IoT & Automation Lab Record

**Lab-1**

**Lab Exercise: Blinking the In-Built LED**

void setup() {

pinMode(LED\_BUILTIN, OUTPUT);

}

void loop() {

digitalWrite(LED\_BUILTIN, HIGH);

delay(1000);

digitalWrite(LED\_BUILTIN, LOW);

delay(1000);

}

****

[@1.0 Blinking the In-Built LED Copy - Wokwi ESP32, STM32, Arduino Simulator](https://wokwi.com/projects/410076818759319553)

**Lab-2**

**Lab Exercise: Blinking an External Red LED**

#define sp 12

void setup() {

  pinMode(sp, OUTPUT);

}

void loop() {

  digitalWrite(sp, HIGH);

  delay(500);

  digitalWrite(sp, LOW);

  delay(500);

  }

****

[@2.0 Blinking an External LED Copy - Wokwi ESP32, STM32, Arduino Simulator](https://wokwi.com/projects/410076903678278657)

**Lab-3**

**Lab Exercise: Reading and Displaying Humidity and Temperature Data**

#include <DHT.h>

#define sp 7

#define DHTTYPE DHT22

DHT dht(sp, DHTTYPE);

float humid, temp;

void setup() {

  Serial.begin(9600);

  dht.begin();

}

void loop() {

  delay(200);

  humid = dht.readHumidity();

  temp = dht.readTemperature();

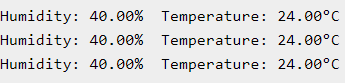
  Serial.print("Humidity: ");

  Serial.print(humid);

  Serial.print(" % Temperature: ");

  Serial.print(temp);

  Serial.println("°C");

  delay(1000);

}

[@3.0 Reading and Displaying Humidity and Temperature Data Copy - Wokwi ESP32, STM32, Arduino Simulator](https://wokwi.com/projects/410077049004657665)

**Lab-4**

**Lab Exercise: A case study on a communication Protocol: MQTT**

**In SystemOS [ Windows11]:**

**- Installed Mosquitto as a Service from Official Eclipse Page [ https://mosquitto.org/download/].**

**- This allows the MQTT Broker to run automatically in the background.**

**- Added mosquittio.exe to the System Environment Variables PATH [ ' C:\Program Files\mosquitto '], which allows us to use MQTT commands directly in the Command Prompt or, Terminal.**

**> Starting @ boot byDefault:**

**net start mosquitto**

**> Stopping:**

**In Elevated CMD > net stop mosquitto**

**> For Transmission: Navigate to [ cd C:/Program Files/mosquitto ]**

**mosquitto.exe -v**

**// -v is a Verbose Output flag, that enables us to see the backend processes, log messages, that'd help us to debug whenever necessary.**

**In Linux [ WSL\*: Ubuntu 22.04 LTS ]:**

- In Terminal > wsl --install -d Ubuntu-22.04 >,then press Enter.

- Restart the machine, and Launch Ubuntu 22.04

- $sudo apt update

- $sudo apt install mosquitto mosquitto-clients

> Starting mosquitto services:

- $sudo systemctl (enable /start) mosquitto

- Mosquitto Broker Service Status can be checked here:

- $sudo systemctl status mosquitto

- Once verified service status, transmission can be carried on.

> Stopping mosquitto services:

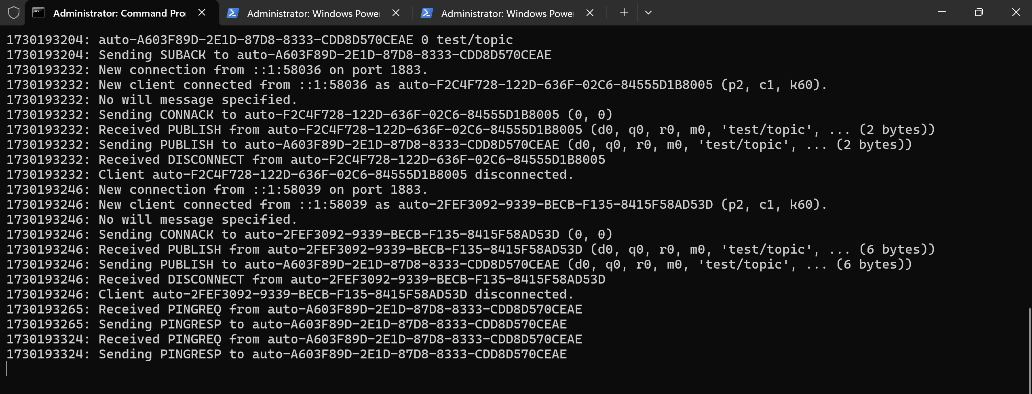
- $sudo systemctl stop mosquitto

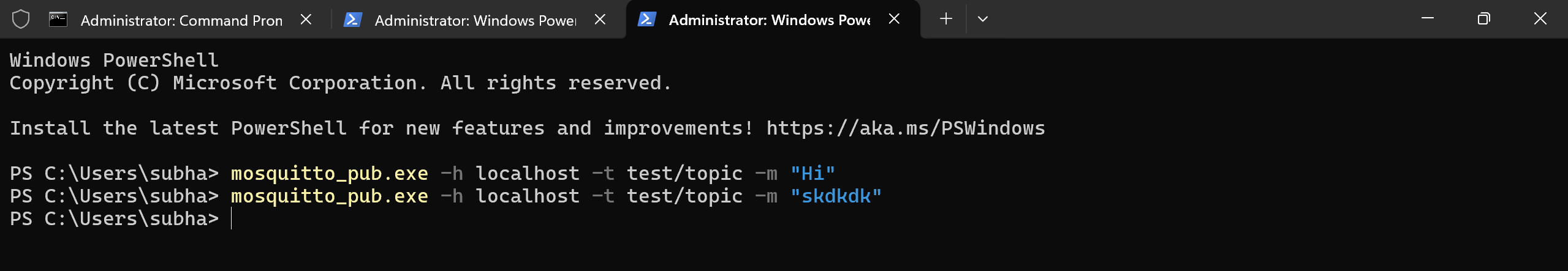
**Testing MQTT Services [ Message Transmission: WinOS11 + Ubuntu 22.04 ]:**

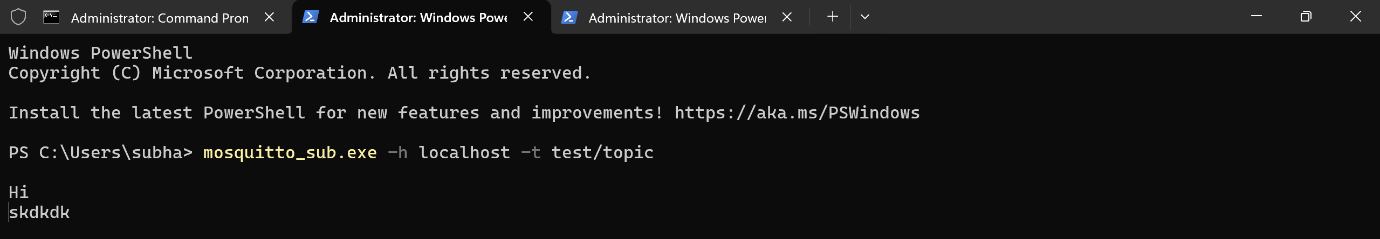
- Open 2 Terminals:

# 1st: mosquitto\_sub.exe -h localhost -t test/topic

# 2nd: mosquitto\_pub.exe -h localhost -t test/topic -m "Hi"

Outputs:





**Lab-5**

**Lab Exercise: Building a web app using Node-Red to fetch DHT sensor data and display it on the web app dashboard**

**Install Node.js:**

**- Installed NodeJS from Official Eclipse Page [ https://nodejs.org/en/download/package-manager ].**

**- Added node.js to the System Environment Variables PATH [ ' C:\Users\subha\AppData\Roaming\npm '], which allows us to use npm commands directly in the Command Prompt or, Terminal.**

**Installing & Initialising NodeRED:**

**- Open Node.js > npm install node-red-dashboard**

**- [PostInstallation] > Command Prompt: node-red**

**> In Client Application, browsed localhost:1880 [ Accessing NodeRED ]**

**- Inside the NodeRED window, a flow was created within the nodes as:**

**> SERIAL-IN (Arduino Uno R3 Board)**

**> DEBUGGER**

**> DHT FUNCTION**

**> 2 GAUGES (Humidity & Temperature)**

**- Serial In Node: Configured it to read from the correct serial port where my Arduino is connected (e.g., COM11) > Set the baud rate to 9600.**

**- Configure the DHT Function as:**

**var m = msg.payload.split(',');**

**if (m.length === 2) {**

**var H = { payload: parseFloat(m[0]) };**

**var T = { payload: parseFloat(m[1]) };**

**return [H, T];**

**} else {**

**return null;**

**}**

**- Adjusting Gauge Nodes:**

**> Humidity:**

**- Title as "Humidity".**

**- Value format as `{{value}}%`.**

**- Minimum value to 0 and the maximum to 100.**

**> Temperatue:**

**- Title as ' Temperature '**

**- Value format as {{value}}°C.**

**- Ensure that Humidity & Temperature are in the same group.**

**Deployment:**

**- Uploaded DHT11 /22 Sketch to the Arduino Board through its IDE:**

**#include <DHT.h>**

**#define DHTPIN 3**

**#define DHTTYPE DHT11**

**DHT dht(DHTPIN, DHTTYPE);**

**void setup() {**

**Serial.begin(9600);**

**dht.begin();**

**}**

**void loop() {**

**float H = dht.readHumidity();**

**float T = dht.readTemperature();**

**if (isnan(H) || isnan(T)) {**

**Serial.println("Failed to read from DHT sensor!");**

**} else {**

**Serial.println(String(H) + "," + String(T));**

**}**

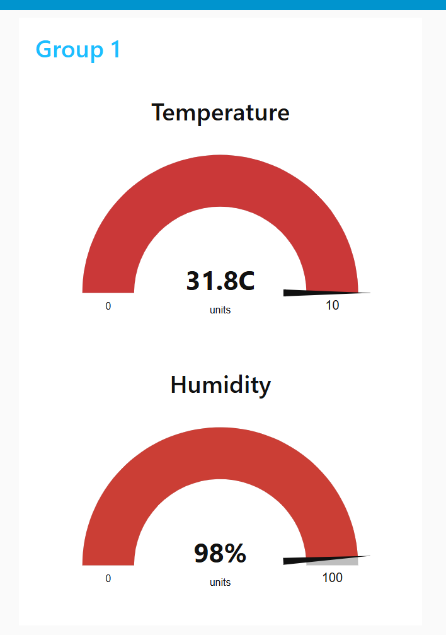
**delay(2000);**

**}**

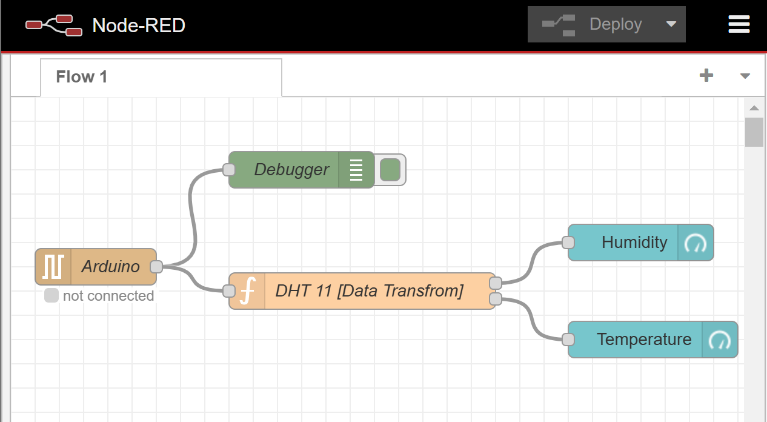
**- After uploading this sketch, close the IDE.**

**- Deploy the flow in NodeRED.**

**- Check the Dashboard in the upper-right corner, for the Humidity and Temperature Gauge.**

****

**Outputs:**

****

**Lab-6**

**Lab Exercise: Interfacing Ultrasonic Sensor**

**UltraSonic [ HC-SR04]: Measures Distance**

**Step#1: Connect 4 jumper wires to the UltraSonic Sensor as [VCC: 5v], [Trigger: 9], [Echo: 8], & GND.**

**Step#2: After configuring the HC-SR04 w/ UNO R3, this sketch is to be uploaded and executed in the IDE:**

**#define PIN\_TRIG 9 // Define the pin for the trigger**

**#define PIN\_ECHO 8 // Define the pin for the echo**

**void setup() {**

**Serial.begin(9600); // Initialize serial communication at 9600 baud**

**pinMode(PIN\_TRIG, OUTPUT); // Set the trigger pin as output**

**pinMode(PIN\_ECHO, INPUT); // Set the echo pin as input**

**}**

**void loop() {**

**// Start a new measurement:**

**digitalWrite(PIN\_TRIG, HIGH); // Set the trigger pin high**

**delayMicroseconds(10); // Wait for 10 microseconds**

**digitalWrite(PIN\_TRIG, LOW); // Set the trigger pin low**

**// Read the result:**

**int duration = pulseIn(PIN\_ECHO, HIGH); // Read the duration of the pulse from the echo pin**

**// Calculate distance in centimeters:**

**Serial.print("Distance in CM: ");**

**Serial.println(duration / 58); // Print the distance in centimeters**

**// Calculate distance in inches:**

**Serial.print("Distance in inches: ");**

**Serial.println(duration / 148); // Print the distance in inches**

**delay(1000); // Wait for 1 second before taking the next measurement**

**}**

**Output: *Distance in CM: 121 ; Distance in inches: 47***

**HC-SR04 + PushButton**

**Step#3: Now, a button can be connected to Digital 2 and diagonally to GND.**

**Step#4: After configuring the Button & the sensor, this sketch is to be uploaded and executed in the IDE:**

**#define PIN\_TRIG 9 // Define the pin for the trigger**

**#define PIN\_ECHO 8 // Define the pin for the echo**

**#define buttonPin 2 // Define the pin for the button**

**void setup() {**

**Serial.begin(9600); // Initialize serial communication at 9600 baud**

**pinMode(PIN\_TRIG, OUTPUT); // Set the trigger pin as output**

**pinMode(PIN\_ECHO, INPUT); // Set the echo pin as input**

**pinMode(buttonPin, INPUT\_PULLUP); // Set the button pin as input with internal pull-up resistor**

**}**

**void loop() {**

**int buttonState = digitalRead(buttonPin); // Read the state of the button**

**if (buttonState == LOW) { // Check if the button is pressed**

**// Start a new measurement:**

**digitalWrite(PIN\_TRIG, HIGH); // Set the trigger pin high**

**delayMicroseconds(10); // Wait for 10 microseconds**

**digitalWrite(PIN\_TRIG, LOW); // Set the trigger pin low**

**// Read the result:**

**int duration = pulseIn(PIN\_ECHO, HIGH); // Read the duration of the pulse from the echo pin**

**// Calculate distance in centimeters:**

**Serial.print("Distance in CM: ");**

**Serial.println(duration / 58); // Print the distance in centimeters**

**// Calculate distance in inches:**

**Serial.print("Distance in Inches: ");**

**Serial.println(duration / 148); // Print the distance in inches**

**} else {**

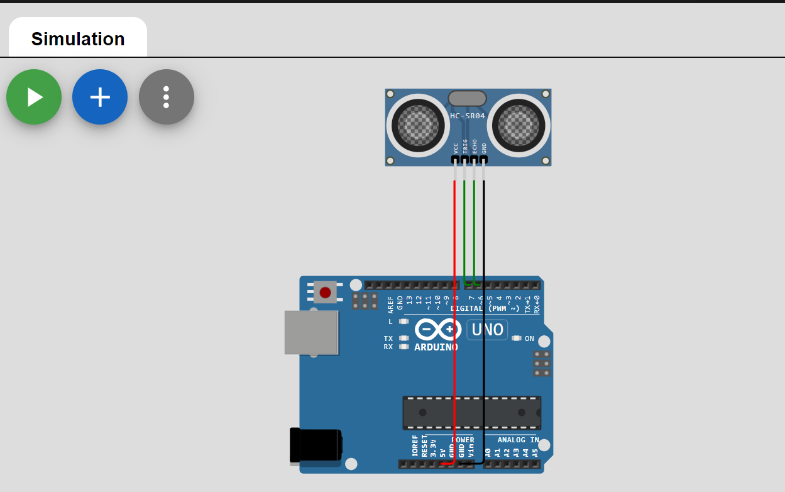
**Serial.println("Button Released!"); // Print message to Serial Monitor**

**}**

**delay(500); // Add a small delay to debounce the button**

**}**

**Output: *Button Released! ... ; Distance in CM: 24 ; Distance in Inches: 9***

****

**Lab-7**

**Lab Exercise: Use of ESP32, upload code on ESP 32 to blink onboard LED**

** Open Wokwi Simulator: Go to** [**Wokwi's ESP32 simulation page**](https://wokwi.com/) **and start a new ESP32 project.**

** Add ESP32 Component:**

* **In the simulator, add the ESP32 chip from the components list. This will give you the virtual ESP32 setup, ready for code.**

** Write the Code:**

* **In the code editor on Wokwi, write a simple program to blink the onboard LED.**

** Code Explanation:**

* **We’ll define the LED pin as GPIO 2.**
* **Use the setup() function to set the LED pin as an output.**
* **Use the loop() function to turn the LED on, wait, then turn it off, and wait again, repeating indefinitely.**

** Run the Simulation:**

* **Click the "Play" button on Wokwi to see the onboard LED blinking.**

**// Define the LED pin**

**#define LED\_PIN 2 // The onboard LED is connected to GPIO 2 on most ESP32 boards**

**void setup() {**

**// Initialize the LED pin as an output**

**pinMode(LED\_PIN, OUTPUT);**

**}**

**void loop() {**

**// Turn the LED on (HIGH is the voltage level)**

**digitalWrite(LED\_PIN, HIGH);**

**// Wait for a second**

**delay(1000);**

**// Turn the LED off by making the voltage LOW**

**digitalWrite(LED\_PIN, LOW);**

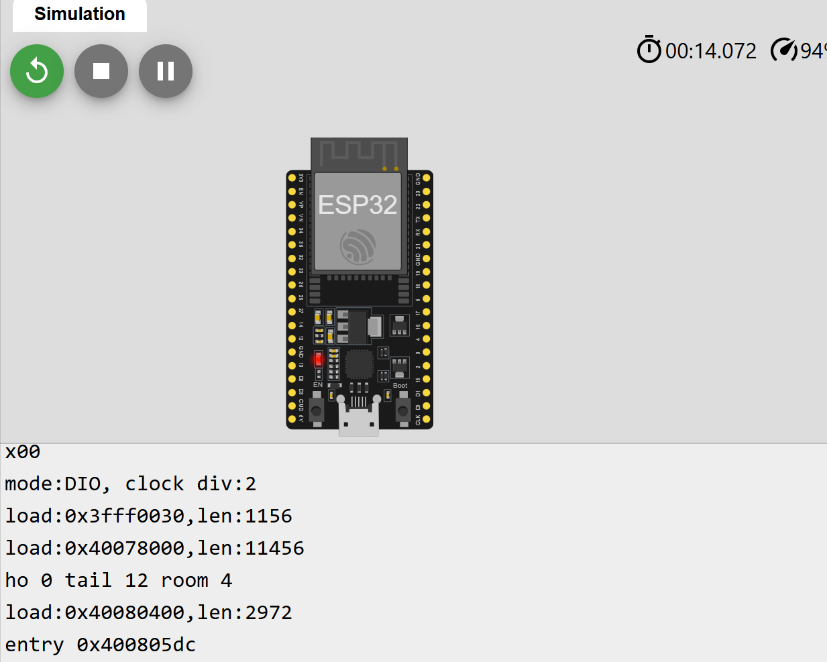
**// Wait for a second**

**delay(1000);**

**}**

**Click "Start Simulation" on Wokwi after entering the code. You should see the onboard LED blink.**

**Output:**

****

**https://wokwi.com/projects/413174757120925697**

**Lab-8**

**Lab Exercise: Use of Breadboard**

**Step 1: Open Wokwi Simulator**

* **Go to** [**Wokwi's Arduino simulation page**](https://wokwi.com/) **and start a new Arduino Uno project.**

**Step 2: Set Up the Breadboard and Components**

* **Arduino Uno: Add an Arduino Uno R3 to your project.**
* **Breadboard: Add a small breadboard for arranging the LEDs and wires.**
* **LEDs: Place 4 LEDs on the breadboard.**

**Step 3: Connect the Components**

* **Power (5V): Connect the 5V pin on the Arduino to the power rail on the breadboard.**
* **Ground (GND): Connect the GND pin on the Arduino to the ground rail on the breadboard.**
* **LED Connections:**
  + **Connect each LED’s cathode (shorter leg) to the ground rail.**
  + **Connect the anode of each LED (via the resistor) to a different digital pin on the Arduino:**
    - **LED 1 → Digital Pin 10**
    - **LED 2 → Digital Pin 11**
    - **LED 3 → Digital Pin 12**
    - **LED 4 → Digital Pin 13**

**Step 4: Write the Code**

* **The code below will make each LED blink in sequence.**

**// Define LED pins**

**const int ledPin1 = 10;**

**const int ledPin2 = 11;**

**const int ledPin3 = 12;**

**const int ledPin4 = 13;**

**void setup() {**

**// Set LED pins as outputs**

**pinMode(ledPin1, OUTPUT);**

**pinMode(ledPin2, OUTPUT);**

**pinMode(ledPin3, OUTPUT);**

**pinMode(ledPin4, OUTPUT);**

**}**

**void loop() {**

**// Turn each LED on and off with delays**

**digitalWrite(ledPin1, HIGH);**

**delay(500);**

**digitalWrite(ledPin1, LOW);**

**digitalWrite(ledPin2, HIGH);**

**delay(500);**

**digitalWrite(ledPin2, LOW);**

**digitalWrite(ledPin3, HIGH);**

**delay(500);**

**digitalWrite(ledPin3, LOW);**

**digitalWrite(ledPin4, HIGH);**

**delay(500);**

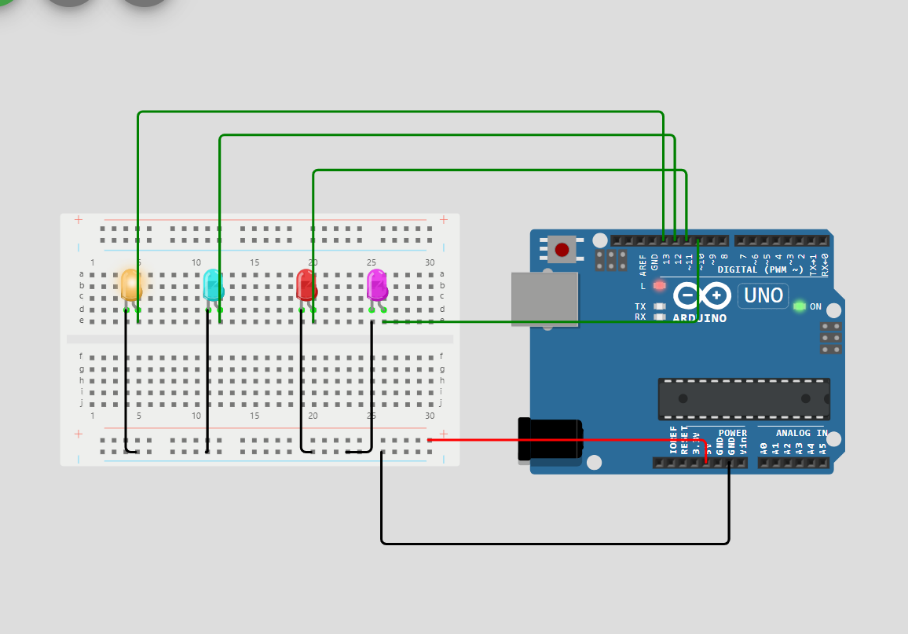
**digitalWrite(ledPin4, LOW);**

**}**

**Run the Simulation**

* **Click the "Start Simulation" button in Wokwi. The LEDs should blink in sequence, each one turning on and off for half a second.**

**Output:**

****

https://wokwi.com/projects/413164738725151745