ISTC IoT WORKSHOP

Three-Day Workshop Manual

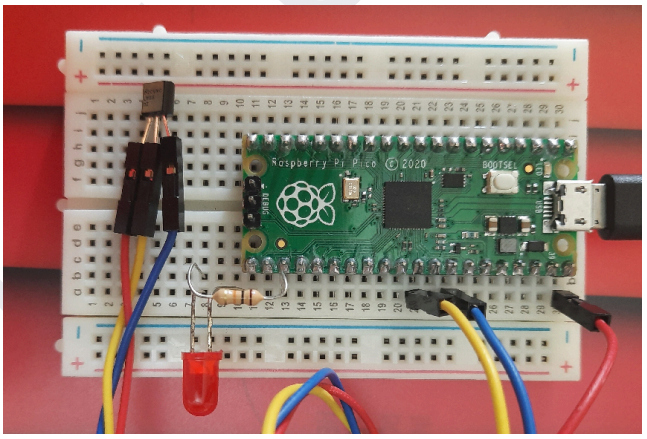
Day 1: Electronics and IoT Basic

Section 1:Microcontroller Basics

Introduction to Microcontrollers:

Imagine a tiny computer, smaller than your phone, that can control the world around it. That's essentially what a microcontroller is! These miniature marvels are the brains behind countless everyday devices, from your smart thermostat to your fitness tracker to even your electric toothbrush.

So, what exactly do microcontrollers do?
Think of them as tiny but mighty managers. They receive information from the environment through sensors, process that information, and then take action by controlling things like LEDs, motors, and other components.

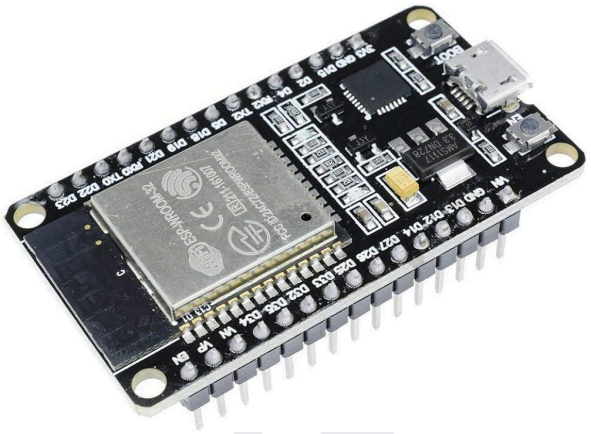


Why are microcontrollers so important for the Internet of Things (IoT)?
Well, they're the key ingredient that allows everyday objects to become "smart" and connected. By embedding microcontrollers into devices, we can collect data, make decisions, and automate tasks, all without human intervention. This opens up a world of possibilities for creating a more connected and efficient world.

In this workshop, we'll be using a powerful microcontroller called the ESP32.
It's like a Swiss Army knife of microcontrollers, packed with features like Wi-Fi, Bluetooth, and plenty of processing power. With the ESP32, we can create sophisticated IoT projects that communicate with the internet and interact with the world around them.

Excited to get started? Let's dive into the world of microcontrollers and unleash your inner inventor!

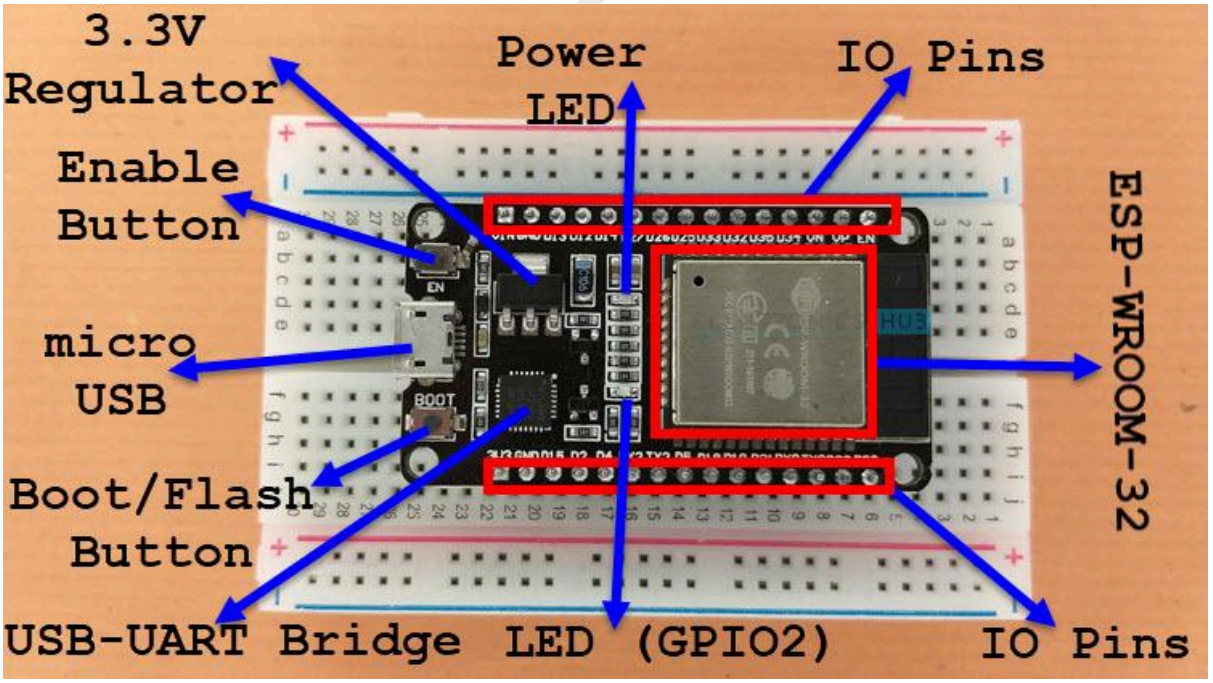
Understanding the ESP32 Board:



Now that you know what microcontrollers are, let's get acquainted with our star player: the ESP32 development board! This little powerhouse is packed with features that will enable us to build amazing IoT projects.
Think of the ESP32 as a mini-computer on a single board. It has its own brain (the **microcontroller** chip), **memory** to store information, and various **input/output** pins that allow it to connect with the **outside world.**

Here's a quick rundown of some of the ESP32's coolest features:

Wi-Fi and Bluetooth connectivity: This means our projects can easily connect to the internet and communicate with other devices wirelessly.
Multiple GPIO pins: These pins act as switches and sensors, allowing us to connect LEDs, buttons, sensors, and other components to interact with the environment.
Analog and digital capabilities: The ESP32 can read both analog and digital signals, making it compatible with a wide range of sensors and actuators.
Powerful processing: The ESP32 is capable of handling complex tasks and calculations.

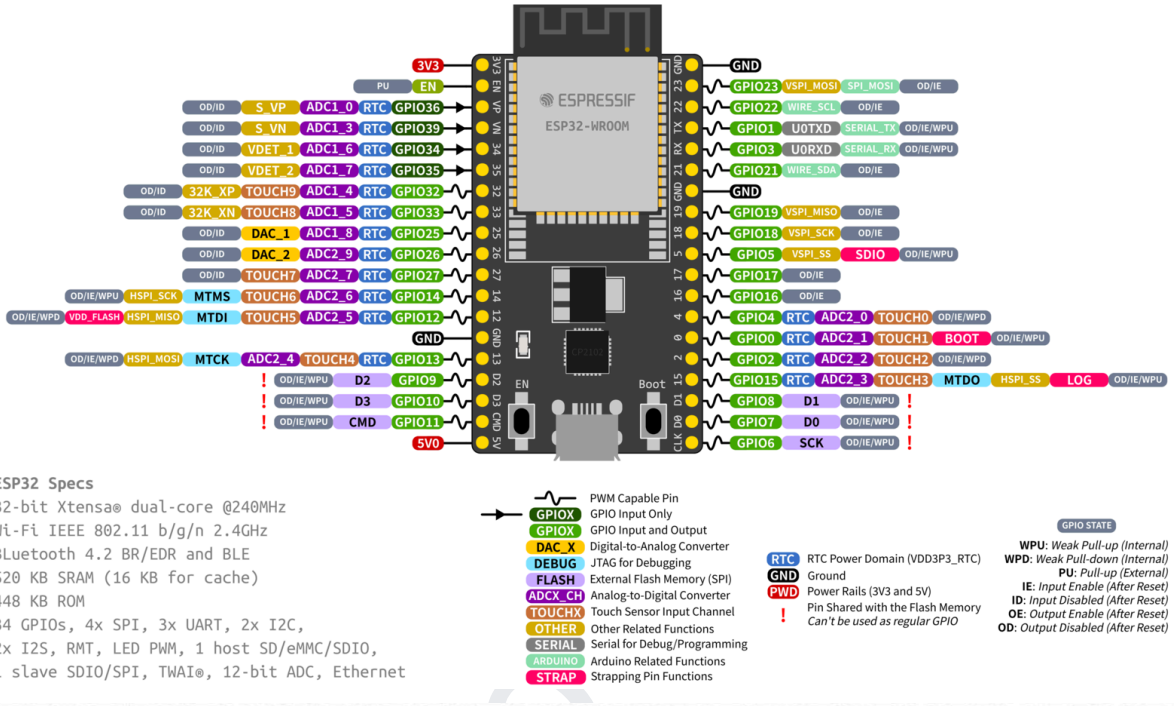


To navigate the ESP32 board, let's explore its key components:

Microcontroller chip: The brains of the operation, responsible for processing information and executing instructions.
USB port: This is how we connect the ESP32 to our computer to upload code and power the board.
Power supply pins: These pins provide power to the board and external components.
GPIO pins: These versatile pins can be used for input or output, depending on our project needs.

Built-in LEDs: These LEDs can be controlled through code and are handy for testing and debugging.

With its impressive features and user-friendly design, the ESP32 is an ideal platform for both beginners and experienced makers to explore the world of IoT. So, get ready to unleash the power of the ESP32 and bring your creative ideas to life!



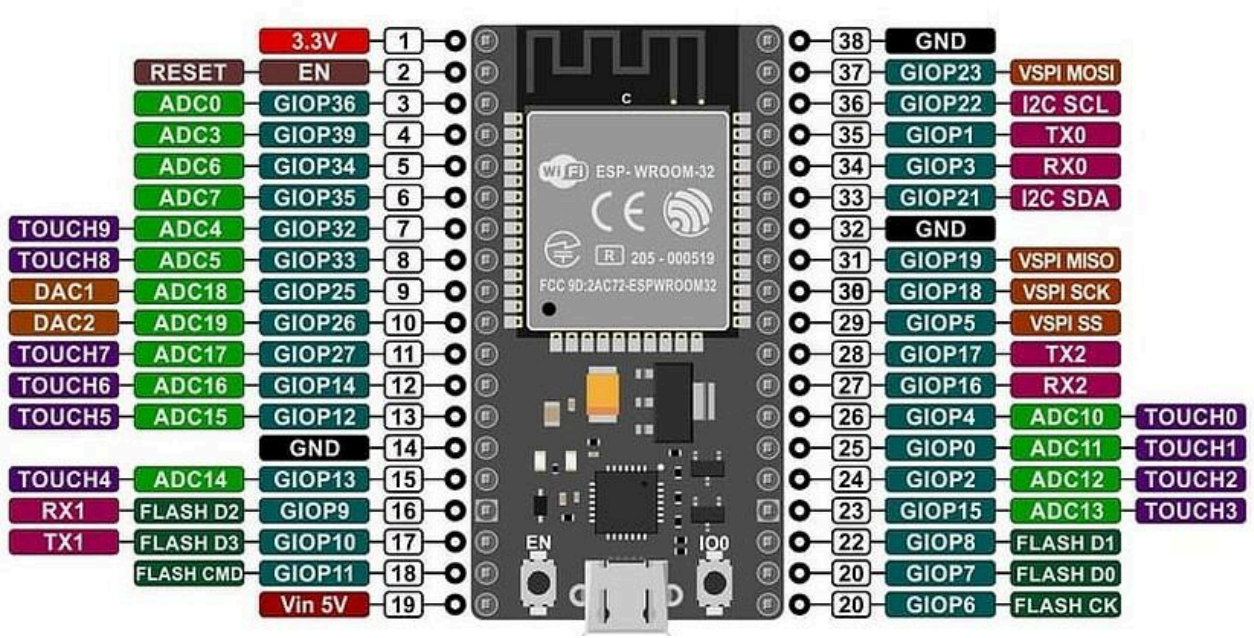
ESP32 Specs

32-bit Xtensa® dual-core @240MHz
Wi-Fi IEEE 802.11 b/g/n 2.4GHz
BLuetooth 4.2 BR/EDR and BLE
520 KB SRAM (16 KB for cache)
448 KB ROM
34 GPIOs, 4x SPI, 3x UART, 2x I2C,
2x I2S, RMT, LED PWM, 1 host SD/eMMC/SDIO,
1 slave SDIO/SPI, TWAI®, 12-bit ADC, Ethernet

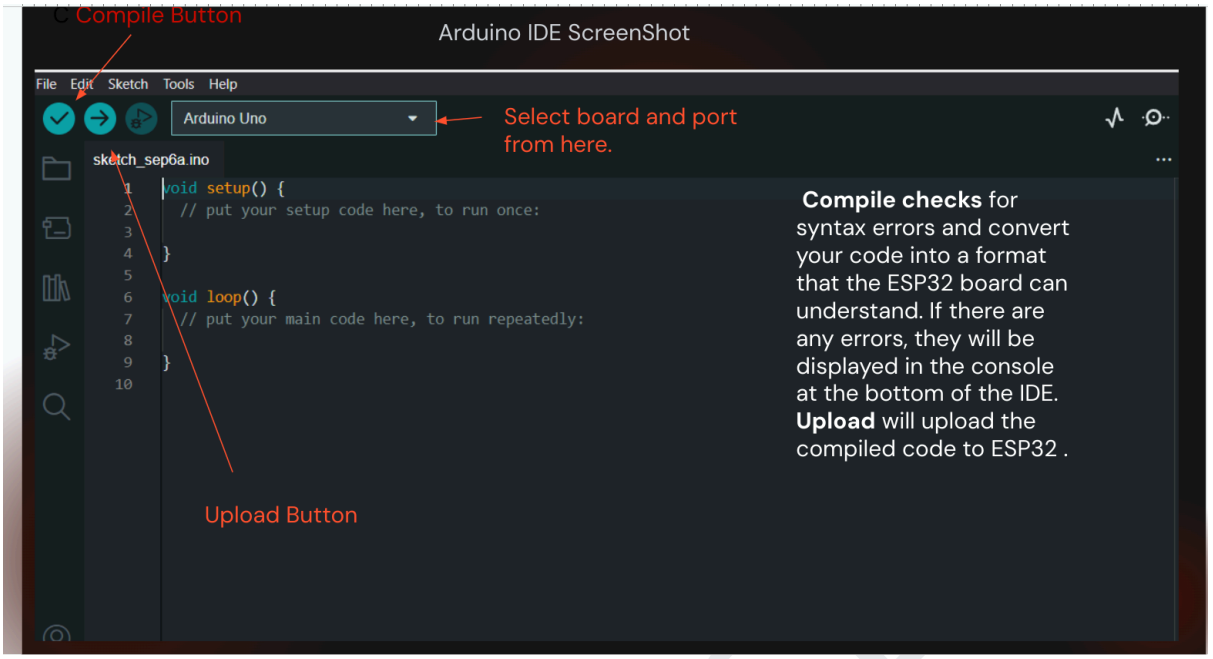
PWM Capable Pin
GPIOX GPIO Input Only
GPIOX GPIO Input and Output
DAC_X Digital-to-Analog Converter
DEBUG JTAG for Debugging
FLASH External Flash Memory (SPI)
ADCX_CH Analog-to-Digital Converter
TOUCHX Touch Sensor Input Channel
OTHER Other Related Functions
SERIAL Serial for Debug/Programming
ARDUINO Arduino Related Functions
STRAP Strapping Pin Functions

RTC RTC Power Domain (VDD3P3_RTC)
GND Ground
PWD Power Rails (3V3 and 5V)
! Pin Shared with the Flash Memory
Can't be used as regular GPIO

GPIO STATE
WPU: *Weak Pull-up (Internal)*
WPD: *Weak Pull-down (Internal)*
PU: *Pull-up (External)*
IE: *Input Enable (After Reset)*
ID: *Input Disabled (After Reset)*
OE: *Output Enable (After Reset)*
OD: *Output Disabled (After Reset)*



Setting Up the Arduino IDE:



Basic Programming Concepts:

What is Programming

Definition: Programming is the process of giving instructions to a computer to perform a specific task.
Analogy: Compare it to giving directions to someone. You need to be clear, and precise, and use a language the person understands.

What is Programming Languages?
Explanation: Programming languages are like human languages, but designed for computers. Each language has its own set of rules and syntax.
Analogy: Compare it to different languages spoken by people around the world. Just like you need to learn a language to communicate with people from a specific region, you need to learn a programming language to communicate with computers.

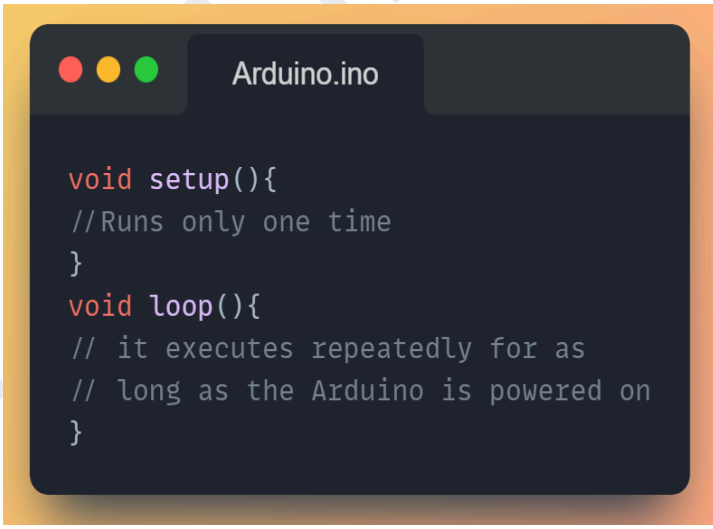
What is Flow Control?
Explanation: Programs have decision-making and repetition structures.
Analogy: If statements are like making decisions (e.g., "If it's raining, take an umbrella"). Loops are like doing repetitive tasks (e.g., "Repeat until the cake is baked").

Why Write Code?
Definition: Code is the set of instructions you write for the computer.
Analogy: Think of it like writing a recipe. You list the ingredients (data), the steps (instructions), and the expected outcome (output).

Variables and Data ?
Definition: Variables are like containers that hold data. Data can be numbers, text, or other types of information.
Analogy: Imagine a box labeled "age" where you can store the age of a person. The box can change its content (value) as people get older.

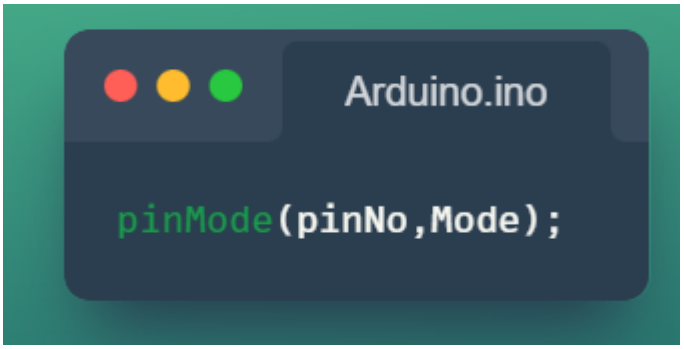
Note: computers can perform millions or even billions of basic operations, like adding or multiplying numbers, in just one second.

Arduino IDE Programming
Void Setup and void Loop



void setup(): This function is executed once when the arduino board is powered on or reset.
It's where you initialize variables, set pin modes, and perform any other setup tasks
void loop(): After setup() runs, the loop() function executes repeatedly for as long as the arduino is powered on.
Whether you're reading sensor data, turning LEDs on and off, or sending data to a computer, it happens here.

A function is a reusable block of code designed to perform a specific task
pinMode() function in Arduino ide

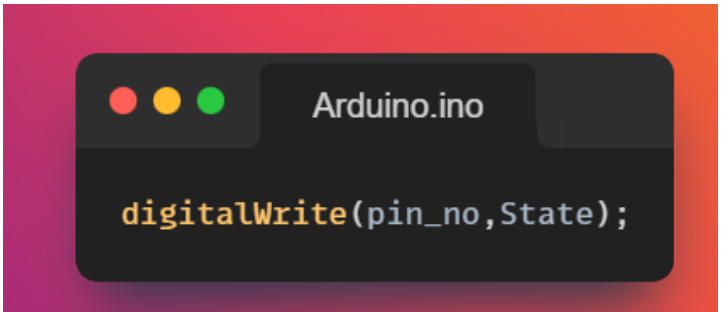


the pinMode() function is used to configure a specific pin as either an input or an output. It's like setting the ground rules for how a particular pin on the Nodemcu will behave.

pinMode(D2, OUTPUT);
Above code set Pin D2 as Output.

The mode can be INPUT, OUTPUT, INPUT_PULLUP.

digitalWrite();

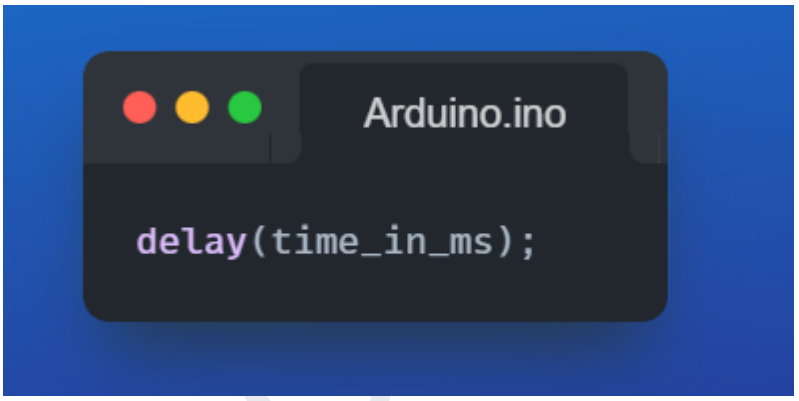


digitalWrite() function is your go-to for setting a digital pin to either HIGH (usually 5V or 3.3V) or LOW (0V). It's like flipping a switch on or off.

digitalWrite(D1, HIGH);
Above set pin D1 as High which means it is turning on pin D1.

digitalWrite(D1, LOW);
Above set pin D1 as LOW which means it is turning on pin D1.

delay();



In Arduino, the delay() function is used to pause the execution of the program for a specified number of milliseconds.
It's like telling your code to take a short break.
This is often used for timing purposes, like blinking an LED on and off.
1 second == 1000 ms

Variable in Arduino ide

In the Arduino IDE, a variable is like a storage box where you can keep data that your program will use.
Just like in algebra where 'x' can hold different values, in Arduino, a variable can hold different types of data like numbers, characters, or even strings of text.

digitalRead(); in Arduino IDE

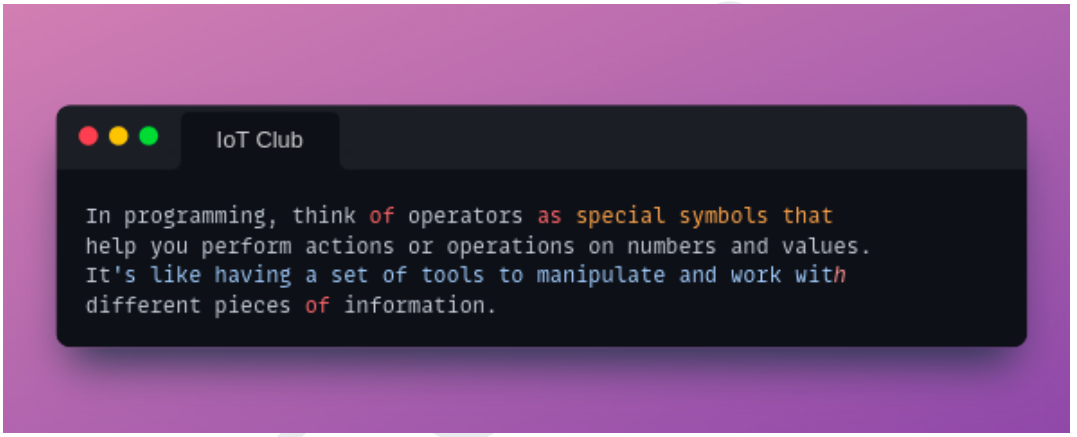
In Arduino, the digitalRead() function is used to read the state of a digital pin, which can be either HIGH or LOW. This is essential when you're dealing with input devices like buttons,

switches, or sensors that have digital outputs.
In this example “value” will store the current state of pin_number

pin_number: The number of the pin you want to read from.
value: The variable that will store the state of the pin, either HIGH or LOW.

Operators in C

Arithmetic Operators:
+ (Addition)
- (Subtraction)
* (Multiplication)
/ (Division)
% (Modulus)
Relational Operators:
== (Equal to)
!= (Not equal to)
< (Less than)
> (Greater than)
<= (Less than or equal to)
>= (Greater than or equal to)




```
gurkirat@gpd1:~/Projects$ gcc test.c -o a.out && ./a.out
Addition: 15
Subtraction: 5
Multiplication: 50
Division: 2
Modulus: 0
```

IoT Club

```c
#include <stdio.h>

int main() {
    int x = 5, y = 10;

    printf("Equal to: %d\n", x == y);
    printf("Not equal to: %d\n", x != y);
    printf("Less than: %d\n", x < y);
    printf("Greater than: %d\n", x > y);
    printf("Less than or equal to: %d\n", x <= y);
    printf("Greater than or equal to: %d\n", x >= y);

    return 0;
}
```

```
gurkirat@gpd1:~/Projects$ gcc test.c -o a.out && ./a.out
Equal to: 0
Not equal to: 1
Less than: 1
Greater than: 0
Less than or equal to: 1
Greater than or equal to: 0
```

Other Operator In C

Logical Operators:
&& (Logical AND)
|| (Logical OR)
! (Logical NOT)

Assignment Operators:
= (Assignment)
+= (Add and assign)
-= (Subtract and assign)
*= (Multiply and assign)
/= (Divide and assign)
%= (Modulus and assign)

Increment and Decrement Operators:
++ (Increment)
-- (Decrement)

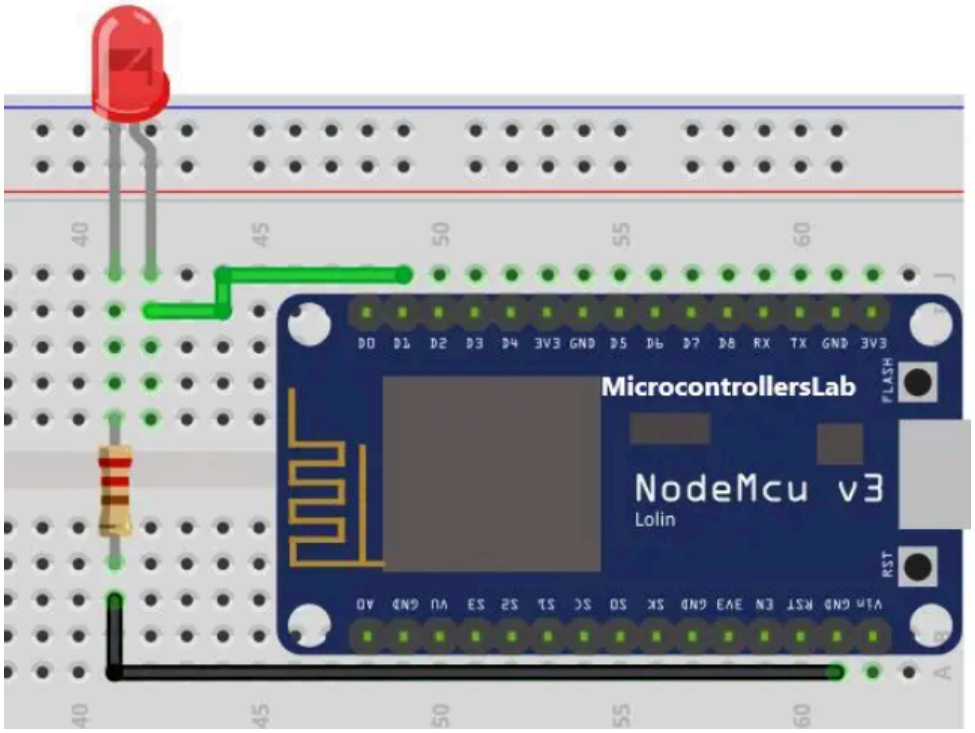
Blinking an LED (Hands-on Activity):

Arduino.ino

```
void setup() {
  pinMode(D1, OUTPUT);  // Set pin D1 as an output pin.
}

void loop() {
  digitalWrite(D1, HIGH);  // Turn the LED on.
  delay(1000);             // Wait for 1 second.
  digitalWrite(D1, LOW);   // Turn the LED off.
  delay(1000);             // Wait for 1 second.
}
```

Instead Of D1,use D22 or 22

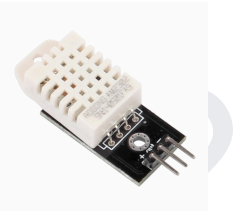


Connect The led similar to above with esp32

Section 2: Sensors and Actuators

Types of Sensors (LDR, DHT22)

LDR (Light Dependent Resistor): It's a sensor that reacts to light. When it's bright, its resistance changes. It's commonly used in applications like streetlights, cameras, and automatic lighting systems.



DHT22: This sensor measures temperature and humidity. it suitable for applications like weather station and environmental monitoring

Reading Sensor Data with ESP32

The ESP32 is a powerful microcontroller with built-in Wi-Fi and Bluetooth capabilities.
To read sensor data with the ESP32, you'll typically connect the sensor to the microcontroller.
To use a sensor with ESP32, you connect them together and write a simple program to collect data from the sensor.
You'll write code using a development environment like Arduino IDE and can see the data on a serial monitor.

Working with LEDs and Controlling Brightness

LEDs (Light Emitting Diodes) are semiconductor devices that emit light when a current passes through them.

You can control the brightness of an LED by adjusting the current flowing through it using techniques like pulse width modulation (PWM).

With the ESP32, you can use PWM functionality to control the

```
const int ledPin = 5; // Pin connected to the LED

void setup() {
  pinMode(ledPin, OUTPUT); // Set the LED pin as an output
}

void loop() {
  // Fade in
  for (int brightness = 0; brightness <= 255; brightness++) {
    analogWrite(ledPin, brightness); // Set the brightness of the LED
    delay(10); // Wait a short amount of time
  }

  // Fade out
  for (int brightness = 255; brightness >= 0; brightness--) {
    analogWrite(ledPin, brightness); // Set the brightness of the LED
    delay(10); // Wait a short amount of time
  }
}
```

brightness of an LED.

By varying the duty cycle of the PWM signal, you can adjust the perceived brightness of the LED.

PWM, or Pulse Width Modulation, is a method of controlling the average power delivered to a device by rapidly switching it on and off at a fixed frequency.

Building a Basic Sensor Circuit (Hands-on Activity)

You connect the sensor and ESP32 together following instructions.
You write a program (code) that tells the ESP32 how to read data from the sensor.
You upload the program to the ESP32, and it starts reading data from the sensor.
For example, if you're using a light sensor, it can tell you how bright the room is.

```
#include <DHT.h>

#define DHTPIN 2      // DHT22 data pin connected to digital pin 2
#define DHTTYPE DHT22 // DHT type is DHT22 (AM2302)

const int LDRPin = A0; // LDR pin connected to analog pin A0

DHT dht(DHTPIN, DHTTYPE); // Initialize DHT sensor

void setup() {
  Serial.begin(9600); // Start serial communication
  dht.begin();        // Start DHT sensor
}

void loop() {
  // Read temperature from DHT sensor
  float temperature = dht.readTemperature();
```

```
  // Read humidity from DHT sensor
  float humidity = dht.readHumidity();

  // Read light intensity from LDR
  int lightValue = analogRead(LDRPin);

  // Print temperature, humidity, and light intensity to serial monitor
  Serial.print("Temperature: ");
  Serial.print(temperature);
  Serial.print(" °C | Humidity: ");
  Serial.print(humidity);
  Serial.print("% | Light Value: ");
  Serial.println(lightValue);

  delay(2000); // Delay for stability
}
```

Day 2: MQTT Communication

Section 1: Introduction to MQTT Protocol

Understanding MQTT and its Advantages

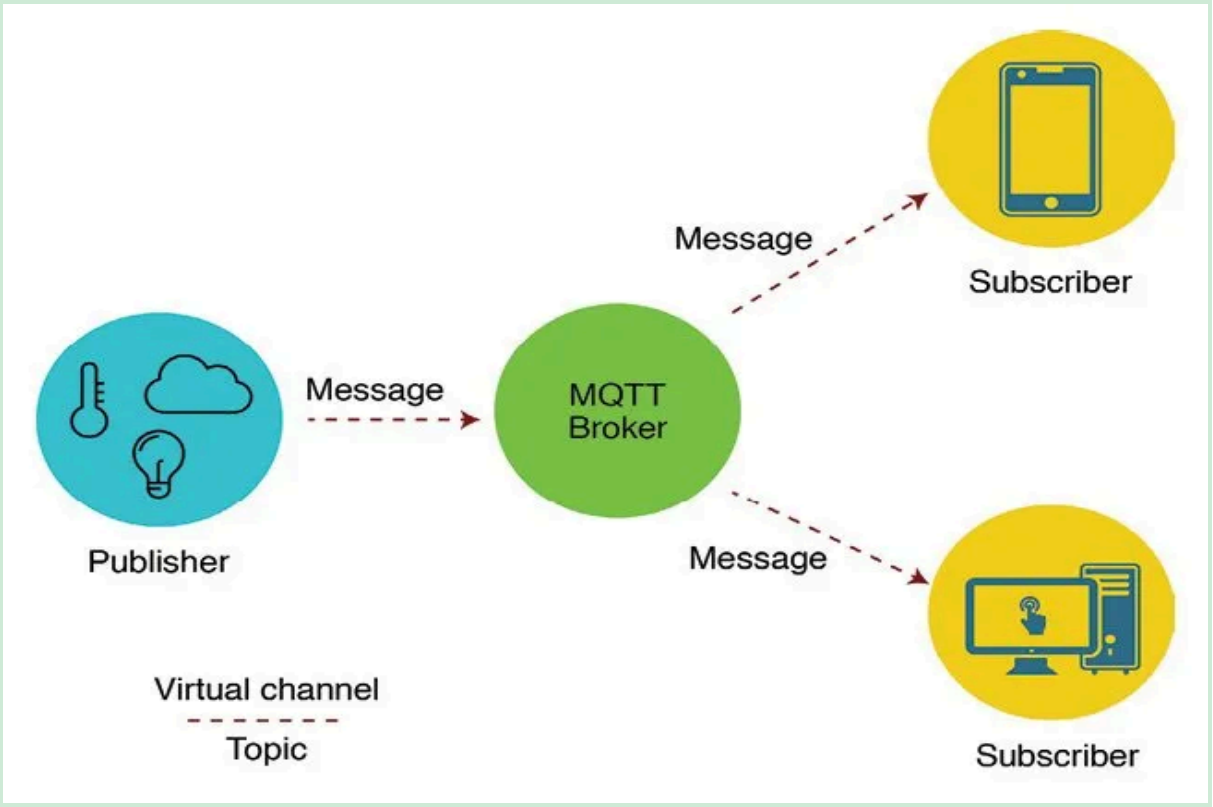
MQTT (Message Queue Telemetry Transport) is a messaging protocol designed for use on top of TCP/IP networks. It's a lightweight and publish-subscribe protocol, meaning devices can publish information (messages) to topics and other devices can subscribe to those topics to receive the information.

Here are some key features and advantages of MQTT:

- **Lightweight:** MQTT uses a small code footprint, making it suitable for devices with limited resources, like battery-powered sensors or microcontrollers used in the Internet of Things (IoT).
- **Low Bandwidth:** MQTT messages are compact, and the protocol minimises data transfer.This is beneficial for devices on networks with limited bandwidth, like cellular connections.

- **Reliable Delivery:** MQTT offers different Quality of Service (QoS) levels to ensure message delivery depending on the application's needs. Some delivery options prioritise reliability, while others focus on speed.
- **Secure:** MQTT can be secured with encryption and access control mechanisms to protect data privacy and integrity.

Overall, MQTT is a versatile messaging protocol that enables efficient and reliable communication between devices in various applications, particularly in the resource-constrained world of IoT.



Publish-Subscribe Model Explained

The publish-subscribe model is a messaging architecture used for communication between applications or devices. In this model, there are two main roles:

- **Publishers:** These are the message senders. They don't need to know who will receive the messages, they just publish them to a specific category or topic.
- **Subscribers:** These are the message receivers. They sign up for specific topics that they are interested in receiving messages about.

Here's a breakdown of how it works:

1. **Publishers** create messages containing data or information.

2. Publishers send these messages to a central server called a **message broker**.
3. The message broker doesn't store the messages themselves, but rather the topics they are associated with.
4. **Subscribers** specify the topics they are interested in.
5. When a message is published that matches a subscriber's interests (topic), the message broker routes the message to the subscriber.

Benefits of the Publish-subscribe model:

- **Decoupling:** Publishers and subscribers don't need to know about each other directly. This makes the system more scalable and flexible.
- **Loose Coupling:** Since publishers and subscribers don't interact directly, changes to one side of the system don't necessarily affect the other side. This simplifies development and maintenance.
- **Scalability:** The system can easily accommodate many publishers and subscribers without significant changes.

Analogy:

Imagine a classroom where students (subscribers) can sign up for different subjects (topics) offered by teachers (publishers). The teacher doesn't need to know who is enrolled in their class, they just present the material (publish messages). Students receive information only for the subjects they are signed up for (subscribed to).

Clients, Brokers, and Servers in MQTT

Clients

- These are the devices that send and receive messages using MQTT. They can be anything from tiny sensors to complex applications.
- **Examples of clients:** Smart home thermostats, wearables that track fitness data, industrial sensors that monitor temperature or pressure, or even mobile apps that receive data from these devices.

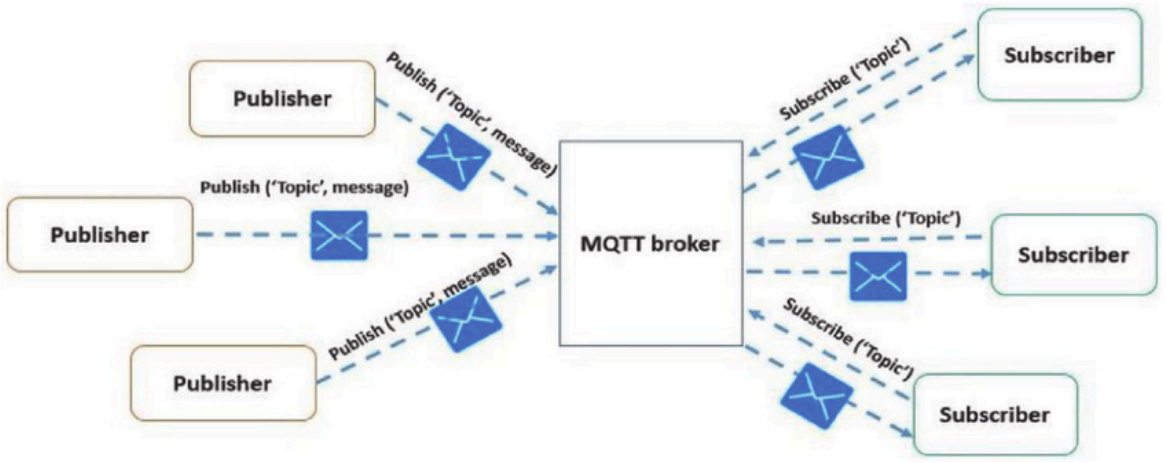
Brokers

- MQTT brokers act as central hubs for communication.They don't store the messages themselves, but they manage the flow of information between clients.
- **Think of a broker as a post office:** Clients (publishers) send their messages to the broker (post office), which then sorts and delivers them to the interested parties (subscribers).
- Brokers handle:

- **Routing messages:** They receive messages from publishers and then deliver them to the relevant subscribers based on their topics.
- **Security:** Brokers can enforce security measures like access control to ensure only authorised devices can publish or subscribe to certain topics.

Servers (MQTT Servers)

- In MQTT terminology, the term "server" is sometimes used interchangeably with "broker."Both refer to the software that manages message flow.
- So, an MQTT server is essentially an MQTT broker.



How they work together:

1. **Publishing:** A client with data to share (publisher) creates a message and sends it to the broker, specifying a topic.
2. **Routing:** The broker receives the message and uses the topic to identify interested subscribers.
3. **Delivery:** The broker delivers the message to all clients who have subscribed to that specific topic.

Section 2: Hand On with MQTT-Explorer

Introduction to the MQTT-Explorer

MQTT Explorer is a handy tool that lets you explore the world of MQTT, which is a messaging protocol used to connect devices on the Internet of Things (IoT). It acts like a central hub for these devices to communicate.

Here's a breakdown of what MQTT Explorer allows you to do:

- **Visualize topics:** Imagine folders and subfolders on a computer. MQTT Explorer lets you see the different topics (categories) that devices are publishing and subscribing to, similar to how you see folders organised on your device.
- **Publish messages:** You can try sending your own messages to specific topics, like pretending to be a sensor sending data.
- **Subscribe to topics:** You can listen in on conversations between devices by subscribing to specific topics.
- **See message history:** MQTT Explorer can keep track of messages that have been published so that you can review past information.

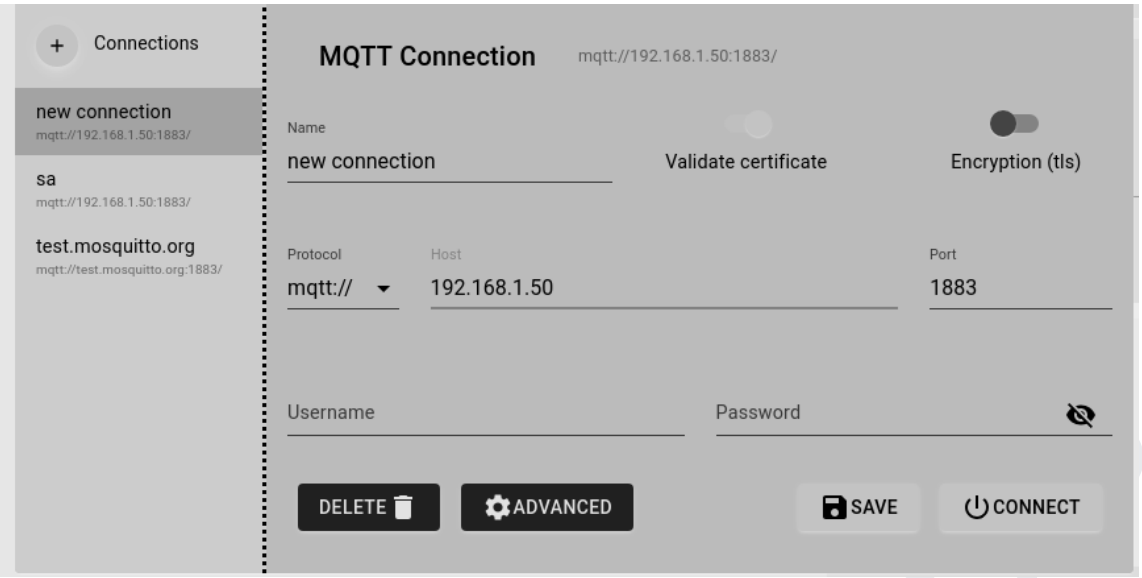
Publishing and Subscribing to Topic

Publishing:

- A device with data to share creates a message.
- The message includes the data and a topic that describes what kind of data it is.
- The device sends the message to an MQTT broker, which acts as a central server that manages message flow.

Subscribing:

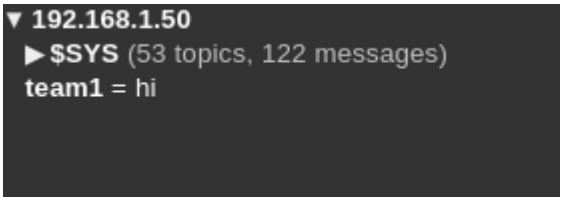
- A device that wants to receive certain types of messages tells the broker which topics it's interested in.
- This is like subscribing to a mailing list for a specific topic.
- When a message is published to a topic that a device is subscribed to, the broker routes the message to that device.



Setting Up The MQTT Explorer

1. **Connect to the Broker**: In the MQTT Explorer application, enter the IP address 192.168.1.50 in the "Host" field. This is the address of our MQTT broker, which acts as the central communication hub for publishing and subscribing to messages.
2. **Establish Connection:** After entering the broker's IP address, click the "Connect" button to establish a connection between the MQTT Explorer and the broker.
3. **Navigate to the Publish Section**: On the right-hand sidebar of the MQTT Explorer, locate and select the "Publish" section. This is where you can send messages (payloads) to specific topics on the broker.

4. **Enter the Topic**: In the "Topic" field, enter the name of the topic you want to publish your message to. Topics are used to categorize and organize messages within the MQTT ecosystem.
5. **Specify the Message Payload**: In the text box provided, enter the data or message you want to publish to the specified topic. This is commonly referred to as the "payload."
6. **Publish the Message**: Once you have entered the topic and payload, click the "Publish" button to send the message to the broker. The broker will then distribute the message to any clients subscribed to the specified topic.
7. **Retain the Message (Optional):** If you want the broker to store the current message and send it to any new subscribers to the topic, click the "Retain" button before publishing. This ensures that the latest message is always available for new subscribers.



8. **Visualize Published Messages**: On the left-hand side of the MQTT Explorer, you can see a list of all the topics and their respective messages. This allows you to monitor and visualize the data being published to the broker.
9. **View Message History**: To view the history of messages for a specific topic, select the desired topic from the list on the left-hand side. The MQTT Explorer will display the previous messages published on that topic, providing you with a chronological record of the data.

Section 3: Sensor Telemetry and Control with MQTT

Connecting ESP32 to Wi-Fi and MQTT Broker

To install ESP32 Board Support:

- Open the Arduino IDE.
- Go to File -> Preferences.
- In the "Additional Board Manager URLs" field, add the following URL: https://dl.espressif.com/dl/package_esp32_index.json.
- Click "OK" to close the Preferences window.
- Go to Tools -> Board -> Boards Manager.
- Search for "esp32" and install "ESP32 by Espressif Systems".

```
#include <WiFi.h>

const char* ssid = "YourWiFiSSID";
const char* password = "YourWiFiPassword";

void setup() {
  Serial.begin(115200);

  // Connect to WiFi
  WiFi.begin(ssid, password);
  Serial.println("Connecting to WiFi");

  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }

  Serial.println("");
  Serial.println("WiFi connected");
  Serial.print("IP address: ");
  Serial.println(WiFi.localIP());
}

void loop() {
  // Your code here
}
```

This is how the code is going to look like.(Replace the ssid and password with your wifi credentials)

Now to connect to the Broker:(PTO)

```ino
#include <WiFi.h>
#include <PubSubClient.h>

// Update these with your Wi-Fi credentials
const char* ssid = "ConForNode1";
const char* password = "12345678";

// MQTT Broker settings
const char* mqtt_server = "192.168.1.50";
const int mqtt_port = 1883; // Default MQTT port

WiFiClient espClient;
PubSubClient client(espClient);

// Callback function for received MQTT messages
void callback(char* topic, byte* payload, unsigned int length) {
  Serial.print("Message arrived [");
  Serial.print(topic);
  Serial.print("] ");
  for (int i = 0; i < length; i++) {
    Serial.print((char)payload[i]);
  }
  Serial.println();
}

void setup() {
  Serial.begin(115200);

  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.println("Connecting to WiFi..");
  }
  Serial.println("Connected to the WiFi network");

  client.setServer(mqtt_server, mqtt_port);
  client.setCallback(callback);

  while (!client.connected()) {
    Serial.println("Connecting to MQTT...");
    if (client.connect("ESP32Client")) {
      Serial.println("Connected to MQTT Broker!");
      // Subscribe to a topic (replace "test/topic" with your desired topic)
      client.subscribe("test/topic");
    } else {
      Serial.print("failed with state ");
      Serial.print(client.state());
      delay(2000);
    }
  }
}

void loop() {
  if (!client.connected()) {
    reconnect();
  }
  client.loop();
}

// Reconnect to MQTT Broker if connection is lost
void reconnect() {
  while (!client.connected()) {
    Serial.println("Attempting MQTT connection...");
    if (client.connect("ESP32Client")) {
      Serial.println("Connected to MQTT Broker!");
      client.subscribe("test/topic");
    } else {
      Serial.print("failed, rc=");
      Serial.print(client.state());
      Serial.println(" try again in 5 seconds");
      delay(5000);
    }
  }
}
```

Controlling LEDs Remotely with MQTT (Hands-on Activity)

```ino
#include <WiFi.h>
#include <PubSubClient.h>

// Update these with your Wi-Fi credentials
const char* ssid = "ConForNode1";
const char* password = "12345678";

// MQTT Broker settings
const char* mqtt_server = "192.168.1.50";
const int mqtt_port = 1883;

// Topic for controlling the LED
const char* ledTopic = "esp32/led";

const int ledPin = 2; // Change this to the actual pin connected to the LED

WiFiClient espClient;
PubSubClient client(espClient);

void setup() {
  Serial.begin(115200);
  pinMode(ledPin, OUTPUT);

  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.println("Connecting to WiFi..");
  }
  Serial.println("Connected to the WiFi network");

  client.setServer(mqtt_server, mqtt_port);
  client.setCallback(callback);

  connectToMQTT();
}

/**
 * Function to continuously check the connection status,
 * and reconnect to MQTT if not connected.
 */
void loop() {
  if (!client.connected()) {
    connectToMQTT();
  }
  client.loop();
}

/**
 * Function to establish connection to MQTT server.
 *
 * @return void
 *
 * @throws None
 */
void connectToMQTT() {
  while (!client.connected()) {
    Serial.println("Connecting to MQTT...");
    if (client.connect("ESP32Client")) {
      Serial.println("Connected to MQTT Broker!");
      client.subscribe(ledTopic);
    } else {
      Serial.print("failed with state ");
      Serial.print(client.state());
      delay(2000);
    }
  }
}
```

Publishing Sensor Data to MQTT Topics (Same as the first code in this section). To get a Sensor data we store it in a variable and the publish it using

Building a Sensor Telemetry System (Hands-on Activity)

Day 3: IoT-Enabled Miniature Room Project

Section 1: Project Overview and Planning

Defining Project Goals and Requirements

Building the Miniature Room