Practical 1:

Aim: Introduction to Arduino circuits and breadboarding. Blinking of LEDs

Hardware requirements: Arduino UNO R3, Breadboard, Resistor, LED.

Code:

void setup()

{

pinMode(LED\_BUILTIN, OUTPUT);

digitalWrite(LED\_BUILTIN, HIGH);

}

void loop()

{

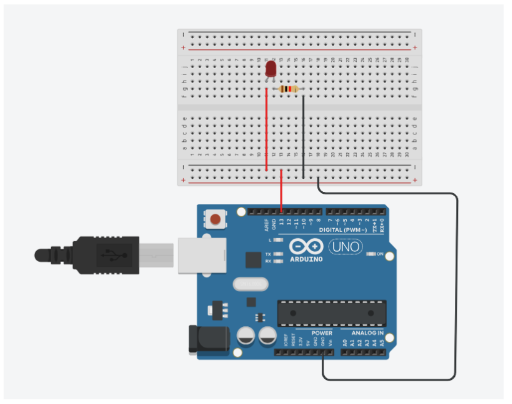
digitalWrite(LED\_BUILTIN, HIGH);

delay(5000); // Wait for 1000 millisecond(s)

digitalWrite(LED\_BUILTIN, LOW);

delay(5000); // Wait for 1000 millisecond(s)

}



```cpp

void setup() {

pinMode(LED\_BUILTIN, OUTPUT);

digitalWrite(LED\_BUILTIN, HIGH);

}

```

In the `setup()` function:

- `pinMode(LED\_BUILTIN, OUTPUT);`: This line configures the pin connected to the built-in LED on the Arduino UNO board (usually pin 13) as an output pin. The `LED\_BUILTIN` constant refers to the pin number associated with the built-in LED on the Arduino board.

- `digitalWrite(LED\_BUILTIN, HIGH);`: This line sets the initial state of the built-in LED to HIGH, which turns the LED on. When the pin is set to HIGH, it provides 5 volts to the LED, causing it to illuminate.

```cpp

void loop() {

digitalWrite(LED\_BUILTIN, HIGH);

delay(5000); // Wait for 5000 milliseconds (5 seconds)

digitalWrite(LED\_BUILTIN, LOW);

delay(5000); // Wait for 5000 milliseconds (5 seconds)

}

```

In the `loop()` function:

- `digitalWrite(LED\_BUILTIN, HIGH);`: This line turns the built-in LED on by setting its pin to HIGH.

- `delay(5000);`: This line causes the Arduino to wait for 5000 milliseconds (5 seconds) before proceeding to the next line of code. During this time, the LED remains on.

- `digitalWrite(LED\_BUILTIN, LOW);`: This line turns the built-in LED off by setting its pin to LOW.

- `delay(5000);`: This line causes the Arduino to wait for another 5000 milliseconds (5 seconds) before looping back to the beginning of the `loop()` function. During this time, the LED remains off.

Now, let's discuss how the circuit is designed:

1. \*\*Connect the LED\*\*: Insert the LED into the breadboard. Connect the anode (longer leg) of the LED to pin 13 on the Arduino UNO board. Connect the cathode (shorter leg) of the LED to the ground (GND) rail on the breadboard.

2. \*\*Insert the Resistor\*\*: Insert a resistor into the breadboard. Connect one leg of the resistor to the cathode of the LED (same row). Connect the other leg of the resistor to the ground (GND) rail on the breadboard.

3. \*\*Connect Arduino UNO\*\*: Insert the Arduino UNO board onto the breadboard. Make sure that pin 13 on the Arduino UNO board is connected to the anode of the LED.

4. \*\*Power Supply\*\*: Connect the Arduino UNO board to your computer using a USB cable. This will provide power to the Arduino board, which in turn powers the LED.

With the circuit connected as described above and the provided code uploaded to the Arduino UNO board, the LED should blink on and off in a 5-second interval.

Practical 2:

Aim: Program using Light Sensitive Sensors

Hardware requirements: Arduino UNO R3, Photoresistor,Resistor, LED.

Code:

int sensorValue=0;

void setup()

{

pinMode(A0, INPUT);

pinMode(9, OUTPUT);

Serial.begin(9600);

}

void loop()

{

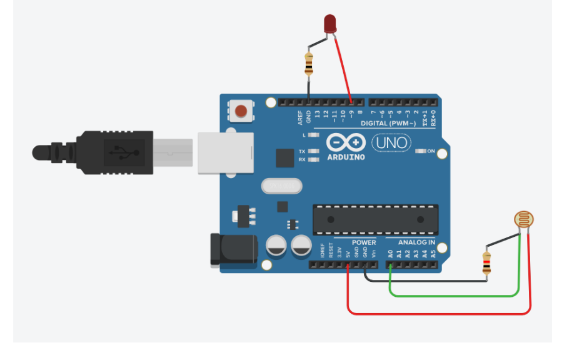
sensorValue= analogRead(A0);

Serial.println(sensorValue);

analogWrite(9,map(sensorValue,0,1023,255,0));

delay(100); // Wait for 100 millisecond(s)

}



Great! Let's break down the provided code and then discuss how the circuit is designed.

```cpp

int sensorValue = 0;

void setup() {

pinMode(A0, INPUT);

pinMode(9, OUTPUT);

Serial.begin(9600);

}

```

In the `setup()` function:

- `pinMode(A0, INPUT);`: This line configures analog pin A0 as an input. This is where the photoresistor is connected. It will be used to measure the amount of light falling on it.

- `pinMode(9, OUTPUT);`: This line configures digital pin 9 as an output. This is where the LED will be connected. The Arduino will control the brightness of the LED based on the light intensity detected by the photoresistor.

- `Serial.begin(9600);`: This line initializes serial communication with a baud rate of 9600. This allows us to print sensor values to the serial monitor for debugging and monitoring purposes.

```cpp

void loop() {

sensorValue = analogRead(A0);

Serial.println(sensorValue);

analogWrite(9, map(sensorValue, 0, 1023, 255, 0));

delay(100); // Wait for 100 millisecond(s)

}

```

In the `loop()` function:

- `sensorValue = analogRead(A0);`: This line reads the analog value from pin A0, which corresponds to the voltage across the photoresistor. The value read represents the intensity of light falling on the photoresistor.

- `Serial.println(sensorValue);`: This line prints the sensor value to the serial monitor. This allows us to observe the light intensity in real-time.

- `analogWrite(9, map(sensorValue, 0, 1023, 255, 0));`: This line maps the sensor value (ranging from 0 to 1023) to the PWM (Pulse Width Modulation) range (ranging from 0 to 255). It then uses the mapped value to adjust the brightness of the LED connected to pin 9. Higher sensor values (indicating more light) will result in a brighter LED, and lower sensor values (indicating less light) will result in a dimmer LED.

- `delay(100);`: This line introduces a delay of 100 milliseconds between consecutive readings. This helps to stabilize the readings and prevents rapid fluctuations in the LED brightness.

Now, let's discuss how the circuit is designed:

1. \*\*Connect the Photoresistor\*\*: Insert the photoresistor into the breadboard. Connect one leg of the photoresistor to analog pin A0 on the Arduino UNO board. Connect the other leg of the photoresistor to the 5V rail on the breadboard.

2. \*\*Insert the Resistor\*\*: Insert a resistor into the breadboard. Connect one leg of the resistor to analog pin A0 (same row as the photoresistor). Connect the other leg of the resistor to the ground (GND) rail on the breadboard.

3. \*\*Connect the LED\*\*: Insert the LED into the breadboard. Connect the anode (longer leg) of the LED to digital pin 9 on the Arduino UNO board. Connect the cathode (shorter leg) of the LED to a current-limiting resistor (to prevent burning the LED) and then to the ground (GND) rail on the breadboard.

4. \*\*Power Supply\*\*: Connect the Arduino UNO board to your computer using a USB cable. This will provide power to the Arduino board, which in turn powers the photoresistor and the LED.

With the circuit connected as described above and the provided code uploaded to the Arduino UNO board, the LED brightness should change based on the intensity of light detected by the photoresistor. You can monitor the sensor values in the serial monitor to observe this behavior.

Practical no. 4

Aim: Programs using humidity sensors

Hardware requirements: Arduino UNO R3, Potentiometer.

Code:

const int analogIn=A2;

int humiditySensorOutput=0;

void setup()

{

Serial.begin(9600);

}

void loop()

{

humiditySensorOutput=analogRead(analogIn);

int humidityPercentage =map(humiditySensorOutput, 0,1023,10,70);

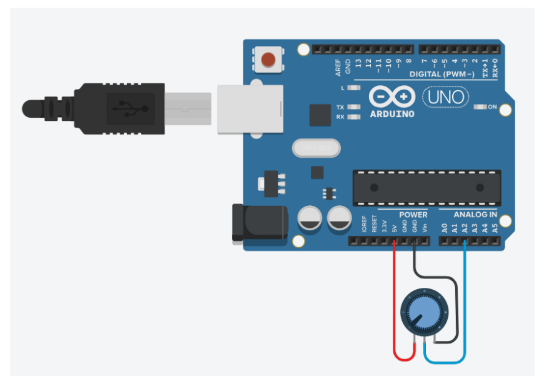
Serial.print("Humity:");

Serial.print(humidityPercentage);

Serial.println("%");

delay(5000);

}



Sure! Let's go through the provided code and then discuss how the circuit can be designed.

```cpp

const int analogIn = A2;

int humiditySensorOutput = 0;

void setup() {

Serial.begin(9600);

}

void loop() {

humiditySensorOutput = analogRead(analogIn);

int humidityPercentage = map(humiditySensorOutput, 0, 1023, 10, 70);

Serial.print("Humidity: ");

Serial.print(humidityPercentage);

Serial.println("%");

delay(5000);

}

```

In this code:

- `const int analogIn = A2;`: This line defines the analog pin A2 as the input pin where the humidity sensor is connected.

- `int humiditySensorOutput = 0;`: This variable will store the raw sensor reading.

- `void setup()`: This is the setup function where initialization tasks are performed.

- `Serial.begin(9600);`: This line initializes serial communication with a baud rate of 9600 bits per second. This is necessary to print data to the serial monitor for debugging purposes.

- `void loop()`: This is the loop function where the main code execution takes place.

- `humiditySensorOutput = analogRead(analogIn);`: This line reads the analog input from the humidity sensor connected to pin A2 and stores the raw sensor value in the variable `humiditySensorOutput`.

- `int humidityPercentage = map(humiditySensorOutput, 0, 1023, 10, 70);`: This line maps the raw sensor reading (ranging from 0 to 1023) to a humidity percentage value (ranging from 10% to 70%). The `map()` function scales the sensor reading linearly to match the desired range.

- `Serial.print("Humidity: ");`: This line prints the label "Humidity: " to the serial monitor.

- `Serial.print(humidityPercentage);`: This line prints the calculated humidity percentage value to the serial monitor.

- `Serial.println("%");`: This line prints the "%" symbol to the serial monitor on a new line.

- `delay(5000);`: This line introduces a delay of 5000 milliseconds (5 seconds) between consecutive readings to slow down the output and avoid flooding the serial monitor with data.

Now, let's discuss how the circuit can be designed:

1. \*\*Connect the Humidity Sensor\*\*: Connect the humidity sensor to analog pin A2 on the Arduino UNO board.

2. \*\*Power Supply\*\*: Ensure the humidity sensor is powered either through the Arduino's 5V pin or using an external power supply if required.

3. \*\*Serial Communication\*\*: Connect the Arduino UNO board to your computer using a USB cable. This allows you to view the humidity readings on the serial monitor.

With the circuit connected as described above and the provided code uploaded to the Arduino UNO board, you should be able to monitor the humidity readings in the serial monitor, displaying the humidity percentage every 5 seconds. Adjustments can be made to the code or circuit as necessary for your specific setup.

Practical no. 6

Aim: Programs using digital infrared motion sensors.

Hardware requirements: Arduino UNO R3, Resistor, LED, PIR sensor.

Code:

int sensorState = 0;

void setup()

{

pinMode (2, INPUT);

pinMode (LED\_BUILTIN, OUTPUT);

}

void loop()

{

sensorState = digitalRead(2);

if (sensorState == HIGH)

{

digitalWrite(LED\_BUILTIN, HIGH);

}

else

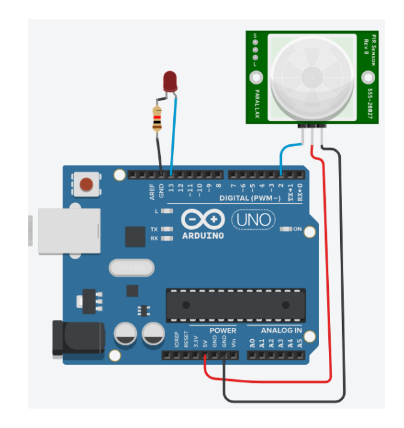
{

digitalWrite(LED\_BUILTIN, LOW);

}

delay(10);

}



Ans:

Sure! Let's review the provided code and then discuss how the circuit can be set up.

```cpp

int sensorState = 0;

void setup() {

pinMode(2, INPUT);

pinMode(LED\_BUILTIN, OUTPUT);

}

void loop() {

sensorState = digitalRead(2);

if (sensorState == HIGH) {

digitalWrite(LED\_BUILTIN, HIGH);

} else {

digitalWrite(LED\_BUILTIN, LOW);

}

delay(10);

}

```

In this code:

- `int sensorState = 0;`: This variable will store the state of the digital input from the PIR (Passive Infrared) motion sensor.

- `void setup()`: This function initializes the pinMode for the PIR sensor's output pin and the built-in LED pin.

- `pinMode(2, INPUT);`: Configures digital pin 2 as an input, which is connected to the output pin of the PIR motion sensor.

- `pinMode(LED\_BUILTIN, OUTPUT);`: Configures the built-in LED pin as an output.

- `void loop()`: This function continuously reads the state of the PIR sensor's output pin and controls the built-in LED based on the motion detected.

- `sensorState = digitalRead(2);`: Reads the state of the digital input pin 2 connected to the PIR sensor. If motion is detected, the sensorState will be HIGH; otherwise, it will be LOW.

- `if (sensorState == HIGH) { digitalWrite(LED\_BUILTIN, HIGH); }`: If motion is detected (sensorState is HIGH), the built-in LED is turned on.

- `else { digitalWrite(LED\_BUILTIN, LOW); }`: If no motion is detected (sensorState is LOW), the built-in LED is turned off.

- `delay(10);`: Adds a short delay of 10 milliseconds to avoid rapid flickering of the LED and to reduce the processing load on the Arduino.

Now, let's discuss how the circuit can be set up:

1. \*\*Connect the PIR Sensor\*\*: Connect the VCC pin of the PIR sensor to the 5V pin of the Arduino, the GND pin to the GND pin of the Arduino, and the OUT pin to digital pin 2 of the Arduino.

2. \*\*Connect the LED\*\*: Connect one leg of the LED (anode) to digital pin 13 of the Arduino (or the LED\_BUILTIN pin), and connect the other leg of the LED (cathode) to a current-limiting resistor (to prevent burning the LED) and then to the GND pin of the Arduino.

3. \*\*Power Supply\*\*: Ensure the PIR sensor is powered through the Arduino's 5V pin.

4. \*\*Serial Communication\*\*: Optionally, you can connect the Arduino UNO board to your computer using a USB cable to monitor the motion detection in the serial monitor. If you want to do this, you can add `Serial.begin(9600);` in the `setup()` function and use `Serial.println()` to print messages.

With the circuit connected as described above and the provided code uploaded to the Arduino UNO board, the built-in LED should turn on when motion is detected by the PIR sensor, and turn off when no motion is detected. Adjustments can be made to the code or circuit as necessary for your specific setup.

Practical no. 7

Aim: Programs using gas sensors.

Hardware requirements: Arduino UNO R3, Breadboard, Resistor, LED, Gas sensor.

Code:

const int LED\_PIN = A1;

const int SENSOR\_PIN = A0;

const int SMOKE\_THRESHOLD=470;

void setup()

{

Serial.begin(9600);

pinMode (LED\_PIN, OUTPUT);

}

void loop()

{

int sensorValue = analogRead(SENSOR\_PIN);

{

if (sensorValue >= SMOKE\_THRESHOLD)

{

digitalWrite(LED\_PIN, LOW);

Serial.print("Smoke Detected!Sensor Value: ");

Serial.println(sensorValue);

}

else

{

digitalWrite(LED\_PIN, HIGH);

Serial.print("No Smoke. Sensor Value: ");

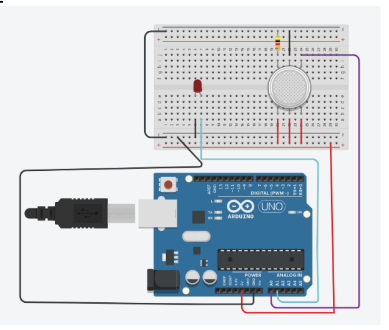
Serial.println(sensorValue);

}

}

delay(1000);

}



Explanation:

Sure, let's go through the provided code and then discuss how the circuit can be set up.

```cpp

const int LED\_PIN = A1;

const int SENSOR\_PIN = A0;

const int SMOKE\_THRESHOLD = 470;

void setup() {

Serial.begin(9600);

pinMode(LED\_PIN, OUTPUT);

}

void loop() {

int sensorValue = analogRead(SENSOR\_PIN);

{

if (sensorValue >= SMOKE\_THRESHOLD) {

digitalWrite(LED\_PIN, LOW);

Serial.print("Smoke Detected! Sensor Value: ");

Serial.println(sensorValue);

} else {

digitalWrite(LED\_PIN, HIGH);

Serial.print("No Smoke. Sensor Value: ");

Serial.println(sensorValue);

}

}

delay(1000);

}

```

In this code:

- `const int LED\_PIN = A1;` and `const int SENSOR\_PIN = A0;`: These lines define the analog pins A1 and A0 as the pins connected to the LED and the gas sensor, respectively.

- `const int SMOKE\_THRESHOLD = 470;`: This variable represents the threshold value above which the gas sensor detects smoke. You may need to adjust this value based on your sensor and environment.

- `void setup()`: This function initializes serial communication and sets the pinMode for the LED pin.

- `Serial.begin(9600);`: Initializes serial communication with a baud rate of 9600 bits per second, allowing us to print data to the serial monitor.

- `pinMode(LED\_PIN, OUTPUT);`: Configures the LED pin as an output.

- `void loop()`: This function continuously reads the analog input from the gas sensor and controls the LED based on the detected smoke level.

- `int sensorValue = analogRead(SENSOR\_PIN);`: Reads the analog input from the gas sensor connected to pin A0 and stores the sensor value in the `sensorValue` variable.

- `if (sensorValue >= SMOKE\_THRESHOLD) { ... } else { ... }`: Checks if the sensor value exceeds the smoke threshold. If the sensor value is greater than or equal to the threshold, it indicates smoke detection, and the LED is turned off. Otherwise, if the sensor value is below the threshold, it indicates no smoke, and the LED is turned on.

- `Serial.print("Smoke Detected! Sensor Value: ");` and `Serial.print("No Smoke. Sensor Value: ");`: These lines print messages to the serial monitor indicating whether smoke is detected or not, along with the current sensor value.

- `delay(1000);`: Adds a delay of 1000 milliseconds (1 second) between consecutive sensor readings to prevent rapid flickering of the LED and to reduce the processing load on the Arduino.

Now, let's discuss how the circuit can be set up:

1. \*\*Connect the Gas Sensor\*\*: Connect the VCC pin of the gas sensor to the 5V pin of the Arduino, the GND pin to the GND pin of the Arduino, and the OUT pin to analog pin A0 of the Arduino.

2. \*\*Connect the LED\*\*: Connect one leg of the LED (anode) to analog pin A1 of the Arduino, and connect the other leg of the LED (cathode) to a current-limiting resistor (to prevent burning the LED) and then to the GND pin of the Arduino.

3. \*\*Power Supply\*\*: Ensure the gas sensor is powered through the Arduino's 5V pin.

4. \*\*Serial Communication\*\*: Optionally, you can connect the Arduino UNO board to your computer using a USB cable to monitor the smoke detection in the serial monitor. If you want to do this, you can add `Serial.begin(9600);` in the `setup()` function and use `Serial.println()` to print messages.

With the circuit connected as described above and the provided code uploaded to the Arduino UNO board, the LED should turn off when smoke is detected by the gas sensor, and turn on when no smoke is detected. Adjustments can be made to the code or circuit as necessary for your specific setup.

Practical no. 8

Aim: Programs using servo motors

Hardware requirements: Arduino UNO R3, Micro Servo Motor.

Code:

#include<Servo.h>

Servo servoBase;

void setup()

{

servoBase.attach(A1);

servoBase.write(0);

}

void loop()

{

for(int i=0;i<=180;i+=20)

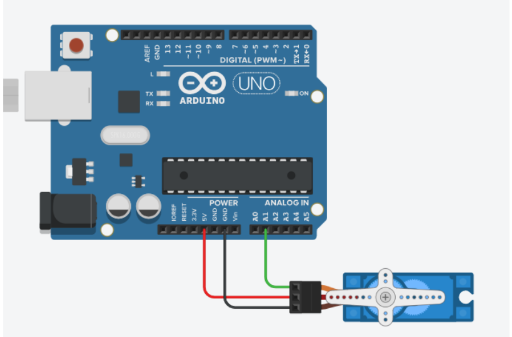
{

servoBase.write(i);

delay(1000);

}

}



Certainly! Let's go through the provided code and then discuss how the circuit can be set up.

```cpp

#include <Servo.h>

Servo servoBase;

void setup() {

servoBase.attach(A1);

servoBase.write(0);

}

void loop() {

for (int i = 0; i <= 180; i += 20) {

servoBase.write(i);

delay(1000);

}

}

```

In this code:

- `#include <Servo.h>`: This line includes the Servo library, which provides functions to control servo motors.

- `Servo servoBase;`: This line declares an instance of the Servo class named `servoBase`, which will be used to control the servo motor.

- `void setup()`: This function is called once when the Arduino starts up or is reset.

- `servoBase.attach(A1);`: This line attaches the servo motor to pin A1 of the Arduino UNO board.

- `servoBase.write(0);`: This line sets the initial position of the servo motor to 0 degrees.

- `void loop()`: This function continuously repeats the code inside its curly braces.

- `for (int i = 0; i <= 180; i += 20) { ... }`: This loop iterates from 0 to 180 degrees with a step of 20 degrees.

- `servoBase.write(i);`: This line sets the position of the servo motor to the current value of `i`, which gradually increases from 0 to 180 degrees.

- `delay(1000);`: This line adds a delay of 1000 milliseconds (1 second) between each position change of the servo motor.

Now, let's discuss how the circuit can be set up:

1. \*\*Connect the Servo Motor\*\*: Connect the signal wire (usually orange or yellow) of the servo motor to pin A1 of the Arduino UNO board. Connect the power wire (usually red) to the 5V pin of the Arduino, and connect the ground wire (usually brown or black) to the GND pin of the Arduino.

2. \*\*Power Supply\*\*: Ensure the servo motor is powered through the Arduino's 5V pin.

3. \*\*Serial Communication\*\*: If you have a serial monitor opened and if you want to monitor the servo movements, you can use the `Serial` functions to print messages.

With the circuit connected as described above and the provided code uploaded to the Arduino UNO board, the servo motor should move gradually from 0 to 180 degrees with a step of 20 degrees, pausing for 1 second between each step. Adjustments can be made to the code or circuit as necessary for your specific setup.