

# **MIT ADT University**

## **Pune**

**MIT School of Computing**

Department of Computer Science and Engineering

F.Y B.Tech (ASH)  
Course : Computer Engineering Workshop

**DIY IoT Kit Lab Manual**  
**[ A.Y 2025-26 ]**

Activity Based Learning for effective implementation of Project Based Learning in  
NEP2020

## INDEX

Sr. No	Activity	Title
1	<b>DIY IoT Kit Activity-1</b>	On-board LED Blinking on NodeMCU
2	<b>DIY IoT Kit Activity-2</b>	Interfacing Ultrasonic Sensor with NodeMCU
3	<b>DIY IoT Kit Activity-3</b>	Interfacing OLED with NodeMCU
4	<b>DIY IoT Kit Activity-4</b>	Interfacing DHT11 with NodeMCU and displaying the output on OLED
5	<b>DIY IoT Kit Activity-5</b>	PWM-based LED Brightness Control using NodeMCU
6	<b>DIY IoT Kit Activity-6</b>	IoT-Based Environmental Data Logging and Visualization on Localhost using NodeMCU

# **Embedded Systems :**

An embedded system is a specialized computer system that is designed to perform a dedicated function or a small set of functions, often within a larger mechanical or electrical system.

It combines hardware (microcontroller, microprocessor, sensors, actuators, etc.) and software (firmware, real-time operating system, control algorithms) to achieve specific tasks.

Unlike general-purpose computers (like PCs or laptops), embedded systems are application-specific, meaning they are optimized for efficiency, reliability, and performance in a particular task.

例子:Microwave oven controller,Airbag system in cars,Pacemaker in medical applications,Flight control system in aircraft

## **Types of Embedded Systems**

Embedded systems can be classified in **two main ways**:

<p><b>1. Based on Performance and Functional Requirements</b></p> <ul style="list-style-type: none"><li>● <b>Small-Scale Embedded Systems</b><ul style="list-style-type: none"><li>○ Use 8-bit or 16-bit microcontrollers.</li><li>○ Limited memory, simple hardware.</li><li>○ Example: Washing machine, digital watch.</li></ul></li><li>● <b>Medium-Scale Embedded Systems</b><ul style="list-style-type: none"><li>○ Use 16-bit or 32-bit microcontrollers/microprocessors.</li><li>○ Run on Real-Time Operating Systems (RTOS).</li></ul></li></ul>	<p><b>2. Based on Real-Time Operation</b></p> <ul style="list-style-type: none"><li>● <b>Real-Time Embedded Systems</b><ul style="list-style-type: none"><li>○ Designed to give output within strict timing constraints.</li><li>○ Two types:<ul style="list-style-type: none"><li>■ <b>Hard Real-Time Systems</b> → Missing a deadline = failure (e.g., pacemaker, anti-lock braking system).</li><li>■ <b>Soft Real-Time Systems</b> → Occasional delays are tolerable (e.g., video streaming, online gaming console).</li></ul></li></ul></li><li>● <b>Standalone Embedded Systems</b></li></ul>
--	---

<ul style="list-style-type: none"> <li>○ Example: ATM machines, home automation systems.</li> </ul> <p><b>• Complex (or Large-Scale) Embedded Systems</b></p> <ul style="list-style-type: none"> <li>○ Use powerful 32-bit or 64-bit processors (ARM Cortex, DSPs, FPGAs).</li> <li>○ Handle complex algorithms, networking, and advanced real-time constraints.</li> <li>○ Example: Autonomous vehicles, medical imaging devices, industrial robots.</li> </ul>	<ul style="list-style-type: none"> <li>○ Work independently without needing a host system.</li> <li>○ Example: MP3 player, microwave.</li> </ul> <p><b>• Networked Embedded Systems</b></p> <ul style="list-style-type: none"> <li>○ Connected to a network (LAN, WAN, Internet, IoT).</li> <li>○ Example: Smart home devices, smart meters.</li> </ul> <p><b>• Mobile Embedded Systems</b></p> <ul style="list-style-type: none"> <li>○ Portable devices with battery operation.</li> <li>○ Example: Smartphones, tablets, handheld GPS.</li> </ul>
--	--

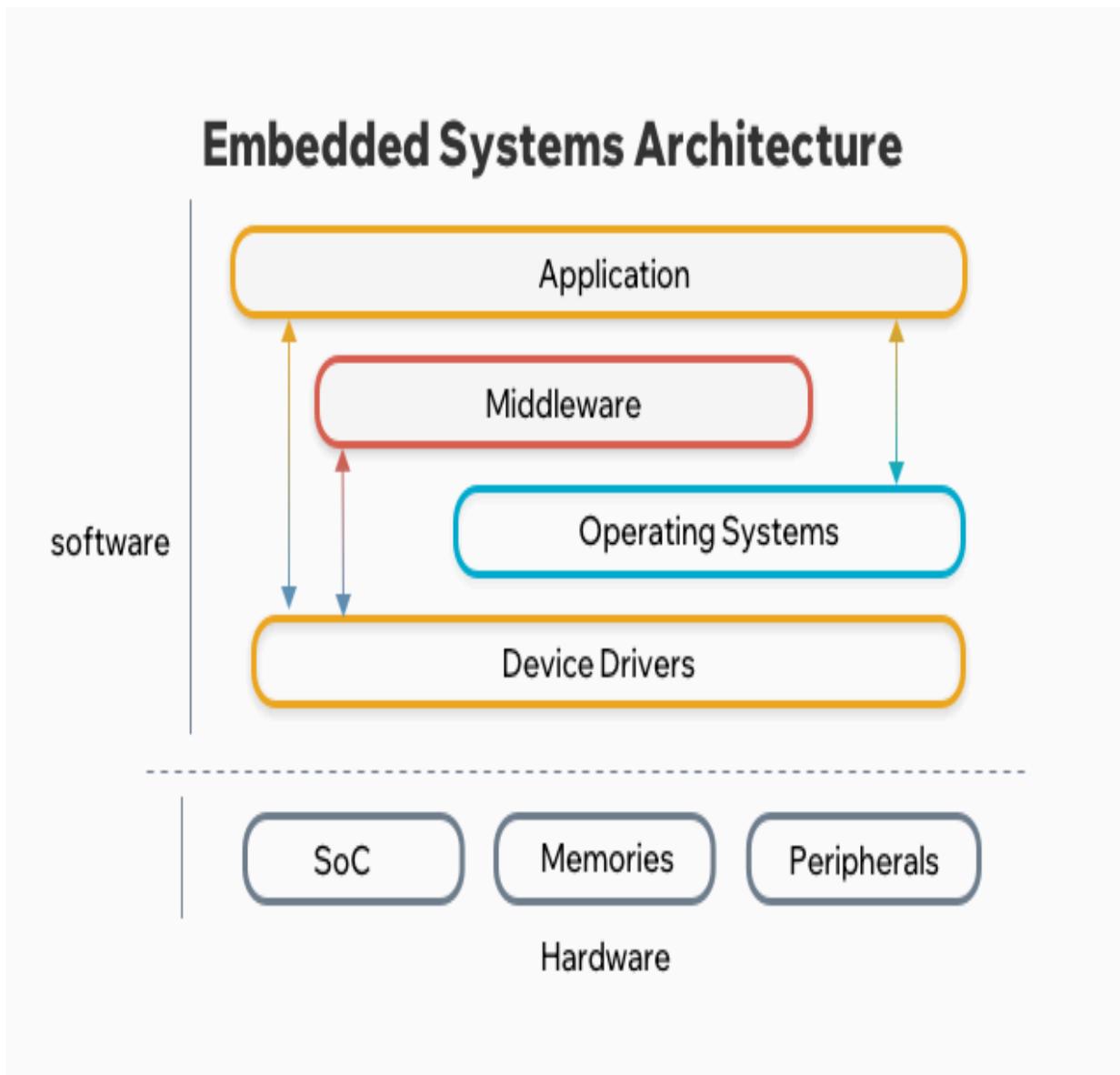
**Application Layer :** The application layer sits at the helm, defining the system's specific function. It houses the application code, the brains behind the operation. For example, If the system controls a traffic light, this layer will handle the logic for switching lights based on timer and sensor inputs. Components of the application layer include:

- **Application Code:** This code, written in languages like C/C++ or assembly, implements the core logic needed to achieve the system's goal. It interacts with the middleware layer (if present) to utilize system resources effectively.
- **Device-Specific Functionality:** For complex systems, this layer contains code tailored to interact with specific devices like sensors or displays.

**Middleware Layer :** The middleware acts as a bridge between the application and the lower levels. It simplifies interactions in the embedded system and also provides essential services. The components of the middleware layer are:

- **Device Drivers:** These software programs act as interpreters, translating application commands into instructions that specific hardware devices can understand.
- **Communication Protocols:** For systems that need to interact with other devices or networks, this layer manages communication protocols like TCP/IP or Bluetooth.
- **Security Services:** In security-critical systems, this layer implements encryption, authentication, and access control mechanisms.

**Operating System (OS) Layer** : The Operating System, usually in complex systems, manages resources like memory, CPU time, and peripherals, enabling multitasking and efficient operation. This layer houses the:



- **Kernel:** The core of the OS, responsible for task scheduling, memory management, and device management.
- **File System:** Manages storage and access to data files needed by the application.
- **Inter-Process Communication (IPC):** Facilitates communication and data exchange between different parts of the application or multiple applications.

**Device Drivers Layer** : This layer directly controls the hardware components through dedicated software programs called device drivers. The device drivers layer contains:

- **Sensor Drivers:** They are responsible for reading data from sensors like temperature, pressure, or motion.
- **Actuator Drivers:** Control actuators like motors, LEDs, or displays to generate outputs based on application commands.
- **Peripheral Drivers:** Manage peripherals like communication interfaces (e.g., UART, SPI) for data exchange with the external world.

**Hardware Layer :** This layer comprises the physical building blocks of the system, which provides the processing power and I/O capabilities. Components include:

- **Processor (CPU):** Executes the instructions provided by the software, performs calculations, and controls data flow.
- **Memory:** Stores program code, data, and intermediate results. It can be volatile (RAM) or non-volatile (ROM/Flash).
- **Peripherals:** Contains devices like sensors, actuators, communication interfaces, displays, and timers that enable the system to interact with the environment and perform its designated tasks.

### Getting Started with NodeMCU (Initial SetUp in Arduino IDE)

<https://projecthub.arduino.cc/PatelDarshil/getting-started-with-nodemcu-esp8266-on-arduino-ide-b193c3>

## Activity No 1

**Title :** On-board LED Blinking on NodeMCU

**Circuit Diagram :** The onboard LED is usually connected to GPIO2 (D4)

**Code :**

```
int ledPin = LED_BUILTIN; // On-board LED pin (D4 / GPIO2)
void setup()
{
    pinMode(ledPin, OUTPUT); // Set LED pin as output
}

void loop()
{
    digitalWrite(ledPin, LOW); // Turn LED ON (LOW = ON for built-in LED)
    delay(1000);           // Wait 1 second
    digitalWrite(ledPin, HIGH); // Turn LED OFF
    delay(1000);           // Wait 1 second
}
```

**Output :** Check the LED Blinking at a rate of 1 second on the board.

## Activity No 2

**Title :** Interfacing Ultrasonic Sensor with NodeMCU

### Theory :

An **ultrasonic sensor** measures **distance** by using **sound waves** that humans can't hear (frequency > 20 kHz).

#### Trigger Pulse

- The microcontroller (e.g., NodeMCU) sends a **10 µs HIGH signal** to the TRIG pin of the sensor.
- This tells the sensor to send an ultrasonic burst.

#### Ultrasonic Burst

- The sensor emits **8 cycles of 40 kHz sound waves** from its **transmitter (T)**.
- These waves travel through the air.

#### Reflection (Echo)

- If the sound hits an object, it bounces back.
- The **receiver (R)** on the sensor detects this reflected wave.

#### Echo Pulse Width

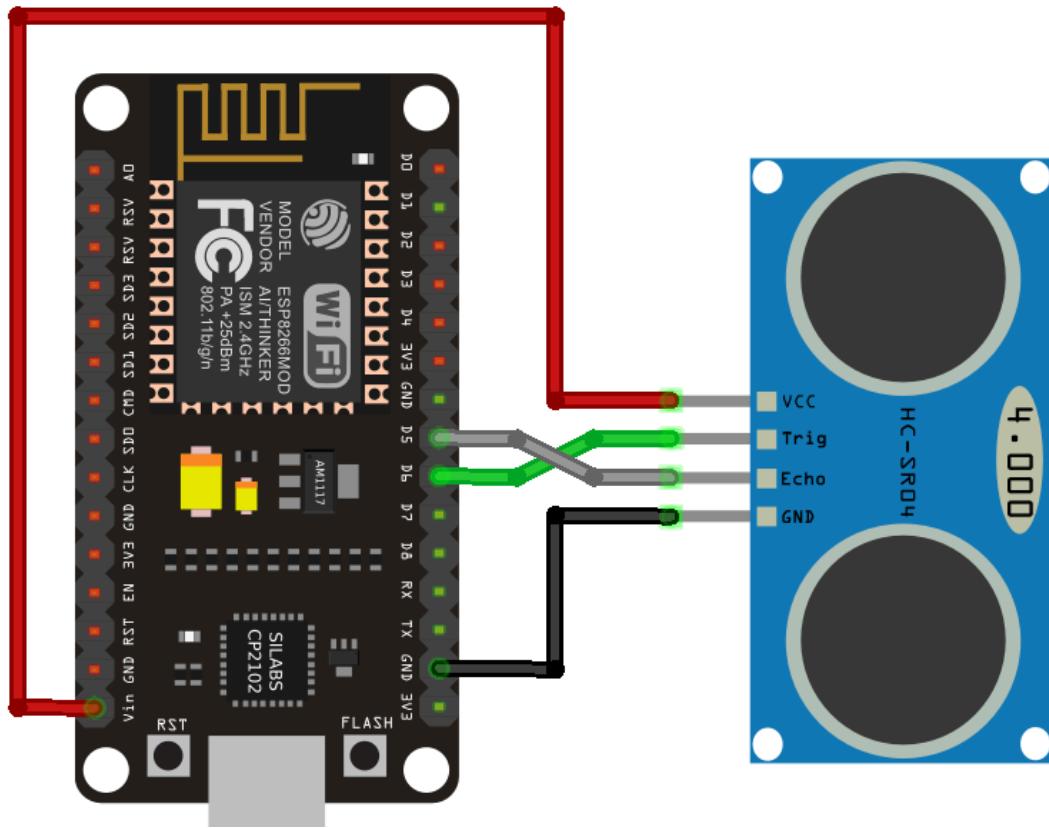
- While waiting for the echo, the sensor keeps the **ECHO pin HIGH**.
- Once the reflection is received, the **ECHO pin goes LOW**.
- The **time duration of the HIGH pulse** on the ECHO pin equals the **round-trip travel time** of the sound wave.

#### Distance Calculation

- Speed of sound in air  $\approx$  **343 m/s** (0.0343 cm/µs).
- Since the wave travels to the object and back, we divide by 2:

$$\text{Distance (cm)} = \frac{\text{Time (\mu s)} \times 0.0343}{2}$$

## Circuit Diagram :



## Code:

```
// HC-SR04 Ultrasonic Sensor with NodeMCU (ESP8266)

#define TRIG_PIN D5 // GPIO14
#define ECHO_PIN D6 // GPIO12

long duration;
float distance;

void setup() {
  Serial.begin(115200); // Start Serial Monitor
```

```
pinMode(TRIG_PIN, OUTPUT); // TRIG as Output
pinMode(ECHO_PIN, INPUT); // ECHO as Input
}

void loop() {
    // --- Trigger the ultrasonic burst ---
    digitalWrite(TRIG_PIN, LOW);
    delayMicroseconds(2);
    digitalWrite(TRIG_PIN, HIGH);
    delayMicroseconds(10);
    digitalWrite(TRIG_PIN, LOW);

    // --- Read echo response ---
    duration = pulseIn(ECHO_PIN, HIGH);

    // --- Calculate distance ---
    // Speed of sound = 343 m/s = 0.0343 cm/µs
    distance = (duration * 0.0343) / 2;

    // Print result
    Serial.print("Distance: ");
    Serial.print(distance);
    Serial.println(" cm");

    delay(500); // small delay
}
```

## Activity No 03

**Title :** Interfacing OLED with NodeMCU to display “Hello World”

### What is an OLED Display?

**OLED** stands for **Organic Light Emitting Diode**.

It is a type of display where **each pixel emits its own light** when an electric current passes through organic compounds. Unlike LCDs, OLEDs don't need a **backlight**, because the pixels themselves generate light. This makes them **brighter, thinner, and more power-efficient**, especially when displaying dark images.

### Basic Working Principle

1. An OLED display is made of **organic thin films** placed between two conductors (anode and cathode).
2. When voltage is applied:
  - o Electrons and holes recombine in the organic layer.
  - o This process releases energy in the form of **visible light**.
3. The color and brightness depend on the **materials used** and the **amount of current** flowing through each pixel.

### Types of OLED Displays (in hobby electronics)

Type	Communication	Example Controller	Description
I <sup>2</sup> C	2-wire (SCL, SDA)	SSD1306, SH1106	Easier wiring (uses only 2 pins)
SPI	4-wire (SCK, MOSI, CS, DC)	SSD1306, SH1106	Faster data transfer

## Arduino IDE Setup

### Step 1: Install Libraries

Go to **Sketch → Include Library → Manage Libraries**, then install:

- Adafruit SSD1306
- Adafruit GFX Library

## Circuit Diagram

OLED → NodeMCU

---

VCC → 3.3V

GND → GND

SCL → D1 (GPIO5)

SDA → D2 (GPIO4)

```
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>

#define SCREEN_WIDTH 128 // OLED display width
#define SCREEN_HEIGHT 64 // OLED display height
#define OLED_RESET -1 // Reset pin (not used)

Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire,
OLED_RESET);

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

```
// Initialize OLED
if(!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) { // Address 0x3C for most
displays
    Serial.println(F("SSD1306 allocation failed"));
    for(;;);
}

display.clearDisplay();
display.setTextSize(1);
display.setTextColor(SSD1306_WHITE);
display.setCursor(0, 10);
display.println(F("Hello, NodeMCU!"));
display.display();
}

void loop() {
    // You can add animations or sensor data here later
}
```

## Activity No 04

**Title :** Interfacing DHT11 with NodeMCU and displaying the output on OLED

**Theory :**

### 1. DHT11 Sensor

#### Overview

The **DHT11** is a basic, low-cost **digital temperature and humidity sensor**. It provides **humidity and temperature readings** in a digital format, so you don't need an analog-to-digital converter (ADC).

#### Features

- Measures **temperature**: 0–50°C, ±2°C accuracy
- Measures **humidity**: 20–90% RH, ±5% accuracy
- **Digital output** via a single wire (data pin)
- Low cost, widely available
- Low power consumption

#### Working Principle

1. DHT11 has a **thermistor** to measure temperature and a **capacitive humidity sensor** to measure humidity.
2. The **microcontroller sends a start signal** to the sensor.
3. DHT11 responds with **40 bits of data**:
  - 16 bits for humidity
  - 16 bits for temperature
  - 8 bits for checksum
4. The microcontroller reads the bits using **timing-sensitive pulse width detection**.
5. Data is processed and converted into actual **temperature (°C)** and **humidity (%)** values.

#### Applications

- Weather monitoring stations

- Home automation
- Greenhouse monitoring
- IoT projects
- HVAC systems

## 2. OLED Display (0.96" SSD1306)

### Overview

An **OLED (Organic Light Emitting Diode) display** is a **self-emissive display**, meaning each pixel emits light individually. Unlike LCDs, OLEDs **do not require a backlight**, making them bright, thin, and power-efficient.

### Features

- Resolution: 128×64 pixels (commonly 0.96" screen)
- I2C interface (SDA, SCL) or SPI interface
- Monochrome (usually white) or RGB variants
- Fast response time, wide viewing angle
- Low power consumption

### Working Principle

1. Each pixel in OLED is made of **organic materials** that emit light when an electric current passes through.
2. **I2C or SPI** protocol allows communication with a microcontroller like NodeMCU.
3. Microcontroller sends **commands and data** to control individual pixels:
  - **Commands:** turn pixels on/off, set cursor, clear screen
  - **Data:** actual pixel pattern for text, graphics, or icons
4. No backlight is required; black pixels are simply off, which saves energy.

### Applications

- Wearable devices
- IoT dashboards
- Smart home displays

- Portable instruments
- Microcontroller projects requiring visual output

### **Circuit Diagram :**

#### **DHT11 to NodeMCU**

<b>DHT11 Pin</b>	<b>NodeMCU Pin</b>
VCC	3.3V
GND	GND
DATA	D4 (GPIO2)

#### **OLED to NodeMCU**

<b>OLED Pin</b>	<b>NodeMCU Pin</b>
VCC	3.3V
GND	GND
SDA	D2 (GPIO4)
SCL	D1 (GPIO5)

**Code:**

```
#include <Adafruit_SSD1306.h>
#include <Adafruit_GFX.h>
#include <DHT.h>

#define SCREEN_WIDTH 128 // OLED width in pixels
#define SCREEN_HEIGHT 64 // OLED height in pixels
#define OLED_RESET     -1 // Reset pin (not used for I2C)
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET);

#define DHTPIN D4      // DHT11 data pin
#define DHTTYPE DHT11 // DHT 11
DHT dht(DHTPIN, DHTTYPE);

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

    // Initialize DHT sensor
    dht.begin();

    // Initialize OLED
    if(!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) { // 0x3C is the I2C
address
        Serial.println(F("SSD1306 allocation failed"));
        for(;;);
    }
    display.clearDisplay();
    display.setTextSize(2);
    display.setTextColor(SSD1306_WHITE);
    display.setCursor(0, 0);
    display.println("DHT11 Ready");
    display.display();
    delay(2000);
}

void loop() {
    // Read temperature and humidity
    float h = dht.readHumidity();
    float t = dht.readTemperature();

    // Check if any reads failed
    if (isnan(h) || isnan(t)) {
        Serial.println(F("Failed to read from DHT sensor!"));
    }
}
```

```
display.clearDisplay();
display.setCursor(0, 0);
display.println("Read Error!");
display.display();
delay(2000);
return;
}

// Display on Serial Monitor
Serial.print(F("Humidity: "));
Serial.print(h);
Serial.print(F("%  Temperature: "));
Serial.print(t);
Serial.println(F("°C"));

// Display on OLED
display.clearDisplay();
display.setCursor(0,0);
display.setTextSize(2);
display.print("T:");
display.print(t);
display.println("C");

display.setCursor(0,30);
display.print("H:");
display.print(h);
display.println("%");

display.display();
delay(2000);
}
```

## Activity No 5

**Title :** PWM-based LED Brightness Control using NodeMCU

 **Objective :** To understand and demonstrate Pulse Width Modulation (PWM) by controlling the brightness of an LED using a NodeMCU board.

### Theory

PWM (Pulse Width Modulation) is a technique to control the average power delivered to an electronic device by switching it ON and OFF rapidly.

- Duty Cycle (%) = (Time LED ON / Total PWM Period) × 100
- A higher duty cycle → LED appears brighter.
- A lower duty cycle → LED appears dimmer.

The NodeMCU (ESP8266) can generate PWM signals using the `analogWrite()` function on most GPIO pins.

Typical frequency: 1 kHz

### Code :

```
// PWM LED Brightness Control on NodeMCU (ESP8266)

// This program demonstrates how Pulse Width Modulation (PWM)
// can be used to gradually vary LED brightness.

int ledPin = D4;      // Assign pin D4 as the LED output pin

int brightness = 0;    // Initialize brightness variable (range: 0 to 1023)

int fadeAmount = 5;    // Amount by which brightness will increase or decrease each
cycle

void setup() {
    pinMode(ledPin, OUTPUT); // Set the LED pin as an output pin to send PWM signal
```

```
}
```

```
void loop() {  
    analogWrite(ledPin, brightness);  
    // Generate PWM signal on ledPin with duty cycle proportional to 'brightness'  
    // analogWrite() on NodeMCU accepts values from 0 (0% duty cycle) to 1023 (100%  
    // duty cycle)  
  
    brightness = brightness + fadeAmount;  
    // Increment or decrement brightness value to create fading effect  
  
    if (brightness <= 0 || brightness >= 1023) {  
        fadeAmount = -fadeAmount;  
        // Reverse the direction of fading when brightness hits minimum (0) or maximum  
        // (1023)  
        // If brightness is increasing and reaches 1023, start decreasing  
        // If brightness is decreasing and reaches 0, start increasing again  
    }  
  
    delay(30);  
    // Wait for 30 milliseconds before updating brightness again  
    // This controls the speed of fading — smaller delay = faster fade effect  
}
```

## Activity No 6

**Title :** IoT-Based Environmental Data Logging and Visualization on Localhost using NodeMCU



### Hardware Setup

- DHT11 sensor → NodeMCU ESP8266
  - VCC → 3.3V
  - GND → GND
  - DATA → D4 (GPIO2)



### 1. NodeMCU ESP8266 as a Web Server

- The NodeMCU ESP8266 is a microcontroller with built-in Wi-Fi.
- It can connect to a local Wi-Fi network and host a web server.
- A web server is a system that delivers web pages (HTML, CSS, JavaScript) to clients (your laptop/phone browser).
- In this experiment, the NodeMCU acts as both:
  - Data acquisition system (collects sensor values).
  - Web server (displays and shares data).



### 2. DHT11 Sensor

- DHT11 is a low-cost digital sensor that measures:

- Temperature (0–50 °C, ±2 °C accuracy).
- Humidity (20–90 %RH, ±5% accuracy).
- It communicates with the microcontroller using a single-wire protocol, meaning only one GPIO pin is needed to transfer data.
- The NodeMCU reads values from the DHT11 periodically (every few seconds).

### 3. NTP (Network Time Protocol)

- The ESP8266 does not have a built-in real-time clock (RTC).
- To get the current time, it connects to an NTP server (like [pool.ntp.org](http://pool.ntp.org)).
- The NTP server provides accurate date and time over the internet.
- In this project, timestamps (HH:MM:SS) from NTP are added to each sensor reading.

### 4. Web Dashboard

- The web page served by NodeMCU contains:
  - HTML → defines structure (tables, buttons).
  - CSS → styles the dashboard (colors, fonts, layout).
  - JavaScript → fetches new sensor values from the ESP8266 without refreshing the page.
- The page updates every 5 seconds using AJAX (asynchronous requests).

### 5. Data Logging & Excel Export

- Each new reading is displayed in the table with timestamp, temperature, humidity.
- JavaScript collects all rows of data and generates a CSV (Comma Separated Values) file.
- CSV is a standard format that can be opened in Microsoft Excel, Google Sheets, or LibreOffice Calc for further analysis.

- This allows the experiment to be used for data analysis and visualization outside the NodeMCU environment.

## 6. Experiment Workflow

1. Power NodeMCU and connect to Wi-Fi.
2. Open browser → enter NodeMCU's IP (shown in Serial Monitor).
3. The dashboard loads, showing sensor values in a table.
4. Every 5 seconds, new readings are added with real-time stamps.
5. At any point, click Download CSV → get Excel-compatible log file.

## 7. Applications

- Environmental monitoring → track room temperature & humidity.
- IoT data logging → collect sensor values over Wi-Fi.
- Smart home systems → monitor indoor climate remotely.
- Research & education → demonstrates integration of sensors, web technologies, and IoT.

Code :

```
#include <ESP8266WiFi.h>

#include <ESP8266WebServer.h>

#include <DHT.h>

#include <time.h>
```

```
// ----- Wi-Fi Settings -----  
  
const char* ssid = "YOUR_WIFI_NAME";  
  
const char* password = "YOUR_WIFI_PASSWORD";  
  
  
// ----- DHT11 Settings -----  
  
#define DHTPIN D4  
  
#define DHTTYPE DHT11  
  
DHT dht(DHTPIN, DHTTYPE);  
  
  
// ----- Web Server -----  
  
ESP8266WebServer server(80);  
  
  
// ----- Data Storage -----  
  
String csvLog = "Date,Time,Temperature (°C),Humidity (%)\n";  
  
String tableRows = ""; // Latest readings prepended  
  
  
// ----- Time Setup -----  
  
void setupTime() {  
  
    // UTC+5:30 offset (India Standard Time)  
  
    configTime(19800, 0, "pool.ntp.org", "time.nist.gov");  
  
    Serial.println("Waiting for time...");  
  
    while (!time(nullptr)) {  
  
        delay(500);  
  
        Serial.print(".");  
    }  
}
```

```

}

Serial.println("\n✓ Time synchronized");

}

// ----- Get Formatted Date & Time -----

void getDateTime(String &dateStr, String &timeStr) {

    time_t now = time(nullptr);

    struct tm *ptm = localtime(&now); // localtime includes UTC+5:30

    char dateBuffer[11]; // DD-MM-YYYY

    sprintf(dateBuffer, "%02d-%02d-%04d", ptm->tm_mday, ptm->tm_mon + 1,
    ptm->tm_year + 1900);

    dateStr = String(dateBuffer);

    char timeBuffer[9]; // HH:MM:SS

    sprintf(timeBuffer, "%02d:%02d:%02d", ptm->tm_hour, ptm->tm_min,
    ptm->tm_sec);

    timeStr = String(timeBuffer);

}

// ----- Serve Dashboard -----


void handleRoot() {

    // HTML page with JS to fetch data every 2 seconds

    String html = R"rawliteral(
<!DOCTYPE html>
<html>

```

```
<head>

<title>NodeMCU DHT11 Dashboard</title>

<style>

    body { font-family: Arial; text-align: center; margin: 20px; }

    h2 { color: #333; }

    table { width: 80%; margin: 20px auto; border-collapse: collapse; }

    th, td { border: 1px solid #ddd; padding: 8px; text-align: center; }

    th { background: #4CAF50; color: white; }

    tr:nth-child(even) { background: #f2f2f2; }

        button { margin: 10px; padding: 10px 20px; background: #4CAF50; color: white; border: none; cursor: pointer; }

        button:hover { background: #45a049; }

</style>

</head>

<body>

    <h2>NodeMCU Temperature & Humidity Dashboard</h2>

    <button onclick="downloadCSV()">Download CSV</button>

    <table id="dataTable">

        <tr><th>Date</th><th>Time</th><th>Temperature (°C)</th><th>Humidity (%)</th></tr>

    </table>

    <script>

        async function fetchData() {

            const response = await fetch('/data');

            const data = await response.json();

        }

    </script>
```

```
const table = document.getElementById('dataTable');

const row = table.insertRow(1); // Insert at top

row.insertCell(0).innerText = data.date;

row.insertCell(1).innerText = data.time;

row.insertCell(2).innerText = data.temperature;

row.insertCell(3).innerText = data.humidity;

}

// Fetch new data every 2 seconds

setInterval(fetchData, 2000);

// CSV download

function downloadCSV() {

fetch('/download').then(res => res.text()).then(csv => {

const blob = new Blob([csv], { type: 'text/csv' });

const url = window.URL.createObjectURL(blob);

const a = document.createElement('a');

a.href = url;

a.download = 'sensor_data.csv';

a.click();

window.URL.revokeObjectURL(url);

});

}
```

```

// Load initial data once

fetchData();

</script>

</body>

</html>

)rawliteral";

server.send(200, "text/html", html);

}

// ----- Serve JSON data for JS -----

void handleData() {

float temperature = dht.readTemperature();

float humidity = dht.readHumidity();

String date, timeStr;

getTime(date, timeStr);

// Save to CSV

csvLog += date + "," + timeStr + "," + String(temperature) + "," + String(humidity)
+ "\n";

String json = "{\"date\":\"" + date + "\",\"time\":\"" + timeStr +
"\",\"temperature\":\"" + String(temperature) + "\",\"humidity\":\"" +
String(humidity) + "\"}";

server.send(200, "application/json", json);

}

```

```
// ----- Serve CSV -----  
  
void handleDownload() {  
    server.send(200, "text/csv", csvLog);  
}  
  
// ----- Setup -----  
  
void setup() {  
    Serial.begin(115200);  
  
    dht.begin();  
  
    // Connect to Wi-Fi  
  
    WiFi.begin(ssid, password);  
  
    Serial.print("Connecting to WiFi");  
  
    while (WiFi.status() != WL_CONNECTED) {  
        delay(500);  
  
        Serial.print(".");  
    }  
  
    Serial.println("\n✓ WiFi connected!");  
  
    Serial.print("🌐 NodeMCU IP Address: ");  
  
    Serial.println(WiFi.localIP());  
  
    // Initialize time  
  
    setupTime();
```

```

// Server routes

server.on("/", handleRoot);

server.on("/data", handleData);

server.on("/download", handleDownload);

server.begin();

Serial.println("✅ Web server started");

}

```

// ----- Loop -----

```

void loop() {

server.handleClient();

}

```

## Output :

NodeMCU Temperature & Humidity Dashboard

[Download CSV](#)

Date	Time	Temperature (°C)	Humidity (%)
28-09-2025	01:32:36	26.50	86.30
28-09-2025	01:32:34	26.50	86.30
28-09-2025	01:32:32	26.50	86.10
28-09-2025	01:32:30	26.50	86.30
28-09-2025	01:32:28	26.50	86.30
28-09-2025	01:32:26	26.50	86.10
28-09-2025	01:32:24	26.50	85.90
28-09-2025	01:32:22	26.50	85.90
28-09-2025	01:32:20	26.40	86.20
28-09-2025	01:32:18	26.40	86.20
28-09-2025	01:32:16	26.50	86.10