UTKARSH MINDS
EMPOWER. EDUCATE. INNOVATE.

ROBOTICS (Basic and Advanced)

UTKARSH MINDS INSTITUTE
(MSME REGISTERED, MICROSOFT FOUNDERS PROGRAM AFFILIATE)
NIRMALA SADAN, KASTURBA CROSS ROAD NO. 1, BORIVALI EAST, MUMBAI 400068, MAHARASHTRA, INDIA
helpdesk@utkarshminds.com / +91 961-999-77-97

UTKARSH MINDS
EMPOWER. EDUCATE. INNOVATE.

Day	Basic Robotics
1	Fundamentals of C++
2	Arduino uno and components
3	Light LED, blinking, traffic signal
4	Temperature sensor
5	Buzzer
6	Light sensor
7	Potentiometer
8	Serial monitor and plotter
9	Resistor, Push button
	Capstone Project
	Advanced Robotics
1	Seven segment display
2	Ultrasonic sensor
3	Gear motors, L293d shield
4	Servo motors
5	RGB LED
6	ESP32
7	LCD display
8	Soldering

UTKARSH MINDS
EMPOWER. EDUCATE. INNOVATE.

	Capstone Project

Fundamental concepts of C programming

Understanding the fundamental concepts of C programming through small experiments is a great way to prepare yourself for writing more complex programs, such as controlling an LED on an Arduino Uno. Below are some simplified programming exercises that build towards the understanding required for blinking an LED with Arduino. Each exercise focuses on a basic concept of C programming and electronics that you will find useful.

1. "Hello, World!" on Your Computer

Objective: Learn the structure of a C program and how to output data.

```c
include <stdio.h>

int main() {
    printf("Hello, World!\n");
    return 0;
}
```

This helps you understand the basic structure of a C program and the `printf` function for output.

2. Simple Arithmetic Calculations

Objective: Familiarize yourself with variables and basic arithmetic operations.

```c
```

```
include <stdio.h>

int main() {
    int a = 5;
    int b = 10;
    int sum = a + b;
    printf("The sum of a and b is %d\n", sum);
    return 0;
}
```

This introduces variables and the arithmetic operations, helping you manipulate values in your programs.

3. Conditional Statements

Objective: Understand how to make decisions within your programs using `if-else` statements.

```c
include <stdio.h>

int main() {
    int ledState = 0; // 0 means OFF, 1 means ON

    if(ledState == 0) {
        printf("LED is OFF\n");
    } else {
        printf("LED is ON\n");
    }
    return 0;
```

```
}
```

This shows how to use conditions to take actions based on the state of variables, mimicking the decision of whether to turn an LED on or off.

4. Using Loops

Objective: Learn to use loops for repetitive tasks.

```c
include <stdio.h>

int main() {
    for(int i = 0; i < 5; i++) {
        printf("Blinking LED\n");
    }
    return 0;
}
```

This helps understand the use of loops to repeat actions, simulating the continuous blinking of an LED.

5. Functions

Objective: Understand how to modularize your code with functions.

```c
include <stdio.h>
```

```
void blinkLED() {
    printf("LED blinked\n");
}

int main() {
    for(int i = 0; i < 5; i++) {
        blinkLED();
    }
    return 0;
}
```

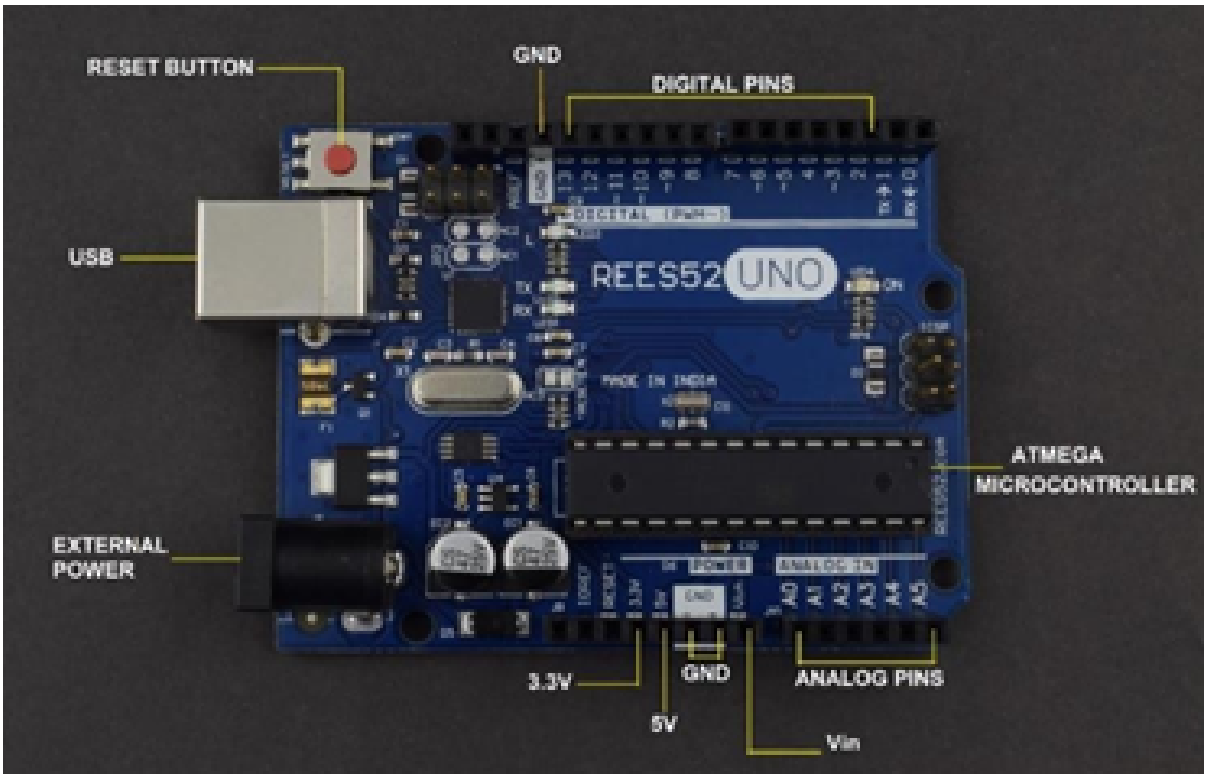
Teaches how to break down your program into reusable parts, critical for larger programs and mimics the separation of concerns (e.g., handling the blinking logic separately).

Translating to Arduino Blink

After understanding these concepts, you'll find it easier to grasp an Arduino program that involves blinking an LED:

```cpp
void setup() {
  pinMode(LED_BUILTIN, OUTPUT); // Initialize the LED pin as an output.
}

void loop() {
  digitalWrite(LED_BUILTIN, HIGH);   // Turn the LED on.
  delay(1000);                       // Wait for a second.
  digitalWrite(LED_BUILTIN, LOW);    // Turn the LED off.
  delay(1000);                       // Wait for a second.
}
```

UTKARSH MINDS
EMPOWER. EDUCATE. INNOVATE.

```

# Fundamental concepts of Arduino uno

The Arduino Uno is a popular open-source microcontroller board based on the ATmega328P microcontroller. It's designed for easy use in a variety of DIY electronics projects. Here's an overview of the basic parts that make up an Arduino Uno:


```

1. ATmega328 Microcontroller: This is the brain of the Arduino Uno, a microcontroller where your sketches (programs) run. It has 32KB of flash memory, 2KB of SRAM, and 1KB of EEPROM.

2. USB Interface: The USB port is used for programming the Arduino Uno from your computer and also can power the board. It connects to the ATmega16U2 chip, which converts USB signals to serial for the ATmega328.

3. Power Jack (Barrel Jack): This is used to power the Arduino Uno with an external power source, like a DC adapter, in the range of 7-12V.

4. Voltage Regulator: It regulates the voltage provided to the Arduino to ensure that it operates at a consistent voltage, typically 5V, even if the input voltage varies.

5. Reset Button: Pressing this button resets the microcontroller. It's useful for restarting the program from the beginning.

6. Digital I/O Pins (14 pins): These pins can be programmed to either read digital signals (high/low) as input or send out digital signals to control external devices.

7. Analog Input Pins (6 pins): These pins can read analog signals. The onboard ADC (Analog to Digital Converter) converts the analog signal to a digital value that can be used in your sketches.

8. PWM Pins: Among the digital I/O pins, some support PWM (Pulse Width Modulation), which can be used to simulate analog output (like dimming an LED).

9. TX/RX LEDs: These LEDs indicate the transmission (TX) and reception (RX) of data (like when programming the board).

10. ICSP Header (In-Circuit Serial Programming): This is used for programming the ATmega328 directly with an external programmer.

11. Crystal Oscillator: It provides the clock signal (16 MHz) that runs the microcontroller.

12. Power LED Indicator: Lights up when the Arduino Uno is powered on.

13. 3.3V and 5V Pins: These provide regulated voltage that can be used to power external components.

14. GND Pins: Ground pins are used to complete the circuit by providing a return path to the power source.

15. AREF Pin: Stands for Analog Reference. It is used to set an external reference voltage for the analog inputs.

These are the basic components that allow the Arduino Uno to interface with a wide variety of sensors, actuators, and other external components, making it a versatile choice for electronics projects.

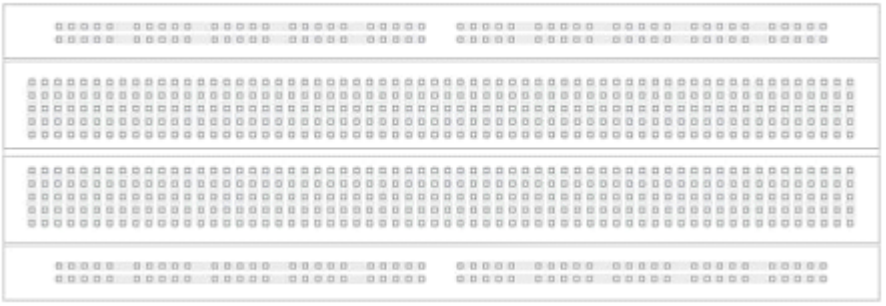
Know your components - Breadboard

- **A breadboard is a solderless device for temporary prototype with electronics and test circuit designs.**

  Most electronic components in electronic circuits can be interconnected by inserting their leads or terminals into the holes and then making connections through wires where appropriate.

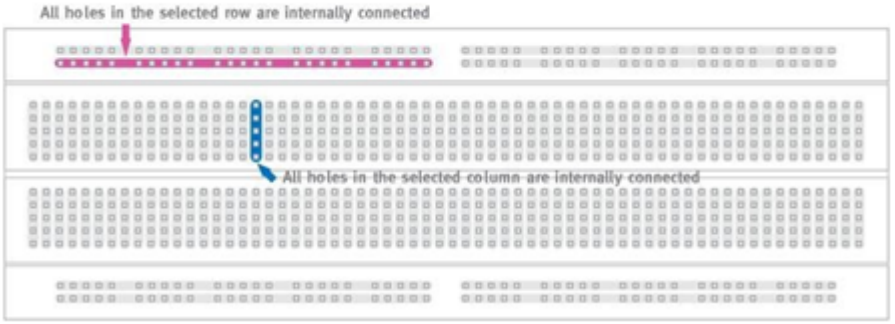
  The breadboard has strips of metal underneath the board and connect the holes on the top of the board. The metal strips are laid out as shown below.

  Note that the top and bottom rows of holes are connected horizontally and split in the middle while the remaining holes are connected vertically

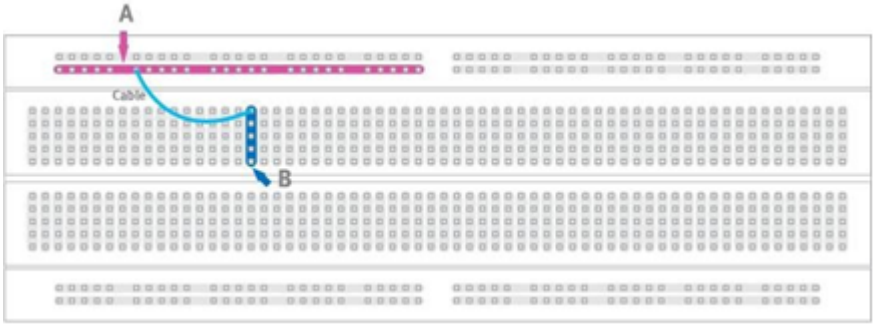


Note how all holes in the selected row are connected together, so the holes in the selected column. The set of connected holes can be called a node:

To interconnect the selected row (node A) and column (node B) a cable going from any hole in the row to any hole in the column is needed:



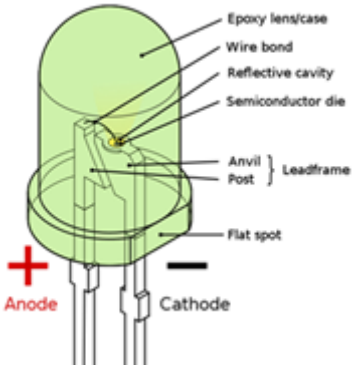
Now the selected column (node B) and row (node A) are interconnected



Know your components - LED

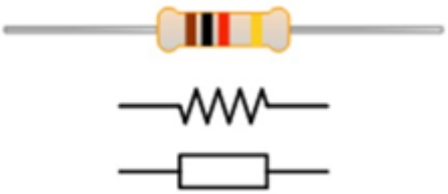
- **The “Light Emitting Diode” or LED as it is more commonly called, is basically just a specialized type of diode as they have very similar electrical characteristics to a PN junction diode.**

  This means that an LED will pass current in its forward direction but block the flow of current in the reverse direction. Flat Sport must be connected to GROUND



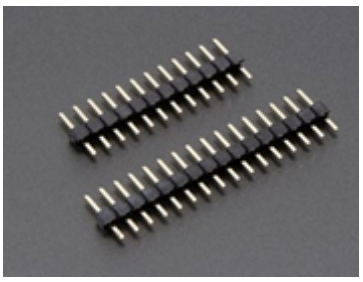
Know your components - Resistor

- **A resistor is a passive electrical component with the primary function to limit the flow of electric current**



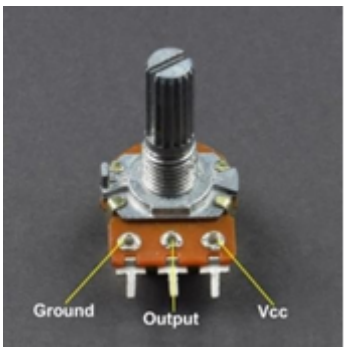
Know your components - Break Away Headers – Straight

- **Breakaway header is like the duct tape of electronics. Its great for connecting things together, soldering to perf-boards, fits into any breakout or breadboard,**



Know your components - 10K Potentiometer

- **You can connect VCC and GND on either first or third terminal of potentiometer. It is variable resistor which has no polarity. By default, GND Pin has 0Ω Value and VCC Pin has 10KΩ Value**



Know your components - Active VS Passive Buzzer

- **An active buzzer will generate a tone using an internal oscillator, so all that is needed is a DC voltage. A passive buzzer requires an AC signal to make a sound. It is like an electromagnetic speaker, where a changing input signal produces the sound, rather than producing a tone automatically.**

1. **Operating Voltage: 3.5-5.5V AC (Alternating current)**

   Rated Voltage: 5Vac

Current Consumption: 25mA

Resonant Frequency: 2300500Hz

Know your components -HC595 shift Register

- **Wide Supply Voltage Range from 2.0V to 6.0V**

  Sinks or sources 8mA at VCC = 4.5V

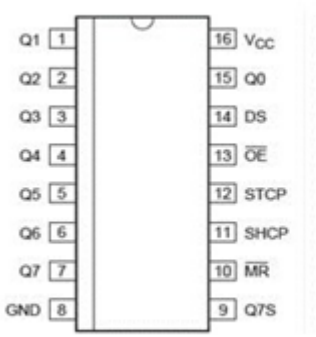
  CMOS low power consumption

  Schmitt Trigger Action at All Inputs

  Inputs accept up to 6.0V

  ESD Protection Tested per JESD 22

Know your components - IR Sensor with HX1838 IR Remote



Symbol	Pin	Description
Q1	1	parallel data output 1
Q2	2	parallel data output 2
Q3	3	parallel data output 3
Q4	4	parallel data output 4
Q5	5	parallel data output 5
Q6	6	parallel data output 6
Q7	7	parallel data output 7
GND	8	ground (0 V)
Q7S	9	serial data output
$\overline{MR}$	10	master reset (active LOW)
SHCP	11	shift register clock input
STCP	12	storage register clock input
$\overline{OE}$	13	output enable input (active LOW)
DS	14	serial data input
Q0	15	parallel data output 0
V_{CC}	16	supply voltage

- **Remote Specifications:**

1. **Works on CR2025 button batteries, capacity : 160 mah**

   Working Distance: more than 8 m (effected by the surrounding environment, the receiver sensitivity, etc)

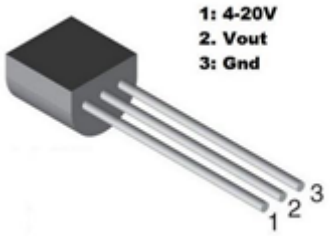
2. **Effective Angle: 60 degrees**
3. **Surface materials: 0.125 mm PET stick**
4. **Effective button life: 20000 times.**
5. **Static current: 3-5 uA**
6. **Dynamic current: 3-5 mA**
7. **Infrared wavelength: 940Nm**
8. **Crystal: the oscillation frequency of 455 KHz**
9. **IR carrier frequency: 38KHz**
10. **Encoding: NEC encoding format**

- **Infrared Receiver Specifications: An infrared receiver tuned to react only to IR of frequency 38 kHz. This IR sensor module consists of a PIN diode and a pre amplifier which are embedded into a single package. The output of TSOP is active low and it gives +5V in off state. When IR waves, from a source, with a center frequency of 38 kHz incident on it, its output goes low. It is perfect for making obstacle sensors and to accept signals from most IR remotes.**

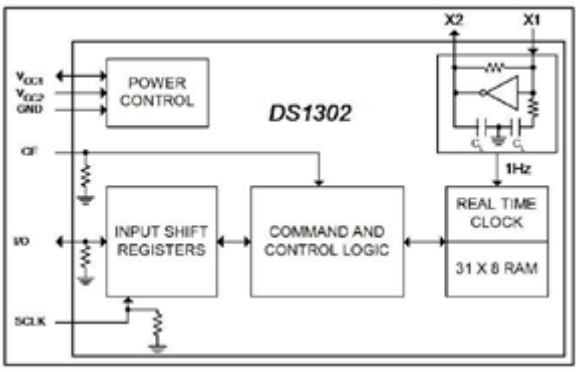
Know your components - LM35 Temperature Sensor

- **The LM35 is one kind of commonly used temperature sensor that can be used to measure temperature with an electrical o/p comparative to the temperature (in °C).**

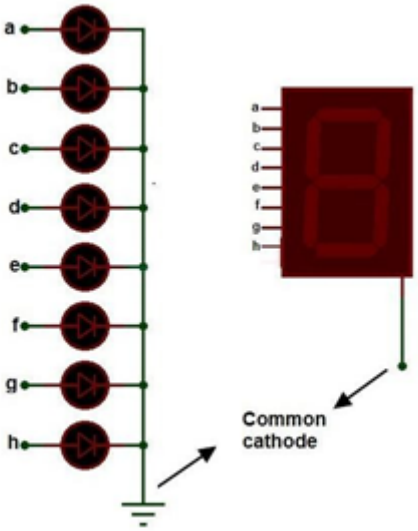
- **It can measure temperature more correctly compare with a thermistor.**
- **This sensor generates a high output voltage than thermocouples and may not need that the output voltage is amplified.**
- **The LM35 has an output voltage that is proportional to the Celsius temperature. The scale factor is .01V/°C.**

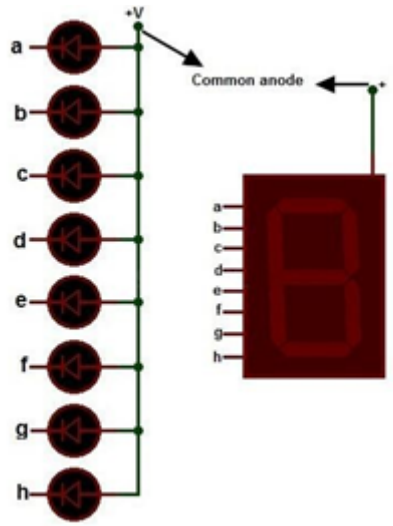


Know your components - DS1302 Real Time Clock



Know your components - 7 Segment Display

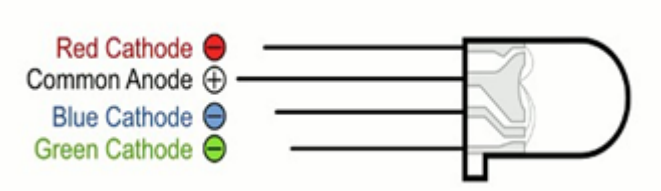




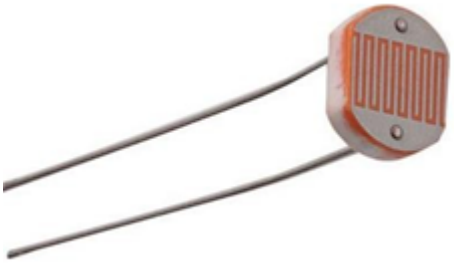
Know your components - Flame Sensor

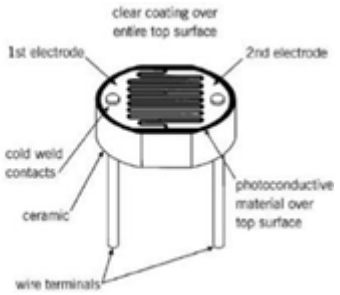
UTKARSH MINDS
EMPOWER. EDUCATE. INNOVATE.

Know your components - RGB Led

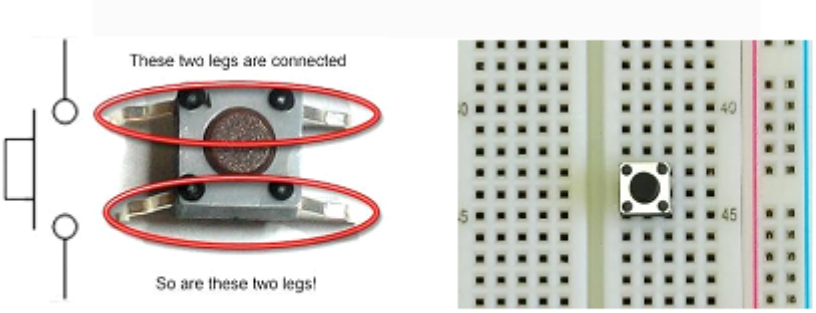


Know your components - LDR

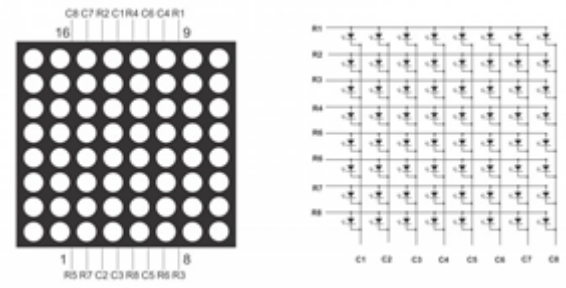




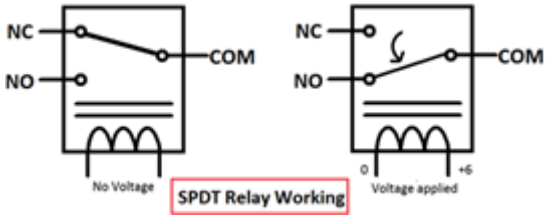
Know your components - Tactile Push Button Switch



Know your components - 88 Dot Matrix Module

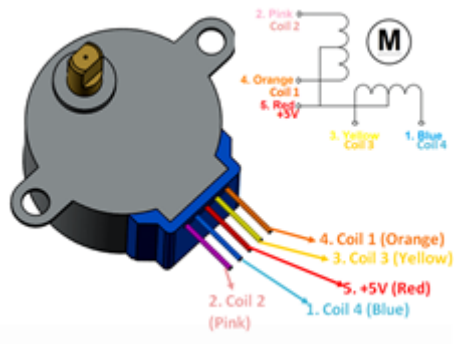


Know your components - 1 channel 5V relay Module

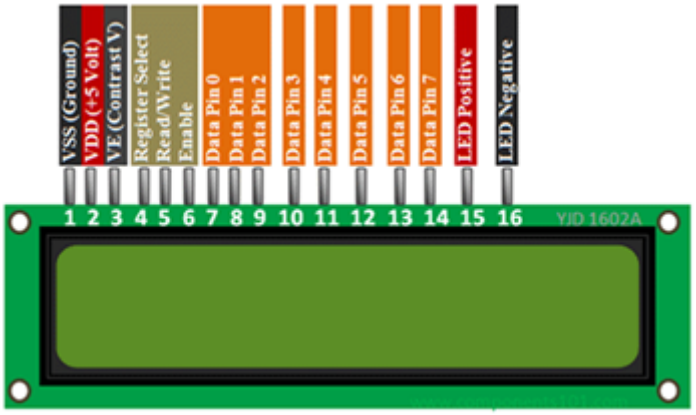


Know your components - ULN2003 Driver Module

Know your components - 5V Stepper Motor



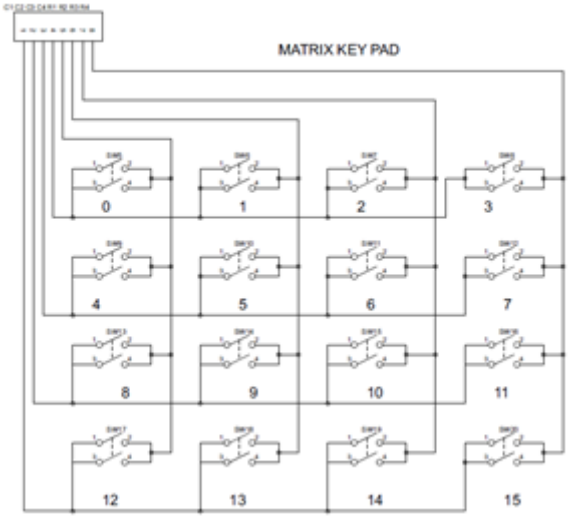
Know your components - 16x2 LCD



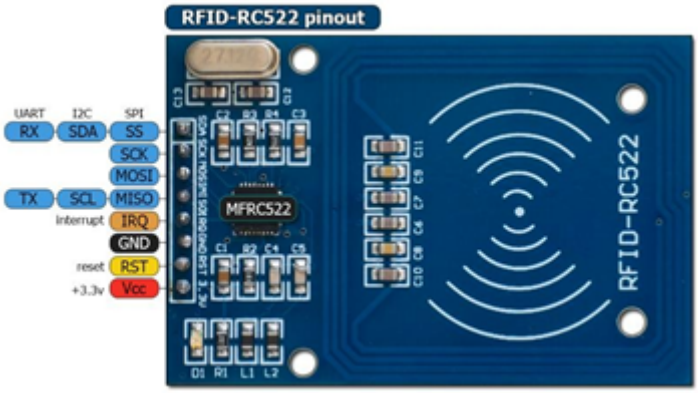
Know your components - Servo motor

UTKARSH MINDS
EMPOWER. EDUCATE. INNOVATE.

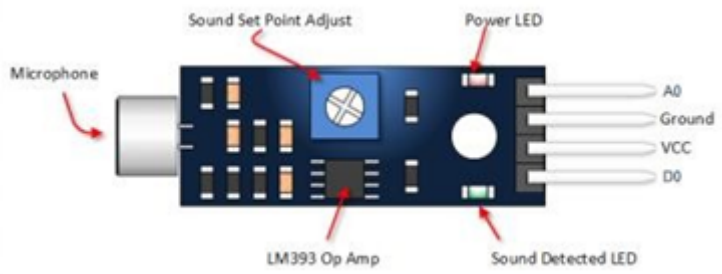
Know your components - Matrix keyboard module



Know your components - RFID Card module



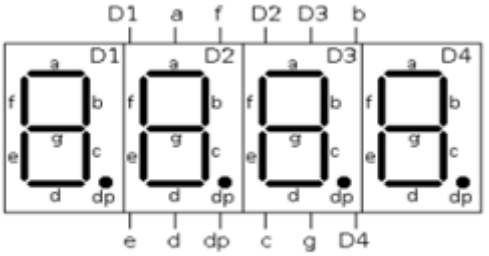
Know your components - Sound Sensor



Know your components - I2C Display

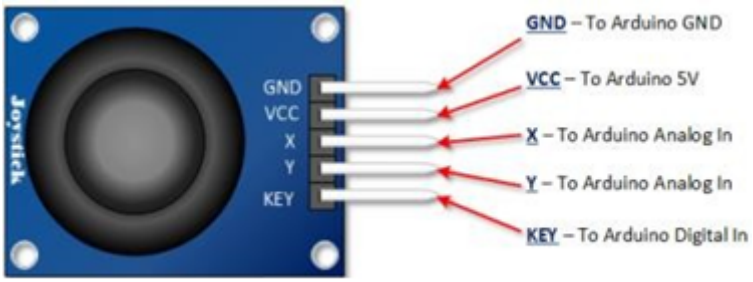
- **LCD Display utilizes an I2C interface, which means that fewer pins are necessary to use this product than would be needed with a regular 16x2 LCD Display (just four connections, VCC, GND, SDA & SCL are required).**

Know your components - 4 Digit (7 Segment) Display

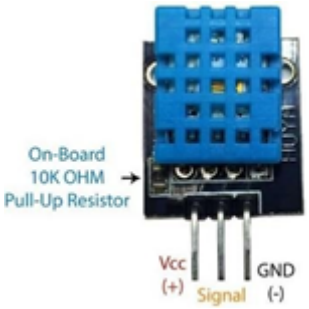


Know your components - Joystick Module

- **A joystick is an input device consisting of a stick that pivots on a base and reports its angle or direction to the device it is controlling. The Joystick Module is a tiny low cost option to add a game like control to your projects. The module has 2 potentiometers for the X and Y axis measurements. Also a switch that can give an additional control option**



Know your components - DHT-11 Temperature Humidity Sensor



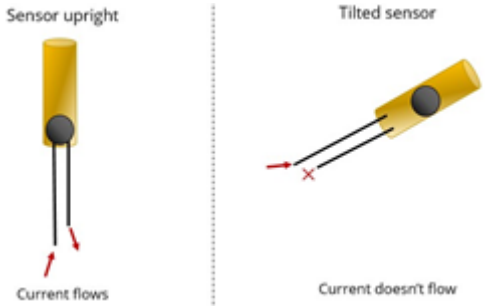
Know your components - Water Level Measurement Sensor

S = Signal Pin (connects to an analog pin on the arduino)
+ = Positive Voltage (connects to +5V terminal on arduino)
- = Ground (connects to ground terminal on arduino)

UTKARSH MINDS
EMPOWER. EDUCATE. INNOVATE.

Know your components - Tilt Sensor

- **Tilt switches transfer a change-of-state to another device. These devices receive a signal from the tilt sensor for changes in motion or orientation and turn on or off. They do this by generating an artificial horizon and measuring angular tilt with respect to this horizon. Not all products open or close a switch. Such as tilt switch alarms, trigger audible or visual responses to notify an operator that a system is out of alignment**



UTKARSH MINDS
EMPOWER. EDUCATE. INNOVATE.

Experiment no - 1

A basic experiment you can start with on an Arduino Uno involves blinking an LED. This experiment is often considered the "Hello, World!" of microcontroller projects. It's simple, requires minimal components, and teaches you the basics of Arduino programming and how to control an output.

What You Need:

- Arduino Uno
- Breadboard
- LED
- 220Ω Resistor
- Jumper Wires

Circuit Setup:

1. Power Off: Start by ensuring that your Arduino Uno is disconnected from your computer or power source.

2. Connect LED: Insert your LED into the breadboard. Note that LEDs have a longer leg (anode, +) and shorter leg (cathode, -).

3. Add Resistor: Connect the 220Ω resistor to the anode (+) side of the LED. The resistor limits current to the LED, protecting it.

4. Wire to Arduino: Connect the other end of the resistor to pin 13 on the Arduino Uno. Then, connect the cathode (-) of the LED (the shorter leg) to one of the GND (ground) pins on the Arduino.

5. Connect Arduino: Now, carefully connect your Arduino Uno to your computer using a USB cable. The board will power on.

Arduino Sketch (Program):

1. Open the Arduino IDE on your computer. If you don't have it installed, download it from the [official Arduino website](https://www.arduino.cc/en/software).

2. Create a New Sketch: Go to File > New to open a new sketch.

3. Write the Code: Enter the following program into the IDE:

```cpp
void setup() {
  // initialize digital pin LED_BUILTIN as an output.
  pinMode(LED_BUILTIN, OUTPUT);
}

void loop() {
  digitalWrite(LED_BUILTIN, HIGH);   // turn the LED on (HIGH is the voltage level)
  delay(1000);                       // wait for a second
  digitalWrite(LED_BUILTIN, LOW);    // turn the LED off by making the voltage LOW
  delay(1000);                       // wait for a second
}
```

4. Upload to Arduino: With your Arduino connected and the correct board and port selected in the Arduino IDE (Tools > Board > Arduino Uno and Tools > Port), click the Upload button (right arrow icon). The IDE will compile the sketch and upload it to your Arduino Uno.

5. Observation: After the upload completes, your LED should start blinking. It will be on for one second, then off for one second, repeatedly.

Explanation:

This program demonstrates basic digital output control. The `pinMode()` function in the `setup()` block configures the built-in LED connected to pin 13 as an output. The `loop()` block then repeatedly turns the LED on and off, with one-second pauses, by alternating digital writes to HIGH and LOW.

This simple experiment introduces you to digital outputs, delays, and the basic structure of an Arduino sketch, including the setup and loop functions. From here, you can start exploring more complex projects, such as reading inputs with buttons, controlling motors, or even building simple robots.

Alright! Let's dive into the Arduino blinking experiment, which is a classic project for beginners. It's like making a digital version of a blinking light, similar to how a lighthouse blinks to guide ships. This simple project is not only fun but also introduces you to the basics of programming and electronics.

Theory Behind the Arduino Blinking Experiment

The core idea of the blinking experiment is to turn an LED (Light Emitting Diode) on and off in a pattern, using an Arduino Uno as the brain behind the operation. An Arduino Uno is a microcontroller board based on the ATmega328P. It reads the instructions (code) you upload to it and performs actions, like controlling an LED.

When you write and upload code to the Arduino that tells it to turn the LED on (by sending electrical power to the LED), wait for a bit, and then turn the LED off (by stopping the power), you create a blinking effect. Then, by waiting again before the cycle repeats, you control the speed of the blinking.

Parts of an Arduino Uno

1. Microcontroller (ATmega328P): This is the brain of the Arduino, where your program (sketch) runs.

2. Digital I/O Pins (14 total) - These pins can read (input) or write (output) digital signals that are either HIGH (5V) or LOW (0V). They are used to connect sensors, motors, LEDs, etc.

3. Analog Input Pins (6 total): These pins can read analog signals, converting them into digital values that can be processed by the microcontroller.

4. USB Connection: This is how you connect the Arduino to a computer to upload code, and it can also power the Arduino.

5. Power Jack: You can also power the Arduino through this using an external power source (7-12V).

6. Reset Button: Resets the microcontroller. This is used to restart your program, for instance.

7. TX/RX LEDs: These blink when data is being transmitted through the USB connection to or from the computer.

8. Built-in LED: Most Arduino Uno boards have a built-in LED (usually on digital pin 13) that you can control.

9. Voltage Regulator: Regulates the voltage provided to the board ensuring it's within safe limits.

Different Apparatus Required for the Blinking Experiment

1. Arduino Uno Board: The central component where all the magic happens.

2. LED: You can use the built-in LED or an external LED.

3. Resistor (220 ohms or 330 ohms): If you're using an external LED, you'll need a resistor to protect the LED from receiving too much current.

4. USB Cable: To connect the Arduino to your computer for coding and power.

5. Breadboard (optional): If you're using an external LED and resistor, a breadboard makes it easier to connect everything without soldering.

6. Jumper Wires (optional): Needed to connect components on the breadboard, or to connect an external LED directly to the Arduino.

The Basic Steps

1. Setup: Connect your Arduino to your computer with the USB cable. If using an external LED, connect it to one of the digital I/O pins (e.g., pin 13) through a resistor. The long leg (anode) of the LED connects to the digital pin and the short leg (cathode) connects to the GND (ground) through the resistor.

2. Programming: Using the Arduino IDE (Integrated Development Environment) on your computer, you write a sketch (the Arduino term for a program) that tells the LED to turn on (HIGH), wait (using `delay()`), turn off (LOW), and wait again. This code is then uploaded to the Arduino Uno.

3. Executing: After uploading, the Arduino runs your code and you see the LED blinking.

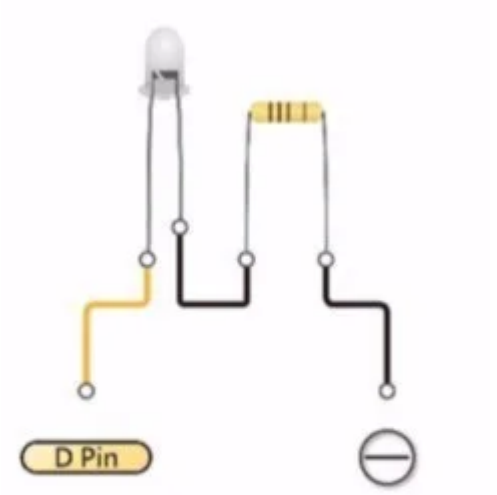
This experiment teaches basic coding and electronics concepts, like controlling output devices (LEDs), using delays, and understanding circuit basics. It's a great starting point for more complex projects as you get more comfortable with microcontrollers and coding.

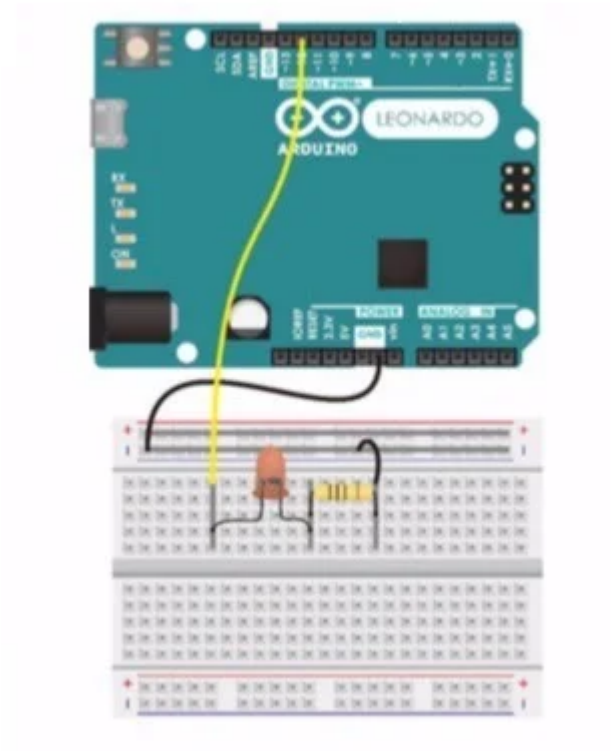
Aim: To make an LED blink using Arduino Uno.

Apparatus:

- Arduino Uno board
- USB cable for Arduino
- 1 LED light
- 1 220-ohm resistor
- Breadboard
- Jumper wires

Theory: So, we're going to use an Arduino Uno to make an LED blink. It might sound simple, but it's actually super cool because it's like the first step into the world of creating stuff with electronics and coding. Arduino Uno is like the brain behind the operation. We write a code (like a set of instructions) and upload it to the Arduino. This code tells the LED when to switch on and off, which makes it blink. The reason we use a resistor is to make sure too much electricity doesn't blast through the LED and fry it. It's all about controlling the flow.





Code:

```cpp
// Define pin number
int ledPin = 13;

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

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

```
  digitalWrite(ledPin, LOW);    // turn the LED off by making the voltage LOW
  delay(1000);                  // wait for a second
}
```

Steps:

1. Connect your Arduino Uno to your computer using the USB cable.

2. Assemble your circuit: Plug one end of the resistor into the pin you're using for your LED (pin 13 in the code), and the other end into one of the breadboard rows. Then connect your LED to the breadboard ensuring the long leg of the LED (the anode) is in the same row as the resistor and the short leg (the cathode) is in another row.

3. Use jumper wires to connect the row with the anode (and the resistor) to pin 13 on the Arduino and the cathode to one of the GND (ground) pins on the Arduino.

4. Now, open the Arduino IDE on your computer. If you don’t have it yet, you'll need to download it from https://www.arduino.cc/en/Main/Software.

5. Type the given code in the Arduino IDE. Don’t worry, it's just telling the Arduino to turn the LED on and off with a delay in between.

6. Select the correct board and port through the Tools menu in the IDE.

7. Upload the code to your Arduino by clicking the "Upload" button.

8. If everything is set up right, your LED should start blinking. Congrats, you did it!

Summary: In this experiment, we successfully made an LED blink using an Arduino Uno. By writing and uploading a simple code to the Arduino, we were able to control the LED's state (on and off) with a delay in between. This project is a basic introduction to electronics and coding, showing how software and hardware interact with each other. It opens the door to more complex projects and is a great start to dive into the world of Arduino and DIY electronics.

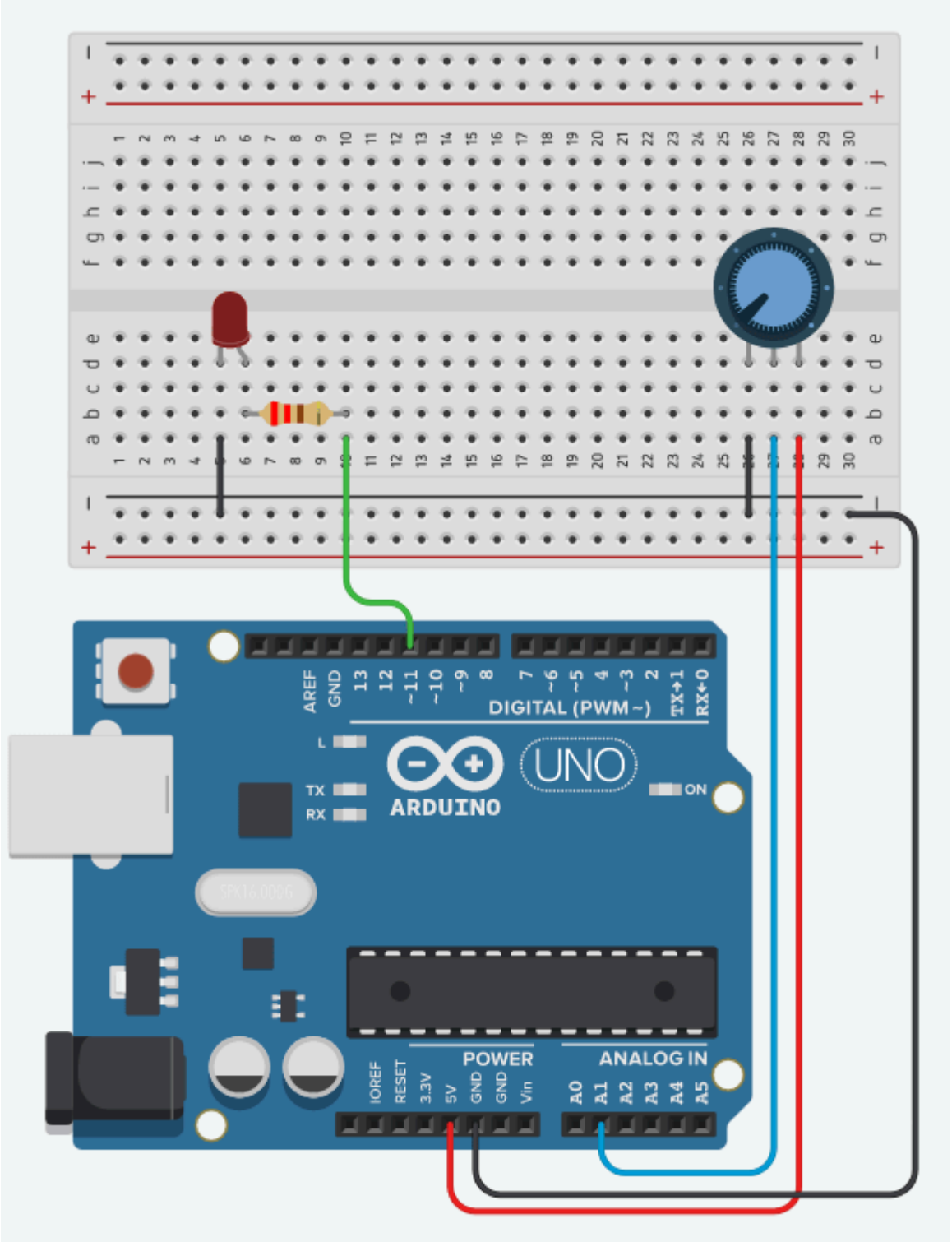
AIM: To make an LED's brightness control using Arduino Uno along with a potentiometer.

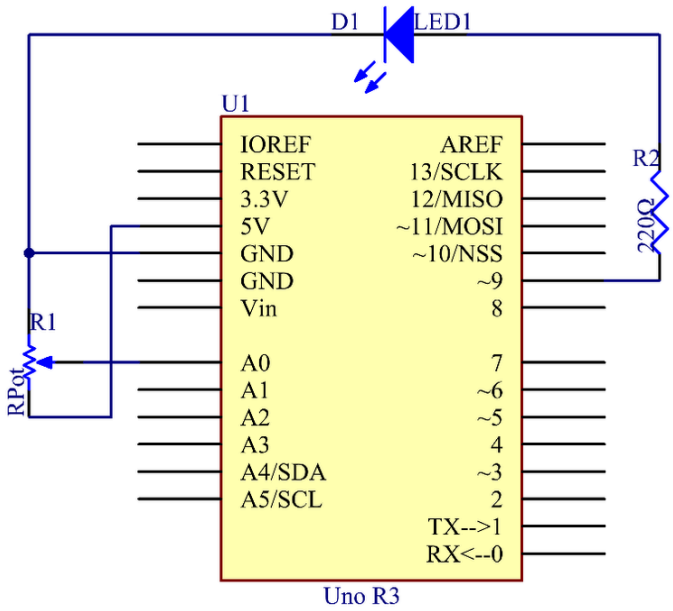
APPARATUS

- Arduino Uno board
- USB Cable (for connecting Arduino to the computer)
- 1 LED
- 1 220Ω resistor (for the LED)
- 1 10kΩ potentiometer
- Breadboard
- Jumper wires

THEORY

The Arduino Uno is a microcontroller board based on the ATmega328P. It has digital and analog input/output (I/O) pins that can be interfaced with various expansion boards (shields) and other circuits. In this experiment, we're going to use a potentiometer (variable resistor) to change the LED's brightness. The Arduino reads the voltage variation from the potentiometer (using one of its analog input pins) and then adjusts the brightness of the LED (connected to one of its PWM digital output pins) accordingly. PWM (Pulse Width Modulation) allows us to simulate analog output like LED dimming.


AREF
GND
13
12
~11
~10
~9
8
7
~6
~5
4
~3
2
TX→1
RX←0
DIGITAL (PWM~)
L
TX
RX
ARDUINO
UNO
ON
POWER
ANALOG IN
IOREF
RESET
3.3V
5V
GND
GND
Vin
A0
A1
A2
A3
A4
A5



STEPS

1. Setup the Circuit:

- Connect the potentiometer to the breadboard. Connect one side pin to 5V on the Arduino and the opposite side pin to GND. Connect the middle pin to analog pin A0 on the Arduino.

- Connect the LED to the breadboard. Attach the long leg (anode) to digital pin 9 through a 220Ω resistor. Connect the short leg (cathode) directly to GND on the Arduino.

- Double-check your connections!

2. Programming:

- Connect the Arduino Uno to your computer using the USB cable.

- Open the Arduino IDE on your computer. If you don't have it, download it from the official Arduino website.

3. Type the Following Code:

```cpp
int ledPin = 9; // LED connected to digital pin 9

int analogPin = A0; // potentiometer connected to analog pin A0

int val = 0; // variable to store the value read
```

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

void loop() {
  val = analogRead(analogPin); // read the input pin
   analogWrite(ledPin, val / 4); // analogRead values go from 0 to 1023, analogWrite values from 0 to 255
}
```

4. Upload the Code:

- In the Arduino IDE, select the correct board and port under Tools.
- Click the "Upload" button to send your code to the Arduino.

5. Test It Out:

- Once the code is uploaded, turn the potentiometer knob. You should see the LED's brightness change in response.

SUMMARY

In this experiment, we successfully controlled the brightness of an LED using a potentiometer and Arduino Uno. By adjusting the potentiometer, we changed the voltage read by an analog input pin (A0) on the Arduino, which in turn, using PWM on a digital output pin (pin 9), altered the brightness of the LED. This project demonstrates the basic principle of analog input and PWM output using Arduino, which can be the foundation for more complex projects involving sensor inputs and variable outputs.

Analog Read Serial

Read a potentiometer, print its state out to the Arduino Serial Monitor.

This example shows you how to read analog input from the physical world using a potentiometer. A **potentiometer** is a simple mechanical device that provides a varying amount of resistance when its shaft is turned. By passing voltage through a potentiometer and into an analog input on your board, it is possible to measure the amount of resistance produced by a potentiometer (or *pot* for short) as an analog value. In this example you will monitor the state of your potentiometer after establishing serial communication between your Arduino and your computer running the Arduino Software (IDE).

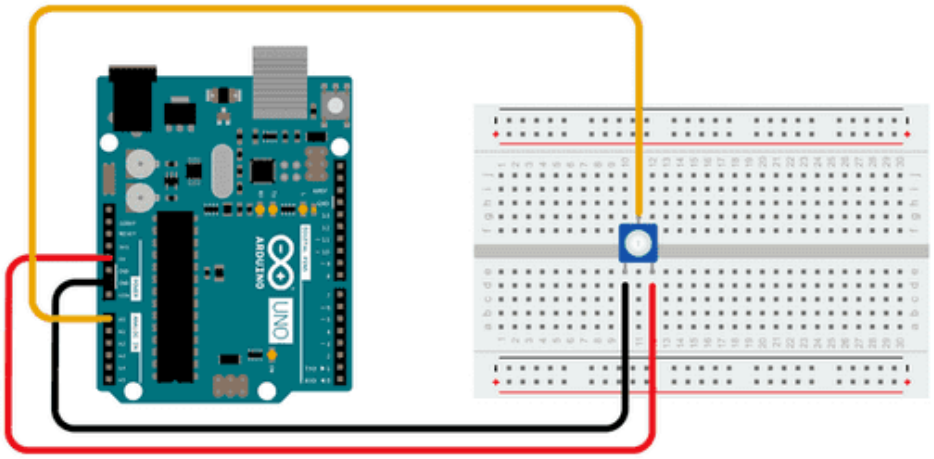
Hardware Required

Arduino Board
10k ohm Potentiometer

Circuit

Connect the three wires from the potentiometer to your board. The first goes from one of the outer pins of the potentiometer to ground. The

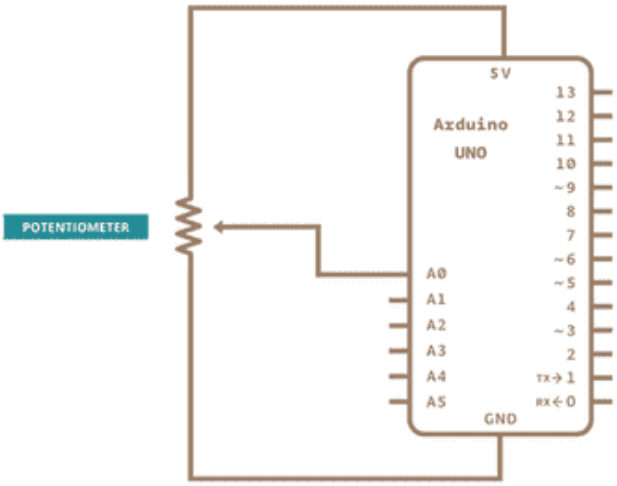
second goes from the other outer pin of the potentiometer to 5 volts. The third goes from the middle pin of the potentiometer to the analog pin A0.



By turning the shaft of the potentiometer, you change the amount of resistance on either side of the wiper, which is connected to the center pin of the potentiometer. This changes the voltage at the center pin. When the resistance between the center and the side connected to 5 volts is close to zero (and the resistance on the other side is close to 10k ohm), the voltage at the center pin nears 5 volts. When the resistances are reversed, the voltage at the center pin nears 0 volts, or ground. This voltage is the *analog voltage* that you're reading as an input.

The Arduino boards have a circuit inside called an *analog-to-digital converter or ADC* that reads this changing voltage and converts it to a number between 0 and 1023. When the shaft is turned all the way in one direction, there are 0 volts going to the pin, and the input value is 0. When the shaft is turned all the way in the opposite direction, there are 5 volts going to the pin and the input value is 1023. In between, analogRead() returns a number between 0 and 1023 that is proportional to the amount of voltage being applied to the pin.

Schematic



Code

In the sketch below, the only thing that you do in the setup function is to begin serial communications, at 9600 bits of data per second, between your board and your computer with the command:

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

Next, in the main loop of your code, you need to establish a variable to store the resistance value (which will be between 0 and 1023, perfect for an

`int`datatype) coming in from your potentiometer:

```
Unset
int sensorValue = analogRead(A0);
```

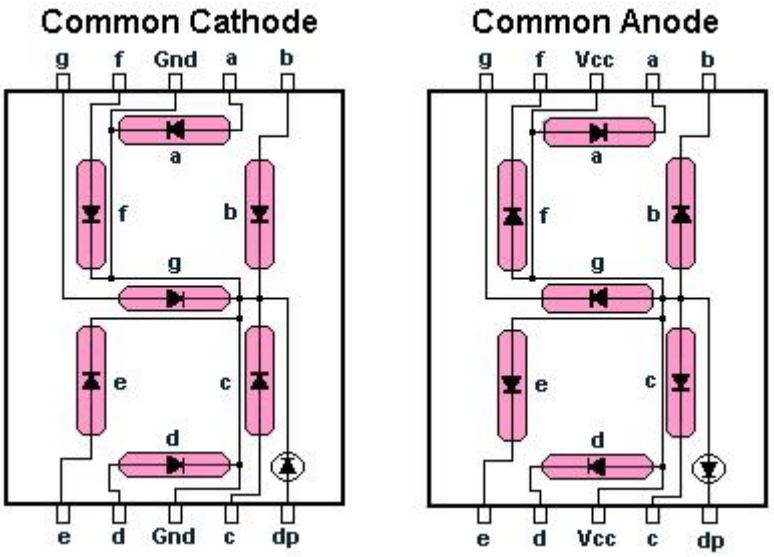
Finally, you need to print this information to your serial monitor window. You can do this with the command Serial.println() in your last line of code:

```
Unset
Serial.println(sensorValue);
```

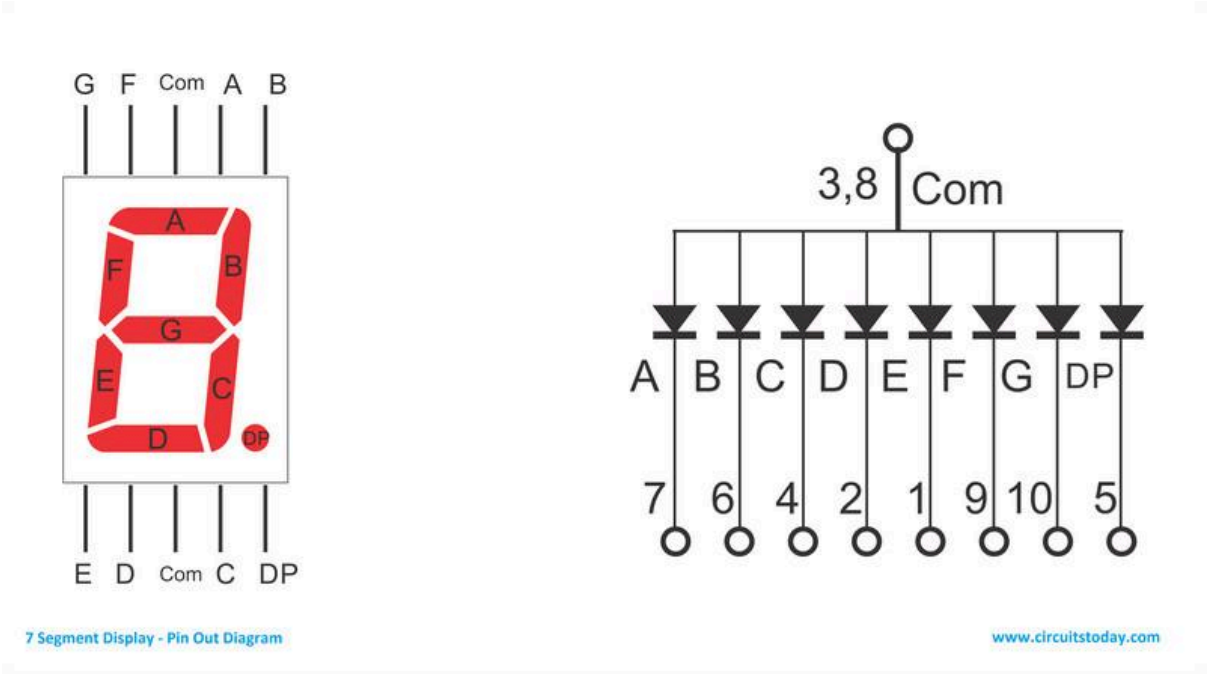
Now, when you open your Serial Monitor in the Arduino Software (IDE) (by clicking the icon that looks like a lens, on the right, in the green top bar or using the keyboard shortcut Ctrl+Shift+M), you should see a steady stream of numbers ranging from 0-1023, correlating to the position of the pot. As you turn your potentiometer, these numbers will respond almost instantly.

Seven segment display

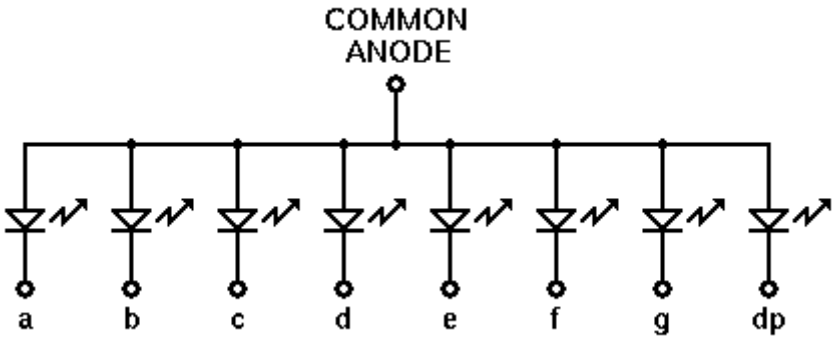
- **A seven segment display** is used to display numbers 0-9 and A to F.



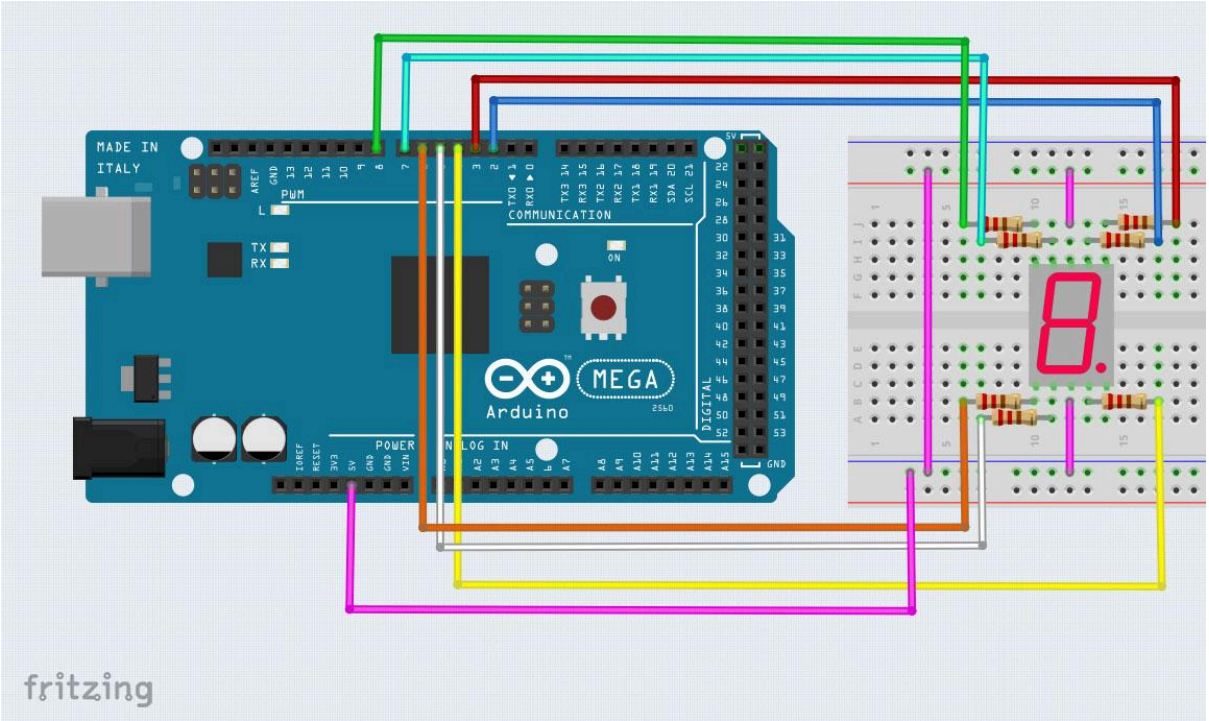
The internal pin connections of seven segment display is shown below.



There are two types: comman anode and comman cathode. Here we use a comman cathode type SSD.



Connections are made as shown below. Board used is arduino mega. Pin 3,8 are connected to VCC of the board as SSD is comman anode type. If it had been comman cathode type SSD pin 3,8 would be at ground.

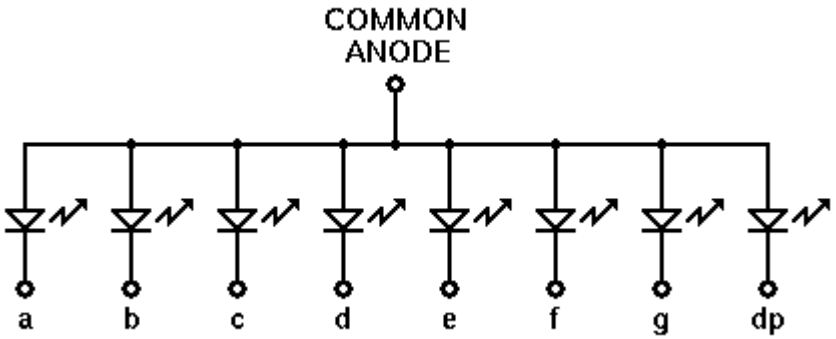



```
void setup()
{
 pinMode(2,OUTPUT);
 pinMode(3,OUTPUT);
 pinMode(4,OUTPUT);
 pinMode(5,OUTPUT);
 pinMode(6,OUTPUT);
 pinMode(7,OUTPUT);
 pinMode(8,OUTPUT);

}

void loop()
{
  // loop to turn leds od seven seg ON
```

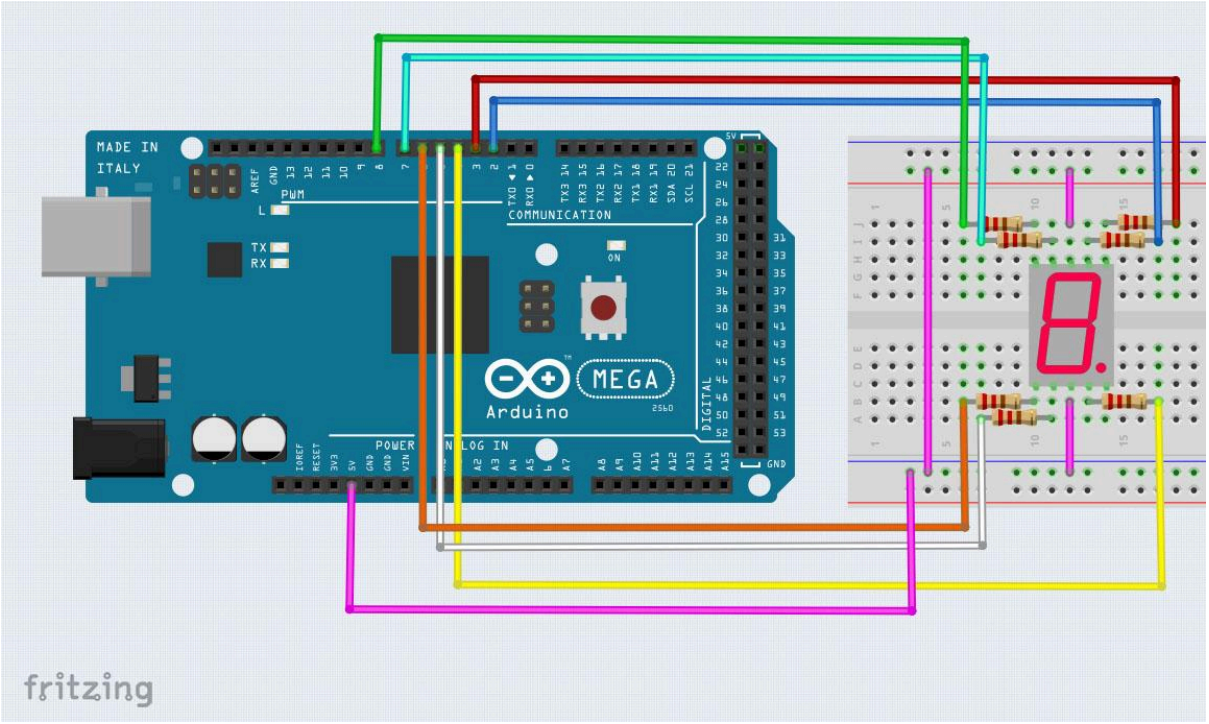
```
 for(int i=2;i<9;i++)
 {
   digitalWrite(i,HIGH);
   delay(600);
 }

 // loop to turn leds od seven seg OFF
 for(int i=2;i<9;i++)
 {
   digitalWrite(i,LOW);
   delay(600);
 }
   delay(1000);
}
```

Seven segment display part 2

- **We use a comman anode type SSD** and alternate turn it on and off.



Connections are made as shown below. Board used is arduino mega. Pin 3,8 are connected to VCC of the board as SSD is comman anode type. If it had been comman cathode type SSD pin 3,8 would be at ground.



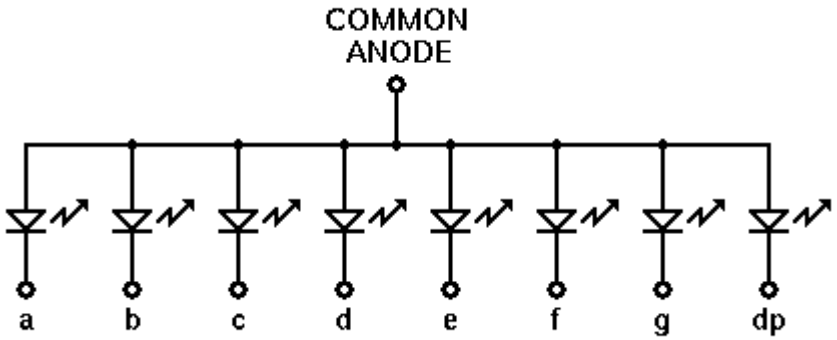

```
void setup()
{
 // define pin modes

 pinMode(0,OUTPUT);
```

```
 pinMode(1,OUTPUT);
 pinMode(2,OUTPUT);
 pinMode(3,OUTPUT);
 pinMode(4,OUTPUT);
 pinMode(5,OUTPUT);
 pinMode(6,OUTPUT);

}

void loop()
{
high();
delay(1000);
low();
delay(1000);
}
void high()
{
  for(int i=0;i<7;i++)
  {
    digitalWrite(i,HIGH);
  }
}
void low()
{
  for(int i=0;i<7;i++)
  {
    digitalWrite(i,LOW);
  }
```

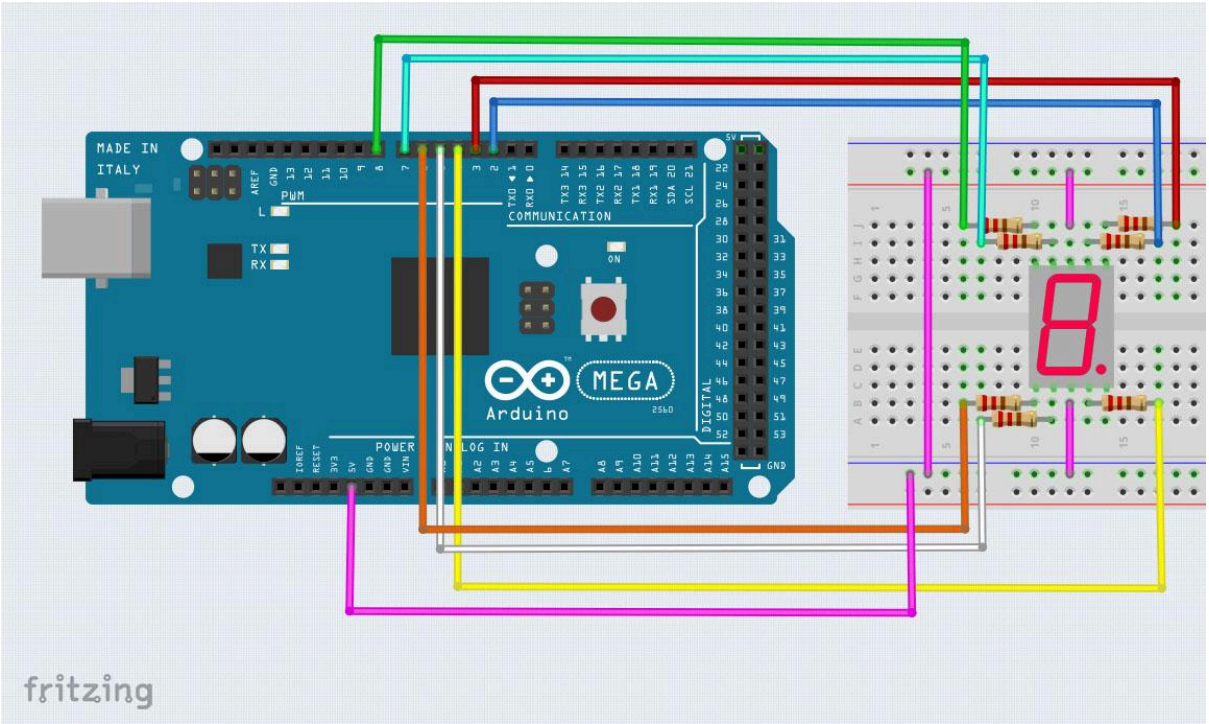
}

Seven segment display part 4

- **We use a comman anode type Seven Segment Display** and use it display number ZERO.



Connections are made as shown below. Board used is arduino mega. Pin 3,8 are connected to VCC of the board.




```
int seg_a = 2; // declare the variables

int seg_b = 3;
```

```
int seg_c = 4;
int seg_d = 5;
int seg_e = 6;
int seg_f = 7;
int seg_g = 8;
int seg_dp = 9;

void setup() {
  pinMode(seg_a,OUTPUT); // configure all pins used to outputs
  pinMode(seg_b,OUTPUT);
  pinMode(seg_c,OUTPUT);
  pinMode(seg_d,OUTPUT);
  pinMode(seg_e,OUTPUT);
  pinMode(seg_f,OUTPUT);
  pinMode(seg_g,OUTPUT);
  pinMode(seg_dp,OUTPUT);
}

void loop() {

  digitalWrite(seg_a,LOW);
  digitalWrite(seg_b,LOW);
  digitalWrite(seg_c,LOW);
  digitalWrite(seg_d,LOW);
  digitalWrite(seg_e,LOW);
  digitalWrite(seg_f,LOW);
  digitalWrite(seg_g,HIGH);
  digitalWrite(seg_dp,HIGH);
```

}

ULTRA SONIC SENSOR

Connections

The connections are very simple:

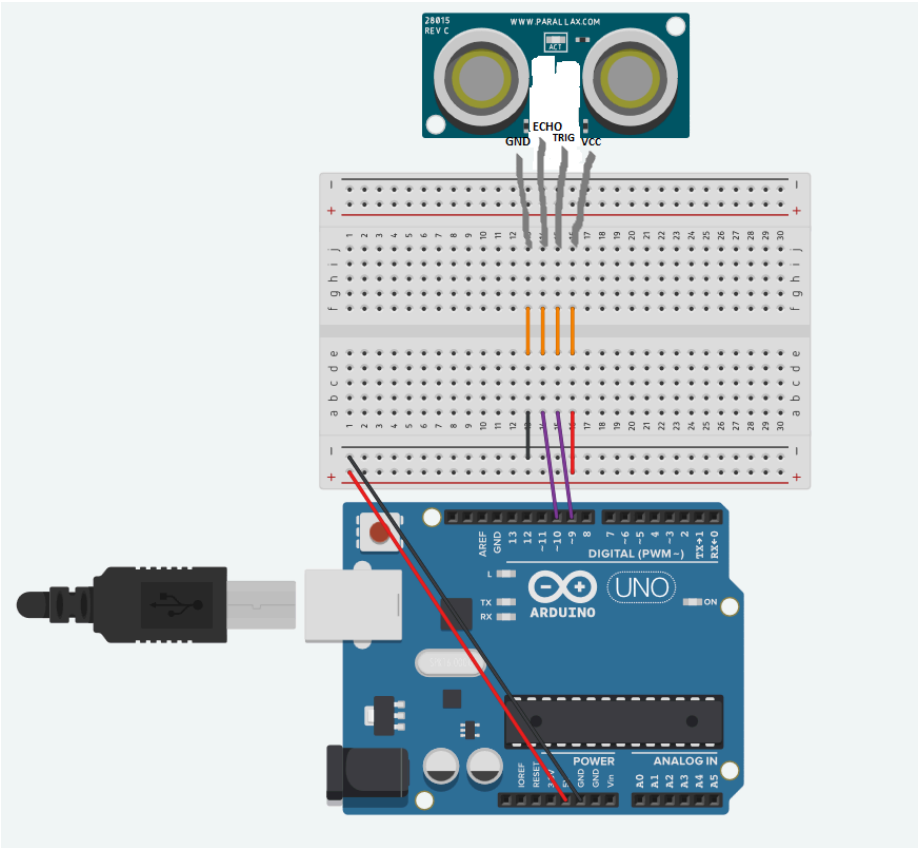
VCC to 5V

GND to GND

Trig to pin 9

Echo to pin 10

You can actually connect Trig and Echo to whichever pins you want, 9 and 10 are just the ones I'm using.



Diagram

Code

First we define the pins that Trig and Echo are connected to.

```
const int trigPin = 9;
const int echoPin = 10;
```

Then we declare 2 floats, duration and distance, which will hold the length of the sound wave and how far away the object is.

```
float duration, distance;
```

Next, in the setup, we declare the Trig pin as an output, the Echo pin as an input, and start Serial communications.

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

Now, in the loop, what we do is first set the trigPin low for 2 microseconds just to make sure that the pin in low first. Then, we set it high for 10 microseconds, which sends out an 8 cycle sonic burst from the transmitter, which then bounces of an object and hits the receiver(Which is connected to the Echo Pin).

```
void loop() {
```

```
        digitalWrite(trigPin, LOW);
        delayMicroseconds(2);
        digitalWrite(trigPin, HIGH);
        delayMicroseconds(10);
        digitalWrite(trigPin, LOW);
```

When the sound waves hit the receiver, it turns the Echo pin high for however long the waves were traveling for. To get that, we can use a handy Arduino function called pulseIn() . It takes 2 arguments, the pin you are listening to(In our case, the Echo pin), and a state(HIGH or LOW). What the function does is waits for the pin to go whichever state you put in, starts timing, and then stops timing when it switches to the other state. In our case we would put HIGH since we want to start timing when the Echo pin goes high. We will store the time in the duration variable. (It returns the time in microseconds)

```
duration = pulseIn(echoPin, HIGH);
```

Now that we have the time, we can use the equation speed = distance/time , but we will make it time x speed = distance because we have the speed. What speed do we have? The speed of sound, of course! The speed of sound is approximately 340 meters per second, but since the pulseIn() function returns the time in microseconds, we will need to have a speed in microseconds also, which is easy to get. A quick Google search for "speed of sound in centimeters per microsecond" will say that it is .0343 c/µS. You could do the math, but searching it is easier. Anyway, with that information, we can calculate the distance! Just multiply the duration by .0343 and then divide it by 2(Because the sound waves travel to the object AND back). We will store that in the distance variable.

```
distance = (duration*.0343)/2;
```

The rest is just printing out the results to the Serial Monitor.

```
        Serial.print("Distance: ");

        Serial.println(distance);

        delay(100); }
```

Pulse Width Modulation (PWM) is a powerful technique used in an array of applications, ranging from dimming an LED to controlling the speed of motors. In Arduino, PWM enables the simulation of analog output using digital output pins. By switching the pins on and off at a high frequency and varying the duration the pin stays high (the duty cycle), you can effectively control the power delivered to a device.

Basic Concepts of PWM in Arduino

PWM signals have two main components:

1. **Frequency:** The rate at which the PWM signal completes one cycle of on-and-off.

2. **Duty cycle:** The percentage of one cycle in which the signal is high (on). A duty cycle can range from 0% (always off) to 100% (always on).

In Arduino Uno, for example, PWM is available on pins 3, 5, 6, 9, 10, and 11. The frequency for pins 5 and 6 is about 980 Hz, and for the rest, it is ~490 Hz.

Example 1: Dimming an LED

This program will gradually increase and decrease the brightness of an LED using PWM.

Hardware Needed:

- Arduino Board (like Arduino Uno)

- 1 LED

- 220 Ohm resistor

- Breadboard

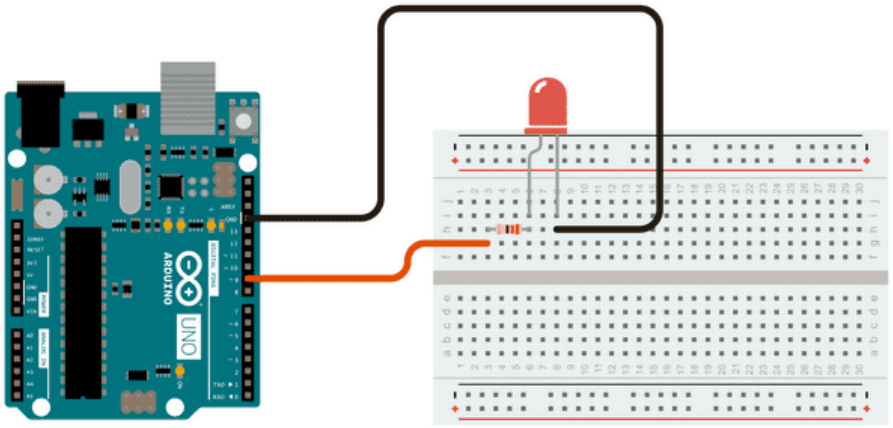
- Jumper wires

Circuit:

- Connect the anode (longer leg) of the LED to a PWM-capable pin on the Arduino (e.g., pin 9).

- Connect the cathode (shorter leg) of the LED to one end of the resistor.

- Connect the other end of the resistor to the GND pin on Arduino.



Code:

```cpp
int ledPin = 9; // PWM pin connected to the LED
int brightness = 0;
int fadeAmount = 5;

void setup() {
  pinMode(ledPin, OUTPUT); // Set the PWM pin as OUTPUT
}

void loop() {
  analogWrite(ledPin, brightness); // Set the brightness of LED
```

```
  brightness = brightness + fadeAmount; // Change the brightness for next time through the loop


  if (brightness <= 0 || brightness >= 255) {
    fadeAmount = -fadeAmount; // Reverse the direction of the fading at the ends of the fade
  }


  delay(30); // Wait for 30 milliseconds to see the dimming effect
}
```

In the code above, `analogWrite()` function is used to output a PWM signal. The brightness is increased gradually until it reaches the upper limit, then decreased by reversing the `fadeAmount`. This creates a fading effect.

RGB LED with an Arduino board

Using an RGB LED with an Arduino board is a common way to get started with electronic components and programming. An RGB LED consists of three individual LEDs, one for each primary color (Red, Green, Blue), all contained within a single package. By adjusting the brightness of each primary color, you can mix them to create a wide range of colors.

Below is a basic example of how to program an RGB LED using an Arduino. This example assumes you have an RGB LED with a common cathode (the long pin of the LED is connected to GND), and you're connecting the other pins (Red, Green, Blue) to digital PWM (Pulse Width Modulation) capable pins on the Arduino for brightness control.

Components Needed

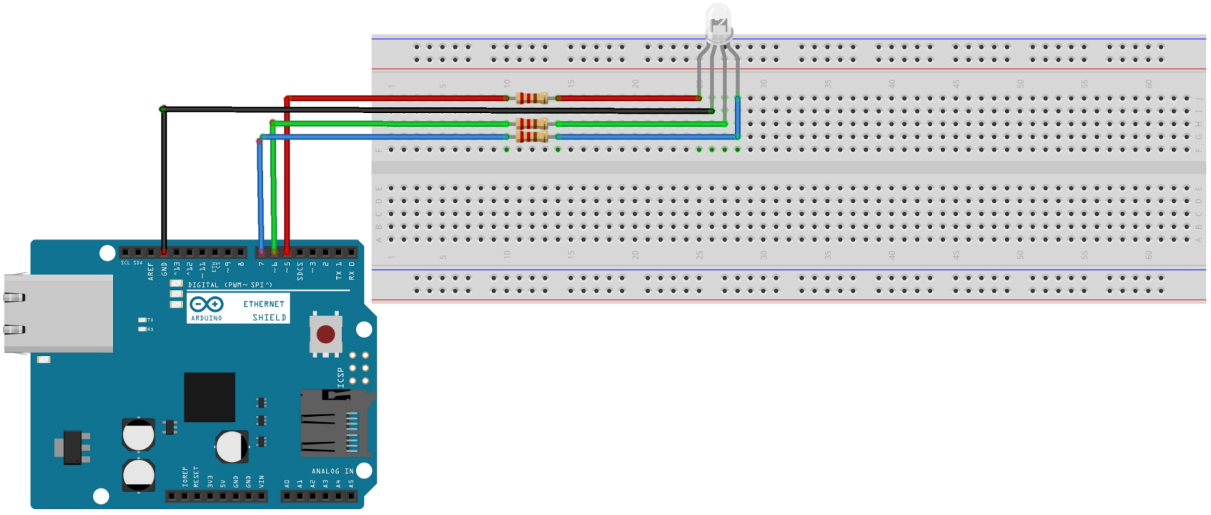
- 1 RGB LED (Common Cathode)

- 3 220Ω Resistor (To limit current to each LED pin)

- Arduino board (Uno, Nano, Mega, or any other)

- Breadboard and jumper wires

Circuit Setup



1. Connect the long pin (common cathode) of the RGB LED to the GND pin on the Arduino.

2. Connect the Red pin of the RGB LED to a PWM-capable digital pin (e.g., pin 11) through a 220

Ω resistor.

3. Repeat the process for the Green and Blue pins of the RGB LED, connecting them to different PWM-capable pins (e.g., pins 10 and 9) through 220Ω resistors.

Arduino Code

```cpp
// Define the RGB LED pins
```

```
int redPin = 11;
int greenPin = 10;
int bluePin = 9;

void setup() {
  // Set the RGB LED pins as output
  pinMode(redPin, OUTPUT);
  pinMode(greenPin, OUTPUT);
  pinMode(bluePin, OUTPUT);
}

void loop() {
  // Set the RGB LED to red
  setColor(255, 0, 0);
  delay(1000); // Wait for 1 second

  // Set the RGB LED to green
  setColor(0, 255, 0);
  delay(1000); // Wait for 1 second

  // Set the RGB LED to blue
  setColor(0, 0, 255);
  delay(1000); // Wait for 1 second

  // Set the RGB LED to white (all colors at full brightness)
  setColor(255, 255, 255);
  delay(1000); // Wait for 1 second

  // Set the RGB LED to off
```

```
  setColor(0, 0, 0);
  delay(1000); // Wait for 1 second
}

void setColor(int red, int green, int blue) {
  // Adjust for common cathode LED
  analogWrite(redPin, red);
  analogWrite(greenPin, green);
  analogWrite(bluePin, blue);
}
```

This simple program cycles the RGB LED through red, green, blue, and white colors, pausing for a second on each color. The `setColor` function takes three parameters (red, green, blue) that range from 0 (off) to 255 (maximum brightness) to set the LED's color.

Remember to adjust the `redPin`, `greenPin`, and `bluePin` variables if you connect your LED to different pins.

Always ensure you're using resistors appropriate for your setup to prevent damaging your LED. The 220Ω resistor value mentioned is a common choice for basic projects, but depending on your specific LED and power source, you might need to calculate the best resistor value for your needs.

Program: Liquid crystal display

The LCD Pinout It has 16 pins and the first one from left to right is the Ground pin. The second pin is the VCC which we connect the 5 volts pin on the Arduino Board.

Next is the Vo pin on which we can attach a potentiometer for controlling the contrast of the display.

Next, The RS pin or register select pin is used for selecting whether we will send commands or data to the LCD.

For example if the RS pin is set on low state or zero volts, then we are sending commands to the LCD like: set the cursor to a specific location, clear the display, turn off the display and so on.

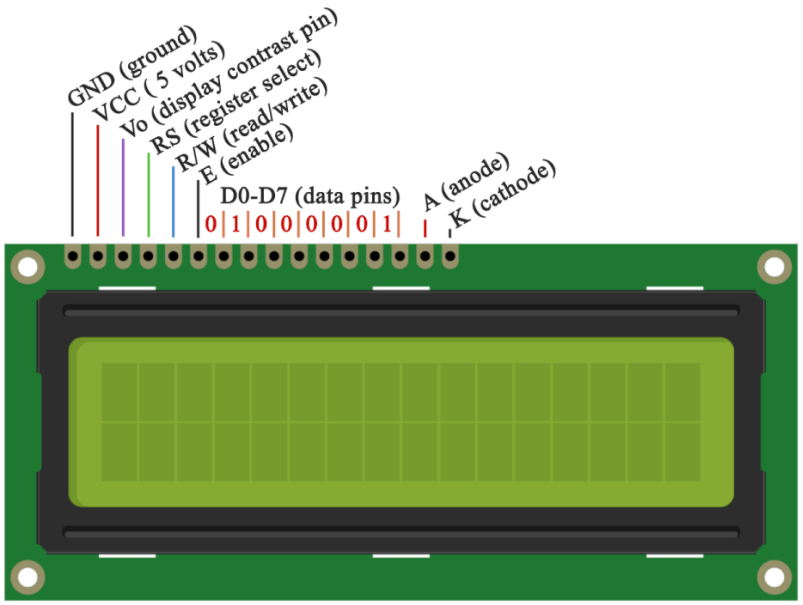
And when RS pin is set on High state or 5 volts we are sending data or characters to the LCD.

LCD Pin-Out Next comes the R / W pin which selects the mode whether we will read or write to the LCD. Here the write mode is obvious and it is used for writing or sending commands and data to the LCD. The read mode is used by the LCD itself when executing the program.

Next is the E pin which enables the writing to the registers, or the next 8 data pins from D0 to D7. So through this pins we are sending the 8 bits data when we are writing to the registers or for example if we want to see the latter uppercase A on the display we will send 0100 0001 i.e. 65 to the registers according to the ASCII table (char 'A' is 65 on ascii scale).

And the last two pins A and K, or anode and cathode are for the LED back light.

After all we don’t have to worry much about how the LCD works, as the Liquid Crystal Library takes care for almost everything. From the Arduino’s official website you can find and see the functions of the library which enable easy use of the LCD. We can use the Library in 4 or 8 bit mode.



Program: LCD to print "Hello World"

LCD Pin 1 is ground, LCD Pin 2 is for VCC.

LCD Pin 3 is V0 is to ground.

LCD RS pin to digital pin 12

LCD RW pin is to ground.

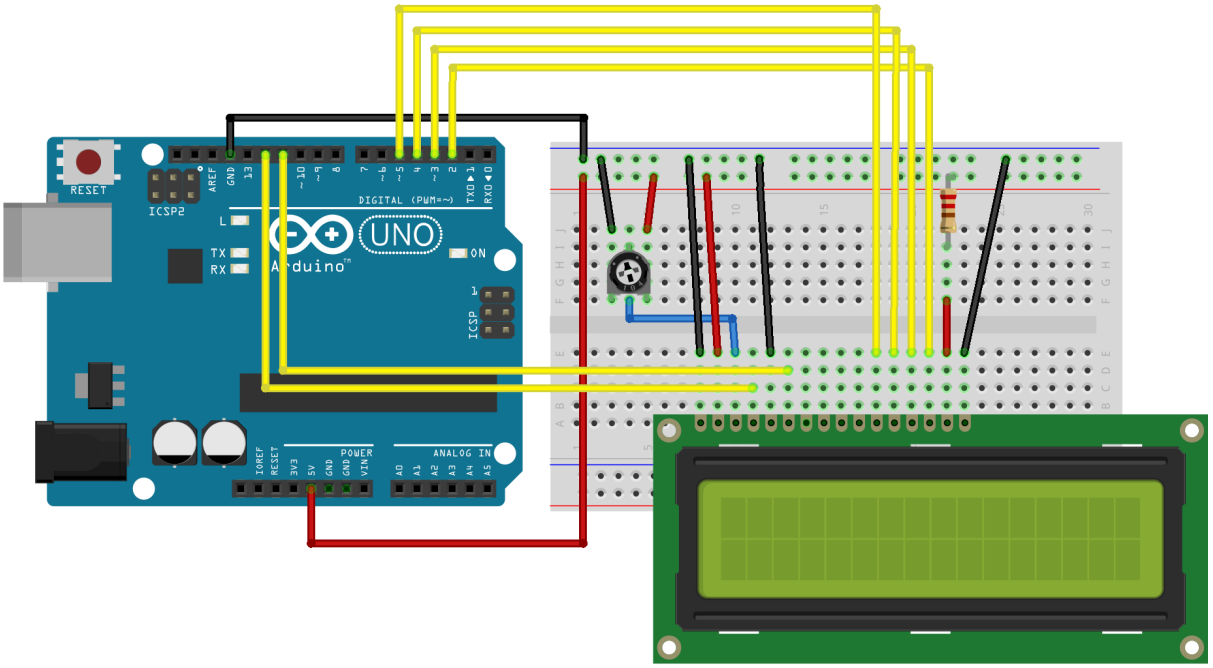
LCD Enable pin to digital pin 11

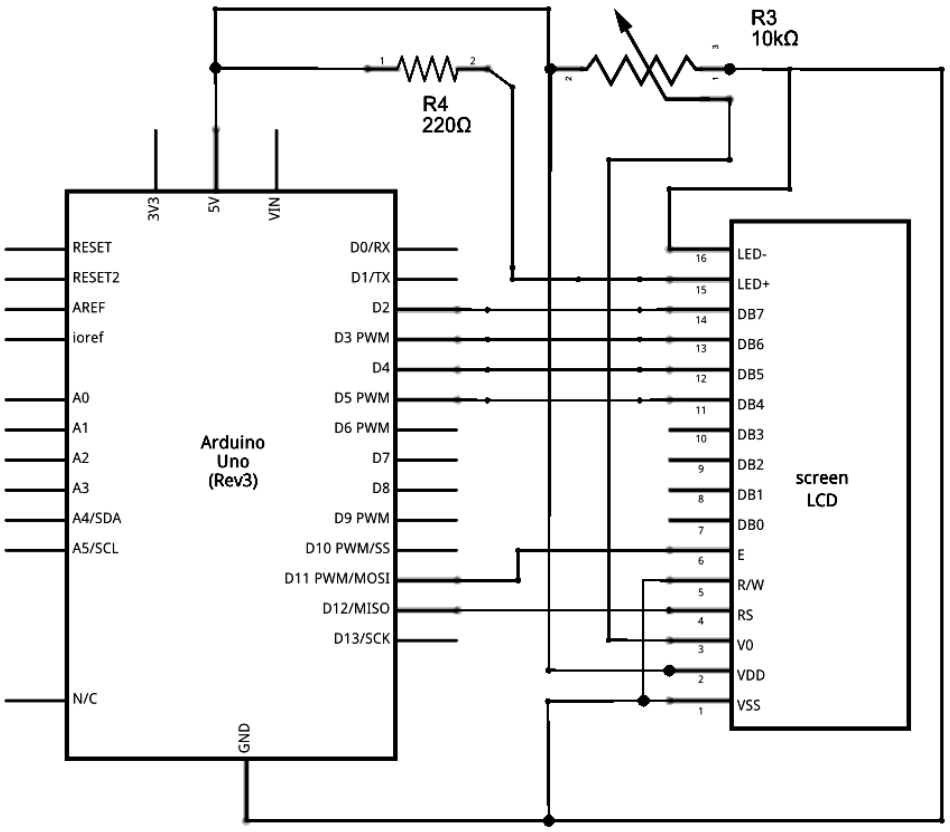
LCD D4 pin to digital pin 5

LCD D5 pin to digital pin 4

LCD D6 pin to digital pin 3

LCD D7 pin to digital pin 2





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

// initialize the library with the numbers of the interface pins
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);

void setup() {
  // set up the LCD's number of columns and rows:
  lcd.begin(16, 2);
  // Print a message to the LCD.
  lcd.print("hello, world!");
}

void loop() {
  // set the cursor to column 0, line 1
  // (note: line 1 is the second row, since counting begins with 0):
  lcd.setCursor(0, 1);
  // print the number of seconds since reset:
  lcd.print(millis() / 1000);
}
```

Copy

Program: LCD to blink

LCD Pin 1 is ground, LCD Pin 2 is for VCC.

LCD Pin 3 is V0 is to ground.

LCD RS pin to digital pin 12

LCD RW pin is to ground.

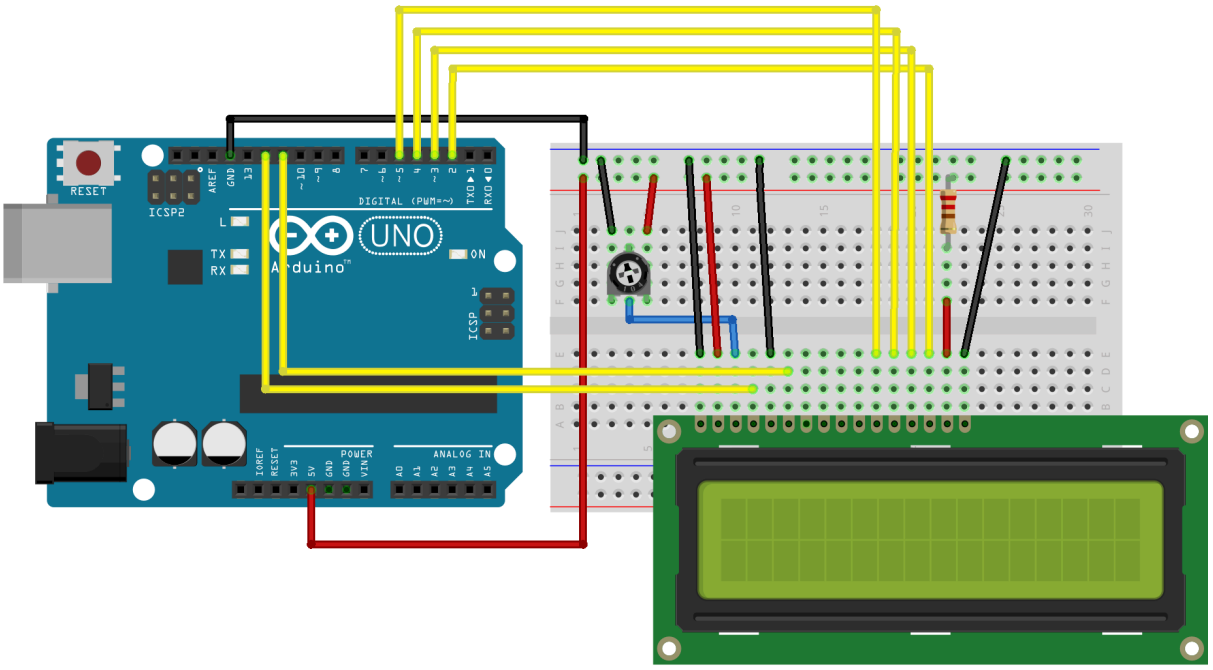
LCD Enable pin to digital pin 11

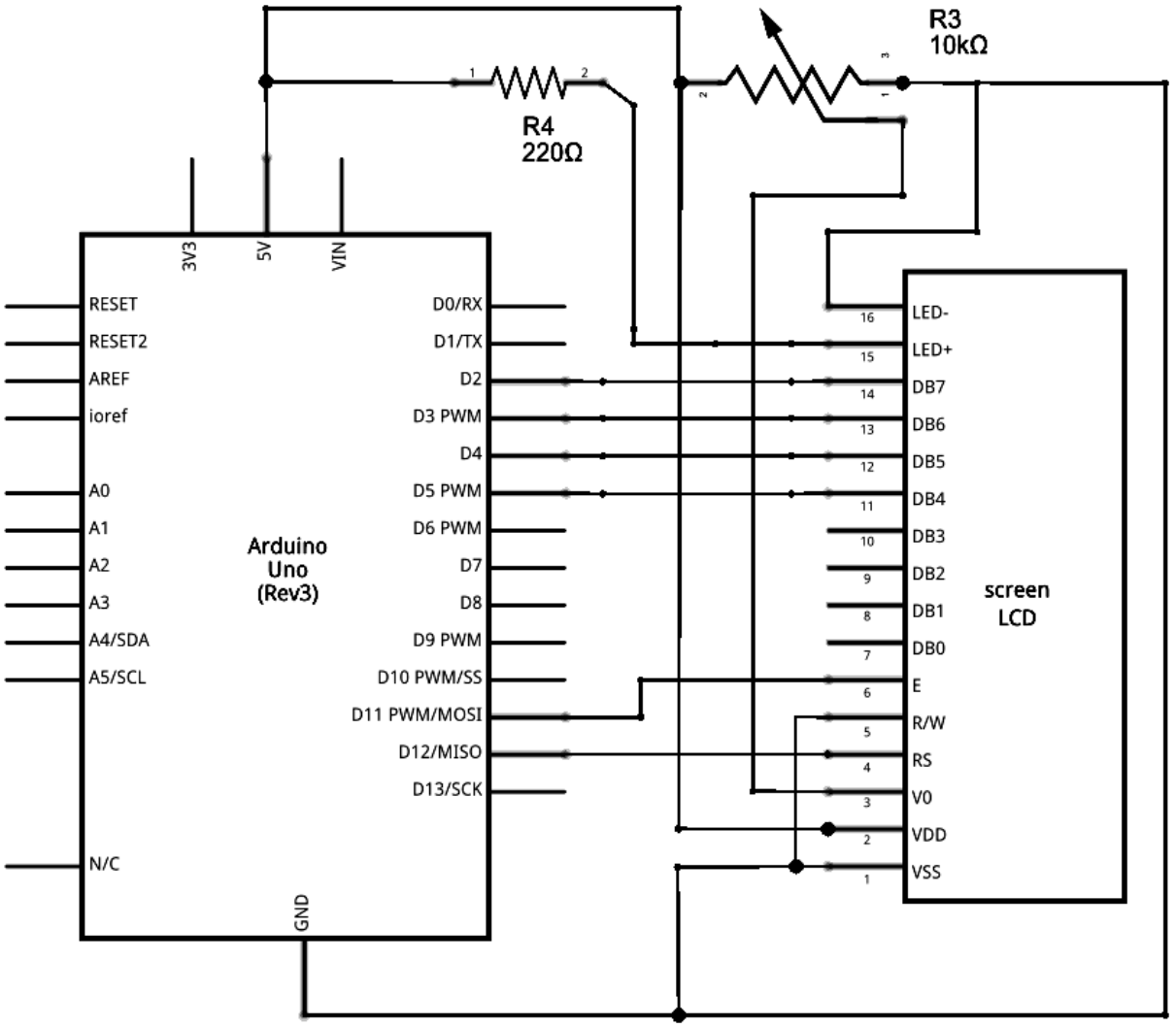
LCD D4 pin to digital pin 5

LCD D5 pin to digital pin 4

LCD D6 pin to digital pin 3

LCD D7 pin to digital pin 2


RESET
ICSP2
UNO
TX
RX
ON
ICSP
DIGITAL (PWM=~)
POWER
ANALOG IN


R3
10kΩ
R4
220Ω
3V3
5V
VIN
RESET
RESET2
AREF
ioref
A0
A1
A2
A3
A4/SDA
A5/SCL
N/C
GND
D0/RX
D1/TX
D2
D3 PWM
D4
D5 PWM
D6 PWM
D7
D8
D9 PWM
D10 PWM/SS
D11 PWM/MOSI
D12/MISO
D13/SCK
Arduino
Uno
(Rev3)
LED-
LED+
DB7
DB6
DB5
DB4
DB3
DB2
DB1
DB0
E
R/W
RS
V0
VDD
VSS
screen
LCD

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

const int rs = 12, en = 11, d4 = 5, d5 = 4, d6 = 3, d7 = 2;
LiquidCrystal lcd(rs, en, d4, d5, d6, d7);

void setup() {
  // set up the LCD's number of columns and rows:
  lcd.begin(16, 2);
  // Print a message to the LCD.
  lcd.print("hello, world!");
}

void loop() {
  // Turn off the blinking cursor:
  lcd.noBlink();
  delay(3000);
  // Turn on the blinking cursor:
  lcd.blink();
  delay(3000);
}
```

Copy

Program: LCD to set cursor on and off

LCD Pin 1 is ground, LCD Pin 2 is for VCC.

LCD Pin 3 is V0 is to ground.

LCD RS pin to digital pin 12

LCD RW pin is to ground.

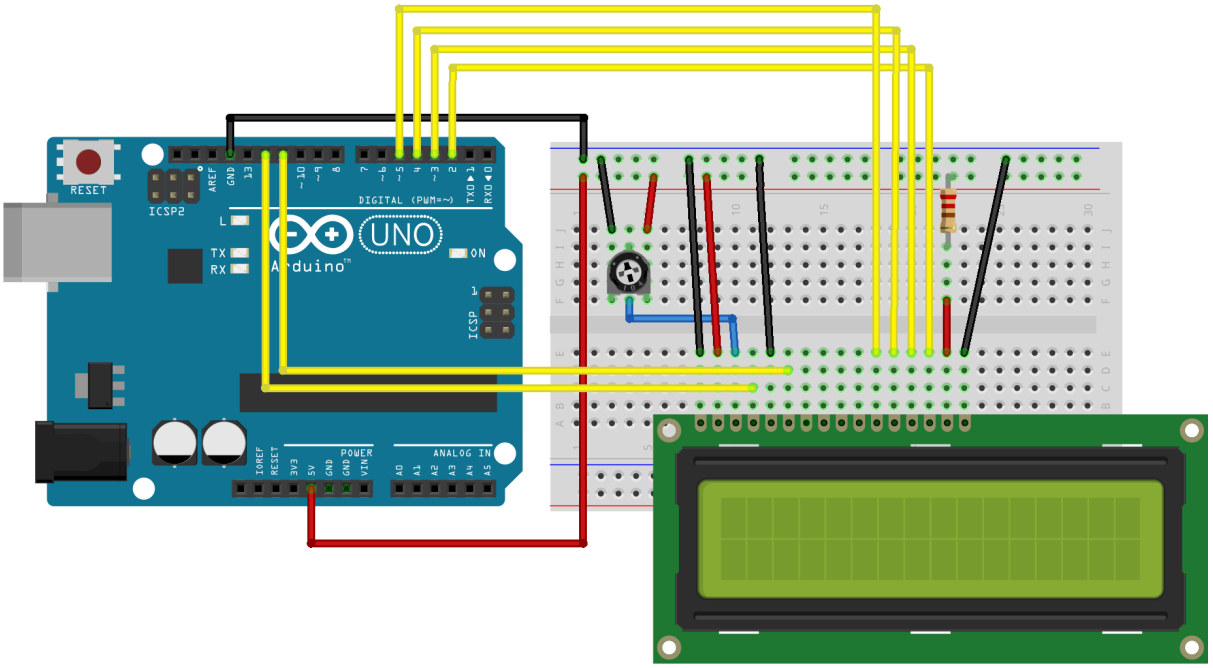
LCD Enable pin to digital pin 11

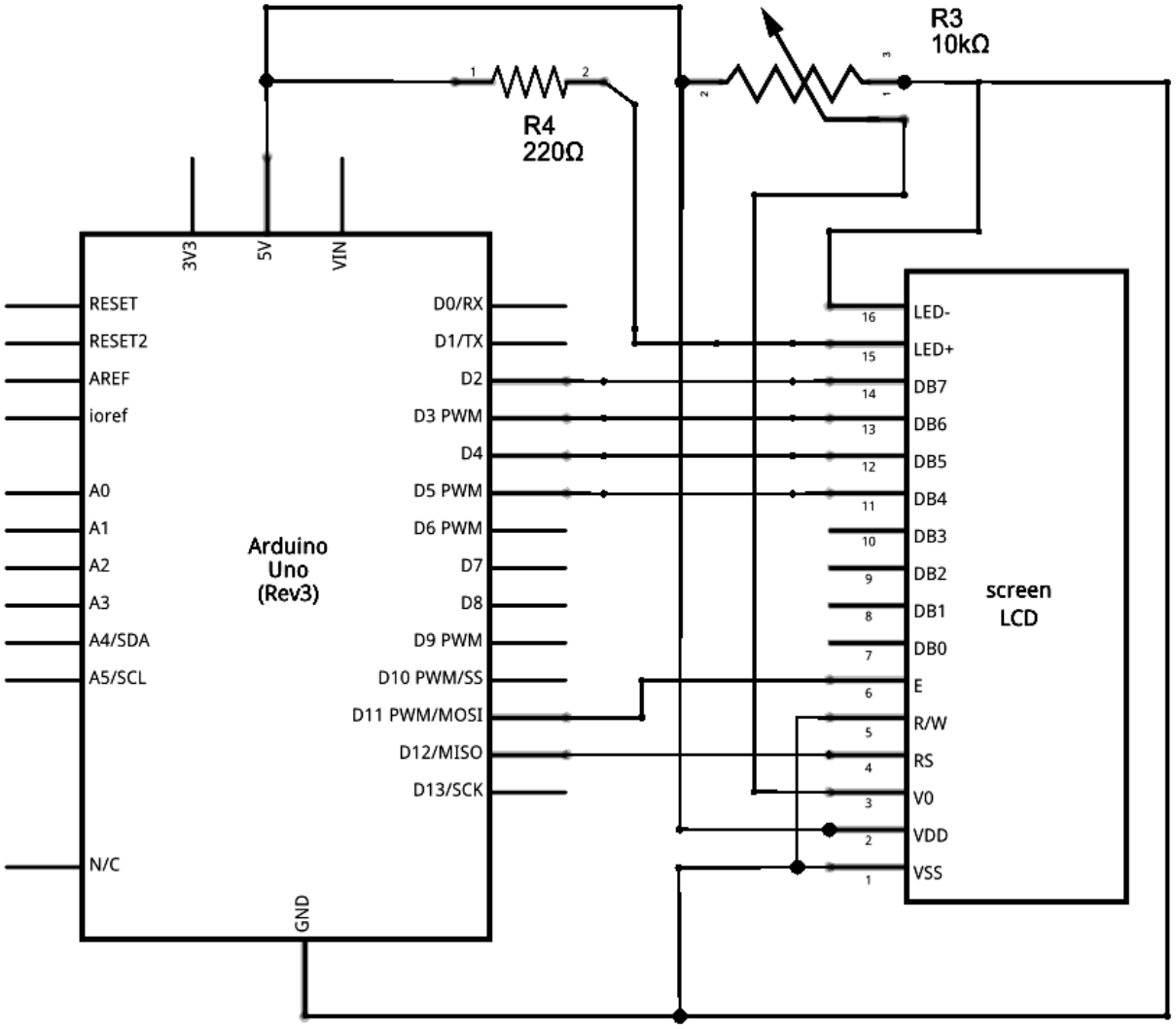
LCD D4 pin to digital pin 5

LCD D5 pin to digital pin 4

LCD D6 pin to digital pin 3

LCD D7 pin to digital pin 2


RESET
ICSP2
UNO
TX
RX
ON
ICSP
DIGITAL (PWM=~)
POWER
ANALOG IN

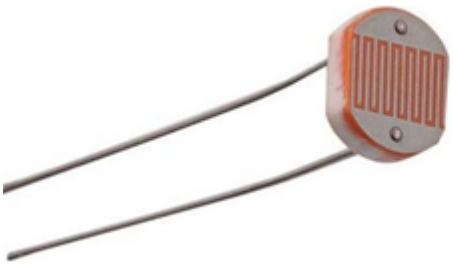

R3
10kΩ
R4
220Ω
3V3
5V
VIN
RESET
RESET2
AREF
ioref
A0
A1
A2
A3
A4/SDA
A5/SCL
N/C
GND
D0/RX
D1/TX
D2
D3 PWM
D4
D5 PWM
D6 PWM
D7
D8
D9 PWM
D10 PWM/SS
D11 PWM/MOSI
D12/MISO
D13/SCK
Arduino
Uno
(Rev3)
LED-
LED+
DB7
DB6
DB5
DB4
DB3
DB2
DB1
DB0
E
R/W
RS
V0
VDD
VSS
screen
LCD

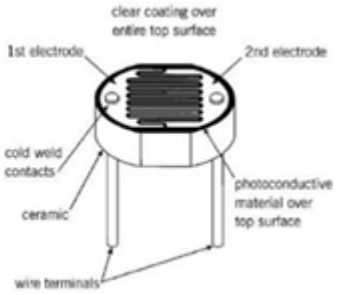
```
// include the library code:
#include <LiquidCrystal.h>

const int rs = 12, en = 11, d4 = 5, d5 = 4, d6 = 3, d7 = 2;
LiquidCrystal lcd(rs, en, d4, d5, d6, d7);

void setup() {
  // set up the LCD's number of columns and rows:
  lcd.begin(16, 2);
  // Print a message to the LCD.
  lcd.print("hello, world!");
}

void loop() {
  // Turn off the blinking cursor:
  lcd.noBlink();
  delay(3000);
  // Turn on the blinking cursor:
  lcd.blink();
  delay(3000);
}
```

UTKARSH MINDS
EMPOWER. EDUCATE. INNOVATE.

Photo Resistor or LDR

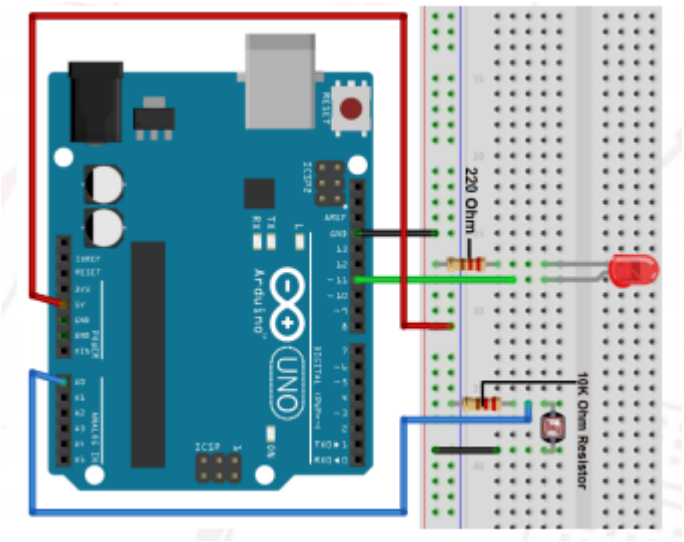
In this Project, we will work on Photo resistor (Photovaristor) which includes the photoelectric effect of semiconductor. Photovaristor is commonly applied in the measurement of light and light control.





Hardware Required Arduino Uno with USB Cable - 1 Breadboard 830 points – 1 Photo resistor or LDR – 1 LED - 1 10KΩ resistor – 1 220Ω resistor – 1 Jumper wire (Male to Male) – 40 pcs

Circuit Connection Attach LED and LDR on the Breadboard. Connect GND of Arduino Uno with Breadboard to make further GND connections. Connect 5V Pin of Arduino Uno with Breadboard for further 5V power supply connections. Connect positive terminal of LED with the Digital Pin 11 of the Arduino Uno. Connect negative terminal of LED with the GND rail of Breadboard via 220Ω resistor. Connect the one leg of LDR with the 5V power supply rail of Breadboard via 10KΩ resistor on one end and connect other end with the Analog Pin A0 of Arduino Uno. Connect other leg of the LDR with the GND rail of Breadboard



The photoelectric effect of semiconductor. Its basic working principle is that if the incident light is intense, its resistance reduces; if the incident light is weak, the resistance increases. Photovaristor is commonly applied in the measurement of light, light control and photovoltaic conversion (convert the change of light into the change of electricity).

Photo resistor is also being widely applied to various light control circuit, such as light control and adjustment, optical switches etc. Photovaristor is an element that changes its resistance as light strength changes.

So we will need to read the analog values. We can refer to the PWM experiment, replacing the potentiometer with photovaristor. When there is change in light strength, there will be corresponding change on the LED.

After the connection, let's begin the program compilation process. The program is similar to the one of PWM. After downloading the program, you can change the light strength around the photovaristor and see corresponding brightness change of the LED.

int potpin=0;// initialize analog pin 0, connected with photovaristor

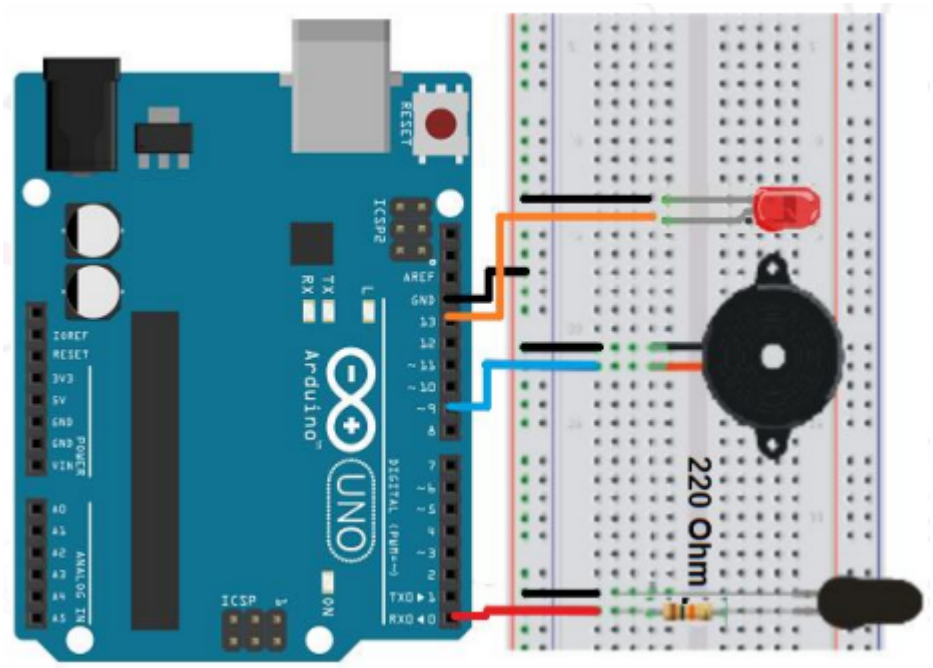
```
void setup() {
  pinMode(11, OUTPUT); // Set ledPin as an output
  Serial.begin(9600);       // Begin serial communication at 9600 bps
}


void loop() {
  ldrValue = analogRead(A0); // Read the value from the LDR
   Serial.print("LDR Value: ");    // Print the LDR value to the serial
monitor
  Serial.println(ldrValue);


  // Check if it's dark
  if (ldrValue < 500) {
    digitalWrite(11, HIGH); // Turn on the LED
  } else {
    digitalWrite(11, LOW);  // Turn off the LED
  }


  delay(1000); // Wait for 1 second before repeating the loop
}
```

Detect Flame using Flame Sensor

Flame sensor (Infrared receiving triode) is specially used on robots to find the fire source. This sensor is of high sensitivity to flame.

Hardware Required Arduino Uno with USB Cable – 1 B10 Buzzer – 1 Flame Sensor – 1 LED - 2 Breadboard 830 points – 1 Jumper Wire (Male to Male) – 40 pcs 220Ω resistor – 2

Circuit Connection Connect GND of Arduino Uno with Breadboard for further GND connections. Attach Buzzer, LED and flame sensor on the Breadboard. Connect the positive terminal of Buzzer with the Digital Pin 9 of the Arduino Uno. Connect negative terminal of LED, Flame sensor and Buzzer with the GND rail of the Breadboard. Connect positive terminal of Flame sensor with Digital Pin 0 of Arduino Uno via 220Ω resistor. Connect positive terminal of LED with Digital Pin 13 of Arduino Uno.



Flame sensor is made based on the principle that infrared ray is highly sensitive to flame. It has a specially designed infrared receiving tube to detect fire, and then convert the flame brightness to fluctuating level signal. The signals are then input into the central processor and be dealt with accordingly.

When it's approaching a fire, the voltage value the analog port reads differs. If you use a multi meter, you can know when there is no fire approaching, the voltage it reads is around 0.3V; when there is fire approaching, the voltage it reads is around 1.0V, the nearer the fire, the higher the voltage. So in the beginning of the program, you can initialize voltage value i (no fire value);

Then, continuously read the analog voltage value j and obtain difference value k=j-i; compare k with 0.6V (123 in binary) to determine whether or not there is a fire approaching; if yes, the buzzer will buzz.

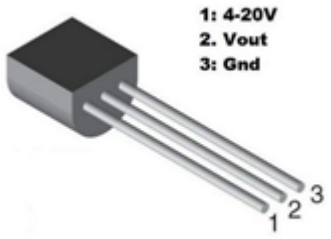
```
int flame=0;// select digital pin 0 for the sensor
int Beep=9;// select digital pin 9 for the buzzer
int val=0;// initialize variable
int led= 13; // Select Digital Pin 13 for the LED
void setup()
{
 pinMode(Beep,OUTPUT);// set buzzer pin as “output”
 pinMode(led,OUTPUT);// set LED pin as “output”
 pinMode(flame,INPUT);// set flame pin as “input”
 Serial.begin(9600);// set baud rate at “9600”
}
void loop()
{
 val=analogRead(flame);// read the analog value of the sensor
```

```
 Serial.println(val);// output and display the analog value
 if(val>=300)// when the analog value is larger than 300, the buzzer will buzz
 {
 digitalWrite(Beep,HIGH);
 digitalWrite(led,HIGH);
 }
else
 {
 digitalWrite(Beep,LOW);
 digitalWrite(led,LOW);
 }
 delay(500);
}
```

Copy

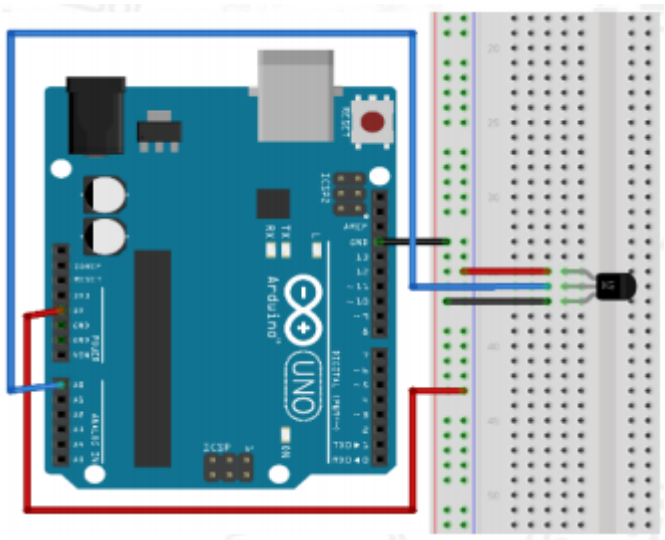
Basic LM35 Temperature Sensor

LM35 is a common and easy-to-use temperature sensor. It does not require other hardware. You just need an analog port to make it work but difficulty lies in compiling code to convert analog value it reads to Celsius temperature



Hardware Required Arduino Uno with USB Cable - 1 Breadboard 830 points – 1 Jumper wires (Male to Male) – 40 pcs LM35 Temperature Sensor Module – 1

Circuit Connection Attach the LM35 Temperature Sensor with the Breadboard. Connect the GND of Arduino Uno with Breadboard for making further GND connections. Connect Pin 5V of Arduino Uno with Breadboard for making further 5V power supply connections. Connect GND of LM35 Temperature Sensor with GND rail of Breadboard. Connect VCC of LM35 Temperature Sensor with 5V power supply rail of Breadboard. Connect Signal Pin of LM35 Temperature Sensor with Analog Pin A0 of Arduino Uno.



The LM35 series are precision integrated-circuit temperature devices with an output voltage linearly-proportional to the Centigrade temperature. The LM35 sensor does not require any external calibration or trimming to provide typical accuracies of ±¼°C at room temperature and ±¾°C over a full −55°C to 150°C temperature range. The LM35 is one kind of commonly used temperature sensor that can be used to measure temperature with an electrical o/p comparative to the temperature (in °C). It can measure temperature more correctly compare with a thermistor. This sensor generates a high output voltage than thermocouples and may not need that the output voltage is amplified. The LM35 has an output voltage that is proportional to the Celsius temperature. The scale factor is .01V/°C.

Lm35 voltage conversion to temperature formula/equation derivation for Arduino Arduino analog pins can measure up-to +5 volts OR the voltage on which it is working normally +5 volts. Arduino analog pin resolution is 1023 starting from 0. On +5 volts input it counts to 1023. Lm35 max voltage output is 1500mV(At 150 degree centigrade). 1500mV is equal to 1500/1000 = 1.5 volts. So Lm35 at max outputs 1.5 voltage. Arduino

analog pin count for 1.5 volts equals to (1.5 / 5)*1023 = 307.5 . At +5 volts its 1023 and at 1.5 volts its 307.5. New Arduino-Lm35 Resolution = 307.5 / 150 = 2.048 . Now if arduino analog pin counts 2.048 its equal to 1 degree change in centigrade/Celsius temperature of LM35.

```
int potPin = 0; // initialize analog pin 0 for LM35 temperature sensor
void setup()
{
Serial.begin(9600);// set baud rate at”9600”
}
void loop()
{
int val;// define variable
val=analogRead(potPin);
float temperatureC=val/2.048;
Serial.print("Temp:");// output and display characters beginning with Tep
Serial.print(temperatureC);// output and display value of dat
Serial.println("C");// display “C” characters
delay(500);// wait for 0.5 second
}
```