**Lab#1 GIT**

Basics:

1. **What is version control? Why is it important?**

Version control is a system that records changes to files or code over time, allowing you to track and manage different versions of a project. It's important because it helps:

Collaboration: Multiple people can work on the same project simultaneously without overwriting each other's work.

History and tracking: You can see who made what changes, when they were made, and why, which helps with debugging and accountability.

Rollbacks and branching: It allows you to revert to previous versions or create branches for experimentation, reducing the risk of breaking the main project.

Backup and recovery: It provides a safety net by preserving historical versions in case of data loss or mistakes.

Code quality: It promotes best practices by encouraging well-documented, organized, and modular code.

1. **What's the difference between Git and GitHub?**

Git and GitHub are related but serve different purposes:

1. **Git:**

Git is a distributed version control system (VCS) used for tracking changes in source code and other text files.

It runs locally on a user's computer and allows them to create, manage, and track different versions of their projects.

Git is primarily a command-line tool, but there are graphical user interfaces available.

It is not tied to any specific platform or service and can be used independently.

1. **GitHub:**

GitHub is a web-based platform that provides hosting for Git repositories.

It offers a graphical user interface for managing Git repositories and collaborating with others on software development projects.

GitHub provides features like issue tracking, pull requests, code review tools, and project management, making it a hub for team collaboration.

GitHub allows developers to host their Git repositories online and offers additional services for collaboration and project management.

**3) Describe the Git workflow (add, commit, push, pull).**

The Git workflow involves four main commands: add, commit, push, and pull.

Add: Use git add to stage changes or new files for the next commit. Staging is like preparing a snapshot of what you want to save in the next commit.

Commit: After staging changes with git add, use git commit to create a new commit with a message describing the changes. Commits are like snapshots of your project's history.

Push: Once you've committed your changes locally, use git push to upload your commits to a remote repository (like GitHub or GitLab). This makes your changes available for others to see and collaborate on.

Pull: To incorporate changes made by others into your local repository, use git pull. It fetches changes from the remote repository and merges them into your local branch.

**4) What is a repository in the context of Git?**

In the context of Git, a repository (or "repo" for short) is a data structure that stores a collection of files and the entire history of changes made to those files over time. It serves as a central place to manage and track the development of a project. Repositories can be stored locally on a developer's computer or hosted remotely on services like GitHub, GitLab, or Bitbucket, allowing for collaboration and version control in software development.

Commits:

1. **What is a commit in Git?**

In Git, a "commit" is a snapshot of the project's files at a specific point in time. It represents a set of changes made to those files, along with a commit message that describes the purpose of the changes. Commits are the building blocks of a project's version history, allowing you to track and document the evolution of the codebase over time.

1. **How is each commit uniquely identified?**

Each commit in Git is uniquely identified by a long string of characters called a "commit hash" or "SHA-1 hash." This hash is a cryptographic checksum of the commit's content and metadata, ensuring that no two commits have the same identifier. Developers use these hashes to reference and work with specific commits in Git.

Remote Repositories:

1. **What is a remote repository in the context of Git?**

In the context of Git, a "remote repository" is a version control repository that is hosted on a server or online platform (e.g., GitHub, GitLab). It serves as a central location where developers can collaborate, share code, and synchronize their work. Remote repositories allow multiple people to work on the same project, making it easier to track changes and coordinate contributions from different team members.

1. **What are the default names that Git uses for the repository you cloned from and your local repository?**

When you clone a Git repository, the default names used are:

1. Cloned Repository (Remote Repository): The name "origin" is the default alias for the repository you cloned from. You can think of it as the nickname for the remote repository.
2. Local Repository: Your local repository typically doesn't have a specific name. It's just referred to by the directory name where you cloned or initialized it.
3. **How do you synchronize changes from a remote repository to your local one, and vice versa?**

To synchronize changes between a remote repository and your local one in Git:

From Remote to Local (Fetch and Merge):

Use git fetch to retrieve changes from the remote repository without merging them.

Then, use git merge or git pull to incorporate the fetched changes into your local branch.

From Local to Remote (Push):

Make your desired changes in your local repository.

Use git push to send those changes to the remote repository, updating it with your modifications.

GitHub Specifics:

1. **What is a pull request?**

A "pull request" (often abbreviated as "PR") is a feature in platforms like GitHub and GitLab that allows developers to propose changes to a project's codebase. It serves as a formal request to merge a set of commits from one branch (usually created by the contributor) into another branch (usually the main or master branch) of the repository. Pull requests facilitate code review, collaboration, and discussion among team members before changes are merged, ensuring code quality and project integrity.

1. **How do you 'fork' a repository on GitHub, and why might you want to?**

To 'fork' a repository on GitHub:

Go to the repository you want to fork.

Click the "Fork" button in the top right corner of the GitHub page.

GitHub will create a copy of the repository in your own GitHub account.

1. **How can you use GitHub to collaborate on open-source projects?**

To collaborate on open-source projects using GitHub:

**Fork the Repository**: Fork the project's repository to create your own copy.

**Clone Your Fork**: Clone your forked repository to your local machine using git clone.

**Create a Branch**: Create a new branch for your work using git checkout -b branch-name.

**Make Changes**: Make changes and commits to your branch, following best practices.

Push Changes: Push your changes to your forked repository on GitHub using git push.

**Open a Pull Request**: Go to the original project's repository and open a pull request (PR) from your branch to the main branch of the original repository.

**Discuss and Collaborate**: Use the PR for discussion and collaboration. Make necessary changes based on feedback.

**Review and Merge**: The project maintainers review your PR and may request changes or approve it. Once approved, they merge it into the main project.

**Sync Your Fork**: Periodically, sync your fork with the original repository using git fetch upstream and git merge upstream/main.

Collaboration and Best Practices:

1. **Why is it important to write clear commit messages?**

Writing clear commit messages is important because:

Communication: It communicates the purpose and context of a commit to other developers, making it easier to understand the changes.

History Tracking: Clear messages help track the project's history, making it easier to debug issues and review changes over time.

Collaboration: It aids collaboration by enabling team members to understand each other's contributions and intentions.

Documentation: Commit messages serve as documentation for the codebase, helping future developers understand the rationale behind specific changes.

Code Quality: Encourages good coding practices and accountability, as developers are more conscious of their changes when they have to explain them in messages.

1. **When collaborating with others, why might it be important to frequently pull the latest changes?**

Frequently pulling the latest changes when collaborating with others is important because it helps you:

Stay Up-to-Date: Ensures you have the most recent codebase, reducing conflicts and compatibility issues.

Collaborate Effectively: Enables you to work on the latest version of the project, making it easier to integrate your changes and resolve conflicts.

Avoid Divergence: Prevents your local branch from diverging too far from the main project, simplifying the merging process.

Stay Informed: Keeps you informed about what others are working on, facilitating better communication and coordination within the team.

**Lab#2 Arduino**

Theory:

int Led = 2;: Declares a variable Led and assigns the value 2 to it, representing the Arduino pin where the LED is connected.

void setup(): This is the setup function that runs once when the Arduino starts. It initializes the LED pin as an output and starts serial communication at a baud rate of 2000.

void loop(): The main loop function that runs repeatedly. Inside the loop, it does the following:

Prints "Led is ON." to the serial monitor.

Turns the LED on (high).

Waits for 1 second (1000 milliseconds).

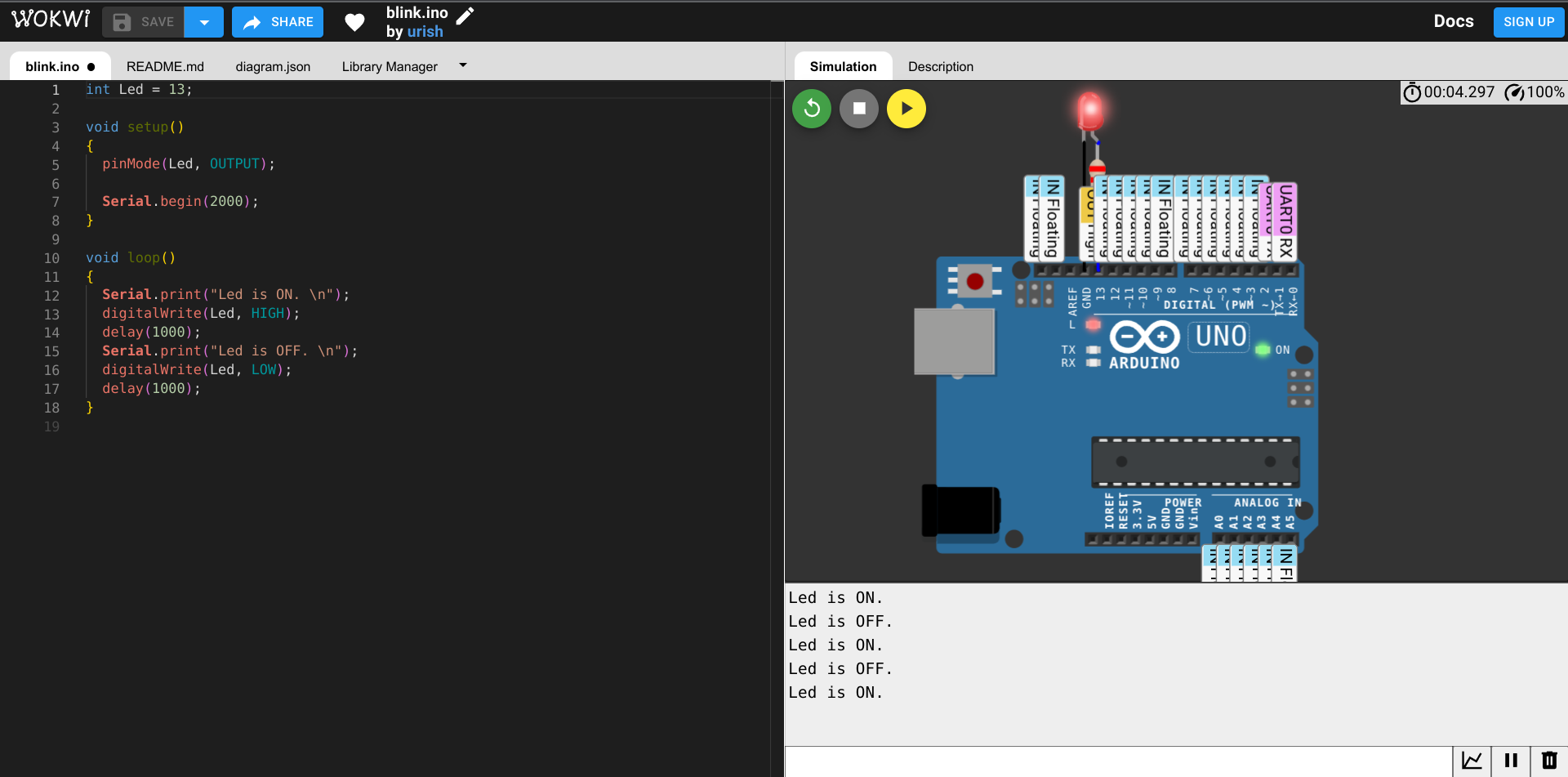
Prints "Led is OFF." to the serial monitor.

Turns the LED off (low).

Waits for another 1 second.

This code creates a simple program that blinks the LED on and off with a 1-second interval while providing serial monitor output to indicate the LED's status.

Code:

int Led = 2;

void setup()

{

pinMode(Led, OUTPUT);

Serial.begin(2000);

}

void loop()

{

Serial.print("Led is ON. \n");

digitalWrite(Led, HIGH);

delay(1000);

Serial.print("Led is OFF. \n");

digitalWrite(Led, LOW);

delay(1000);

}

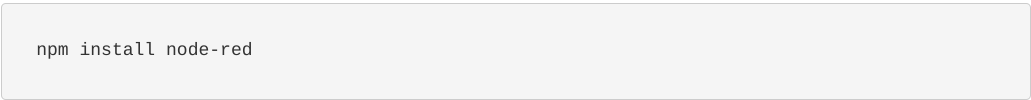
**Lab#3 Downloading Nodejs, node-red and Mosquitto**

To install Node.js, Node-RED, and Mosquitto on a Linux system (e.g., Ubuntu), you can follow these steps:

1. Node.js Installation:



1. Installing Node-red



1. Downloading and installing mosquitto

Visiting this url <https://mosquitto.org/files/binary/linux/>

Downloading preferred .tar.gz file

Extracting .tar.gz file

**Lab #4 Red MQTT Installation and testing**

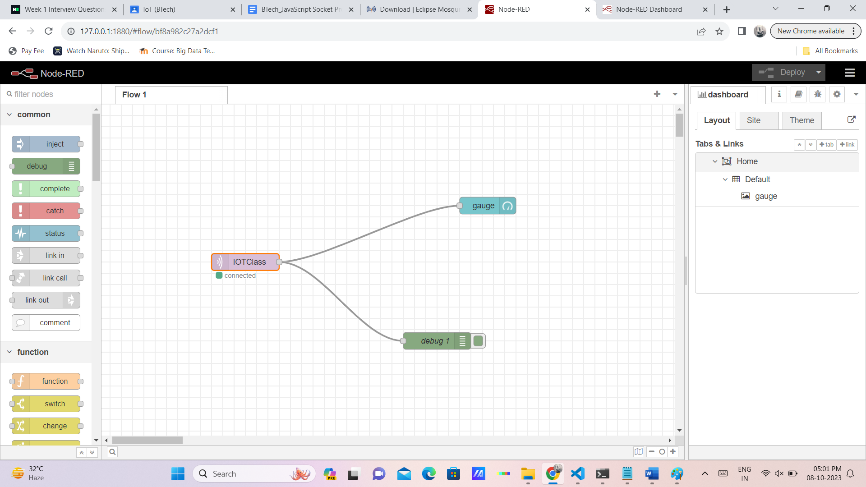
Theory:

RedMQTT is an application designed to read mobile sensor data and forward it to a local or cloud-based MQTT server.

Procedure:

1. Install RedMQTT on a smartphone.
2. Open node red in pc by typing ‘node-red’ in cmd. Also open hivemq broker.
3. Get the broker name, address and port from hivemq.
4. Open node red tab and add mqtt-in, gauge, and debug to the platform. Connect them as shown in the screenshot.
5. Double click on the mqtt-in. Add broker name, address and port. Then, add a topic name and save it.
6. Go to the RedMQTT app and go to settings. Add the same brover information.
7. Then, go to publish on the app. Select the broker name, enter the same topic name and select the sensor.
8. Now publish the node-red flow and the gauge will show the sensor data.

Screenshots:



**#LAB 5**

Aim: Interfacing DHT sensor with Arduino

Theory:

* DHT11 sensor measures and provides humidity and temperature values serially over a single wire.
* It can measure relative humidity in percentage (20 to 90% RH) and temperature in degree Celsius in the range of 0 to 50°C.
* It has 4 pins; one of which is used for data communication in serial form.

Code:

#include "DHT.h"

DHT dht;

void setup()

{

Serial.begin(9600);

Serial.println();

Serial.println("Status\tHumidity (%)\tTemperature (C)\t(F)");

dht.setup(2); /\* set pin for data communication \*/

}

void loop()

{

delay(dht.getMinimumSamplingPeriod()); /\* Delay of amount equal to sampling period \*/

float humidity = dht.getHumidity(); /\* Get humidity value \*/

float temperature = dht.getTemperature(); /\* Get temperature value \*/

Serial.print(dht.getStatusString()); /\* Print status of communication \*/

Serial.print("\t");

Serial.print(humidity, 1);

Serial.print("\t\t");

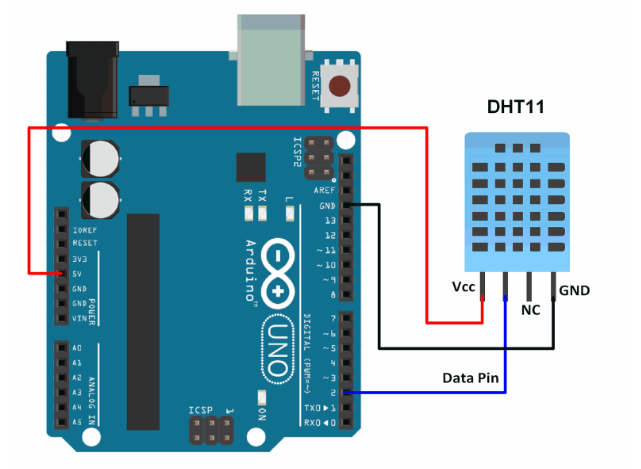
Serial.print(temperature, 1);

Serial.print("\t\t");

Serial.println(dht.toFahrenheit(temperature), 1); /\* Convert temperature to Fahrenheit units \*/

}

Output:



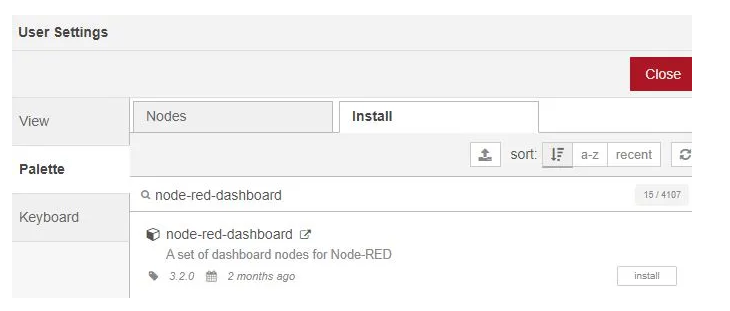
**#LAB 6**

Aim: Gathering DHT sensor data and visualizing it using an app built by Node-RED.

Theory: nil

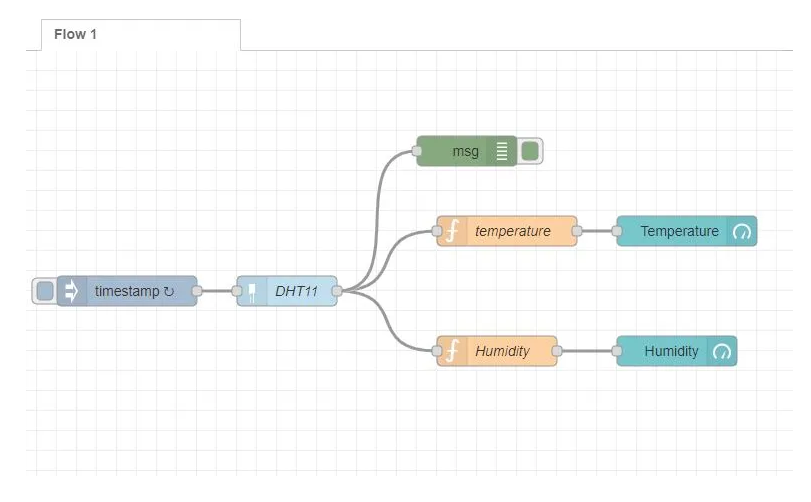
Procedure:

* Open palette and install node red dashboard along with node-red-contrib-dht-sensor

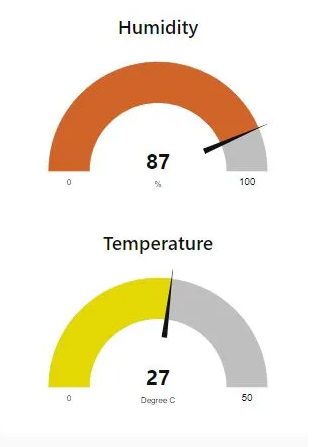


* Create the flow like in the image using the following tags:
  + Inject
  + Function
  + Debug
  + rpi-DHT22
  + Gauge

Set the properties of above tags as said in the following link: https://iotstarters.com/building-node-red-dashboard-with-dht11-sensor/



Output:



**#LAB 7**

Aim: making a led blind only with a switch

Theory: nil

Code:

const int switchPin = 2; // Pin connected to the external switch

const int ledPin = 13;   // Pin connected to the LED

int switchState = HIGH;   // Variable to store the current state of the switch

int lastSwitchState = HIGH; // Variable to store the previous state of the switch

void setup() {

  pinMode(switchPin, INPUT); // Set the switch pin as INPUT

  pinMode(ledPin, OUTPUT);   // Set the LED pin as OUTPUT

}

void loop() {

  // Read the state of the switch

  switchState = digitalRead(switchPin);

  // If the switch state has changed (button released), start blinking the LED

  if (switchState != lastSwitchState && switchState == HIGH) {

    while (switchState == HIGH) {

      digitalWrite(ledPin, HIGH); // Turn the LED on

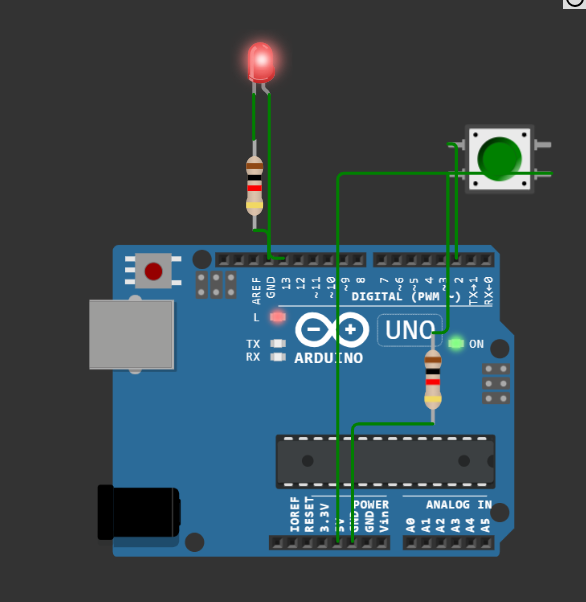
      delay(500);                  // Wait for 500 milliseconds (0.5 seconds)

      digitalWrite(ledPin, LOW);  // Turn the LED off

      delay(500);                  // Wait for another 500 milliseconds

      switchState = digitalRead(switchPin); // Read the state of the switch again

    }

  }

  // Save the current switch state for the next iteration

  lastSwitchState = switchState;

}

**#Lab 8**

Aim: led and dht work alternatively using push button

Theory: nil

Code:

#include <DHT.h>

const int buttonPin = 2;  // Pin connected to the push button

const int ledPin = 13;    // Pin connected to the LED

const int dhtPin = 3;     // Pin connected to the DHT sensor

DHT dht(dhtPin, DHT22); // Create a DHT object

int buttonState = HIGH;  // Variable to store the current state of the button

int lastButtonState = HIGH; // Variable to store the previous state of the button

int currentState = 0; // Variable to keep track of the current state

void setup() {

  pinMode(buttonPin, INPUT);   // Set the button pin as INPUT

  pinMode(ledPin, OUTPUT);     // Set the LED pin as OUTPUT

  pinMode(dhtPin, INPUT);      // Set the DHT sensor pin as INPUT

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

  dht.begin();                 // Initialize the DHT sensor

}

void loop() {

  // Read the state of the button

  buttonState = digitalRead(buttonPin);

  // If the button state has changed (button pressed), toggle the state

  if (buttonState != lastButtonState && buttonState == LOW) {

    currentState = 1 - currentState; // Toggle between 0 and 1

    // Perform actions based on the current state

    if (currentState == 0) {

      // State 0: Blink the LED

      blinkEveryTwoSeconds();

    } else {

      // State 1: Print DHT sensor information

      printDHTSensorInfo();

    }

  }

  // Save the current button state for the next iteration

  lastButtonState = buttonState;

}

void blinkEveryTwoSeconds() {

  digitalWrite(ledPin, HIGH); // Turn the LED on

  delay(1000); // Wait for 1 second

  digitalWrite(ledPin, LOW); // Turn the LED off

  delay(1000); // Wait for 1 second

}

void printDHTSensorInfo() {

  // Read temperature and humidity from the DHT sensor

  float temperature = dht.readTemperature();

  float humidity = dht.readHumidity();

  // Print the sensor information to the Serial Monitor

**Serial**.print("Temperature: ");

**Serial**.print(temperature);

**Serial**.print(" °C, Humidity: ");

**Serial**.print(humidity);

**Serial**.println(" %");

  // Add any additional actions based on the sensor data, e.g., controlling an LED

  if (temperature > 25.0) {

    digitalWrite(ledPin, HIGH); // Turn on the LED if the temperature is above 25.0 °C

  } else {

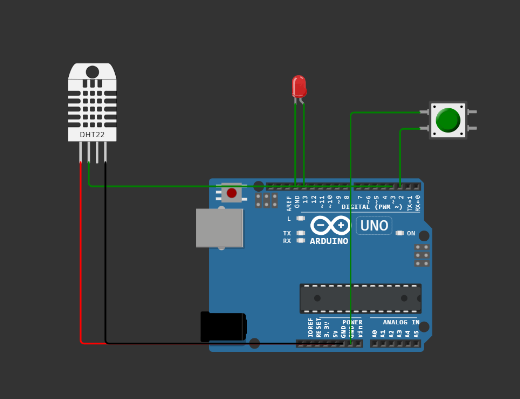
    digitalWrite(ledPin, LOW);  // Turn off the LED otherwise

  }

  delay(2000); // Wait for 2 seconds before printing the next sensor information

}

Output:



**#Lab 9**

Aim: using ultrasonic sensor and getting a reading out of it

Theory: nil

Code:

#define triggerpin 3

#define echopin 2

int dist;

long dur;

void setup() {

pinMode(triggerpin, OUTPUT);

pinMode(echopin, INPUT);

**Serial**.begin(9600);

**Serial**.println("Measurement begins");

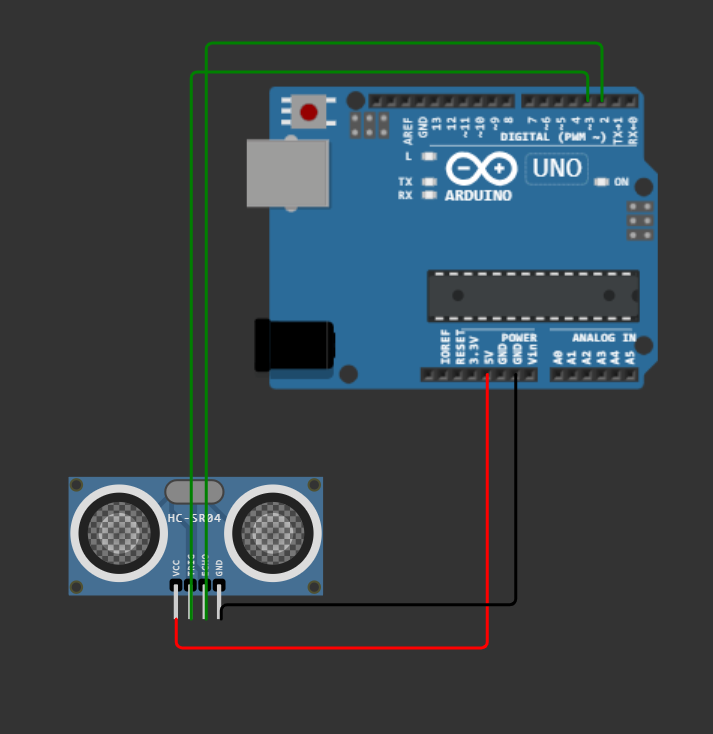
delay(100);

}

void loop() {

  digitalWrite(triggerpin,LOW);

  delayMicroseconds(2);

  digitalWrite(triggerpin, HIGH);

  delayMicroseconds(5);

  digitalWrite(triggerpin,LOW);

  delayMicroseconds(2);

  dur=pulseIn(echopin,HIGH);

  dist=dur\*0.344/2;

**Serial**.print("Distance is: ");

**Serial**.print(dist);

**Serial**.print("cm");

}

Output:

3cm

1432cm

1555cm