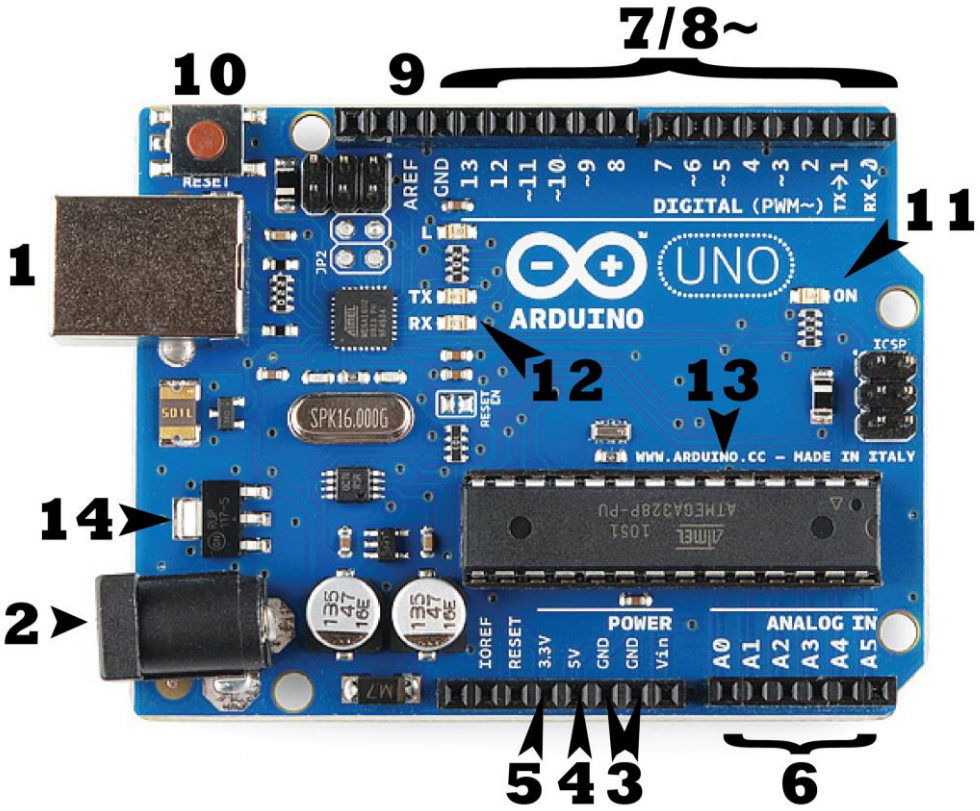
Experiment No. 1

Objective: Introduction to Arduino platform and programming and introduction to various sensors and various actuators & its application

Theory:

Arduino is an open-source prototyping platform in electronics based on easy-to-use hardware and software. Subtly speaking, Arduino is a microcontroller based prototyping board which can be used in developing digital devices that can read inputs like finger on a button, touch on a screen, light on a sensor etc. and turning it in to output like switching on an LED, rotating a motor, playing songs through a speaker etc.

The Arduino board can be programmed to do anything by simply programming the microcontroller on board using a set of instructions for which, the Arduino board consists of a USB plug to communicate with your computer and a bunch of connection sockets that can be wired to external devices like motors, LEDs etc.



Power (USB / Barrel Jack)

Every Arduino board needs a way to be connected to a power source. The Arduino UNO can be powered from a USB cable coming from your computer or a wall power supply (like this) that is terminated in a barrel jack. In the picture above the USB connection is labeled (1) and the barrel jack is labeled (2). The USB connection is also how you will load code onto your Arduino board. More on how to program with Arduino can be found in our Installing and Programming Arduino tutorial. NOTE: Do NOT use a power supply greater than 18 Volts as you will overpower (and thereby destroy) your Arduino. The recommended voltage for most Arduino models is between 6 and 12 Volts.

Pins (5V, 3.3V, GND, Analog, Digital, PWM, ARE)

The pins on your Arduino are the places where you connect wires to construct a circuit (probably in conjuction with a breadboard and some wire. They usually have black plastic

'headers' that allow you to just plug a wire right into the board. The Arduino has several different kinds of pins, each of which is labeled on the board and used for different functions.

- GND (3): Short for 'Ground'. There are several GND pins on the Arduino, any of which can be used to ground your circuit.
- 5V (4) & 3.3V (5): As you might guess, the 5V pin supplies 5 volts of power, and the 3.3V pin supplies 3.3 volts of power. Most of the simple components used with the Arduino run happily off of 5 or 3.3 volts.
- Analog (6): The area of pins under the 'Analog In' label (A0 through A5 on the UNO) are Analog In pins. These pins can read the signal from an analog sensor (like a temperature sensor) and convert it into a digital value that we can read.
- Digital (7): Across from the analog pins are the digital pins (0 through 13 on the UNO). These pins can be used for both digital input (like telling if a button is pushed) and digital output (like powering an LED).
- PWM (8): You may have noticed the tilde (~) next to some of the digital pins (3, 5, 6, 9, 10, and 11 on the UNO). These pins act as normal digital pins, but can also be used for something called Pulse-Width Modulation (PWM).
- AREF (9): Stands for Analog Reference. Most of the time you can leave this pin alone. It is sometimes used to set an external reference voltage (between 0 and 5 Volts) as the upper limit for the analog input pins.

Reset Button

the Arduino has a reset button (10). Pushing it will temporarily connect the reset pin to ground and restart any code that is loaded on the Arduino.

Power LED Indicator

Just beneath and to the right of the word "UNO" on your circuit board, there's a tiny LED next to the word 'ON' (11). This LED should light up whenever you plug your Arduino into a power source. If this light doesn't turn on, there's a good chance something is wrong. Time to re-check your circuit!

TX RX LEDs

TX is short for transmit, RX is short for receive. These markings appear quite a bit in electronics to indicate the pins responsible for serial communication. In our case, there are two places on the Arduino UNO where TX and RX appear - once by digital pins 0 and 1, and a second time next to the TX and RX indicator LEDs (12). These LEDs will give us some nice visual indications whenever our Arduino is receiving or transmitting data (like when we're loading a new program onto the board).

Main IC

Main IC The black thing with all the metal legs is an IC, or Integrated Circuit (13). Think of it as the brains of our Arduino. The main IC on the Arduino is slightly different from board type to board type, but is usually from the ATmega line of IC's from the ATMEL company. This can be important, as you may need to know the IC type (along with your board type) before loading up a new program from the Arduino software. This information can usually be found in writing on the top side of the IC.

Voltage Regulator

Voltage Regulator The voltage regulator (14) is not actually something you can (or should) interact with on the Arduino. But it is potentially useful to know that it is there and what it's for. The voltage regulator does exactly what it says - it controls the amount of voltage that is let into the Arduino board. Think of it as a kind of gatekeeper; it will turn away an extra voltage that might harm the circuit. Of course, it has its limits, so don't hook up your Arduino to anything greater than 18 volts.

Arduino Programming Language

The Arduino Programming Language is basically a framework built on top of C++. It's not a real programming language.

A program written in the Arduino Programming Language is called **sketch**. A sketch is normally saved with the .ino extension (from Arduino).

The main difference from "normal" C or C++ is that you wrap all your code into 2 main functions.

One is called **setup(),** the other is called **loop().** The first is called once, when program starts, the second is repeatedly called while your program is running.

We don't have a main() function like you are used to in C/C++ as the entry point for a program. Once you compile your sketch, the IDE will make sure the end result is a correct C++ program and will basically add the missing glue by preprocessing it.

Hardware and Software

Arduino boards are generally based on microcontrollers from Atmel Corporation like 8, 16 or 32 bit AVR architecture based microcontrollers.

The important feature of the Arduino boards is the standard connectors. Using these connectors, we can connect the Arduino board to other devices like LEDs or add-on modules called Shields.

The Arduino boards also consists of on board voltage regulator and crystal oscillator. They also consist of USB to serial adapter using which the Arduino board can be programmed using USB connection.

In order to program the Arduino board, we need to use IDE provided by Arduino. The Arduino IDE is based on Processing programming language and supports C and C++.

Types of Arduino Boards

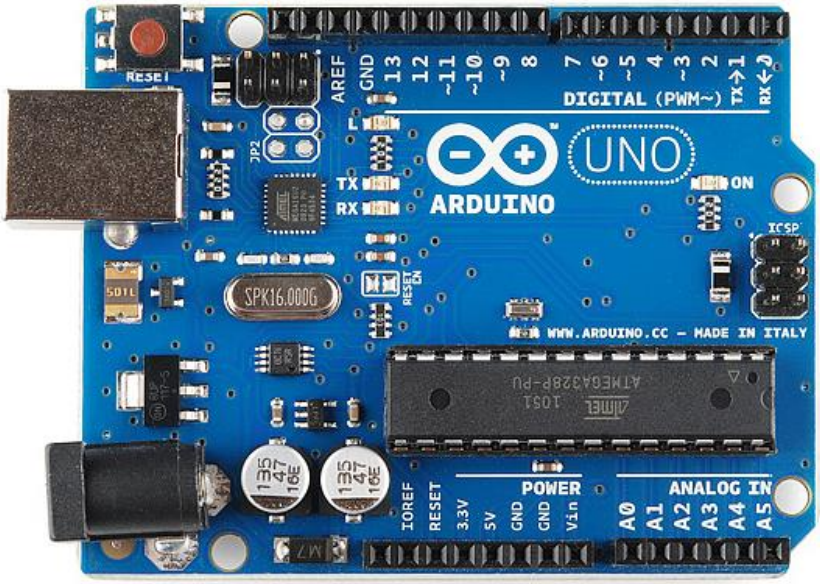
There are many types of Arduino boards available in the market but all the boards have one thing in common: they can be programmed using the Arduino IDE. The reasons for different types of boards are different power supply requirements, connectivity options, their applications etc.

Arduino boards are available in different sizes, form factors, different no. of I/O pins etc. Some of the commonly known and frequently used Arduino boards are Arduino UNO, Arduino Mega, Arduino Nano, Arduino Micro and Arduino Lilypad.

There are add-on modules called Arduino Shields which can be used to extend the functionalities of the Arduino boards. Some of the commonly used shields are Arduino Proto shield, Arduino WiFi Shield and Arduino Yun Shield.

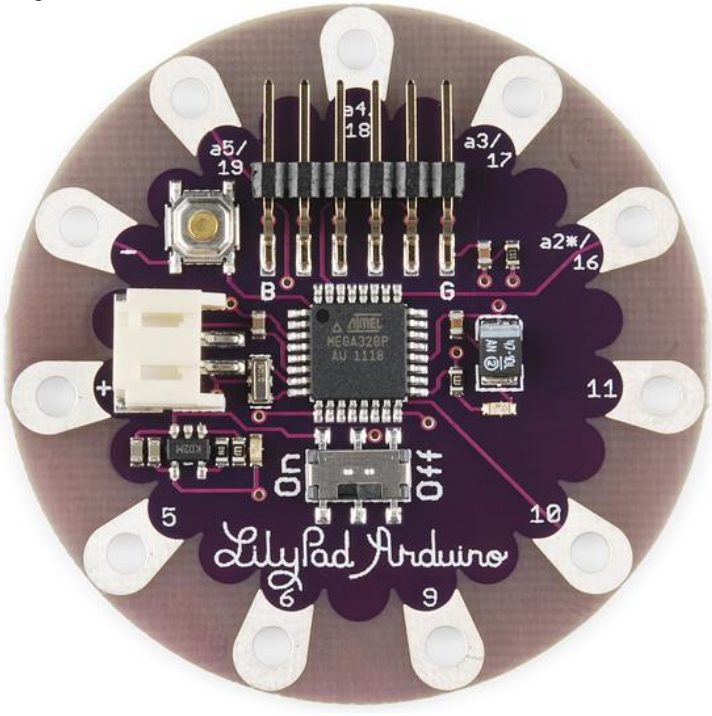
Arduino Uno (R3)

The Uno is a huge option for your initial Arduino. It consists of 14-digital I/O pins, where 6-pins can be used as PWM(pulse width modulation outputs), 6-analog inputs, a reset button, a power jack, a USB connection and more. It includes everything required to hold up the microcontroller; simply attach it to a PC with the help of a USB cable and give the supply to get started with a AC-to-DC adapter or battery.



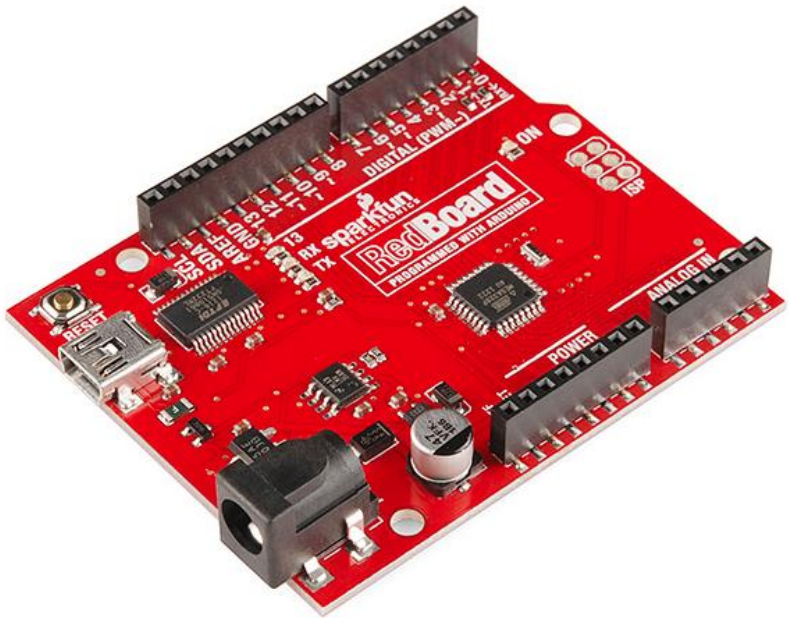
LilyPad Arduino Board

The Lily Pad Arduino board is a wearable e-textile technology expanded by Leah "Buechley"and considerately designed by "Leah and SparkFun". Each board was imaginatively designed with huge connecting pads & a smooth back to let them to be sewn into clothing using conductive thread. This Arduino also comprises of I/O, power, and also sensor boards which are built especially for e-textiles. These are even washable!



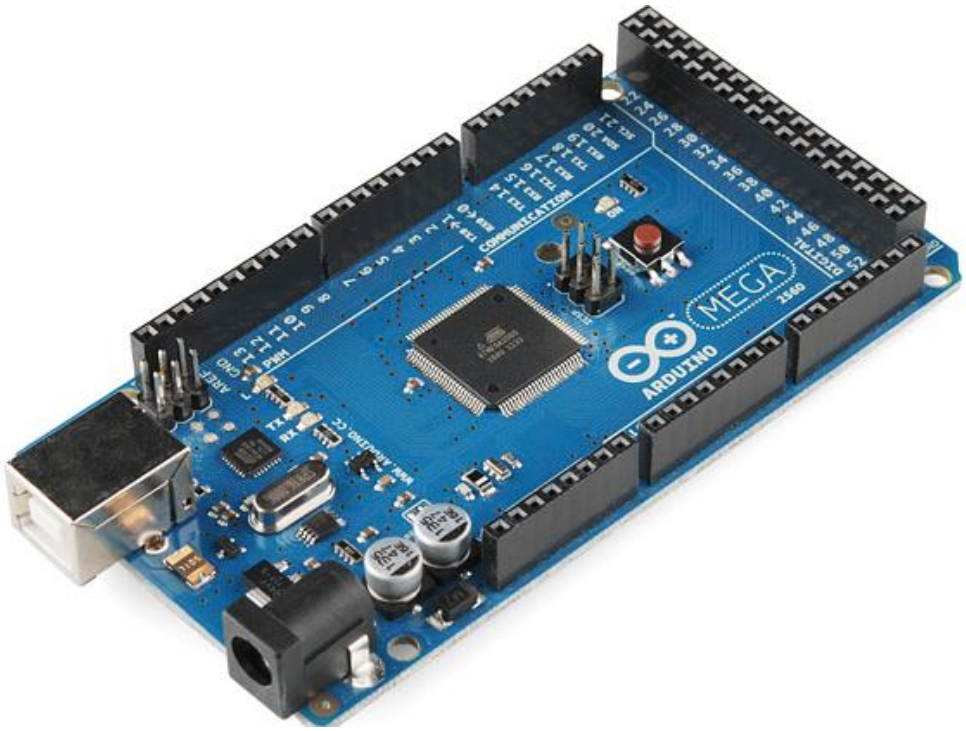
RedBoard Arduino Board

The RedBoard aAduino board can be programmed using a Mini-B USB cable using the Arduino IDE. It will work on Windows 8 without having to modify your security settings.It is more constant due to the USB or FTDI chip we used and also it is entirely flat on the back. Creating it is very simple to utilize in the project design. Just plug the board, select the menu option to choose an Arduino UNO and you are ready to upload the program. You can control the RedBoard over USB cable using the barrel jack.



Arduino Mega (R3) Board

The Arduino Mega is similar to the UNO's big brother. It includes lots of digital I/O pins (from that, 14-pins can be used as PWM o/ps), 6-analog inputs, a reset button, a power jack, a USB connection and a reset button. It includes everything required to hold up the microcontroller; simply attach it to a PC with the help of a USB cable and give the supply to get started with a AC-to-DC adapter or battery.The huge number of pins make this Arduino board very helpful for designing the projects that need a bunch of digital i/ps or o/ps like lots buttons.



Arduino Leonardo Board

The first development board of an Arduino is the Leonardo board. This board uses one microcontroller along with the USB. That means, it can be very simple and cheap also. Because this board handles USB directly, program libraries are obtainable which let the Arduino board to follow a keyboard of the computer, mouse, etc.

Introduction to Sensors and Actuators:

Sensors and actuators are two critical components of every closed loop control system. Such a system is also called a mechatronics system. Mechatronics system consists of a sensing unit, a controller, and an actuating unit. Sensing unit consist of additional components such as filters, amplifiers, modulators, and other signal conditioners. Controller accepts information from sensing unit, makes decisions based on control algorithm, and outputs commands to actuating unit. Actuating unit consists of an actuator and optionally a power supply and a coupling mechanism.

What is a Sensors?

Device that when exposed to a physical phenomenon (temperature, displacement, force, etc.) produces a proportional output signal (electrical, mechanical, magnetic, etc.). Transducer synonymous with sensors. Sensor is a device that responds to a change in physical phenomenon while transducer is a device that converts one form of energy into another form of energy. Sensors are transducers when they sense one form of energy input and output in a different form of energy. For example, a thermocouple responds to a temperature change (thermal energy) and outputs a proportional change in electromotive force (electrical energy). Therefore, a thermocouple can be called a sensor or transducer

What is an Actuator:

Actuators are devices which drive a machine (robot) including its grippers. Muscles of a human arm and hand. While human arm provides motion, hand is used for object manipulation. So, actuators in robots (machine) provides motion while grippers manipulates objects. An actuator system comprises of several subsystems, namely,

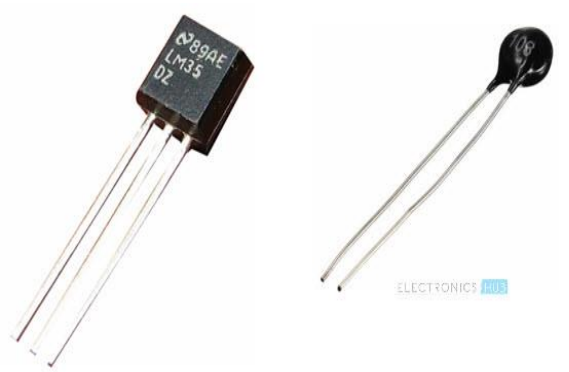
(i) Power supply; (ii) Power amplifier;
(iii) Servomotor; (iv) Transmission system.

Different Types of Sensors

The following is a list of different types of sensors that are commonly used in various applications. All these sensors are used for measuring one of the physical properties like Temperature, Resistance, Capacitance, Conduction, Heat Transfer etc.

1. Temperature Sensor
2. Proximity Sensor
3. Accelerometer
4. IR Sensor (Infrared Sensor)
5. Pressure Sensor
6. Light Sensor
7. Ultrasonic Sensor
8. Smoke, Gas and Alcohol Sensor
9. Touch Sensor
10. Color Sensor
11. Humidity Sensor
12. Position Sensor
13. Magnetic Sensor (Hall Effect Sensor)
14. Microphone (Sound Sensor)
15. Tilt Sensor
16. Flow and Level Sensor
17. PIR Sensor
18. Touch Sensor
19. Strain and Weight Sensor

Temperature Sensor: One of the most common and most popular sensors is the Temperature Sensor. A Temperature Sensor, as the name suggests, senses the temperature i.e., it measures the changes in the temperature.



LM35 - Temperature Sensor IC 10KΩ NTC Thermistor

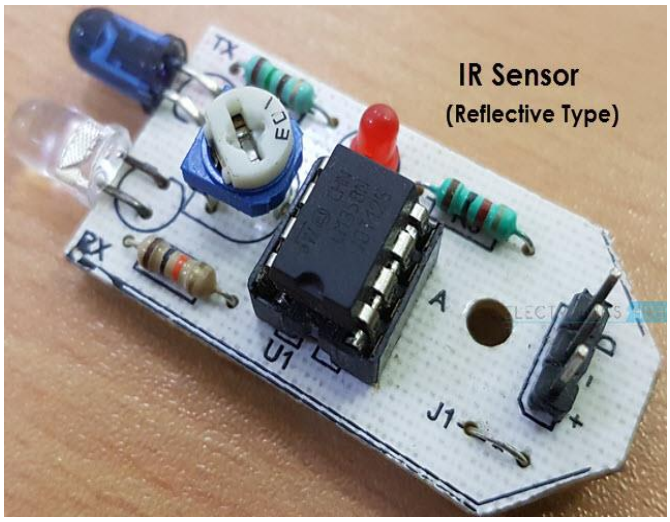
There are different types of Temperature Sensors like Temperature Sensor ICs (like LM35, DS18B20), Thermistors, Thermocouples, RTD (Resistive Temperature Devices), etc.
Temperature Sensors can be analog or digital. In an Analog Temperature Sensor, the changes in the Temperature correspond to change in its physical property like resistance or voltage. LM35 is a classic Analog Temperature Sensor. Digital Temperature Sensor, the output is a discrete digital value (usually, some numerical data after converting analog value to digital value). DS18B20 is a simple Digital Temperature Sensor. Temperature Sensors are used everywhere like computers, mobile phones, automobiles, air conditioning systems, industries etc.

Proximity Sensors: A Proximity Sensor is a non-contact type sensor that detects the presence of an object. Proximity Sensors can be implemented using different techniques like Optical (like Infrared or Laser), Sound (Ultrasonic), Magnetic (Hall Effect), Capacitive, etc.

Inductive Proximity Sensor

Some of the applications of Proximity Sensors are Mobile Phones, Cars (Parking Sensors), industries (object alignment), Ground Proximity in Aircrafts, etc.
Proximity Sensor in Reverse Parking is implemented in this Project: REVERSE PARKING SENSOR CIRCUIT.

Infrared Sensor (IR Sensor): IR Sensors or Infrared Sensor are light based sensor that are used in various applications like Proximity and Object Detection. IR Sensors are used as proximity sensors in almost all mobile phones.



There are two types of Infrared or IR Sensors: Transmissive Type and Reflective Type. In Transmissive Type IR Sensor, the IR Transmitter (usually an IR LED) and the IR Detector (usually a Photo Diode) are positioned facing each other so that when an object passes between them, the sensor detects the object.
The other type of IR Sensor is a Reflective Type IR Sensor. In this, the transmitter and the detector are positioned adjacent to each other facing the object. When an object comes in front of the sensor, the infrared light from the IR Transmitter is reflected from the object and is detected by the IR Receiver and thus the sensor detects the object.
Different applications where IR Sensor is implemented are Mobile Phones, Robots, Industrial assembly, automobiles street light.

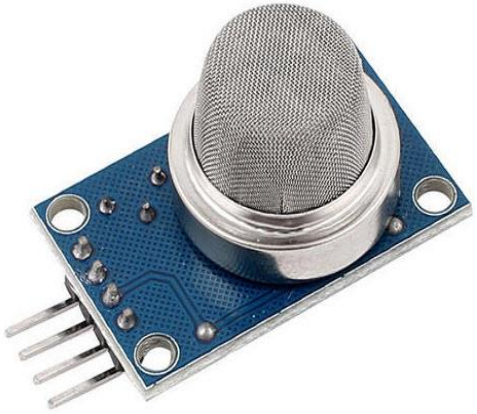
Ultrasonic Sensor: An Ultrasonic Sensor is a non-contact type device that can be used to measure distance as well as velocity of an object. An Ultrasonic Sensor works based on the properties of the sound waves with frequency greater than that of the human audible range.

Using the time of flight of the sound wave, an Ultrasonic Sensor can measure the distance of the object (similar to SONAR). The Doppler Shift property of the sound wave is used to measure the velocity of an object.
Arduino based Range Finder is a simple application using Ultrasonic Sensor

Light Sensor: Sometimes also known as Photo Sensors, Light Sensors are one of the important sensors. A simple Light Sensor available today is the Light Dependent Resistor or LDR. The property of LDR is that its resistance is inversely proportional to the intensity of the ambient light i.e., when the intensity of light increases, its resistance decreases and vise-versa.
By using LDR is a circuit, we can calibrate the changes in its resistance to measure the intensity of Light. There are two other Light Sensors (or Photo Sensors) which are often used

in complex electronic system design. They are Photo Diode and Photo Transistor. All these are **Analog Sensors.**

There are also **Digital Light Sensors** like BH1750, TSL2561, etc., which can calculate intensity of light and provide a digital equivalent value.
Smoke and Gas Sensors: One of the very useful sensors in safety related applications are Smoke and Gas Sensors. Almost all offices and industries are equipped with several smoke detectors, which detect any smoke (due to fire) and sound an alarm.
Gas Sensors are more common in laboratories, large scale kitchens and industries. They can detect different gases like LPG, Propane, Butane, Methane (CH4), etc.

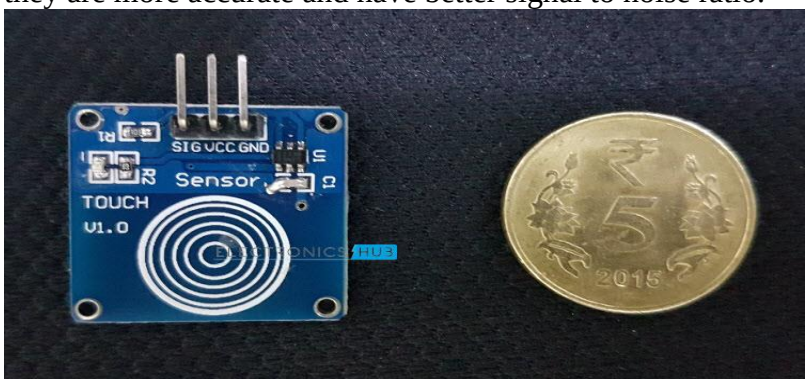


Now-a-days, smoke sensors (which often can detect smoke as well gas) are also installed in most homes as a safety measure.
The "MQ" series of sensors are a bunch of cheap sensors for detecting CO, CO2, CH4, Alcohol, Propane, Butane, LPG etc. You can use these sensors to build your own Smoke

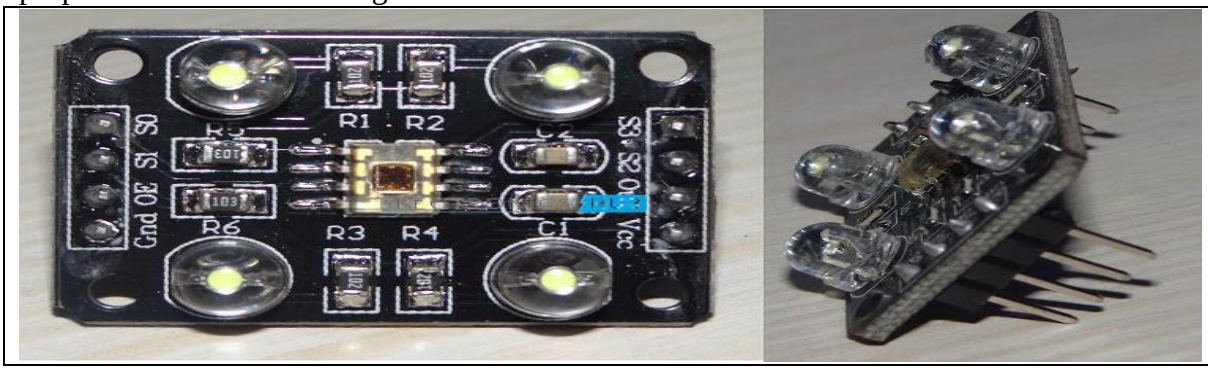
Sensor Application.
Alcohol Sensor: As the name suggests, an Alcohol Sensor detects alcohol. Usually, alcohol sensors are used in breathalyzer devices, which determine whether a person is drunk or not. Law enforcement personnel uses breathalyzers to catch drunk-and-drive culprits.

Touch Sensor: We do not give much importance to touch sensors but they became an integral part of our life. Whether you know or not, all touch screen devices (Mobile Phones, Tablets, Laptops, etc.) have touch sensors in them. Another common application of touch sensor is trackpads in our laptops.

Touch Sensors- detect touch of a finger or a stylus. Often touch sensors are classified into Resistive and Capacitive type. Almost all modern touch sensors are of Capacitive Types as they are more accurate and have better signal to noise ratio.



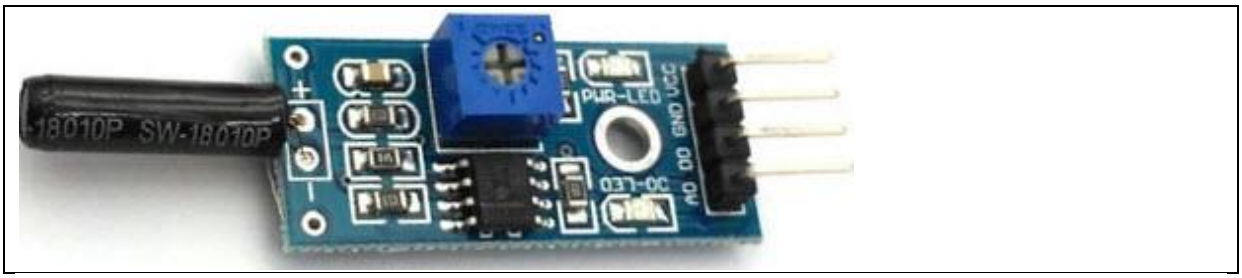
Color Sensor: A Color Sensor is an useful device in building color sensing applications in the field of image processing, color identification, industrial object tracking etc. The TCS3200 is a simple Color Sensor, which can detect any color and output a square wave proportional to the wavelength of the detected color.



Humidity Sensor: If you see Weather Monitoring Systems, they often provide temperature as well as humidity data. So, measuring humidity is an important task in many applications and Humidity Sensors help us in achieving this.
Often all humidity sensors measure relative humidity (a ratio of water content in air to maximum potential of air to hold water). Since relative humidity is dependent on temperature of air, almost all Humidity Sensors can also measure Temperature.

Humidity Sensors are classified into Capacitive Type, Resistive Type and Thermal Conductive Type. DHT11 and DHT22 are two of the frequently used Humidity Sensors in DIY Community (the former is a resistive type while the latter is capacitive type).

Tilt Sensor: Often used to detect inclination or orientation, Tilt Sensors are one of the simplest and inexpensive sensors out there. Previously, tilt sensors are made up of Mercury (and hence they are sometimes called as Mercury Switches) but most modern tilt sensors contain a roller ball.



Introduction to Actuator:

Actuators are mechanical or electro-mechanical devices that, upon being operated electrically, manually, or by various fluids, allow controlled and sometimes limited movements or positioning. They refer to that component of a machine that helps carry out the moving and controlling of a mechanism or system; take for instance opening a valve. To put it simply, they can be called movers.

Actuators basically need a control signal and a source of energy. Upon receiving a control signal, the actuator uses energy from the source to bring about a mechanical motion. The control system can be a human, a fixed mechanical or electronic system, or even software-based, say a printer driver, or a robot control system.

Examples of actuators include electric motors, stepper motors, electroactive polymers, screw jacks, servomechanism, solenoids and hydraulic cylinders.

Types of Actuators

1. **Mechanical actuators**

Mechanical actuators create movement by converting one kind of motion, such as rotary motion, into another kind, such as linear motion. Say for instance, a rack and a pinion. Another example is that of a chain block hoisting weight where the mechanical motion of the chain is used to lift a load. The functioning of mechanical actuators relies on the combinations of their structural components, such as gears and rails, or pulleys and chains. High reliability, simplicity of utilisation, easier maintenance and greater precision of positioning are some of the advantages. They can be categorised into hydraulic, pneumatic and electric actuators.

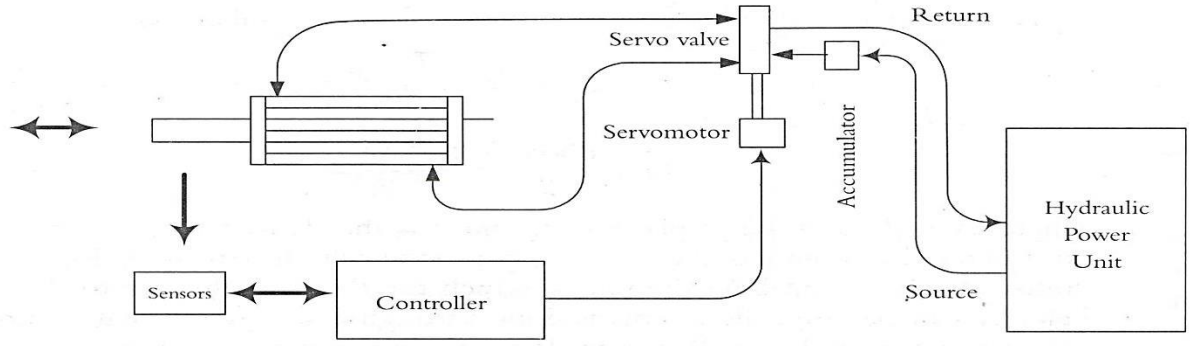
2. Hydraulic actuators

Hydraulic actuators have a cylinder or fluid motor that uses hydraulic power to generate mechanical motion, which in turn leads to linear, rotatory or oscillatory motion. Given the fact that liquids are nearly impossible to compress, a hydraulic actuator can exert a large force. When the fluid enters the lower chamber of the actuator's hydraulic cylinder, pressure inside increases and exerts a force on the bottom of the piston, also inside the cylinder. The pressure causes the sliding piston to move in a direction opposite to the force caused by the spring in the upper chamber, making the piston move upward and opening the valve. The downside with these actuators is the need for many complementary parts and possibility of fluid leakage.

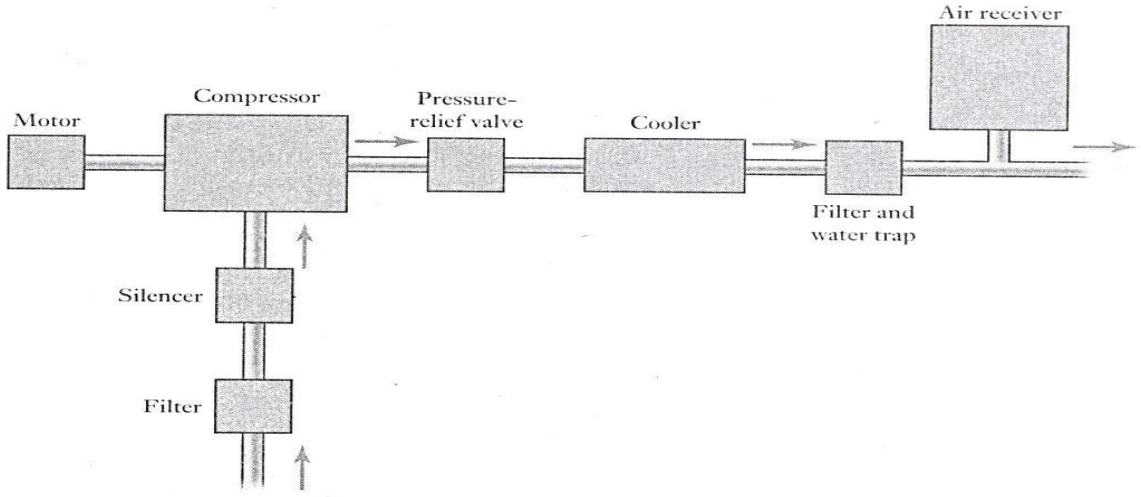
3. Pneumatic actuators

Pneumatic actuators convert energy in the form of compressed air into mechanical motion. Here pressurised gas or compressed air enters a chamber thus building up the pressure inside. Once this pressure goes above the required pressure levels in contrast to the atmospheric pressure outside the chamber, it makes the piston or gear move kinetically in a controlled manner, thus leading to a straight or circular mechanical motion. Examples include pneumatic cylinders, air cylinders, and air actuators. Cheaper and often more powerful than other actuators, they can quickly start or stop as no power source has to be stored in reserve for operation. Often used with valves to control the flow of air through the valve, these actuators generate considerable force through relatively small pressure changes.

Examples of maker projects using pneumatic actuators include lifting devices and humanoid robots with arms and limbs, typically used for lifting.



Hydraulic Actuator Setup



Pneumatic Actuator Setup

5. Electric Linear actuators

Now let us look at the electrical actuators, those that run on electricity.

Taking off from the two basic motions of linear and rotary, actuators can be classified into these two categories: linear and rotary. Electric linear actuators take electrical energy and turn it into straight line motions, usually for positioning applications, and they have a push and pull function. They convert energy from the power source into linear motion using mechanical transmission, electro-magnetism, or thermal expansion; they are typically used whenever tilting, lifting, pulling and pushing are needed. They are also known for offering precision and smooth motion control; this is why they are used in industrial machinery, in computer peripherals such as disk drives and printers, opening and closing dampers, locking doors and for braking machine motions. They are also used in 3d printers and for controlling valves. Some of them are unpowered and manually operated with a rotating knob or handwheel. Electric linear actuators

6. Electric Rotary actuators

Consisting of motors and output shaft mechanisms with limited rotary travel, electric rotary actuators convert electrical energy into rotary motion. Used in a wide range of industries where positioning is needed, and driven by various motor types, voice coils, these actuators work as per specifications such as the intended application, drive method, number of positions, output configuration, mounting configuration, physical dimensions and electrical characteristics. A common use is for controlling valves such as ball or butterfly valves. Other applications include automation applications where a gate, door or valve needs controlled movement to certain rotational positions.

7. Electromechanical actuators

Electromechanical actuators are mechanical actuators where there's an electric motor in place of the control knob or handle. The rotary motion of the motor leads to linear displacement. The inclined plane concept is what drives most electromechanical actuators; the lead screw's threads work like a ramp converting the small rotational force by magnifying it over a long distance, thus allowing a big load to be moved over a small distance. While there are many design variations among electromechanical actuators available today, most have the lead screw and the nut incorporated into the motion. The biggest advantages are their greater accuracy in relation to pneumatics, their longer lifecycle and low maintenance effort required. On the other hand, they do not boast the highest speed.

8.Electrohydraulic actuators

Instead of hydraulic systems, electrohydraulic actuators have self-contained actuators functioning solely on electrical power. They are basically used to actuate equipment such as multi-turn valves, or electric-powered construction and excavation equipment. In case of controlling the flow of fluid through a valve, a brake is typically installed above the motor to prevent the fluid pressure from forcing open the valve. The main advantage here is that these actuators help do away with the need for separate hydraulic pumps and tubing, simplifying system architectures and enhancing reliability and safety. Originally developed for the aerospace industry, today they are found in many other industries where hydraulic power is used.

9. Thermal actuators

A thermal actuator is a non-electric motor that generates linear motion in response to temperature changes. Its main components are a piston and a thermal sensitive material. When there is a rise in temperature, the thermal-sensitive materials begin to expand in response, driving the piston out of the actuator. Similarly, upon detecting a drop in the temperature, the thermal-sensitive materials inside contract, making the piston retract. Thus these actuators can be used for carrying out tasks such as releasing latches, working switches and opening or closing valves. They have many applications, particularly in the aerospace, automotive, agricultural and solar industries.

10 Magnetic actuators

Magnetic actuators are those that use magnetic effects to produce motion of a part in the actuator. They usually come in the following categories: moving coil actuator, moving magnet actuator, moving iron actuator and electromagnetic actuator. In case of the first kind (moving coil actuator), a mobile coil driven by a current is placed in a static magnetic field, where it is subject to the Lorentz force. This force is proportional to the applied current. Moving magnet actuators work differently; here mobile permanent magnet is placed between two magnet poles and is s

witched from one pole to the other using coils. Such actuators can generate high forces but are not easily controlled. In moving iron actuators, a soft magnetic part placed into a coil system moves in a fashion that keeps the system magnetic energy to a minimum. Lastly, electromagnetic actuators are the ones comprising electric motors such as Brushless DC motors (BLDC) and stepper motors. These magnetic actuators are used for various purposes such as valve control, pump and compressor actuation, locking mechanisms, aerospace engineering, vibration generation, fast positioning etc.

Experiment No. 2
Objective: Introduction with running a blinking LED and fading LED with PWM

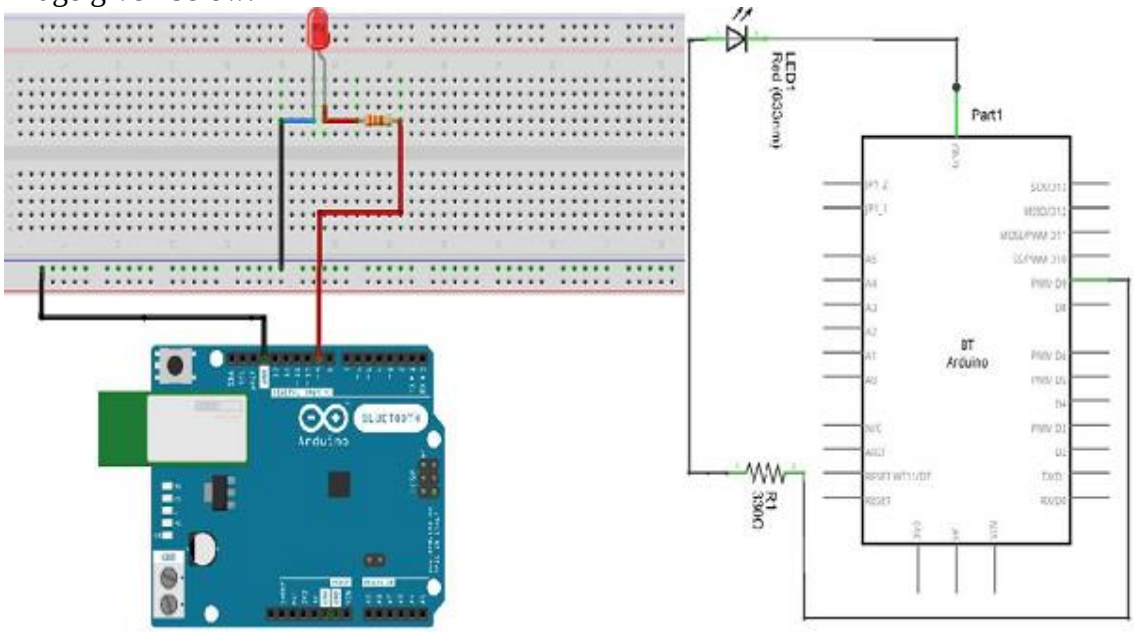
Components Required

You will need the following components −

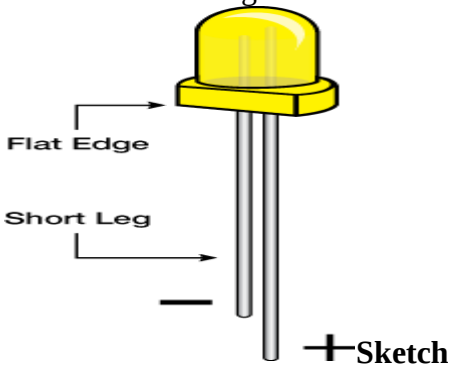
- 1 × Breadboard
- 1 × Arduino Uno
- 1 × LED
- 1 × 220Ω Resistor
- 2 × Jumper

Procedure

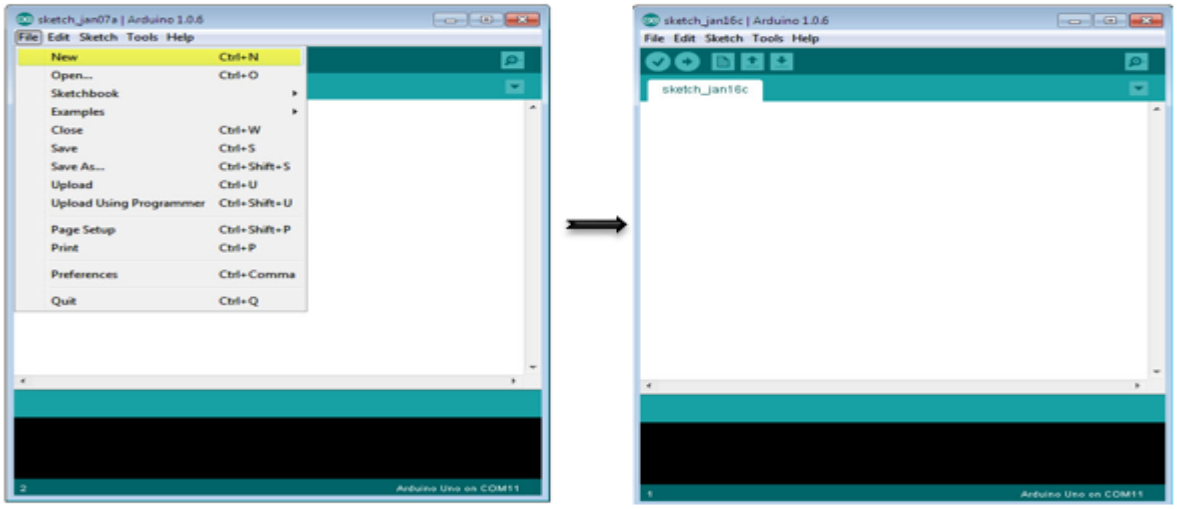
Follow the circuit diagram and hook up the components on the breadboard as shown in the image given below.



Note − To find out the polarity of an LED, look at it closely. The shorter of the two legs, towards the flat edge of the bulb indicates the negative terminal.



Open the Arduino IDE software on your computer. Coding in the Arduino language will control your circuit. Open the new sketch File by clicking New.



Arduino Code 1

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

// the setup function runs once when you press reset or power the board

void setup()
      {
            pinMode(8, OUTPUT);       // initialize digital pin 02 as an output.
       }
void loop()                    // the loop function runs over and over again forever
      {
           digitalWrite(8, HIGH);        // turn the LED on (HIGH is the voltage level)
           delay(1000);                  // wait for a second
           digitalWrite(8, LOW);        // turn the LED off by making the voltage LOW
           delay(1000);                  // wait for a second
       }
```

Code to Note

pinMode(8, OUTPUT) − Before you can use one of Arduino's pins, you need to tell Arduino Uno R3 whether it is an INPUT or OUTPUT. We use a built-in "function" called pinMode() to do this.

digitalWrite(8, HIGH) − When you are using a pin as an OUTPUT, you can command it to be HIGH (output 5 volts), or LOW (output 0 volts).

```
#define LED 13
int count;
void setup()
          {
            pinMode(LED, OUTPUT);
          }
void loop()
{
   while (count <7)
          {
```

```
        digitalWrite(LED, HIGH);
        delay(1000);
        digitalWrite(LED, LOW);
        delay(1000);
        count = count +1;
        }
}
```

Fading using loop

```
int ledPin = 9;
void setup() {
}
void loop() {
for(int fade=0 ; fade<=255 ; fade+=5)
{
analogWrite(ledPin,fade);
delay(30);
}
for(int fade=255 ; fade>=0 ; fade-=5)
{
analogWrite(ledPin,fade);
delay(30);
}
}
```

Result

You should see your LED turn on and off. If the required output is not seen, make sure you have assembled the circuit correctly, and verified and uploaded the code to your board.

Project

Design a program that will flash the LED connected to pin 13 four times. Make the flash consist of 1 second on and 2 seconds off. Use pin 8 to reset the flashing so that it will restart when pin 8 is grounded.

Experiment No. 3(A)

Objective: Arduino IDE and Operators in IDE

Arduino Integrated Development Environment - or Arduino Software (IDE) - contains a text editor for writing code, a message area, a text console, a toolbar with buttons for common functions and a series of menus. It connects to Arduino hardware to upload programs and communicate with them.

Writing Sketches

Programs written using Arduino Software (IDE) are called **sketches**. These sketches are written in text editor and are saved with file extension. ino. Editor has features for cutting/pasting and for searching/replacing text. Message area gives feedback while saving and exporting and also displays errors. Console displays text output by Arduino Software (IDE), including complete error messages and other information. Bottom right hand corner of the window displays the configured board and serial port. Toolbar buttons allow you to verify and upload programs, create, open, and save sketches, and open serial monitor.

NB: Versions of the Arduino Software (IDE) prior to 1.0 saved sketches with the extension .pde. It is possible to open these files with version 1.0, you will be prompted to save the sketch with the. ino extension on save.

 ***Verify*** Checks your code for errors compiling it.

 ***Upload*** Compiles your code and uploads it to the configured board. If you are using an external programmer with your board, you can hold down the "shift" key on your computer when using this icon. The text will change to "Upload using Programmer"

 ***New*** Creates a new sketch.

 ***Open*** Presents a menu of all the sketches in your sketchbook. Clicking one will open it within the current window overwriting its content.

Note: due to a bug in Java, this menu doesn't scroll; if you need to open a sketch late in the list, use the **File | Sketchbook** menu instead.

 ***Save*** Saves your sketch.

 ***Serial Monitor*** Opens the serial monitor.

Additional commands are found within the five menus: **File**, **Edit**, **Sketch**, **Tools**, **Help**. The menus are context sensitive, which means only those items relevant to the work currently being carried out are available.

File

- *New* Creates a new instance of the editor, with the bare minimum structure of a sketch already in place.
- *Open* Allows to load a sketch file browsing through the computer drives and folders.
- *Open Recent* Provides a short list of the most recent sketches, ready to be opened.
- *Sketchbook* Shows the current sketches within the sketchbook folder structure; clicking on any name opens the corresponding sketch in a new editor instance.
- *Examples* Any example provided by the Arduino Software (IDE) or library shows up in this menu item. All the examples are structured in a tree that allows easy access by topic or library.
- *Close* Closes the instance of the Arduino Software from which it is clicked.
- *Save* Saves the sketch with the current name. If the file hasn't been named before, a name will be provided in a "Save as.." window.
- *Save as...* Allows to save the current sketch with a different name.
- *Page Setup* It shows the Page Setup window for printing.
- *Print* Sends the current sketch to the printer according to the settings defined in Page Setup.
- *Preferences* Opens the Preferences window where some settings of the IDE may be customized, as the language of the IDE interface.
- *Quit* Closes all IDE windows. The same sketches open when Quit was chosen will be automatically reopened the next time you start the IDE.

Edit

- *Undo/Redo* Goes back of one or more steps you did while editing; when you go back, you may go forward with Redo.
- *Cut* Removes the selected text from the editor and places it into the clipboard.
- *Copy* Duplicates the selected text in the editor and places it into the clipboard.
- *Copy for Forum* Copies the code of your sketch to the clipboard in a form suitable for posting to the forum, complete with syntax coloring.
- *Copy as HTML* Copies the code of your sketch to the clipboard as HTML, suitable for embedding in web pages.
- *Paste* Puts the contents of the clipboard at the cursor position, in the editor.
- *Select All* Selects and highlights the whole content of the editor.
- *Comment/Uncomment* Puts or removes the // comment marker at the beginning of each selected line.

- *Increase/Decrease Indent* Adds or subtracts a space at the beginning of each selected line, moving the text one space on the right or eliminating a space at the beginning.
- *Find* Opens the Find and Replace window where you can specify text to search inside the current sketch according to several options.
- *Find Next* Highlights the next occurrence - if any - of the string specified as the search item in the Find window, relative to the cursor position.
- *Find Previous* Highlights the previous occurrence - if any - of the string specified as the search item in the Find window relative to the cursor position.

Sketch

- *Verify/Compile* Checks your sketch for errors compiling it; it will report memory usage for code and variables in the console area.
- *Upload* Compiles and loads the binary file onto the configured board through the configured Port.
- *Upload Using Programmer* This will overwrite the bootloader on the board; you will need to use Tools > Burn Bootloader to restore it and be able to Upload to USB serial port again. However, it allows you to use the full capacity of the Flash memory for your sketch. Please note that this command will NOT burn the fuses. To do so *Tools -> Burn Bootloader* command must be executed.
- *Export Compiled Binary* Saves a .hex file that may be kept as archive or sent to the board using other tools.
- *Show Sketch Folder* Opens the current sketch folder.
- *Include Library* Adds a library to your sketch by inserting #include statements at the start of your code. For more details, see libraries below. Additionally, from this menu item you can access the Library Manager and import new libraries from .zip files.
- *Add File* Adds a supplemental file to the sketch (it will be copied from its current location). The file is saved to the data subfolder of the sketch, which is intended for assets such as documentation. The contents of the data folder are not compiled, so they do not become part of the sketch program.

Tools

- *Auto Format* This formats your code nicely: i.e. indents it so that opening and closing curly braces line up, and that the statements inside curly braces are indented more.
- *Archive Sketch* Archives a copy of the current sketch in .zip format. The archive is placed in the same directory as the sketch.
- *Fix Encoding & Reload* Fixes possible discrepancies between the editor char map encoding and other operating systems char maps.
- *Serial Monitor* Opens the serial monitor window and initiates the exchange of data with any connected board on the currently selected Port. This usually resets the board, if the board supports Reset over serial port opening.
- *Board* Select the board that you're using. See below for descriptions of the various boards.
- *Port* This menu contains all the serial devices (real or virtual) on your machine. It should automatically refresh every time you open the top-level tools menu.
- *Programmer* For selecting a hardware programmer when programming a board or chip and not using the onboard USB-serial connection. Normally you won't need this, but if you're burning a bootloader to a new microcontroller, you will use this.
- *Burn Bootloader* The items in this menu allow you to burn a bootloader onto the microcontroller on an Arduino board. This is not required for normal use of an Arduino board but is useful if you purchase a new ATmega microcontroller (which normally come without a bootloader). Ensure that you've selected the correct board from the **Boards** menu before burning the bootloader on the target board. This command also set the right fuses.

Help

Here you find easy access to a number of documents that come with the Arduino Software (IDE). You have access to Getting Started, Reference, this guide to the IDE and other documents locally, without an internet connection. The documents are a local copy of the online ones and may link back to our online website.

- *Find in Reference* This is the only interactive function of the Help menu: it directly selects the relevant page in the local copy of the Reference for the function or command under the cursor.

Sketchbook

The Arduino Software (IDE) uses the concept of a sketchbook: a standard place to store your programs (or sketches). The sketches in your sketchbook can be opened from the **File > Sketchbook** menu or from the **Open** button on the toolbar. The first time you run the Arduino software, it will automatically create a directory for your sketchbook. You can view or change the location of the sketchbook location from with the **Preferences** dialog. Beginning with version 1.0, files are saved with a .ino file extension. Previous versions use the .pde extension. You may still open .pde named files in version 1.0 and later, the software will automatically rename the extension to .ino**.**

Tabs, Multiple Files, and Compilation

Allows you to manage sketches with more than one file (each of which appears in its own tab). These can be normal Arduino code files (no visible extension), C files (.c extension), C++ files (.cpp), or header files (.h).

Before compiling the sketch, all the normal Arduino code files of the sketch (.ino, .pde) are concatenated into a single file following the order the tabs are shown in. The other file types are left as is.

Uploading

Before uploading your sketch, you need to select the correct items from the **Tools > Board** and **Tools > Port** menus. The boards are described below. On the Mac, the serial port is probably something like **/dev/tty.usbmodem241** (for an UNO or Mega2560 or Leonardo) or **/dev/tty.usbserial-1B1** (for a Duemilanove or earlier USB board), or **/dev/tty.USA19QW1b1P1.1** (for a serial board connected with a Keyspan USB-to-Serial adapter). On Windows, it's probably **COM1** or **COM2** (for a serial board) or **COM4**, **COM5**, **COM7**, or higher (for a USB board) - to find out, you look for USB serial device in the ports section of the Windows Device Manager. On Linux, it should be **/dev/ttyACMx** , **/dev/ttyUSBx** or similar. Once you've selected the correct serial port and board, press the upload button in the toolbar or select the **Upload** item from the **Sketch** menu. Current Arduino boards will reset automatically and begin the upload. With older boards (pre-Diecimila) that lack auto-reset, you'll need to press the reset button on the board just before starting the upload. On most boards, you'll see the RX and TX LEDs blink as the sketch is uploaded. The Arduino Software (IDE) will display a message when the upload is complete, or show an error.

When you upload a sketch, you're using the Arduino **bootloader**, a small program that has been loaded on to the microcontroller on your board. It allows you to upload code without using any additional hardware. The bootloader is active for a few seconds when the board resets; then it starts whichever sketch was most recently uploaded to the microcontroller. The bootloader will blink the on-board (pin 13) LED when it starts (i.e. when the board resets).

Libraries

Libraries provide extra functionality for use in sketches, e.g. working with hardware or manipulating data. To use a library in a sketch, select it from the **Sketch > Import Library** menu. This will insert one or more **#include** statements at the top of the sketch and compile the library with your sketch. Because libraries are uploaded to the board with your sketch, they increase the amount of space it takes up. If a sketch no longer needs a library, simply delete its **#include** statements from the top of your code. There is a list of libraries in the reference. Some libraries are included with the Arduino software. Others can be downloaded from a variety of sources or through the Library Manager. Starting with version 1.0.5 of the IDE, you do can import a library from a zip file and use it in an open sketch. See these instructions for installing a third-party library.

Third-Party Hardware

Support for third-party hardware can be added to the **hardware** directory of your sketchbook directory. Platforms installed there may include board definitions (which appear in the board menu), core libraries, bootloaders, and programmer definitions. To install, create the **hardware** directory, then unzip the third-party platform into its own sub-directory. (Don't use "arduino" as the sub-directory name or you'll override the built-in Arduino platform.) To uninstall, simply delete its directory.

Serial Monitor

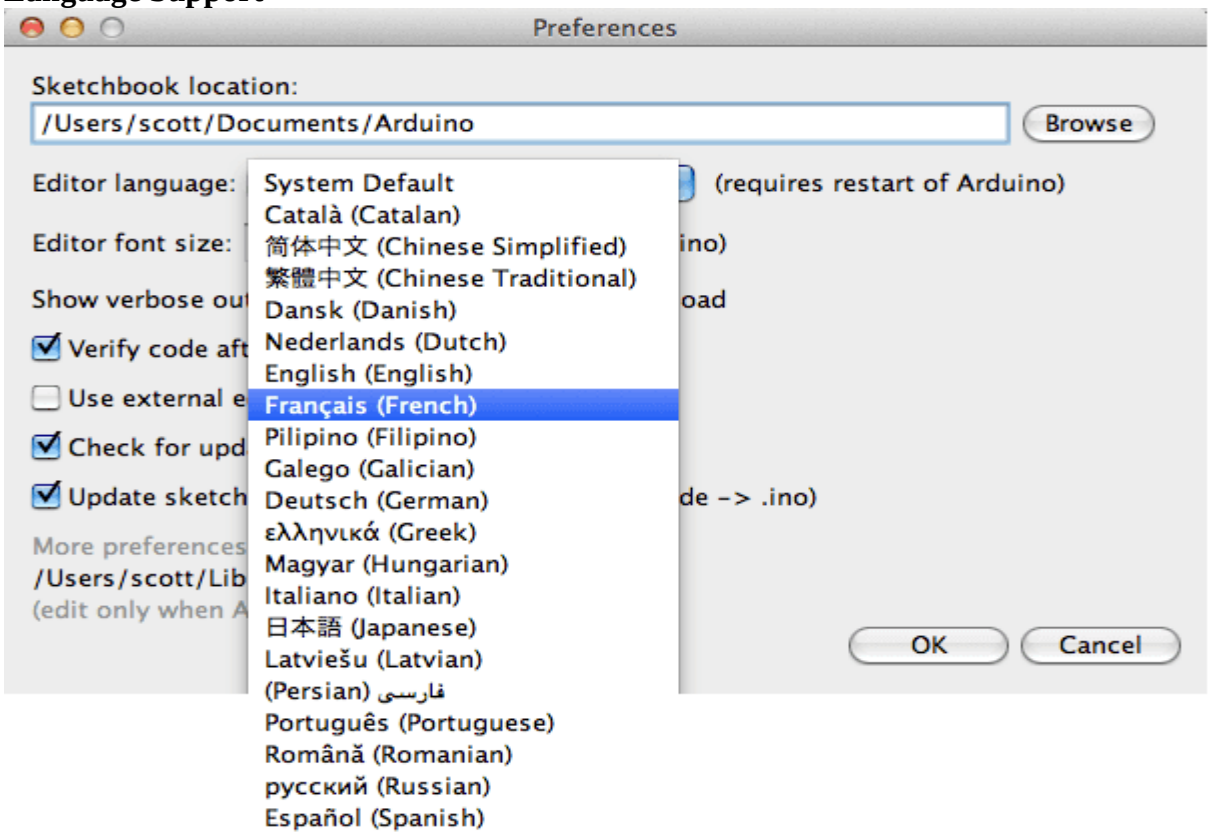
This displays serial sent from the Arduino board over USB or serial connector. To send data to the board, enter text and click on the "send" button or press enter. Choose the baud rate from the drop-down menu that matches the rate passed to **Serial.begin** in your sketch. Note that on Windows, Mac or Linux the board will reset (it will rerun your sketch) when you connect with the serial monitor. Please note that the Serial Monitor does not process control characters; if your sketch needs a complete management of the serial communication with control characters, you can use an external terminal program and connect it to the COM port assigned to your Arduino board.

You can also talk to the board from Processing, Flash, MaxMSP, etc (see the interfacing page for details).

Preferences

Some preferences can be set in the preferences dialog (found under the **Arduino** menu on the Mac, or **File** on Windows and Linux). The rest can be found in the preferences file, whose location is shown in the preference dialog.

Language Support



Since version 1.0.1 , the Arduino Software (IDE) has been translated into 30+ different languages. By default, the IDE loads in the language selected by your operating system. (Note: on Windows and possibly Linux, this is determined by the locale setting which controls currency and date formats, not by the language the operating system is displayed in.)

If you would like to change the language manually, start the Arduino Software (IDE) and open the **Preferences** window. Next to the **Editor Language** there is a dropdown menu of currently supported languages. Select your preferred language from the menu, and restart the software to use the selected language. If your operating system language is not supported, the Arduino Software (IDE) will default to English.

You can return the software to its default setting of selecting its language based on your operating system by selecting **System Default** from the **Editor Language** drop-down. This setting will take effect when you restart the Arduino Software (IDE). Similarly, after

changing your operating system's settings, you must restart the Arduino Software (IDE) to update it to the new default language.

Boards

The board selection has two effects: it sets the parameters (e.g. CPU speed and baud rate) used when compiling and uploading sketches; and sets and the file and fuse settings used by the burn bootloader command. Some of the board definitions differ only in the latter, so even if you've been uploading successfully with a particular selection you'll want to check it before burning the bootloader. Arduino Software (IDE) includes the built in support for the boards in the following list, all based on the AVR Core. The Boards Manager included in the standard installation allows to add support for the growing number of new boards based on different cores like Arduino Due, Arduino Zero, Edison, Galileo and so on.

- *Arduino Yún* An ATmega32u4 running at 16 MHz with auto-reset, 12 Analog In, 20 Digital I/O and 7 PWM.
- *Arduino Uno* An ATmega328P running at 16 MHz with auto-reset, 6 Analog In, 14 Digital I/O and 6 PWM.
- *Arduino Diecimila or Duemilanove w/ ATmega168* An ATmega168 running at 16 MHz with auto-reset.
- *Arduino Nano w/ ATmega328P* An ATmega328P running at 16 MHz with auto-reset. Has eight analog inputs.
- *Arduino Mega 2560* An ATmega2560 running at 16 MHz with auto-reset, 16 Analog In, 54 Digital I/O and 15 PWM.
- *Arduino Mega* An ATmega1280 running at 16 MHz with auto-reset, 16 Analog In, 54 Digital I/O and 15 PWM.
- *Arduino Mega ADK* An ATmega2560 running at 16 MHz with auto-reset, 16 Analog In, 54 Digital I/O and 15 PWM.
- *Arduino Leonardo* An ATmega32u4 running at 16 MHz with auto-reset, 12 Analog In, 20 Digital I/O and 7 PWM.
- *Arduino Micro* An ATmega32u4 running at 16 MHz with auto-reset, 12 Analog In, 20 Digital I/O and 7 PWM.
- *Arduino Esplora* An ATmega32u4 running at 16 MHz with auto-reset.
- *Arduino Mini w/ ATmega328P* An ATmega328P running at 16 MHz with auto-reset, 8 Analog In, 14 Digital I/O and 6 PWM.
- *Arduino Ethernet* Equivalent to Arduino UNO with an Ethernet shield: An ATmega328P running at 16 MHz with auto-reset, 6 Analog In, 14 Digital I/O and 6 PWM.

- *Arduino Fio* An ATmega328P running at 8 MHz with auto-reset. Equivalent to Arduino Pro or Pro Mini (3.3V, 8 MHz) w/ ATmega328P, 6 Analog In, 14 Digital I/O and 6 PWM.
- *Arduino BT w/ ATmega328P* ATmega328P running at 16 MHz. The bootloader burned (4 KB) includes codes to initialize the on-board Bluetooth® module, 6 Analog In, 14 Digital I/O and 6 PWM..
- *LilyPad Arduino USB* An ATmega32u4 running at 8 MHz with auto-reset, 4 Analog In, 9 Digital I/O and 4 PWM.
- *LilyPad Arduino* An ATmega168 or ATmega132 running at 8 MHz with auto-reset, 6 Analog In, 14 Digital I/O and 6 PWM.
- *Arduino Pro or Pro Mini (5V, 16 MHz) w/ ATmega328P* An ATmega328P running at 16 MHz with auto-reset. Equivalent to Arduino Duemilanove or Nano w/ ATmega328P; 6 Analog In, 14 Digital I/O and 6 PWM.
- *Arduino NG or older w/ ATmega168* An ATmega168 running at 16 MHz*without* auto-reset. Compilation and upload is equivalent to Arduino Diecimila or Duemilanove w/ ATmega168, but the bootloader burned has a slower timeout (and blinks the pin 13 LED three times on reset); 6 Analog In, 14 Digital I/O and 6 PWM.
- *Arduino Robot Control* An ATmega328P running at 16 MHz with auto-reset.
- *Arduino Robot Motor* An ATmega328P running at 16 MHz with auto-reset.
- *Arduino Gemma* An ATtiny85 running at 8 MHz with auto-reset, 1 Analog In, 3 Digital I/O and 2 PWM.

For instructions on installing support for other boards, see third-party hardware above.

The operators are widely used in Arduino programming from basics to advanced levels. It plays a crucial role in every programming concept like C, C++, Java, etc.

The operators are used to solve logical and mathematical problems. For example, to calculate the temperature given by the sensor based on some analog voltage.

The types of Operators classified in Arduino are:

1. Arithmetic Operators
2. Compound Operators
3. Boolean Operators
4. Comparison Operators
5. Bitwise Operators

Arithmetic Operators

There are six basic operators responsible for performing mathematical operations in Arduino, which are listed below:

Assignment Operator (=): Used to set variable's value. It is quite different from equal symbol (=) normally used in mathematics.

Addition (+): Used for addition of two numbers. For example, P + Q.

Subtraction (-): Used to subtract one value from the another. For example, P - Q.

Multiplication (*): Used to multiply two numbers. For example, P * Q.

Division (/): Used to determine result of one number divided with another. For example, P/Q.

Modulo (%): Used to calculate remainder after division of one number by another number.

Example 1: Consider the below code.

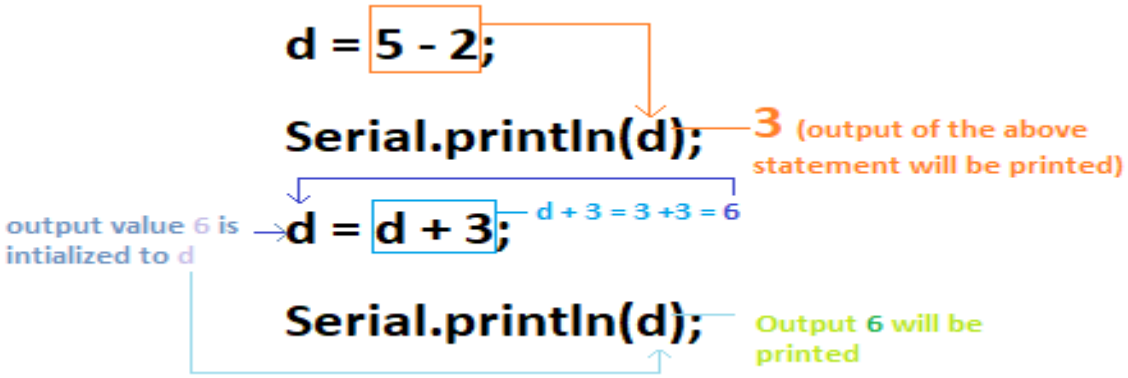
```
1. int b;
2. void setup ( )
3. {
4. Serial.begin( 9600 );
5. }
6. void loop ( )
7. {
8. b = 5 + 2;
9. Serial.println(b);
10. }
```

In above code, we have assigned the result of the addition of two numbers to b before printing it to the console. For output, click on **Upload** and **Serial Monitor button** present on toolbar.

Example 2: Consider the below code:

```
1. int d;
2. void setup ( )
3. {
4. Serial.begin( 9600 );
5. }
6. void loop ( )
7. {
8. d = 5 - 2;
9. Serial.println(d);
10. d = d + 3;
11. Serial.println(d);
12. }
```

Here, d= d +3 is not operated as a usual mathematical operation. It is the assignment operator where right of the function is evaluated first and is assigned to the left of the equal sign.



If we want decimal values to be printed, we need to use the **float** instead of **int**.

Example 3: Consider the below code:

```
1. int b;
2. void setup ( )
3. {
4. Serial.begin( 9600 );
5. }
6. void loop ( )
7. {
8. b = 20.0 / 3; // decimal value is used to force the compiler to print decimal value.
9. Serial.println(b);
10. }
```

Output: 6.66

Order of mathematical operations

Let's understand the order of operations considered by the Arduino while performing calculation:

1. Parentheses ( )
2. Multiplication, division, and modulo
3. Addition and subtraction

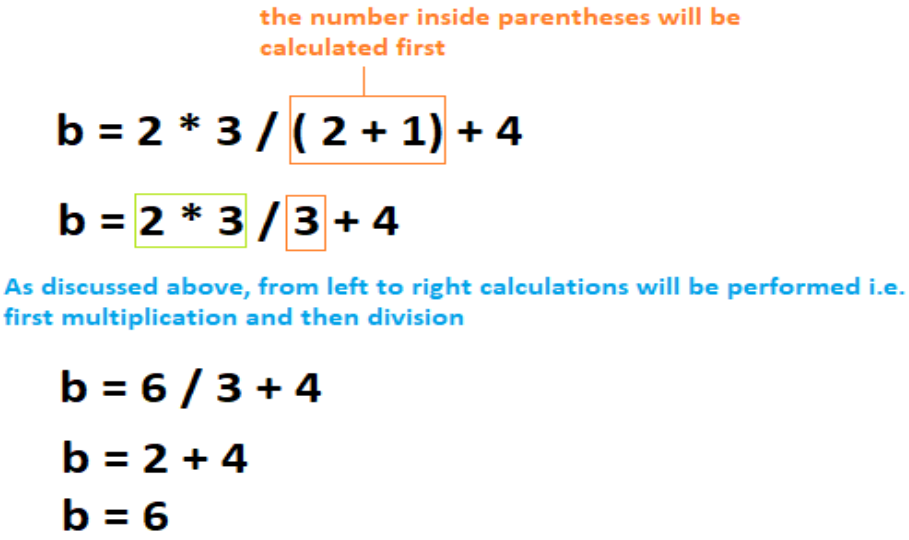
If there are multiple operations next to each other, they will be computed from left to right.

Example 4: Consider the below code:

```
1. int c;
2. void setup ( )
3. {
4. Serial.begin( 9600 );
```

```
5. }
6. void loop ( )
7. {
8. c = 2 * 3 / (2 + 1)  + 4;
9. Serial.println(c);
10. }
```

Output: 6



Compound Operators

Compound operators perform two or more calculations at once. Result of right operand is assigned to left operand. Same condition will apply to all the compound operators, which are listed below:

Let's consider a variable **b**.

- **b + +**

Here, b = b + 1. It is called **increment operator**.

- **b + =**

For example, b + = 4. It means, b = b+ 4.

- **b - -**

Here, b = b - 1. It is called as the **decrement operator**.

- **b - =**

For example, b - = 3. It means, b = b - 3.

- **b * =**

For example, b * = 6. It means, b = b * 6.

- **b / =**

For example, b / = 5. It means, b = b / 5.

- **b % =**

For example, b % = 2. It means, b = b % 2.

Now, let's use the above operators with two variables, b and c.

- b + = c (b = b + c)
- b - = c (b = b - c)
- b * = c (b = b * c)
- b / = c (b = b / c)
- b % = c (b = b % c)

Boolean Operators

The Boolean Operators are **NOT (!), Logical AND (& &), and Logical OR (| |)**.

- **Logical AND (& &):** The result of the condition is true if both the operands in the condition are true.
  - **if** (a = = b & & b = = c)

Above statement is true if both conditions are true. If any of conditions is false, statement will be false.

- **Logical OR (| |):** Result of condition is true, if either of variables in condition is true.
  - **if** (a > 0 | | b > 0)

The above statement is true, if either of above condition (a> 0 or b > 0) is true.

- **NOT (!):** It is used to reverse the logical state of the operand.
  - For example, a! = 2.

The NOT operator returns value 1 or TRUE when specified operand is FALSE. It also reverses the value of the specified expression.

Comparison Operators

The comparison operators are used to compare the value of one variable with the other. The comparison operators are listed below:

- **less than (<):** The less than operator checks that the value of the left operand is less than the right operand. The statement is true if the condition is satisfied.

Example: Consider the below code.

```
1. int b;
2. int c ;
```

```
3. void setup ( )
4. {
5. Serial.begin( 9600 );
6. }
7. void loop ( )
8. {
9. b = 3;
10. c = 5;
11. if ( b < 4 )
12. Serial.println(b);
13. if ( c < 4)
14. Serial.println( c);
15. }
```

Output: 3

In above code, if any of two statements is correct, corresponding value of variable will be printed. Here, only first condition is correct. Hence, value of b will be printed.

- **greater than (>):** Less than operator checks that value of left side of a statement is greater than right side. Statement is true if the condition is satisfied. For example, a > b. If a is greater than b, the condition is true, else false.
- **equal to (= =):** It checks the value of two operands. If the values are equal, the condition is satisfied. For example, a = = b. The above statement is used to check if the value of a is equal to b or not.
- **not equal to (! =):** It checks the value of two specified variables. If the values are not equal, the condition will be correct and satisfied. For example, a! = b.
- **less than or equal to (< =):** The less or equal than operator checks that the value of left side of a statement is less or equal to the value on right side. The statement is true if either of the condition is satisfied. For example, a < = b. It checks the value of a is less or equal than b.
- **greater than or equal to (> =):** The greater or equal than operator checks that the value of the left side of a statement is greater or equal to the value on the right side of that statement. The statement is true if the condition is satisfied. For example, a > = b. It checks the value of a is greater or equal than b. If either of the condition satisfies, the statement is true.

Bitwise Operators

The Bitwise operators operate at the **binary level**. These operators are quite easy to use. There are various bitwise operators. Some of the popular operators are listed below:

1. **bitwise NOT (~):** The bitwise NOT operator acts as a complement for reversing the bits. For example, if b = 1, the NOT operator will make the value of b = 0.
   - 0 0 1 1 // Input or operand 1 (decimal value 3)
   - 1 1 0 0 // Output (reverses the input bits) decimal value is 12
2. **bitwise XOR (^):** The output is 0 if both the inputs are same, and it is 1 if the two input bits are different. For example,
   1. 1 0 0 1 // input 1 or operand 1
   2. 0 1 0 1 // input 2
   3. 1 1 0 0 // Output (resultant - XOR)
3. **bitwise OR (|):** Output is 0 if both of the inputs in the OR operation are 0. Otherwise, the output is 1. The two input patterns are of 4 bits. For example,
   1. 1 1 0 0 // input 1 or operand 1
   2. 0 0 0 1 // input 2
   3. 1 1 0 1 // Output (resultant - OR)
4. **bitwise AND (&):** Output is 1 if both the inputs in the AND operation are 1. Otherwise, the output is 0. The two input patterns are of 4 bits. For example,
   1. 1 1 0 0 // input 1 or operand 1
   2. 0 1 0 1 // input 2
   3. 0 1 0 0 // Output (resultant - AND)
5. **bitwise left shift (< <):** The left operator is shifted by the number of bits defined by the right operator.
6. **bitwise right shift (> >):** The right operator is shifted by the number of bits defined by the left operator.

Experiment No. 3(B)

Aim: Frequently used functions in Arduino IDE.

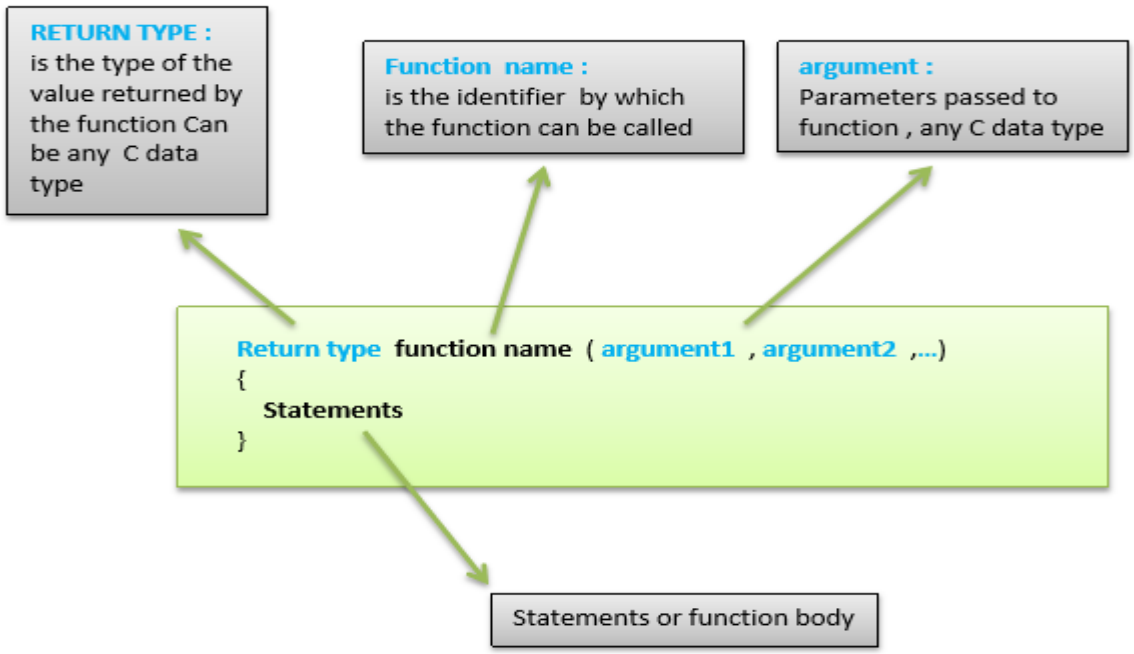
Functions allow structuring the programs in segments of code to perform individual tasks. The typical case for creating a function is when one needs to perform the same action multiple times in a program.

Standardizing code fragments into functions has several advantages −

- Functions help the programmer stay organized. Often this helps to conceptualize the program.
- Functions codify one action in one place so that the function only has to be thought about and debugged once.
- This also reduces chances for errors in modification, if the code needs to be changed.
- Functions make the whole sketch smaller and more compact because sections of code are reused many times.
- They make it easier to reuse code in other programs by making it modular, and using functions often makes the code more readable.

There are two required functions in an Arduino sketch or a program i.e. setup () and loop(). Other functions must be created outside the brackets of these two functions.

The most common syntax to define a function is −



Function Declaration

A function is declared outside any other functions, above or below the loop function.

We can declare the function in two different ways −

The first way is just writing the part of the function called **a function prototype** above the loop function, which consists of −

- Function return type
- Function name
- Function argument type, no need to write the argument name

Function prototype must be followed by a semicolon (;).

The following example shows the demonstration of the function declaration using the first method.

Example

```
int sum_func (int x, int y) // function declaration
{
  int z = 0;
  z = x+y ;
  return z; // return the value
}

void setup ()
{
  Statements // group of statements
}

void loop ()
{
  int result = 0 ;
  result = Sum_func (5,6) ; // function call
}
```

The second part, which is called the function definition or declaration, must be declared below the loop function, which consists of −

- Function return type
- Function name
- Function argument type, here you must add the argument name
- The function body (statements inside the function executing when the function is called)

The following example demonstrates the declaration of function using the second method.

Example

```
int sum_func (int , int ) ; // function prototype

void setup ()
{
  Statements // group of statements
}
void loop ()
{
  int result = 0 ;
  result = Sum_func (5,6) ; // function call
}
int sum_func (int x, int y) // function declaration
{
  int z = 0;
  z = x+y ;
  return z; // return the value
}
```

The second method declares function above loop function.

Experiment No. 4

Aim: Control Structure writing programs for if else, for and while.

A programming language's control structures allow you to take action based on certain conditions. The Arduino control structures are very similar to the control structure of the C language and include:

- if
- if...else
- for
- switch case
- while
- do... while
- break
- continue
- return
- goto

Conditional Statements

If statements and *while* statements are called *conditional statements*. They evaluate a particular condition, and execute a series of statements in brackets if the condition is true { }. Here's an example:

```
if (pin_b == LOW)
{
digitalWrite(ledPin, HIGH);// This tells the LED to turn on
}
```

"if" statement consists of word "if" followed by a condition in parentheses. If condition is true, then statements between brackets that follow will be executed. In this case, if digital value of pin_b is LOW, then program will set value of ledPin to HIGH. Statements between brackets are executed only once. While loop is similar to if statement, but in this case, statements between brackets are executed over and over until condition becomes false.

Here's an example:

```
while (pin_0 == LOW)
 {
 digitalWrite(ledPin, HIGH);
 delay(100);
 digitalWrite(ledPin, LOW);
 delay(100);
 switchValue = digitalRead(pin_0); // This stores a new value into the variable switchValue
 }
```

In this example, the program will toggle the state the LED (i.e. blink) until a user presses a switch causing the digital value at pin_0 to go HIGH.

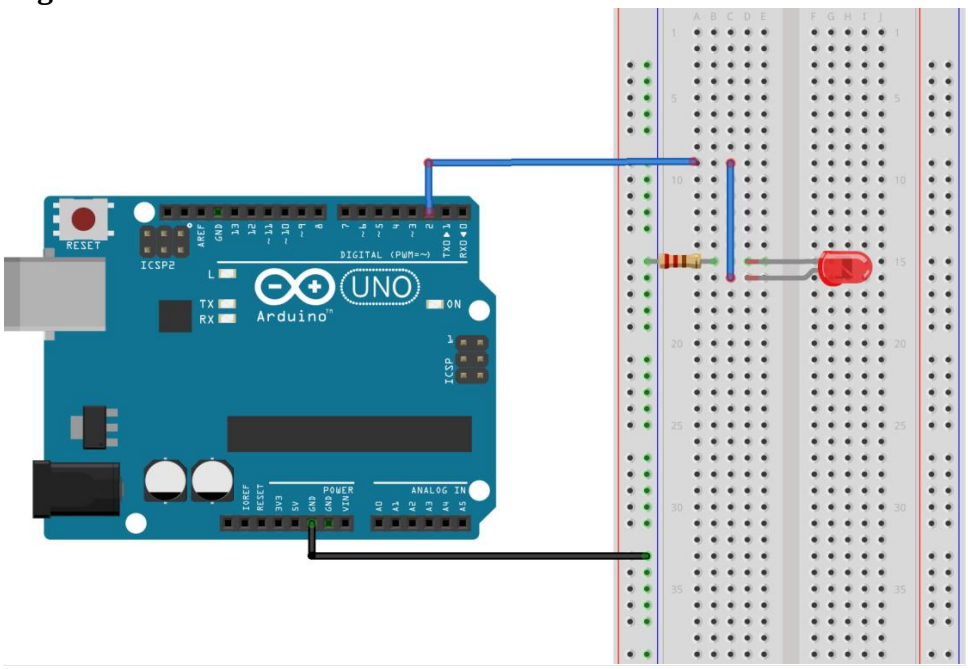
EXPERIMENT: FOR LOOP

Using For loop change brightness of an LED. Assume LED to be connected to pin 2 of Arduino.

Hardware Required

- 1 x Arduino UNO
- 1 x breadboard
- 1 x LED
- 1 X 10k ohm resistor
- Jumper wires

Wiring Diagram



Code: Changing Brightness Using For Loop

```
int LEDpin = 13;
void setup()
{
 pinMode(LEDpin, OUTPUT);
}
void loop()
{
  for(int value = 0; value<=255; value++)
          {
             analogWrite(LEDpin, value);
             delay(30);
          }
  delay(50);
  for(int value = 255; value>=0; value--)
          {
              analogWrite(LEDpin, value);
             delay(30);
           }
  delay(50);
}
```

EXPERIMENT: if-else conditions

```
void setup()
{
   pinMode(LED_BUILTIN, OUTPUT);
}
void loop() {
   int sensorValue = analogRead(A0);
   if (sensorValue > 700) {
     digitalWrite(LED_BUILTIN, HIGH);
```

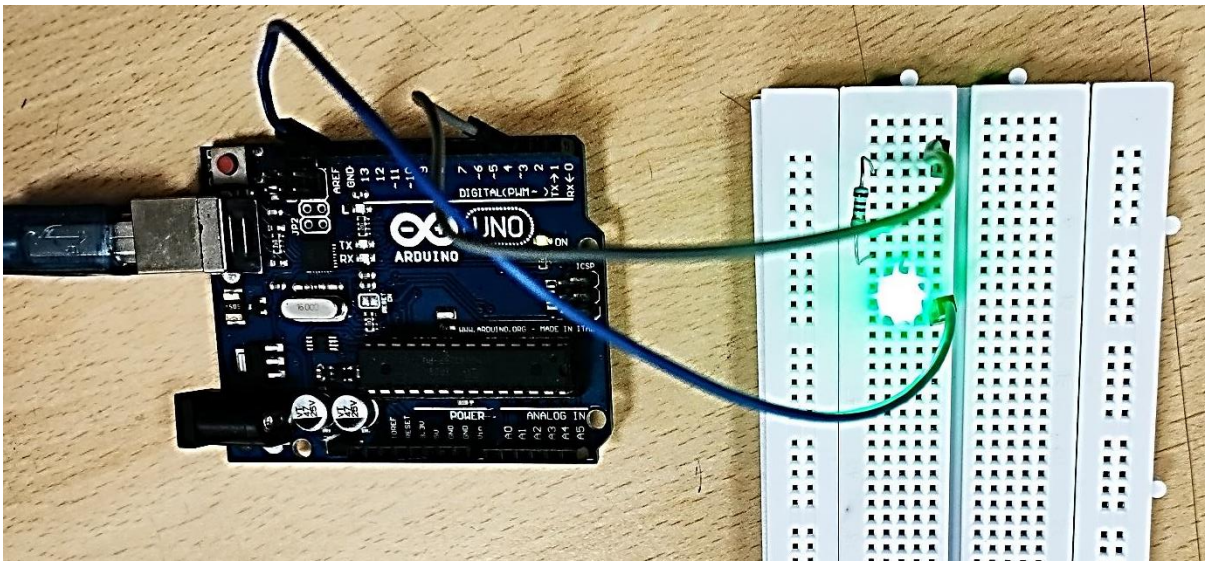
```
  }
  else {
     digitalWrite(LED_BUILTIN, LOW);

  }
  delay(10);
}
```

EXPERIMENT: Do-while loop

```
int sensorPin = A0;
int LEDpin = 13;
void setup() {
  pinMode(sensorPin, INPUT);
  pinMode(LEDpin, OUTPUT);
}
void loop() {
do
  {
    digitalWrite(LEDpin, HIGH);
    delay(500);
    digitalWrite(LEDpin, LOW);
    delay(500);
 }
while (analogRead (sensorPin > 700));
 }
```

Wiring Diagram



Result

PASTE YOUR SCREENSHOT OF THE SUCCESSFUL RUN OF PROGRAM INCLUDING HARDWARE CONNECTION AND ARDUINO IDE

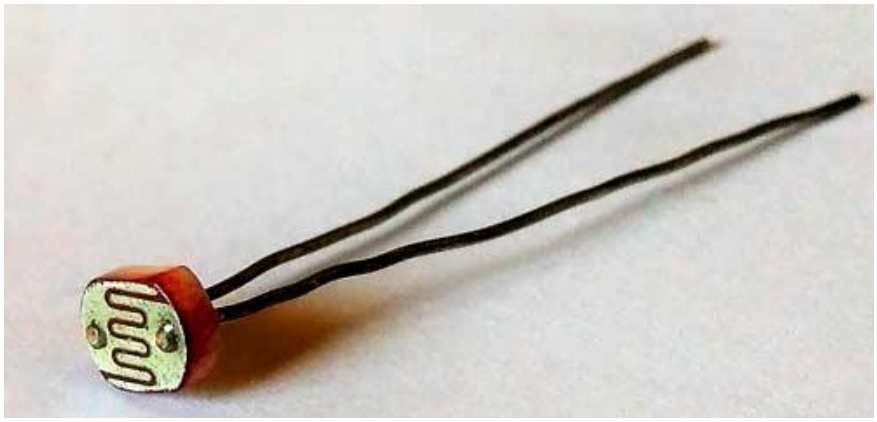
Experiment No. 5

Aim: a. Measuring light with Lux and a photo resistor demonstration

b. Reading and writing digital and analog values. Digital and analog read/write demonstration.

Theory

LDR stands for Light Dependent Resistor. LDR is a type of resistor that changes resistance as light on its surface changes i.e., less light or more darkness on LDR surface causes its resistance to increase. LDR is connected in series with a 10k resistor. LDR is used to detect light levels and switch an LED on or off.

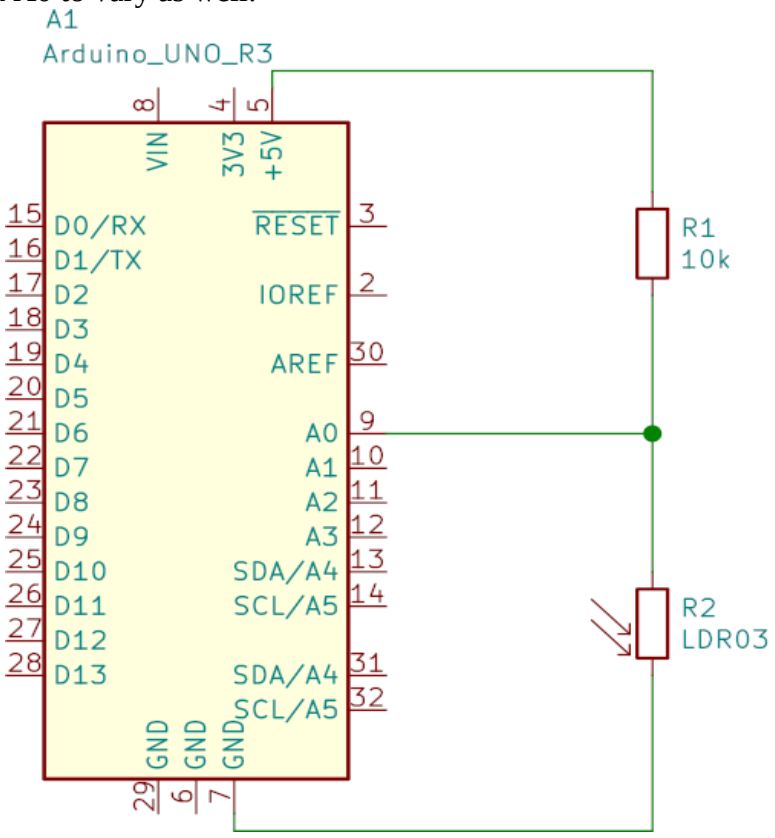


Working of LDR with Arduino Uno

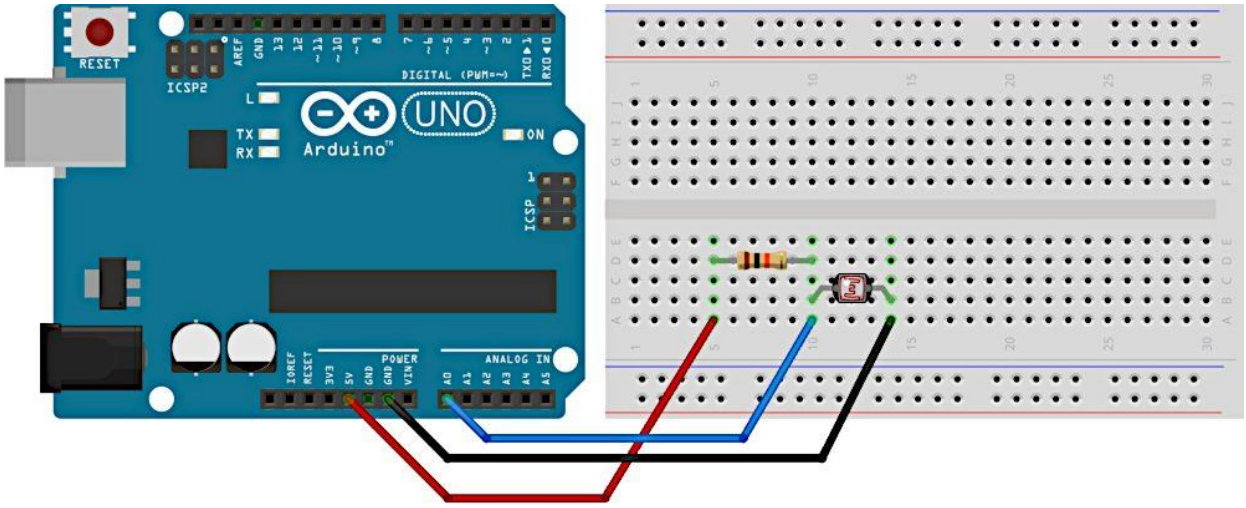
- LDR gives out analog voltage when connected to VCC (5V), which varies with input light intensity.
- i.e., greater the intensity of light, greater will be corresponding voltage from LDR.
- LDR gives out an analog voltage, as it is connected to analog input pin on Arduino.
- Arduino, with its built-in ADC (analog-to-digital converter), converts analog voltage (from 0-5V) into a digital value in range of (0-1023).
- When there is sufficient light in its environment or on its surface, converted digital values read from LDR through Arduino will be in range of 800-1023.

Connection Diagram

- Here, 10k resistor R1 and LDR R2 form a voltage divider.
- i.e, voltage at junction of R1 and R2 is divided voltage from 5V that is across them.
- As light varies on LDR surface, so does its resistance which causes voltage between GND and A0 to vary as well.



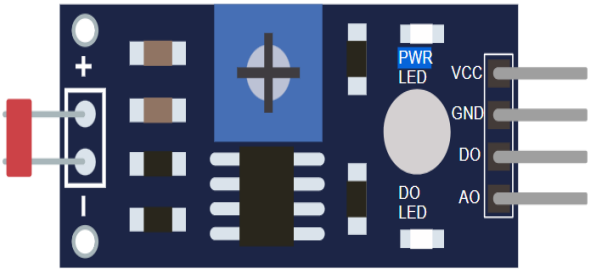
Arduino Uno LDR Breadboard Circuit



LDR Module Pin Description

Pins on sensor

- **VCC** - Positive power supply,
- **GND** - Ground
- **D_O** - Digital output
- **A_O** - Analog output



Code

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

void loop()
{
  int sensorValue = analogRead(A0);
  if (sensorValue > 700)
  {
    digitalWrite(LED_BUILTIN, HIGH);
  }
  else
  {
    digitalWrite(LED_BUILTIN, LOW);
  }
  delay(10);
}
```

Result

PASTE YOUR SCREENSHOT OF THE SUCCESSFUL RUN OF PROGRAM INCLUDING HARDWARE CONNECTION AND ARDUINO IDE

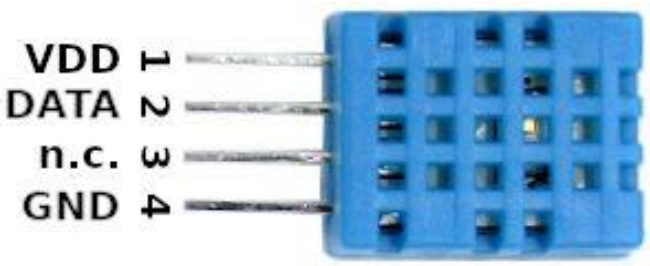
Experiment No. 6

Aim: To measure temperature and humidity using DHT sensor

Theory

DHT11 -Low-cost digital sensor for sensing temperature and humidity. It is easily interfaced with any microcontroller such as Arduino, Raspberry Pi, etc… to measure humidity and temperature instantaneously. It is available as a sensor and as a module. Difference between sensor and module is of Pull-up resistor and a power-on LED. It has resistive humidity sensing and negative temperature coefficient (NTC). 8 bit MCU is connected in it which is responsible for its fast response. DHT11 sensor consists of a capacitive humidity sensing element and a thermistor for sensing temperature. Humidity sensing capacitor has two electrodes with a moisture-holding substrate as a dielectric between them. Change in capacitance value occurs with change in humidity levels. IC measure, process this changed resistance values and change them into digital form. For measuring temperature this sensor uses Negative Temperature coefficient thermistor, which causes a decrease in its resistance value with an increase in temperature. To get a larger resistance value even for smallest change in temperature, this sensor is usually made up of semiconductor ceramics or polymers.

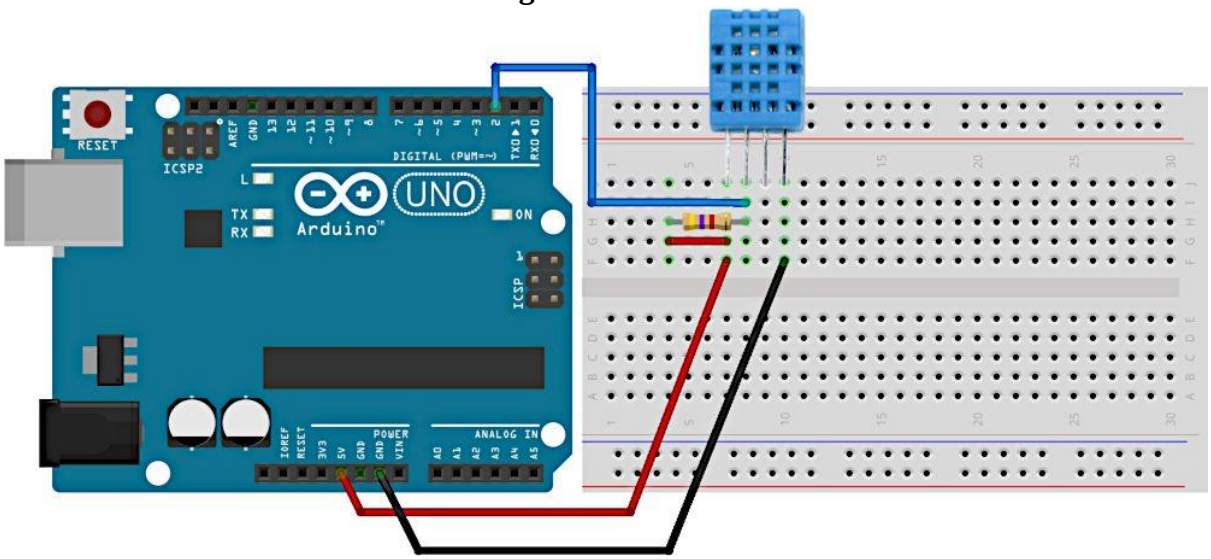
DHT Sensor Pin out

PIN 1,2,3,4 (from left to right) • **PIN 1-** 3.3V-5V Power supply • **PIN 2-** Data • **PIN 3-** Null • **PIN 4-** Ground	

DHT Sensor Library

- Arduino supports a special library for DHT11 and DHT22 sensors
- Provides function to read temperature and humidity values from data pin
- dht.readHumidity()
- dht.readTemperature()
- Install DHT Sensor Library
- Go to sketch -> Include Library -> Manage Library

Arduino and DHT Sensor Circuit Diagram



DHT 22Code

```
#include <DHT.h>
DHT dht(2, DHT22);
void setup() {
  Serial.begin(9600);
  dht.begin();
}

void loop() {
  float temperature = dht.readTemperature();
  float humidity = dht.readHumidity();
  Serial.print("Humidity: ");
  Serial.print(humidity);
  Serial.print("%  Temperature: ");
  Serial.print(temperature);
  Serial.println("°C ");
  delay(2000);
}
```

DHT 11Code

```
#include "DHT.h"
#define DHTPIN 2
#define DHTTYPE DHT11
// DHT dht(2,DHT11)
DHT dht(DHTPIN, DHTTYPE);
void setup() {
Serial.begin(9600);
Serial.println("DHT11 test!");
dht.begin();
}
void loop() {
delay(2000);
float humidity = dht.readHumidity();
float temperature = dht.readTemperature();
if (isnan(humidity) || isnan(temperature) )
```

```
//isnan = is NOT A NUMBER which return true when it is not a number
{
Serial.println("Failed to read from DHT sensor!");
return;
}
```

Result

PASTE YOUR SCREENSHOT OF THE SUCCESSFUL RUN OF PROGRAM INCLUDING HARDWARE CONNECTION AND ARDUINO IDE

Experiment No. 7

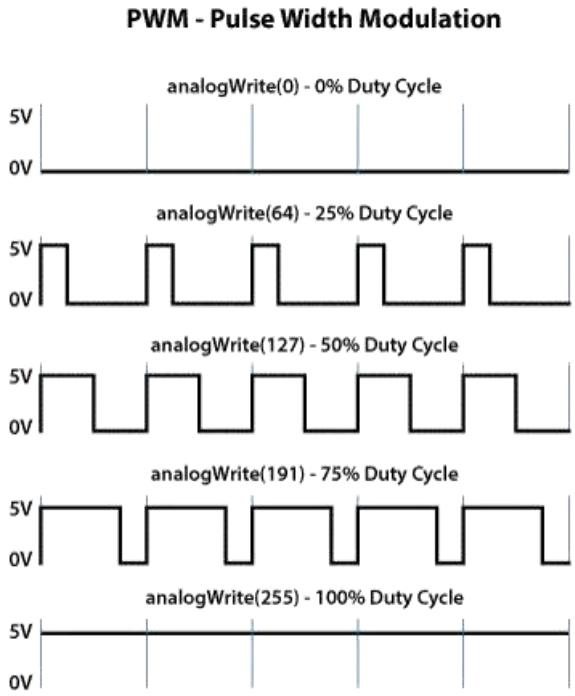
Aim: Custom functions that can be created for specitic Needs. Using a RGB LED create a custom function program.

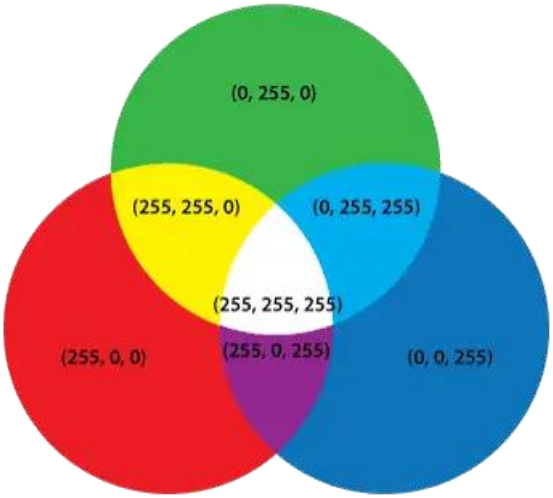
Theory

RGB - Stands for "Red Green Blue." RGB refers to three hues of light that can be mixed together to create different colors. Combining red, green, and blue light is standard method of producing color images on screens, such as TVs, computer monitors, and smartphone screens. Types of RGB led's: Common cathode and Common anode. In common cathode RGB led, cathode of all led's is common and PWM signals is given to anode of led's. In common anode RGB led, anode of all led's is common and PWM signals is given to cathode of led's.

How does an RGB LED work?

- RGB LED can emit different colors by mixing 3 basic colors red, green and blue.
- Consists of 3 separate LEDs red, green and blue packed in a single case.
- Comprises of 4 leads, one lead for each of 3 colors and one common cathode or anode depending on RGB LED type.
- Cathode will be connected to ground and 3 anodes will be connected through 220 Ohms resistors to 3 digital pins on Arduino Board that can provide PWM signal.
- Using PWM for simulating analog output will provide different voltage levels to LEDs to get desired colors

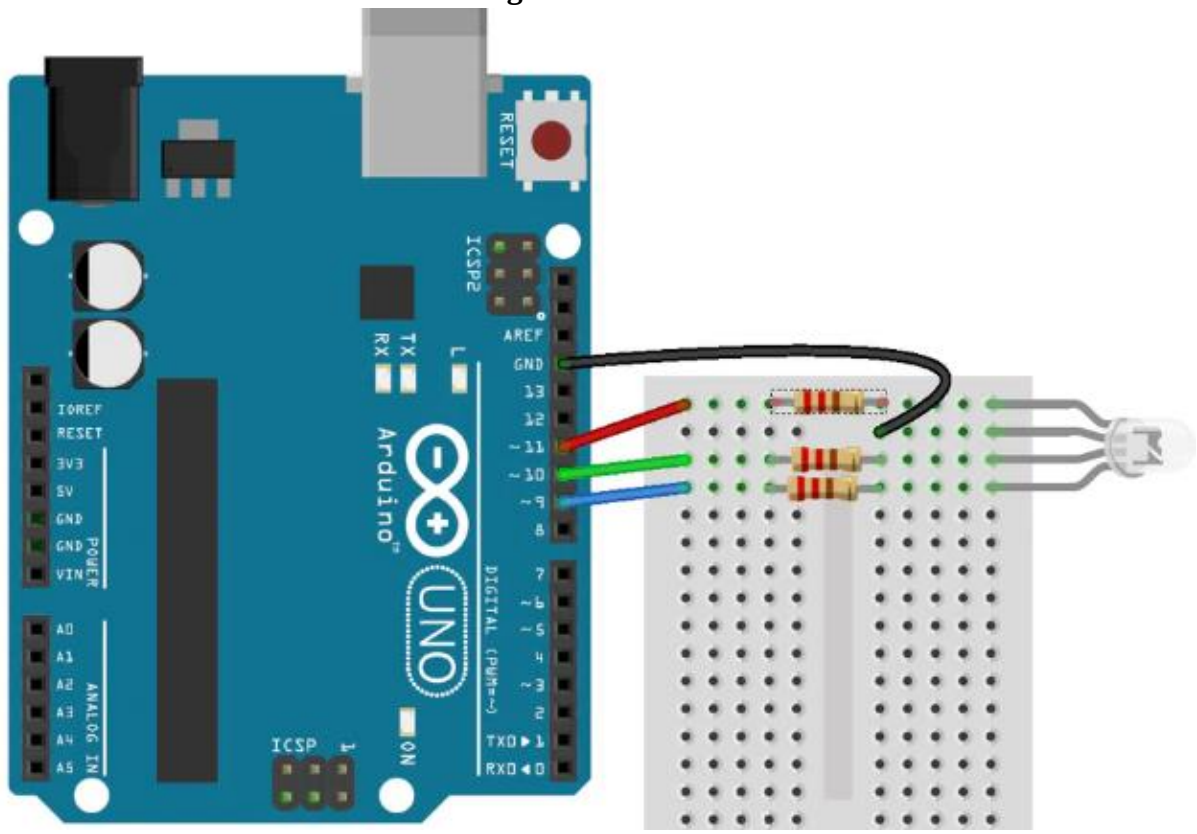




RGB Sensor Pin out

PIN 1,2,3,4 (from left to right) • **Pin D5 - R** • **Pin D4 - G** • **Pin D3 - B**	

Arduino and RGB Sensor Circuit Diagram



RGB LED Code

```
#define LED1RED 5
#define LED1BLUE 3
#define LED1GREEN 4
void setup() {
pinMode(LED1RED, OUTPUT);
pinMode(LED1BLUE, OUTPUT);
pinMode(LED1GREEN, OUTPUT);
Serial.begin(9600);
}
void loop() {
digitalWrite(LED1RED, HIGH);
Serial.println("LED1RED ");
delay(1000);
```

```
digitalWrite(LED1RED, LOW);
Serial.println("LED1RED");
delay(1000);
digitalWrite(LED1BLUE, HIGH);
Serial.println("LED1BLUE");
delay(1000);
digitalWrite(LED1BLUE, LOW);
Serial.println("LED1BLUE");
delay(1000);
digitalWrite(LED1GREEN, HIGH);
Serial.println("LED1GREEN");
delay(1000);
digitalWrite(LED1GREEN, LOW);
Serial.println("LED1GREEN");
delay(1000);
}
```

Using RGB sensor to create functions for special needs such as setcolor in the following case:

```
int redPin= 7;
int greenPin = 6;
int bluePin = 5;
void setup() {
  pinMode(redPin, OUTPUT);
  pinMode(greenPin, OUTPUT);
  pinMode(bluePin, OUTPUT);
}
void loop() {
  setColor(255, 0, 0); // Red Color
  delay(1000);
  setColor(0, 255, 0); // Green Color
  delay(1000);
  setColor(0, 0, 255); // Blue Color
  delay(1000);
  setColor(255, 255, 255); // White Color
  delay(1000);
 setColor(170, 0, 255); // Purple Color
  delay(1000);
}
void setColor(int redValue, int greenValue, int blueValue) {
  analogWrite(redPin, redValue);
  analogWrite(greenPin, greenValue);
  analogWrite(bluePin, blueValue);
}
```

Result

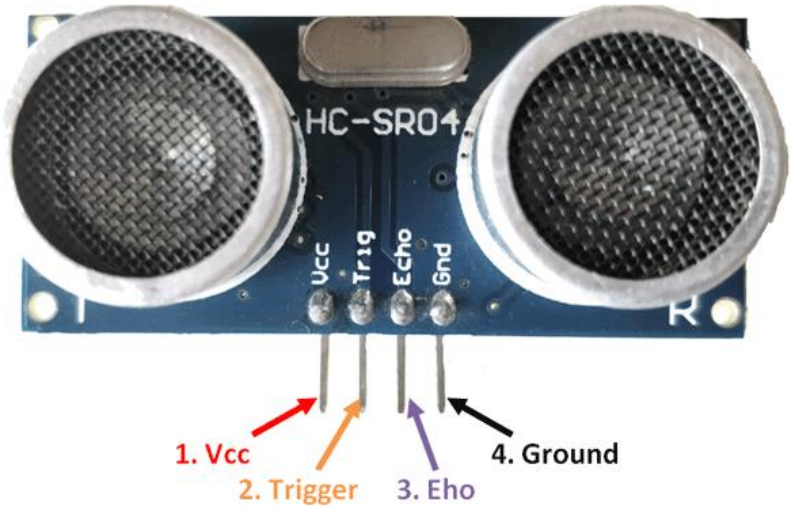
PASTE YOUR SCREENSHOT OF THE SUCCESSFUL RUN OF PROGRAM INCLUDING HARDWARE CONNECTION AND ARDUINO IDE

Experiment No. 8

Aim: To understand the working and programming of Ultrasonic Sensor.

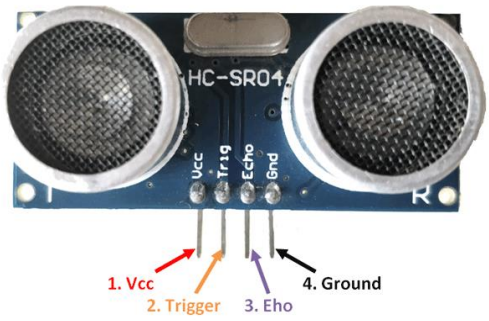
Theory

Ultrasonic sensors measure distance by using ultrasonic waves. Sensor head emits an ultrasonic wave and receives wave reflected back from target. Ultrasonic sensors measure distance to target by measuring time between emission and reception. Optical sensor has a Tx and Rx, whereas an ultrasonic sensor uses a single ultrasonic element for both emission and reception. In a reflective model ultrasonic sensor, a single oscillator emits and receives ultrasonic waves alternately. This enables miniaturization of the sensor head.



HC-SR04 Ultrasonic Module has 4 pins, Ground, VCC, Trig and Echo. Ground and VCC pins of module needs to be connected to Ground and 5 volts pins on Arduino Board respectively and trig and echo pins to any Digital I/O pin on Arduino Board. To generate ultrasound, set Trig on a High State for 10 us to send out an 8 cycle sonic burst which will travel at speed sound and will be received in Echo pin. Echo pin will output time in microseconds sound wave travelled.

Ultrasonic Sensor Pin out

PIN 1,2,3,4 (from left to right) • **Ground – G** • **12 – trig** • **11 – echo** • **5Volt – Vcc**	

Ultrasonic Sensor Code

```
#define trigPin 11
#define echoPin 12
#define ledPin 13
void setup() {
Serial.begin(9600);
pinMode(trigPin,OUTPUT);
pinMode(echoPin,INPUT);
pinMode(ledPin,OUTPUT);
}
void loop() {
long duration , distance;
digitalWrite(trigPin,LOW);
delayMicroseconds(2);
digitalWrite(trigPin,HIGH);
delayMicroseconds(10);
digitalWrite(trigPin,LOW);
duration = pulseIn(echoPin,HIGH);
distance = (duration/2) / 29.1;
if(distance<10)
{
digitalWrite(ledPin,HIGH);
}else{
digitalWrite(ledPin,LOW);
}
Serial.print(distance);
Serial.println("cm");
delay(1500);
}
```

Result

PASTE YOUR SCREENSHOT OF THE SUCCESSFUL RUN OF PROGRAM INCLUDING HARDWARE CONNECTION AND ARDUINO IDE

Experiment No. 9

Aim: To understand the working and programming of Servo Motor.

Theory

A servo motor is a self-contained electrical device, that rotate parts of a machine with high efficiency and with great precision. Output shaft of this motor can be moved to a particular angle, position and velocity that a regular motor does not have. Servo Motor utilizes a regular motor and couples it with a sensor for positional feedback. Servo motor is a closed-loop mechanism that incorporates positional feedback in order to control the rotational or linear speed and position. Motor is controlled with an electric signal, either analog or digital, which determines the amount of movement which represents the final command position for the shaft.

Servo Motor Pin out

- Servo motors have three wires: power, ground, and signal.
- Power wire is typically red, and connected to 5V pin on Arduino board.
- Ground wire is typically black or brown and connected to ground pin on board.
- Signal pin is typically yellow or orange and should be connected to PWM pin on board.

Servo Motor Code 1

```
#include <Servo.h>
Servo myservo; // create servo object to control a servo
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)
{
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)
{
myservo.write(pos); // tell servo to go to position in variable 'pos'
delay(15); // waits 15ms for the servo to reach the position
```

```
}
}
```

Servo Motor Code 2

```
#include <Servo.h>
Servo myservo; // create servo object to control a servo
void setup() {
Serial.begin(9600);
myservo.attach(9); // attaches the servo on pin 9 to the servo object
}

void loop() {
int val;
while (Serial.available()>0)
{
val = Serial.parseInt();
if(val!=0)
{
Serial.println(val);
myservo.write(val);
}
delay(15);
}
```

Result

PASTE YOUR SCREENSHOT OF THE SUCCESSFUL RUN OF PROGRAM INCLUDING HARDWARE CONNECTION AND ARDUINO IDE

Experiment No. 10

Aim: Adding an LCD screen and sketch walkthrough

Theory

The LiquidCrystal library allows you to control LCD displays that are compatible with the Hitachi HD44780 driver. There are many of them out there, and you can usually tell them by the 16-pin interface. The LCDs have a parallel interface, meaning that the microcontroller has to manipulate several interface pins at once to control the display.

The interface consists of the following pins:

A register select (RS) pin that controls where in the LCD's memory you're writing data to. You can select either the data register, which holds what goes on the screen, or an instruction register, which is where the LCD's controller looks for instructions on what to do next.

A Read/Write (R/W) pin that selects reading mode or writing mode

An Enable pin that enables writing to the registers

8 data pins (D0 -D7). The states of these pins (high or low) are the bits that you're writing to a register when you write, or the values you're reading when you read.

There's also a display contrast pin (Vo), power supply pins (+5V and GND) and LED Backlight (Bklt+ and BKlt-) pins that you can use to power the LCD, control the display contrast, and turn on and off the LED backlight, respectively.

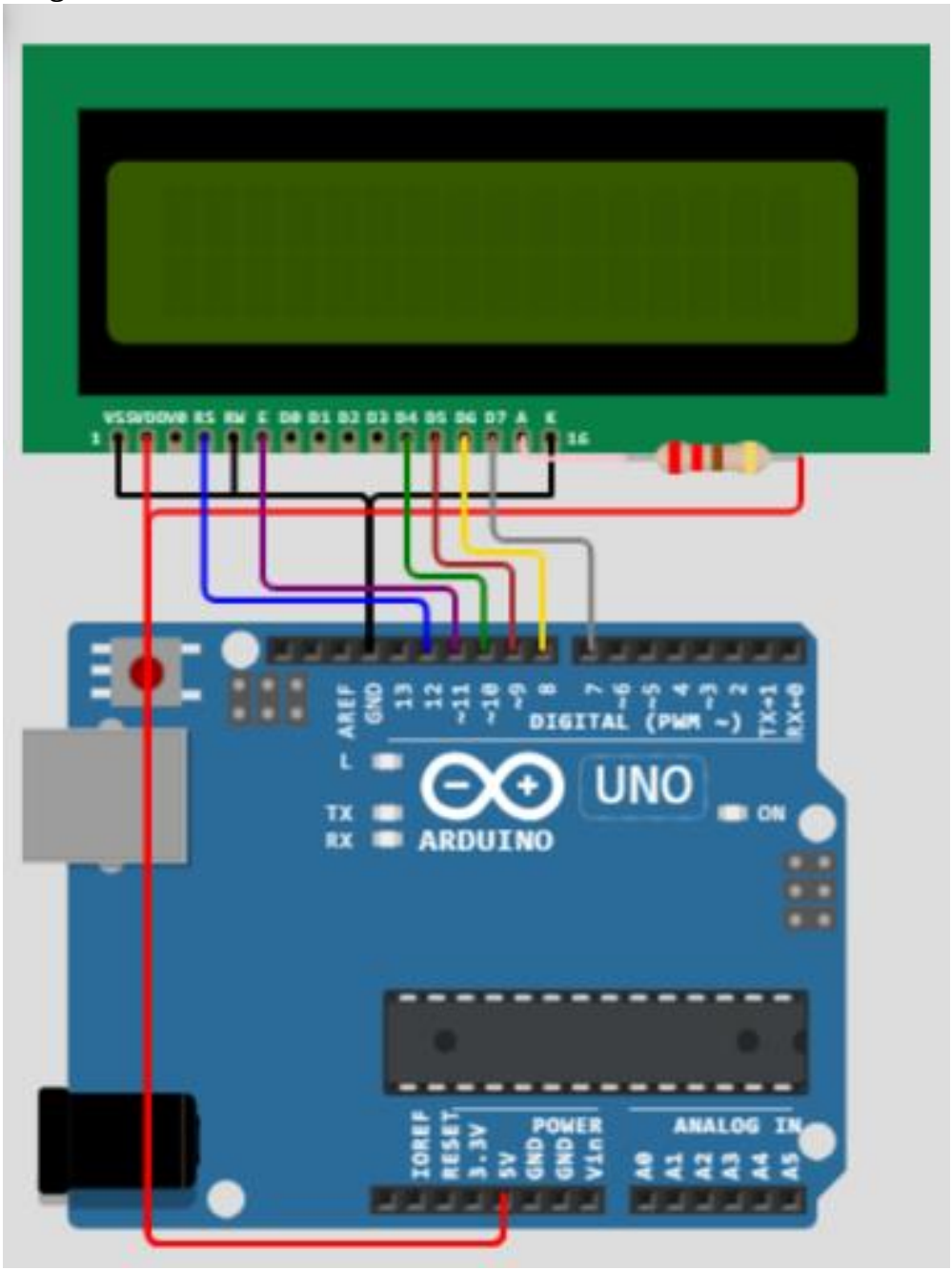
The process of controlling the display involves putting the data that form the image of what you want to display into the data registers, then putting instructions in the instruction register. The LiquidCrystal Library simplifies this for you so you don't need to know the low-level instructions. Hitachi-compatible LCDs can be controlled in two modes: 4-bit or 8-bit. The 4-bit mode requires seven I/O pins from the Arduino, while the 8-bit mode requires 11 pins. For displaying text on the screen, you can do most everything in 4-bit mode, so example shows how to control a 16x2 LCD in 4-bit mode.

Arduino and LCD Pin out

Name	Description	Arduino Pin*
VSS	Ground	GND.1
VDD	Supply voltage	5V
V0	Contrast adjustment (not simulated)	
RS	Command/Data select	12
RW	Read/Write. Connect to Ground.	GND.1

E	Enable	11
D0 – D3	Parallel data 0 - 3 (optional) †	
D4	Parallel data 4	10
D5	Parallel data 5	9
D6	Parallel data 6	8
D7	Parallel data 7	7
A	Backlight anode	5V / 6‡
K	Backlight cathode	GND.1

Wiring Diagram

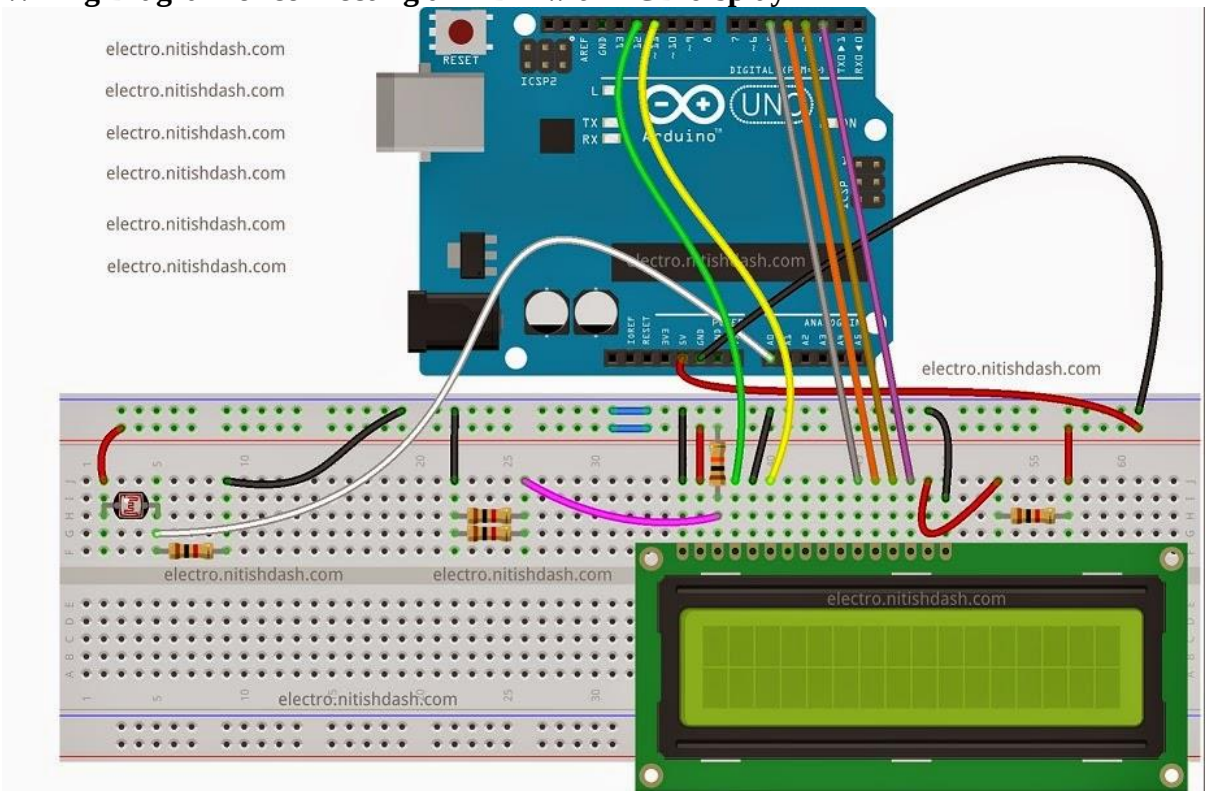


Simple Hello Program

```
#include <LiquidCrystal.h>
LiquidCrystal lcd(12, 11, 10, 9, 8, 7);
void setup() {
  lcd.begin(16, 2);
  // you can now interact with the LCD, e.g.:
  lcd.print("Hello World!");
}

void loop() {
  // ...
}
```

Wiring Diagram of connecting an LDR with LCD display



Code

```
/*  The circuit:
 * LCD RS pin to digital pin 12
 * LCD Enable pin to digital pin 11
 * LCD D4 pin to digital pin 10
 * LCD D5 pin to digital pin 9
 * LCD D6 pin to digital pin 8
 * LCD D7 pin to digital pin 7
 * LCD R/W pin to ground
 * LCD VSS pin to ground
 * LCD VCC pin to 5V
 * 10K resistor:  * ends to +5V and ground  * wiper to LCD VO pin (pin 3)
*/

 #include <LiquidCrystal.h>
LiquidCrystal lcd(7, 8, 9, 10, 11 , 12);
```

```
void setup() {
  lcd.begin(16, 2);
  lcd.setCursor(0,1);
  lcd.write("LIGHT: ");
}
void loop()
{
  int sensorValue = analogRead(A0);
  lcd.print("Room: ");
  if (sensorValue > 700)
  {
    lcd.print("Light!");
  }
  else
  {
    lcd.print("Dark  ");
  }
  delay(100);
}
```

Result

PASTE YOUR SCREENSHOT OF THE SUCCESSFUL RUN OF PROGRAM INCLUDING HARDWARE CONNECTION AND ARDUINO IDE

Experiment No. 11

Aim: Design an IoT application using any controller (Arduino Uno or Node MCU – ESP 8266), any sensor (IR sensor, RBG LED, Ultrasonic Sensor, DHT11, BMP290, LDR, LUX) and using any other accessories as required.

Experiment No. 12

Aim: Upload data from a single sensor to ThingSpeak using ESP8266 (Node MCU)

Components Required:

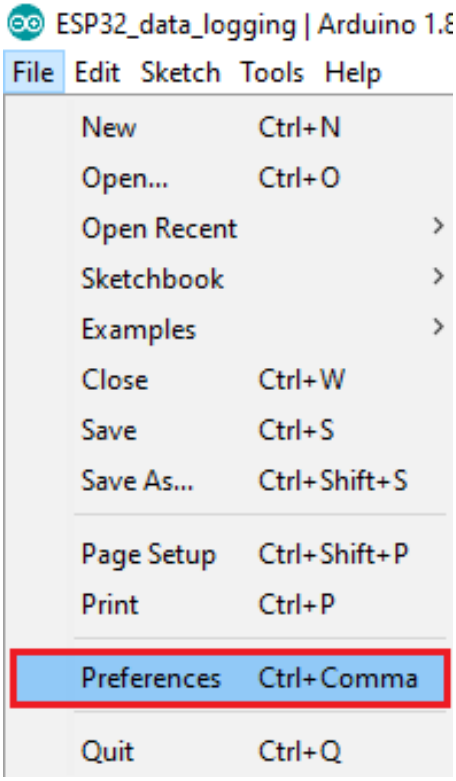
- ESP8266 (NodeMCU) development board
- DHT11 temperature and humidity sensor
- Breadboard and jumper wires
- USB cable for powering the NodeMCU
- A computer with Arduino IDE installed

Theory of DHT 11

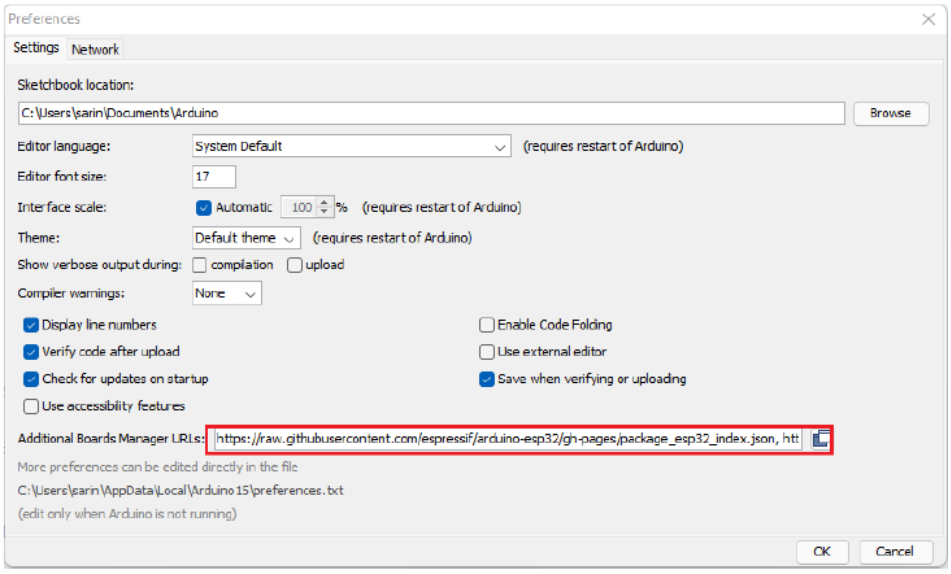
The DHT11 is a low-cost digital humidity and temperature sensor that uses a capacitive humidity sensor and a thermistor to measure the surrounding air's conditions. Its low price and compact size make it an attractive option for various applications, such as climate control systems, weather stations, and home automation. The sensor features a single-wire digital interface, making it easy to add to microcontroller-based projects. The data provided by the DHT11 sensor includes relative humidity and temperature readings, and it has a wide operating voltage range of 3.3V-5V.

To install the ESP32 board in your Arduino IDE, follow the following instructions:

i. In your Arduino IDE, go to File> Preferences

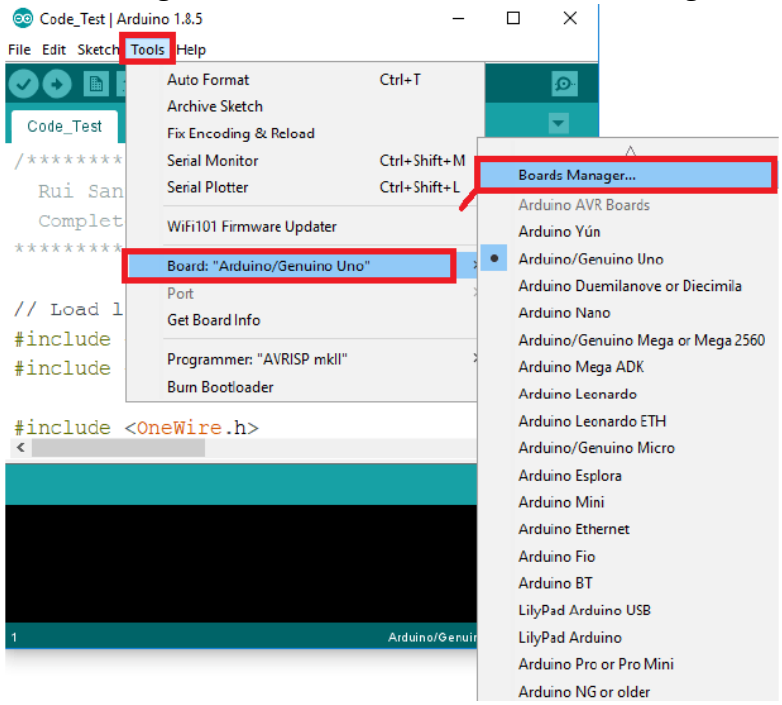


ii. Enter the following into the "Additional Board Manager URLs" field: https://raw.githubusercontent.com/espressif/arduino-esp32/gh-pages/package_esp32_index.json

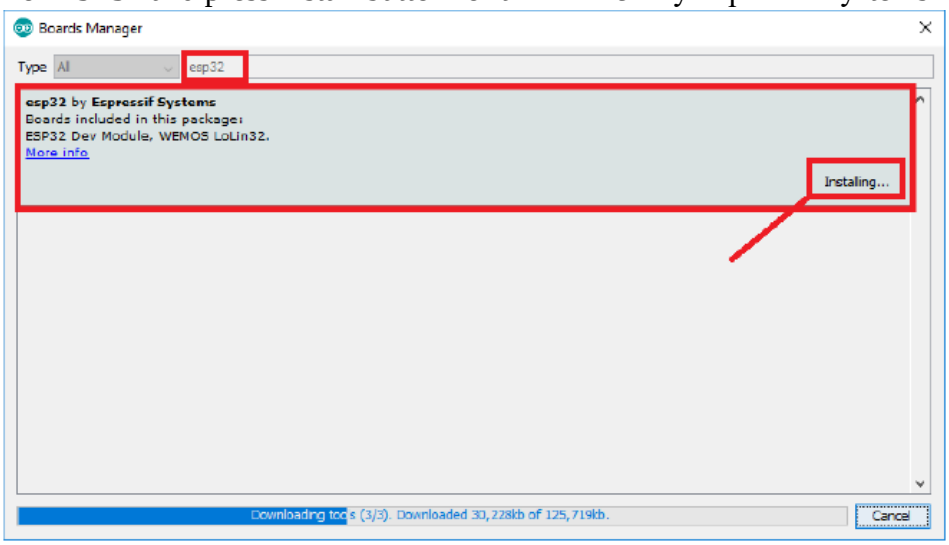


Then, click the "OK" button

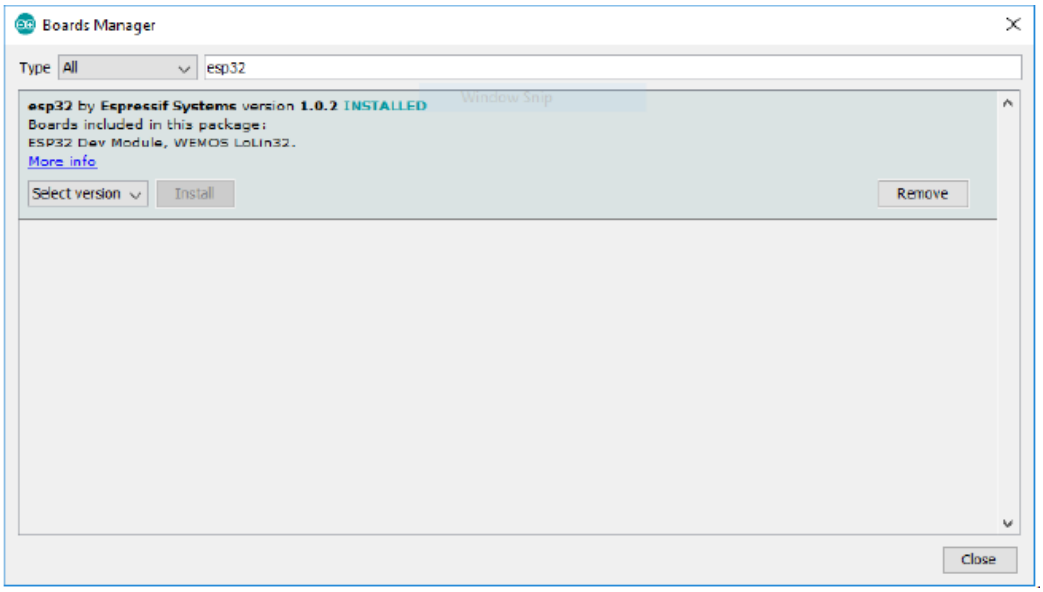
iii. Open the Boards Manager. Go to Tools > Board > Boards Manager



iv. Search for ESP32 and press install button for the "ESP32 by Espressif Systems"



v. That's it. It should be installed after a few seconds

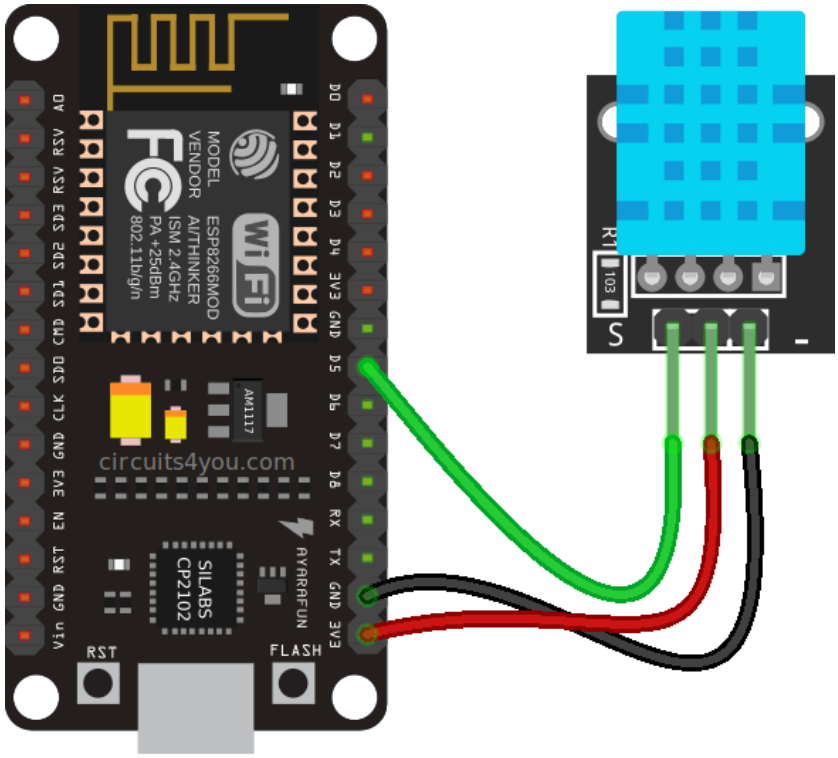


vi. Create an Account on ThingSpeak: Visit the ThingSpeak website (https://thingspeak.com) and create an account. Note down your ThingSpeak channel ID and API key, which will be used later to send data.

vii. Wire the components as shown in figure.

viii. Install Required Libraries Launch the Arduino IDE and install the necessary libraries by going to "Sketch" -> "Include Library" -> "Manage Libraries." Search for and install the following libraries: "DHT sensor library" by Adafruit, "ESP8266WiFi" by ESP8266 Community

Connection Diagram



Code Snippet

```
#include <ESP8266WiFi.h>
#include <DHT.h>

#define DHTPIN D4
#define DHTTYPE DHT11

const char* ssid = "YOUR_WIFI_SSID";
const char* password = "YOUR_WIFI_PASSWORD";
const char* server = "api.thingspeak.com";
const String apiKey = "YOUR_API_KEY";

DHT dht(DHTPIN, DHTTYPE);

WiFiClient client;

void setup() {
  Serial.begin(115200);
  delay(10);
  dht.begin();
  connectToWiFi();
}

void loop() {
  delay(2000);
  float temperature = dht.readTemperature();
  float humidity = dht.readHumidity();

  if (isnan(temperature) || isnan(humidity)) {
    Serial.println("Failed to read from DHT sensor!");
    return;
  }

  String url = "/update?api_key=" + apiKey + "&field1=" +
String(temperature) + "&field2=" + String(humidity);

  if (client.connect(server, 80)) {
    client.print(String("GET ") + url + " HTTP/1.1\r\n" +
                 "Host: " + server + "\r\n" +
                 "Connection: close\r\n\r\n");
    delay(10);
    client.stop();
    Serial.println("Data sent to ThingSpeak successfully!");
  } else {
    Serial.println("Connection to ThingSpeak failed!");
  }
}

void connectToWiFi() {
  Serial.print("Connecting to WiFi...");
```

```
  WiFi.begin(ssid, password);

  while (WiFi.status() != WL_CONNECTED) {
    delay(100);
  }

  Serial.println("Connected to WiFi!");
}
```

ix. Modify Wi-Fi and ThingSpeak Details: Replace "YOUR_WIFI_SSID" with the name of your Wi-Fi network, "YOUR_WIFI_PASSWORD" with your Wi-Fi password, and "YOUR_API_KEY" with your ThingSpeak API key.
x. Upload the Sketch
xi. Monitor the Serial Output: After the upload is complete, open the serial monitor by clicking "Tools" -> "Serial Monitor" in the Arduino IDE. Make sure the baud rate is set to 115200. You should see the NodeMCU connecting to the Wi-Fi network and sending data to ThingSpeak. Check the serial monitor for any error messages.
xii. Verify Data on ThingSpeak: Go to your ThingSpeak channel and open the corresponding channel view. You should see the temperature and humidity data being updated in real-time.

Experiment No. 13

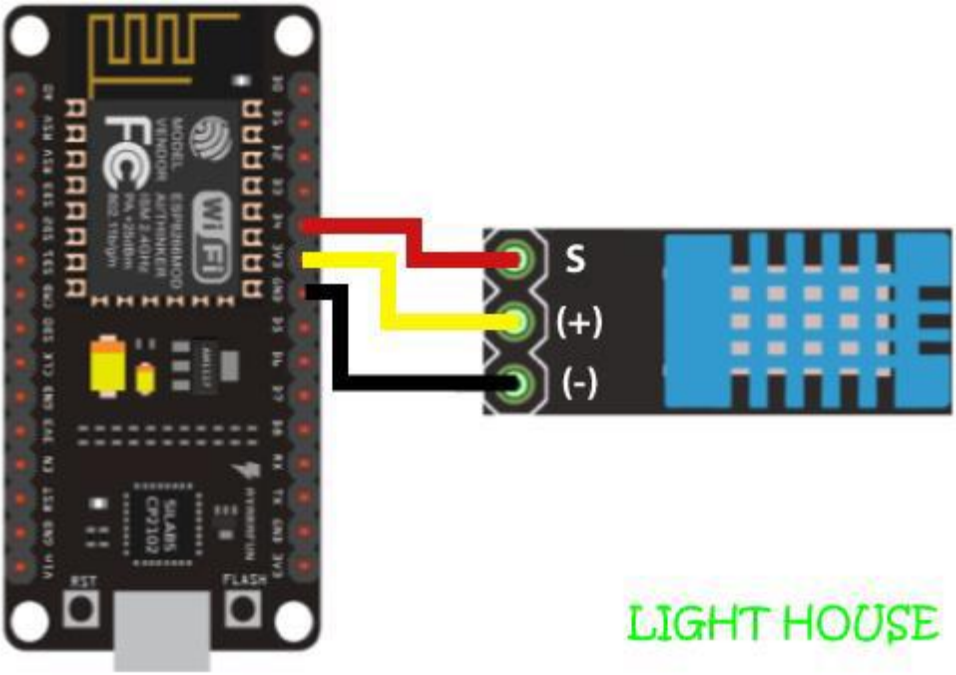
Aim: Interface DHT11 (Humidity Sensor) Using NodeMCU

Components Required

- Breadboard
- Micro USB Cable
- ESP8266 NodeMCU
- DHT11 Humidity and Temperature sensor
- Jumper Wires
- Arduino IDE

Wiring: Wire the components as follows:

- Connect the VCC pin of the DHT11 sensor to the 3.3V pin of the NodeMCU.
- Connect the GND pin of the DHT11 sensor to the GND pin of the NodeMCU.
- Connect the data pin of the DHT11 sensor to any GPIO pin of the NodeMCU.
- Connect the NodeMCU to your computer using a USB cable.



Code

```
#include <DHT.h>
#define DHTPIN 2
#define DHTTYPE DHT11

DHT dht(DHTPIN, DHTTYPE);
void setup() {
  Serial.begin(9600);
  dht.begin();
}
void loop() {
  delay(2000);
```

```
  float temperature = dht.readTemperature();
  float humidity = dht.readHumidity();

  if (isnan(temperature) || isnan(humidity)) {
    Serial.println("Failed to read from DHT sensor!");
    return;
  }

  Serial.print("Temperature: ");
  Serial.print(temperature);
  Serial.print(" °C\t");
  Serial.print("Humidity: ");
  Serial.print(humidity);
  Serial.println(" %");
}
```

Experiment No. 14

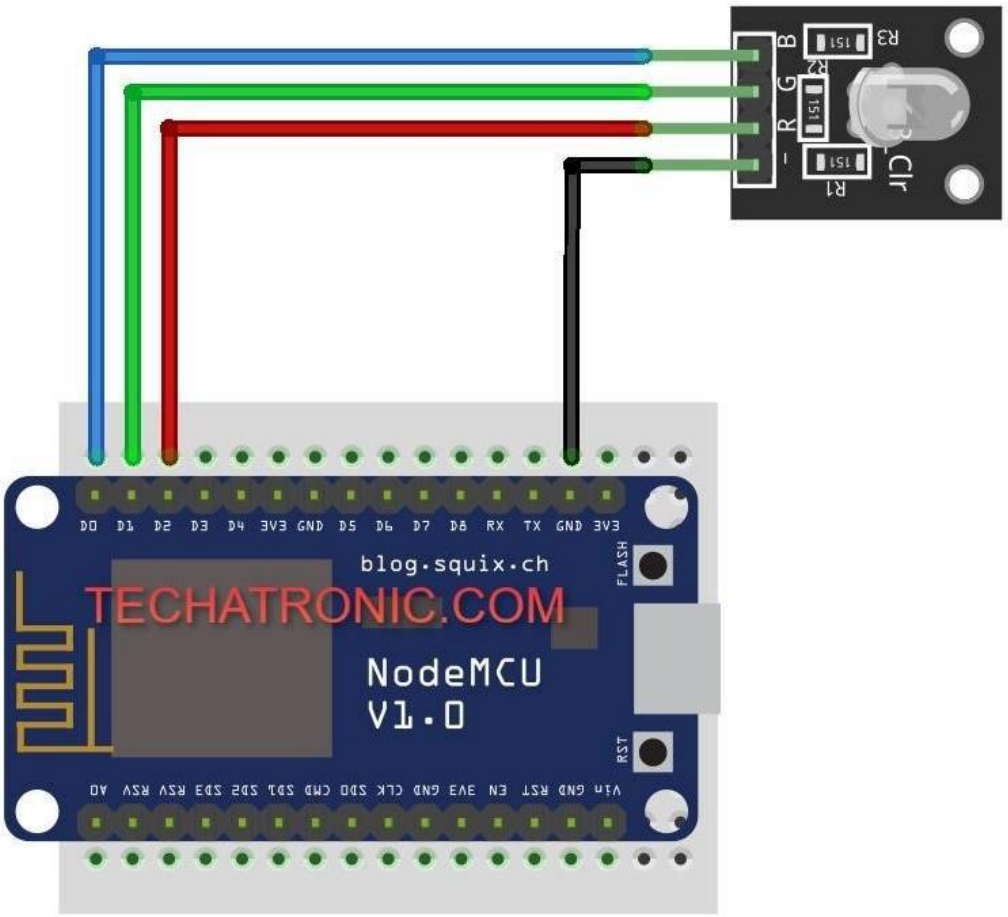
Aim: Interface RGB Led Using NodeMCU

Components Required

- Breadboard
- Micro USB Cable
- ESP8266 NodeMCU
- RGB Led sensor
- Jumper Wires
- Arduino IDE

Wiring: Wire the components as follows:

- GND pin of module – GND pin of NodeMCU.
- R pin (red light) of module – digital-2 pin of NodeMCU.
- G pin (green light) of module – digital-1 pin of NodeMCU.
- B pin (blue color) of module – digital-0 pin of NodeMCU.
- Connect the NodeMCU to your computer using a USB cable.



Code

```
#define RED_PIN    12     // Digital pin 2 of NodeMCU
#define GREEN_PIN  13     // Digital pin 1 of NodeMCU
#define BLUE_PIN   14     // Digital pin 0 of NodeMCU
```

```
void setup() {
  pinMode(RED_PIN, OUTPUT);
  pinMode(GREEN_PIN, OUTPUT);
  pinMode(BLUE_PIN, OUTPUT);
}

void loop() {
  setColor(255, 0, 0);   // Red
  delay(1000);
  setColor(0, 255, 0);   // Green
  delay(1000);
  setColor(0, 0, 255);   // Blue
  delay(1000);
  setColor(255, 255, 255);   // White
  delay(1000);
}

void setColor(int red, int green, int blue) {
  analogWrite(RED_PIN, red);
  analogWrite(GREEN_PIN, green);
  analogWrite(BLUE_PIN, blue);
}
```