MULTIMEDIA UNIVERSITY

Unleashing the Power of IoT with ESP32

Dr. Tan Wooi Haw
Faculty of Engineering
Multimedia University

Learning Objectives

- Introduces students to the world of IoT with ESP32.
- Provides hands-on experience with real-world applications.
- Encourages problem-solving through hands-on exercises.

What is ESP32?

- A small and low-cost computing device that can connect to the internet and communicate wirelessly.
- Developed by Espressif Systems, a company specializing in wireless communications.
- Built-in Wi-Fi and Bluetooth for wireless connectivity.
- Various input and output pins for connecting sensors and actuators.
- Supports multiple programming languages, including Arduino and MicroPython.

Benefits for IoT Projects

- Easy to connect to the internet for remote monitoring and control.
- Can interface with a wide range of sensors and devices.
- Ideal for creating smart home devices, wearable electronics, and other IoT applications.
- Common Uses:
  - ✓ Home automation systems (e.g. controlling lights and appliances).
  - ✓ Environmental monitoring (e.g. such as measuring temperature and humidity).
  - ✓ Wearable devices (e.g. fitness trackers).

ESP32 Development Boards

- The ESP32 family consists of several variants, each with specific features tailored to different applications.
- Below are some embedded development boards based on the ESP32 module.

DOIT ESP32 DEVKIT V1

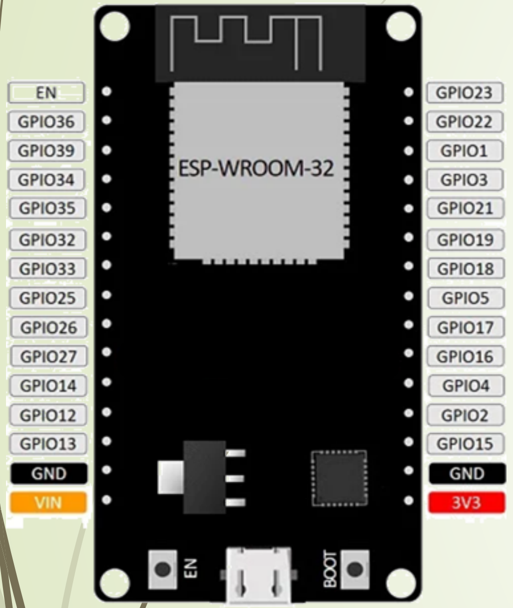
ESP32S NodeMCU

ESP32 Thing

WEMOS LOLIN32

Huzzah32

DOIT ESP32 Devkit V1



- DOIT ESP32 Devkit V1 operates at 3.3V and is powered through a micro-USB connection at 5 V or directly on the 3.3V VIN pin.
- The board has a total of 25 general purpose input/output (GPIO) pins that can be connected to other electronic devices.
- Programs can be written to:
  - ✓ control digital output devices such as LEDs, relays, or actuators.
  - ✓ read digital input devices such as buttons, switches, or digital sensors.
  - ✓ read analog sensors such as light sensors (LDR), or potentiometers.
  - ✓ control analog output devices such as motors or dimmable lights.

General Purpose Input/Output (GPIO)

- Not all GPIO pins can be used freely:

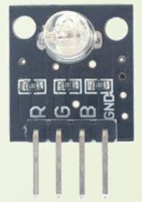
GPIO	Input	Output	Notes
1	TX pin	OK	debug output at boot
2	OK	OK	connected to on-board LED
3	OK	RX pin	HIGH at boot
4	OK	OK	
5	OK	OK	outputs PWM signal at boot
12	OK	OK	boot fail if pulled high
13	OK	OK	
14	OK	OK	outputs PWM signal at boot
15	OK	OK	outputs PWM signal at boot
16	OK	OK	
17	OK	OK	
18	OK	OK	
19	OK	OK	
21	OK	OK	
22	OK	OK	
23	OK	OK	
25	OK	OK	
26	OK	OK	
27	OK	OK	
32	OK	OK	
33	OK	OK	
34	OK	-	input only
35	OK	-	input only
36	OK	-	input only
39	OK	-	input only

Hardware Connections (1)

- ESP32: To serve as the tiny computer for programming and interfacing with various sensors and components.
- ESP32 baseboard: To provide an easy way of connecting additional sensors and components to the ESP32.
- LED: To demonstrate digital output control.
- Push-button switch: To allow user interactions.
- Potentiometer: To provide variable analog input.

ESP32

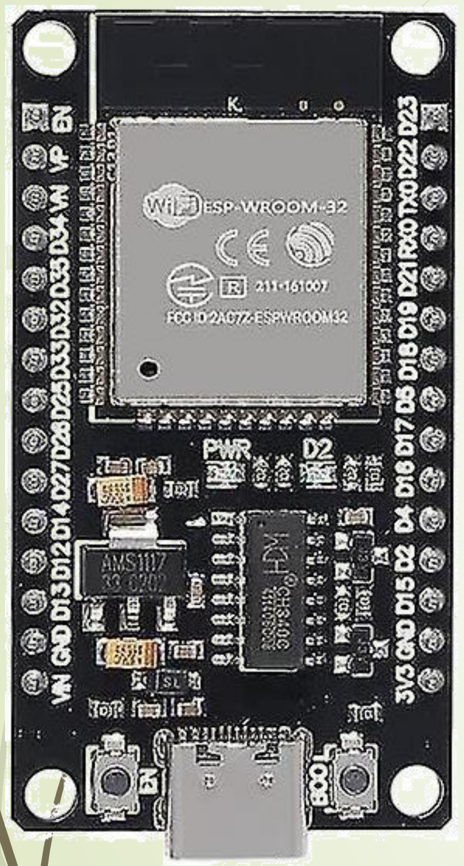
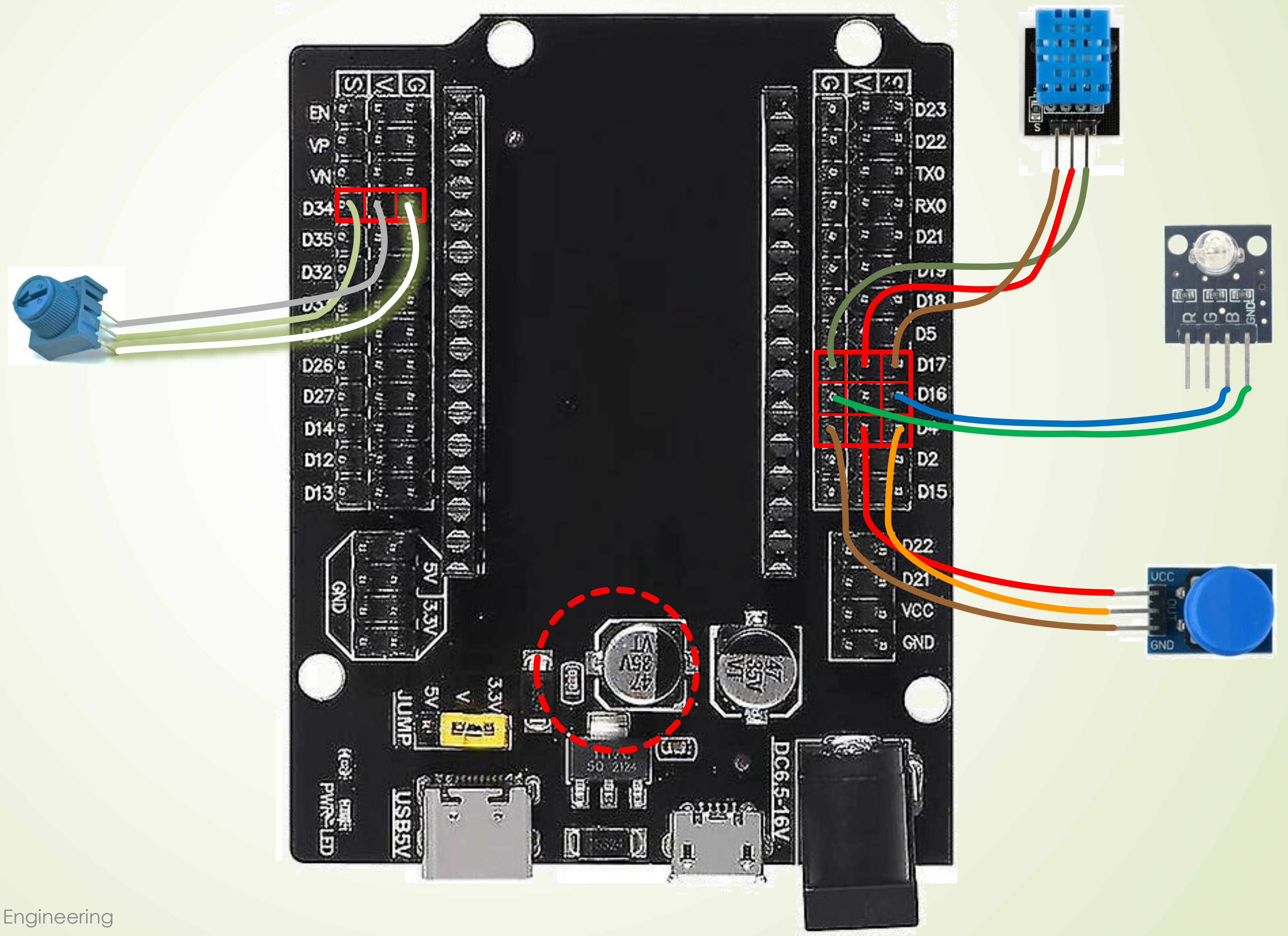
ESP32 baseboard


LED

Push-button switch

Potentiometer

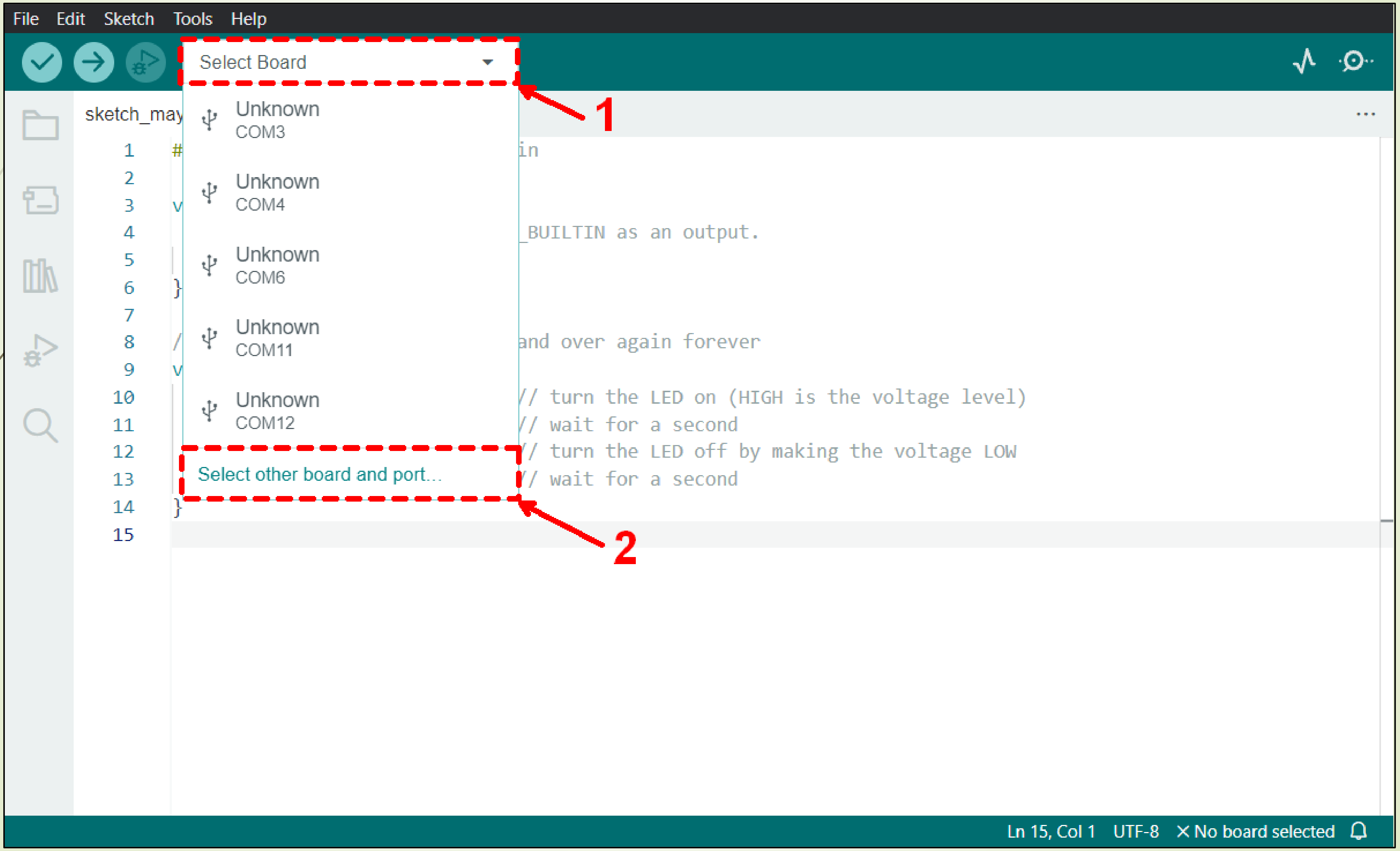
Hardware Connections (2)

Arduino IDE (1)

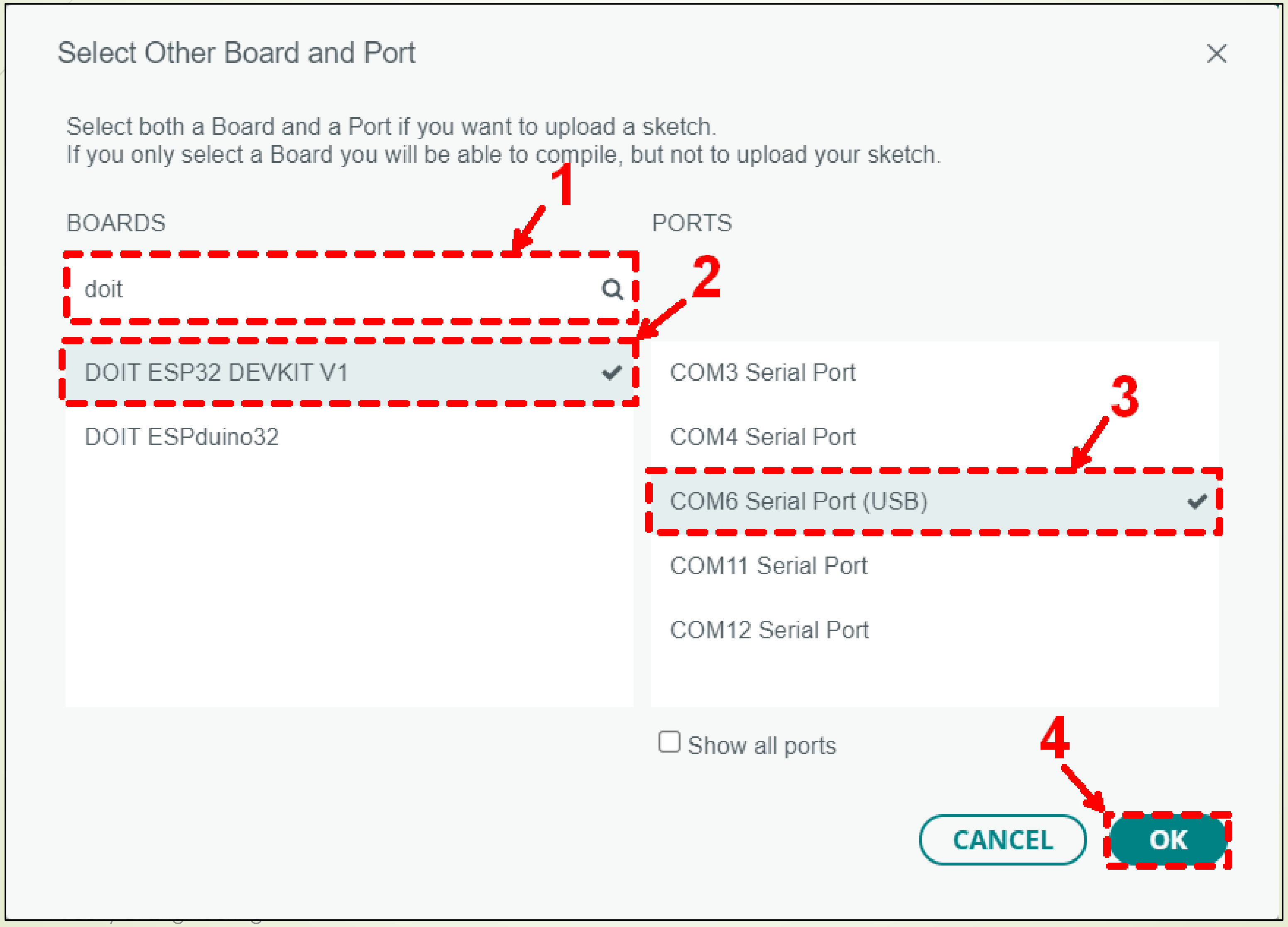
- Arduino is an open-source electronics ecosystem that combines user-friendly hardware and software.
- The open-source Arduino IDE allows users to write and upload code to compatible boards, including the ESP32.
- It provides a simple, user-friendly interface for programming Arduino compatible boards.
- ESP32 is compatible with a wide range of Arduino libraries, making it easy to interface with sensors, displays, actuators, and other hardware.
- When programming using the Arduino IDE, a simplified version of C++ is used.

Arduino IDE (2)



```
#                                   in

v

                                    _BUILTIN as an output.

}

/                                   and over again forever
v
                                    // turn the LED on (HIGH is the voltage level)
                                    // wait for a second
                                    // turn the LED off by making the voltage LOW
                                    // wait for a second
}
```

Arduino IDE (3)



Select Other Board and Port

Select both a Board and a Port if you want to upload a sketch.
If you only select a Board you will be able to compile, but not to upload your sketch.

Arduino Sketch

- An Arduino program is known as a sketch.
- When a new sketch is created in Arduino, there are two empty functions, **setup()** and **loop()**.

```
void setup() {
  // Initialization code goes here
}

void loop() {
  // Main program code goes here
}
```

Typically runs once when power is first supplied to the board or after a system reset

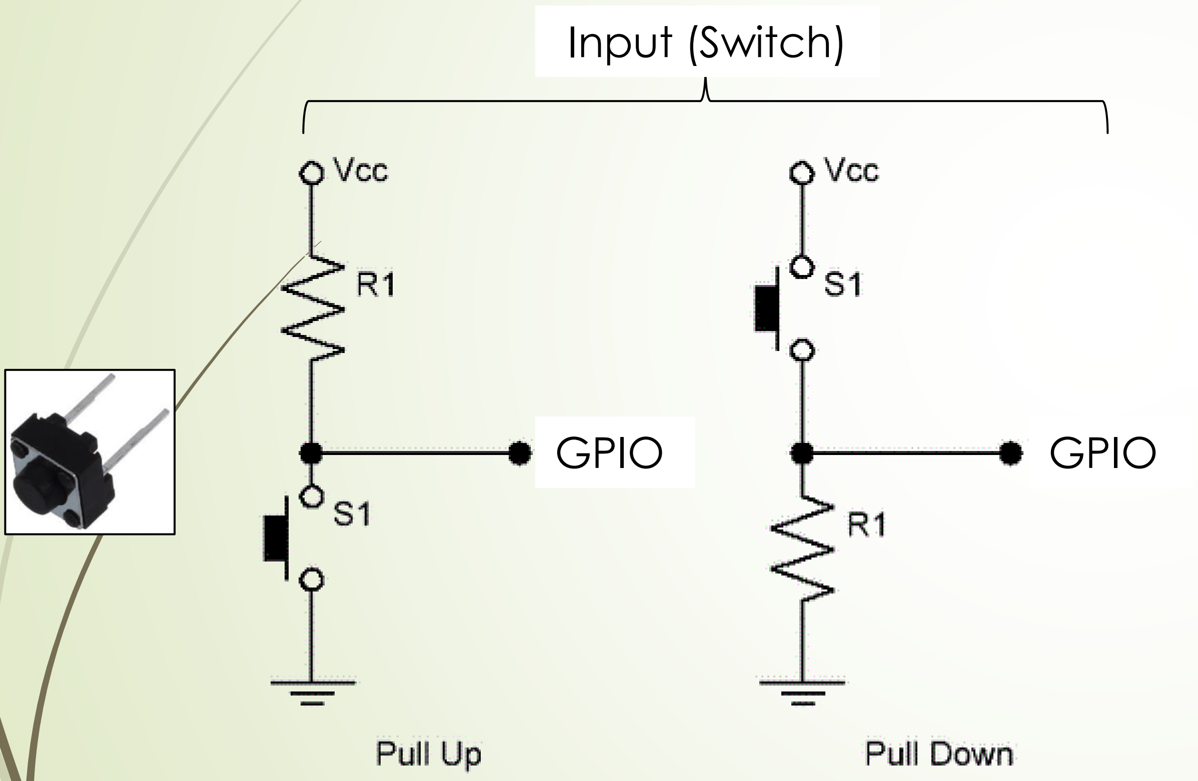
Repeatedly execute a set of instructions, allowing the board to continuously perform tasks or respond to inputs

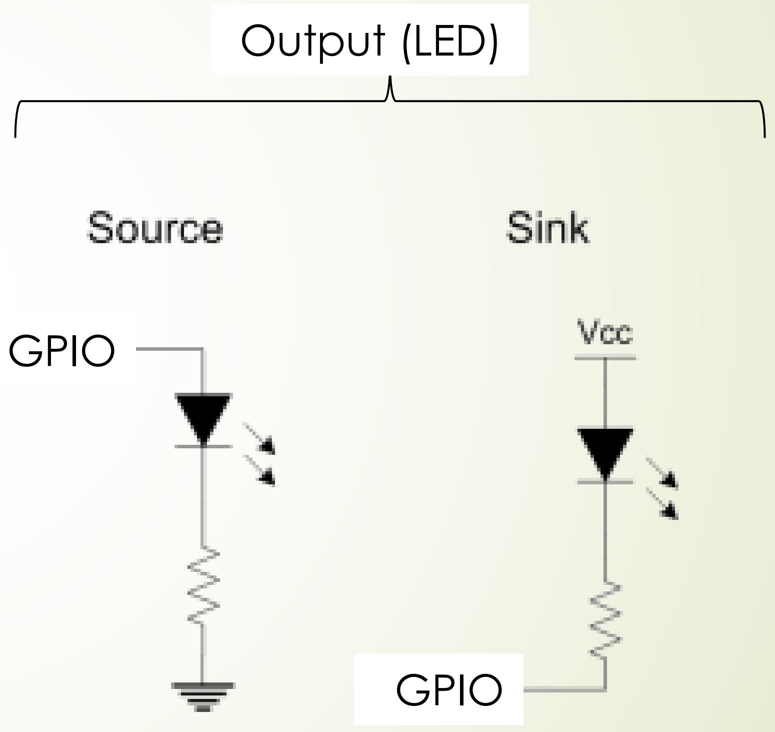
Digital Input / Output (1)

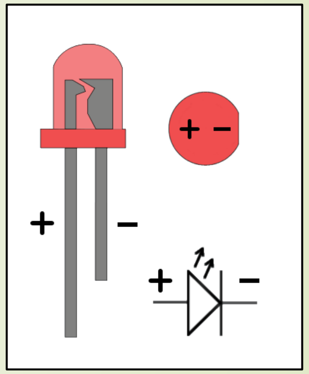
- The Arduino IDE provides built-in functions for both digital input and output.
- The following functions are provided for digital input and output.

Function name	Description
pinMode(pin, mode)	Configures the specified pin to behave either as an input or output. pin: the GPIO pin number to set the mode of. mode: INPUT, OUTPUT
digitalRead(pin)	Reads the value from a specified GPIO pin, either HIGH or LOW. pin: the GPIO pin number.
digitalWrite(pin, value)	Write a HIGH or a LOW value to a GPIO pin. pin: the GPIO pin number. value: HIGH or LOW.

Digital Input / Output (2)







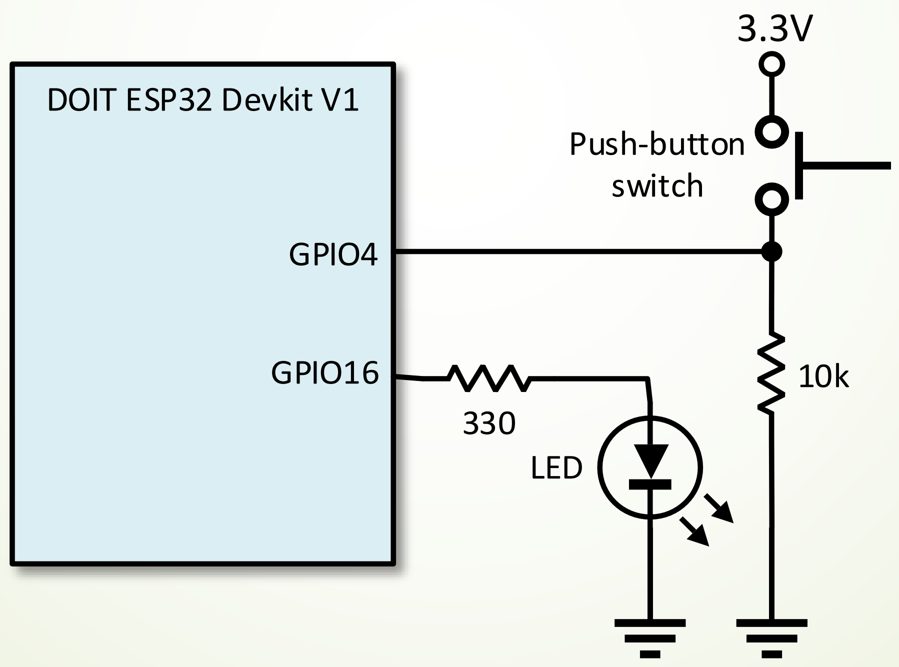
Digital Input / Output (3)

- The following functions can be used to create delay:

Function name	Description
delay(ms)	Pause execution for specified number of milliseconds
delayMicroseconds(micros)	Pause execution for specified number of microseconds

Digital Input / Output (4)

- An LED and a push-button switch are connected to DOIT ESP32 Devkit V1 board as shown below.



Digital Input / Output (5)

- The following Arduino sketch blinks the LED:

```
#define ledPin 16  // the LED pin

void setup() {
  // initialize ledPin as an output.
  pinMode(ledPin, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {
  digitalWrite(ledPin, HIGH); // turn the LED on (HIGH is the voltage level)
  delay(1000);                // wait for a second
  digitalWrite(ledPin, LOW);  // turn the LED off by making the voltage LOW
  delay(1000);                // wait for a second
}
```

Digital Input / Output (6)

- The following Arduino sketch reads the push-button to turn off/on the LED:

```
#define buttonPin 4 // the pushbutton pin
#define ledPin 16   // the LED pin

int buttonState = 0; // variable for storing the pushbutton status

void setup() {
  pinMode(buttonPin, INPUT); // initialize the pushbutton pin as an input
  pinMode(ledPin, OUTPUT); // initialize the LED pin as an output
}

void loop() {
  buttonState = digitalRead(buttonPin); // read the state of the pushbutton value
  digitalWrite(ledPin, buttonState); // turn LED on
}
```

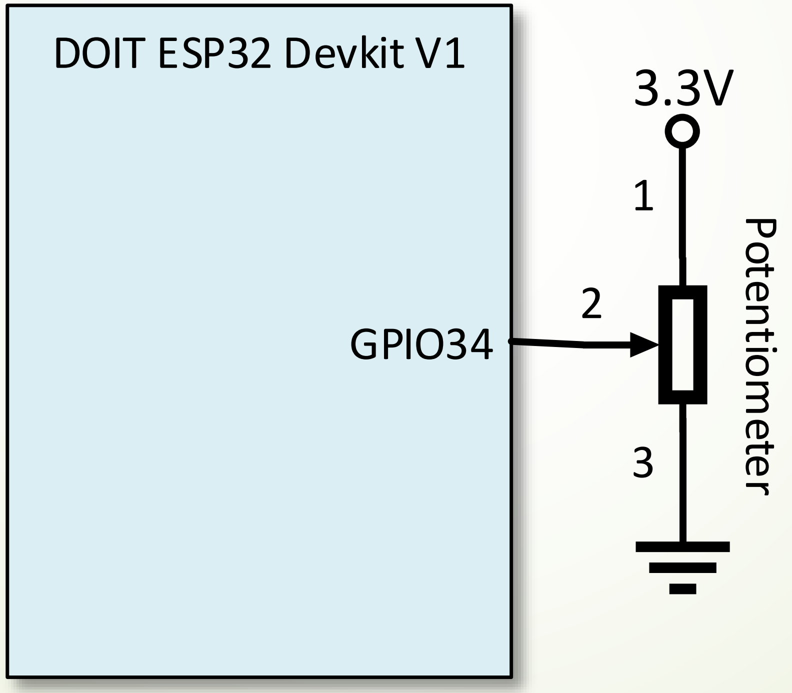
Analog Input (1)

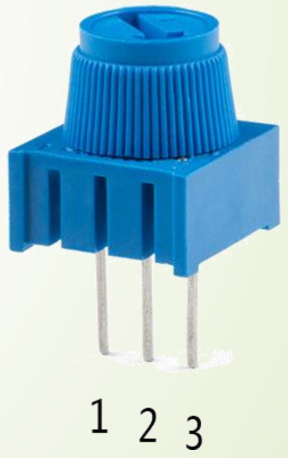
- The analog input of the ESP32 allows it to read varying voltage levels from sensors or other devices.
- The ESP32 has several analog-to-digital converter (ADC) pins that convert these analog signals into digital values for processing.
- The 12-bit ADC converts a voltage, between 0 and 3.3 V, on an analog input pin to a digital value between 0 and 4095.
- The following function is provided for analog input:

Function name	**Description**
analogRead(ADCpin)	Reads the analog value on the specified ADC pin and returns a value between 0 and 4095 representing the voltage on the pin. The ADCpin parameter specifies the analog input pin to read.

Analog Input (2)

- A potentiometer to DOIT ESP32 Devkit V1 board as shown below:





Analog Input (3)

- The following Arduino sketch reads the voltage value from the middle pin of a potentiometer and print it to the serial monitor:

```
#define potPin 34 // middle pin of potentiometer
int potValue = 0; // variable for storing the potentiometer value

void setup() {
  Serial.begin(115200); // initializes the serial communication
  delay(1000); // delays one second
}

void loop() {
  // Reading potentiometer value
  potValue = analogRead(potPin); // reads the analog value
  Serial.println(potValue); // prints the potentiometer value to serial monitor
  delay(500);  // delays 0.5 second
}
```

Sensors and Actuators (1)

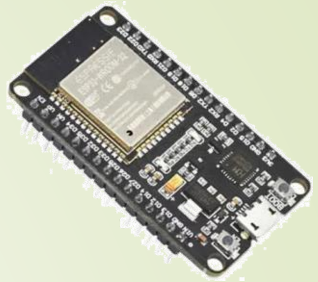
ESP32: To serve as the tiny computer for programming and interfacing with various sensors and components.

ESP32 baseboard: To provide an easy way of connecting additional sensors and components to the ESP32.

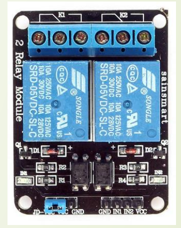
Relay: To control high power devices.

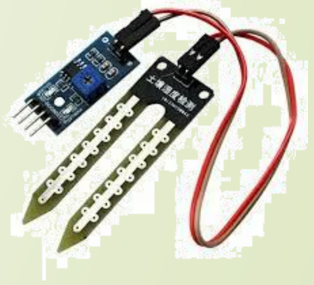
Soil moisture sensor: To provide soil moisture level.

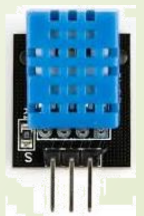
DHT11 sensor: To measure ambient temperature and humidity.


ESP32

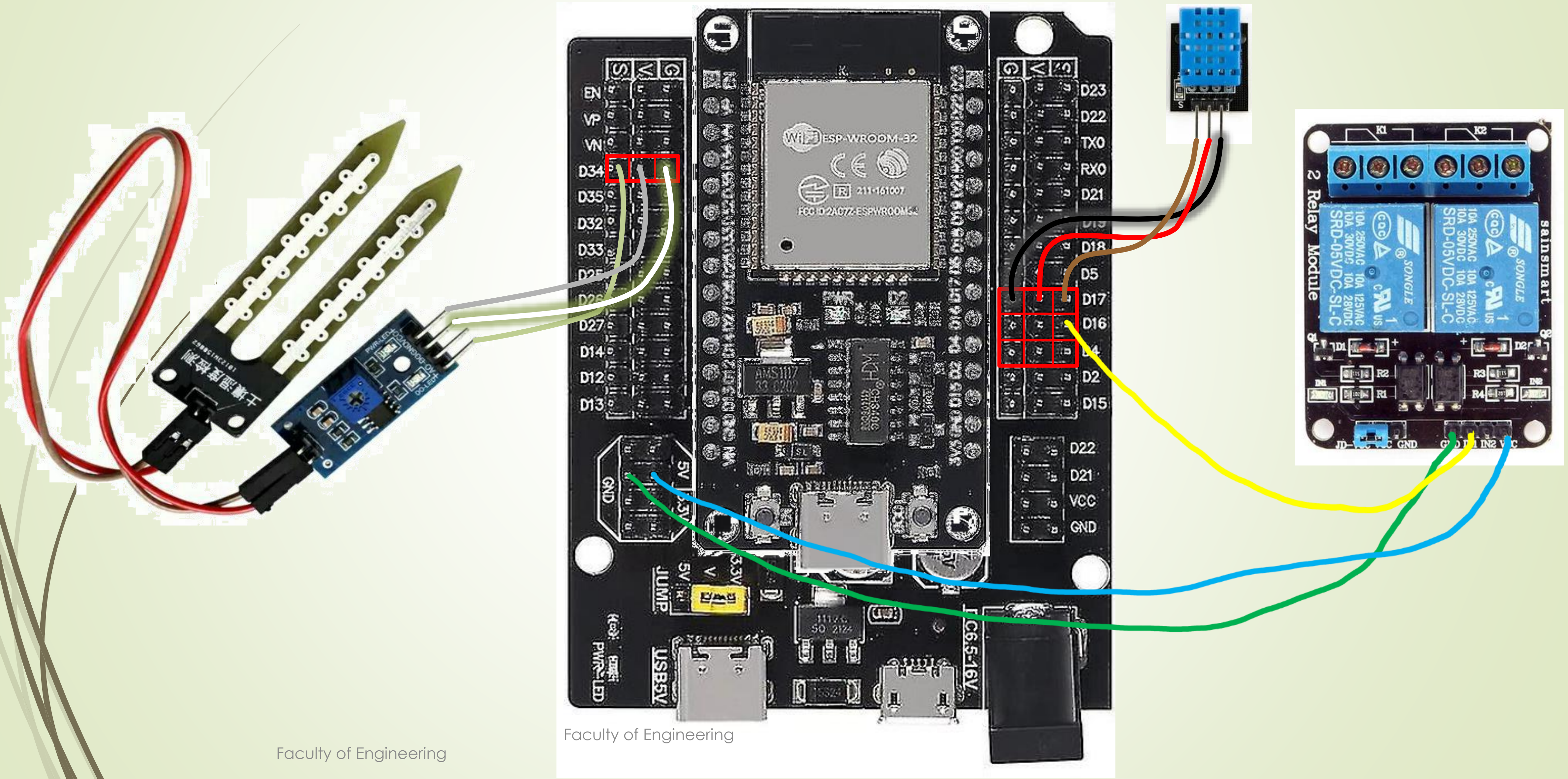
ESP32 baseboard


Relay


Soil moisture sensor


DHT11 sensor

Sensors and Actuators (2)

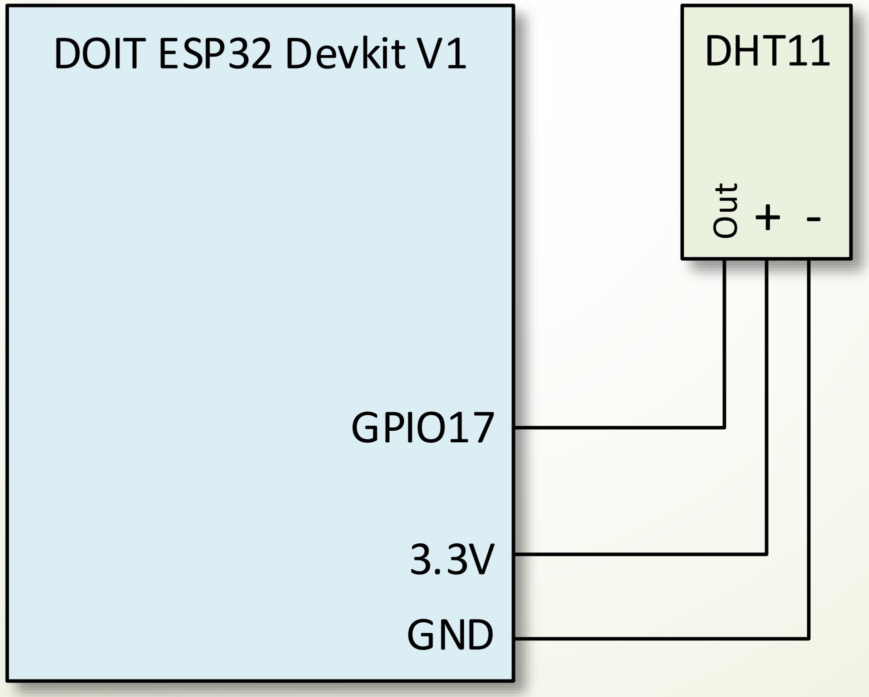


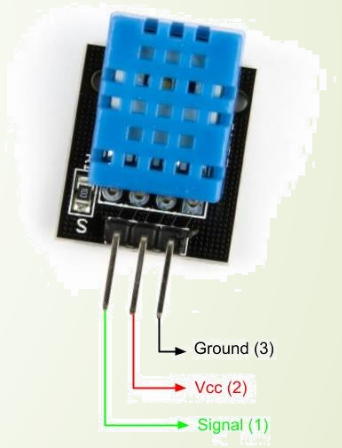
Humidity & Temperature Sensor (1)

- DHT11 is a commonly used temperature and humidity sensor.
- This sensor provides digital output data and does not require an analog input pin for reading values.
- It is capable of measuring temperature from 0 to 50 degrees Celsius with a 2% accuracy, and humidity from 20 to 80% with a 5% accuracy.
- The downside is the sampling rate cannot be more than 1 HZ (once every second).
- There are many installable Arduino libraries for the DHT sensors.
- It is recommended to install “DHT sensor library” by Adafruit.

Humidity & Temperature Sensor (2)

- A DHT11 sensor module is connected to the ESP32 board as shown below:





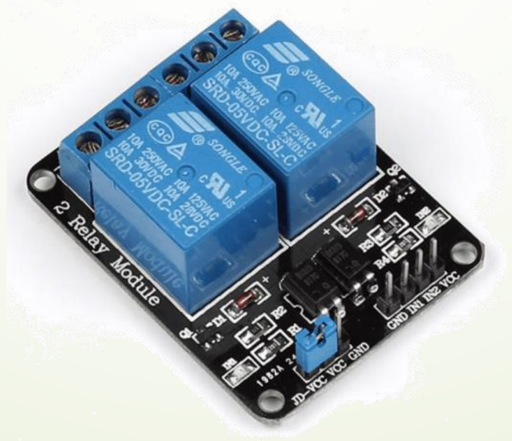
Humidity & Temperature Sensor (3)

- The following Arduino sketch reads the DHT11 sensor and print the temperature and humidity:

```
#include "DHT.h"
#define DHTPIN 17     // Digital pin connected to the DHT sensor
#define DHTTYPE DHT11   // DHT 11

// Create a DHT object.
DHT dht(DHTPIN, DHTTYPE);

void setup() {
  dht.begin(); // Initialize the DHT sensor
  Serial.begin(115200); // Initialize the serial communication
}

void loop() {
  float h = dht.readHumidity();
  float t = dht.readTemperature();
  // Check if any reads failed and exit early (to try again).
  if (isnan(h) || isnan(t)) {
    Serial.println("Failed to read from DHT sensor!");
    return;
  }
  // Print the humidity and temperature to the serial port
  Serial.printf("Humidity: %.2f, Temperature: %.2f\n", h, t);
  delay(2000);
}
```

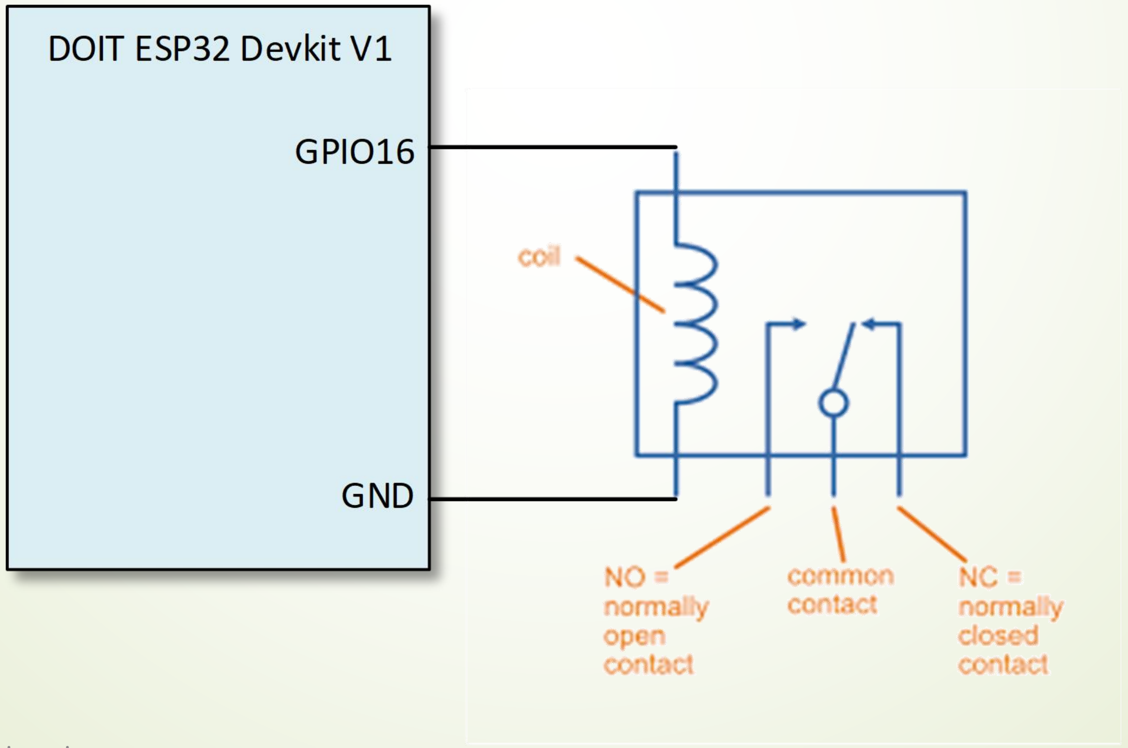
Relay (1)

- A relay can control high-voltage devices like lights and appliances using low-voltage signals from a microcontroller.
- Below is a 2-channel relay module with two relays, each capable of switching up to 10A at 250VAC or 30VDC.
- The module is active low, meaning it activates when it receives a low signal.
- It is commonly used in home automation projects and can be easily interfaced with microcontrollers like Arduino or ESP32.



Relay (2)

- An active low relay module is connected to the ESP32 board as shown below:



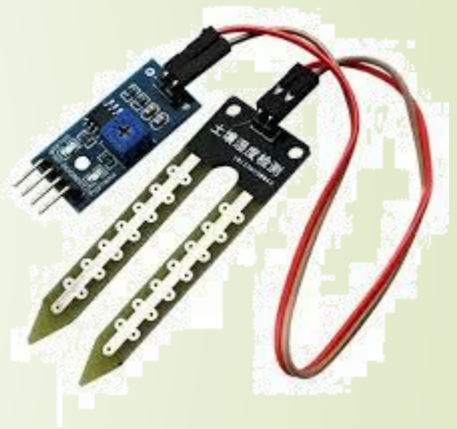
Relay (3)

- The following Arduino sketch activates the relay for 2s after every 30s:

```
#define relayPin 16  // the relay pin

void setup() {
  // initialize relayPin as an output.
  pinMode(relayPin, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {
  digitalWrite(relayPin, HIGH); // deactivate the relay
  delay(30000);                 // wait for 30 seconds
  digitalWrite(relayPin, LOW);  // activate the relay
  delay(2000);                  // wait for 2 seconds
}
```

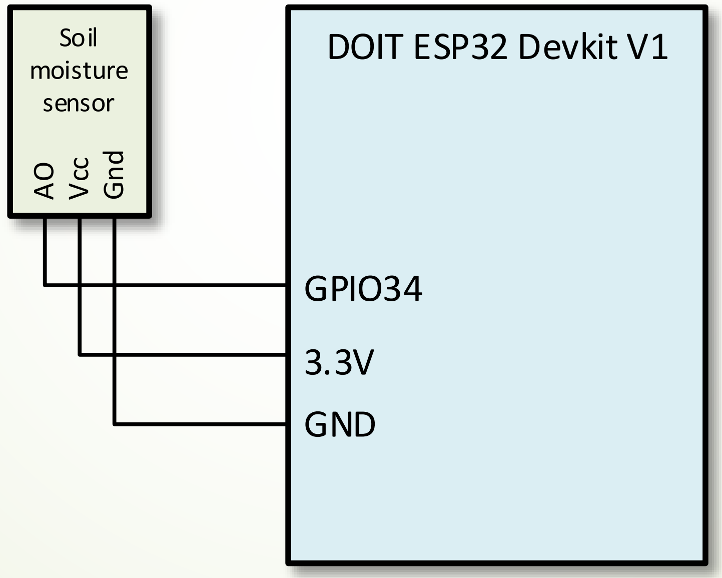
Soil Moisture Sensor (1)

- A soil moisture sensor measures the moisture level in soil to help monitor and manage water needs for plants.
- The sensor consists of two parts: sensor probes and a module board.
- The sensor uses two probes that are inserted into the soil and electric current is passed through the soil using the two probes.
- It then reads the resistance to determine the moisture level.
- Water in the soil increases conductivity (less resistance), making it easier for electricity to flow.
- Dry soil has poor conductivity (more resistance), making it harder for electricity to flow.



Soil Moisture Sensor (2)

- A soil moisture sensor is connected to the ESP32 board as shown below:



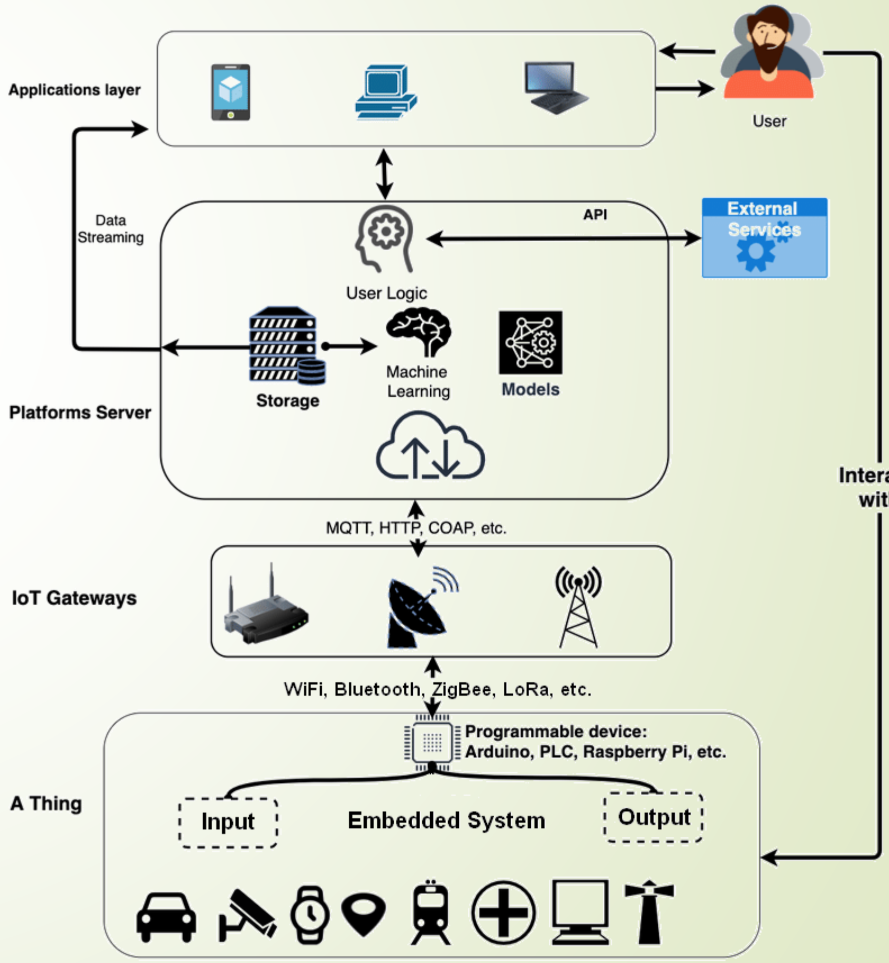
Soil Moisture Sensor (3)

- The following Arduino sketch reads the analog output of the soil moisture sensor and print the corresponding digital value (0 – moistest, 4095 – dryest).

```
#define soilPin 34 // AO pin of soil moisture sensor
int moisture = 0; // variable for storing the moisture value

void setup() {
  Serial.begin(115200); // initializes the serial communication
  delay(1000); // delays one second
}

void loop() {
  // Reading potentiometer value
  moisture = analogRead(soilPin); // reads the analog value
  Serial.println(moisture); // prints the moisture value to serial monitor
  delay(1000);  // delays 1 second
}
```

End-to-end IoT System

- Embedded system includes programmable devices like Arduino, PLC (Programmable Logic Controller), Raspberry Pi, etc.
- They collects data from various Input devices such as sensors (e.g., temperature sensors, motion detectors, GPS, etc.).
- Output devices such as actuators, displays, or any other device are controlled to perform actions based on the data received or processed.

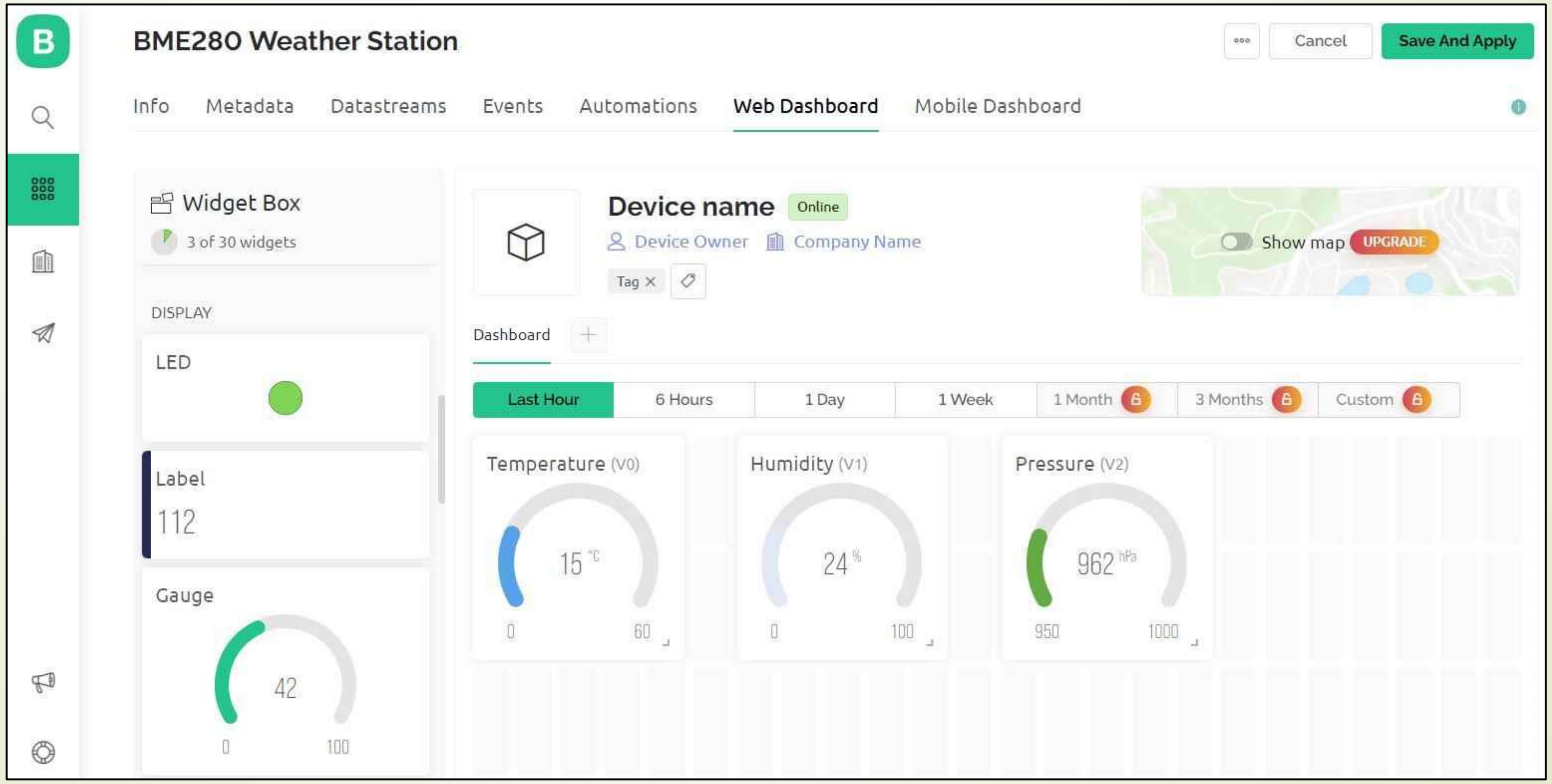


Blynk IoT Platform

- Blynk (https://blynk.cloud) is a popular platform for building and managing IoT projects efficiently.
- It simplifies connecting hardware to the internet and offers remote monitoring and control via a user-friendly interface.
- Blynk Cloud provides robust cloud services for seamless communication between hardware and the Blynk mobile app.
- It ensures secure data exchange, scalability, reliability, and multiple device connections, suitable for both small and large projects.
- Blynk App allows users to create custom dashboards for monitoring and controlling devices.
- It offers widgets like buttons and graphs for easy interface design, enabling real-time interaction with IoT devices from anywhere.

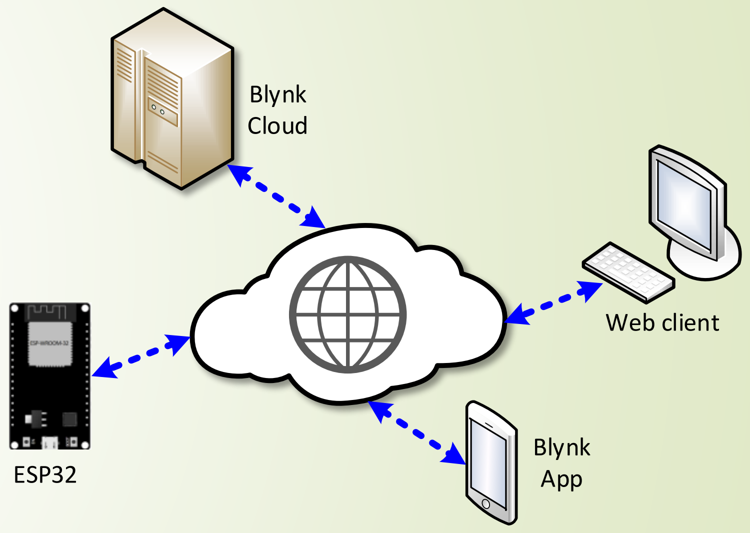
Blynk Dashboard

- A customizable interface that enables users to monitor and control IoT devices through web and mobile applications.



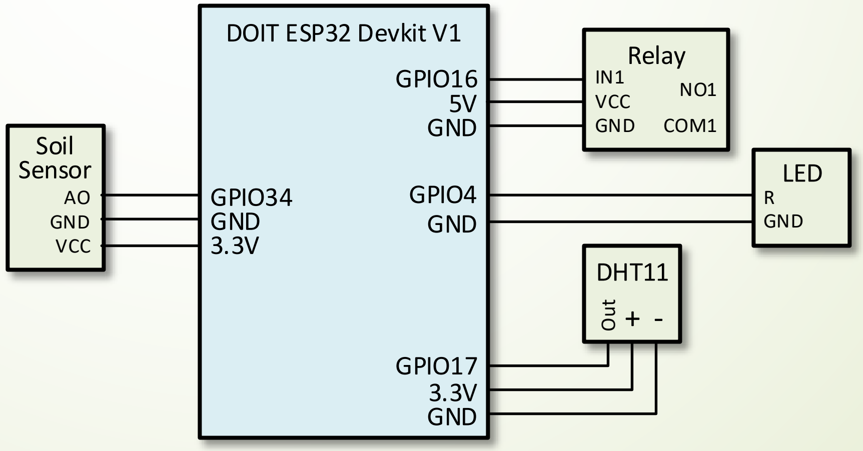
An ESP32 IoT System

- The figure on the right illustrates the ESP32 microcontroller's connectivity to the Blynk server via the cloud.
- This connection facilitates the remote monitoring and control of farming conditions through the Blynk app and a web client.
- The Blynk platform acts as an intermediary, allowing data flow from the ESP32 to the user's mobile device.
- The web interface, enabling users to observe real-time data and control the devices remotely.



Hands-on

- Improve the system by including the following features:
  - Add an LED widget on the Blynk dashboard; turn on the LED when the soil is dry and turn it off when the soil is moist.
  - Connect to the LED module; turn on the LED when the soil is dry and turn it off when the soil is moist.



Thank you