INTERNET OF THINGS

UNIT – 3

I. INTRODUCTION TO ARDUINO

Arduino is an open-source electronics platform based on easy-to-use hardware and software.
It consists of a microcontroller board and an Integrated Development Environment (IDE) used for writing, compiling, and uploading code to the board.

- This board includes digital I/O pins-14, a power jack, analog i/ps-6, ceramic resonator-A16 MHz, a USB connection, an RST button, and an ICSP header.

All these can support the microcontroller for further operation by connecting this board to the computer. The power supply of this board can be done with the help of an AC to DC adapter, a USB cable, otherwise a battery.

- The ATmega328 is one kind of single-chip microcontroller formed with Atmel within the mega AVR family. The architecture of this Arduino Uno is a customized Harvard architecture with 8 bit RISC processor core.

II. KEY FEATURES OF ARDUINO

Microcontroller: At the heart of every Arduino board is a microcontroller chip that serves as the brain of the board. This microcontroller can be programmed to perform various tasks, such as reading sensor data, controlling motors, and communicating with other devices.

Arduino IDE (Integrated Development Environment): Arduino provides a user-friendly software development environment called the Arduino IDE. It simplifies the process of writing, uploading, and managing code for Arduino boards. The IDE uses a C/C++-like programming language that is easy for beginners to learn.

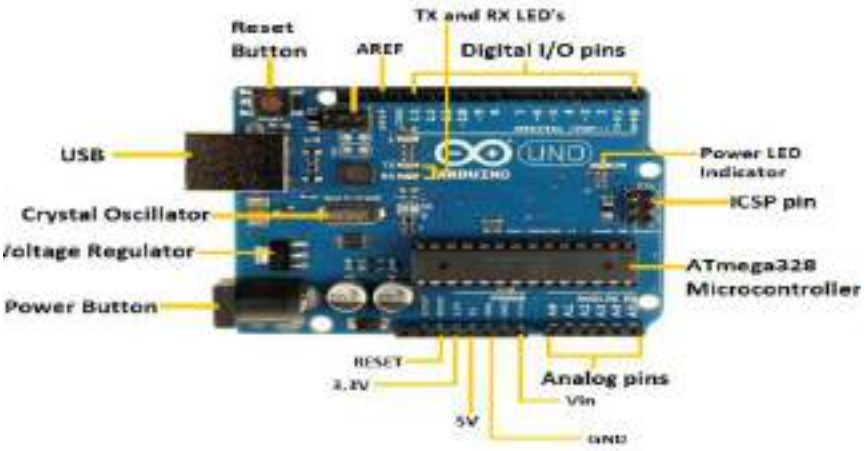
Open Source: Arduino is open-source, which means the hardware and software designs are freely available to the public. This open nature has led to a vibrant community of developers, makers, and educators who contribute to and expand upon the platform.

Variety of Boards: There are several types of Arduino boards available, each with its own set of features and capabilities. Some common Arduino boards include the Arduino Uno, Arduino Nano, and Arduino Mega. Each board is designed for specific use cases, ranging from simple projects to more complex ones.

Extensible: Arduino boards can be expanded and customized using shields or modules. Shields are add-on boards that provide additional functionalities like Wi-Fi, Bluetooth, GPS, and motor control. This extensibility allows you to tailor your Arduino project to your specific needs.

III. Arduino Hardware PIN Diagram

Arduino boards feature a variety of pins and headers that are crucial for interfacing with external components, sensors, and circuits.



1. Power USB

Arduino board can be powered by using the USB cable from your computer. All you need to do is connect the USB cable to the USB connection.

2. Power (Barrel Jack)

Arduino boards can be powered directly from the AC mains power supply by c connecting it to the Barrel Jack (2).

3. Voltage Regulator

The function of the voltage regulator is to control the voltage given to the Arduino board and stabilize the DC voltages used by the processor and other elements.

4. Crystal Oscillator

The crystal oscillator helps Arduino in dealing with time issues.
How does Arduino calculate time? The answer is, by using the crystal oscillator.
The number printed on top of the Arduino crystal is 16.000H9H. It tells us that the frequency is 16,000,000 Hertz or 16 MHz.

5. Arduino Reset

Reset your Arduino board, i.e., start your program from the beginning. reset the UNO board in two ways. First, by using the reset button (17) on the board. Second, you can connect an external reset button to the Arduino pin labelled RESET (5).

6. Pins (3.3, 5, GND, Vin)

3.3V (6): Supply 3.3 output volt
5V (7): Supply 5 output volt
Most of the components used with Arduino board works fine with 3.3 volt and 5 volts.
GND (8) (Ground): There are several GND pins on the Arduino, any of which can be used to ground your circuit.
Vin (9): This pin also can be used to power the Arduino board from an external power source, like AC mains power supply.

7. Analog pins

The Arduino UNO board has five analog input pins A0 through A5.
These pins can read the signal from an analog sensor like the humidity sensor or temperature sensor and convert it into a digital value that can be read by the microprocessor.

8. Main microcontroller

Each Arduino board has its own microcontroller (11). it is the brain of your board.
The main IC (integrated circuit) on the Arduino is slightly different from board to board.
The microcontrollers are usually of the ATMEL Company.

9. ICSP pin

ICSP (12) is an AVR, a tiny programming header for the Arduino consisting of MOSI, MISO, SCK, RESET, VCC, and GND.

10. Power LED indicator

This LED should light up when you plug your Arduino into a power source to iindicate that your board is powered up correctly. If this light does not turn on, then there is something wrong with the connection.

TX and RX LEDs

- First, at the digital pins 0 and 1, to indicate the pins responsible for serial communication. Second, the TX and RX led (13).
- The TX led flashes with different speed while sending the serial data.
- The speed of flashing depends on the baud rate used by the board. RX flashes during
- the receiving process.

Digital I / O

- The Arduino UNO board has 14 digital I/O pins (15) (of which 6 provide PWM (Pulse Width Modulation) output.
- These pins can be configured to work as input digital pins to read logic values (0 or 1) or as digital output pins to drive different modules like LEDs, relays, etc.
- The pins labelled "~" can be used to generate PWM.

AREF

- AREF stands for Analog Reference.
- It is sometimes, used to set an external reference voltage (between 0 and 5 Volts) as the upper limit for the analog input pins.

INTERNET OF THINGS

UNIT – 3

I. ARDUINO PROGRAMMING

- Arduino programming is built upon the C and C++ programming languages.
- It's tailored to the specific features and capabilities of Arduino microcontroller boards.
- It involves writing code to control the behaviour of an Arduino board.

BASIC CODE

An Arduino sketch (program) typically consists of two main functions: setup () and loop ().
setup (): It is executed once when the board starts or is reset. It's used for initializing variables, configuring pins, and setting up any hardware.
loop (): It is executed repeatedly after setup () completes.
It contains the main logic of your program and runs indefinitely until the board is turned off or reset**.**

II. Arduino Function Libraries

Arduino input/output functions

Digital i/o functions

- Used to Configure Digital I/O pins as Input or Output.
- Can be enabled / disabled. Logic Level can be HIGH or LOW
- Input values are readable
- Output values are writable
- To configure the digital pins as input or output and to read or write/load the digital value, 3 functions are used
- pinMode(), 2. digitalWrite(), 3.digitalRead()

pinMode()

Description

- Configures the specified pin to behave either as an input or an output.
  Syntax: pinMode (pin, mode)
- **pin:** Arduino Digital i/o pin number.
  mode: INPUT, OUTPUT, or INPUT_PULLUP
- **Returns:** Nothing

Example:

```
void setup() {
    pinMode(13, OUTPUT);        // sets the digital pin 13 as output
        }
```

digitalWrite()

Description: Write a HIGH or a LOW value to a digital pin.

If the pin has been configured as an OUTPUT with pinMode(), its voltage will be set to the corresponding value: 5V (or 3.3V on 3.3V boards) for HIGH, 0V (ground) for LOW.

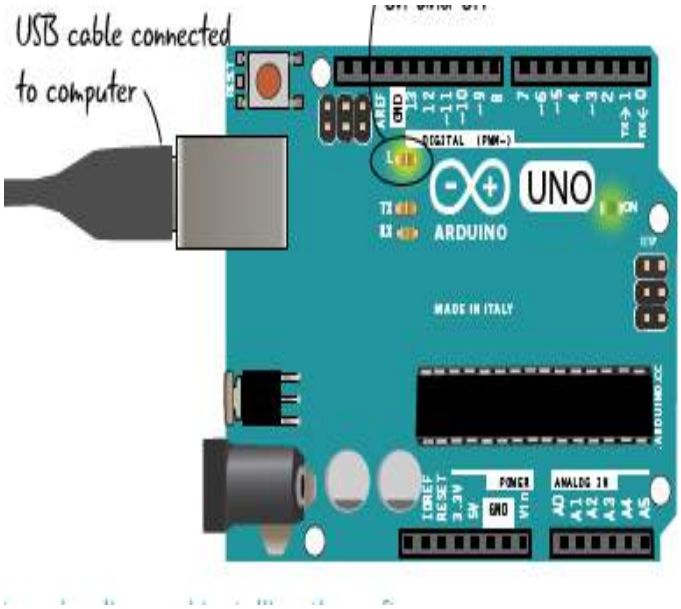
If the pin is configured as an INPUT, digitalWrite() will enable (HIGH) or disable (LOW) the internal pullup on the input pin.

Syntax: digitalWrite(pin,value)

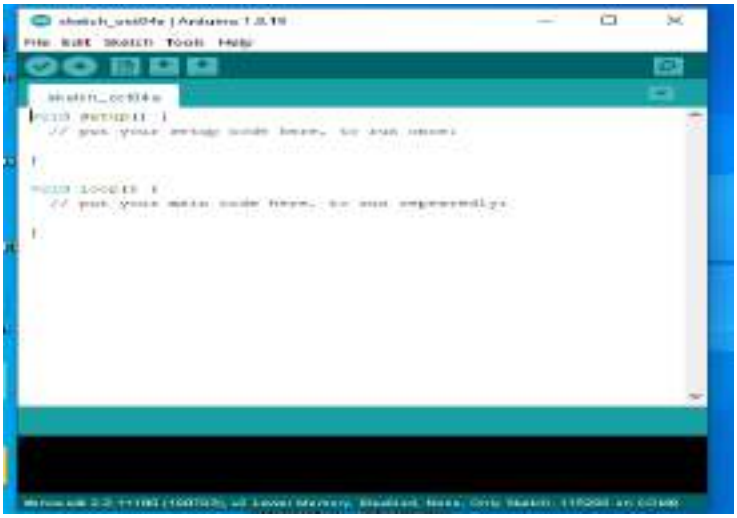
pin: the Arduino pin number.
value: HIGH or LOW.
Returns: Nothing

ARDUINO IDE FEATURES AND BLINK EXAMPLE

CONNECT ARDUINO BOARD TO COMPUTER

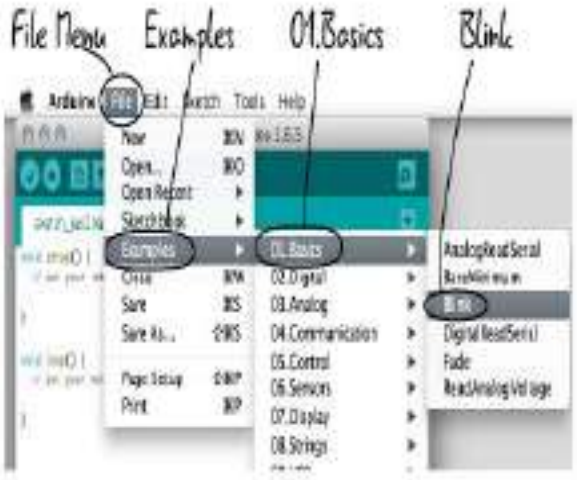


OPEN THE ARDUINO IDE



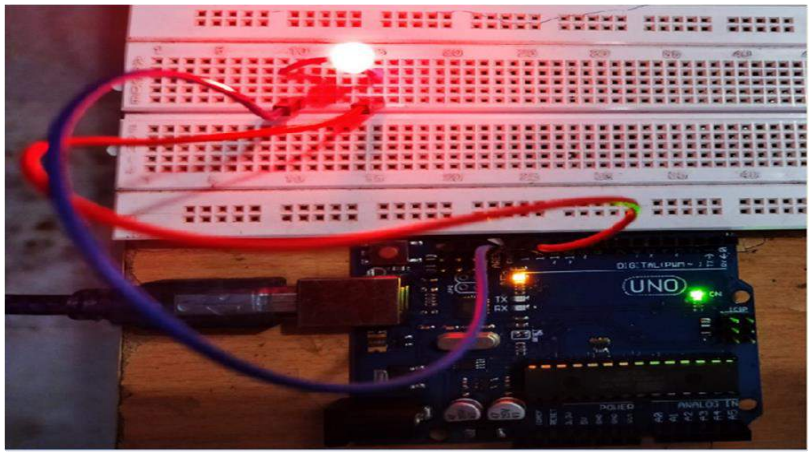
IDE TOOL SPECIFICATIONS

Writing and uploading your first sketch (Blinking LED)



Example Code://*The code makes the digital pin 13 OUTPUT and Toggles it HIGH and LOW*

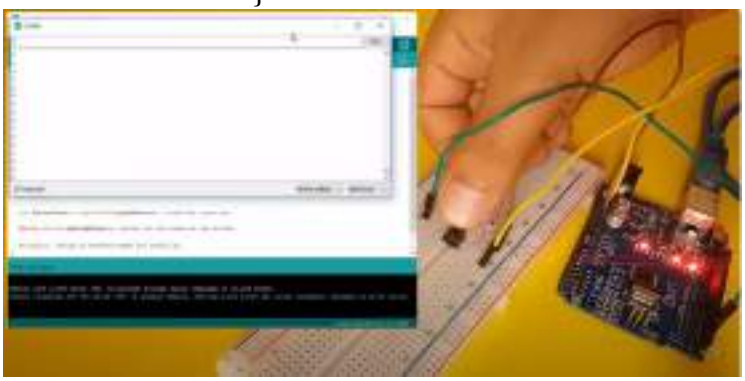
```
void setup() {
        pinMode(13, OUTPUT);  // sets the digital pin 13 as output
          }
void loop() {
            digitalWrite(13, HIGH); // sets the digital pin 13 on
            delay(1000);            // waits for a second
            digitalWrite(13, LOW);  // sets the digital pin 13 off
            delay(1000);            // waits for a second
           }
```



digitalRead()
Description
Reads the value from a specified digital pin, either HIGH or LOW.
Syntax
digitalRead(pin)
Parameters
pin: the Arduino pin number you want to read
Returns
HIGH or LOW

Sketch 2: Setting a Pin as an Input and Using digitalRead():

```
const int buttonPin = 2; // Define the pin number for the button
        void setup() {
                    pinMode(buttonPin, INPUT); // Set pin 2 as an input
                    Serial.begin(9600); // Initialize serial communication
                      }
        void loop() {
                    int buttonState = digitalRead(buttonPin); // Read the state of the button
                    Serial.println(buttonState);// Print the button state (HIGH or LOW) to the Serial
                                                                                          Monitor
                   delay(100); // Delay for stability
                    }
```



VARIABLES

Variables in Arduino programming are used to store and manipulate data.

1. Variable Declaration: To use a variable, you need to declare it.
his tells the Arduino IDE what type of data the variable will holdint myNumber; // Declares an integer variable named myNumber

2. Assigning Values: Once declared, you can assign a value to the variable.
myNumber = 10; // Assigns the value 10 to the variable myNumber
You can also declare and initialize a variable in a single step:
int myNumber = 10; // Declares and initializes the variable myNumber with the value 10

3. Variable Types: Arduino supports various data types for variables. Some common ones include:
int: Used for integers (whole numbers).
float: Used for floating-point numbers (numbers with decimal points).
char: Used for individual characters.
bool: Used for boolean values (true/false).

4. Using Variables: Once you've declared and assigned values to variables, you can use them in your code.

```
int a = 5;
int b = 3;
int sum = a + b; // Adds the values of variables a and b and stores the result in sum
```

EXAMPLE SKETCH

Write an Arduino sketch that demonstrates the use of variables to blink an LED connected to pin 13 at different intervals:

```
int ledPin = 13; // Define the pin number for the LED
int interval1 = 500; // Interval for the first blink (in milliseconds)
int interval2 = 1000; // Interval for the second blink (in milliseconds)
void setup() {
              pinMode(ledPin, OUTPUT); // Set the LED pin as an output
               }
void loop() {
               digitalWrite(ledPin, HIGH); // Turn on the LED
               delay(interval1); // Wait for interval1 milliseconds
               digitalWrite(ledPin, LOW); // Turn off the LED
               delay(interval2); // Wait for interval2 milliseconds
             }
```

INTERNET OF THINGS

UNIT – 3

I. Input / Output Functions – Analog

Analog Pins (Analog Input):

1. Analog pins are used to read analog signals, such as sensor readings or voltage levels.
2. They provide a range of values between 0 and 1023 based on the input voltage.
3. Labelled as "A" followed by a number (e.g., A0, A5).

Analog Pins (Analog Output):

1. Analog output pins on Arduino are typically the PWM (Pulse Width Modulation) pins.
2. These pins can generate analog-like output voltages by rapidly switching between HIGH and LOW states at a fixed frequency and varying the duty cycle of the signal.
3. Labelled with ~ symbol on Digital Pins

analogRead() : The analogRead() function reads the analog voltage from the sensor, converting it to a digital value between 0 and 1023.

analogWrite(): The analogWrite() function is used to vary the voltages by adjusting the duty cycle of the PWM signal.

analogRead()

Description:

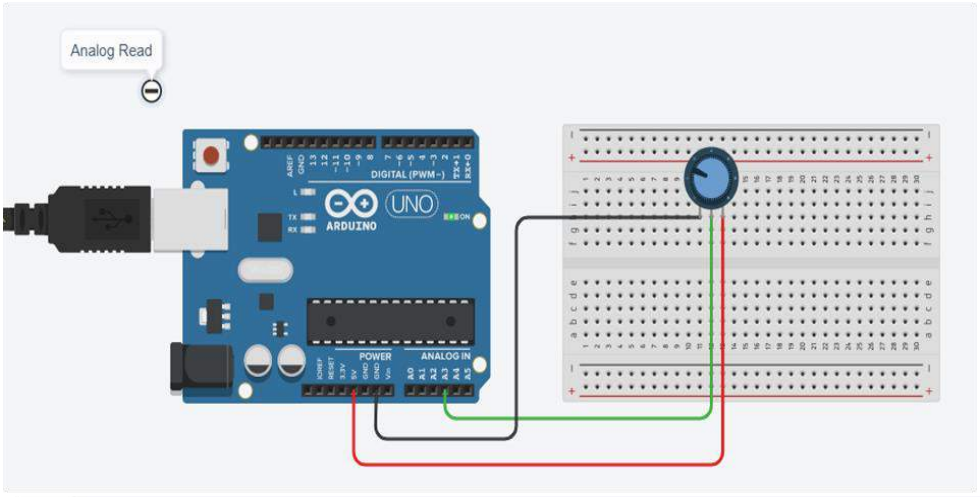
Reads the value from the specified analog pin.

Syntax:
analogRead(pin)

Parameters
pin: the name of the analog input pin to read from(A0 to A5 on Arduino UNO, A0 to A6 on MKR boards, A0 to A7 on the Mini and Nano, A0 to A15 on the Mega).

/The code reads the voltage on analogPin and displays it.

```
intanalogPin = A3; // potentiometer wiper (middle terminal) connected to analog pin 3
                   // outside leads to ground and +5V
intval = 0;  // variable to store the value read

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

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



analogWrite()

Description: Writes an analog value (PWM Wave) to a pin. Can be used to light a LED at varying brightnesses or drive a motor at various speeds.

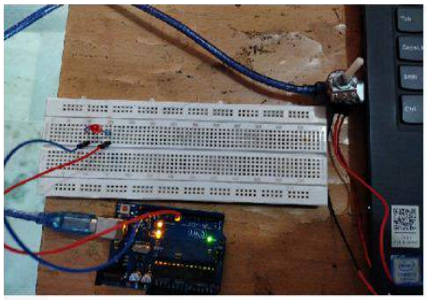
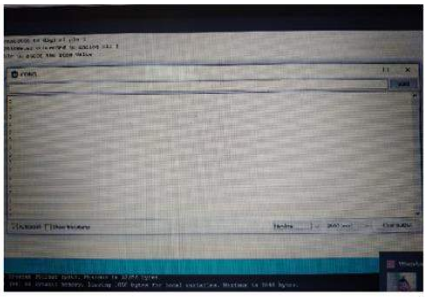
After a call to analogWrite(), the pin will generate a steady rectangular wave of the specified duty cycle until the next call to analogWrite() (or a call to digitalRead() or digitalWrite() on the same pin.

Syntax: analogWrite(pin, value)
Parameters: - pin: the Arduino pin to write to. Allowed data types: int.
value: the duty cycle: between 0 (always off) and 255 (always on). Allowed data types: int.

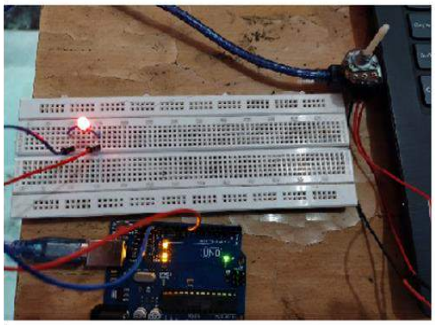
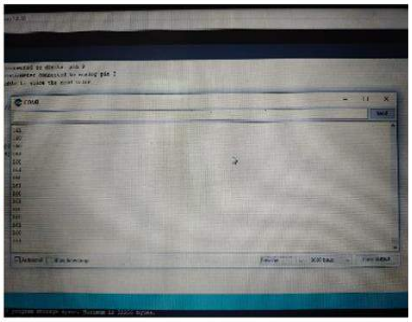
Example

```
//Sets the output to the LED proportional to the value read from the potentiometer.
intledPin = 9;       // LED connected to digital pin 9
intanalogPin = 3;   // potentiometer connected to analog pin 3
intval = 0;            // variable to store the read value

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

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

OUTPUT
LED OFF

LED ON

Serial Communication

Serial communication in Arduino is a method used for sending data between the Arduino board and other devices, such as computers, other Arduinos, or microcontrollers.
It allows you to transmit and receive data through the USB port or hardware serial Ports.
UART (Universal Asynchronous Receiver/Transmitter):
UART is responsible for transmitting and receiving serial data using two pins: TX (transmit) and RX (receive).

Baud Rate:
Baud rate is the speed at which data is transmitted over the serial connection.
It is measured in bits per second (bps).
Common baud rates include 9600, 19200, and 115200.

3. Serial Communication Functions:

Arduino provides a Serial library with functions to facilitate serial communication:
Serial.begin(baudRate): Initializes serial communication with the specified baud rate.
Serial.print(data): Sends data (numbers, characters, strings, etc.) to the serial port.
Serial.println(data): Sends data to the serial port followed by a newline character ('\n').
Serial.read(): Reads incoming serial data byte by byte.
Serial.available(): Returns the number of bytes available to read from the serial port buffer

4. Serial Monitor:
The Serial Monitor which allows you to view and send data between the Arduino and the computer. It's commonly used for debugging and monitoring serial communication.

Serial Communication Sketch Example

```
void setup() {
                    Serial.begin(9600); // Initialize serial communication at 9600 baud rate
                   }
 void loop() {
// Check if there is incoming data available to read
if (Serial.available() > 0)
  {
     // Read the incoming byte
     char receivedChar = Serial.read();
     Serial.print("Received character: ");
     Serial.println(receivedChar);
  }
// Send data to the Serial Monitor
Serial.print("Hello, world! ");
// Send binary data (ASCII value of 'A') over the serial port
Serial.write('A');
delay(1000); // Wait for 1 second  }
```

CONDITIONAL EXECUTION

1. Conditional execution, often implemented using conditional statements,
2. Conditional statements allows a program to make decisions and execute different blocks of code based on certain conditions.
3. In Arduino programming, conditional execution achieved using if, else if, else, and switch.

1. if Statement:

   The if statement is used to execute a block of code if a specified condition is true.

   Example: -

```
int sensorValue = analogRead(A0);
if (sensorValue > 500)
  {
     digitalWrite(LED_BUILTIN, HIGH); // Turn on the LED if sensor value is
                                                    greater than 500
  }
```

2. if-else Statement:

The if-else statement allows you to execute one block of code if the condition is true, and another block of code if the condition is false.

Example: -

```
int sensorValue = analogRead(A0);
if (sensorValue > 500)
  {
     digitalWrite(LED_BUILTIN, HIGH); // Turn on the LED if sensor value is
                                                    greater than 500
  }
else {
     digitalWrite(LED_BUILTIN, LOW); // Turn off the LED otherwise
  }
```

3. else if Statement:

The else if statement allows you to specify multiple conditions to be checked sequentially.

Example: -

```
int sensorValue = analogRead(A0);
if (sensorValue > 800)
    {
      // High sensor value, take some action
     }
     else if (sensorValue > 500)
           {
              // Medium sensor value, take some other action
            }
else {
           // Low sensor value, take another action
      }
```

3. switch Statement:

The switch statement provides an efficient way to select among multiple options based on the value of an expression.

Example: -

```
int option = 2;
switch (option) {
case 1:
        // Execute code for option 1
break;
case 2:
       // Execute code for option 2
break;
case 3:
        // Execute code for option 3
break;
default:
          // Execute code if none of the above cases match
 break;
                      }
```

INTERNET OF THINGS

UNIT – 3

I. Loops

Loops are control structures that allow you to execute a block of code repeatedly.
There are different types of loops available in Arduino, including for loop, while loop, and do-while loop.

1. for loop

The for loop is used when you know exactly how many times you want to repeat a block of code.

```
void setup() {
        // Initialization code here
    }
void loop() {
        // Loop code here
      for (int i = 0; i < 10; i++) {
      // Execute this block of code 10 times
      // i will be 0, 1, 2, ..., 9 in each iteration
                                    }
    }
```

2. While loop

The while loop is used when you want to repeat a block of code as long as a condition is true.

```
void setup() {
          // Initialization code here
      }
void loop() {
          // Loop code here
 int i = 0;
 while (i < 10) {
              // Execute this block of code as long as i is less than 10
            i++;
       }
    }
```

3. do-while Loop:

The do-while loop is similar to the while loop, but it always executes the block of code at least once before checking the condition.

```
void setup()
{
```

```
            // Initialization code here
  }
void loop() {
           // Loop code here
int i = 0;
do {
            // Execute this block of code at least once
           i++;
    } while (i < 10);
              }
```

Arrays

Arrays in Arduino are a collection of variables of the same type that are accessed by an index. They allow you to store multiple values of the same data type under one variable name.
Declaration and Initialization:
You can declare and initialize arrays in Arduino like this:

```
 // Declare an array of integers with a fixed size
int myArray[5];
// Initialize an array with values
int myArray[3] = {10, 20, 30};
```

Accessing Elements:

You can access individual elements of an array using square brackets [] and the index of the element (indices start from 0):

```
int myArray[3] = {10, 20, 30};
int value = myArray[1]; // Accesses the second element (20)
```

Iterating Through an Array:
You can use loops to iterate through the elements of an array:

```
 int myArray[3] = {10, 20, 30};
for (int i = 0; i < 3; i++)
    {
                Serial.println(myArray[i]); // Print each element of the array
    }
```

Arrays sketch

```
// Declare and initialize an array of integers

int myArray[] = {10, 20, 30, 40, 50};
void setup() {
                        Serial.begin(9600); // Initialize serial communication
                }
void loop() {
```

```
                    // Print the elements of the array
for (int i = 0; i < 5; i++)
        {
                Serial.print("Element ");
                Serial.print(i);
                Serial.print(": ");
                Serial.println(myArray[i]);
           }

                delay(1000); // Delay for stability
          }
```

Functions

Functions are blocks of code that perform a specific task. They allow you to break down your program into smaller, more manageable parts, making it easier to understand, debug, and maintain.

Creating a Function:

You can create a function in Arduino by specifying its return type, name, and parameters (if any), followed by a block of code enclosed in curly braces {}.

```
// Function declaration
void myFunction() {
                    // Function body
                   // Code to perform a specific task
                          }
```

Calling a Function:

To use a function, you simply call its name followed by parentheses ().

```
void setup() {
                    // Other setup code

                   // Call the function
myFunction();
                    }
 void loop() {
                 // Main loop code
               }
  // Function definition
  void myFunction() {
                      // Function body
                      // Code to perform a specific task
                          }
```

Function Parameters and Return Values:

You can pass parameters to a function to provide input values, and you can specify a return type to indicate the type of value the function will return.

```
// Function declaration with parameters
 int add(int x, int y) {
 return x + y; // Return the sum of x and y
                          }
 void setup() {
                 // Other setup code
                // Call the function and store the result in a variable
 int result = add(3, 5);
 Serial.println(result); // Print the result to the Serial Monitor
                  }
 void loop() {
                    // Main loop code
                }
```

Strings

In Arduino, strings can be manipulated using either the built-in String class or C-style character arrays (also known as C-strings).

Using the String Class:

The String class in Arduino provides a convenient way to manipulate strings. It allows you to perform operations such as concatenation, comparison, and conversion easily.

Example:

```
String firstName = "John";
String lastName = "Doe";
String fullName = firstName + " " + lastName;
Serial.println(fullName); // Output: John Doe
```

Using C-style Character Arrays (C-strings):

C-style character arrays (or C-strings) are arrays of characters terminated by a null character ('\0'). They are commonly used in C and C++ for handling strings.

Example:

```
char firstName[] = "John";
char lastName[] = "Doe";
char fullName[20]; // Allocate space for the full name
strcpy(fullName, firstName); // Copy first name to full name
strcat(fullName, " "); // Concatenate a space
```

```
strcat(fullName, lastName); // Concatenate last name
Serial.println(fullName); // Output: John Doe
```

Strings Sketch

```
String message = "Hello, Arduino!"; // Declare and initialize a String object
void setup() {
                Serial.begin(9600); // Initialize serial communication
              }
void loop() {
                Serial.println(message); // Print the message to the Serial Monitor

                delay(1000); // Delay for stability
            }
```

INTERNET OF THINGS

UNIT – 3

I. INTERFACING SENSORS AND ACTUATORS

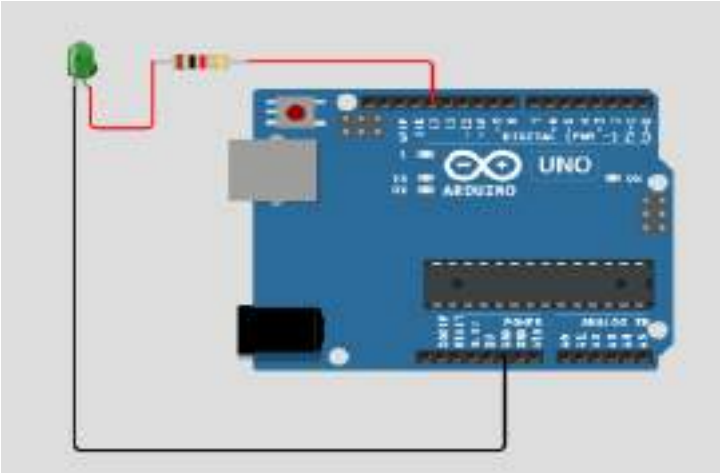
Controlling LED

- ### Integration of LED and Switch

LED: It's an electronic component used to emit light when an electric current passes through it.

These are the most common LEDs available in various colours, including red, green, blue, yellow, and white. They are widely used for indicator lights, displays, and general lighting.

Interfacing: Connect the longer lead (anode) to a current-limiting resistor and voltage source (e.g., 5V for most standard LEDs) and the shorter lead (cathode) to ground.
Use a current-limiting resistor to protect the LED from excess current. For example, a 220-330ohm resistor is commonly used.



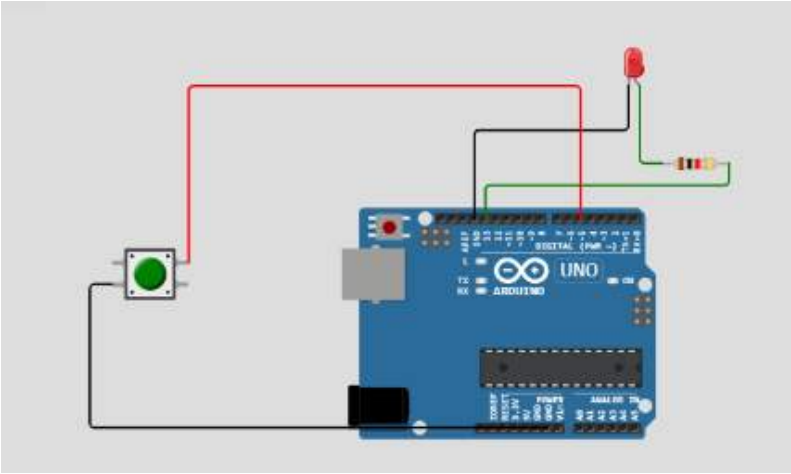
CONTROLLING LED SKETCH

```
const int ledPin = 13; // Digital pin connected to the LED
void setup() {
                pinMode(ledPin, OUTPUT); // Set the LED pin as an output
                  }
void loop() {
              digitalWrite(ledPin, HIGH); // Turn on the LED
              delay(1000); // Wait for 1 second
              digitalWrite(ledPin, LOW); // Turn off the LED
               delay(1000); // Wait for 1 second   }
```

INTEGRATION OF LED AND SWITCH

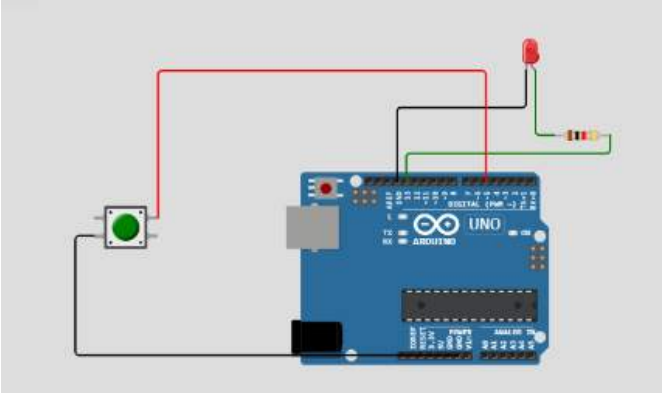
Switch: A switch is an electronic component that allows or interrupts the flow of electric current.
It has two states: open (off) and closed (on). When the switch is pressed or toggled, It changes its state from one to the other.

Interfacing: Connect one terminal of the push button to a digital pin on the Arduino (e.g., Pin 2). Connect the other terminal of the push button to the ground (GND) of the Arduino. Connect a resistor (typically 10k ohms) from the same digital pin (e.g., Pin 2) to the 5V supply (to create an external pull-up resistor).



INTEGRATION OF LED AND SWITCH

```
const int buttonPin = 5;  // Pin connected to the push button
const int ledPin = 13;    // Pin connected to an LED
void setup() {
                pinMode(buttonPin, INPUT_PULLUP);   // Set buttonPin as an input
                pinMode(ledPin, OUTPUT);     // Set ledPin as an output
                Serial.begin(9600);
              }
void loop() {
                  int buttonState = digitalRead(buttonPin);  // Read the state of the button
                  Serial.println(buttonState);
                   if (buttonState == HIGH)
                     {
                        digitalWrite(ledPin, HIGH);  // Turn on the LED when the button is pressed
                     }
                   else
                     {
                         digitalWrite(ledPin, LOW);   // Turn off the LED when the button is not
                                                              pressed
                      }
              }
```


UNO
ARDUINO

INTERNET OF THINGS

UNIT – 3

INTEGRATION OF LED AND LIGHT SENSOR (LDR)

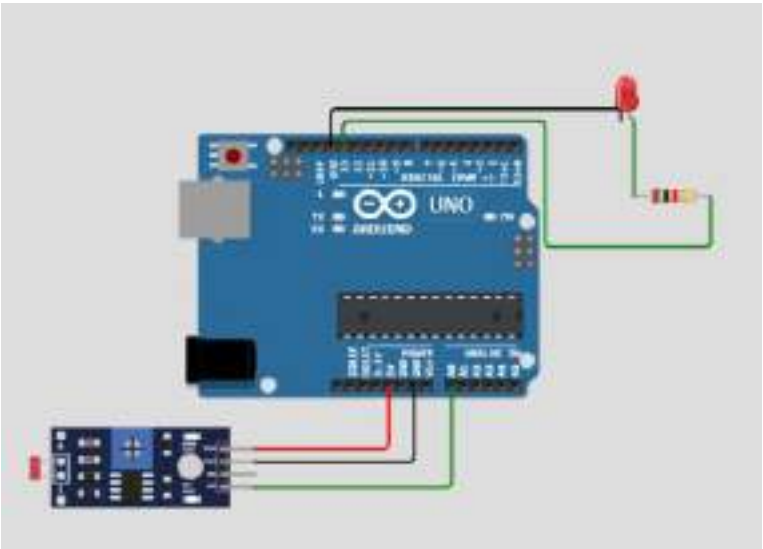
Photo resistor or Light Dependent Resistor:

A photo resistor changes its resistance based on the intensity of light falling on it.
In bright light, its resistance decreases, while in darkness, its resistance increases.

Interfacing:

LDR Connection: VCC to 5V: Connect one leg of the LDR to the 5V pin on the Arduino board.

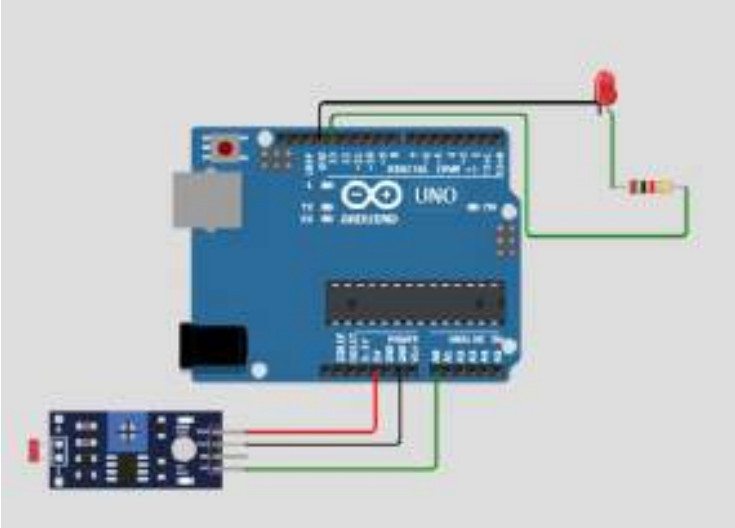
GND to GND: Connect the other leg of the LDR to any ground (GND) pin on the Arduino board.
Analog to A0: Connect the leg of the LDR (the sensing leg) to analog pin A0 on the Arduino



INTEGRATION OF LED AND LIGHT SENSOR (LDR)

```
// Define the analog pin where the LDR is connected

const int ldrPin = A0;
const int ledPin = 13; // Pin connected to an LED
const int THRESHOLD = 500; // Threshold value for light intensity
void setup() {
        pinMode(ledPin, OUTPUT);
```

```
                Serial.begin(9600);
               }

 void loop() {
                    // Read the analog value from the LDR
 int ldrValue = analogRead(ldrPin);
   if (ldrValue > THRESHOLD)
      {
              digitalWrite(ledPin, HIGH);  // Turn on the LED
      }
    else {
                digitalWrite(ledPin, LOW);  // Turn off the LED
           }
  Serial.print("LDR Value: ");
  Serial.println(ldrValue);
  delay(500);
}
```

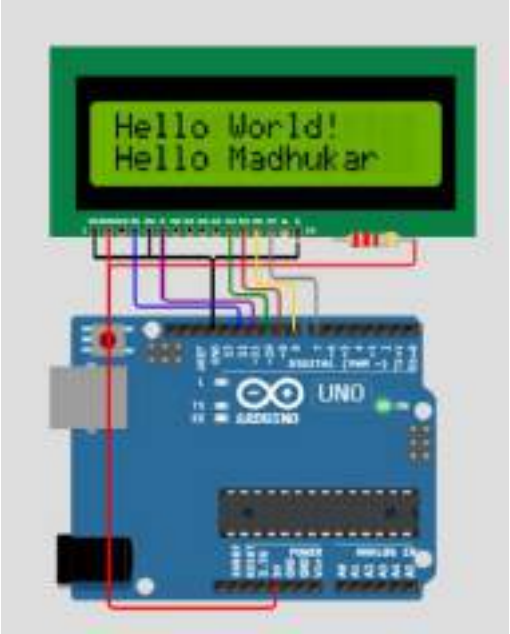


INTERFACING LCD

1. "LCD" stands for Liquid Crystal Display.
2. An LCD is a flat electronic display that uses the light-modulating properties of liquid crystals to display text, images, or other information.
3. They are available in different sizes and configurations, such as 16x2 (16 characters wide, 2 rows), 20x4, and others.
4. Arduino provides libraries like Liquid Crystal that simplify the process of controlling LCDs,

Connections

Connect the VSS pin of the LCD to GND.
Connect the VDD pin of the LCD to 5V.
Connect the V0 pin of the LCD to a variable resistor (potentiometer) to adjust contrast.
Connect the RS pin of the LCD to digital pin 12 of the Arduino.
Connect the RW pin of the LCD to GND (for write mode).
Connect the EN pin of the LCD to digital pin 11 of the Arduino.
Connect the D4-D7 pins of the LCD to digital pins 5-8 of the Arduino.
Connect the A (anode) pin of the LCD backlight to a 220-ohm resistor, and connect that resistor to 5V.
Connect the K (cathode) pin of the LCD backlight to GND.



Interfacing LCD Sketch

```
// LCD1602 to Arduino Uno connection example

#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()
    {
            // Set the cursor to the start of the second line
        lcd.setCursor(0, 1);
        lcd.print("Learn Arduino");
    }
```

I. INTERFACING SENSORS AND ACTUATORS

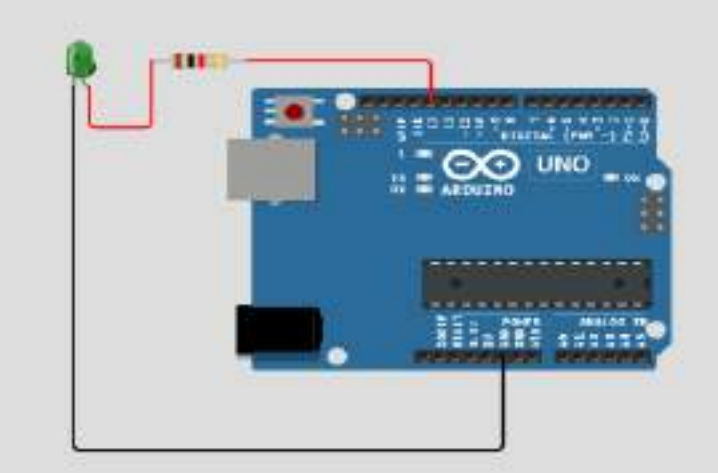
Controlling LED

- ### Integration of LED and Switch

LED: It's an electronic component used to emit light when an electric current passes through it.

These are the most common LEDs available in various colours, including red, green, blue,
yellow, and white. They are widely used for indicator lights, displays, and general lighting.

Interfacing: Connect the longer lead (anode) to a current-limiting resistor and voltage source
(e.g., 5V for most standard LEDs) and the shorter lead (cathode) to ground.
Use a current-limiting resistor to protect the LED from excess current. For example, a 220-330ohm
resistor is commonly used.



CONTROLLING LED SKETCH

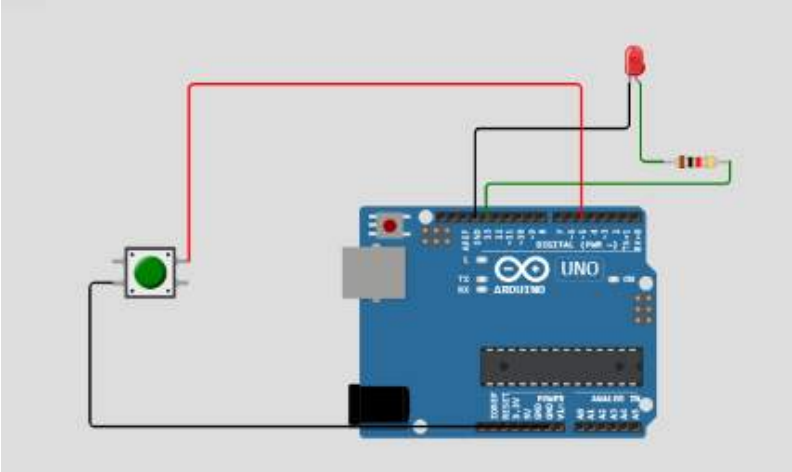
```
const int ledPin = 13; // Digital pin connected to the LED
void setup() {
                pinMode(ledPin, OUTPUT); // Set the LED pin as an output
                  }
void loop() {
              digitalWrite(ledPin, HIGH); // Turn on the LED
              delay(1000); // Wait for 1 second
              digitalWrite(ledPin, LOW); // Turn off the LED
```

delay(1000); // Wait for 1 second **}**

INTEGRATION OF LED AND SWITCH

Switch: A switch is an electronic component that allows or interrupts the flow of electric current.
It has two states: open (off) and closed (on). When the switch is pressed or toggled, It changes its state from one to the other.

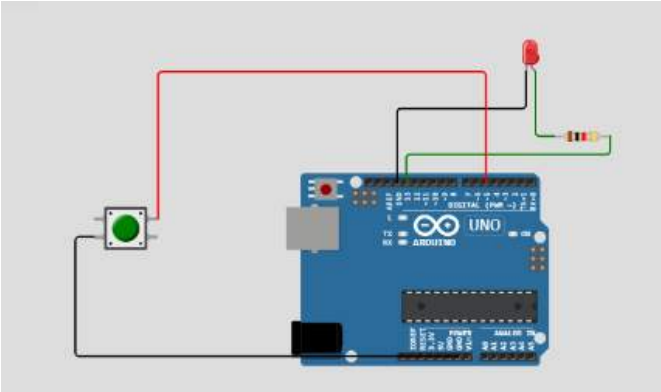
Interfacing: Connect one terminal of the push button to a digital pin on the Arduino (e.g., Pin 2).
Connect the other terminal of the push button to the ground (GND) of the Arduino.
Connect a resistor (typically 10k ohms) from the same digital pin (e.g., Pin 2) to the 5V supply (to create an external pull-up resistor).



INTEGRATION OF LED AND SWITCH

```
const int buttonPin = 5;  // Pin connected to the push button
const int ledPin = 13;    // Pin connected to an LED
void setup() {
                pinMode(buttonPin, INPUT_PULLUP);   // Set buttonPin as an input
                pinMode(ledPin, OUTPUT);     // Set ledPin as an output
                Serial.begin(9600);
              }
void loop() {
                  int buttonState = digitalRead(buttonPin);  // Read the state of the button
                  Serial.println(buttonState);
                   if (buttonState == HIGH)
                     {
                         digitalWrite(ledPin, HIGH);  // Turn on the LED when the button
      is pressed
                     }
                   else
```

```
    {
        digitalWrite(ledPin, LOW);   // Turn off the LED when the button
                                        is not pressed
    }
}
```



INTEGRATION OF LED AND LIGHT SENSOR (LDR)

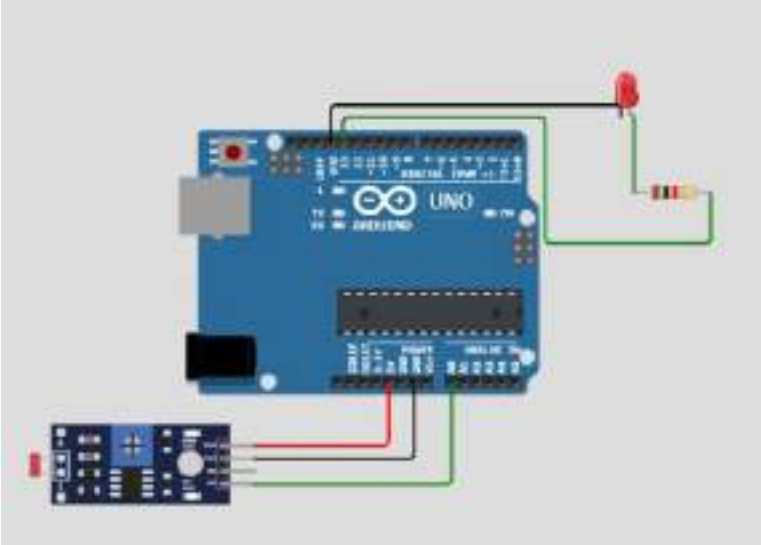
Photo resistor or Light Dependent Resistor:

A photo resistor changes its resistance based on the intensity of light falling on it.
In bright light, its resistance decreases, while in darkness, its resistance increases.

Interfacing:

LDR Connection: VCC to 5V: Connect one leg of the LDR to the 5V pin on the Arduino board.

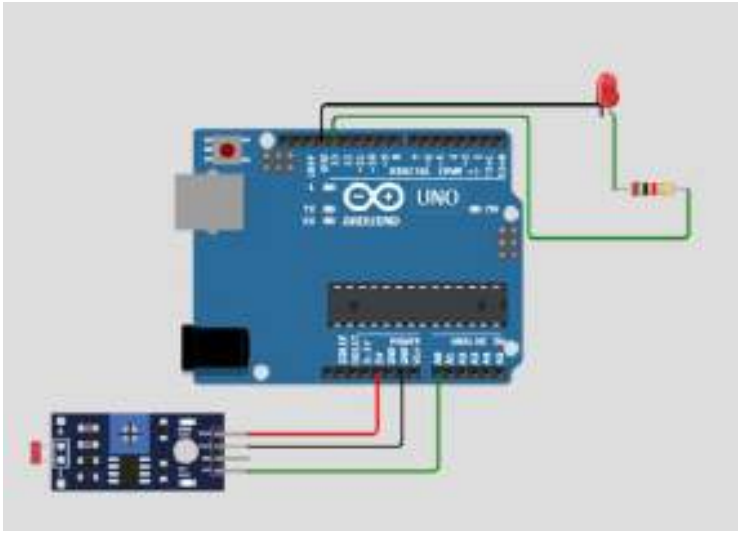
GND to GND: Connect the other leg of the LDR to any ground (GND) pin on the Arduino board.
Analog to A0: Connect the leg of the LDR (the sensing leg) to analog pin A0 on the Arduino



INTEGRATION OF LED AND LIGHT SENSOR (LDR)

```
// Define the analog pin where the LDR is connected

const int ldrPin = A0;
const int ledPin = 13; // Pin connected to an LED
const int THRESHOLD = 500; // Threshold value for light intensity
void setup() {
                pinMode(ledPin, OUTPUT);
                 Serial.begin(9600);
                }

 void loop() {
                    // Read the analog value from the LDR
```

```
 int ldrValue = analogRead(ldrPin);
   if (ldrValue > THRESHOLD)
      {
              digitalWrite(ledPin, HIGH);  // Turn on the LED
      }
    else {
               digitalWrite(ledPin, LOW);  // Turn off the LED
           }
  Serial.print("LDR Value: ");
  Serial.println(ldrValue);
  delay(500);
}
```

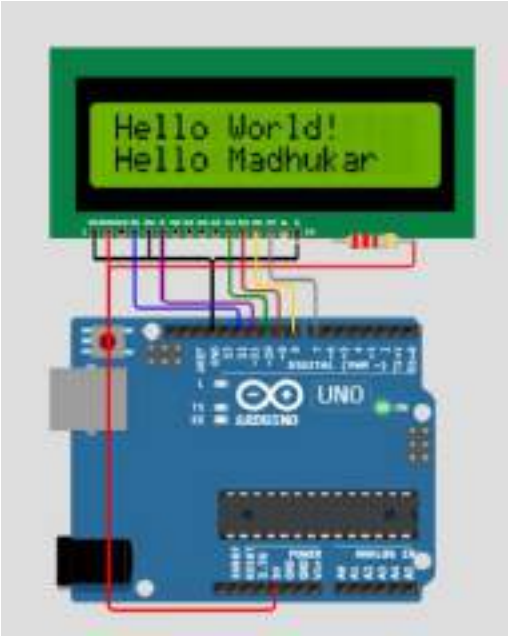


INTERFACING LCD

1. "LCD" stands for Liquid Crystal Display.
2. An LCD is a flat electronic display that uses the light-modulating properties of liquid crystals to display text, images, or other information.
3. They are available in different sizes and configurations, such as 16x2 (16 characters wide, 2 rows), 20x4, and others.
4. Arduino provides libraries like Liquid Crystal that simplify the process of controlling LCDs,

Connections

Connect the VSS pin of the LCD to GND.
Connect the VDD pin of the LCD to 5V.
Connect the V0 pin of the LCD to a variable resistor (potentiometer) to adjust contrast.
Connect the RS pin of the LCD to digital pin 12 of the Arduino.
Connect the RW pin of the LCD to GND (for write mode).
Connect the EN pin of the LCD to digital pin 11 of the Arduino.
Connect the D4-D7 pins of the LCD to digital pins 5-8 of the Arduino.
Connect the A (anode) pin of the LCD backlight to a 220-ohm resistor, and connect that resistor to 5V.
Connect the K (cathode) pin of the LCD backlight to GND.



Interfacing LCD Sketch

```
// LCD1602 to Arduino Uno connection example

#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()
    {
          // Set the cursor to the start of the second line
        lcd.setCursor(0, 1);
        lcd.print("Learn Arduino");
    }
```


Hello World!
Learn Arduino
UNO

INTERNET OF THINGS

UNIT – 3

Interfacing OLED with Arduino

Experiment 3 : Monitoring the voltage level of the battery and indicating the same using multiple LED's & OLED with Arduino/ESP32/Raspberry Pi.

AIM: To interface & write a program to monitor battery voltage level and indicating on OLED display and multiple LED's.

Equipment Required:

· Arduino UNO
· OLED Display
· Resistors 1kΩ, 10KΩ
· LED card
· 9V Hi-watt Battery
· Jumper wires

Pin wiring
Because the OLED display uses I2C communication protocol, wiring is very simple. You just need to connect to the Arduino Uno I2C pins as shown in the table below.

Pin	**Wiring to Arduino Uno**
Vin	5V
GND	GND
SCL	A5
SDA	A4

To control the OLED display you need the adafruit_SSD1306.h and the adafruit_GFX.h libraries. Follow the next instructions to install those libraries.

1. Open your Arduino IDE and go to Sketch > Include Library > Manage Libraries. The Library Manager should open.

2. Type “SSD1306” in the search box and install the SSD1306 library from Adafruit.

Program:

```
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#define OLED_RESET 4 Adafruit_SSD1306 display(OLED_RESET);
int greenLed = 2;
int yellowLed = 3;
 int redLed = 4;
int analogValue = 0;
float voltage = 0;
int ledDelay = 1000;
void setup()
{
display.begin(SSD1306_SWITCHCAPVCC, 0x3C); display.display();
display.clearDisplay();
Serial.begin(9600);
pinMode(greenLed, OUTPUT);
pinMode(yellow Led, OUTPUT);  pinMode(redLed,OUTPUT);
}
void loop()
{
analogValue = analogRead(A0);

voltage = 0.0048*analogValue; display.setTextSize(1); display.setTextColor(WHITE);
display.setCursor(0,0);
display.println(" Battery Voltage"); display.println("");
display.println(""); display.setTextSize(3);
display.print(" ");
display.println(voltage); display.display(); delay (200);
display.clear Display();
Serial.println(voltage);
if( voltage >= 1.35 )
digitalWrite(greenLed, HIGH);
else if (voltage > 1.2 && voltage < 1.35) digitalWrite(yellowLed, HIGH);
else if( voltage <= 1.2)
digitalWrite(redLed, HIGH);
delay(ledDelay);
digitalWrite(redLed, LOW); digitalWrite(yellowLed, LOW); digitalWrite(greenLed, LOW);
}
```

INTERNET OF THINGS

UNIT – 3

Interfacing LCD with Arduino

Experiment 4: Generatation of a random value similar to dice game and display the same using a 16X2 LCD with Arduino/ESP32/Raspberry Pi

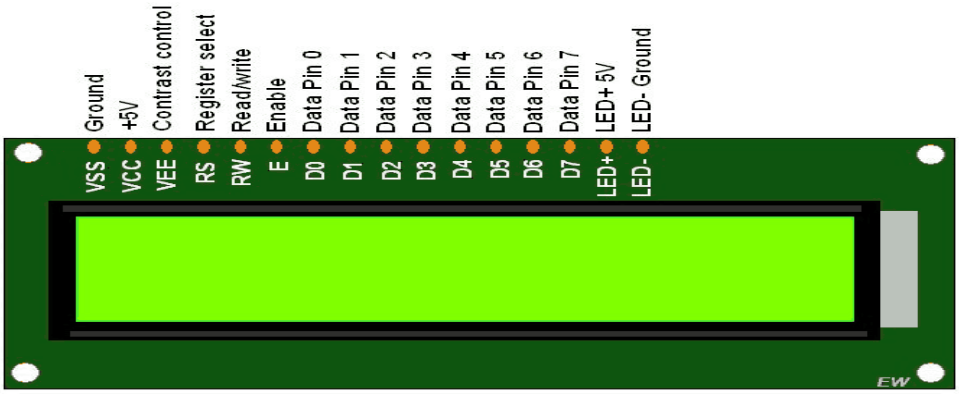
AIM:To interface & write a program to generate a random value similar to dice game simulation and display on LCD using Arduino.

Components Required:

· Arduino UNO
· 16X2 LCD
· Push button switch
· Resistor 470Ω, 1kΩ, 10KΩ
· Jumper wires
LCD(Liquid Crystal Display)

LCDs (Liquid Crystal Displays) are used in embedded system applications for displaying various parameters and status of the system.

LCD 16x2 is a 16-pin device that has 2 rows that can accommodate 16 characters each.LCD 16x2 can be used in 4-bit mode or 8-bit mode.
It is also possible to create custom characters.
It has 8 data lines and 3 control lines that can be used for control purposes.



Pin No	Function	Name
1	Ground (0V)	Ground
2	Supply voltage; 5V (4.7V – 5.3V)	Vcc
3	Contrast adjustment; through a variable resistor	VEE
4	Selects command register when low; and data register when high	Register Select
5	Low to write to the register; High to read from the register	Read/write
6	Sends data to data pins when a high to low pulse is given	Enable
7	8-bit data pins	DB0
8		DB1
9		DB2
10		DB3
11		DB4
12		DB5
13		DB6
14		DB7
15	Backlight VCC (5V)	Led+
16	Backlight Ground (0V)	Led-

Program:

```
#include <LiquidCrystal.h>
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);
  int randomnumber;

void setup()
            {           // put your setup code here, to run once:
              lcd.begin(16, 2);
              randomSeed(7);
              pinMode(8, INPUT);
            }


   void loop()
    {
              lcd.setCursor(2, 0);
              lcd.print("Dice value is:");
              int DICEROLL = digitalRead(8);
              if(DICEROLL==1)
```

```
        randomnumber = random(1,7);
         lcd.setCursor(6, 1);
         lcd.print(randomnumber);
}
```

OUTPUT: