**Introduction to IoT**

The **Internet of Things (IoT)** refers to a network of physical objects—**"things"**—embedded with sensors, software, and other technologies to connect and exchange data with other devices and systems over the internet.

**Examples of IoT Devices:**

* Smart thermostats
* Wearable fitness trackers
* Connected home appliances
* Industrial sensors

**Components of IoT**



**1. Sensors / Devices**

These collect data from the environment.  
Examples: temperature sensors, motion sensors, light sensors.

**2. Connectivity**

Enables data transfer to the cloud or other devices.  
Technologies: Wi-Fi, Bluetooth, GSM, Zigbee, LoRa, etc.

**3. Data Processing**

Processes the collected data to make decisions.  
This may happen on:

* The microcontroller itself (local or edge computing)
* A server or cloud platform (remote processing)  
  Example: If motion is detected → Turn on a light.

**4. User Interface**

Allows users to interact with the system.  
Examples:

* A mobile app showing temperature
* A web dashboard to monitor data or control devices

**Essential Hardware Components**

* **Microcontrollers:**
  + E.g., Arduino Uno, ESP8266; act as the "brains" of IoT devices.
* **Sensors:**
  + Detect physical and environmental changes.
  + Examples:
    - Motion sensors (e.g., PIR sensors)
    - Temperature sensors (e.g., DHT11)
    - Light sensors (e.g., LDRs)
* **Actuators:**
  + Perform actions in response to processed data.
  + Examples:
    - Buzzers sounding an alarm
    - LEDs indicating system status
* **Communication Modules:**
  + Enable data transmission.
  + Examples:
    - Wi-Fi modules (e.g., ESP8266)
    - GSM modules for SMS alerts

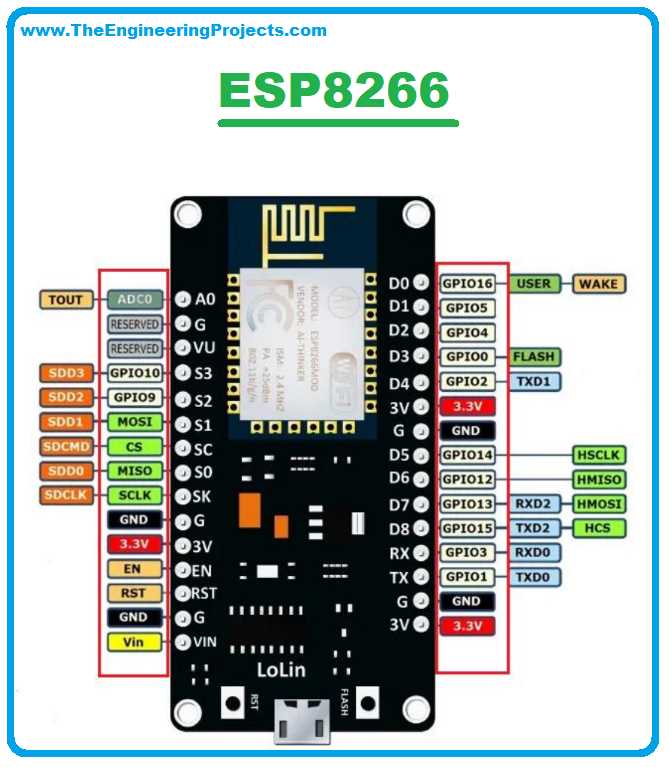
**Common Boards and Their Pinouts**

* + - 1. **ESP8266 (NodeMCU)**

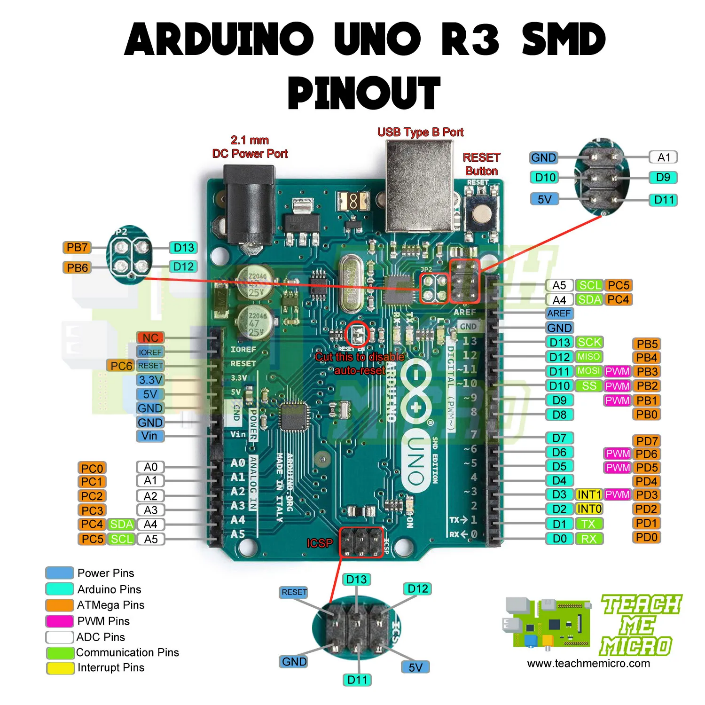
**Overview**: The ESP8266 is a low-cost Wi-Fi microchip with full TCP/IP stack and microcontroller capability. The NodeMCU is a development board that incorporates the ESP8266.

**Pinout Highlights**:

* **Digital I/O Pins (D0–D8)**: Used for digital input or output.
* **Analog Input (A0)**: Used to read analog signals (0–1V).
* **Power Pins**:
  + **Vin**: Can be used to supply voltage to the board.
  + **3V3**: Provides 3.3V output.
  + **GND**: Ground pins.



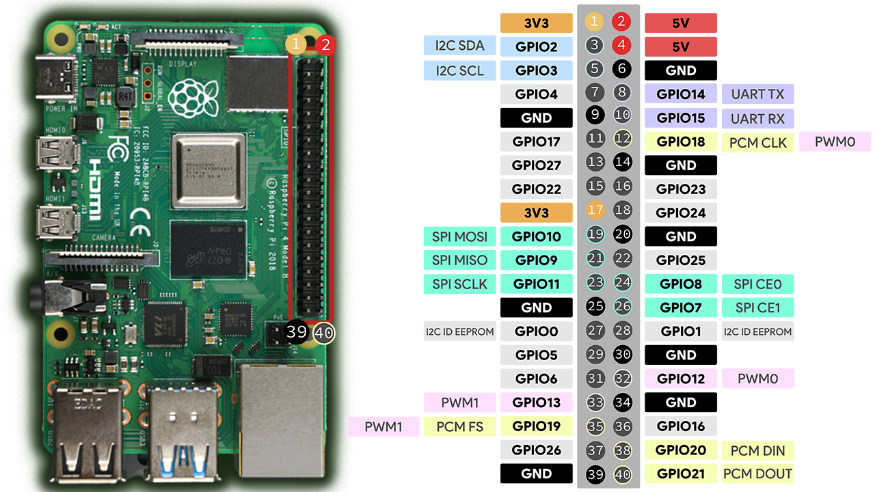
* + - 1. **Arduino Uno**
* **Digital I/O Pins (0–13):** Used for digital input or output. Pins 0 (RX) and 1 (TX) are also used for serial communication.
* **Analog Input Pins (A0–A5):** Used to read analog signals.
* **PWM Pins:** Pins 3, 5, 6, 9, 10, and 11 support Pulse Width Modulation.
* **Power Pins:**
  + **5V:** Provides regulated 5V output.
  + **3.3V:** Provides 3.3V output.
  + **GND:** Ground pins.
* **Reset Pin:** Used to reset the microcontroller.

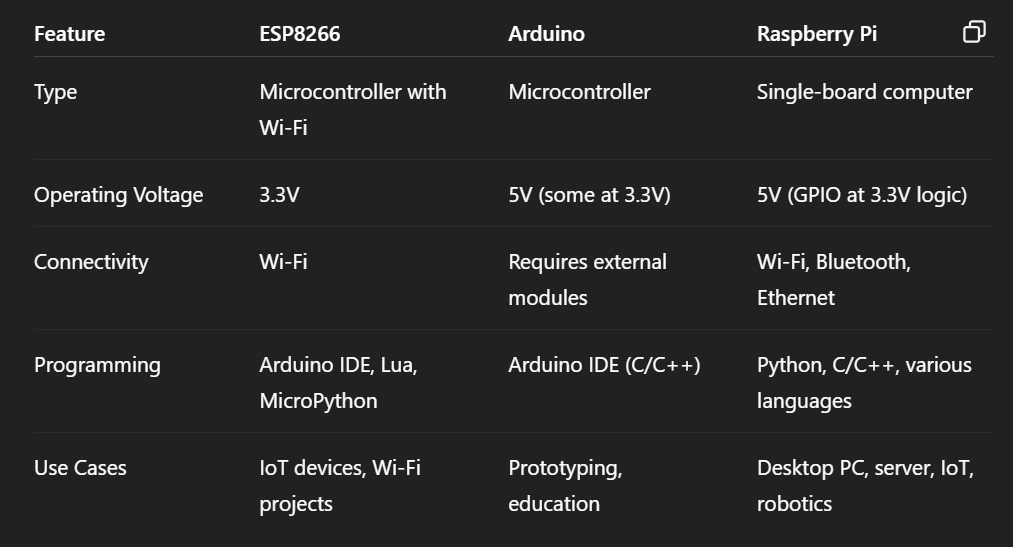


* + - * 1. **Raspberry Pi**

The Raspberry Pi (models B+, 2, 3, 4, and 5) features a 40-pin header that allows for interfacing with various electronic components. These pins are categorized based on their functions:

* **Power Pins**: Provide voltage to external components.
* **Ground Pins**: Serve as the common ground reference.
* **GPIO Pins**: Configurable as input or output for general-purpose use.
* **Special Function Pins**: Support specific protocols like I2C, SPI, UART, and PWM.



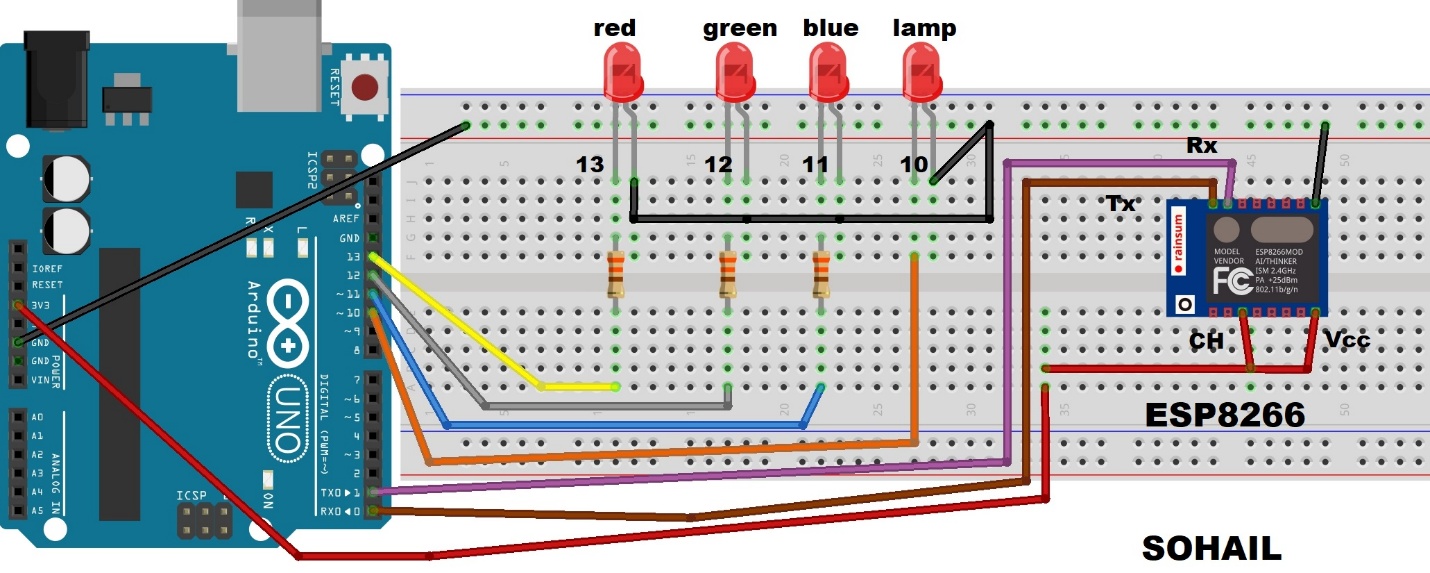


**Example Use Case: ESP8266 (NodeMCU) to control an LED via a web browser.**

**Components Needed:**

* **NodeMCU ESP8266** (e.g., ESP-12E)
* **LED** (any color)
* **220Ω Resistor**
* **Breadboard** and **Jumper Wires**
* **Micro USB Cable** (to connect NodeMCU to your computer)

**Circuit Diagram:**



**Connections:**

* **LED Anode (+)** → NodeMCU **D1** pin (GPIO5)
* **LED Cathode (-)** → Ground (GND) through a 220Ω resistor

**Programming the NodeMCU:**

1. **Install the Necessary Libraries:**
   * Open the Arduino IDE.
   * Go to **Sketch** → **Include Library** → **Manage Libraries**.
   * Search for and install the **ESP8266** board package.
2. **Select the NodeMCU Board:**
   * Go to **Tools** → **Board** → **NodeMCU 1.0 (ESP-12E Module)**.
3. **Write the Code:**

*#include <ESP8266WiFi.h>*

*const char\* ssid = "Your\_SSID";*

*const char\* password = "Your\_PASSWORD";*

*WiFiServer server(80);*

*void setup() {*

*pinMode(D1, OUTPUT);*

*Serial.begin(115200);*

*delay(10);*

*WiFi.begin(ssid, password);*

*while (WiFi.status() != WL\_CONNECTED) {*

*delay(1000);*

*Serial.println("Connecting to WiFi...");*

*}*

*server.begin();*

*Serial.println("Server started");*

*Serial.print("IP Address: ");*

*Serial.println(WiFi.localIP());*

*}*

*void loop() {*

*WiFiClient client = server.available();*

*if (!client) {*

*return;*

*}*

*String request = client.readStringUntil('\r');*

*client.flush();*

*if (request.indexOf("/LED=ON") != -1) {*

*digitalWrite(D1, HIGH);*

*} else if (request.indexOf("/LED=OFF") != -1) {*

*digitalWrite(D1, LOW);*

*}*

*client.print("HTTP/1.1 200 OK\r\n");*

*client.print("Content-Type: text/html\r\n");*

*client.print("\r\n");*

*client.print("<!DOCTYPE HTML>\r\n");*

*client.print("<html>\r\n");*

*client.print("<body>\r\n");*

*client.print("<h1>LED Control</h1>\r\n");*

*client.print("<p><a href=\"/LED=ON\">Turn On</a></p>\r\n");*

*client.print("<p><a href=\"/LED=OFF\">Turn Off</a></p>\r\n");*

*client.print("</body>\r\n");*

*client.print("</html>\r\n");*

*delay(1);*

*client.stop();*

*}*

1. **Upload the Code:**
   * Connect the NodeMCU to your computer via the micro USB cable.
   * Select the correct port from **Tools** → **Port**.
   * Click the **Upload** button in the Arduino IDE.

**Accessing the Web Server:**

1. **Find the NodeMCU's IP Address:**
   * After uploading the code, open the **Serial Monitor** in the Arduino IDE.
   * Set the baud rate to **115200**.
   * Note the IP address displayed in the Serial Monitor (e.g., 192.168.1.100).
2. **Control the LED:**
   * Open a web browser and enter the NodeMCU's IP address:

http://192.168.1.100

* + You should see a simple webpage with two links: "Turn On" and "Turn Off".
  + Clicking "Turn On" will light up the LED, and "Turn Off" will turn it off.

**IoT Security Challenges**

* Insecure interfaces (e.g., default credentials)
* Unencrypted data transmission
* Lack of firmware updates
* Physical tampering with devices

Mitigations:

* TLS/SSL, firmware signing, authentication, secure boot