a powerful motor rated for high voltage, high amp operation, this translates into heavier power load requirements.

While this is unlikely to affect projects that rely on power from wall sockets, projects that are intended to be more portable such as unmanned vehicles, robots and wearables will have to take the motors requirements into account as the batteries powering the project will need to be set up to provide sufficient power. For this reason, motors that are as small, light and efficient as possible are always favored when it comes to developing portable electronics in order to minimize power draw and weight (both of the motor itself and the size of the power source needed to operate it).

There are many other factors that go into picking out the right motor, but having a good understanding of these 3 main criteria should be enough to steer you in the right direction. While any electric motor can, in theory, be adapted to serve practically any function with a bit of ingenuity, you should now have everything you need to pick out the optimal motor/s that best suit your project.

## **8.2.3 - DC Motor Control Mechanism**

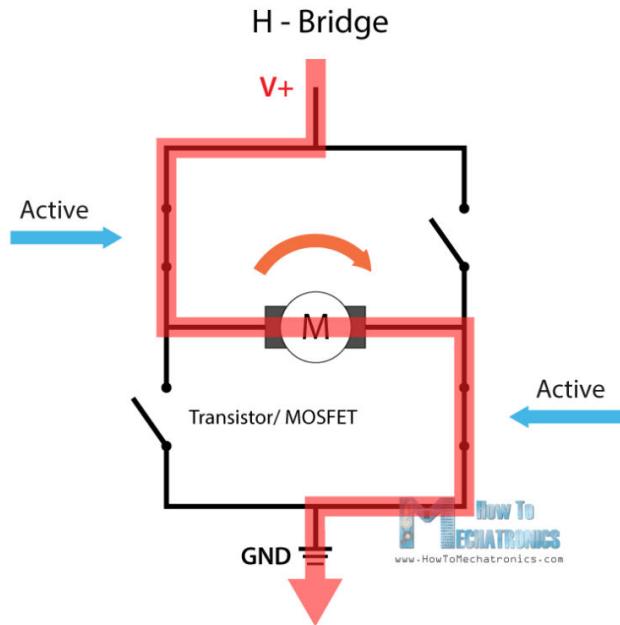
### **8.2.3.1 - Motor Direction Control**

Direction of normal DC motor depends on the polarity of applied voltage. If we flip the polarity the direction will also change. In a robot or rc car we use a technique called H-bridge .

#### **8.2.3.1.1 - H-Bridge DC Motor Control**

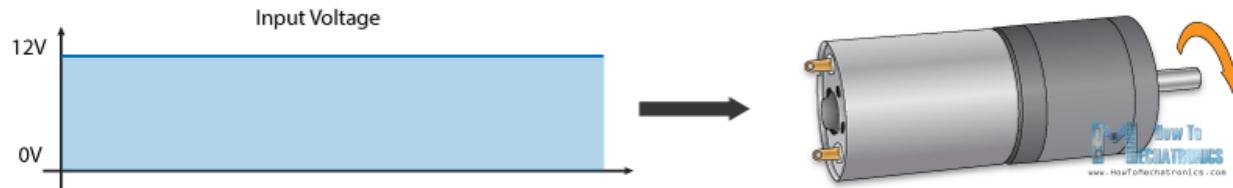
[reference - <https://howtomechatronics.com/tutorials/arduino/arduino-dc-motor-control-tutorial-l298n-pwm-h-bridge/>]

On the other hand, for controlling the rotation direction, we just need to inverse the direction of the current flow through the motor, and the most common method of doing that is by using an H-Bridge. An H-Bridge circuit contains four switching elements, transistors or MOSFETs, with the motor at the center forming an H-like configuration. By activating two particular switches at the same time we can change the direction of the current flow, thus change the rotation direction of the motor. So if we combine these two methods, the PWM and the H-Bridge, we can have a complete control over the DC motor. There are many DC motor drivers that have these features and the L298N is one of them.



### 8.2.3.2 - DC Motor Speed Control

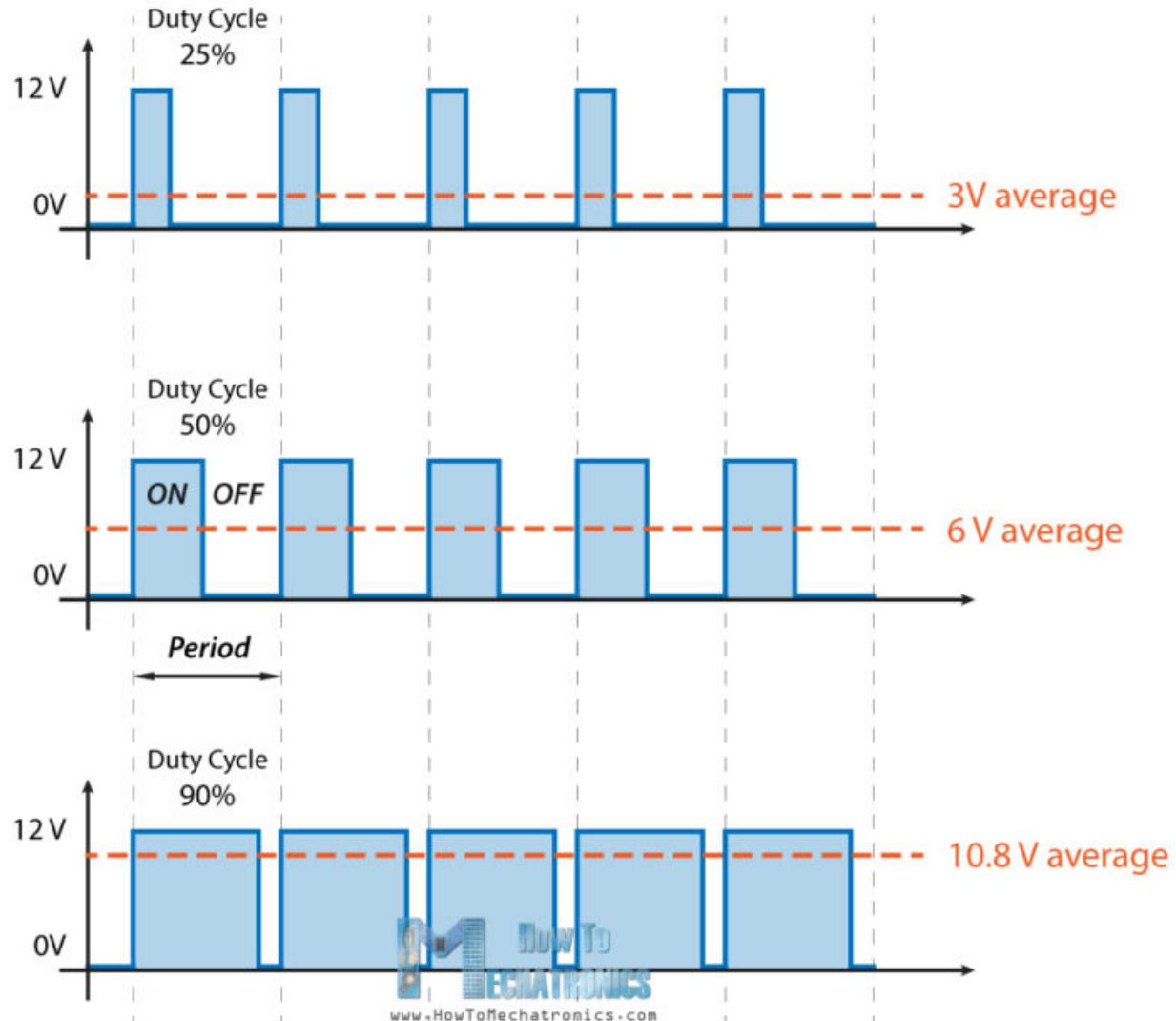
We can control the speed of the DC motor by simply controlling the input voltage to the motor and the most common method of doing that is by using PWM signal.



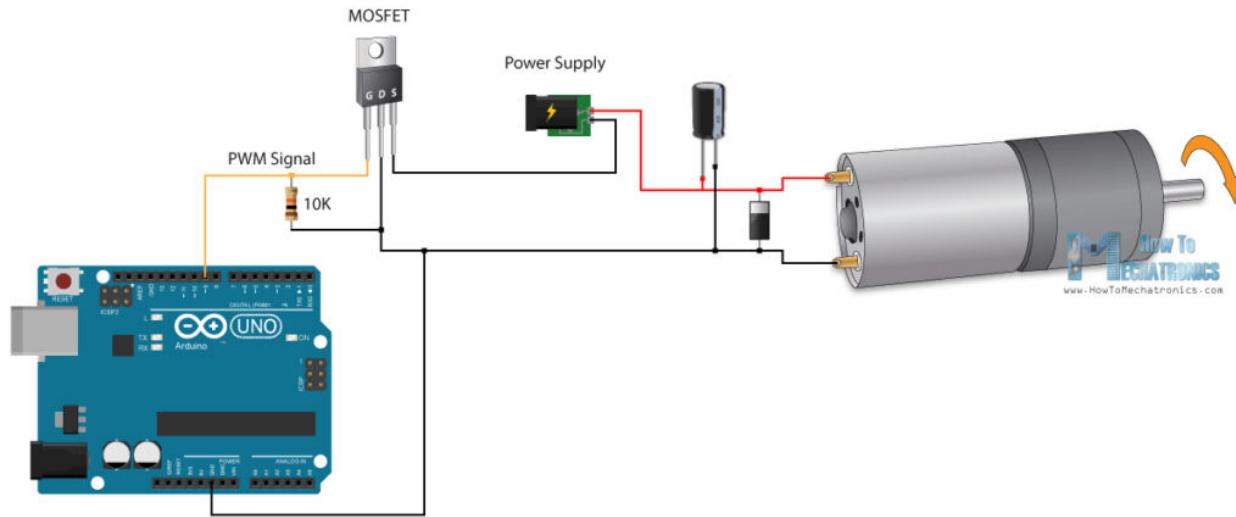
### PWM DC Motor Control

PWM, or pulse width modulation is a technique which allows us to adjust the average value of the voltage that's going to the electronic device by turning on and off the power at a fast rate. The average voltage depends on the duty cycle, or the amount of time the signal is ON versus the amount of time the signal is OFF in a single period of time.

## Pulse Width Modulation



So depending on the size of the motor, we can simply connect an Arduino PWM output to the base of transistor or the gate of a MOSFET and control the speed of the motor by controlling the PWM output. The low power Arduino PWM signal switches on and off the gate at the MOSFET through which the high power motor is driven.



#### 8.2.4 - DC Motor Driver IC

There are several H-bridge motor driver IC available in market. We can also make H-brighe driving circuit using MOSFET or Transistor. Some of IC's are L293D, L298N, L298P, L298P013TR, L6203, LMD18200T/NOPB, LMD18200T/NOPB, L293DNE, DRV8840 etc. Among them L293D and L298N are commonly used in arduino based robot.

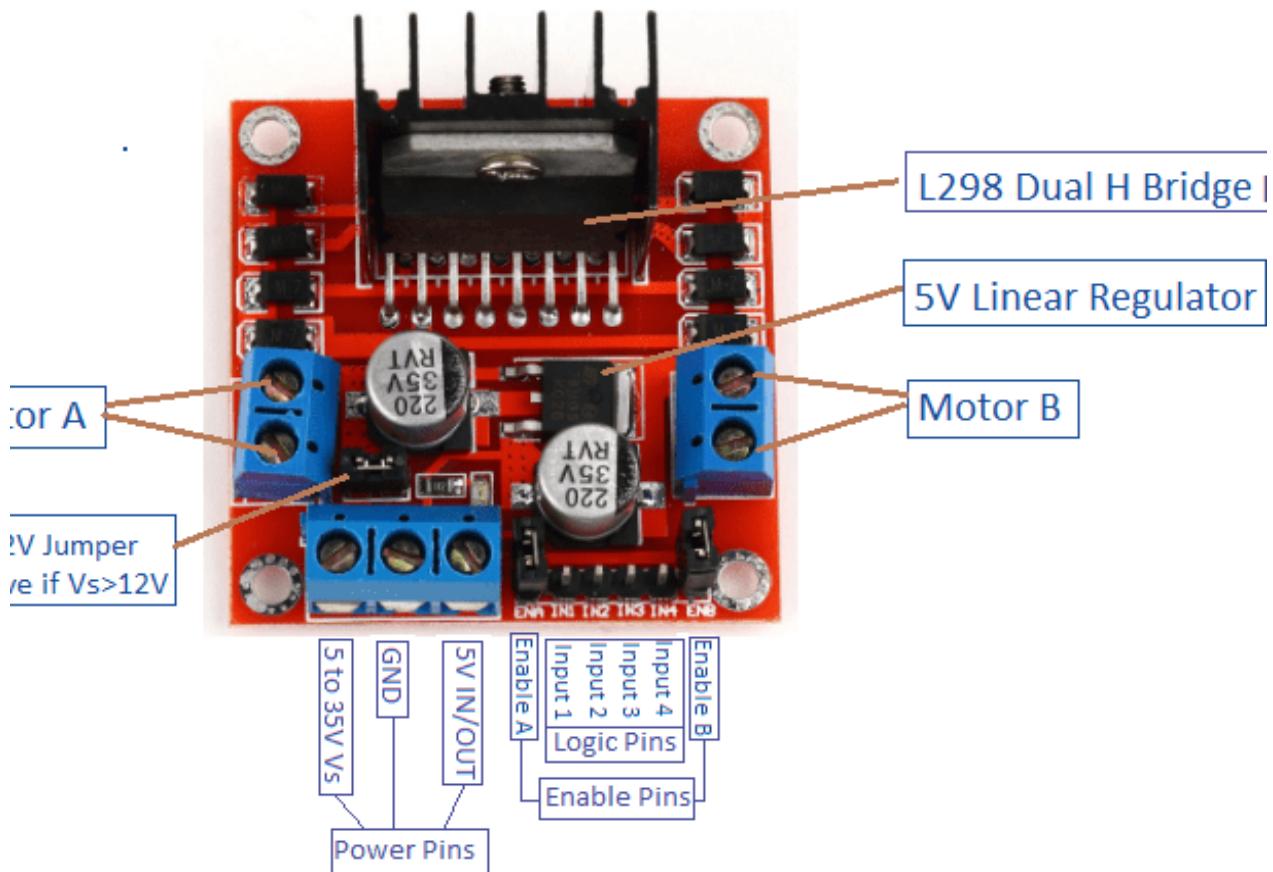
##### 8.2.4.1 - L298N Driver Connection

[reference - <https://howtomechatronics.com/tutorials/arduino/arduino-dc-motor-control-tutorial-l298n-pwm-h-bridge/>]

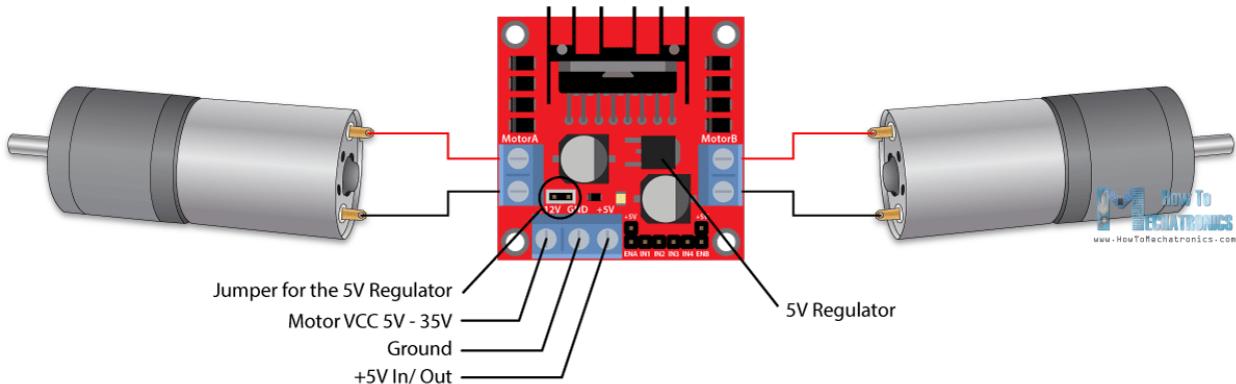
The L298N is a dual H-Bridge motor driver which allows speed and direction control of two DC motors at the same time. The module can drive DC motors that have voltages between 5 and 35V, with a peak current up to 2A.

The L298N is a dual H-Bridge motor driver which allows speed and direction control of two DC motors at the same time. The module can drive DC motors that have voltages between 5 and 35V, with a peak current up to 2A.

| ENA <sub>A</sub> | IN1 <sub>A</sub> | IN2 <sub>A</sub> | The State of DC Motor A <sub>A</sub> |
|------------------|------------------|------------------|--------------------------------------|
| 0 <sub>A</sub>   | X <sub>A</sub>   | X <sub>A</sub>   | Stop <sub>A</sub>                    |
| 1 <sub>A</sub>   | 0 <sub>A</sub>   | 0 <sub>A</sub>   | Brake <sub>A</sub>                   |
| 1 <sub>A</sub>   | 0 <sub>A</sub>   | 1 <sub>A</sub>   | Rotate Clockwise <sub>A</sub>        |
| 1 <sub>A</sub>   | 1 <sub>A</sub>   | 0 <sub>A</sub>   | Rotate Counterclockwise <sub>A</sub> |
| 1 <sub>A</sub>   | 1 <sub>A</sub>   | 1 <sub>A</sub>   | Brake <sub>A</sub>                   |

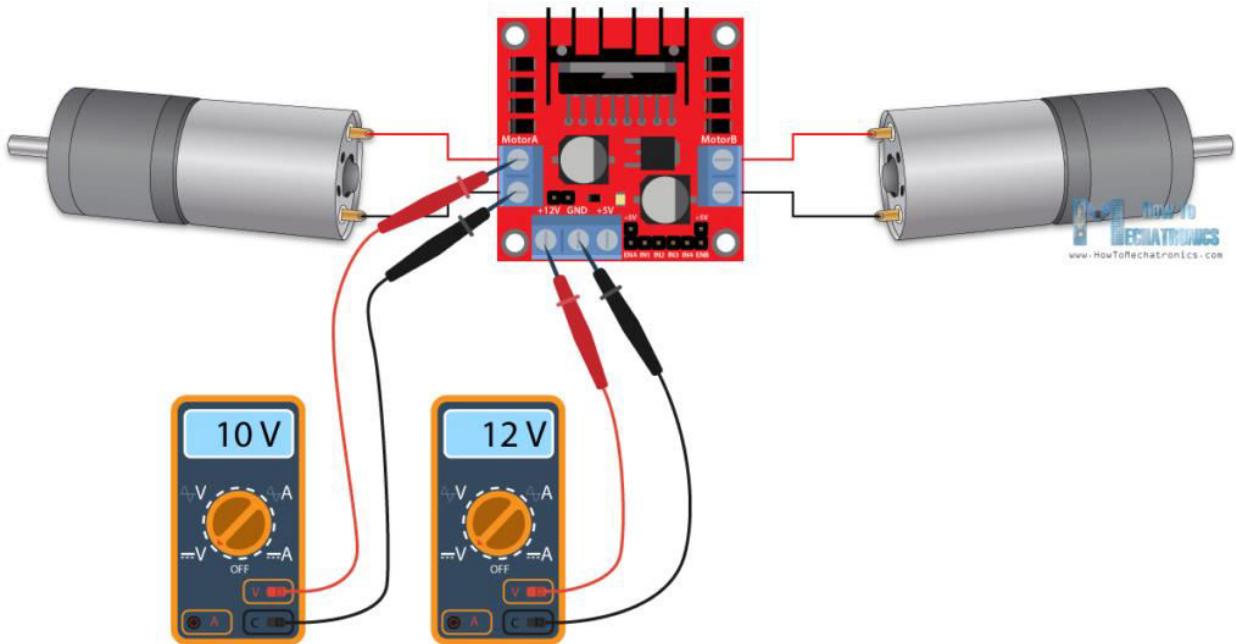


Let's take a closer look at the pinout of L298N module and explain how it works. The module has two screw terminal blocks for the motor A and B, and another screw terminal block for the Ground pin, the VCC for motor and a 5V pin which can either be an input or output.



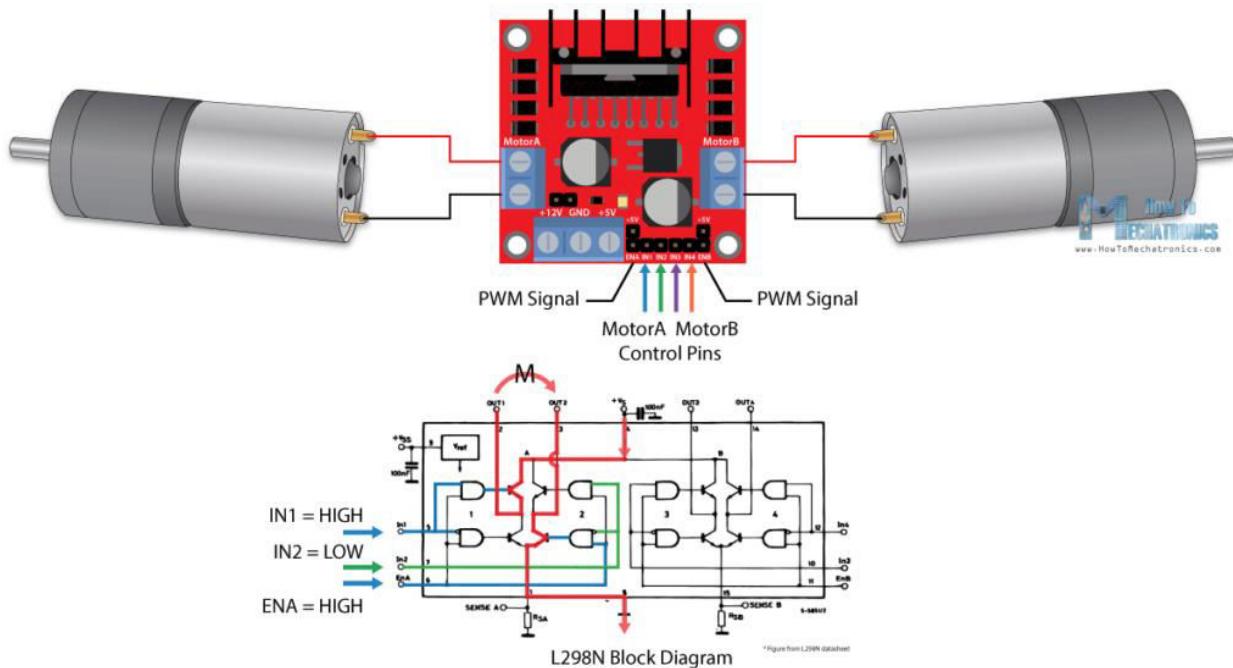
This depends on the voltage used at the motors VCC. The module have an onboard 5V regulator which is either enabled or disabled using a jumper. If the motor supply voltage is up to 12V we can enable the 5V regulator and the 5V pin can be used as output, for example for powering our Arduino board. But if the motor voltage is greater than 12V we must disconnect the jumper because those voltages will cause damage to the onboard 5V regulator. In this case the 5V pin will be used as input as we need connect it to a 5V power supply in order the IC to work properly.

We can note here that this IC makes a voltage drop of about 2V. So for example, if we use a 12V power supply, the voltage at motors terminals will be about 10V, which means that we won't be able to get the maximum speed out of our 12V DC motor.



Next are the logic control inputs. The Enable A and Enable B pins are used for enabling and controlling the speed of the motor. If a jumper is present on this pin, the motor will be enabled and work at maximum speed, and if we remove the jumper we can connect a PWM input to this

pin and in that way control the speed of the motor. If we connect this pin to a Ground the motor will be disabled.



Next, the Input 1 and Input 2 pins are used for controlling the rotation direction of the motor A, and the inputs 3 and 4 for the motor B. Using these pins we actually control the switches of the H-Bridge inside the L298N IC. If input 1 is LOW and input 2 is HIGH the motor will move forward, and vice versa, if input 1 is HIGH and input 2 is LOW the motor will move backward. In case both inputs are same, either LOW or HIGH the motor will stop. The same applies for the inputs 3 and 4 and the motor B.

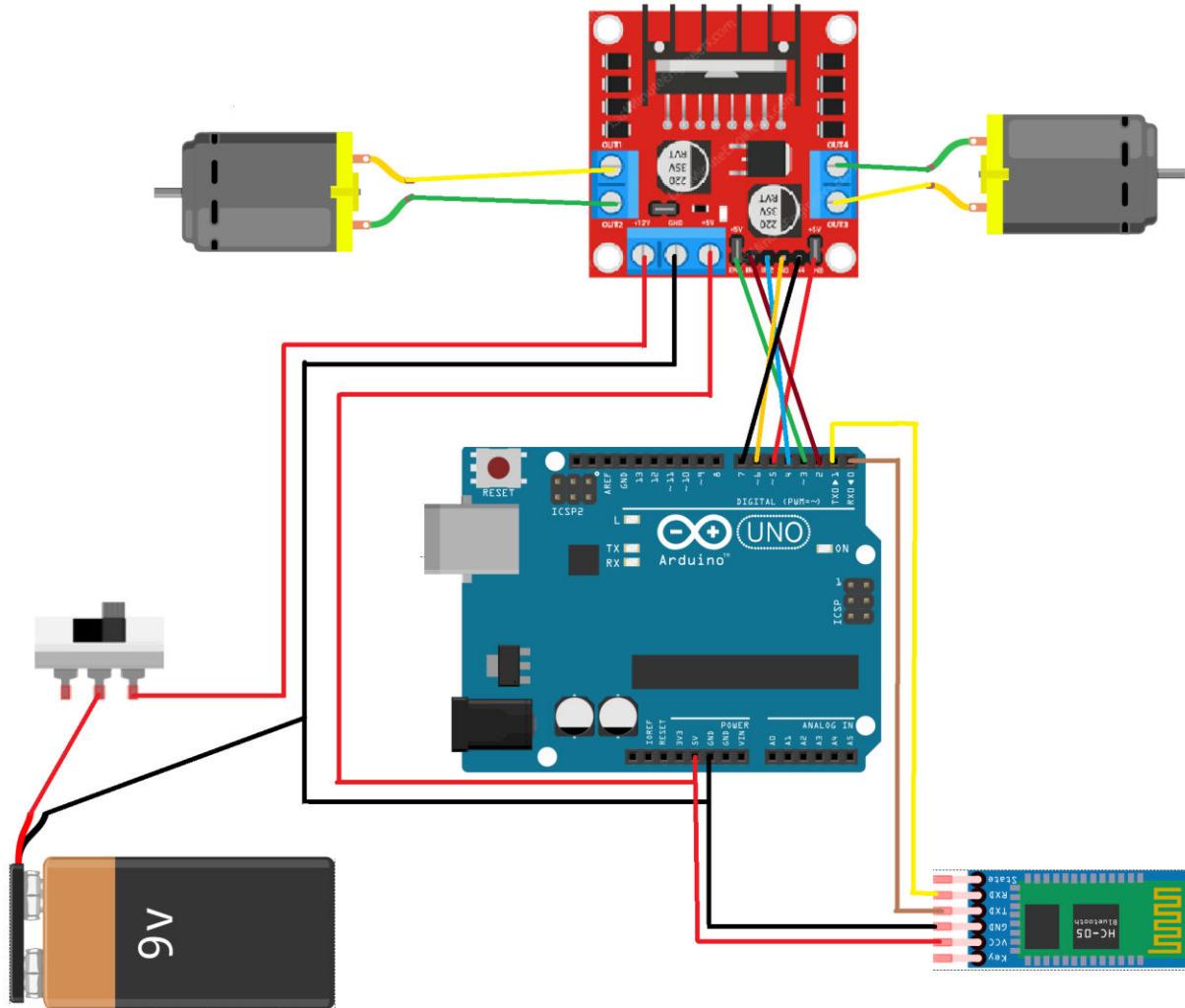
### 8.2.5 - DC Motor Control with Serial

Steps:

1. Circuit Connection
2. Learn Switch Case
3. Upload Code to Arduino
4. Control Robot From Serial Monitor

#### 8.2.5.1 - Circuit Connection

This is the complete circuit diagram of Bluetooth control robot but for learning purpose we will control our robot with Serial Monitor first. So make circuit arrangement like this diagram but disconnect the bluetooth module.



### 8.2.5.2 - Switch Case

[reference - <https://www.arduino.cc/reference/en/language/structure/control-structure/switchcase/>]

Like if statements, switch case controls the flow of programs by allowing programmers to specify different code that should be executed in various conditions. In particular, a switch statement compares the value of a variable to the values specified in case statements. When a case statement is found whose value matches that of the variable, the code in that case statement is run.

The break keyword exits the switch statement, and is typically used at the end of each case. Without a break statement, the switch statement will continue executing the following expressions ("falling-through") until a break, or the end of the switch statement is reached.

**var:** a variable whose value to compare with various cases. **Allowed data types:** int, char  
**label1, label2:** constants. **Allowed data types:** int, char

## Syntax

```
switch (var) {
 case label1:
 // statements
 break;
 case label2:
 // statements
 break;
 default:
 // statements
 break;
}
```

### 8.2.5.3 - Upload Code to Arduino

```
// Bluetooth Arduino Car using L298N Motor Driver
// Programmed by CRUX
/*
 * Connection:::::::::::
 *
 * Bluetooth Module:
 * * Bluetooth 5V -> Arduino 5V
 * * Bluetooth GND -> Arduino GND
 * * Bluetooth RX -> Arduino TX
 * * Bluetooth TX -> Arduino RX
 *
 * Motor Driver:
 * * Motor Driver 12V -> From Battery Positive (+)
 * * Motor Driver GND -> From Battery Negative(-)
 * * Motor Driver ENA -> Arduino 3
 * * Motor Driver IN1 -> Arduino 2
 * * Motor Driver IN2 -> Arduino 4
 * * Motor Driver IN3 -> Arduino 6
 * * Motor Driver IN4 -> Arduino 7
 * * Motor Driver ENB -> Arduino 5
 */
```

```

//Declear arduino desired pin number as constant integer
const int motor1Pin1 = 2; //motor 1, pin 1
const int motor1Pin2 = 4; //motor 1, pin 2
const int enablem1Pin3 = 3; //motor 1 enable pin
const int motor2Pin1 = 6; //motor 2, pin 1
const int motor2Pin2 = 7; //motor 2, pin 2
const int enablem2Pin3 = 5; //motor 2 enable pin

char serialA; //declear a character variable to store serial data from
bluetooth module

void setup() {

 Serial.begin(9600); //start serial communication with bluetooth module at
9600 baud rate

 //set all the pinmode as output
 pinMode(motor1Pin1, OUTPUT);
 pinMode(motor1Pin2, OUTPUT);
 pinMode(enablem1Pin3, OUTPUT);
 pinMode(motor2Pin1, OUTPUT);
 pinMode(motor2Pin2, OUTPUT);
 pinMode(enablem2Pin3, OUTPUT);

}

void loop() {

 //ckeck if the serial data is available
 if (Serial.available() > 0) {
 serialA = Serial.read(); //store the serial data from bluetooth
module into 'serialA' variable
 Serial.println(serialA); //print stored data from variable 'serialA'
in arduino serial monitor to check the data
 }

 //use switch case statement to control the robot using the data stored
in 'serialA' variable
 switch (serialA) {
 // forward

```

```
case 'F':
 digitalWrite(motor1Pin1, HIGH);
 digitalWrite(motor1Pin2, LOW);
 digitalWrite(motor2Pin1, LOW);
 digitalWrite(motor2Pin2, HIGH);
 digitalWrite(enablem1Pin3, HIGH);
 digitalWrite(enablem2Pin3, HIGH);
 break;

 // left
case 'L':
 digitalWrite(motor1Pin1, HIGH);
 digitalWrite(motor1Pin2, LOW);
 digitalWrite(motor2Pin1, LOW);
 digitalWrite(motor2Pin2, LOW);
 digitalWrite(enablem1Pin3, HIGH);
 digitalWrite(enablem2Pin3, LOW);
 break;

 // right
case 'R':
 digitalWrite(motor1Pin1, LOW);
 digitalWrite(motor1Pin2, LOW);
 digitalWrite(motor2Pin1, LOW);
 digitalWrite(motor2Pin2, HIGH);
 digitalWrite(enablem1Pin3, LOW);
 digitalWrite(enablem2Pin3, HIGH);
 break;

 // backward
case 'B':
 digitalWrite(motor1Pin1, LOW);
 digitalWrite(motor1Pin2, HIGH);
 digitalWrite(motor2Pin1, HIGH);
 digitalWrite(motor2Pin2, LOW);
 digitalWrite(enablem1Pin3, HIGH);
 digitalWrite(enablem2Pin3, HIGH);
 break;

 // Stop
case 'S':
 digitalWrite(motor1Pin1, LOW);
 digitalWrite(motor1Pin2, LOW);
```

```

 digitalWrite(motor2Pin1, LOW);
 digitalWrite(motor2Pin2, LOW);
 digitalWrite(enablem1Pin3, LOW);
 digitalWrite(enablem2Pin3, LOW);

}
}

```

#### **8.2.5.3 - Control Robot From Serial Monitor**

Open arduino serial monitor and make sure baud rate is **9600** (or what you have given in code) . Now if we write **F** in serial monitor and hit enter both motor will start to forward direction . Likewise **L** for Left, **R** for Right, **B** for backward and **S** for stop.

#### **College Level:**

**6 Motor Drive using 3 Motor Driver**  
**(Teacher will explain in class room )**

## **Class 9:**

- 9.1 - Function in Arduino
- 9.2 - Project : Bluetooth Control Robot Car (Part 2)
- 9.2.1 - Bluetooth Module (HC-05)
- 9.2.2 - HC-05 connection with Arduino
- 9.2.3 - Chat between bluetooth and arduino serial monitor

#### **College Level:**

- 9.3 - Arduino Preprocessor
- 9.4 - Arduino RAM & ROM

### **9.1 - Function in Arduino**

[reference - <https://startingelectronics.org/software/arduino/learn-to-program-course/01-program-structure-flow/>]

#### 9.1.1 - Overview

Function is necessary to organise our code as well as do our task easily and efficiently without writing same thing again and again. In arduino coding there are two main functions which are `setup()` and `loop()`. When we power up arduino `setup()` function executed just once than programm enter in `loop()` function . We can declare other function. Function has following rules.

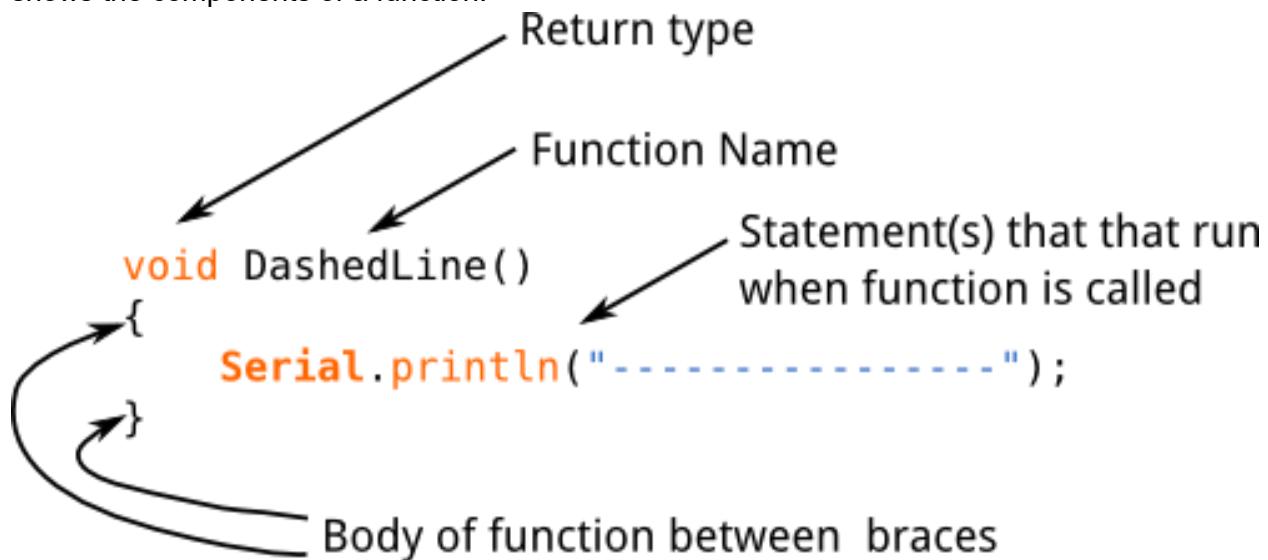
- All functions must have a unique name, **setup** is one example of a unique function name (**setup** and **loop** are special functions in Arduino programming and form part of the structure of a basic sketch).
- The function name is followed by opening and closing parentheses () that may or may not contain something.
- All functions must have a return type. Both **setup** and **loop** have a **void** return type.
- The body of a function consists of an opening and closing brace ({ and }).

### 9.1.2 - The Structure of a Function

Before a function can be used in a sketch, it must be created. The following code is an example of a function that was created to print a dashed line in the Arduino IDE.

```
void DashedLine()
{
 Serial.println("-----");
}
```

The code above that creates the function is called the function definition. The image below shows the components of a function.



### 9.1.3 - Function Name

When we create a function, it must be given a name. The naming convention for functions is the same as for variables:

- The function name can be made up of alphanumeric characters (A to Z; a to z; 0 to 9) and the underscore (\_).
- The function name may not start with a number i.e. the numbers 0 to 9.

- A function name must not be used that is the same as a language keyword or existing function.

The function name ends with parentheses (). Nothing is passed to the example function above, so the parentheses are empty. Passing values or parameters to functions will be explained later in this tutorial.

#### **9.1.4 - Return Type**

A function must have a return type. The example function does not return anything, so has a return type of void. Returning a value from a function will be explained in the next part of this course.

#### **9.1.5 - Function Body**

The function body is made up of statements placed between braces {}. The statements make up the functionality of the function (what the function will do when it is called).

When a function is used, it is said to be "called". We will look at how to call a function next.

#### **9.1.6 - Calling a Function**

To use the function that was created above, it must be called in a sketch as shown in the sketch below.

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

 DashedLine();
 Serial.println("| Program Menu |");
 DashedLine();
}

void loop() {}

void DashedLine()
{
 Serial.println("-----");
}
```

In the sketch above, the **DashedLine()** function is created at the bottom of the file and then called twice at the top of the file as shown in the image below.

```

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

 DashedLine(); ← Function is called here
 Serial.println("| Program Menu |");
 DashedLine(); ← Function is called again
}

void loop() {}

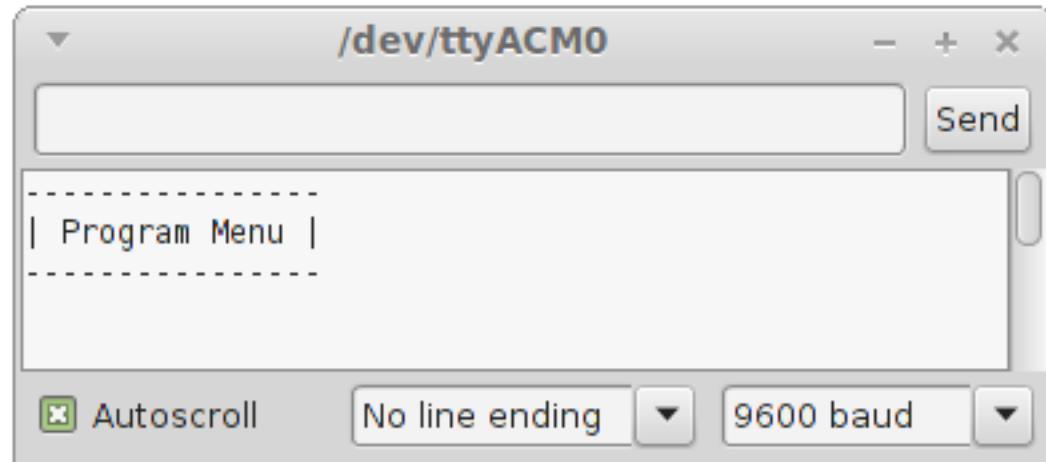
void DashedLine()
{
 Serial.println("-----"); } } Function is created here
}

```

### Calling a Function in an Arduino Sketch

To call a function, use the function name followed by opening and closing parentheses. Finally terminate the statement that calls the function with a semicolon.

Load the sketch to an Arduino and then open the terminal window. The sketch prints some text in a box as shown below.



The first time that the function is called, it prints the dashed line shown in the top of the image. Text is then written to the serial monitor window by the statement below the function call. The function is then called again to print the same dashed line that completes the box.

### **9.1.7 - Why Use Functions**

The function used in the example above is very simple, so all the benefits of using functions will not be seen immediately.

One advantage of using functions is that they avoid having to write the same code over and over again in a sketch which saves time and memory. Every time that a function is called, we are just reusing code that has been written once.

If a function needs to be modified, it only has to be done once and the modifications will take effect every place in a sketch that the function is called. If a function was not used, each place that the statements are found in a sketch to do a particular task would need to be located and modified.

Functions can be used to break a sketch up into pieces which make it more modular and easier to understand. Functions can be reused in other sketches.

### **9.1.8 -Passing a Value to a Function**

In the sketch above, the length of the line that the function prints out is fixed in the function. If we change the text that is in the box, it may not fit in the box properly. The function needs to be modified so that we can tell it what size line it must draw.

The above function can be modified to pass a value to it that will tell it how many characters long to make the line that it draws.

The modified sketch is shown below.

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

 // draw the menu box
 DashedLine(24);
 Serial.println("| Program Options Menu |");
 DashedLine(24);
}

void loop() {}

void DashedLine(int len)
```

```
{
 int i;
 // draw the line
 for (i = 0; i < len; i++) {
 Serial.print("-");
 }
 // move the cursor to the next line
 Serial.println("");
}
```

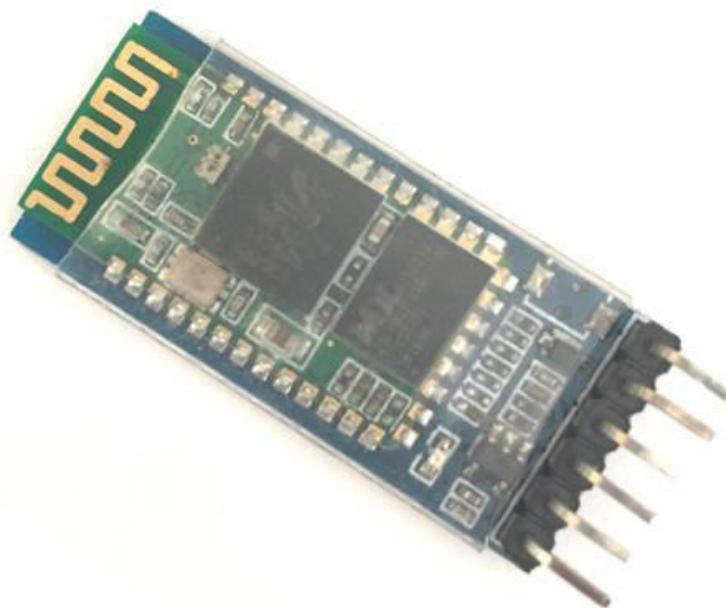
## 9.2 - Project : Bluetooth Control Robot Car (Part 2)

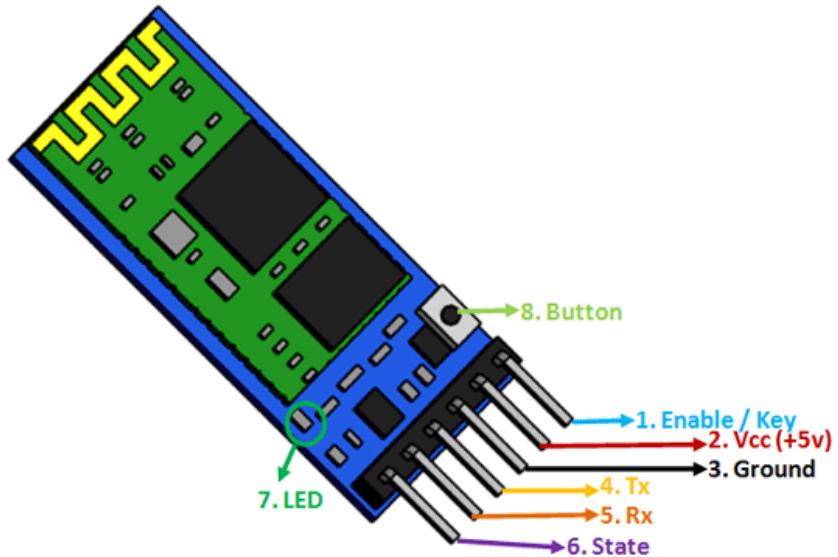
In this section we will learn about Bluetooth module and make a project by which we can chat between Arduino serial monitor and smartphone . This project will ultimately help us to understand and build complete bluetooth control robot/car. Further we will be able to build complex bluetooth control robot using this project.

### 9.2.1 - Bluetooth Module (HC-05)

[reference - <https://components101.com/wireless/hc-05-bluetooth-module>]

#### 9.2.1.1 - Diagram





### 9.2.1.2 - Pin Configuration

| Pin Number | Pin Name     | Description                                                                                                          |
|------------|--------------|----------------------------------------------------------------------------------------------------------------------|
| 1          | Enable / Key | This pin is used to toggle between Data Mode (set low) and AT command mode (set high). By default it is in Data mode |
| 2          | Vcc          | Powers the module. Connect to +5V Supply voltage                                                                     |
| 3          | Ground       | Ground pin of module, connect to system ground.                                                                      |

|   |                   |                                                                                                                                                                                                                                                                               |
|---|-------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 4 | TX<br>Transmitter | – Transmits Serial Data. Everything received via Bluetooth will be given out by this pin as serial data.                                                                                                                                                                      |
| 5 | RX<br>Receiver    | – Receive Serial Data. Every serial data given to this pin will be broadcasted via Bluetooth                                                                                                                                                                                  |
| 6 | State             | The state pin is connected to on board LED, it can be used as a feedback to check if Bluetooth is working properly.                                                                                                                                                           |
| 7 | LED               | Indicates the status of Module <ul style="list-style-type: none"> <li>• Blink once in 2 sec: Module has entered Command Mode</li> <li>• Repeated Blinking: Waiting for connection in Data Mode</li> <li>• Blink twice in 1 sec: Connection successful in Data Mode</li> </ul> |
| 8 | Button            | Used to control the Key/Enable pin to toggle between Data and command Mode                                                                                                                                                                                                    |

### 9.2.1.3 - HC-05 Default Settings

Default Bluetooth Name: “HC-05”

Default Password: 1234 or 0000

Default Communication: Slave

Default Mode: Data Mode

Data Mode Baud Rate: 9600, 8, N, 1

Command Mode Baud Rate: 38400, 8, N, 1

Default firmware: LINVOR

#### **9.2.1.4 - HC-05 Technical Specifications**

- Serial Bluetooth module for Arduino and other microcontrollers
- Operating Voltage: 4V to 6V (Typically +5V)
- Operating Current: 30mA
- Range: <100m
- Works with Serial communication (USART) and TTL compatible
- Follows IEEE 802.15.1 standardized protocol
- Uses Frequency-Hopping Spread spectrum (FHSS)
- Can operate in Master, Slave or Master/Slave mode
- Can be easily interfaced with Laptop or Mobile phones with Bluetooth
- Supported baud rate: 9600,19200,38400,57600,115200,230400,460800.

#### **9.2.1.5 - Other Bluetooth Modules**

HC-04, HC-06, HM-11, ESP32, CSR8645

#### **9.2.1.6 - Where to use HC-05 Bluetooth module**

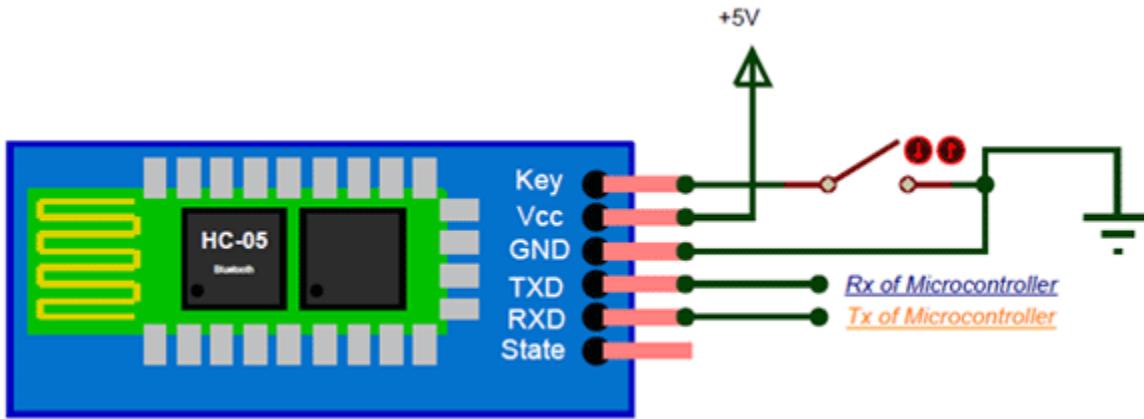
The **HC-05** is a very cool module which can add two-way (full-duplex) wireless functionality to your projects. You can use this module to communicate between two microcontrollers like Arduino or communicate with any device with Bluetooth functionality like a Phone or Laptop. There are many android applications that are already available which makes this process a lot easier. The module communicates with the help of USART at 9600 baud rate hence it is easy to interface with any microcontroller that supports USART. We can also configure the default values of the module by using the command mode. So if you looking for a Wireless module that could transfer data from your computer or mobile phone to microcontroller or vice versa then this module might be the right choice for you. However do not expect this module to transfer multimedia like photos or songs; you might have to look into the CSR8645 module for that.

#### **9.2.1.7 - How to Use the HC-05 Bluetooth module**

The **HC-05** has two operating modes, one is the Data mode in which it can send and receive data from other Bluetooth devices and the other is the AT Command mode where the default

device settings can be changed. We can operate the device in either of these two modes by using the key pin as explained in the pin description.

It is very easy to pair the HC-05 module with microcontrollers because it operates using the Serial Port Protocol (SPP). Simply power the module with +5V and connect the Rx pin of the module to the Tx of MCU and Tx pin of module to Rx of MCU as shown in the figure below



During power up the key pin can be grounded to enter into Command mode, if left free it will by default enter into the data mode. As soon as the module is powered you should be able to discover the Bluetooth device as "HC-05" then connect with it using the default password 1234 and start communicating with it. The name password and other default parameters can be changed by entering into the command mode. see details here [https://www.itead.cc/wiki/Serial\\_Port\\_Bluetooth\\_Module\\_\(Master/Slave\) : HC-05](https://www.itead.cc/wiki/Serial_Port_Bluetooth_Module_(Master/Slave) : HC-05)

### 9.2.1.8 - Applications

1. Wireless communication between two microcontrollers
2. Communicate with Laptop, Desktops and mobile phones
3. Data Logging application
4. Consumer applications
5. Wireless Robots
6. Home Automation

## 9.2.2 - HC-05 connection with Arduino via Software Serial

### 9.2.2.1 - Software Serial

[reference - <https://www.arduino.cc/en/Reference/SoftwareSerial>]

The SoftwareSerial library has been developed to allow serial communication on other digital pins of the Arduino, using software to replicate the functionality (hence the name "SoftwareSerial"). It is possible to have multiple software serial ports with speeds up to 115200 bps. A parameter enables inverted signaling for devices which require that protocol.

### Limitations

The library has the following known limitations:

1. If using multiple software serial ports, only one can receive data at a time.

2. Not all pins on the Mega and Mega 2560 support change interrupts, so only the following can be used for RX: 10, 11, 12, 13, 14, 15, 50, 51, 52, 53, A8 (62), A9 (63), A10 (64), A11 (65), A12 (66), A13 (67), A14 (68), A15 (69).

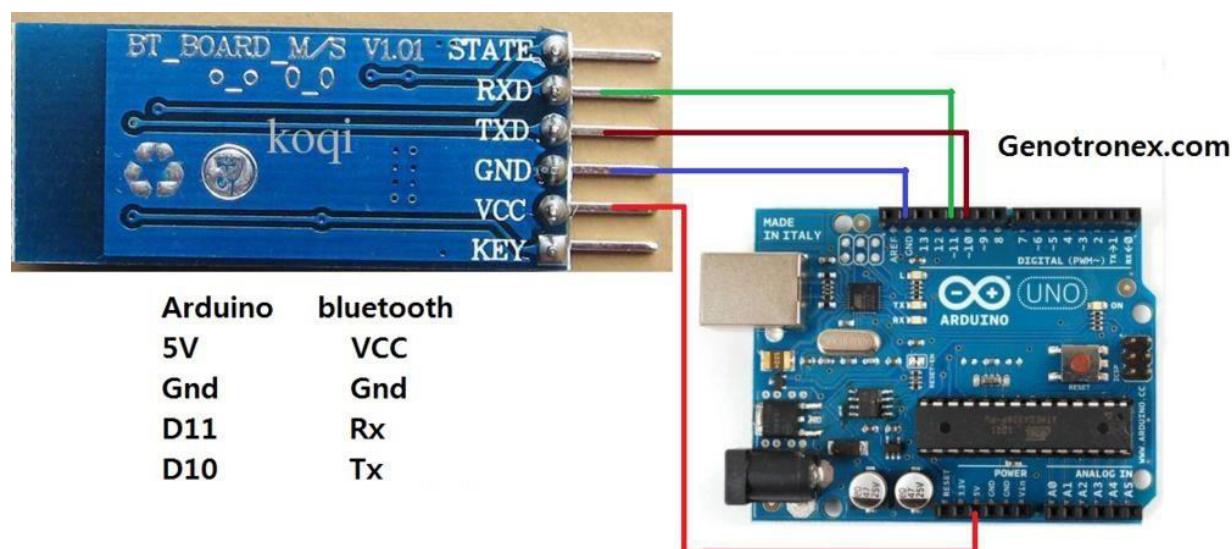
3. Not all pins on the Leonardo and Micro support change interrupts, so only the following can be used for RX: 8, 9, 10, 11, 14 (MISO), 15 (SCK), 16 (MOSI).

4. On Arduino or Genuino 101 the current maximum RX speed is 57600bps

5. On Arduino or Genuino 101 RX doesn't work on Pin 13

If your project requires simultaneous data flows, see Paul Stoffregen's [AltSoftSerial library](#). AltSoftSerial overcomes a number of other issues with the core SoftwareSerial, but has its own limitations. Refer to the [AltSoftSerial site](#) for more information.

### 9.2.2.2 - Software Serial HC-05 Connection



### 9.2.3 - Chat between bluetooth and arduino serial monitor

**Step 1:** Upload This Code

```
#include <SoftwareSerial.h>

SoftwareSerial mySerial(10, 11); // RX, TX

void setup()
{
 // Open serial communications and wait for port to open:
 Serial.begin(9600);
 mySerial.begin(9600);
 mySerial.println("Serial Communication Started");
}
```

```

void loop() // run over and over
{
 if (mySerial.available())
 Serial.write(mySerial.read());
 if (Serial.available())
 mySerial.write(Serial.read());
}

```

**Step 2 :**

Download Android App “Bluetooth Terminal” by Developer/Company “Qwerty”  
link: <https://play.google.com/store/apps/details?id=Qwerty.BluetoothTerminal>



Now pair your smartphone with HC-05. Default password would be 1234 or 0000 than open this app and connect with HC-05 . Write anything and send. It should appear in arduino serial monitor. Now you can chat between Bluetooth Terminal app and arduino serial monitor.

College Level:

## 9.3 - Arduino Preprocessor

[reference - <https://www.deviceplus.com/how-tos/arduino-guide/arduino-preprocessor-directives-tutorial/>]

### 9.3.1 - Overview

While compiling arduino code, several things happened:

1. Arduino IDE performed something called “syntactic check”, to make sure what you wrote is actual C/C++ source code. This is the point at which the compilation will halt in case you misspelled a function or forgot a semicolon.
2. After syntax check, Arduino IDE starts another program called *preprocessor*. This is a very simple program that doesn’t really care if the file is a C/C++ source code. Since we will talk more about this step later on, we’ll just assume that the result is a file called “extended source code”, which is just still just a text file.
3. Then, the extended source code was handed over to another program called *compiler*. The compiler (in the Arduino IDE case it’s *avr-gcc*) takes in text source and produces assembly file. This a lower programming language that is still human readable, but much closer to machine code – it’s basically just processor-specific instructions. This part is the reason why you have to select the correct Arduino board before you start compiling a sketch – different boards have different processors, which in turn have different instruction sets.
4. The next program that processed your sketch is called *assembler*. It generates an “object file”. This is mostly machine code, but it can also contain “references” to objects in other object files. This allows Arduino IDE to “pre-compile” some libraries that will always be used when writing Arduino sketch, making the whole process a lot faster.
5. The final stage is called linking and it’s done by yet another program called – unsurprisingly – *linker*. The linker takes the object file and adds everything that’s missing to make it into an executable – mainly symbols from other object files. After this, the program is completely converted into machine code and can be programmed to the board.

In the above text, I mentioned that preprocessor is essentially very simple: it just takes in text input, searches from some keywords, does some operations according to what it finds, and then outputs a different text. Even though it’s very simple, it’s also extremely powerful, because it allows you to do things that would otherwise be very complicated – if not impossible – in plain C/C++ language.

The preprocessor works by searching for lines that start with the hash sign (#) and have some text after it. This is called *preprocessor directive* and it's a sort of "command" for the preprocessor. The full list of all supported directives with detailed documentation can be found here:

<https://gcc.gnu.org/onlinedocs/cpp/Index-of-Directives.html#Index-of-Directives>

In the following text, I will focus mainly on `#include`, `#define` and conditional directives, since these are the most useful on Arduino, but if you want to know more about some more "exotic" directives, like `#assert` or `#pragma`, this is the place to get official information.

This is probably the best-known preprocessor directive, not only amongst Arduino enthusiasts, but in C/C++ programming in general. The reason is simple: it's used to include libraries. But how exactly does it happen? The exact syntax is as follows:

```
#include <file>
```

or

```
#include "file"
```

The difference between the two is subtle and mainly comes down to where exactly the preprocessor searches for the *file*. In the first case, the preprocessor searches only in directories specified by the IDE. In the second case, the preprocessor first looks through the folder containing the source, and only if the *file* isn't there, it moves on the same directories it would search in the first case. Since the folder containing libraries is specified in Arduino IDE, there's no major difference between the two when including a library.

When the preprocessor finds the file, it simply copy-pastes the contents into the source code in place of the `#include` directive. However, if no such file can be found in any of the directories, it will raise a fatal error and stop the compilation.

### 9.3.2 - Practical Implementation(`#define` directive)

Now let's think a situation where we want to active and deactivate serial output . We can do it using define directive. anything defined by `#defined` directive is called *macro*. In this code if we comment out //define DEBUG we will not see anything in serial monitor.

```
#define DEBUG
void setup()
{
 #ifdef DEBUG
```

```

Serial.begin(9600);
#endif
}

void loop()
{
#ifndef DEBUG
 Serial.println("Debug mode is On");
#endif

}

```

## 9.4 - Arduino Memory

[reference - <https://learn.adafruit.com/memories-of-an-arduino/arduino-memories>]

### 9.4.1 - Overview

There are 3 types of memory in an Arduino:

**Flash or Program Memory**  
**SRAM**  
**EEPROM**

#### 9.4.1.1 - Flash Memory

Flash memory is used to store your program image and any initialized data. You can execute program code from flash, but you can't modify data in flash memory from your executing code. To modify the data, it must first be copied into SRAM

Flash memory is the same technology used for thumb-drives and SD cards. It is non-volatile, so your program will still be there when the system is powered off.

Flash memory has a finite lifetime of about 100,000 write cycles. So if you upload 10 programs a day, every day for the next 27 years, you might wear it out.

#### 9.4.1.2 -SRAM

SRAM or Static Random Access Memory, can be read and written from your executing program. SRAM memory is used for several purposes by a running program:

- **Static Data** - This is a block of reserved space in SRAM for all the global and static variables from your program. For variables with initial values, the runtime system copies the initial value from Flash when the program starts.
- **Heap** - The heap is for dynamically allocated data items. The heap grows from the top of the static data area up as data items are allocated.
- **Stack** - The stack is for local variables and for maintaining a record of interrupts and function calls. The stack grows from the top of memory down towards the heap. Every

interrupt, function call and/or local variable allocation causes the stack to grow. Returning from an interrupt or function call will reclaim all stack space used by that interrupt or function.

## Optimising SRAM

See details here : <https://learn.adafruit.com/memories-of-an-arduino/optimizing-sram>

- Remove Unused Variables
- Use F()
- Reserve() your strings
- Move constant data to PROGMEM
- Reduces Buffer Size
- Reduce Oversized Variables
- Prefer local to global allocation

### 9.4.1.3 -EEPROM

[reference - <https://www.makeuseof.com/tag/use-arduino-eeprom-save-data-power-cycles/>  
<https://www.arduino.cc/en/Reference/EEPROM>]

EEPROM is another form of non-volatile memory that can be read or written from your executing program. It can only be read byte-by-byte, so it can be a little awkward to use. It is also slower than SRAM and has a finite lifetime of about 100,000 write cycles (you can read it as many times as you want). As EEPROM value can survive in power down situation it's very useful for many projects. Let's say you want to make a digital fan regulator which can vary fan speed in 5 steps. Now if you don't store its speed steps in EEPROM in case of power failure it will not resume from its previous state rather it will start from its default state which is not convenient.

#### EEPROM Read and Write

Now that the theory is out the way, let's look at how to read and write some data! First, include the library (this comes with the Arduino IDE):

```
#include <EEPROM.h>
```

Now write some data:

```
EEPROM.write(0, 12);
```

This writes the number 12 to EEPROM location 0. Each write takes 3.3 milliseconds (ms, 1000ms = 1 second). Notice how you cannot write letters (char), only the numbers from zero to 255 are allowed. This is why EEPROM is ideal for settings or high scores, but not so good for player names or words. It is possible to store text using this method (you could map each letter of the alphabet to a number), however you will need to have multiple memory locations — one location for each letter.

**Here's how you read that data:**

```
EEPROM.read(0);
```

Zero is the address you wrote to previously. If you have not written to an address before, it will return the maximum value (255).

There are some slightly more useful methods available. Say you wanted to store a decimal place or string:

```
EEPROM.put(2, "12.67");
```

This writes the data to multiple locations — something that would be easy to write yourself, but handy none the less. You will still need to keep track of how many locations this has written to, so you don't accidentally overwrite your data! You have to use the get method to retrieve this data again:

```
float f = 0.00f;
EEPROM.get(2, f);
```

The value from get is stored into the float f variable. Notice how this is initialized with 0.00f as the value. The f lets the compiler know that you might want to store a large number in this variable, so it sets up some additional configurations during compilation.

**See this video :** <https://www.youtube.com/watch?v=jQoqpDTtlx4>

### **Simple Example of EEPROM :**

Now we will write a value(e.g 65) in a EEPROM address(e.g 100) then we will read it. To write a value in EEPROM upload this code

```
#include <EEPROM.h>

void setup()
{
 Serial.begin(9600);
 EEPROM.write(100, 65); //Write 65 in EEPROM Address 100
 delay(10); //give some time to write
}

void loop()
{}
```

Power off the arduino to check EEPROM . Now upload following code to read the value from same address .

```
#include <EEPROM.h>

void setup()
{
 Serial.begin(9600);
 Serial.println(EEPROM.read(100));
 delay(100);
}

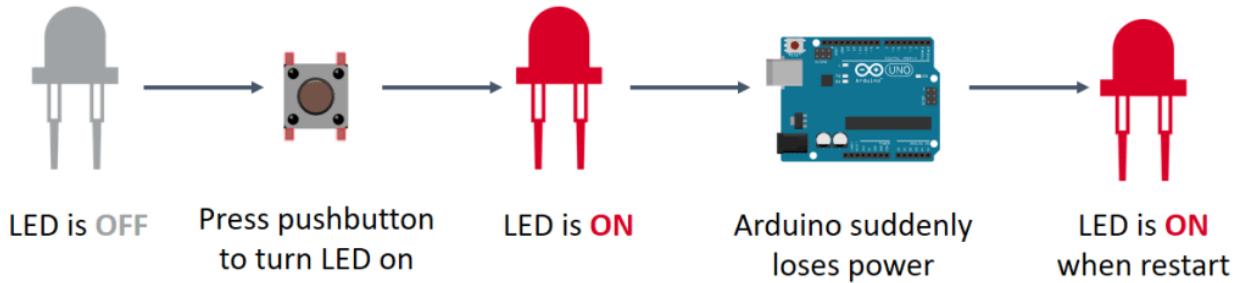
void loop()
{
}
```

See this link for better understanding <https://randomnerdtutorials.com/arduino-eeprom-explained-remember-last-led-state/>

In this example, we're going to show you how to make the Arduino remember the stored LED state, even when we reset the Arduino or the power goes off.

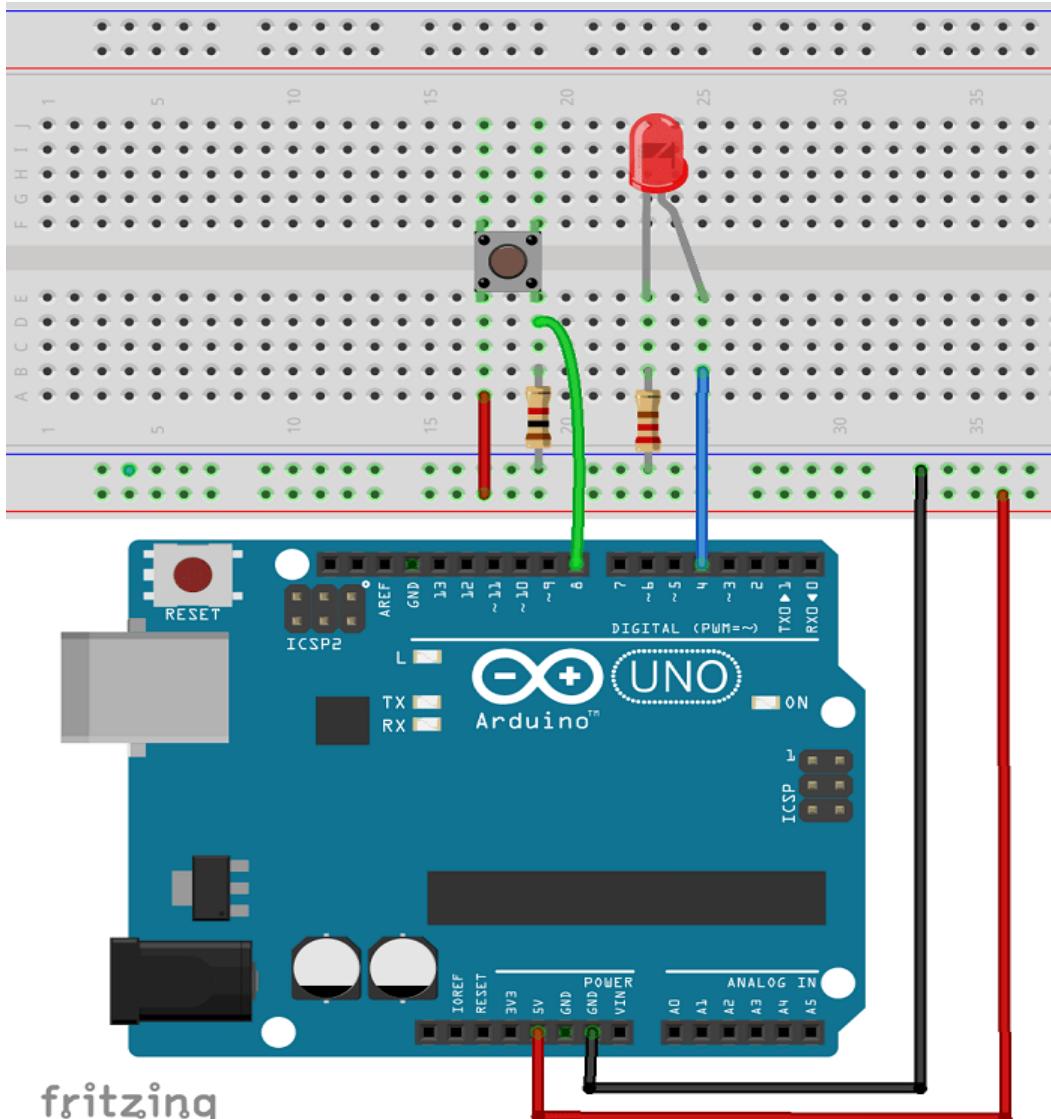
The following figure shows what we're going to exemplify:

### Save LED state on EEPROM



### Schematics

Here's the circuit schematics for this project. This is just a pushbutton that will turn an LED on and off.



**Code :**

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

```
#include <EEPROM.h>

const int buttonPin = 8; // pushbutton pin
const int ledPin = 4; // LED pin

int ledState; // variable to hold the led state
int buttonState; // the current reading from the input pin
int lastButtonState = LOW; // the previous reading from the input pin

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

void setup() {
 // set input and output
 pinMode(buttonPin, INPUT);
 pinMode(ledPin, OUTPUT);

 // set initial LED state
 digitalWrite(ledPin, ledState);

 // initialize serial monitor
 Serial.begin (9600);

 //check stored LED state on EEPROM using function defined at the end
 //of the code
 checkLedState();
}

void loop() {
 // read the state of the switch into a local variable
 int reading = digitalRead(buttonPin);

 if(reading != lastButtonState) {
 // reset the debouncing timer
 lastDebounceTime = millis();
 }
}
```

```
if((millis() - lastDebounceTime) > debounceDelay) {
 // whatever the reading is at, it's been there for longer
 // than the debounce delay, so take it as the actual current state:

 // if the button state has changed:
 if(reading != buttonState) {
 buttonState = reading;

 // only toggle the LED if the new button state is HIGH
 if(buttonState == HIGH) {
 ledState = !ledState;
 }
 }
}

// set the LED state
digitalWrite(ledPin, ledState);
// save the current LED state in the EEPROM
EEPROM.update(0, ledState);
// save the reading. Next time through the loop,
// it'll be the lastButtonState
lastButtonState = reading;
}

// Prints and updates the LED state
// when the Arduino board restarts or powers up
void checkLedState() {
 Serial.println("LED status after restart: ");
 ledState = EEPROM.read(0);
 if(ledState == 1) {
 Serial.println ("ON");
 digitalWrite(ledPin, HIGH);
 }
 if(ledState == 0) {
 Serial.println ("OFF");
 digitalWrite(ledPin, LOW);
 }
}
```

## Class 10 :

10.1 - Project : Bluetooth Control Robot Car (Part 3)

10.1.1 - Connect All Together

10.1.2 - Upload Code

10.1.3 - Download App, Connect & Test

10.1.4 - Tools: Glue Gun, PVC, Soldering Iron

10.1.5 - Assemble All & Drive

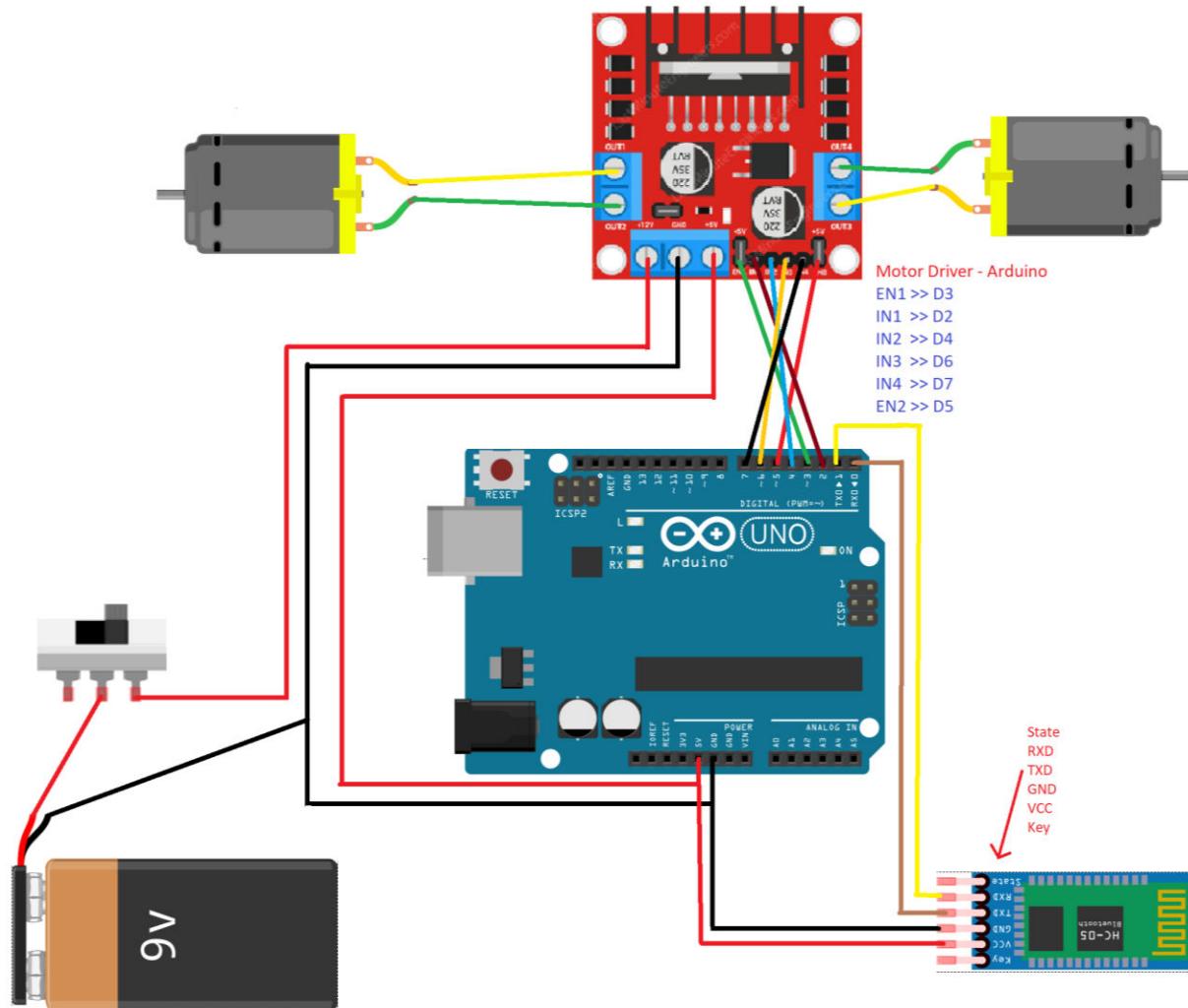
### 10.1 - Project : Bluetooth Control Robot Car (Part 3)

#### **10.1.1 - Connect All Together**

Connect all together like this diagram.

```
/*
 * Connection:::::::::::
 *
 * Bluetooth Module:
 * Bluetooth 5V -> Arduino 5V
 * Bluetooth GND -> Arduino GND
 * Bluetooth RX -> Arduino TX
 * Bluetooth TX -> Arduino RX
 *
 * Motor Driver:
 * Motor Driver 12V -> From Battery Positive (+)
 * Motor Driver GND -> From Battery Negative(-)
 * Motor Driver ENA -> Arduino 3
 * Motor Driver IN1 -> Arduino 2
 * Motor Driver IN2 -> Arduino 4
 * Motor Driver IN3 -> Arduino 6
 * Motor Driver IN4 -> Arduino 7
 * Motor Driver ENB -> Arduino 5
 */

```



### 10.1.2 - Upload Code

```
//Declear arduino desired pin number as constant integer
const int motor1Pin1 = 2; //motor 1, pin 1
const int motor1Pin2 = 4; //motor 1, pin 2
const int enable1Pin3 = 3; //motor 1 enable pin
const int motor2Pin1 = 6; //motor 2, pin 1
const int motor2Pin2 = 7; //motor 2, pin 2
const int enable2Pin3 = 5; //motor 2 enable pin

char serialA; //declear a character variable to store serial data from
bluetooth module
void setup() {
```

```

Serial.begin(9600); //start serial communication with bluetooth module at
9600 baud rate

//set all the pinmode as output
pinMode(motor1Pin1, OUTPUT);
pinMode(motor1Pin2, OUTPUT);
pinMode(enablem1Pin3, OUTPUT);
pinMode(motor2Pin1, OUTPUT);
pinMode(motor2Pin2, OUTPUT);
pinMode(enablem2Pin3, OUTPUT);

}

void loop() {

//ckeck if the serial data is available
if (Serial.available() > 0) {
 serialA = Serial.read(); //store the serial data from bluetooth
module into 'serialA' variable
 Serial.println(serialA); //print stored data from variable 'serialA'
in arduino serial monitor to check the data
}

//use switch case statement to control the robot using the data stored
in 'serialA' variable
switch (serialA) {
 // forward
case 'F':
 digitalWrite(motor1Pin1, HIGH);
 digitalWrite(motor1Pin2, LOW);
 digitalWrite(motor2Pin1, LOW);
 digitalWrite(motor2Pin2, HIGH);
 digitalWrite(enablem1Pin3, HIGH);
 digitalWrite(enablem2Pin3, HIGH);
 break;

 // left
case 'L':
 digitalWrite(motor1Pin1, HIGH);
 digitalWrite(motor1Pin2, LOW);
 digitalWrite(motor2Pin1, LOW);
 digitalWrite(motor2Pin2, LOW);
 digitalWrite(enablem1Pin3, HIGH);
}
}

```

```

digitalWrite(enablem2Pin3, LOW);
break;

 // right
case 'R':
 digitalWrite(motor1Pin1, LOW);
 digitalWrite(motor1Pin2, LOW);
 digitalWrite(motor2Pin1, LOW);
 digitalWrite(motor2Pin2, HIGH);
 digitalWrite(enablem1Pin3, LOW);
 digitalWrite(enablem2Pin3, HIGH);
 break;

 // backward
case 'B':
 digitalWrite(motor1Pin1, LOW);
 digitalWrite(motor1Pin2, HIGH);
 digitalWrite(motor2Pin1, HIGH);
 digitalWrite(motor2Pin2, LOW);
 digitalWrite(enablem1Pin3, HIGH);
 digitalWrite(enablem2Pin3, HIGH);
 break;

 // Stop
case 'S':
 digitalWrite(motor1Pin1, LOW);
 digitalWrite(motor1Pin2, LOW);
 digitalWrite(motor2Pin1, LOW);
 digitalWrite(motor2Pin2, LOW);
 digitalWrite(enablem1Pin3, LOW);
 digitalWrite(enablem2Pin3, LOW);

}

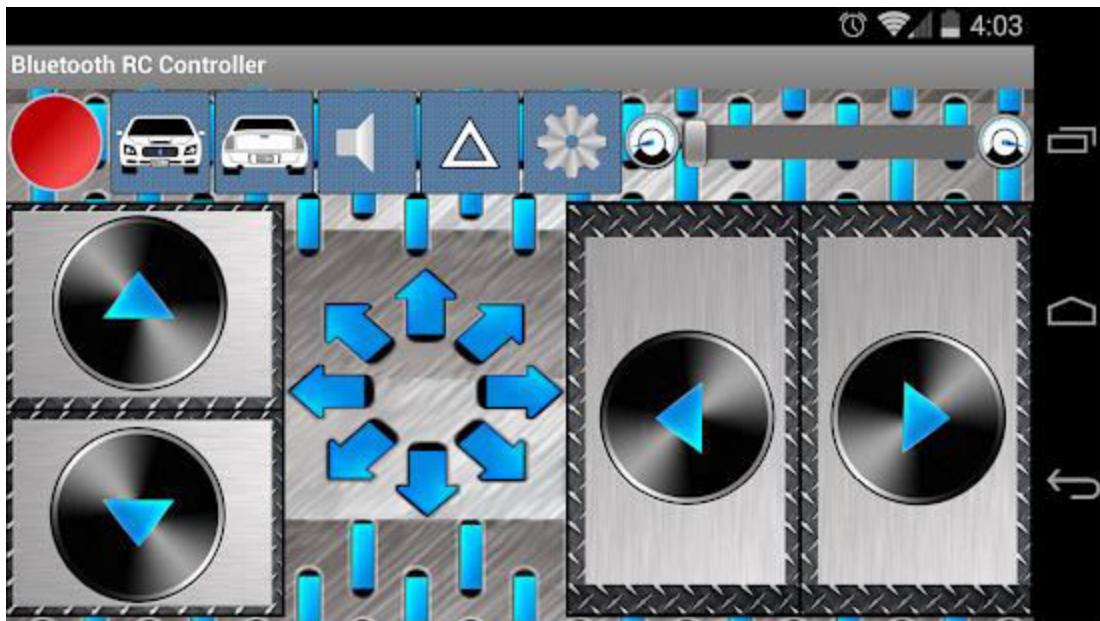
}

```

### 10.1.3 - Download App, Connect & Test

Download “Arduino Bluetooth RC Car” App from this link :

<https://play.google.com/store/apps/details?id=braulio.calle.bluetoothRCcontroller>



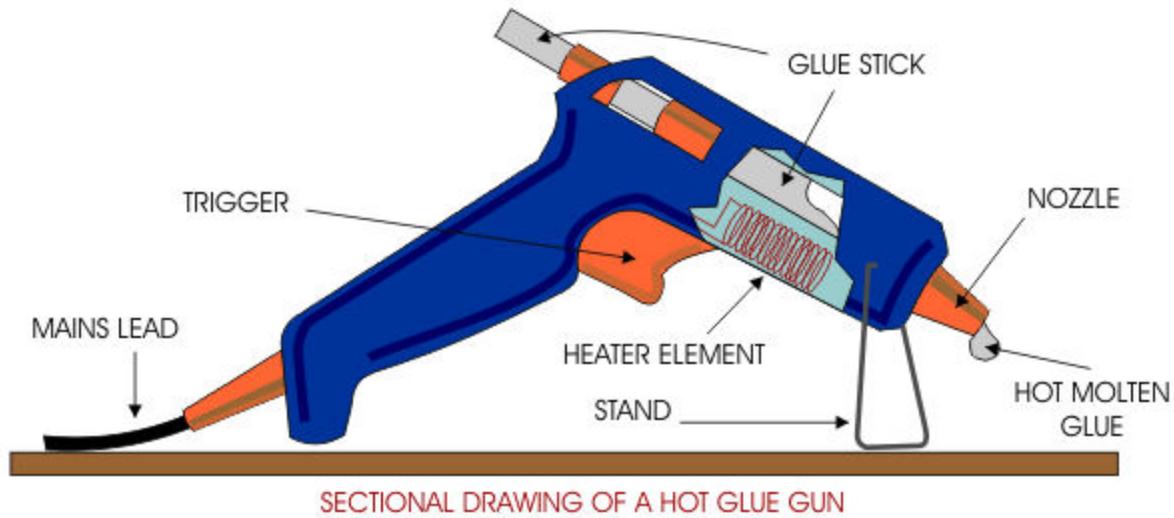
than pair and connect with your robot. Now you should be able to control your robot.

#### 10.1.4 - Glue Gun, Soldering Iron, PVC

Now we will learn about some tools and material which is common for elementary robot building.

##### 10.1.4.1 - Glue Gun

First insert the glue stick and power up the glue gun. Give the glue gun a couple minutes to soften the glue. Once it's been sufficiently melted, the glue will ooze out when you pull the trigger. For most glue guns, the heating process will take around two minutes. Larger and industrial-grade glue guns may require up to five minutes to heat the glue enough to make it easily dispensable.



#### 10.1.4.2 - Soldering Iron

See here <https://www.weller-tools.com/how-to-use-soldering-iron/>

#### 10.1.4.3 - PVC Sheet

PVC Sheet can be used to build basic robot structure. It's easy to cut and easy to join by applying super glue or glue gun.



#### 10.1.5 - Assemble All & Drive

Use PVC, Glue gun and soldering iron to make a final structure. Before applying power you should check every connection. If everything is ready than drive and learn.

## Class 11:

11.1 - Ohm's Law & Power Details

11.2 - Project : Sonar Sensor Integration and Show data in Serial & LCD (Part 1)

### College Level:

11.3 - Interface Laser Distance Measuring Sensor

## 11.1 - Ohm's Law & Power

[reference - [https://www.electronics-tutorials.ws/dccircuits/dcp\\_2.html](https://www.electronics-tutorials.ws/dccircuits/dcp_2.html)  
<https://learn.sparkfun.com/tutorials/voltage-current-resistance-and-ohms-law/all>]

The relationship between Voltage, Current and Resistance in any DC electrical circuit was firstly discovered by the German physicist Georg Ohm.

Georg Ohm found that, at a constant temperature, the electrical current flowing through a fixed linear resistance is directly proportional to the voltage applied across it, and also inversely proportional to the resistance. This relationship between the Voltage, Current and Resistance forms the basis of **Ohms Law** and is shown below.

### **Ohms Law Relationship**

$$\text{Current, } (I) = \frac{\text{Voltage, } (V)}{\text{Resistance, } (R)} \text{ in Amperes, } (A)$$

By knowing any two values of the Voltage, Current or Resistance quantities we can use Ohms Law to find the third missing value. Ohms Law is used extensively in electronics formulas and calculations so it is “very important to understand and accurately remember these formulas”.

#### **To find the Voltage, ( V )**

$$[ V = I \times R ] \quad V \text{ (volts)} = I \text{ (amps)} \times R \text{ (\Omega)}$$

#### **To find the Current, ( I )**

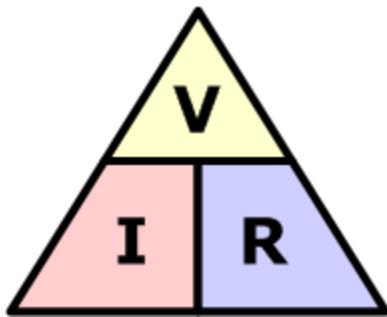
$$[ I = V \div R ] \quad I \text{ (amps)} = V \text{ (volts)} \div R \text{ (\Omega)}$$

#### **To find the Resistance, ( R )**

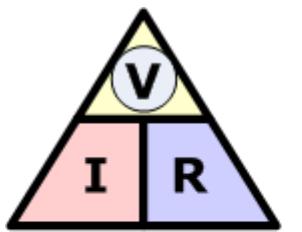
$$[ R = V \div I ] \quad R \text{ (\Omega)} = V \text{ (volts)} \div I \text{ (amps)}$$

It is sometimes easier to remember this Ohms law relationship by using pictures. Here the three quantities of V, I and R have been superimposed into a triangle (affectionately called the Ohms Law Triangle) giving voltage at the top with current and resistance below. This arrangement represents the actual position of each quantity within the Ohms law formulas.

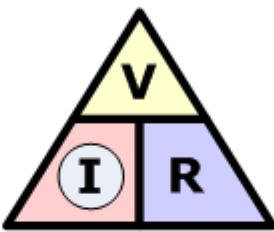
### **Ohms Law Triangle**



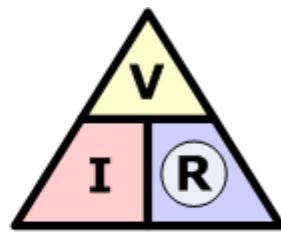
Transposing the standard Ohm's Law equation above will give us the following combinations of the same equation:



$$\textcircled{V} = I \times R$$



$$\textcircled{I} = \frac{V}{R}$$



$$\textcircled{R} = \frac{V}{I}$$

Then by using Ohms Law we can see that a voltage of 1V applied to a resistor of 1Ω will cause a current of 1A to flow and the greater the resistance value, the less current that will flow for a given applied voltage. Any Electrical device or component that obeys "Ohms Law" that is, the current flowing through it is proportional to the voltage across it ( $I \propto V$ ), such as resistors or cables, are said to be "Ohmic" in nature, and devices that do not, such as transistors or diodes, are said to be "Non-ohmic" devices.

### **Electrical Power in Circuits**

Electrical Power, ( P ) in a circuit is the rate at which energy is absorbed or produced within a circuit. A source of energy such as a voltage will produce or deliver power while the connected load absorbs it. Light bulbs and heaters for example, absorb electrical power and convert it into either heat, or light, or both. The higher their value or rating in watts the more electrical power they are likely to consume.

The quantity symbol for power is P and is the product of voltage multiplied by the current with the unit of measurement being the Watt ( W ). Prefixes are used to denote the various multiples or sub-multiples of a watt, such as: milliwatts (mW =  $10^{-3}W$ ) or kilowatts (kW =  $10^3W$ ).

Then by using Ohm's law and substituting for the values of V, I and R the formula for electrical power can be found as:

### **To find the Power (P)**

$$[ P = V \times I ] \quad P (\text{watts}) = V (\text{volts}) \times I (\text{amps})$$

Also:

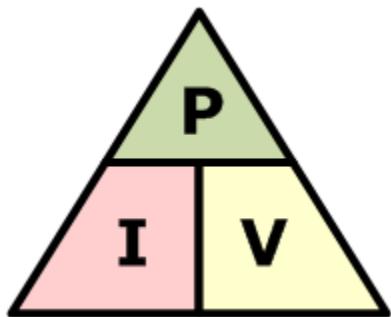
$$[ P = V^2 \div R ] \quad P (\text{watts}) = V^2 (\text{volts}) \div R (\Omega)$$

Also:

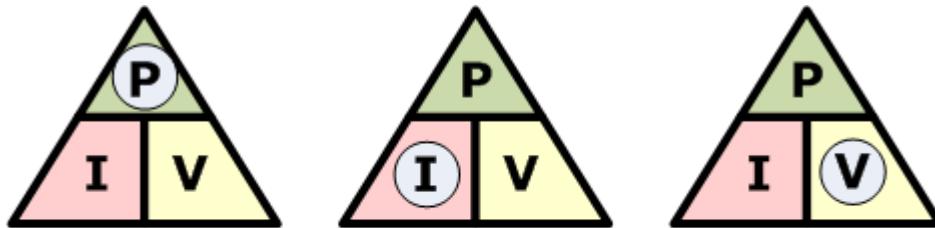
$$[ P = I^2 \times R ] \quad P (\text{watts}) = I^2 (\text{amps}) \times R (\Omega)$$

Again, the three quantities have been superimposed into a triangle this time called a Power Triangle with power at the top and current and voltage at the bottom. Again, this arrangement represents the actual position of each quantity within the Ohms law power formulas.

### The Power Triangle



and again, transposing the basic Ohms Law equation above for power gives us the following combinations of the same equation to find the various individual quantities:



$$\textcircled{P} = I \times V$$

$$\textcircled{I} = \frac{P}{V}$$

$$\textcircled{V} = \frac{P}{I}$$

So we can see that there are three possible formulas for calculating electrical power in a circuit. If the calculated power is positive, (+P) in value for any formula the component absorbs the power, that is it is consuming or using power. But if the calculated power is negative, (-P) in value the component produces or generates power, in other words it is a source of electrical power such as batteries and generators.

### Electrical Power Rating

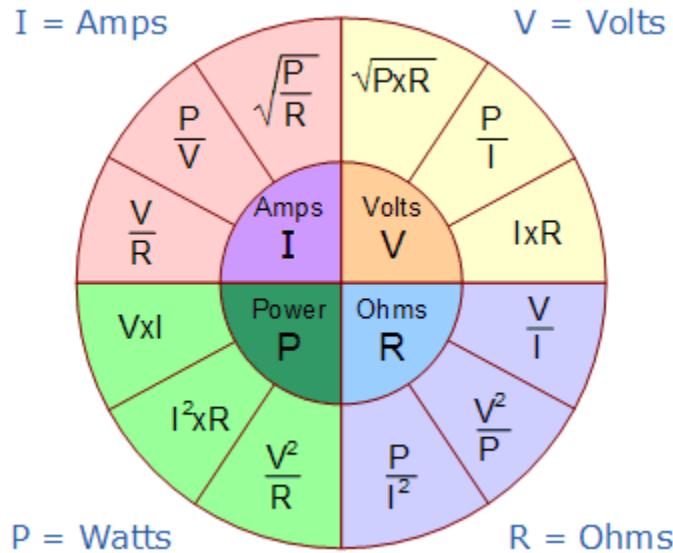
Electrical components are given a “power rating” in watts that indicates the maximum rate at which the component converts the electrical power into other forms of energy such as heat, light or motion. For example, a 1/4W resistor, a 100W light bulb etc.

Electrical devices convert one form of power into another. So for example, an electrical motor will convert electrical energy into a mechanical force, while an electrical generator converts mechanical force into electrical energy. A light bulb converts electrical energy into both light and heat.

Also, we now know that the unit of power is the *WATT*, but some electrical devices such as electric motors have a power rating in the old measurement of “Horsepower” or hp. The relationship between horsepower and watts is given as: 1hp = 746W. So for example, a two-horsepower motor has a rating of 1492W, (2 x 746) or 1.5kW.

### Ohm's Law Pie Chart

To help us understand the the relationship between the various values a little further, we can take all of the Ohm's Law equations from above for finding Voltage, Current, Resistance and of course Power and condense them into a simple Ohm's Law pie chart for use in AC and DC circuits and calculations as shown.



As well as using the *Ohm's Law Pie Chart* shown above, we can also put the individual Ohm's Law equations into a simple matrix table as shown for easy reference when calculating an unknown value.

### Ohms Law Matrix Table

| Ohms Law Formulas    |                     |                          |                         |                     |
|----------------------|---------------------|--------------------------|-------------------------|---------------------|
| Known Values         | Resistance (R)      | Current (I)              | Voltage (V)             | Power (P)           |
| Current & Resistance | ---                 | ---                      | $V = I \times R$        | $P = I^2 \times R$  |
| Voltage & Current    | $R = \frac{V}{I}$   | ---                      | ---                     | $P = V \times I$    |
| Power & Current      | $R = \frac{P}{I^2}$ | ---                      | $V = \frac{P}{I}$       | ---                 |
| Voltage & Resistance | ---                 | $I = \frac{V}{R}$        | ---                     | $P = \frac{V^2}{R}$ |
| Power & Resistance   | ---                 | $I = \sqrt{\frac{P}{R}}$ | $V = \sqrt{P \times R}$ | ---                 |
| Voltage & Power      | $R = \frac{V^2}{P}$ | $I = \frac{P}{V}$        | ---                     | ---                 |

## 11.2 - Project : Sonar Sensor Integration and Show data in Serial & LCD (Part 1)

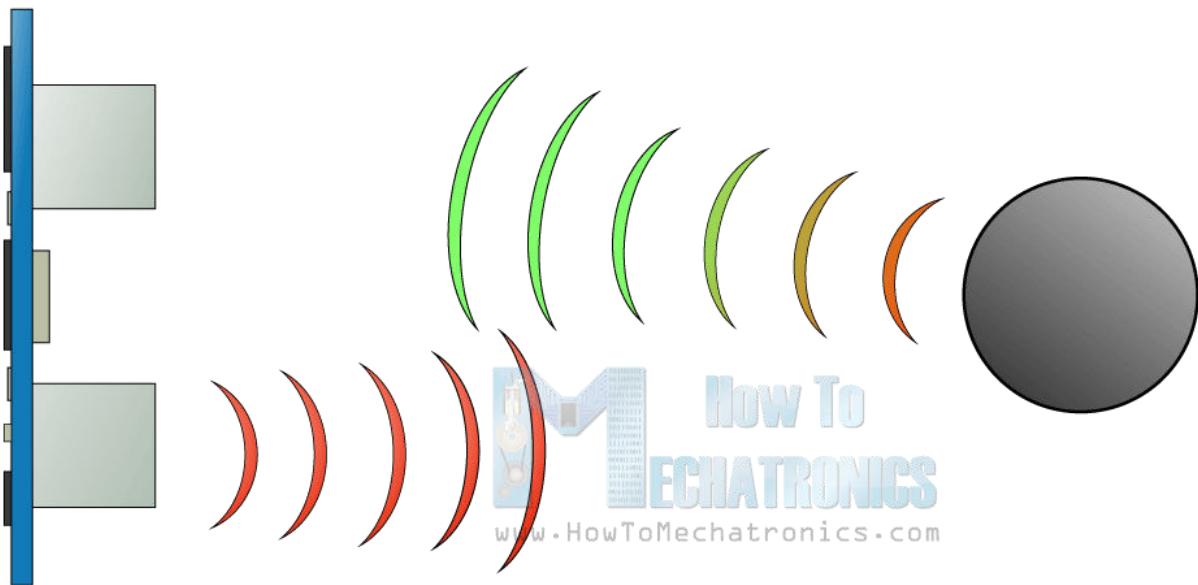
In this section we will learn about sonar sensor (HC-SR04) integration with arduino. The HC-SR04 ultrasonic sensor uses SONAR to determine the distance of an object just like the bats do. It offers excellent non-contact range detection with high accuracy and stable readings in an easy-to-use package from 2 cm to 400 cm or 1" to 13 feet.

The operation is not affected by sunlight or black material, although acoustically, soft materials like cloth can be difficult to detect. It comes complete with ultrasonic transmitter and receiver module.

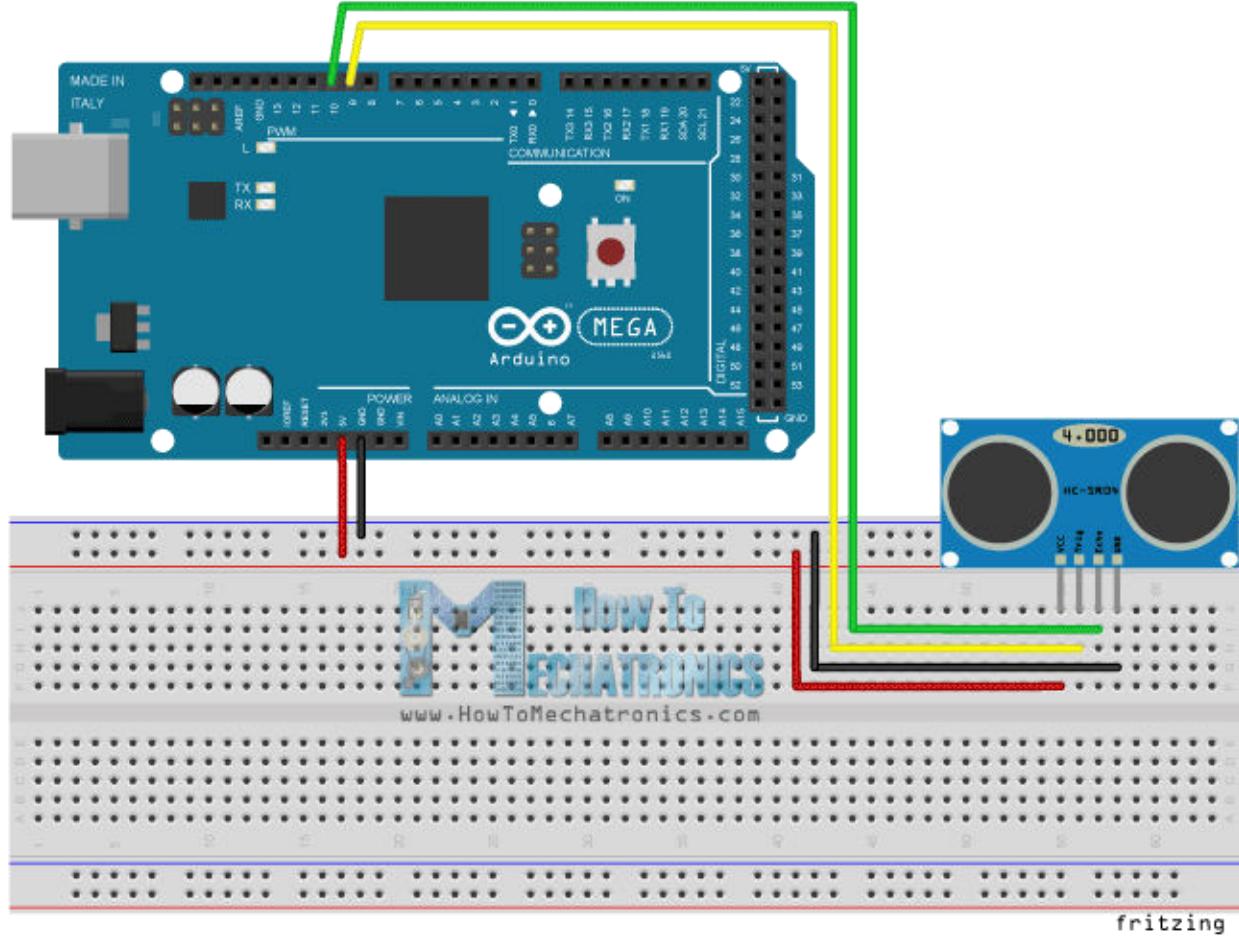
### 11.2.1 - How It Works – Ultrasonic Sensor

[reference - <https://howtomechatronics.com/tutorials/arduino/ultrasonic-sensor-hc-sr04/>]

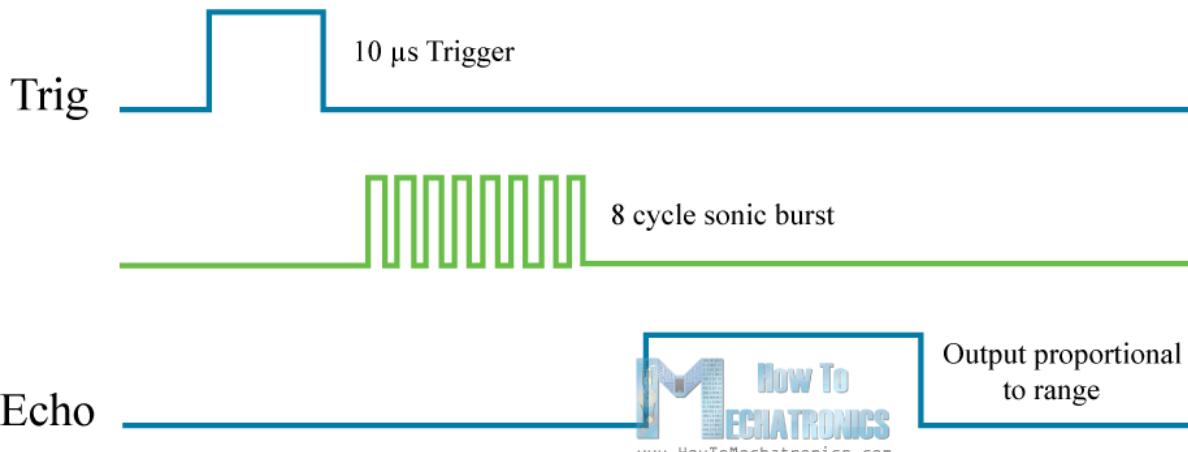
It emits an ultrasound at 40 000 Hz which travels through the air and if there is an object or obstacle on its path it will bounce back to the module. Considering the travel time and the speed of the sound you can calculate the distance.



The HC-SR04 Ultrasonic Module has 4 pins, Ground, VCC, Trig and Echo. The Ground and the VCC pins of the module needs to be connected to the Ground and the 5 volts pins on the Arduino Board respectively and the trig and echo pins to any Digital I/O pin on the Arduino Board.

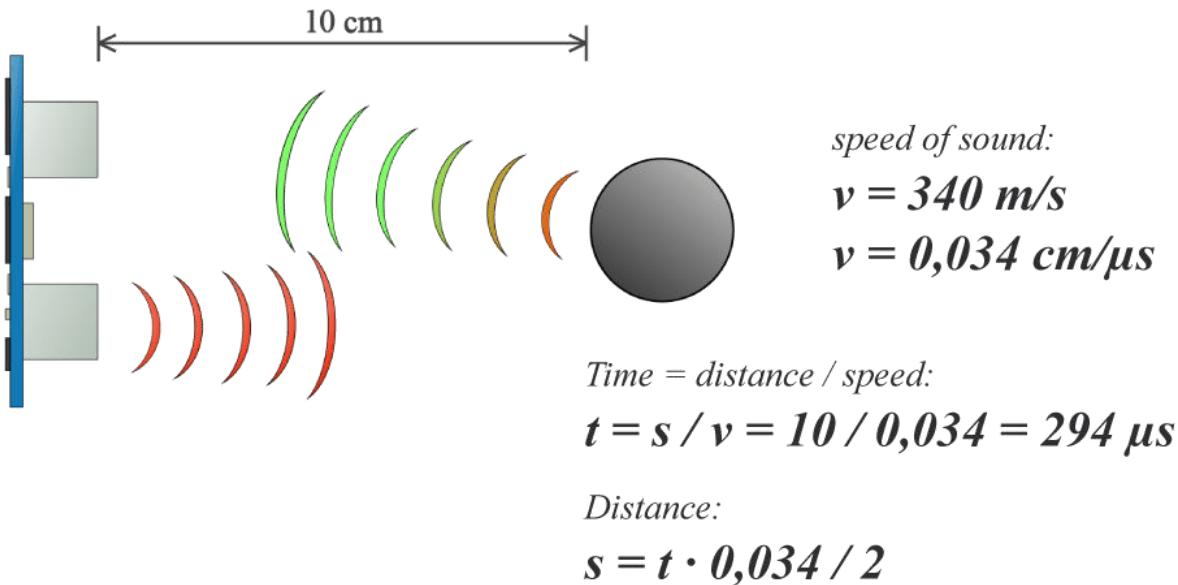


In order to generate the ultrasound you need to set the Trig on a High State for 10  $\mu$ s. That will send out an 8 cycle sonic burst which will travel at the speed sound and it will be received in the Echo pin. The Echo pin will output the time in microseconds the sound wave traveled.



For example, if the object is 10 cm away from the sensor, and the speed of the sound is 340 m/s or 0.034 cm/ $\mu$ s the sound wave will need to travel about 294  $\mu$ s. But what you will

get from the Echo pin will be double that number because the sound wave needs to travel forward and bounce backward. So in order to get the distance in cm we need to multiply the received travel time value from the echo pin by 0.034 and divide it by 2.



### 11.2.2 - Component

- 1) HC-SR04 Sonar Sensor
- 2) Breadboard
- 3) Arduino

If you wanna buy those component from Bangladesh contact Cybernetics Robo Store (+8801761500020)

### 11.2.3 - Source Code

First you have to define the Trig and Echo pins. In this case they are the pins number 9 and 10 on the Arduino Board and they are named trigPin and echoPin. Then you need a Long variable, named "duration" for the travel time that you will get from the sensor and an integer variable for the distance.

In the setup you have to define the trigPin as an output and the echoPin as an Input and also start the serial communication for showing the results on the serial monitor.

In the loop first you have to make sure that the trigPin is clear so you have to set that pin on a LOW State for just 2  $\mu\text{s}$ . Now for generating the Ultra sound wave we have to set the trigPin on HIGH State for 10  $\mu\text{s}$ . Using the **pulseIn()** function you have to read the travel time and put that value into the variable "duration". This function has 2 parameters, the first one is the name of the echo pin and for the second one you can write either HIGH or LOW. In this case, HIGH means that the **pulseIn()**function will wait for the pin to go HIGH caused by the bounced sound wave and it will start timing, then it will wait for the pin to go LOW when the sound wave will end which will stop the timing. At the end the function will return the length of the pulse in microseconds. For getting the distance we will multiply the duration by 0.034 and divide it by 2

as we explained this equation previously. At the end we will print the value of the distance on the Serial Monitor.

```
/*
 * Ultrasonic Sensor HC-SR04 and Arduino Tutorial
 *
 * by Dejan Nedelkovski,
 * www.HowToMechatronics.com
 *
 */
// defines pins numbers
const int trigPin = 9;
const int echoPin = 10;

// defines variables
long duration;
int distance;

void setup() {
pinMode(trigPin, OUTPUT); // Sets the trigPin as an Output
pinMode(echoPin, INPUT); // Sets the echoPin as an Input
Serial.begin(9600); // Starts the serial communication
}

void loop() {
// Clears the trigPin
digitalWrite(trigPin, LOW);
delayMicroseconds(2);

// Sets the trigPin on HIGH state for 10 micro seconds
digitalWrite(trigPin, HIGH);
delayMicroseconds(10);
digitalWrite(trigPin, LOW);

// Reads the echoPin, returns the sound wave travel time in microseconds
duration = pulseIn(echoPin, HIGH);

// Calculating the distance
distance= duration*0.034/2;

// Prints the distance on the Serial Monitor
Serial.print("Distance: ");
Serial.println(distance);
```

{

College Level:

## 11.3 - Interface Laser Distance Measuring Sensor

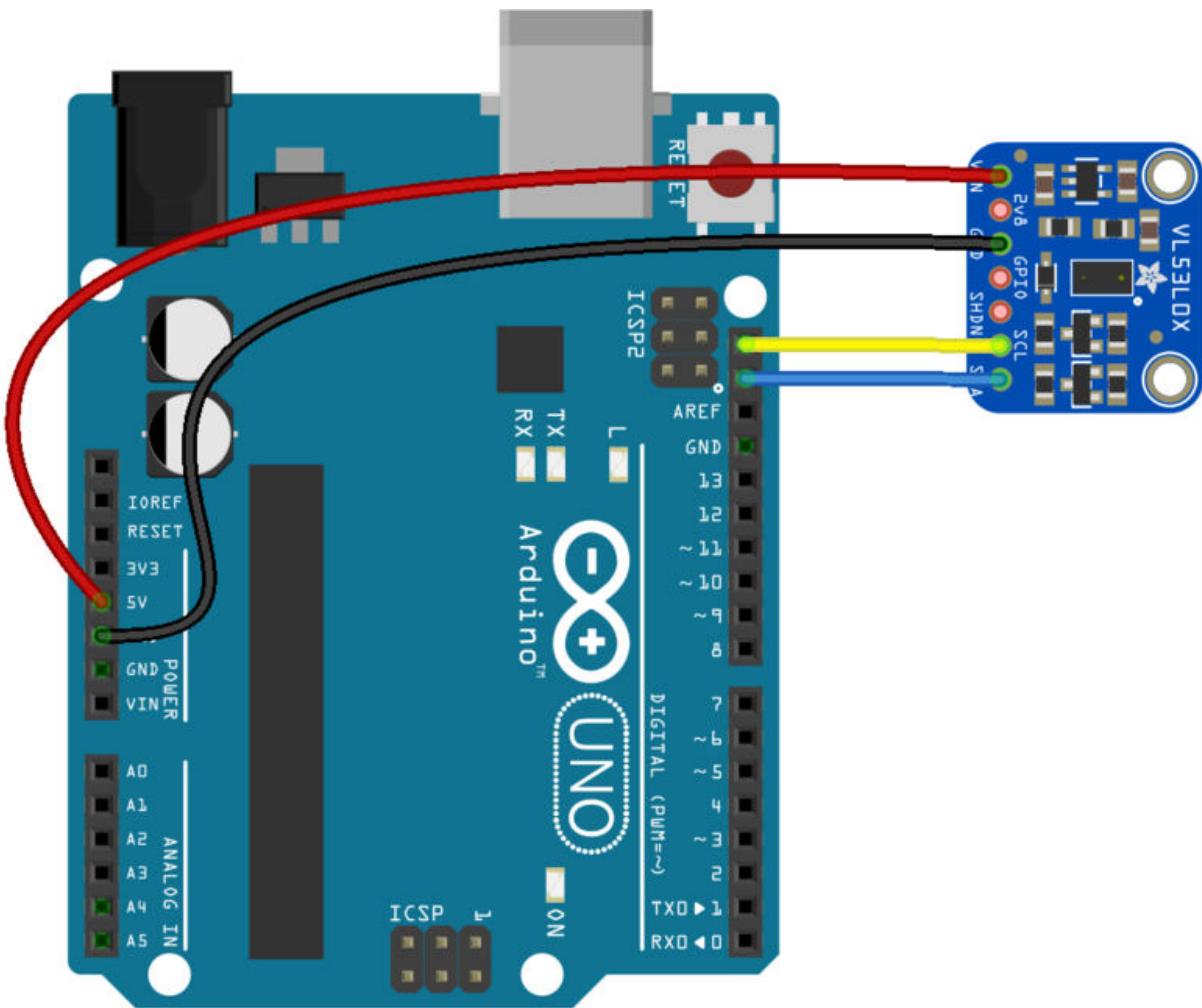
[reference - This article is directly copied from <https://learn.adafruit.com/adafruit-vl53l0x-micro-lidar-distance-sensor-breakout/arduino-code>]

### 11.3.1 - Sensor

You can buy sensor from adafruit, aliexpress or any other shop. If you are in Bangladesh you can buy from Cybernetics Robo Store (Call : +8801761 5000 20 or <https://www.facebook.com/CyberneticsRoboAcademy/>)

### 11.3.2 - Connection with Arduino

You can easily wire this breakout to any microcontroller, we'll be using an Arduino. For another kind of microcontroller, just make sure it has I2C, then port the API code. We strongly recommend using an Arduino to start though!



fritzing

- Connect **Vin** to the power supply, 3-5V is fine. Use the same voltage that the microcontroller logic is based off of. For most Arduinos, that is 5V
- Connect **GND** to common power/data ground
- Connect the **SCL** pin to the I2C clock **SCL** pin on your Arduino. On an UNO & '328 based Arduino, this is also known as **A5**, on a Mega it is also known as **digital 21** and on a Leonardo/Micro, **digital 3**
- Connect the **SDA** pin to the I2C data **SDA** pin on your Arduino. On an UNO & '328 based Arduino, this is also known as **A4**, on a Mega it is also known as **digital 20** and on a Leonardo/Micro, **digital 2**

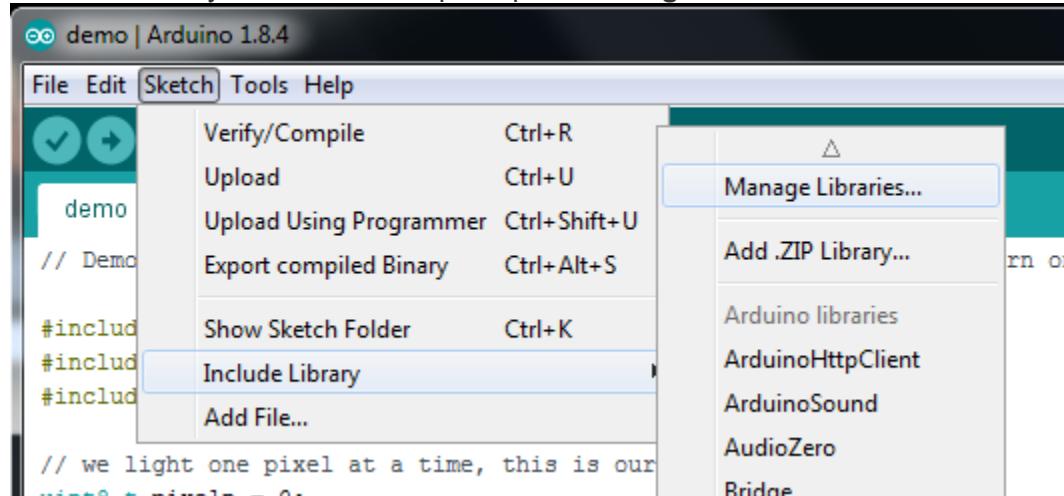
The VL53L0X has a default I2C address of **0x29!**

You *can* change it, but only in software. That means you have to wire the SHUTDOWN pin and hold all but one sensor in reset while you reconfigure one sensor at a time

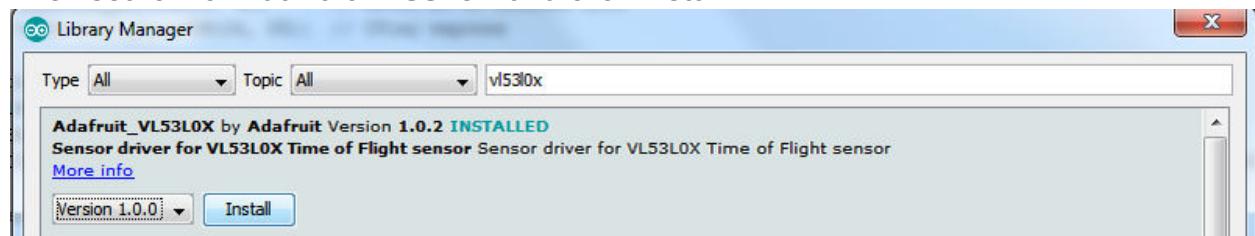
### 11.3.3 - Download Adafruit\_VL53L0X Library

To begin reading sensor data, you will need to install the [Adafruit VL53L0X Library](#).

The easiest way to do that is to open up the **Manage Libraries...** menu in the Arduino IDE



Then search for **Adafruit VL53L0X** and click **Install**

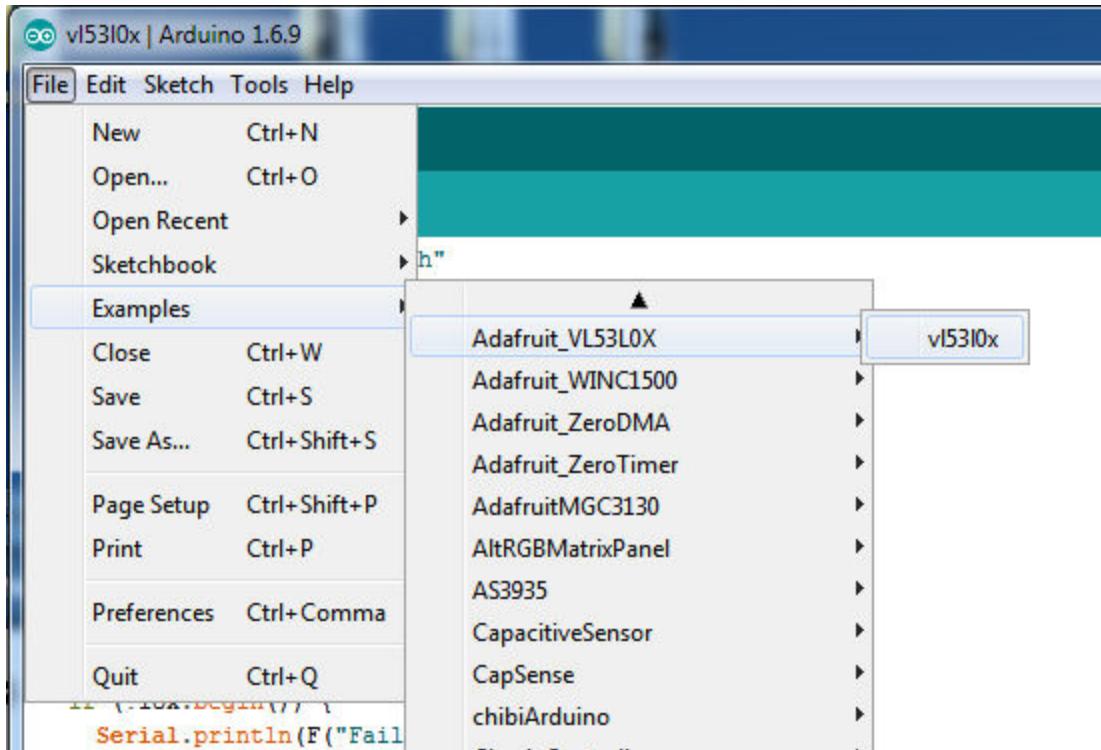


Arduino library installation at:

<http://learn.adafruit.com/adafruit-all-about-arduino-libraries-install-use>

### 11.3.4 - Load Demo

Open up **File->Examples->Adafruit\_VL53L0X->vl53l0x** and upload to your Arduino wired up to the sensor



That's it! Now open up the serial terminal window at 115200 speed to begin the test.

```

VL53L0X API Simple Ranging example

Reading a measurement... out of range
Reading a measurement... Distance (mm): 61
Reading a measurement... Distance (mm): 41
Reading a measurement... Distance (mm): 33
Reading a measurement... Distance (mm): 50
Reading a measurement... Distance (mm): 89
Reading a measurement... Distance (mm): 135
Reading a measurement... Distance (mm): 195
Reading a measurement... Distance (mm): 220
Reading a measurement... Distance (mm): 160
Reading a measurement... Distance (mm): 97
Reading a measurement... Distance (mm): 64
Reading a measurement... Distance (mm): 81
Reading a measurement... Distance (mm): 240
Reading a measurement... Distance (mm): 349
Reading a measurement... Distance (mm): 398
Reading a measurement... Distance (mm): 438
Reading a measurement... Distance (mm): 433
Reading a measurement... Distance (mm): 453
Reading a measurement... Distance (mm): 454
Reading a measurement... Distance (mm): 462
Reading a measurement... Distance (mm): 452

```

Autoscroll      Both NL & CR      115200 baud

Move your hand up and down to read the sensor data. Note that when nothing is detected, it will say the reading is out of range

### 11.3.5 - Connecting Multiple Sensors

I<sup>2</sup>C only allows one address-per-device so you have to make sure each I<sup>2</sup>C device has a unique address. The default address for the VL53L0X is **0x29** but you *can* change this in software.

To set the new address you can do it one of two ways. During initialization, instead of calling `lox.begin()`, call `lox.begin(0x30)` to set the address to 0x30. Or you can, later, call `lox.setAddress(0x30)` at any time.

The good news is its easy to change, the annoying part is each *other* sensor has to be in shutdown. You can shutdown each sensor by wiring up to the **XSHUT** pin to a microcontroller pin. Then perform something like this pseudo-code:

1. Reset all sensors by setting all of their XSHUT pins low for delay(10), then set all XSHUT high to bring out of reset
2. Keep sensor #1 awake by keeping XSHUT pin high
3. Put all other sensors into shutdown by pulling XSHUT pins low
4. Initialize sensor #1 with `lox.begin(new_i2c_address)` Pick any number but 0x29 and it must be under 0x7F. Going with 0x30 to 0x3F is probably OK.
5. Keep sensor #1 awake, and now bring sensor #2 out of reset by setting its XSHUT pin high.
6. Initialize sensor #2 with `lox.begin(new_i2c_address)` Pick any number but 0x29 and whatever you set the first sensor to
7. Repeat for each sensor, turning each one on, setting a unique address.

Note you must do this *every* time you turn on the power, the addresses are not permanent!

## Class 12:

### Project : Sonar Sensor Integration and Show data in Serial & LCD (Part 2)

#### 12.1 - LCD Interfacing with Arduino

[reference - Arduino Cookbook , page-334

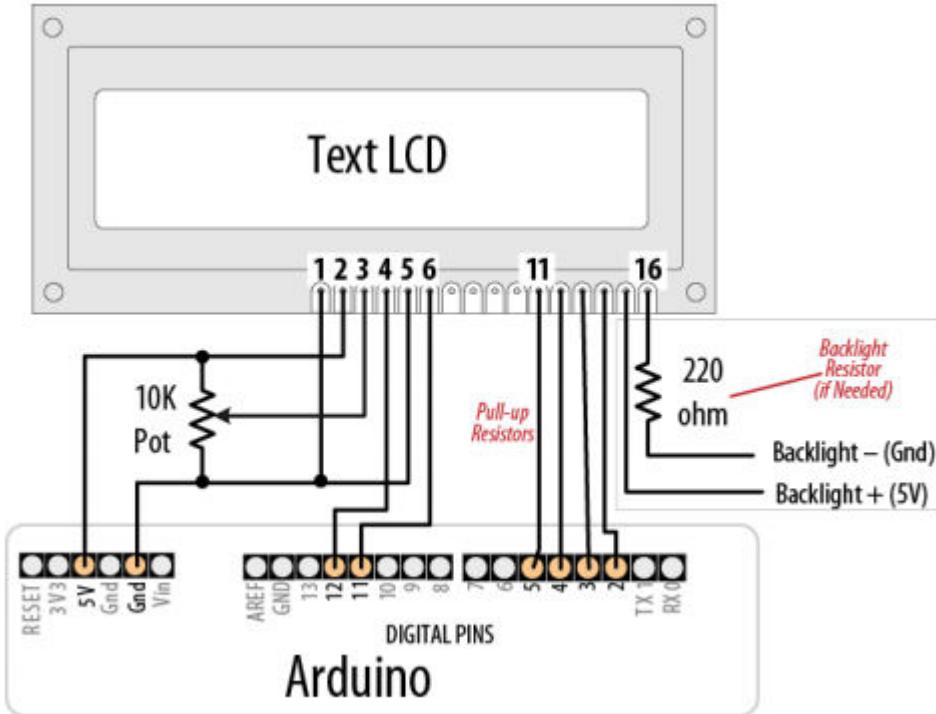
<https://www.arduino.cc/en/Reference/LiquidCrystalPrint>

<https://playground.arduino.cc/Code/LCD/>

]

##### 12.1.1 - Overview & Connection

The arduino software has LiquidCrystal library for driving LCD displays based on the HD44780 chip. To make the display working, you need to wire the power, data, and control pins properly. Connect the data and status lines to digital output pins, and wire up a contrast potentiometer and connect the power lines. If our display has a backlight, this needs connection, usually through a resistor.



### LCD pin connections

| LCD pin | Function                   | Arduino pin |
|---------|----------------------------|-------------|
| 1       | Gnd or OV or Vss           | Gnd         |
| 2       | +5V or Vdd                 | 5V          |
| 3       | V <sub>o</sub> or contrast |             |
| 4       | RS                         | 12          |
| 5       | R/W                        |             |
| 6       | E                          | 11          |
| 7       | D0                         |             |
| 8       | D1                         |             |
| 9       | D2                         |             |
| 10      | D3                         |             |
| 11      | D4                         | 5           |
| 12      | D5                         | 4           |
| 13      | D6                         | 3           |
| 14      | D7                         | 2           |
| 15      | A or analog                |             |
| 16      | K or cathode               |             |

#### 12.1.2 - First Test with LCD

The Arduino software includes the LiquidCrystal library for driving LCD displays based on the HD44780 chip. Connect the display according to above diagram(section 12.1.1) . Double-check the wiring before you apply power, as you can damage the LCD if you connect the power pins incorrectly. To run the HelloWorld sketch provided with Arduino, click the IDE Files menu item and navigate to Examples→Library→LiquidCrystal→HelloWorld.

```
#include <LiquidCrystal.h>

// initialize the library by associating any needed LCD interface pin
// with the arduino pin number it is connected to
```

```

const int rs = 12, en = 11, d4 = 5, d5 = 4, d6 = 3, d7 = 2;
LiquidCrystal lcd(rs, en, d4, d5, d6, d7);

void setup() {
 // set up the LCD's number of columns and rows:
 lcd.begin(16, 2);
 // Print a message to the LCD.
 lcd.print("hello, world!");
}

void loop() {
 // set the cursor to column 0, line 1
 // (note: line 1 is the second row, since counting begins with 0):
 lcd.setCursor(0, 1);
 // print the number of seconds since reset:
 lcd.print(millis() / 1000);
}

```

### 12.1.3 - LCD Details

See this link <http://www.circuitbasics.com/how-to-set-up-an-lcd-display-on-an-arduino/>

Here is the description of some common functions used to control LCD with arduino.

#### **LiquidCrystal()**

The LiquidCrystal() function sets the pins the Arduino uses to connect to the LCD. You can use any of the Arduino's digital pins to control the LCD. Just put the Arduino pin numbers inside the parentheses in this order: LiquidCrystal(RS, E, D4, D5, D6, D7). RS, E, D4, D5, D6, D7 are the LCD pins.

For example, say you want LCD pin D7 to connect to Arduino pin 12. Just put "12" in place of D7 in the function like this: LiquidCrystal(RS, E, D4, D5, D6, 12). This function needs to be placed before the void setup() section of the program.

#### **lcd.begin()**

This function sets the dimensions of the LCD. It needs to be placed before any other LiquidCrystal function in the void setup() section of the program. The number of rows and columns are specified as lcd.begin(columns, rows). For a 16x2 LCD, you would use lcd.begin(16, 2), and for a 20x4 LCD you would use lcd.begin(20, 4).

#### **lcd.clear()**

This function clears any text or data already displayed on the LCD. If you use lcd.clear() with lcd.print() and the delay() function in the void loop() section, you can make a simple blinking text program:

```
#include <LiquidCrystal.h>
```

```
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);

void setup() {
 lcd.begin(16, 2);
}

void loop() {
 lcd.print("hello, world!");
 delay(500);
 lcd.clear();
 delay(500);
}
```

### **lcd.home()**

This function places the cursor in the upper left hand corner of the screen, and prints any subsequent text from that position. For example, this code replaces the first three letters of “hello world!” with X’s:

```
#include <LiquidCrystal.h>

LiquidCrystal lcd(12, 11, 5, 4, 3, 2);

void setup() {
 lcd.begin(16, 2);
 lcd.print("hello, world!");
}

void loop() {
 lcd.home();
 lcd.print("XXX");
}
```

### **lcd.setCursor()**

Similar, but more useful than lcd.home() is lcd.setCursor(). This function places the cursor (and any printed text) at any position on the screen. It can be used in the void setup() or void loop() section of your program.

The cursor position is defined with lcd.setCursor(column, row). The column and row coordinates start from zero (0-15 and 0-1 respectively). For example, using lcd.setCursor(2, 1) in the void setup() section of the “hello, world!” program above prints “hello, world!” to the lower line and shifts it to the right two spaces:

```
#include <LiquidCrystal.h>
```

```
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);

void setup() {
 lcd.begin(16, 2);
 lcd.setCursor(2, 1);
 lcd.print("hello, world!");
}

void loop() {
```

### **lcd.write()**

You can use this function to write different types of data to the LCD, for example the reading from a temperature sensor, or the coordinates from a GPS module. You can also use it to print custom characters that you create yourself (more on this below). Use lcd.write() in the void setup() or void loop() section of your program.

### **lcd.print()**

This function is used to print text to the LCD. It can be used in the void setup() section or the void loop() section of the program.

To print letters and words, place quotation marks (" ") around the text. For example, to print hello, world!, use lcd.print("hello, world!").

To print numbers, no quotation marks are necessary. For example, to print 123456789, use lcd.print(123456789).

lcd.print() can print numbers in decimal, binary, hexadecimal, and octal bases. For example:

- lcd.print(100, DEC) prints "100";
- lcd.print(100, BIN) prints "1100100"
- lcd.print(100, HEX) prints "64"
- lcd.print(100, OCT) prints "144"

### **lcd.Cursor()**

This function creates a visible cursor. The cursor is a horizontal line placed below the next character to be printed to the LCD.

The function lcd.noCursor() turns the cursor off. lcd.cursor() and lcd.noCursor() can be used together in the void loop() section to make a blinking cursor similar to what you see in many text input fields:

```
#include <LiquidCrystal.h>

LiquidCrystal lcd(12, 11, 5, 4, 3, 2);

void setup() {
```

```

lcd.begin(16, 2);
lcd.print("hello, world!");
}

void loop() {
 lcd.cursor();
 delay(500);
 lcd.noCursor();
 delay(500);
}

```

This places a blinking cursor after the exclamation point in “hello, world!”

Cursors can be placed anywhere on the screen with the `lcd.setCursor()` function. This code places a blinking cursor directly below the exclamation point in “hello, world!”:

```

#include <LiquidCrystal.h>

LiquidCrystal lcd(12, 11, 5, 4, 3, 2);

void setup() {
 lcd.begin(16, 2);
 lcd.print("hello, world!");
}

void loop() {
 lcd.setCursor(12, 1);
 lcd.cursor();
 delay(500);
 lcd.setCursor(12, 1);
 lcd.noCursor();
 delay(500);
}

```

### **lcd.blink()**

This function creates a block style cursor that blinks on and off at approximately 500 milliseconds per cycle. Use it in the void `loop()` section. The function `lcd.noBlink()` disables the blinking block cursor.

### **lcd.display()**

This function turns on any text or cursors that have been printed to the LCD screen. The function `lcd.noDisplay()` turns off any text or cursors printed to the LCD, without clearing it from the LCD’s memory.

These two functions can be used together in the void loop() section to create a blinking text effect. This code will make the “hello, world!” text blink on and off:

```
#include <LiquidCrystal.h>

LiquidCrystal lcd(12, 11, 5, 4, 3, 2);

void setup() {
 lcd.begin(16, 2);
 lcd.print("hello, world!");
}

void loop() {
 lcd.display();
 delay(500);
 lcd.noDisplay();
 delay(500);
}
```

### **Icd.scrollDisplayLeft()**

This function takes anything printed to the LCD and moves it to the left. It should be used in the void loop() section with a delay command following it. The function will move the text 40 spaces to the left before it loops back to the first character. This code moves the “hello, world!” text to the left, at a rate of one second per character:

```
#include <LiquidCrystal.h>

LiquidCrystal lcd(12, 11, 5, 4, 3, 2);

void setup() {
 lcd.begin(16, 2);
 lcd.print("hello, world!");
}

void loop() {
 lcd.scrollDisplayLeft();
 delay(1000);
}
```

Text strings longer than 40 spaces will be printed to line 1 after the 40th position, while the start of the string will continue printing to line 0.

### **Icd.scrollDisplayRight()**

This function behaves like Icd.scrollDisplayLeft(), but moves the text to the right.

**lcd.autoscroll()**

This function takes a string of text and scrolls it from right to left in increments of the character count of the string. For example, if you have a string of text that is 3 characters long, it will shift the text 3 spaces to the left with each step:

```
#include <LiquidCrystal.h>

LiquidCrystal lcd(12, 11, 5, 4, 3, 2);

void setup() {
 lcd.begin(16, 2);
}

void loop() {
 lcd.setCursor(0, 0);
 lcd.autoscroll();
 lcd.print("ABC");
 delay(500);
}
```

Like the lcd.scrollDisplay() functions, the text can be up to 40 characters in length before repeating. At first glance, this function seems less useful than the lcd.scrollDisplay() functions, but it can be very useful for creating animations with custom characters.

**lcd.noAutoscroll()**

lcd.noAutoscroll() turns the lcd.autoscroll() function off. Use this function before or after lcd.autoscroll() in the void loop() section to create sequences of scrolling text or animations.

**lcd.rightToLeft()**

This function sets the direction that text is printed to the screen. The default mode is from left to right using the command lcd.leftToRight(), but you may find some cases where it's useful to output text in the reverse direction:

```
#include <LiquidCrystal.h>

LiquidCrystal lcd(12, 11, 5, 4, 3, 2);

void setup() {
 lcd.begin(16, 2);
 lcd.setCursor(12, 0);
 lcd.rightToLeft();
```

```

lcd.print("hello, world!");
}

void loop() {
}

```

This code prints the “hello, world!” text as “!dlrow ,olleh”. Unless you specify the placement of the cursor with `lcd.setCursor()`, the text will print from the (0, 1) position and only the first character of the string will be visible.

### **`lcd.createChar()`**

This command allows you to create your own custom characters. Each character of a 16×2 LCD has a 5 pixel width and an 8 pixel height. Up to 8 different custom characters can be defined in a single program. To design your own characters, you’ll need to make a binary matrix of your custom character from an [LCD character generator](#) or map it yourself. This code creates a degree symbol (°):

```

#include <LiquidCrystal.h>

LiquidCrystal lcd(12, 11, 5, 4, 3, 2);
byte customChar[8] = {
 0b00110,
 0b01001,
 0b01001,
 0b00110,
 0b00000,
 0b00000,
 0b00000,
 0b00000
};

void setup()
{
 lcd.createChar(0, customChar);
 lcd.begin(16, 2);
 lcd.write((uint8_t)0);
}

void loop() {
}

```

See this video : <https://www.youtube.com/watch?v=Mr9FQKcrGpA>

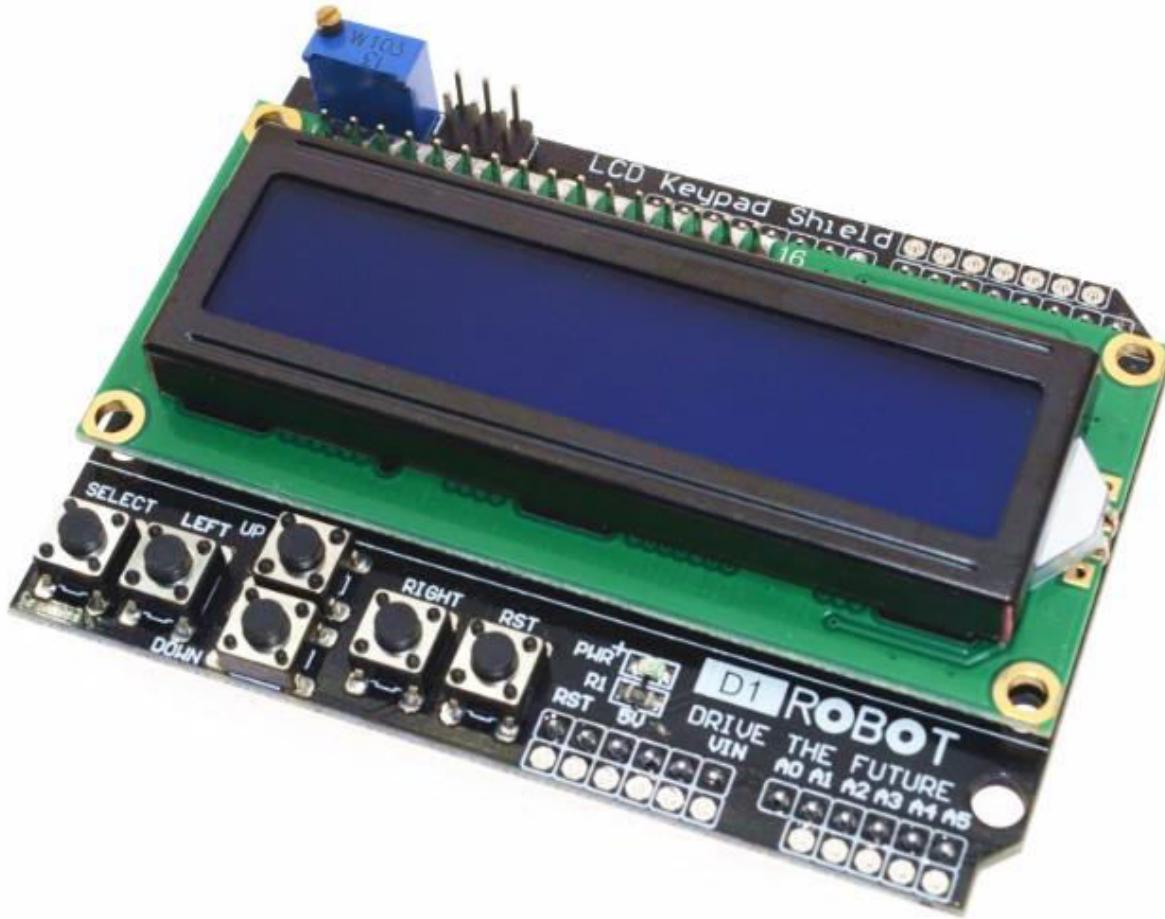
**Advanced Project : LCD(16\*2) Menu System**

web: <http://eeenthusiast.com/arduino-lcd-tutorial-display-menu-system-scrolling-menu-changeable-variables-projects/>

youtube : <https://www.youtube.com/watch?v=Q58mQFwWv7c>

## 12.2 - LCD Shield Interfacing with Arduino

Now a days LCD shield with 16\*2 LCD and some buttons are available in market. You can buy from this link <http://s.click.aliexpress.com/e/bCc4lyi8> . It has few buttons which produce different analog value based on clicked button. So you just need a one analog pin to read all buttons. For example if we press LEFT button we will have almost 2.35 Volt in A0 . For this particular shield you have to select pin like **LiquidCrystal Lcd(8, 9, 4, 5, 6, 7);**



### Example Code :

```
#include <LiquidCrystal.h>
//const int rs = 8, en = 9, d4 = 4, d5 = 5, d6 = 6, d7 = 7;
```

```

LiquidCrystal lcd(8, 9, 4, 5, 6, 7);

void setup() {
 // set up the LCD's number of columns and rows:
 lcd.begin(16, 2); // (column, row)
 lcd.clear();
}

void loop() {
 lcd.setCursor(0, 0);
 lcd.print("www.cruxbd.com");
 lcd.setCursor(0, 1);
 float sensorValue = analogRead(0);
 float voltage = sensorValue * (5.0 / 1023.0);
 lcd.print(voltage);
 if(voltage > 2.2 && voltage < 2.4){
 lcd.setCursor(10, 1);
 lcd.print("Left");
 }
 else{
 lcd.setCursor(10, 1);
 lcd.print("Nothing");
 }
}

```

#### **Youtube Link :**

**Menu Using Arduino** 1) <https://www.youtube.com/watch?v=4zCxL3suIQ>

**General Tutorial about LCD Shield** 2) [https://www.youtube.com/watch?v=naASSiS\\_9rEw](https://www.youtube.com/watch?v=naASSiS_9rEw)

**Game Using LCD Shield** 3) <https://www.youtube.com/watch?v=O69huEK0YT4>

### 12.3 - Show sonar sensor data in LCD

First see previous class for Sonar sensor (11.2.1). Then interface LCD and upload following code.

```

/*
 * Ultrasonic Sensor HC-SR04 and Arduino Tutorial
 *
 * by Dejan Nedelkovski,
 * www.HowToMechatronics.com
 *
 */

```

```
#include <LiquidCrystal.h> // includes the LiquidCrystal Library

LiquidCrystal lcd(1, 2, 4, 5, 6, 7); // Creates an LCD object. Parameters:
(rs, enable, d4, d5, d6, d7)

const int trigPin = 9;
const int echoPin = 10;

long duration;
int distanceCm, distanceInch;

void setup() {
lcd.begin(16,2); // Initializes the interface to the LCD screen, and
specifies the dimensions (width and height) of the display

pinMode(trigPin, OUTPUT);
pinMode(echoPin, INPUT);
}

void loop() {
digitalWrite(trigPin, LOW);
delayMicroseconds(2);

digitalWrite(trigPin, HIGH);
delayMicroseconds(10);
digitalWrite(trigPin, LOW);

duration = pulseIn(echoPin, HIGH);
distanceCm= duration*0.034/2;
distanceInch = duration*0.0133/2;

lcd.setCursor(0,0); // Sets the location at which subsequent text written
to the LCD will be displayed
lcd.print("Distance: "); // Prints string "Distance" on the LCD
lcd.print(distanceCm); // Prints the distance value from the sensor
lcd.print(" cm");
delay(10);
lcd.setCursor(0,1);
lcd.print("Distance: ");
lcd.print(distanceInch);
lcd.print(" inch");
delay(10);
```

{}

## College Level

### 12.4 - I2C Protocol , I2C LCD

<https://i2c.info/>

## Class 13:

- 13.1 - Project :Make a LFR(Line Follower Robot) (Part 1) - Introduction
- 13.2 - Basic LFR Mechanism
- 13.3 - Line Detection & Control Mechanism
- 13.4 - LFR Circuit & Sensor Array Interfacing
- 13.5 - Sensor Data Show in Serial

### 13.1 - Project :Make a LFR(Line Follower Robot) (Part 1) - Introduction

Line follower is an autonomous robot which follows either black line in white area or white line in black area. Robot must be able to detect particular line and keep following it. Infrared based sensors are used to detect line. For special situations such as cross overs where robot can have more than one path which can be followed, predefined path must be followed by the robot. Large line follower robots are usually used in industries for assisting the automated production process. They are also used in military applications, human assistance purpose, delivery services etc.

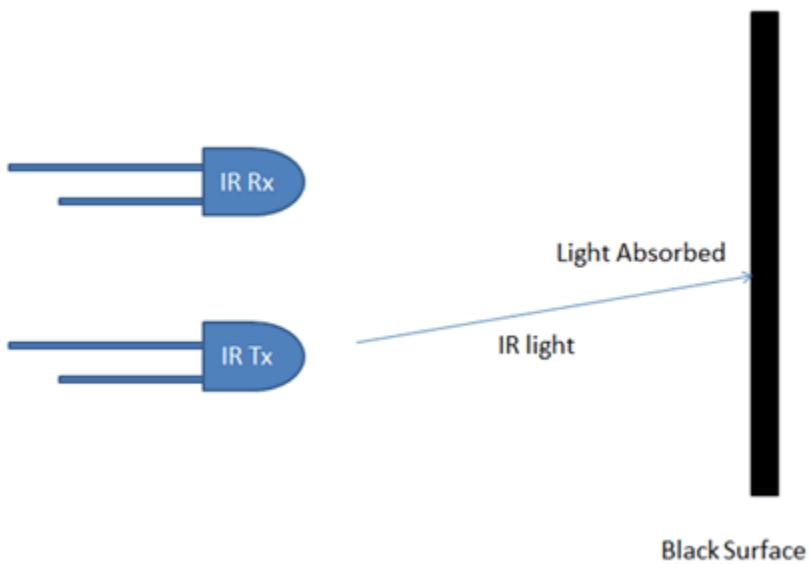
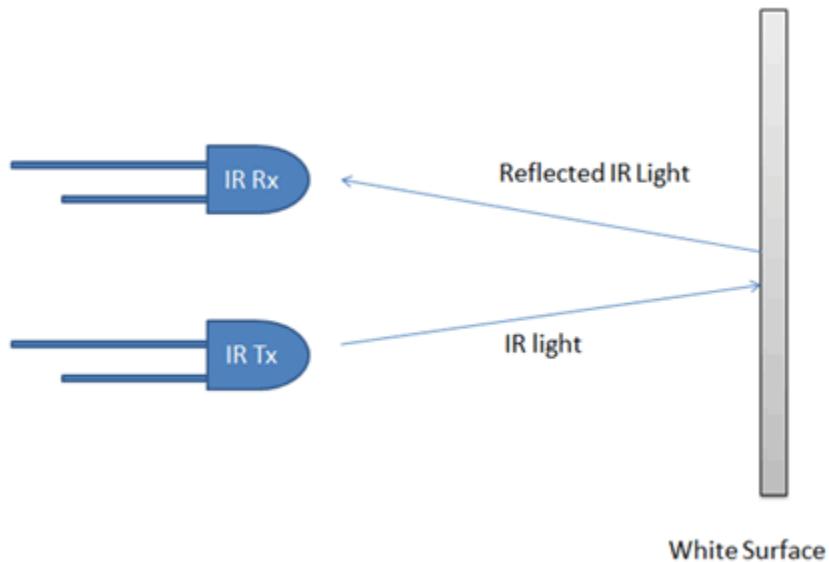
Line follower Robot is one of the first robots that beginners and students would get their first robotic experience with. In this project, we have designed a simple Line Follower Robot using Arduino and some other components.

Some advance line follower robot use PID algorithm which is not our task in this section.

## 13.2 - Basic LFR Mechanism

[reference - <https://circuitdigest.com/microcontroller-projects/line-follower-robot-using-arduino> ]

Concept of working of line follower is related to light. We use here the behavior of light at black and white surface. When light fall on a white surface it is almost full reflected and in case of black surface light is completely absorbed. This behavior of light is used in building a line follower robot.



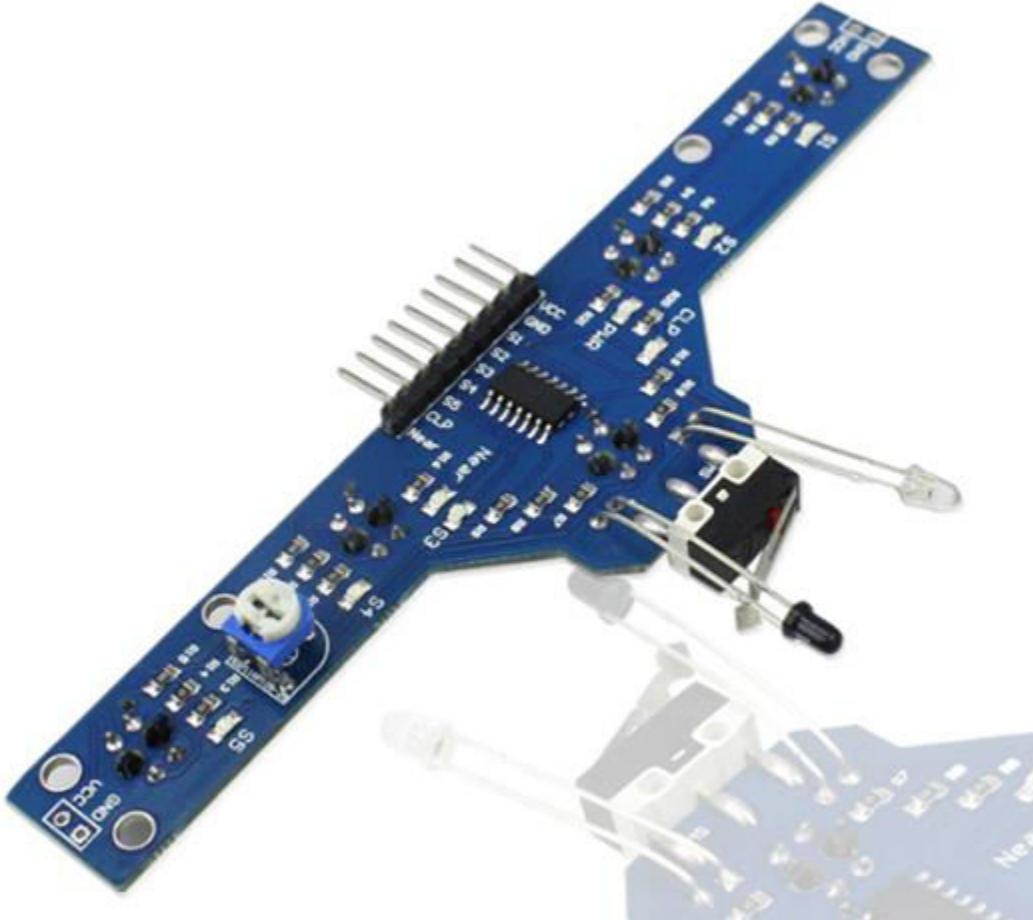
In this arduino based line follower robot we have used IR Transmitters and IR receivers also called photo diodes. They are used for sending and receiving light. IR transmits infrared lights. When infrared rays falls on white surface, it's reflected back and caught by photodiodes which generates some voltage changes. When IR light falls on a black surface, light is absorb by the black surface and no rays are reflected back, thus photo diode does not receive any light or rays.

Here in this arduino line follower robot when sensor senses white surface then arduino gets 1 as input and when senses black line arduino gets 0 as input.

The whole arduino line follower robot can be divided into 3 sections: sensor section, control section and driver section.

### **Sensor section:**

This section contains IR diodes, potentiometer, Comparator (Op-Amp) and LED's. We will use line follower sensor module which will make our task much more easier. Sensor module link : <https://www.facebook.com/commerce/products/2430668250295729/>



This is actually a line follower sensor array. Here five sensors detect line . There is a extra infrared sensor placed in front of sensor module to detect obstacle. Some technical parameters of this sensor module are :

Detection range is 0 to 4 cm (black and white line sensor) 0 to 5 cm (adjustable distance detection)

The input voltage: 3.0 to 5.5 V

Output format: digital output (v)

### **Control Section:**

Arduino is used for controlling whole the process of line follower robot. The outputs of sensors(S1, S2, S3, S4, S5) are connected with arduino. Arduino read these signals and send commands to driver circuit to drive line follower. There are five led's on the upper portion of sensor module which help us to visually debug. If you place sensor module in black line you can see led of respective sensor will on and give us a output 5V .

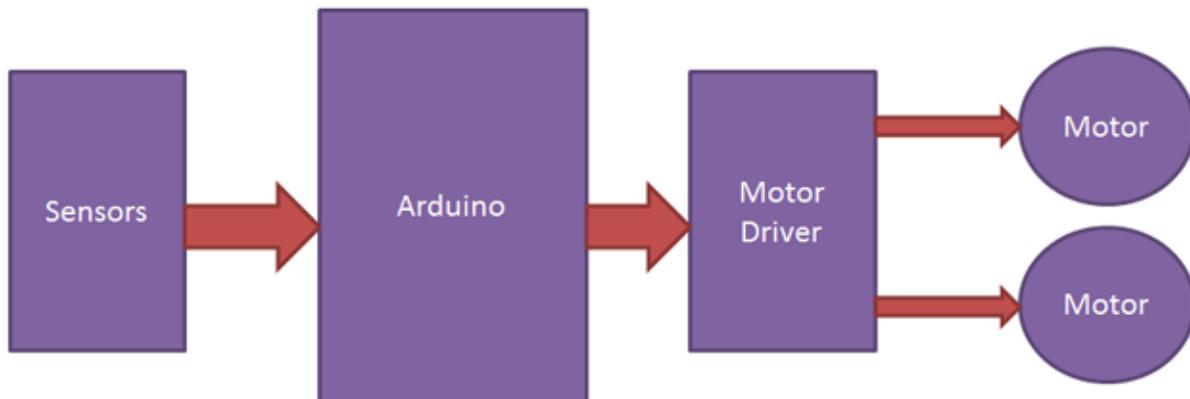
### **Driver section:**

Driver section consists motor driver and two DC motors. Motor driver is used for driving motors because arduino does not supply enough voltage and current to motor. So we add a motor driver circuit to get enough voltage and current for motor. Arduino sends commands to this motor driver and then it drive motors. We hope you already have learned about motor driver in previous classes.

## 13.3 - Line Detection & Control Mechanism

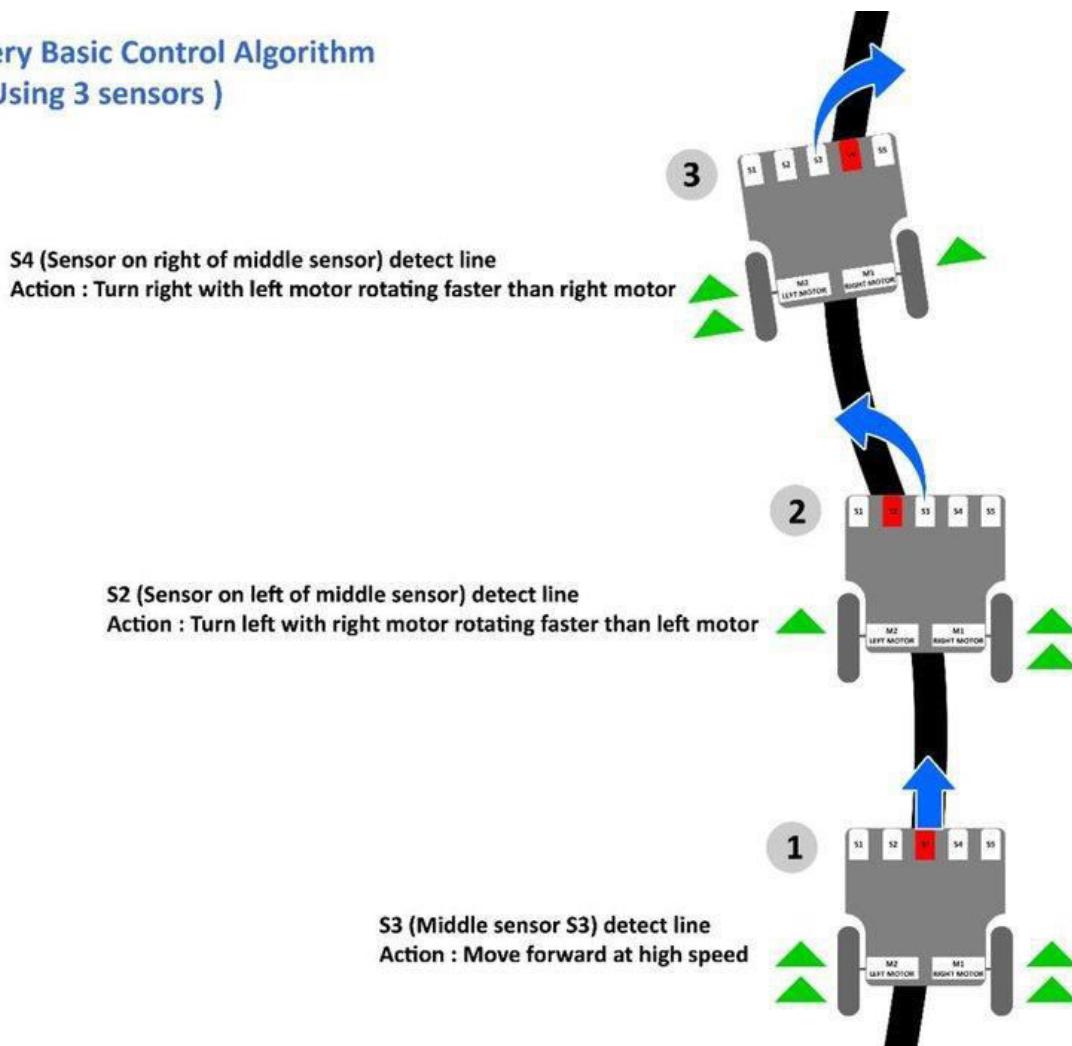
[reference - <https://www.instructables.com/id/Arduino-based-Desktop-Line-Follower-jolliBot/>]

Working of line follower is very interesting. Line follower robot senses black line by using sensor and then sends the signal to arduino. Then arduino drives the motor according to sensors' output.



Now for our convenience we are assuming we have only three sensors. Following picture shows the control mechanism .

### Very Basic Control Algorithm ( Using 3 sensors )



## 13.4 - LFR Circuit & Sensor Array Interfacing

### Sensor Connection with Arduino

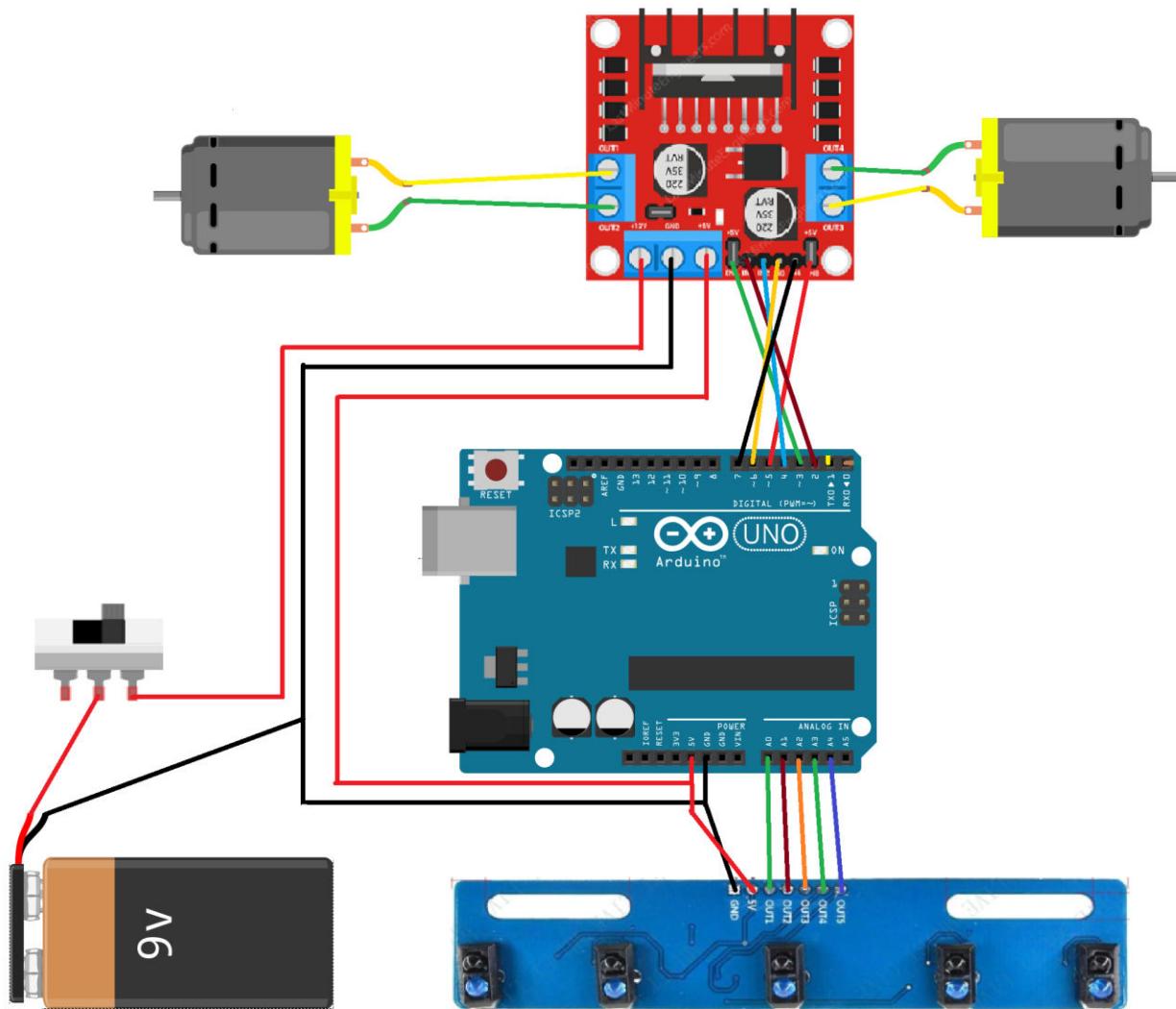
Sensor output marked as S1, S2 , S3, S4 , S5 which represent five sensors. Connections are :  
 S1>A0(S1 output with arduino A0), S2>A1, S3>A2, S4>A3, S5>A4

### Motor Driver Connection with Arduino

```
const int motor1Pin1 = 2; //D2 > Motor Driver IN1
const int motor1Pin2 = 4; //D4 > Motor Driver IN2
const int enablem1Pin3 = 3; //D3 > Motor Driver EN A
const int motor2Pin1 = 6; //D6 > Motor Driver IN3 motor 2, pin 1
const int motor2Pin2 = 7; //D7 > Motor Driver IN4 motor 2, pin 2
const int enablem2Pin3 = 5; //D5 > Motor Driver EN B
```

const has been used because we are not going to change the pin number while robot is operating and const will save ram space of arduino.

We will read sensor value as analog read. Analog value will be less than 1000 if black line is detected .



### 13.5 - Code Upload & Serial Data

Arduino Code :

```
//Black Line Below 1000 (100, 200)
//1000 Up = White
//sensor link: https://www.facebook.com/commerce/products/2267154666679394/
int level = 1000;
```

```

int i = 0;
//Sensor Connection : S1>A0,S2>A1,S3>A2,S4>A3,S5>A4
int Pin[5] = {A4, A3, A2, A1, A0};
int sensorData[5] = {};

char state = 'S';
//Declear arduino desired pin number as constant integer
const int motor1Pin1 = 2; //D2 > Motor Driver IN1
const int motor1Pin2 = 4; //D4 > Motor Driver IN2
const int enablem1Pin3 = 3; //D3 > Motor Driver EN A
const int motor2Pin1 = 6; //D6 > Motor Driver IN3 motor 2, pin 1
const int motor2Pin2 = 7; //D7 > Motor Driver IN4 motor 2, pin 2
const int enablem2Pin3 = 5; //D5 > Motor Driver EN B

void setup() {
 Serial.begin(9600);
 for(i=0; i<5; i++){
 pinMode(Pin[i], INPUT); //set sensor pin as input
 }

 //set motor pin as output
 pinMode(motor1Pin1, OUTPUT);
 pinMode(motor1Pin2, OUTPUT);
 pinMode(enablem1Pin3, OUTPUT);
 pinMode(motor2Pin1, OUTPUT);
 pinMode(motor2Pin2, OUTPUT);
 pinMode(enablem2Pin3, OUTPUT);
}

void loop() {
 for(i=0; i<5; i++){
 sensorData[i] = analogRead(Pin[i]);
 Serial.print(" ");
 Serial.print(sensorData[i]);
 }
 Serial.println();

 //int Pin[5] = {A4, A3, A2, A1, A0};
 //Left 4 sensors are in BLACK > Turn Left
 if(sensorData[0]<level && sensorData[1]<level && sensorData[2]<level &&
 sensorData[3]<level) state = 'L';
 // Left 3 sensors are in BLACK > Turn Left
}

```

```

 else if(sensorData[0]<level && sensorData[1]<level &&
sensorData[2]<level) state = 'L';
// Left sensor no 2 & 3 are in BLACK > Turn Left
 else if(sensorData[1]<level && sensorData[2]<level) state = 'L';
// Left sensor no 1 & 2 are in BLACK > Turn Left
else if(sensorData[0]<level || sensorData[1]<level) state = 'L';

 else if(sensorData[1]<level && sensorData[2]<level &&
sensorData[3]<level && sensorData[4]<level) state = 'R';
 else if(sensorData[2]<level && sensorData[3]<level &&
sensorData[4]<level) state = 'R';
 else if(sensorData[2]<level && sensorData[3]<level) state = 'R';
 else if(sensorData[3]<level || sensorData[4]<level) state = 'R';

 else if(sensorData[0]>level && sensorData[1]>level &&
sensorData[2]>level && sensorData[3]>level && sensorData[4]>level) state =
'B';
 else if(sensorData[0]<level && sensorData[1]<level &&
sensorData[2]<level && sensorData[3]<level && sensorData[4]<level) state =
'S';

else state = 'F';

switch (state) {
 // forward
case 'F':
 digitalWrite(motor1Pin1, HIGH);
 digitalWrite(motor1Pin2, LOW);
 digitalWrite(motor2Pin1, LOW);
 digitalWrite(motor2Pin2, HIGH);
 analogWrite(enablem1Pin3, 50);
 analogWrite(enablem2Pin3, 60);
 break;

 // left
case 'L':
 digitalWrite(motor1Pin1, HIGH);
 digitalWrite(motor1Pin2, LOW);
 digitalWrite(motor2Pin1, LOW);
 digitalWrite(motor2Pin2, HIGH);
 analogWrite(enablem1Pin3, 50);
 analogWrite(enablem2Pin3, 0);
 break;
}

```

```

 // right
case 'R':
 digitalWrite(motor1Pin1, HIGH);
 digitalWrite(motor1Pin2, LOW);
 digitalWrite(motor2Pin1, LOW);
 digitalWrite(motor2Pin2, HIGH);
 analogWrite(enablem1Pin3, 0);
 analogWrite(enablem2Pin3, 60);
 break;

 // backward
case 'B':
 digitalWrite(motor1Pin1, LOW);
 digitalWrite(motor1Pin2, HIGH);
 digitalWrite(motor2Pin1, HIGH);
 digitalWrite(motor2Pin2, LOW);
 analogWrite(enablem1Pin3, 0);
 analogWrite(enablem2Pin3, 80);
 break;

 // Stop
case 'S':
 digitalWrite(motor1Pin1, LOW);
 digitalWrite(motor1Pin2, LOW);
 digitalWrite(motor2Pin1, LOW);
 digitalWrite(motor2Pin2, LOW);
 analogWrite(enablem1Pin3, 0);
 analogWrite(enablem2Pin3, 0);
}
}

```

Now without connecting battery just connect arduino and sensor with computer and open serial monitor. You can make a black line using black tape and white paper. Face sensors in black line and see serial monitor data which should give you a clear understanding what is going on.

## Class 14:

14.1 - LFR Part-2 : Make a complete LFR Robot

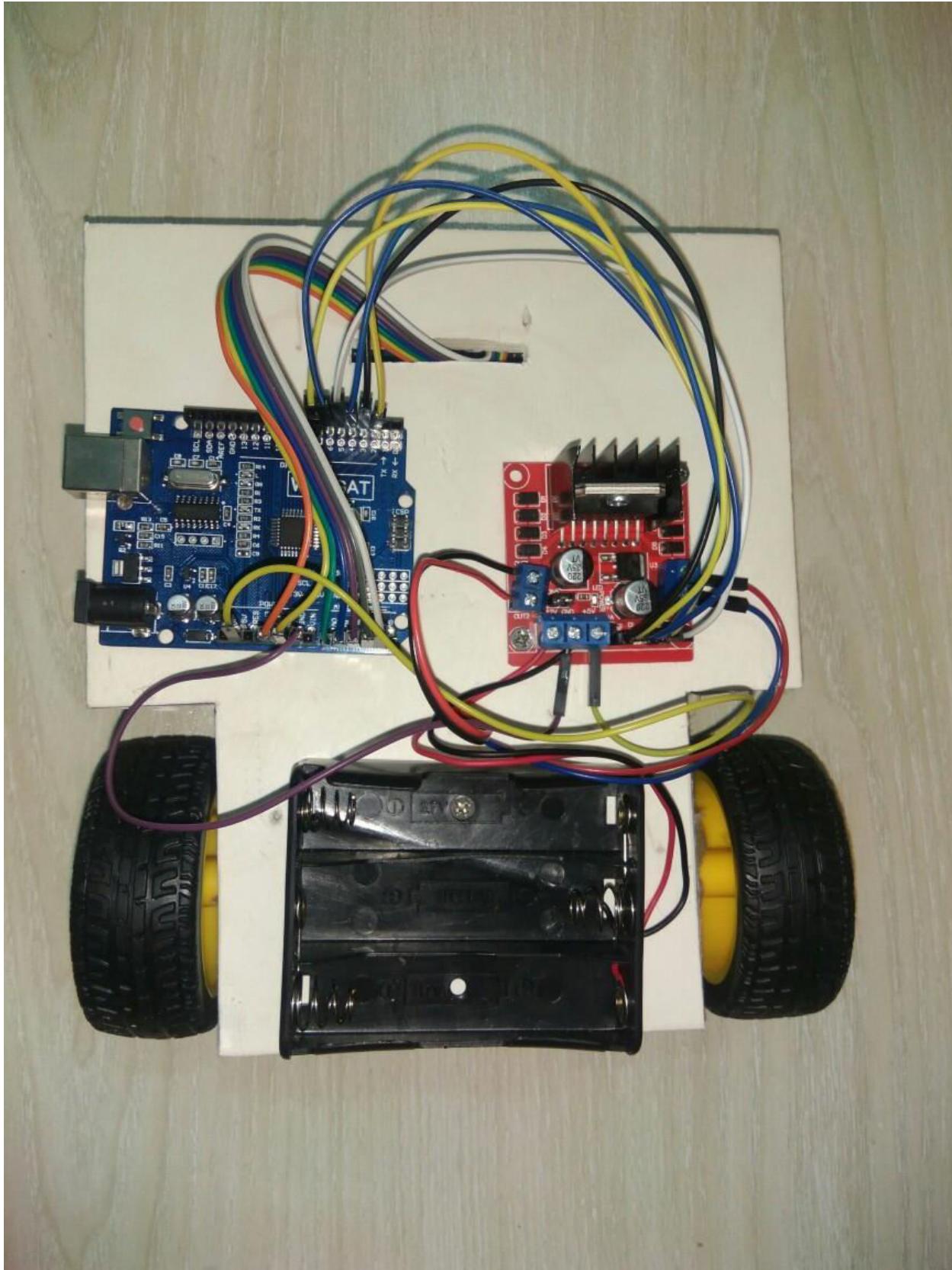
14.2 - Guide to Improvement

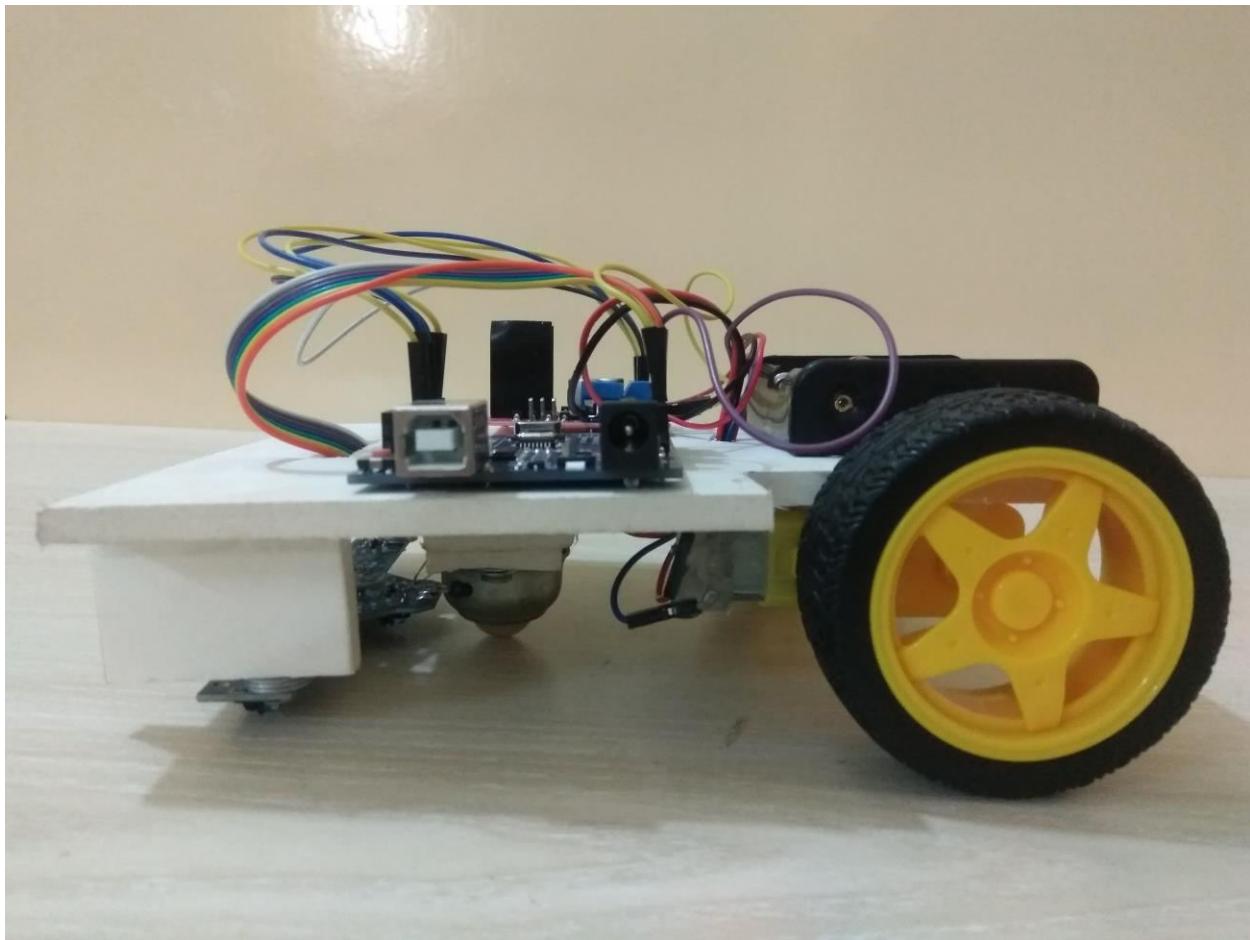
**College Level :**

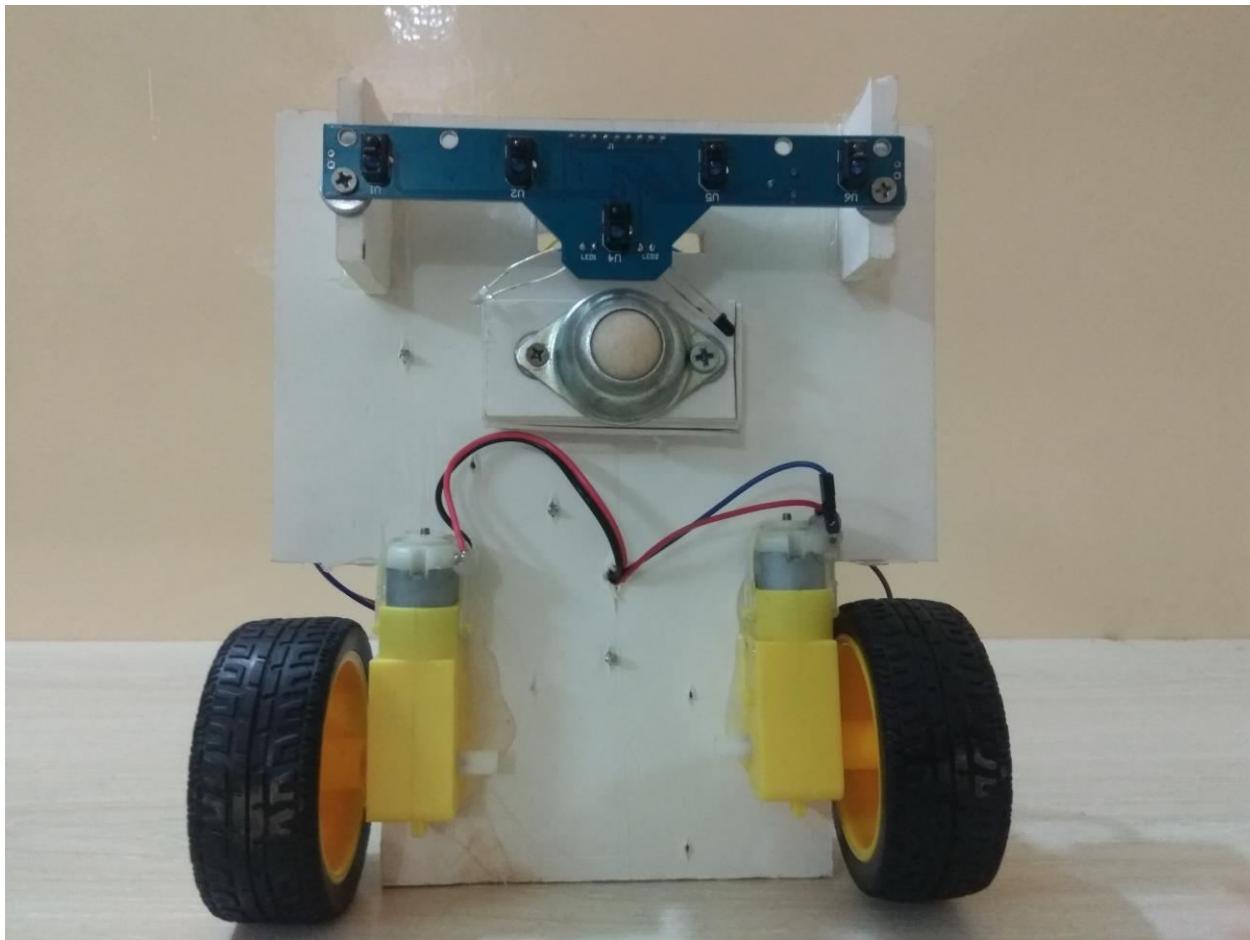
14.3 - DIY Sensor circuit with TCRT-5000

## 14.1 - LFR Part-2 : Make a complete LFR Robot

Connect all parts and you can mount them on PVC board like this picture below. Use gluegun and soldering iron.







## 14.2 - Guide to Improvement

Lots of improvement you can make because it's just a very basic line follower robot.

- 1) Use better motor . Micro metal gearmotor with small wheel should be a good option.
- 2) Use better structure for robot
- 3) This sensor array is not so good. Use a good sensor module which has 1 cm separation between two sensors. Try to make a sensor array by yourself.
- 4) Try to learn PID line follower .
- 5) Use a good ball caster .
- 6) More improvement ? Think yourself and suggest us.

**College Level :**

### 14.3 - DIY Sensor circuit with TCRT-5000

[reference - <https://www.electrodragon.com/analogread-from-a-tcrt5000-sensor/>  
<https://www.dwengo.org/sensors>]

This sensor consists of an infrared LED (IR-LED) which sends out light, and a phototransistor. The phototransistor conducts current in proportion to the IR-light that is reflected by an object in close proximity to the sensor. After soldering the right resistors, this sensor can be used to measure small distances (up to 1 cm) to objects, or detect contrast differences on a surface. This makes this type of sensor ideal to make for instance line-following robots. In line follower robots it can be used to detect obstacles or the end of a table. Due to its limited range, the sensor should be placed at most about 1 cm above the ground. Sensor link:

<https://www.facebook.com/commerce/products/2267154666679394/>

#### **TCRT5000 Specifications**

IR sensor with transistor output

Operating Voltage: 5V

Diode forward Current: 60mA

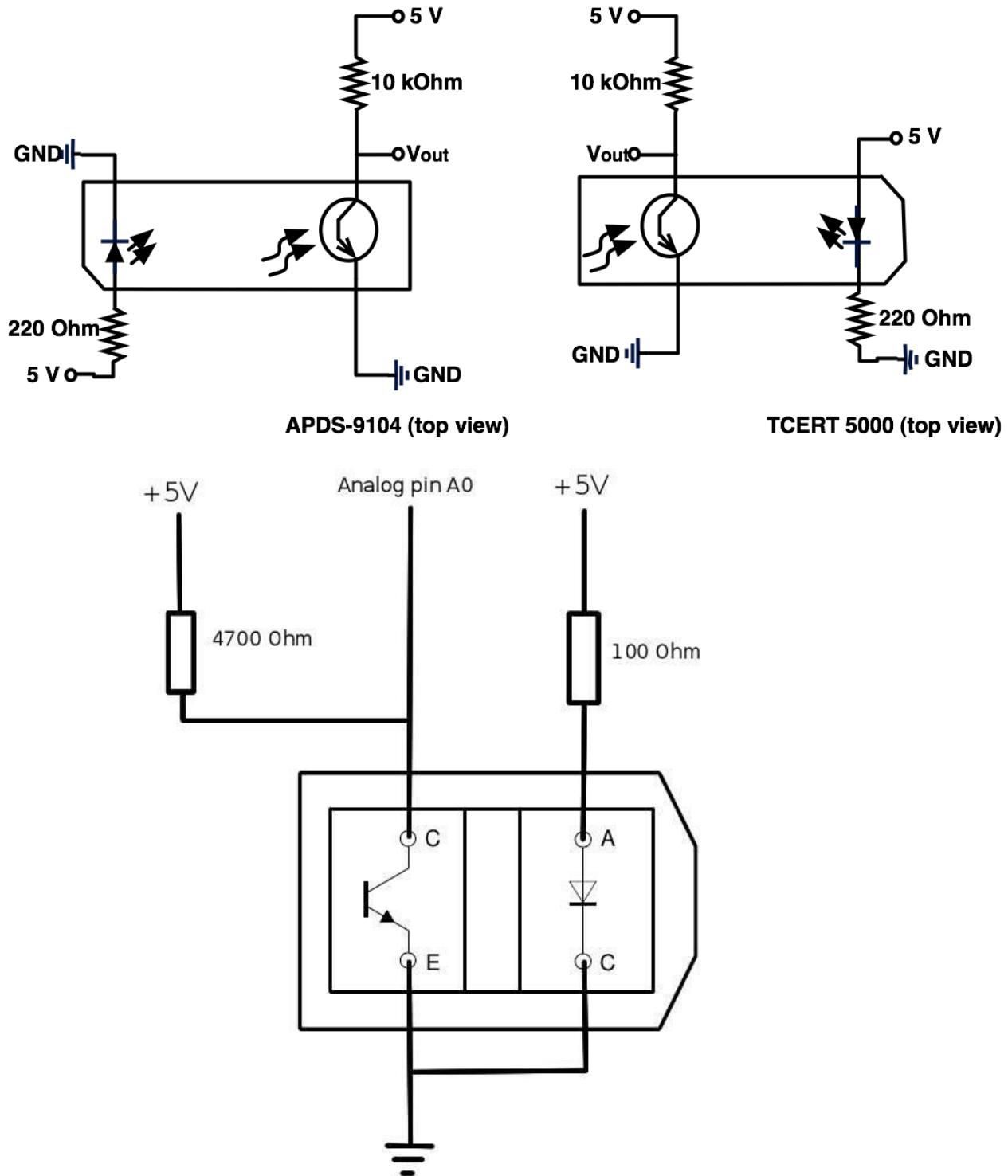
Output: Analog or digital data

Transistor collector current: 100mA (maximum)

Operating temperature: -25°C to +85°C



**Circuit :**

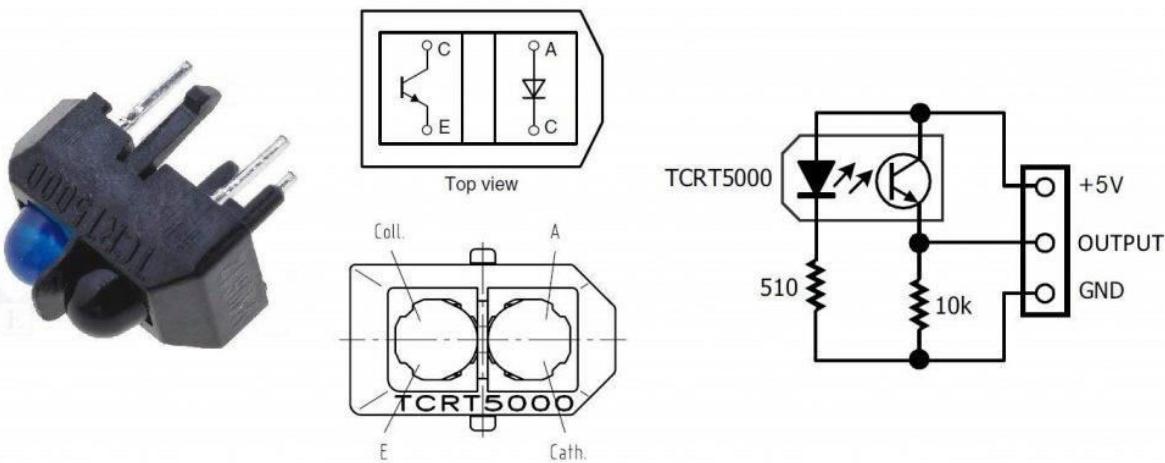


you can try with various resistor(within limit) for adjustment according to your line follower robot.

#### Working mechanism :

Phototransistor act like a normal transistor except it has no physical pin for base. It's an npn transistor. So when photodiode emit infrared light the base of phototransistor activated which

results current flow through collector and emitter. This transistor does not have a base pin because the biasing of the transistor is controlled by the amount of IR light it receives. So basically the IR light from the photodiode hits an object/surface and returns back to the photo transistor to bias it. So in this circuit we will get lower voltage if white line is detected(opposite of our previous sensor array module) . If we want our output opposite (white line will cause output HIGH) we have to make following circuit.



here 10K resistor is connected with emitter of phototransistor. So for white line IR reflection will be occurred which will pass current through phototransistor thus output will be high.

## Class 15:

Project: Temperature Sensor Interfacing and Show Temperature

15.1 - LM35 Temperature Sensor Interfacing

15.2 - Waterproof NTC Thermistor Interfacing

**College Level:**

15.3 - Basic Oscilloscope Operation

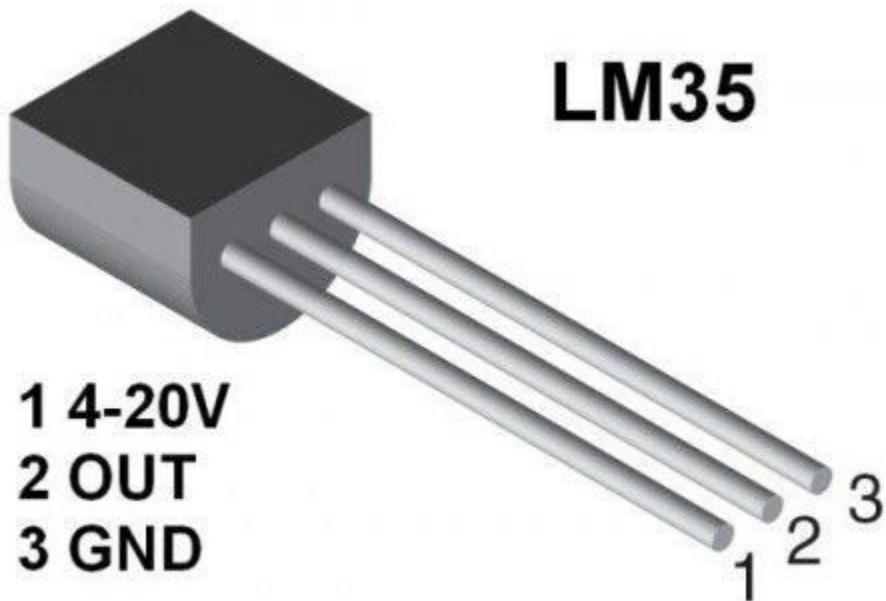
## 15 - Project: Temperature Sensor Interfacing and Show Temperature

### 15.1 - LM35 Temperature Sensor Interfacing

#### 15.1.1 - Introduction

The LM35 series are precision integrated-circuit temperature devices with an output voltage linearly proportional to the Centigrade temperature. LM35 is three terminal linear temperature sensor from National semiconductors. It can measure temperature from -55 degree Celsius to

+150 degree Celsius. The voltage output of the LM35 increases 10mV per degree Celsius rise in temperature. LM35 can be operated from a 5V supply and the stand by current is less than 60uA. The pin out of LM35 is shown in the figure below.

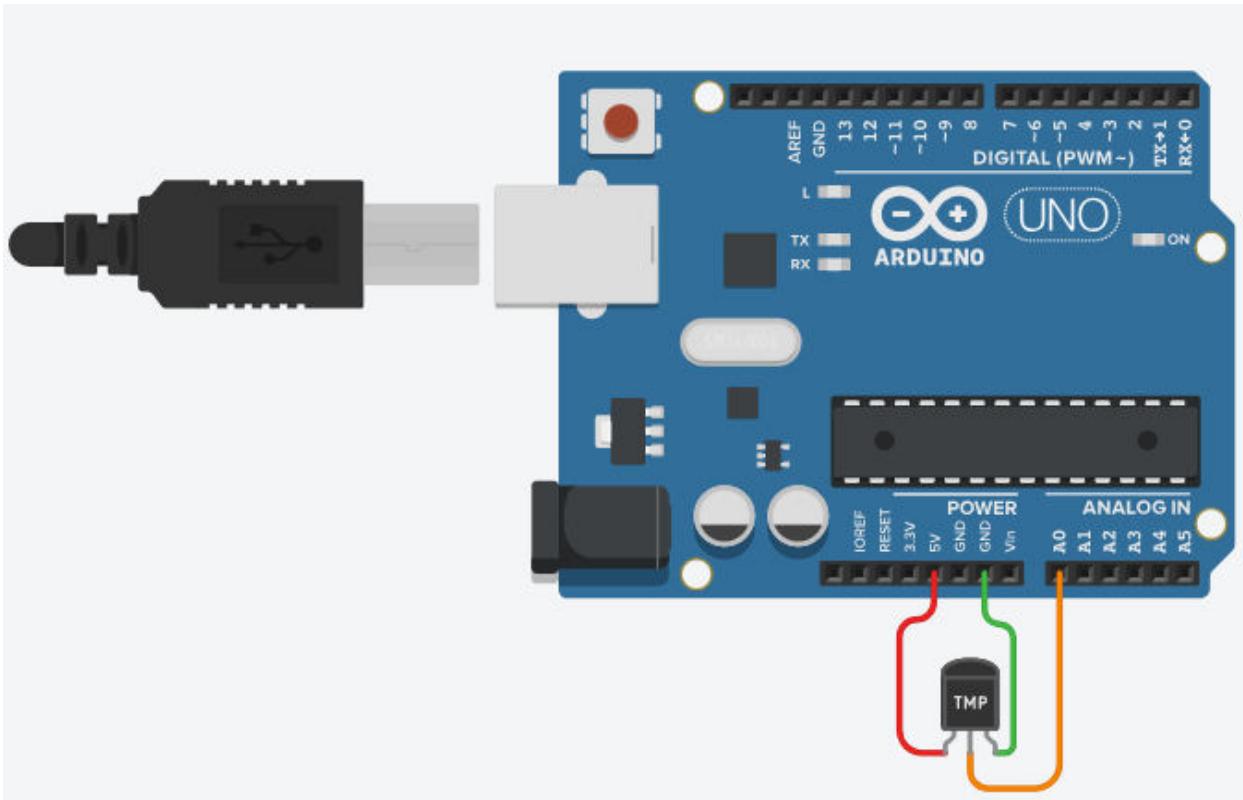


#### 15.1.2 - LM35 Features :

- Calibrated Directly in Celsius (Centigrade)
- Linear + 10-mV/°C Scale Factor
- 0.5°C Ensured Accuracy (at 25°C)
- Rated for Full -55°C to 150°C Range
- Suitable for Remote Applications
- Low-Cost Due to Wafer-Level Trimming
- Operates from 4 V to 30 V
- Less than 60- $\mu$ A Current Drain
- Low Self-Heating, 0.08°C in Still Air

- Non-Linearity Only  $\pm 1/4^\circ\text{C}$  Typical
- Low-Impedance Output,  $0.1 \Omega$  for 1-mA Load

### 15.1.3 - Circuit Diagram



### 15.1.4 - Arduino Code & Output

```
const int sensor = A0; // Assigning analog pin A5 to variable 'sensor'
float tempc; //variable to store temperature in degree Celsius
float tempf; //variable to store temperature in Fahrenheit
float vout; //temporary variable to hold sensor reading

void setup() {
 pinMode(sensor, INPUT); // Configuring sensor pin as input
 Serial.begin(9600);
}

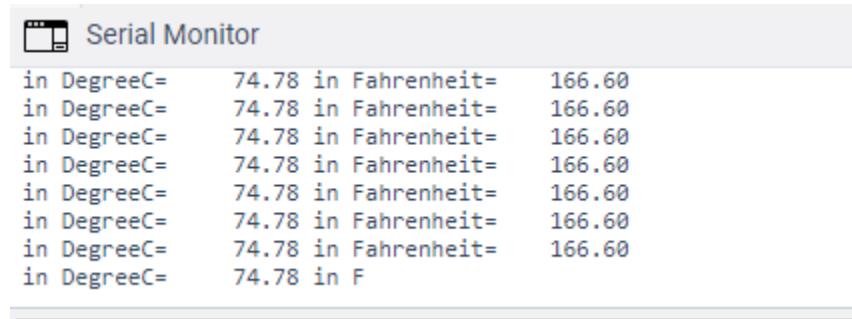
void loop() {
 vout = analogRead(sensor); //Reading the value from sensor
 vout = (vout * 500) / 1023;
 tempc = vout; // Storing value in Degree Celsius
```

```

tempf = (vout * 1.8) + 32; // Converting to Fahrenheit
Serial.print("in DegreeC=");
Serial.print("\t");
Serial.print(tempc);
Serial.print(" ");
Serial.print("in Fahrenheit=");
Serial.print("\t");
Serial.print(tempf);
Serial.println();
delay(500); //Delay of 1 second for ease of viewing
}

```

### Output :



The screenshot shows the Arduino Serial Monitor window. The title bar says "Serial Monitor". The main area displays a series of lines of text, each consisting of "in DegreeC=" followed by a value of "74.78" and "in Fahrenheit=" followed by a value of "166.60". This pattern repeats eight times. Below these lines, there is a single line of text: "in DegreeC= 74.78 in F".

```

in DegreeC= 74.78 in Fahrenheit= 166.60
in DegreeC= 74.78 in F

```

### 15.1.5 - Homework

Connect a LCD module and show temperature in LCD

## 15.2 - Waterproof NTC Thermistor Interfacing

### 15.2.1 - Introduction

Thermistors are variable resistors that change their resistance with temperature. They are classified by the way their resistance responds to temperature changes. In Negative Temperature Coefficient (NTC) thermistors, resistance decreases with an increase in temperature. In Positive Temperature Coefficient (PTC) thermistors, resistance increases with an increase in temperature.

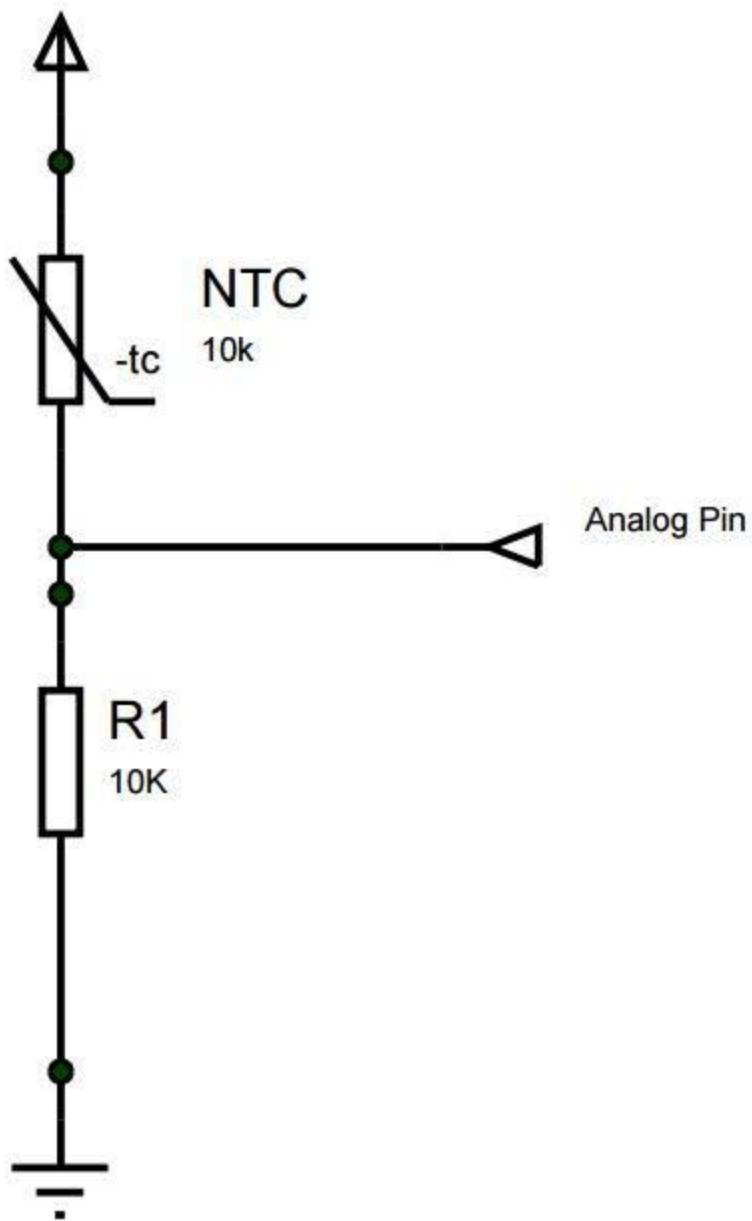
NTC thermistors are the most common, and that's the type we'll be using in this project. NTC thermistors are made from a semiconducting material (such as a metal oxide or ceramic) that's been heated and compressed to form a temperature sensitive conducting material.

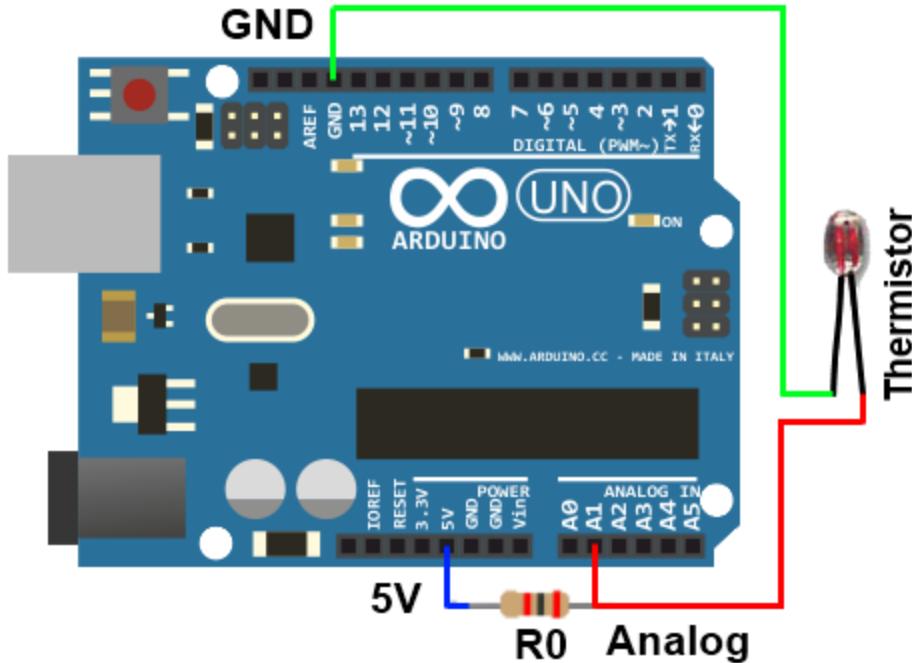
We will use a waterproof NTC 10K Thermistor. You can buy :

<http://s.click.aliexpress.com/e/WdIDdf2>



#### 15.2.2 - Circuit





### 15.2.3 - Code

We have to download a library from Arduino IDE > Sketch > Include Library > Manage Library > NTC\_Thermistor by Yuri Selimov

Open example from File > Examples > NTC Thermistor > Thermistor  
Edit the code for 10K NTC.

```
/**
 pin - an analog port number to be attached to the thermistor.
 R0 - reference resistance.
 Rn - nominal resistance.
 Tn - nominal temperature in Celsius.
 B - b-value of a thermistor.
 */
#include <NTC_Thermistor.h>

#define SENSOR_PIN A1
#define REFERENCE_RESISTANCE 10000
#define NOMINAL_RESISTANCE 100000
#define NOMINAL_TEMPERATURE 25
#define B_VALUE 3950

NTC_Thermistor* thermistor = NULL;

void setup() {
```

```
Serial.begin(9600);
thermistor = new NTC_Termistor(
 SENSOR_PIN,
 REFERENCE_RESISTANCE,
 NOMINAL_RESISTANCE,
 NOMINAL_TEMPERATURE,
 B_VALUE
);
}

void loop() {
 const double celsius = thermistor->readCelsius();
 const double kelvin = thermistor->readKelvin();
 const double fahrenheit = thermistor->readFahrenheit();
 Serial.print("Temperature: ");
 Serial.print(String(celsius) + " C, ");
 Serial.print(String(kelvin) + " K, ");
 Serial.println(String(fahrenheit) + " F");
 delay(500);
}
```

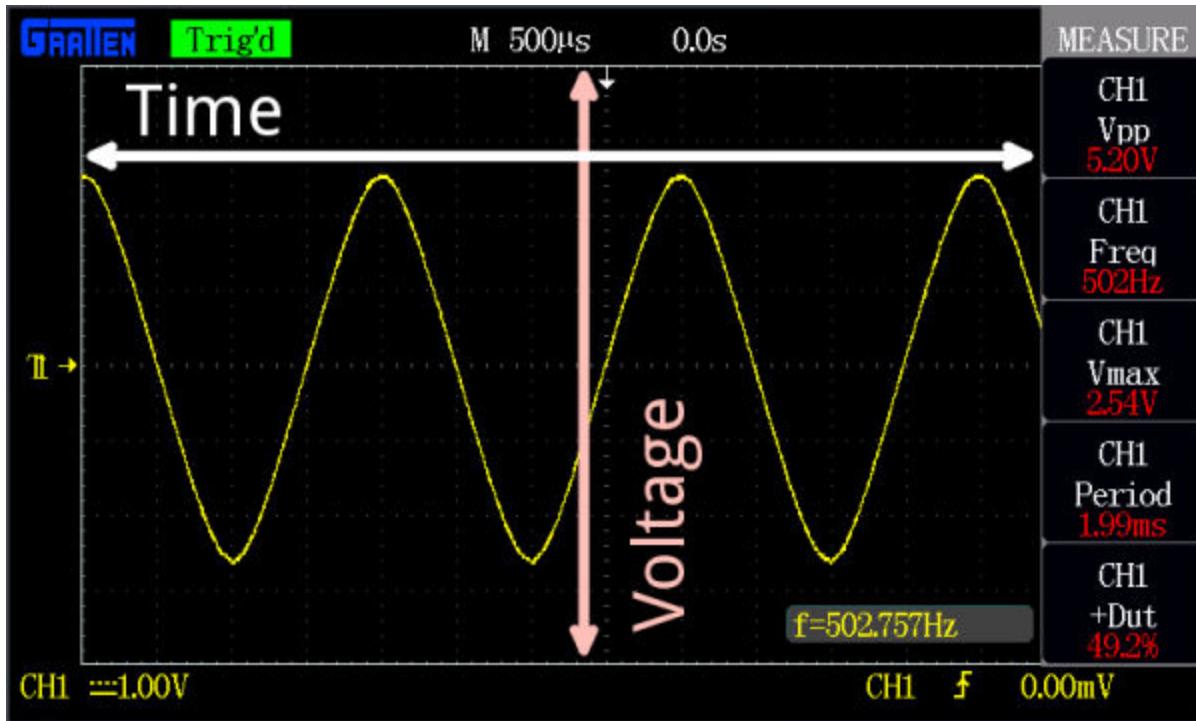
## College Level:

### 15.3 - Basic Oscilloscope Operation

[reference - <https://learn.sparkfun.com/tutorials/how-to-use-an-oscilloscope/all>]

#### 15.3.1 - Introduction

The main purpose of an oscilloscope is to graph an electrical signal as it varies over time. Most scopes produce a two-dimensional graph with time on the x-axis and voltage on the y-axis.



Controls surrounding the scope's screen allow you to adjust the scale of the graph, both vertically and horizontally -- allowing you to zoom in and out on a signal. There are also controls to set the trigger on the scope, which helps focus and stabilize the display.

### 15.3.2 - What Can Scopes Measure

In addition to those fundamental features, many scopes have measurement tools, which help to quickly quantify frequency, amplitude, and other waveform characteristics. In general a scope can measure both time-based and voltage-based characteristics:

#### Timing characteristics:

**Frequency and period** -- Frequency is defined as the number of times per second a waveform repeats. And the period is the reciprocal of that (number of seconds each repeating waveform takes). The maximum frequency a scope can measure varies, but it's often in the 100's of MHz (1E6 Hz) range.

**Duty cycle** -- The percentage of a period that a wave is either positive or negative (there are both positive and negative duty cycles). The duty cycle is a ratio that tells you how long a signal is "on" versus how long it's "off" each period.

**Rise and fall time** -- Signals can't instantaneously go from 0V to 5V, they have to smoothly rise. The duration of a wave going from a low point to a high point is called the rise time, and fall time measures the opposite. These characteristics are important when considering how fast a circuit can respond to signals.

### Voltage characteristics:

**Amplitude** -- Amplitude is a measure of the magnitude of a signal. There are a variety of amplitude measurements including peak-to-peak amplitude, which measures the absolute difference between a high and low voltage point of a signal. Peak amplitude, on the other hand, only measures how high or low a signal is past 0V.

**Maximum and minimum voltages** -- The scope can tell you exactly how high and low the voltage of your signal gets.

**Mean and average voltages** -- Oscilloscopes can calculate the average or mean of your signal, and it can also tell you the average of your signal's minimum and maximum voltage.

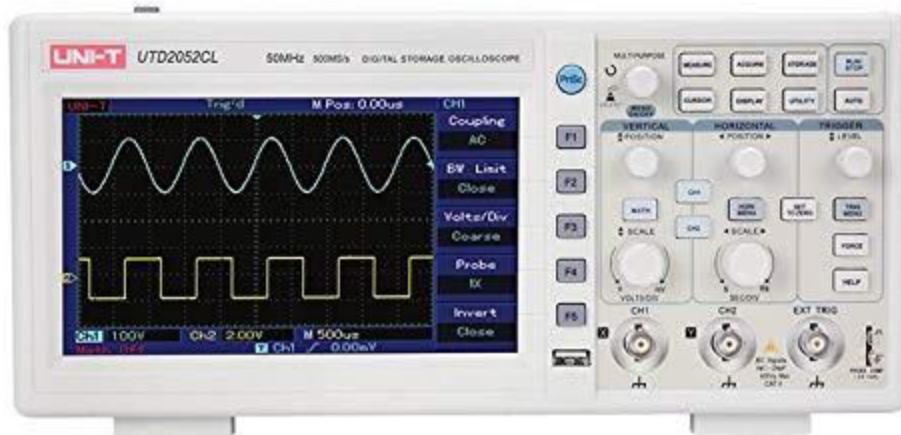
#### 15.3.3 - When to use oscilloscope

The o-scope is useful in a variety of troubleshooting and research situations, including:

- Determining the frequency and amplitude of a signal, which can be critical in debugging a circuit's input, output, or internal systems. From this, you can tell if a component in your circuit has malfunctioned.
- Identifying how much noise is in your circuit.
- Identifying the shape of a wave -- sine, square, triangle, sawtooth, complex, etc.
- Quantifying phase differences between two different signals.

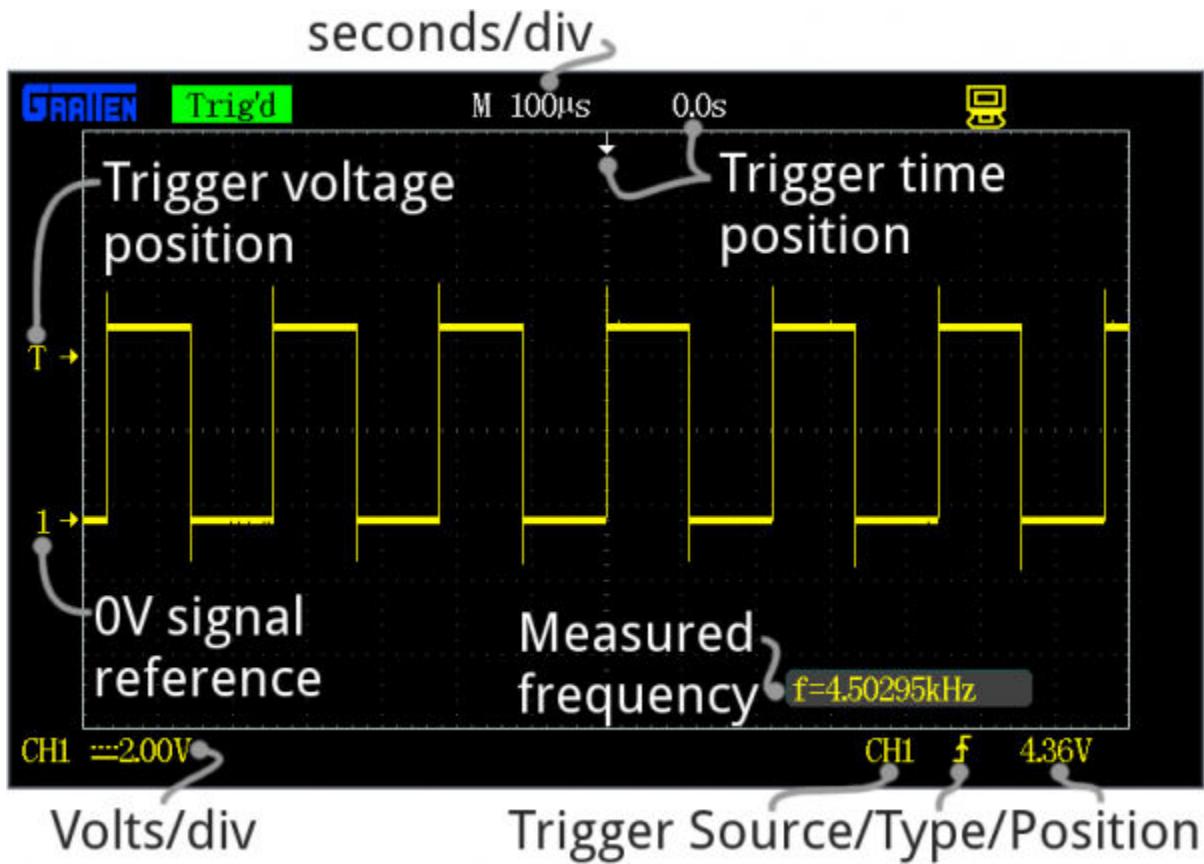
#### 15.3.4 - Anatomy of An O-Scope

While no scopes are created exactly equal, they should all share a few similarities that make them function similarly. On this page we'll discuss a few of the more common systems of an oscilloscope: the display, horizontal, vertical, trigger, and inputs.



#### 15.3.4.1 - The Display

An oscilloscope isn't any good unless it can display the information you're trying to test, which makes the display one of the more important sections on the scope.



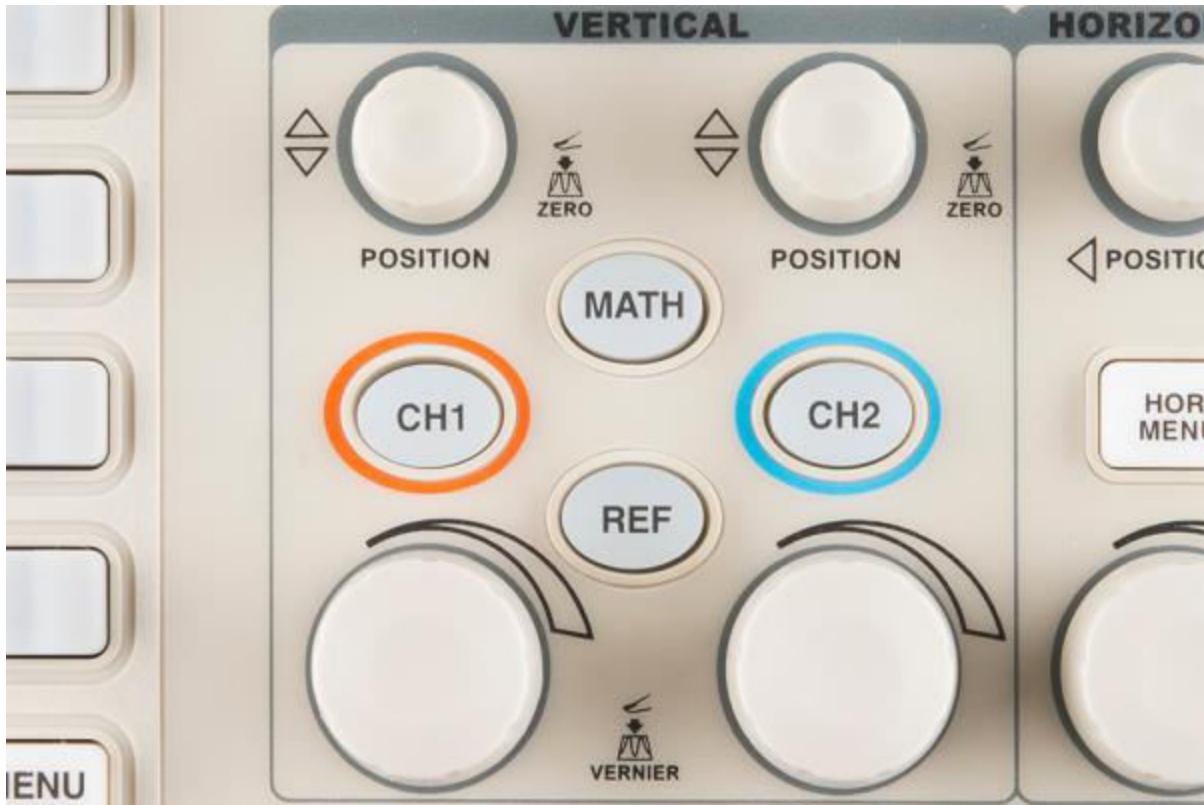
Every oscilloscope display should be criss-crossed with horizontal and vertical lines called divisions. The scale of those divisions are modified with the horizontal and vertical systems. The vertical system is measured in “volts per division” and the horizontal is “seconds per division”. Generally, scopes will feature around 8-10 vertical (voltage) divisions, and 10-14 horizontal (seconds) divisions.

Older scopes (especially those of the analog variety) usually feature a simple, monochrome display, though the intensity of the wave may vary. More modern scopes feature multicolor LCD screens, which are a great help in showing more than one waveform at a time.

Many scope displays are situated next to a set of about five buttons -- either to the side or below the display. These buttons can be used to navigate menus and control settings of the scope.

#### 15.3.4.2 - Vertical System

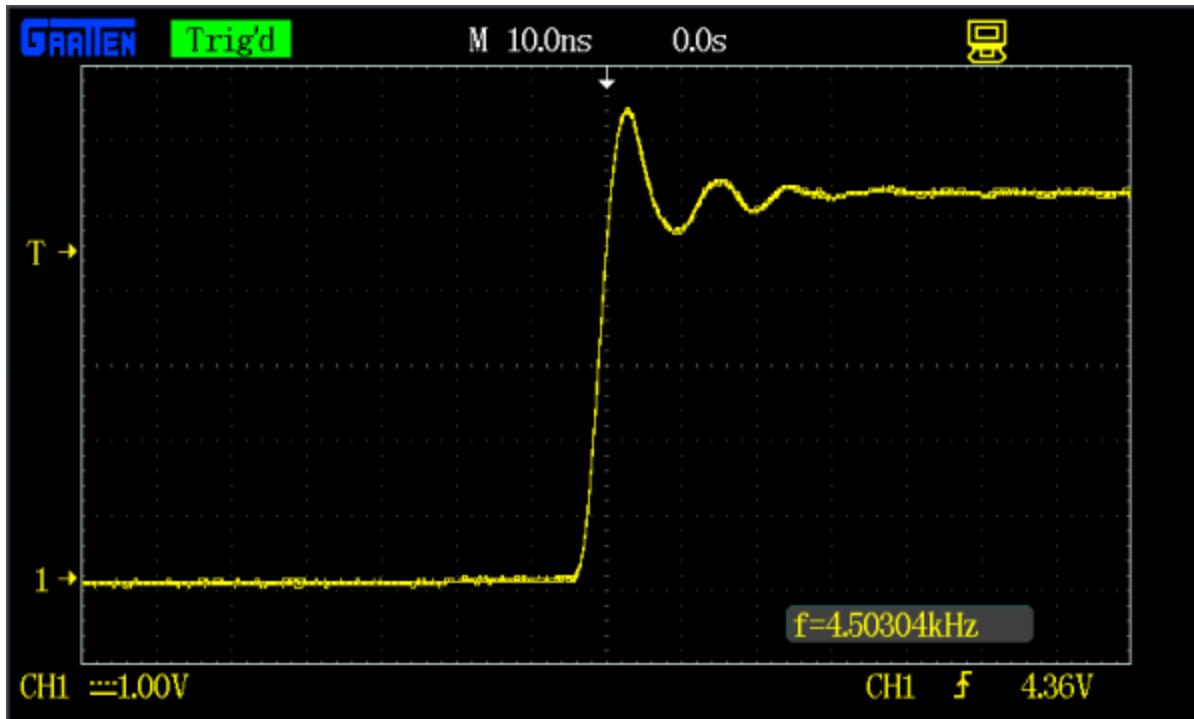
The vertical section of the scope controls the voltage scale on the display. There are traditionally two knobs in this section, which allow you to individually control the vertical position and volts/div.



The more critical volts per division knob allows you to set the vertical scale on the screen. Rotating the knob clockwise will decrease the scale, and counter-clockwise will increase. A smaller scale – fewer volts per division on the screen – means you’re more “zoomed in” to the waveform.

The display on the GA1102, for example, has 8 vertical divisions, and the volts/div knob can select a scale between 2mV/div and 5V/div. So, zoomed all the way in to 2mV/div, the display can show waveform that is 16mV from top to bottom. Fully “zoomed out”, the scope can show a waveform ranging over 40V. (The probe, as we’ll discuss below, can further increase this range.)

The position knob controls the vertical offset of the waveform on the screen. Rotate the knob clockwise, and the wave will move down, counter-clockwise will move it up the display. You can use the position knob to offset part of a waveform off the screen



Using both the position and volts/div knobs in conjunction, you can zoom in on just a tiny part of the waveform that you care about the most. If you had a 5V square wave, but only cared about how much it was ringing on the edges, you could zoom in on the rising edge using both knobs.

#### 15.3.4.3 - Horizontal System

The horizontal section of the scope controls the time scale on the screen. Like the vertical system, the horizontal control gives you two knobs: position and seconds/div.

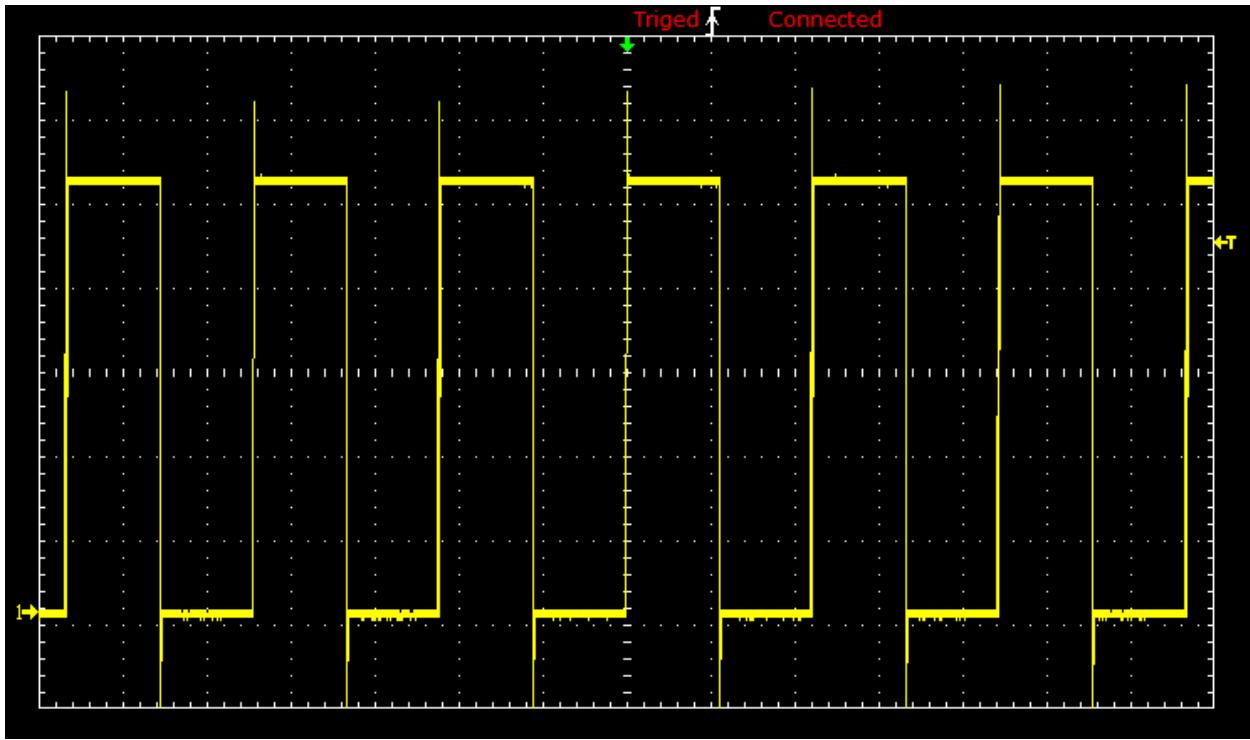


The seconds per division (s/div) knob rotates to increase or decrease the horizontal scale. If you rotate the s/div knob clockwise, the number of seconds each division represents will decrease – you'll be “zooming in” on the time scale. Rotate counter-clockwise to increase the time scale, and show a longer amount of time on the screen.

Using the GA1102 as an example again, the display has 14 horizontal divisions, and can show anywhere between 2nS and 50s per division. So zoomed all the way in on the horizontal scale, the scope can show 28nS of a waveform, and zoomed way out it can show a signal as it changes over 700 seconds.

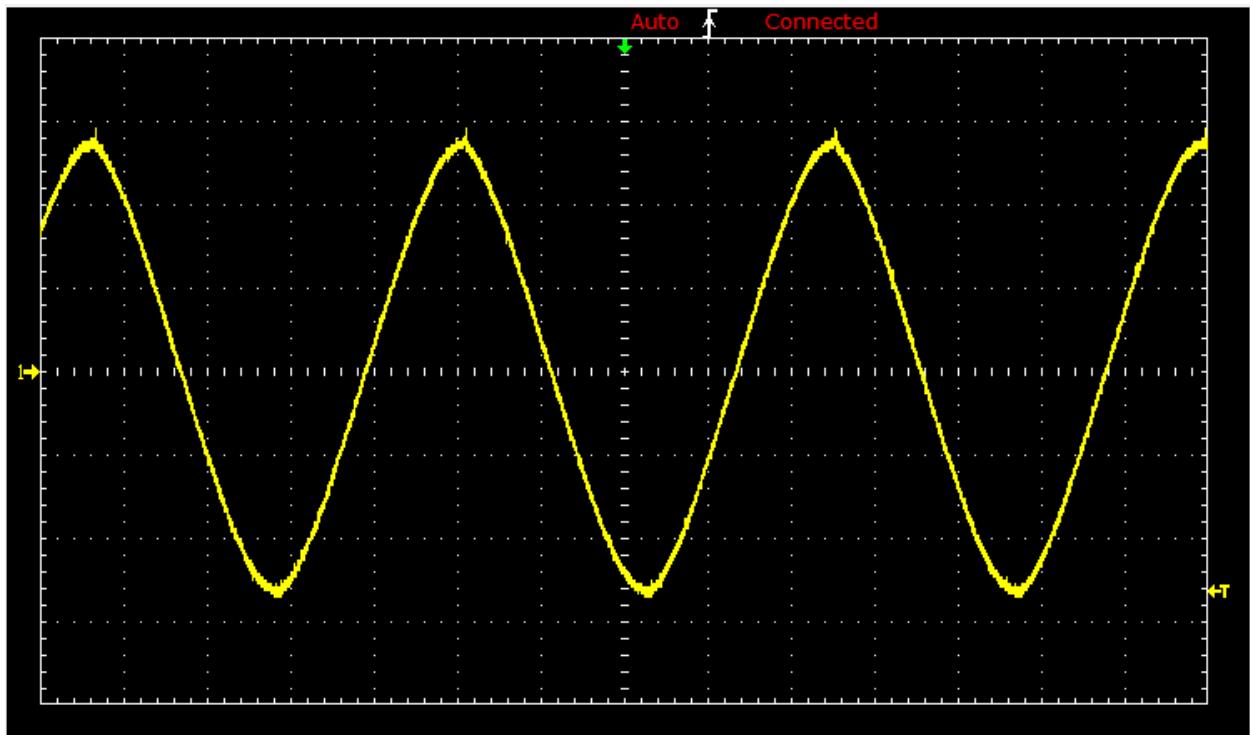
The position knob can move your waveform to the right or left of the display, adjusting the horizontal offset.

Using the horizontal system, you can adjust how many periods of a waveform you want to see. You can zoom out, and show multiple peaks and troughs of a signal:



#### 15.3.4.4 - Trigger System

The trigger section is devoted to stabilizing and focusing the oscilloscope. The trigger tells the scope what parts of the signal to “trigger” on and start measuring. If your waveform is periodic, the trigger can be manipulated to keep the display static and unflinching. A poorly triggered wave will produce seizure-inducing sweeping waves like this:



The trigger section of a scope is usually comprised of a level knob and a set of buttons to select the source and type of the trigger. The level knob can be twisted to set a trigger to a specific voltage point.



A series of buttons and screen menus make up the rest of the trigger system. Their main purpose is to select the trigger source and mode. There are a variety of trigger types, which manipulate how the trigger is activated:

- An **edge trigger** is the most basic form of the trigger. It will key the oscilloscope to start measuring when the signal voltage passes a certain level. An edge trigger can be set to catch on a rising or falling edge (or both).
- A **pulse trigger** tells the scope to key in on a specified “pulse” of voltage. You can specify the duration and direction of the pulse. For example, it can be a tiny blip of 0V → 5V → 0V, or it can be a seconds-long dip from 5V to 0V, back to 5V.
- A **slope trigger** can be set to trigger the scope on a positive or negative slope over a specified amount of time.
- More complicated triggers exist to focus on standardized waveforms that carry video data, like NTSC or PAL. These waves use a unique synchronizing pattern at the beginning of every frame.

You can also usually select a triggering mode, which, in effect, tells the scope how strongly you feel about your trigger. In automatic trigger mode, the scope can attempt to draw your waveform even if it doesn't trigger. Normal mode will only draw your wave if it sees the specified trigger. And single mode looks for your specified trigger, when it sees it it will draw your wave then stop.

#### 15.3.4.5 - Oscilloscope Probe

An oscilloscope is only good if you can actually connect it to a signal, and for that you need probes. Probes are single-input devices that route a signal from your circuit to the scope. They have a sharp tip which probes into a point on your circuit. The tip can also be equipped with hooks, tweezers or clips to make latching onto a circuit easier. Every probe also includes a ground clip, which should be secured safely to a common ground point on the circuit under test.



Most probes have a  $9M\Omega$  resistor for attenuating, which, when combined with a standard  $1M\Omega$  input impedance on a scope, creates a  $1/10$  voltage divider. These probes are commonly called 10X attenuated probes. Many probes include a switch to select between 10X and 1X (no attenuation). Attenuated probes are great for improving accuracy at high frequencies, but they will also reduce the amplitude of your signal. If you're trying to measure a very low-voltage signal, you may have to go with a 1X probe. You may also need to select a setting on your scope to tell it you're using an attenuated probe, although many scopes can automatically detect this.



### 15.3.5 - Using an Oscilloscope

The infinite variety of signals out there means you'll never operate an oscilloscope the same way twice. But there are some steps you can count on performing just about every time you test a circuit. On this page we'll show an example signal, and the steps required to measure it.

#### 15.3.5.1 - Probe Selection and Setup

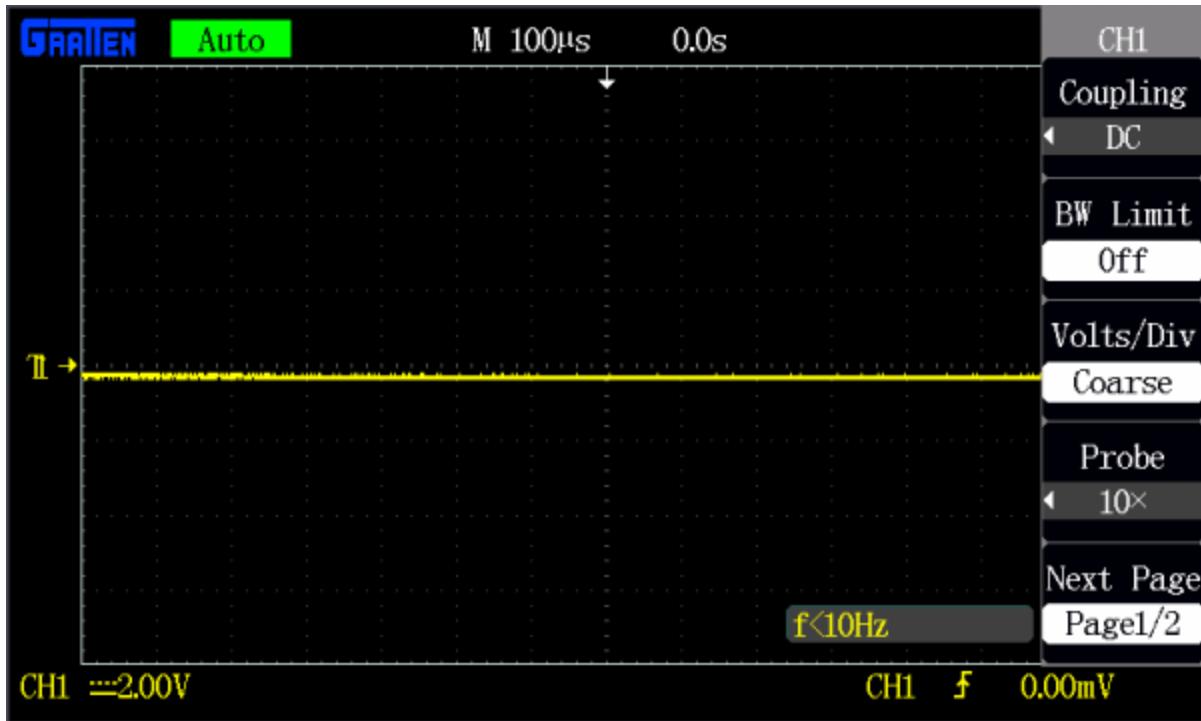
First off, you'll need to select a probe. For most signals, the simple passive probe included with your scope will work perfectly fine.

Next, before connecting it to your scope, set the attenuation on your probe. 10X -- the most common attenuation factor -- is usually the most well-rounded choice. If you're trying to measure a very low-voltage signal though, you may need to use 1X.

#### 15.3.5.2 - Connect the Probe and Turn the Scope On

Connect your probe to the first channel on your scope, and turn it on. Have some patience here, some scopes take as long to boot up as an old PC.

When the scope boots up you should see the divisions, scale, and a noisy, flat line of a waveform



The screen should also show previously set values for time and volts per div. Ignoring those scales for now, make these adjustments to put your scope into a standard setup:

- Turn channel 1 on and channel 2 off.
- Set channel 1 to DC coupling.
- Set the trigger source to channel 1 -- no external source or alternate channel triggering.
- Set the trigger type to rising edge, and the trigger mode to auto (as opposed to single).
- Make sure the scope probe attenuation on your scope matches the setting on your probe (e.g. 1X, 10X).

#### **15.3.5.3 -Testing the Probe**

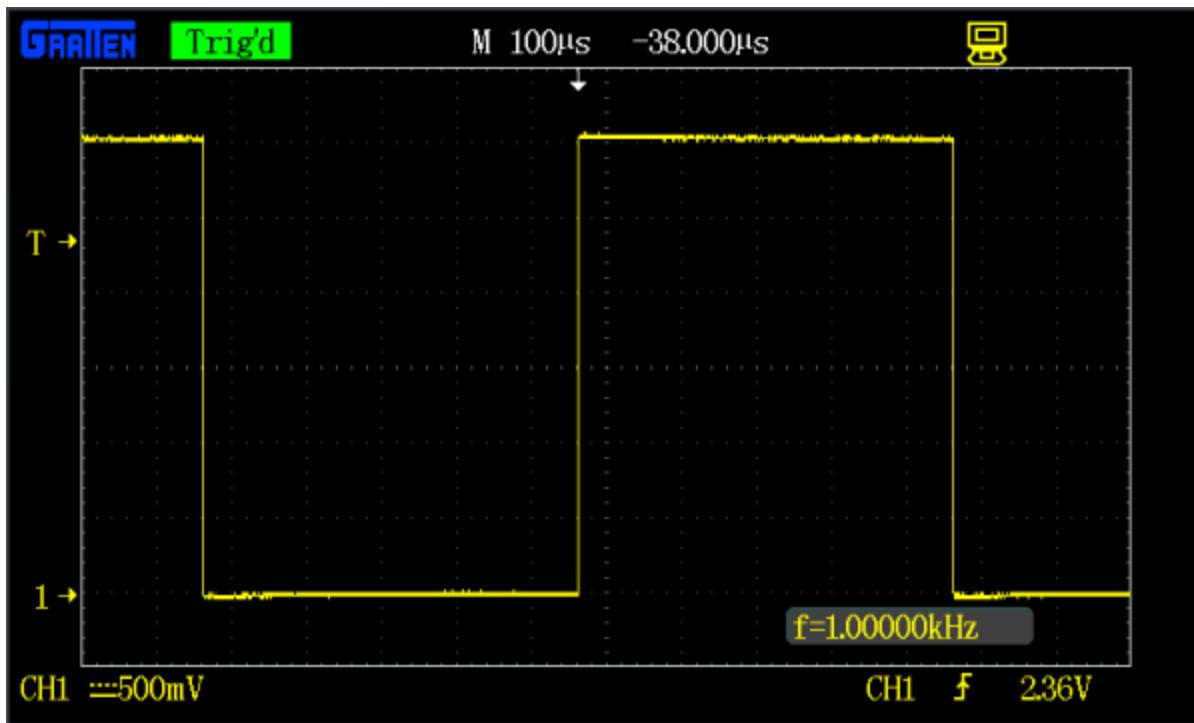
Let's connect that channel up to a meaningful signal. Most scopes will have a built-in frequency generator that emits a reliable, set-frequency wave -- on the GA1102CAL there is a 1kHz square wave output at the bottom-right of the front panel. The frequency generator output has two separate conductors -- one for the signal and one for ground. Connect your probe's ground clip to the ground, and the probe tip to the signal output.



As soon as you connect both parts of the probe, you should see a signal begin to dance around your screen. Try fiddling with the horizontal and vertical system knobs to maneuver the waveform around the screen. Rotating the scale knobs clockwise will "zoom into" your waveform, and counter-clockwise zooms out. You can also use the position knob to further locate your waveform.

If your wave is still unstable, try rotating the trigger position knob. Make sure the trigger isn't higher than the tallest peak of your waveform. By default, the trigger type should be set to edge, which is usually a good choice for square waves like this.

Try fiddling with those knobs enough to display a single period of your wave on the screen.



Now you are ready to measure. See this link for details :

<https://learn.sparkfun.com/tutorials/how-to-use-an-oscilloscope/all#screen>

## Class 16:

- 16.1 - Introduction to RFID
- 16.2 - Introduction to Fingerprint Sensor
- 16.3 - Do it yourself / Homework
- College Level :**
- 16.4 - Introduction to IoT

### 16.1 - Introduction to RFID

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

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

RFID Module Link : <https://www.facebook.com/commerce/products/2182106941854155/>

## 16.2 - Introduction to Fingerprint Sensor

Fingerprint sensor modules, like the one in the following figure, made fingerprint recognition more accessible and easy to add to your projects. This means that it is super easy to make fingerprint collection, registration, comparison and search. These modules come with FLASH memory to store the fingerprints and work with any microcontroller or system with TTL serial. These modules can be added to security systems, door locks, time attendance systems, and much more.

Fingerprint Module Link: <https://www.facebook.com/commerce/products/2245226832162914/>



## 16.3 - Do it yourself / Homework

- 1) Interface Soil Hygrometer

Link: <https://www.facebook.com/commerce/products/2833406566700084/>

- 2) Interface Digital Barometer

Link: <https://www.facebook.com/commerce/products/2315599445150290/>

- 3) Interface LDR with Arduino

Link : <https://www.facebook.com/commerce/products/2159341974141502/>

- 4) PWM & Servo Control from Processing

Link : <https://youtu.be/6wCFE6ZnVWI>

## 16.4 - Introduction to IoT

[https://en.wikipedia.org/wiki/Internet\\_of\\_things](https://en.wikipedia.org/wiki/Internet_of_things)

<https://internetofthingsagenda.techtarget.com/definition/Internet-of-Things-IoT>

The Internet of Things (IoT) is the network of physical devices, vehicles, home appliances, and other items embedded with electronics, software, sensors, actuators, and connectivity which enables these things to connect and exchange data, creating opportunities for more direct integration of the physical world into computer-based systems, resulting in efficiency improvements, economic benefits, and reduced human exertions. The number of IoT devices increased 31% year-over-year to 8.4 billion in 2017 and it is estimated that there will be 30 billion devices by 2020. The global market value of IoT is projected to reach \$7.1 trillion by 2020. IoT involves extending internet connectivity beyond standard devices, such as desktops, laptops, smartphones and tablets, to any range of traditionally dumb or non-internet-enabled physical devices and everyday objects. Embedded with technology, these devices can communicate and interact over the internet, and they can be remotely monitored and controlled.

To learn about IoT Follow : [https://github.com/Nabilphysics/ESP32\\_IoT\\_long\\_course\\_crux/wiki](https://github.com/Nabilphysics/ESP32_IoT_long_course_crux/wiki)

**Arduino Project From Internet :**

**Intermediate :**

1) Sonar Sensor(HC-SR04) with LCD

web: <https://howtomechatronics.com/tutorials/arduino/ultrasonic-sensor-hc-sr04/>

youtube: <https://www.youtube.com/watch?v=ZejQOX69K5M>

**Advanced :**

1) LCD(16\*2) Menu System

web: <http://eeenthusiast.com/arduino-lcd-tutorial-display-menu-system-scrolling-menu-changeable-variables-projects/>

youtube : <https://www.youtube.com/watch?v=Q58mQFwWv7c>