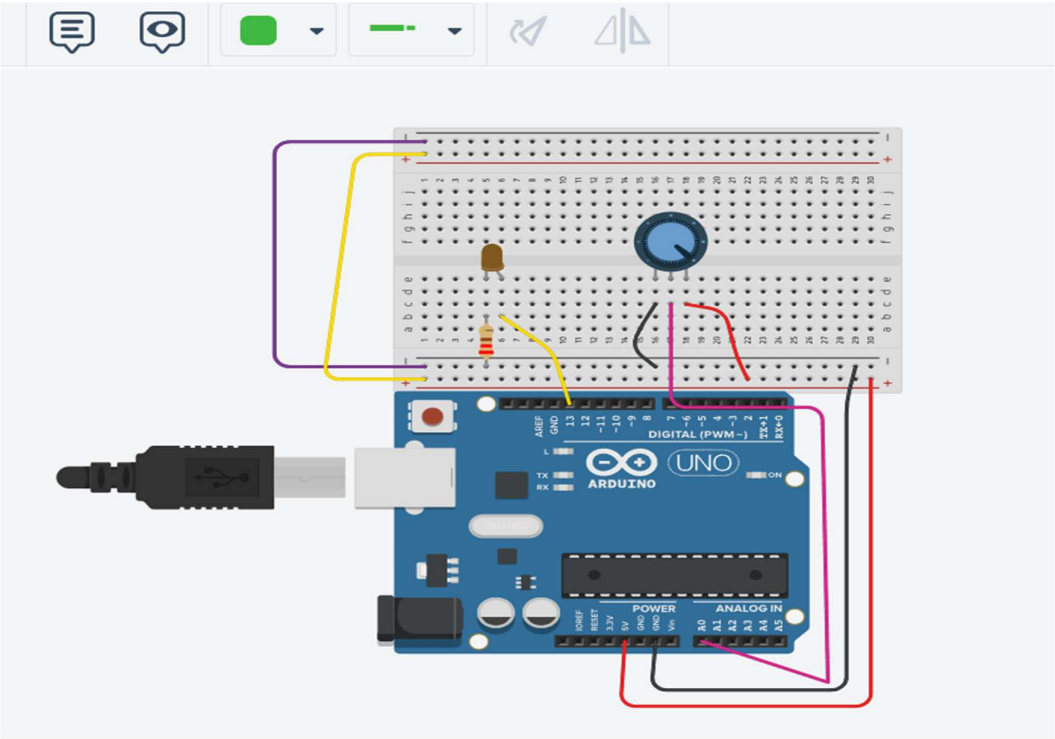
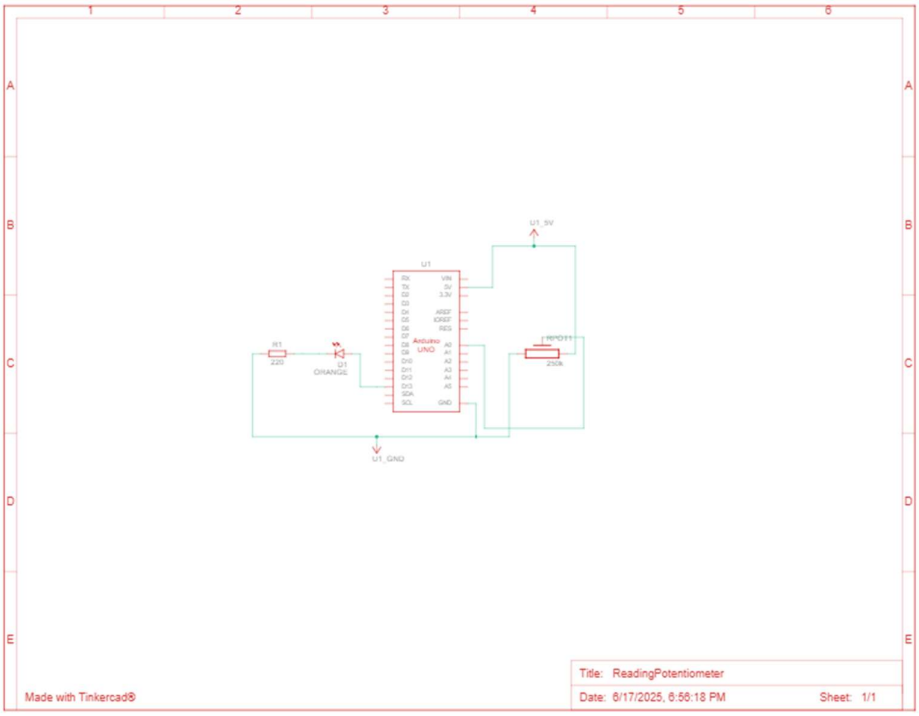
Title: Controlling blinking rate of an LED using Potentiometer.

Introduction: A potentiometer is a variable resistor that outputs an analog voltage depending on its rotation. It is a manually adjustable variable resistor with 3 terminals. Two of the terminals are connected to the opposite ends of a resistive element, and the third terminal connects to a sliding contact, called a wiper, moving over the resistive element. The potentiometer essentially functions as a variable resistance divider. The resistive element can be seen as two resistors in series (the total potentiometer resistance), where the wiper position determines the resistance ratio of the first resistor to the second resistor. If a reference voltage is applied across the end terminals, the position of the wiper determines the output voltage of the potentiometer.





Procedure:

1.Connected One outer pin of the Potentiometer to **5V** on Arduino,

2.Connected the Other outer pin to **GND.**

3.Connected Middle pin to **A0** on Arduino (analog input).

4.Connected the cathode of the LED to GND.
5. Added a 220Ω resistor to the anode.

6.Connected the LED resistor to Arduino pin 13.

7.Connected power rails to Arduino.

Code:

```
int sensorValue = 0;
void setup() {
  pinMode(A0, INPUT);
  pinMode(13, OUTPUT);
}
void loop() {
  sensorValue = analogRead(A0);
  digitalWrite(13, HIGH);
  delay(sensorValue);
  digitalWrite(13, LOW);
  delay(sensorValue);
}
```

Discussion: This project demonstrates how analog input from a sensor (in this case, a potentiometer) can control digital output behavior. The analogRead() function reads values between 0 and 1023, which is then directly used as a delay time (in milliseconds). This delay determines how long the LED stays on and off, thus controlling its blink rate.

When the potentiometer is turned to a low resistance (closer to 0), the delay is shorter, and the LED blinks faster.

When turned to higher resistance (closer to 1023), the delay increases, and the LED blinks slower.

Title: Temperature Monitoring System Using TMP36 and RGB LEDs

---

1. Introduction

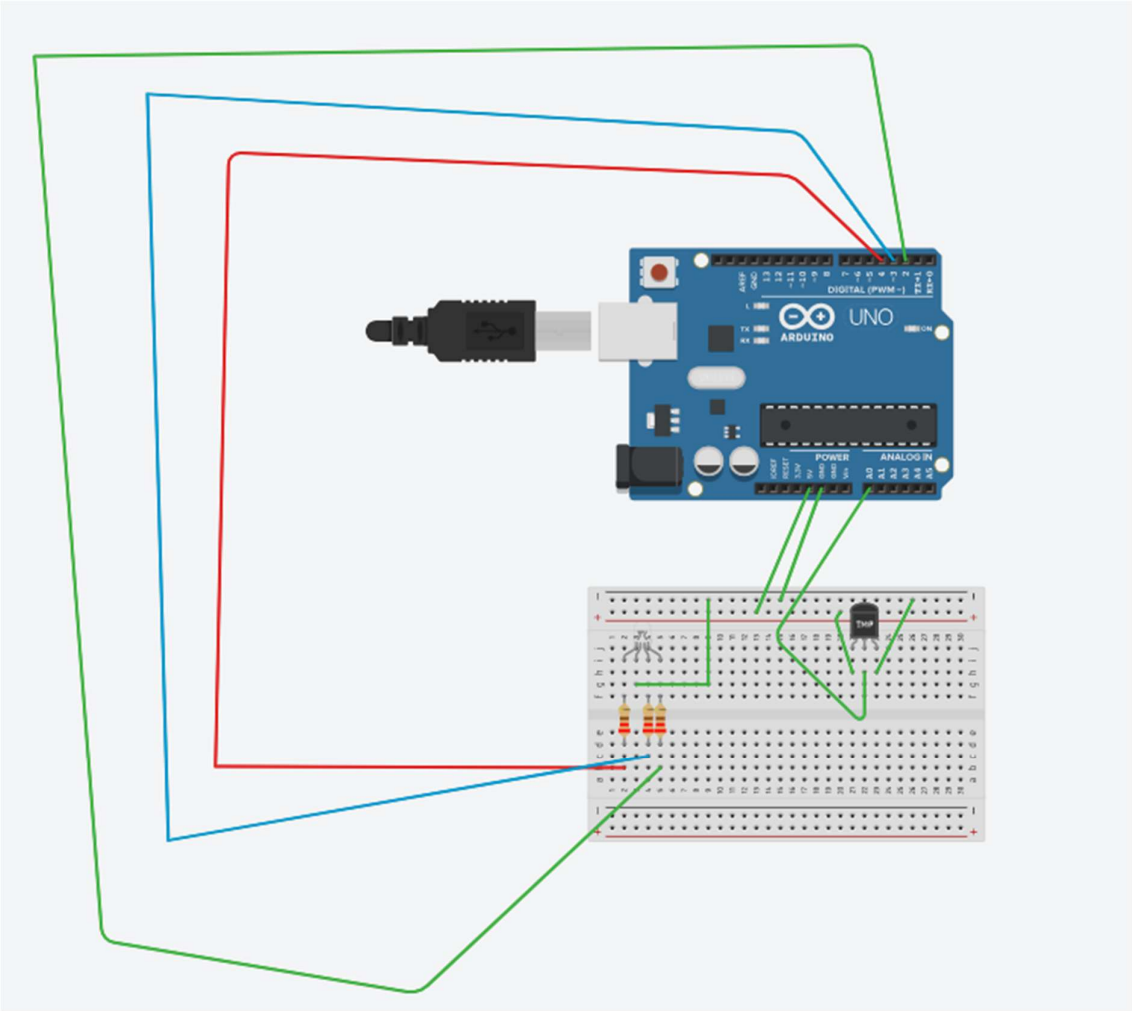
The TMP36 sensor outputs a voltage linearly proportional to the ambient temperature in degrees Celsius. This analog signal is read by the Arduino's analog pin and interpreted to classify the temperature into three distinct zones:

- **Cold** (Blue LED)
- **Normal** (Green LED)
- **Hot** (Red LED)

2. Components Required

- Arduino Uno
- TMP36 Temperature Sensor
- RGB LED (or separate Red, Green, Blue LEDs)
- 3 × 220Ω Resistors
- Breadboard and Jumper Wires
- USB Cable for Arduino

3. Circuit Diagram



4. Procedure

1. **Sensor Connections:**
    - TMP36 **Vout** → Arduino **A0**
    - TMP36 **VCC** → Arduino **5V**
    - TMP36 **GND** → Arduino **GND**
2. **LED Connections:**
    - **Red LED anode** → Arduino **Pin 4** via 220Ω resistor
    - **Green LED anode** → Arduino **Pin 2** via 220Ω resistor
    - **Blue LED anode** → Arduino **Pin 3** via 220Ω resistor

- All **cathodes → GND**

3. **Software Initialization:**
   - Set A0 as **analog input**
   - Set digital pins 2, 3, 4 as **outputs**
   - Use analogRead() to read the TMP36 sensor value
   - Define thresholds based on experimental readings:
     - **val < 139 → Cold** → Blue LED
     - **139 ≤ val ≤ 147 → Normal** → Green LED
     - **val > 147 → Hot** → Red LED
4. **Serial Monitoring:**
   - Initialized Serial.begin(9600) to observe real-time temperature values
   - Added delay(1000) to ensure stable LED transitions and avoid flickering

5. Arduino Code

```
const int BLED = 2;  // Blue LED connected to digital pin 2
const int GLED = 3;  // Green LED connected to digital pin 3
const int RLED = 4;  // Red LED connected to digital pin 4
const int TEMP = A0; // Temperature sensor connected to analog pin A0

const int LOWER_BOUND = 139; // Lower Threshold
const int UPPER_BOUND = 147; // Upper Threshold
int val = 0; // Variable to hold analog reading

void setup() {
  pinMode(TEMP, INPUT);
```

```
  pinMode(BLED, OUTPUT); // Set Blue LED as output
  pinMode(GLED, OUTPUT); // Set Green LED as output
  pinMode(RLED, OUTPUT); // Set Red LED as output
  Serial.begin(9600);
}

void loop() {
  val = analogRead(TEMP);
  Serial.println(val);

  if (val < LOWER_BOUND) { // Temperature is low - show Blue
    digitalWrite(RLED, LOW);   // Turn off Red (Common Anode)
    digitalWrite(GLED, LOW);   // Turn off Green
    digitalWrite(BLED, HIGH);    // Turn on Blue
  }
  else if (val > UPPER_BOUND) { // Temperature is high - show Red
    digitalWrite(RLED, HIGH);    // Turn on Red
    digitalWrite(GLED, LOW);   // Turn off Green
    digitalWrite(BLED, LOW);   // Turn off Blue
  }
  else { // Temperature is normal - show Green
    digitalWrite(RLED, LOW);   // Turn off Red
    digitalWrite(GLED, HIGH);    // Turn on Green
    digitalWrite(BLED, LOW);   // Turn off Blue
  }
 delay(500); // Optional: add a small delay for stability
}
```

6. Conclusion

This lab provided a hands-on experience in building an embedded system that reads, interprets, and reacts to environmental data. The integration of the TMP36 sensor with RGB LEDs showcased the real-world application of sensors, conditional programming, and user feedback mechanisms. This foundational exercise builds a strong base for more complex projects involving automation, smart devices, or IoT systems.

Title: LDR Light Sensor Project

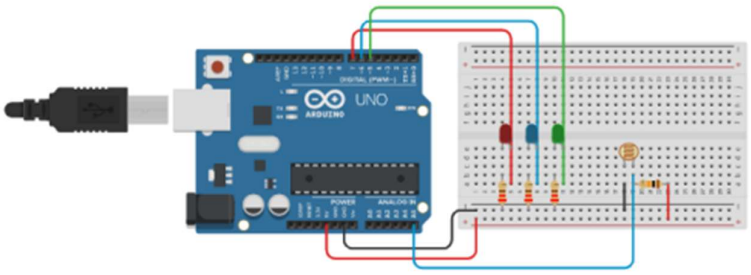
Introduction:

This project demonstrates a simple yet effective smart street lighting system using an LDR (Light Dependent Resistor) and an Arduino. The system is designed to automatically adjust the brightness of an LED based on surrounding light levels.

The LDR is a light-sensitive sensor whose resistance changes depending on the amount of ambient light—resistance decreases in bright light and increases in darkness. The Arduino reads these changes and uses Pulse Width Modulation (PWM) to control the LED's brightness. In bright conditions, the LED dims or turns off; in low light, it brightens automatically.

This method mimics real-world smart streetlights, improving energy efficiency by ensuring lights are only fully active when needed.

Circuit diagram:



Procedure:

One terminal of the LDR was connected to the 5V pin on the Arduino.

The other terminal of the LDR was connected to analog pin A0.

A 10kΩ resistor was placed between A0 and GND to form a voltage divider, allowing accurate light level readings.

The longer leg (anode) of the LED was connected to digital pin 5 through a 220Ω resistor to limit current.

The shorter leg (cathode) of the LED was connected directly to GND.

The Arduino was powered and connected to the computer using a USB cable, allowing both power supply and code uploading.

In the code, A0 was set as an input to read the LDR values.

Pin 5 was defined as a PWM output to control the LED brightness.

The analogRead(A0) function was used to get real-time light intensity data from the LDR.

These raw readings (typically between 54 and 974) were mapped to a PWM range of 255 to 0, so that brighter light results in a dimmer LED and vice versa.

The analogWrite(5, brightness) function was used to apply the calculated brightness to the LED.

Serial.print() was included to display LDR readings in the Serial Monitor for debugging or analysis.

A short 500 ms delay was added in the loop to ensure smooth brightness changes and avoid flickering.

Code:

```
#include<Arduino.h>

const int LDR = A0;

const int LED = 5;


void setup() {
  pinMode(LED, OUTPUT);
  Serial.begin(9600);
}


void loop() {
  int ldrValue = analogRead(LDR);
  int brightness = map(ldrValue, 54, 974, 255, 0);
```

```
  brightness = constrain(brightness, 0, 255);
  analogWrite(LED, brightness);
  Serial.print("LDR: ");
  Serial.print(ldrValue);
  Serial.print(" | LED Brightness: ");
  Serial.println(brightness);
  delay(500);
}
```

Discussion:

This system worked by taking advantage of a key property of the LDR: its resistance goes up when the surrounding light goes down. Because of that, the voltage read by the Arduino on the analog pin dropped in darker conditions. That lower voltage was then converted (or *mapped*) into a higher PWM value, which made the LED glow brighter in the dark and dimmer in bright light. This inverse relationship helped the LED respond smoothly and proportionally to changes in light.

The Serial Monitor displayed real-time readings, which showed that the sensor was working accurately and the system was reacting just as expected. Adding a short 500 ms delay between each reading helped keep the LED transitions smooth, preventing any flickering or jittery behavior.

In the end, this project showed that even with basic components, it's possible to create a low-cost and reliable light-sensitive system. It's a simple example, but the concept can be scaled up for practical use—like in smart streetlights or other automated lighting systems that adjust to their surroundings.