

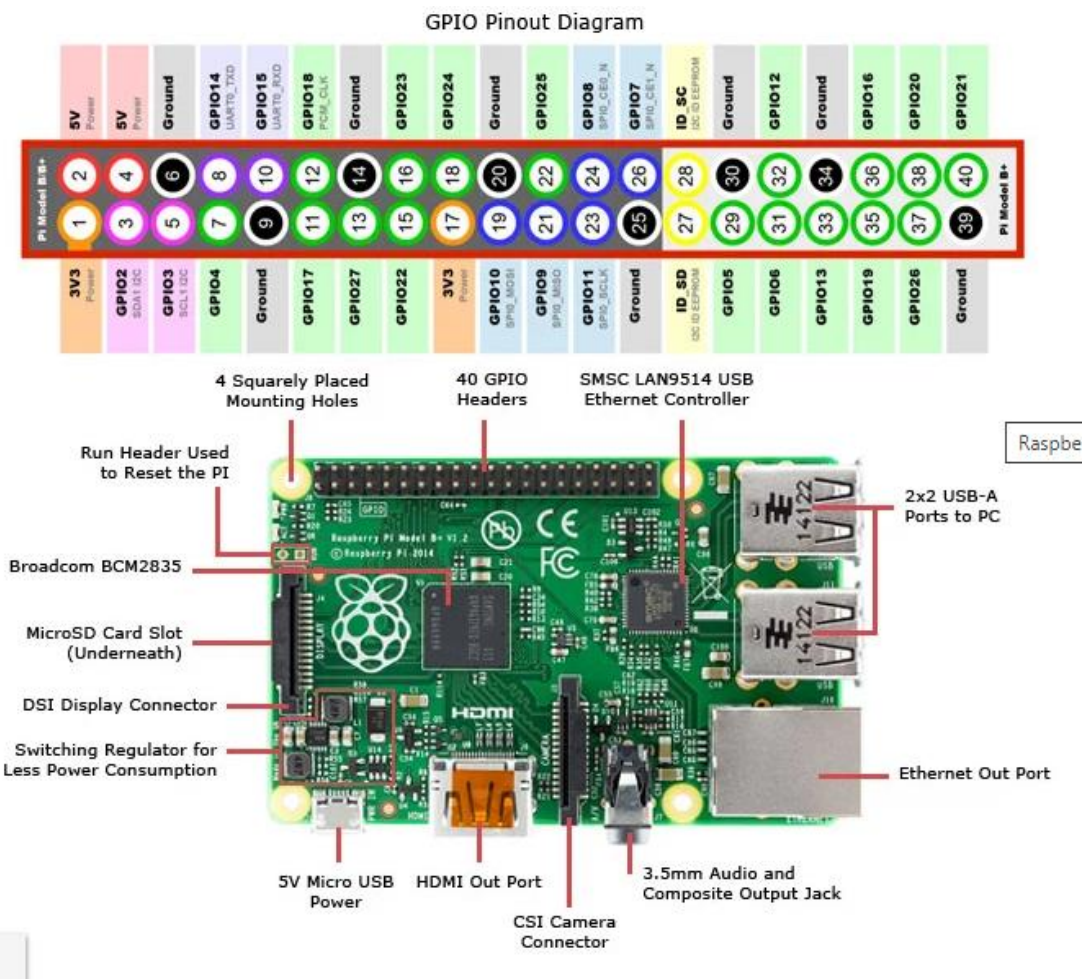
# Practical 1: Preparing Raspberry Pi: Hardware preparation and Installation

**Aim:** To study the raspberry pi with working and functionality

## Introduction:

What is a Raspberry Pi? Raspberry pi is the name of the “credit card-sized computer board” developed by the Raspberry pi foundation, based in the U.K. It gets plugged in a TV or monitor and provides a fully functional computer capability.

## Circuit diagram:



## 1. Central Processing Unit (CPU)

Every computer has a Central Processing Unit, and so does the Raspberry Pi. It is the computer's brain and carries out instructions using logical and mathematical operations. Raspberry Pi makes use of the ARM11 series processor on its boards.

## 2. HDMI port

Raspberry Pi board has an HDMI or High Definition Multimedia Interface port that allows the device to have video options of the output from the computer displayed. An HDMI cable connects the Raspberry Pi to an HDTV. The supported versions include 1.3 and 1.3. It also comes with an RCA port for other display options.

### 3. Graphic Processing Unit (GPU)

This unit, GPU or Graphic Processing Unit, is another part of the Raspberry pi board. Its primary purpose is to hasten the speed of image calculations.

### 4. Memory (RAM)

Random Access Memory is a core part of a computer's processing system. It is where real-time information is stored for easy access. The initial Raspberry Pi had 256MB RAM. Over the years, developers gradually and significantly improved the size. Different Raspberry Pi models come with varying capacities. The model with the maximum capacity presently is the Raspberry Pi 4 with 8GB RAM space.

### 5. Ethernet port

The Ethernet port is a connectivity hardware feature available on B models of Raspberry Pi. The Ethernet port enables wired internet access to the minicomputer. Without it, software updates, web surfing, etc., would not be possible using the Raspberry Pi. The Ethernet port found on Raspberry computers uses the RJ45 Ethernet jack. With this component, Raspberry Pi can connect to routers and other devices.

### 6. SD card slot

Like most other regular computers, Raspberry Pi must have some sort of storage device. However, unlike conventional PCs, it does not come with a hard drive, nor does it come with a memory card. The Raspberry Pi board has a Secure Digital card or SD card slot where users must insert SD cards for the computer to function. The SD card functions like a hard drive as it contains the operating system necessary for turning the system on. It also serves to store data.

### 7. General Purpose Input and Output (GPIO) pins

These are upward projecting pins in a cluster on one side of the board. The oldest models of the Raspberry Pi had 26 pins, but most have 40 GPIO pins. These pins are pretty sensitive and should be handled carefully. They are essential parts of the Raspberry Pi device as they add to its diverse applications. GPIO pins are used to interact with other electronic circuits. They can read and control the electric signals from other boards or devices based on how the user programs them.

### 8. LEDs

These are a group of five light-emitting diodes. They signal the user on the present status of the Raspberry Pi unit. Their function covers:

- ☐ PWR (Red): This functions solely to indicate power status. When the unit is on, it emits a red light and only goes off when the unit is switched off, or disconnected from the power source.
- ☐ ACT (Green): This flashes to indicate any form of SD card activity.
- ☐ LNK (Orange): LNK LED gives off an orange light to signify that active Ethernet connectivity has been established.
- ☐ 100 (Orange): This light comes on during Ethernet connection when the data speed reaches 100Mbps.
- ☐ FDX (Orange): FDX light also comes during Ethernet connection. It shows that the connection is a full-duplex.

### 9. USB ports

Universal service bus (USB) ports are a principal part of Raspberry Pi. They allow the computer to connect to a keyboard, mouse, hard drives, etc. The first model of Raspberry Pi had only two USB 2.0 ports. Subsequent models increased this number to four. Raspberry Pi 4 and Pi 400, much newer models, come with a mix of USB 2.0 and USB 3.0 ports.

#### 10. Power source

Raspberry Pi has a power source connector that typically uses a 5V micro USB power cable. The amount of electricity any Raspberry Pi consumes depends on what it's used for and the number of peripheral hardware devices connected.

# Practical 2: Introduction to Arduino

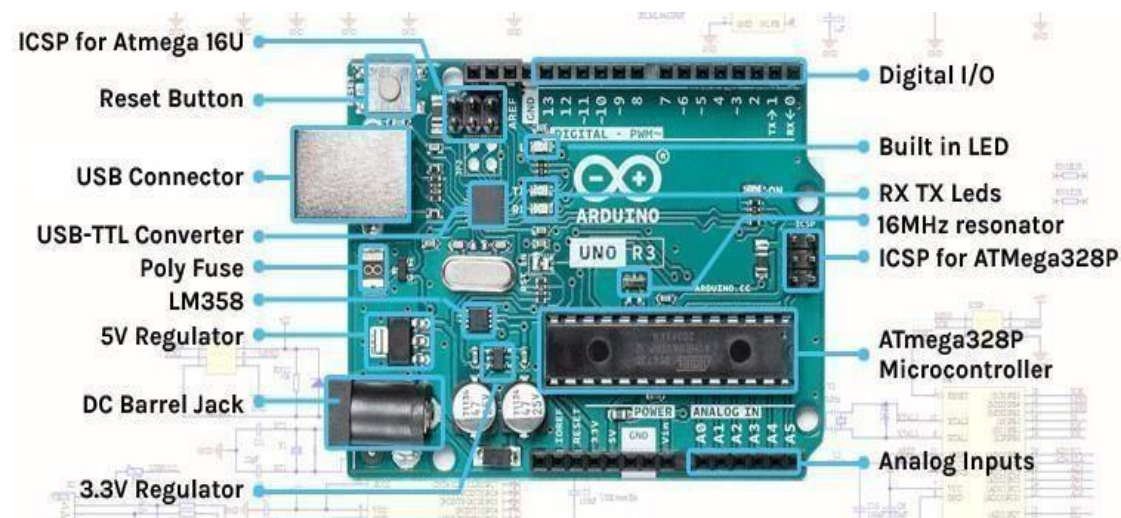
## Aim:

- 1.To study the basics of Arduino circuits and bread-boarding
- 2.Blinking of LEDs

Simulation Environment :TinkerCAD (Free online simulator)

Part A: Basics of Arduino Circuits

## Theory:



Arduino is an open-source electronics platform that has gained immense popularity for its ease of use and versatility. It was created in 2005 by a group of Italian engineers and is now maintained and developed by the Arduino community.

The heart of the Arduino platform is a microcontroller, which is a small, programmable computer on a single integrated circuit (IC) chip.

Arduino boards, which house these microcontrollers, provide a user-friendly environment for creating interactive electronic projects, prototypes, and various applications.

## Key Components of Arduino:

- 1.Microcontroller: The core of an Arduino board is the microcontroller. The most commonly used microcontroller in Arduino is the ATmega series from Atmel (now a part of Microchip Technology). These microcontrollers come in different variations and are the brains behind your Arduino projects.
- 2.Input/output Pins: Arduino boards have a set of digital and analog pins that can be used to read data (inputs) or send data (outputs). Digital pins work with binary signals (0 or 1), while analog pins can read a range of values. The number and types of pins vary among different Arduino board models.

3.Power Supply: Arduino boards can be powered via USB, an external power supply, or a battery. Some boards have built-in voltage regulators, which make them compatible with a range of power sources.

4.USB Port: Arduino boards often feature a USB port for programming and power supply. This allows you to connect the board to your computer and upload code.

5.Reset Button: A reset button is provided to restart the Arduino, allowing you to upload new code or reset the program.

6.LED Indicator: Many Arduino boards include a built-in LED (Light Emitting Diode) on pin 13, which can be used for testing and basic visual feedback.

#### Arduino Software:

The Arduino platform comes with its integrated development environment (IDE). The Arduino IDE is a software tool that allows you to write, compile, and upload code to the Arduino board. Key features of the IDE include:

- Programming Language: Arduino uses a simplified version of the C/C++ programming language. It provides a set of libraries and functions tailored for easy interaction with the hardware.

- Code Library: Arduino has a vast library of pre-written code and functions that simplify common tasks, making it accessible to beginners.

- Serial Monitor: The IDE includes a serial monitor that allows you to communicate with the Arduino board and view debugging information. -

Community Support: The Arduino community is large and active, offering forums, tutorials, and extensive documentation to help users troubleshoot issues and learn.

# Practical 3: Blinking of LEDs

## Components Used:

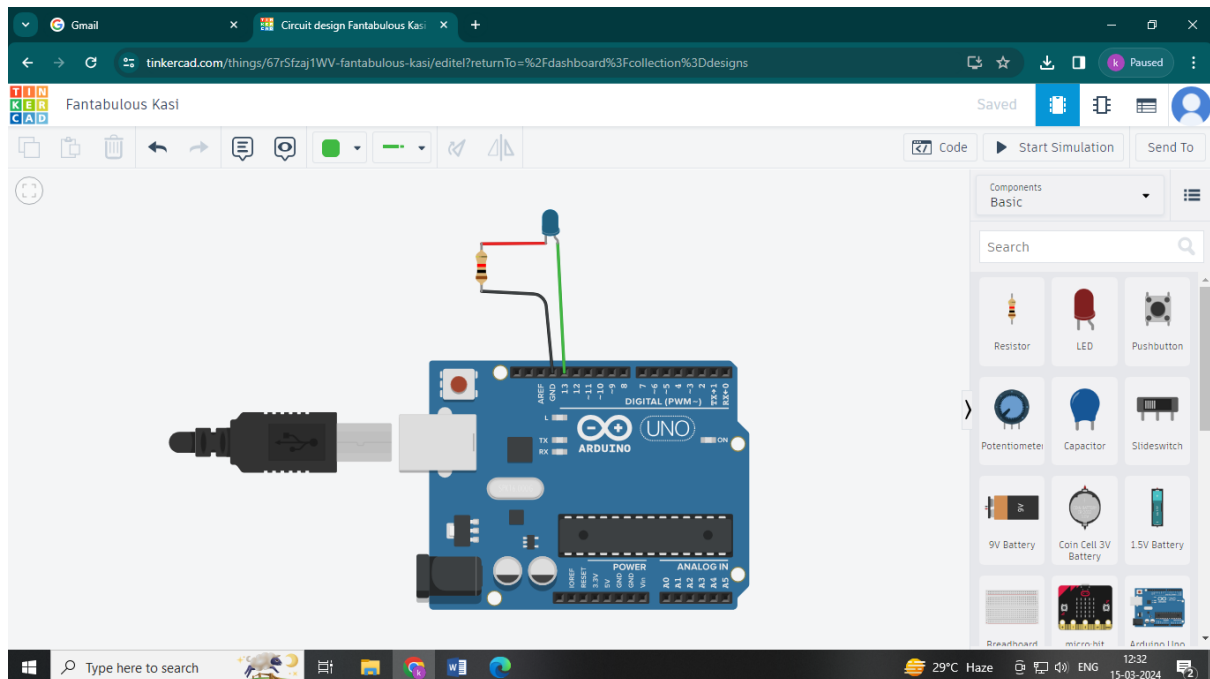
- 1.Arduino UNO
- 2.Breadboard
- 3.LED
- 4.Resistor (330  $\Omega$ )

The Following is the Circuit diagram we need to implement using the TinkerCAD simulation environment,

Arduino UNO is used to blink the LED continuously, we connect the pin 13 to the anode of the LED and cathode of the LED is connected to a resistor (330  $\Omega$ ) to limit the current passing through the LED. If large current flows through the LED then it may damage the LED (in real world environment).

The other end of the LED is terminated to the ground connection of the Arduino to complete the circuit.

## Circuit Diagram:

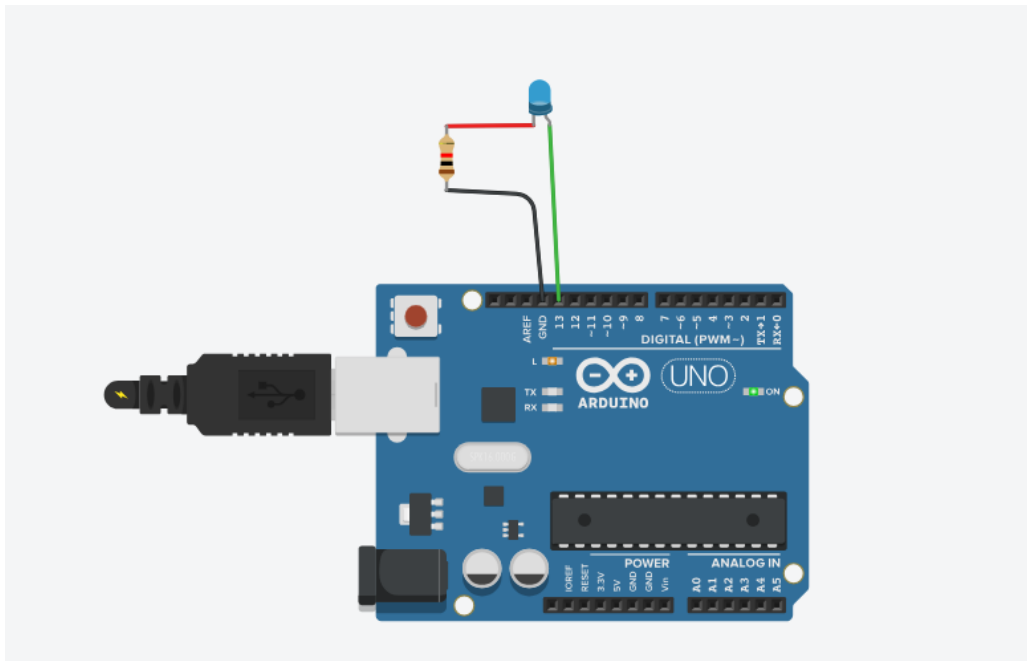


**Code:** The following C++ code is used in the given case

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

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

Output:



# Practical 4: Program using Light Sensitive Sensors

## Aim:

To study the working of Light sensor using Arduino

Simulation Environment :TinkerCAD (Free online simulator)

Components: Arduino UNO, LED, Photodiode and Resistors

## Theory:

The goal of this practical is to create a system that can automatically control the brightness of an LED based on the light detected by a photodiode. This project leverages the principles of light sensing and feedback control.

## Components:

- a)Photodiode: A photodiode is a light-sensitive semiconductor device that generates a current or voltage proportional to the incident light's intensity. It acts as the input sensor in this system.
- b)LED: An LED (Light Emitting Diode) is used as the output device. It emits light and can be controlled to vary its brightness.
- c)Arduino: The Arduino microcontroller is the brain of the project. It reads data from the photodiode, processes it, and controls the LED's brightness accordingly.

## Working:

### a)Photodiode Operation:

The photodiode is connected to one of the Arduino's analog input pins. When exposed to light, the photodiode generates a current or voltage that is directly proportional to the light intensity.

Arduino reads the analog voltage from the photodiode using one of its analog pins.

### b)Control Algorithm:

The Arduino is programmed with an algorithm that translates the analog reading from the photodiode into a control signal for the LED.

The algorithm typically involves mapping the photodiode's output to the LED's brightness. For example, when the photodiode detects more light, the LED becomes brighter, and vice versa.

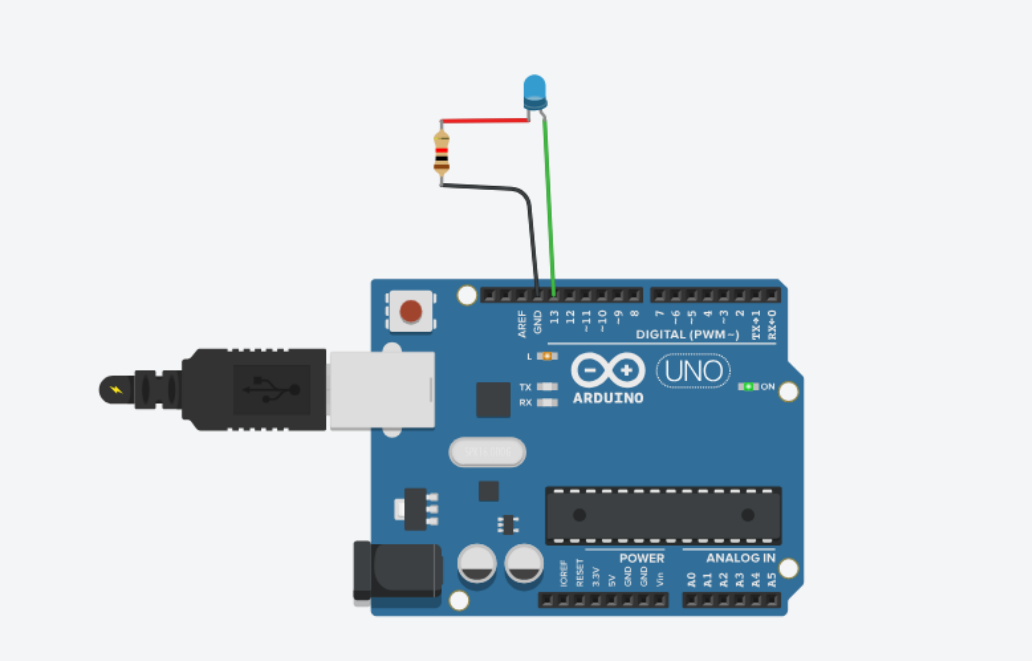
### c)Feedback Loop:

The system operates in a feedback loop. As light conditions change, the photodiode detects the variations and sends this information to the Arduino. The Arduino processes the data and adjusts the LED's brightness in real- time based on the input from the photodiode.

This closed-loop system ensures that the LED's brightness is always synchronized with the surrounding light levels.

Circuit Diagram:





# Practical 5: Program using Light Sensitive Sensors

## Aim:

To study the working of Light sensor using Arduino

Simulation Environment :TinkerCAD (Free online simulator)

Components: Arduino UNO, LED, Photodiode and Resistors

## Theory:

The goal of this practical is to create a system that can automatically control the brightness of an LED based on the light detected by a photodiode. This project leverages the principles of light sensing and feedback control.

## Components:

- a)Photodiode: A photodiode is a light-sensitive semiconductor device that generates a current or voltage proportional to the incident light's intensity. It acts as the input sensor in this system.
- b)LED: An LED (Light Emitting Diode) is used as the output device. It emits light and can be controlled to vary its brightness.
- c)Arduino: The Arduino microcontroller is the brain of the project. It reads data from the photodiode, processes it, and controls the LED's brightness accordingly.

## Working:

### a)Photodiode Operation:

The photodiode is connected to one of the Arduino's analog input pins. When exposed to light, the photodiode generates a current or voltage that is directly proportional to the light intensity.

Arduino reads the analog voltage from the photodiode using one of its analog pins.

### b)Control Algorithm:

The Arduino is programmed with an algorithm that translates the analog reading from the photodiode into a control signal for the LED.

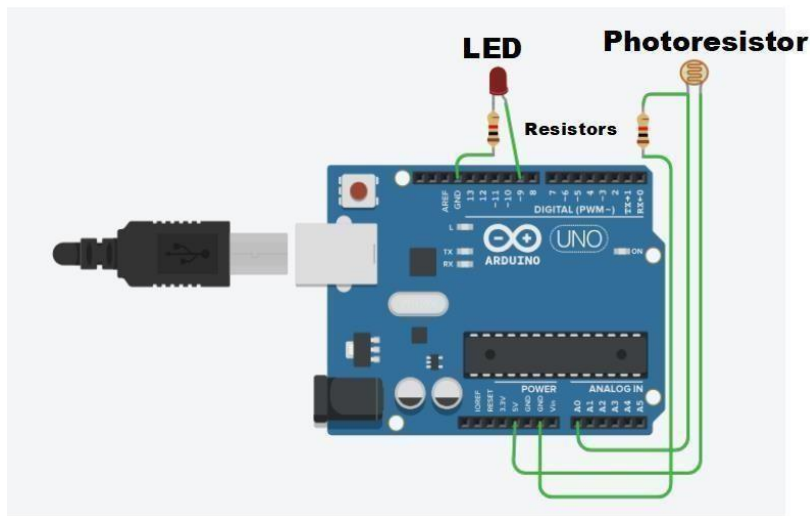
The algorithm typically involves mapping the photodiode's output to the LED's brightness. For example, when the photodiode detects more light, the LED becomes brighter, and vice versa.

### c)Feedback Loop:

The system operates in a feedback loop. As light conditions change, the photodiode detects the variations and sends this information to the Arduino. The Arduino processes the data and adjusts the LED's brightness in real- time based on the input from the photodiode.

This closed-loop system ensures that the LED's brightness is always synchronized with the surrounding light levels.

### Circuit Diagram:



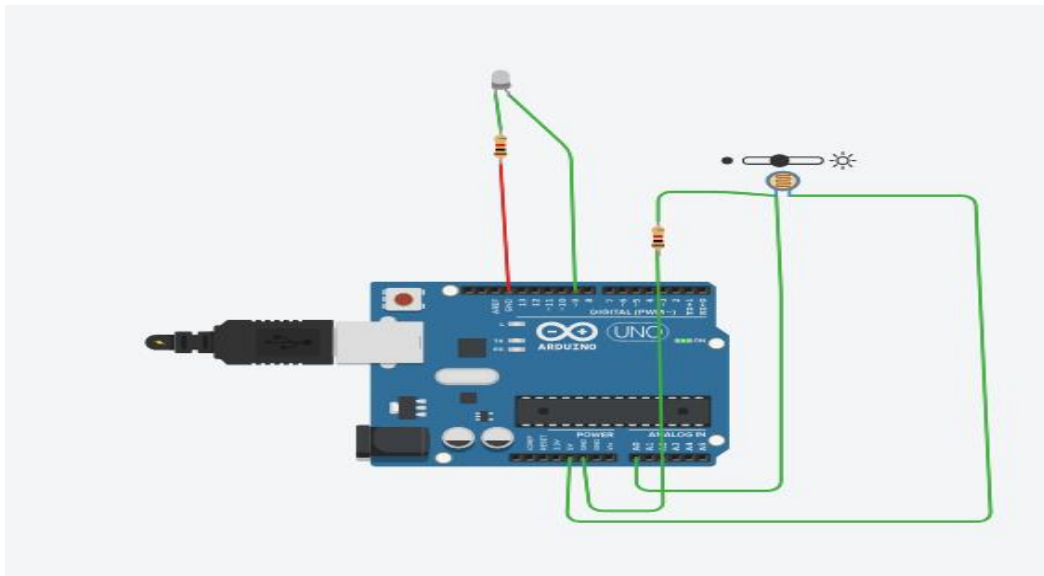
### Pin Connections:

Arduin	Photoresistor	LED
0		
5V	Right pin	
GND (Power)	Left pin through a Series Resistor	
A0	Left pin	
Pin 9		Anode
GND (Digital )		Cathode through a Series Resistor

**Code:** The following C++ code is used in the given case

```
int lightSensorValue = 0; void setup()
{
  pinMode(A0, INPUT); pinMode(9, OUTPUT); Serial.begin(9600);
}
void loop()
{
  lightSensorValue = analogRead(A0); Serial.println(lightSensorValue);
  int ledBrightness = map(lightSensorValue, 0, 1023, 0, 255); analogWrite(9, ledBrightness);
  delay(100);
}
```

**Output:**



# Practical 6 : Program using temperature sensors

## Aim:

To study the working of Temperature sensors using Arduino

Simulation Environment: TinkerCAD (Free online simulator)

Components: Arduino UNO, Temperature Sensor TMP 36

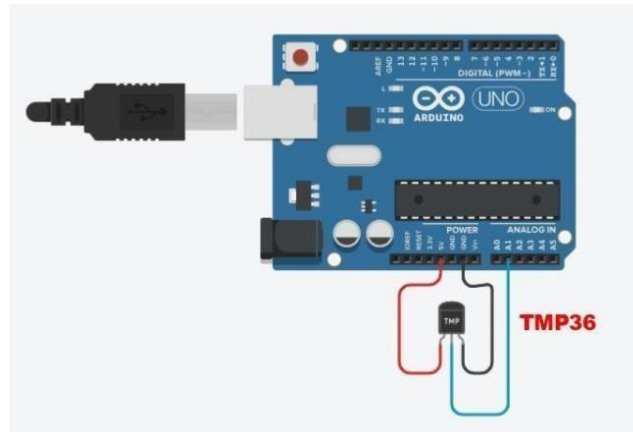
## Theory:

The TMP36 is a low-cost analog temperature sensor that can be easily integrated with Arduino boards. It provides an analog voltage output that varies linearly with temperature. This practical aims to show how to measure and display real-time temperature data using a TMP36 temperature sensor and an Arduino. The temperature data will be displayed through suitable method.

The TMP36 temperature sensor is a precision analog sensor. It generates an output voltage that is linearly proportional to the Celsius temperature. It typically has three pins: VCC, GND, and OUT. The sensor's output voltage increases by 10 mV per degree Celsius. At 25°C, it outputs 750 mV.

The demonstration showcases the practical application of the TMP36 temperature sensor in conjunction with an Arduino board for real-time temperature monitoring. It highlights how to interface the sensor, read its analog output, and display the temperature information. This knowledge can be applied to various temperaturesensing applications, including weather stations, environmental monitoring, and more.

## Circuit Diagram:



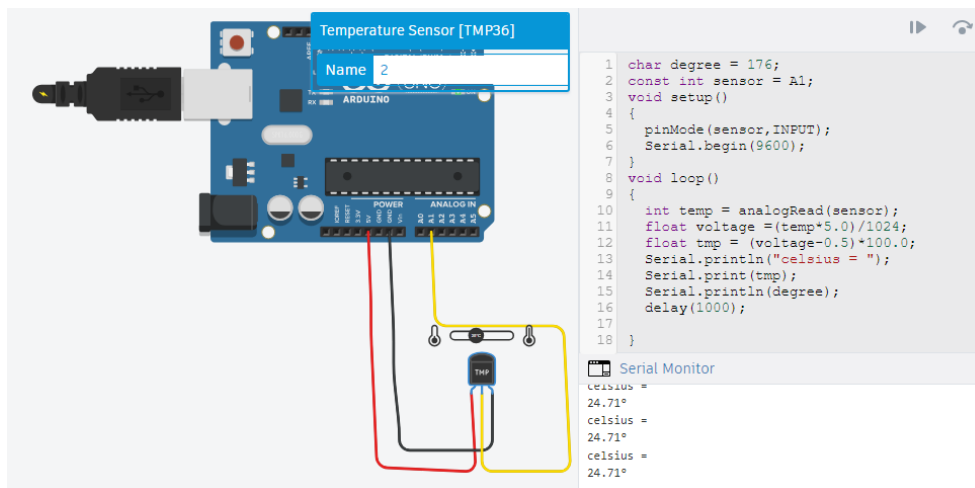
## Pin Connections:

Arduino	TMP36 Sensor
5V	Left pin
GND	Right pin
A1	Center pin

### Code:

```
char degree = 176; const int sensor = A1; void setup()
{
pinMode(sensor, INPUT); Serial.begin(9600);
}
void loop()
{
int tmp = analogRead(sensor); float voltage = (tmp*5.0)/1024;
float tmpCel = (voltage-0.5)* 100.0; Serial.print("Celsius:"); Serial.print(tmpCel);
Serial.println(degree);
delay(1000);
}
```

### Output:



The image shows a digital temperature sensor (TMP36) connected to an Arduino Uno. The sensor is plugged into the breadboard, and its pins are connected to the Arduino's power and ground pins. The Arduino is connected to a computer via a USB cable. The Serial Monitor window displays the output of the code, showing the temperature in Celsius.

Temperature Sensor [TMP36]  
Name: 2

```
1 char degree = 176;
2 const int sensor = A1;
3 void setup()
4 {
5   pinMode(sensor, INPUT);
6   Serial.begin(9600);
7 }
8 void loop()
9 {
10  int temp = analogRead(sensor);
11  float voltage = (temp*5.0)/1024;
12  float tmp = (voltage-0.5)*100.0;
13  Serial.println("celsius = ");
14  Serial.print(tmp);
15  Serial.println(degree);
16  delay(1000);
17 }
18
```

Serial Monitor

celsius =  
24.71°  
celsius =  
24.71°  
celsius =  
24.71°

# Practical 7: Programs using Servo Motors

## Aim:

To control the motion of a Servo motor using Arduino

Simulation Environment: TinkerCAD (Free online simulator)

**Components:** Arduino UNO, Servo motor

## Theory:

A micro servo motor is a small-sized servo motor designed for applications where space is limited. Servo motors, in general, are devices that incorporate a feedback mechanism to control the speed and position of the motor accurately. They are commonly used in robotics, remote-controlled vehicles, and various other projects where precise control of movement is required.

A micro servo motor functions as follows:

**Motor:** The motor inside the servo is responsible for producing the mechanical motion. It typically consists of a DC motor.

**Gear Train:** Servos have a gear train that converts the high-speed, low-torque output of the motor into low-speed, high-torque motion.

**Control Circuitry:** The control circuitry is responsible for interpreting the signals received from an external source (like an Arduino) and translating them into precise movements.

**Potentiometer (Feedback Device):** Most servo motors have a potentiometer (a variable resistor) connected to the output shaft. This potentiometer provides feedback to the control circuitry about the current position of the motor. When we connect a micro servo motor to an Arduino, we typically use a library (such as the Servo library in Arduino) to control its movements.

The working of Servo motor interfaced with Arduino can be understood as follows. The movement of a servo motor attached to an Arduino is controlled by sending a series of pulses to the servo motor. These pulses are typically generated using a technique called Pulse Width Modulation (PWM).

a. **Pulse Width Modulation (PWM):** Arduino boards have digital pins that can output PWM signals. PWM is a technique where the duration of a pulse is varied while the frequency remains constant. In the case of servo motors, the pulse width is crucial because it determines the position to which the servo motor should move.

b. **Servo Library:** Arduino provides a Servo library that simplifies the task of controlling servo motors. This library abstracts the details of generating PWM signals, making it easier to control the servo.

c. **Attach Function:** In the Arduino code, you first use the `attach` function to associate a servo object with a specific pin on the Arduino to which the signal wire of the servo is connected.

d. **Write Function:** To move the servo to a specific position, you use the

`write` function. The argument passed to this function is the desired angle. The angle corresponds to the position to which the servo should move. For example,

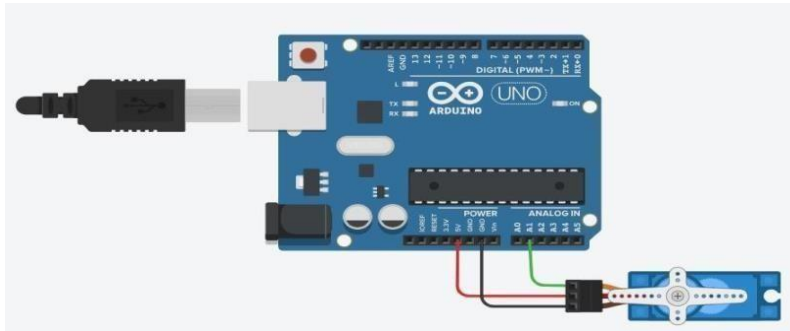
`myservo.write(90);` would move the servo to the 90-degree position.

e. **Pulse Generation:** Internally, the Servo library translates the angle specified in the `write` function into an appropriate pulse width. The library generates the necessary PWM signal, and the Arduino outputs this signal through the specified digital pin.

f. Control Loop: The servo motor's control circuitry interprets the PWM signal and adjusts the position of the motor accordingly. The feedback mechanism (potentiometer) inside the servo constantly provides information about the motor's current position to ensure that it reaches and maintains the desired position.

g. Looping or Sequential Control: In a loop or sequence of commands, you can vary the angles sent to the servo to make it move continuously or in a specific pattern.

### Circuit Diagram:



### Pin Connections:

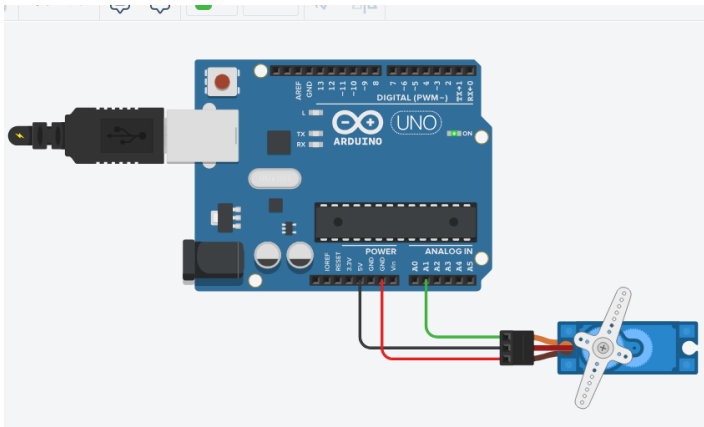
Arduino	Servo Motor
5V	Power
GND	Ground
Pin A1	Signal

### Code:

```
#include<Servo.h> Servo servoBase; void setup()
{
servoBase.attach(A1); servoBase.write(0);
}
void loop()
{
for (int i =0; i <= 180; i += 10)
{
servoBase.write(i); delay(2000);
}
}
```



## Output:



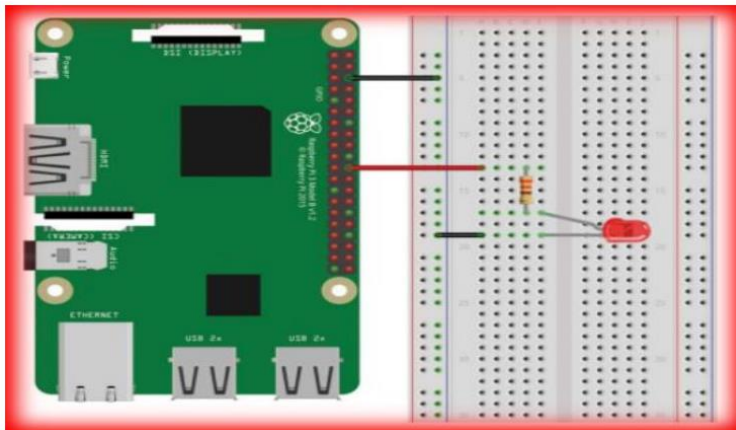
# Practical no 8: Blinking of LED with raspberry pi

**Components:** Raspberry pi system ,LED ,register ,wire

**Code:**

```
import RPi.GPIO as GPIO    ## Import GPIO library
import time
GPIO.setmode(GPIO.BOARD)   ## Use board pin numbering
GPIO.setup(11, GPIO.OUT)   ## Setup GPIO Pin 11 to OUT
while True:
    GPIO.output(11,1)       ## Turn on Led
    time.sleep(1)           ## Wait for one second
    GPIO.output(11,0)      ## Turn off Led
    time.sleep(1)           ## Wait for one second
```

**Output:**



# Practical 8

**Aim:** Program with 8x8 LED grid with formula

**Components:** Raspberry pi, 8x8 LED grid, wires

**Code:**

```
import re
import time
import argparse

from luma.led_matrix.device import max7219
from luma.core.interface.serial import spi, noop
from luma.core.render import canvas
from luma.core.virtual import viewport
from luma.core.legacy import text, show_message
from luma.core.legacy.font import proportional, CP437_FONT, TINY_FONT, SINCLAIR_FONT, LCD_FONT

def demo(n, block_orientation, rotate, msg):
    # create matrix device
    serial = spi(port=0, device=0, gpio=noop())
    device = max7219(serial, cascaded=n or 1, block_orientation=block_orientation, rotate=rotate or 0)
    show_message(device, msg, fill="white", font=proportional(LCD_FONT), scroll_delay=0.1)
    time.sleep(3)
    pass

if __name__ == "__main__":
    try:
        text_display = raw_input("Enter message to be display on 8x8 matrix - ")
        demo(1, 0, 0, text_display)
    except KeyboardInterrupt:
        pass
    finally:
        print "Program exit..."
```

**Output:**

