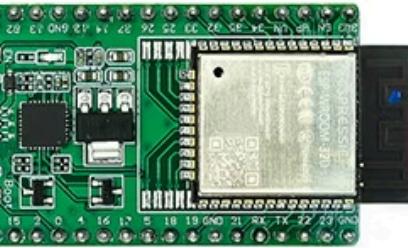
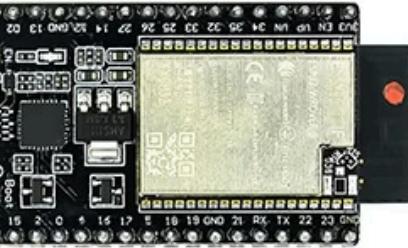


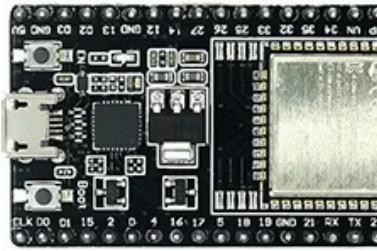
WROOM-32D



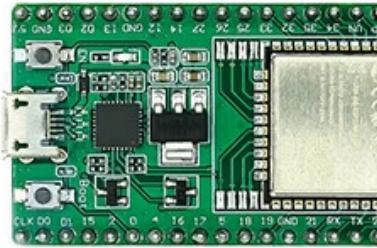
WROOM-32D



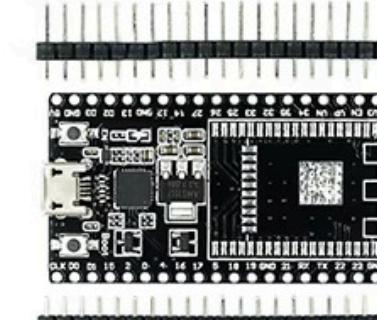
WROVER



WROOM-



WROOM-



ESP32-DEVKITC

# Introducción al ESP32

El ESP32 es un microcontrolador de bajo costo y alto rendimiento que permite la conexión a Internet y el control de diversos dispositivos electrónicos. En este proyecto, utilizaremos el ESP32 para crear un sistema de domótica que incluye diversas funcionalidades como la conexión a Internet, el control de periféricos y la comunicación con sensores.



3 colaboradores

# Instalación del ESP32 en Windows

## 1 Descarga e Instalación del Driver

Es necesario descargar e instalar el driver del ESP32 desde el siguiente [enlace](#).

## 2 Agregar Board Managers al IDE de Arduino

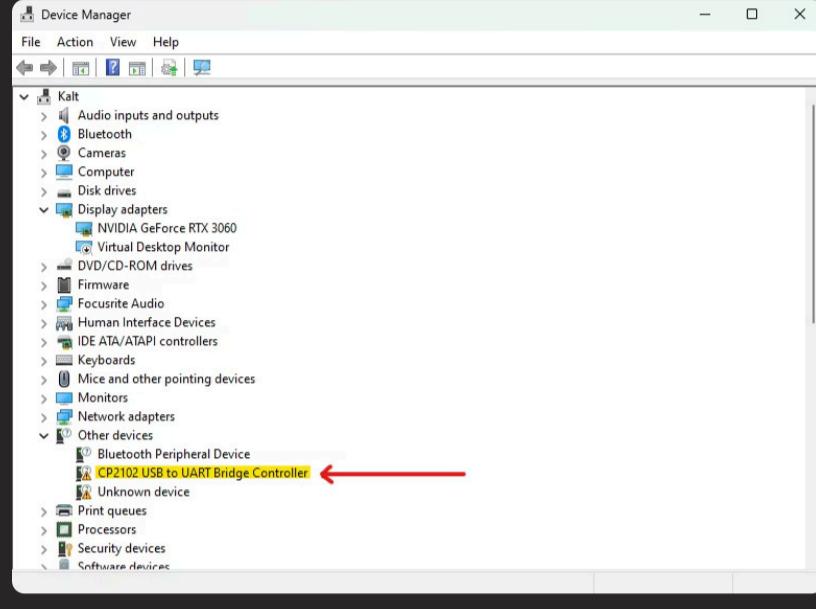
Desde las preferencias del IDE de Arduino, agregue los board managers del ESP32 utilizando el siguiente [enlace](#).

## 3 Agregar Tarjeta ESP32 de ESPRESSIF

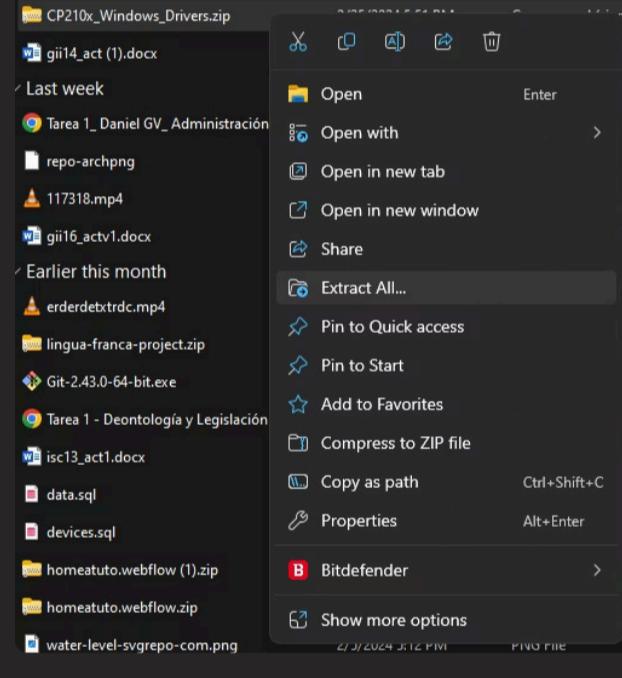
Finalmente, agregue la tarjeta ESP32 de ESPRESSIF en el board manager del IDE.

### Descarga e Instalación del Driver

Para que nuestra computadora pueda reconocer la tarjeta ESP32 de la manera correcta debemos instalar el driver correspondiente. Antes de instalar el driver, el ESP32 se verá de la siguiente manera en el administrador de dispositivos:



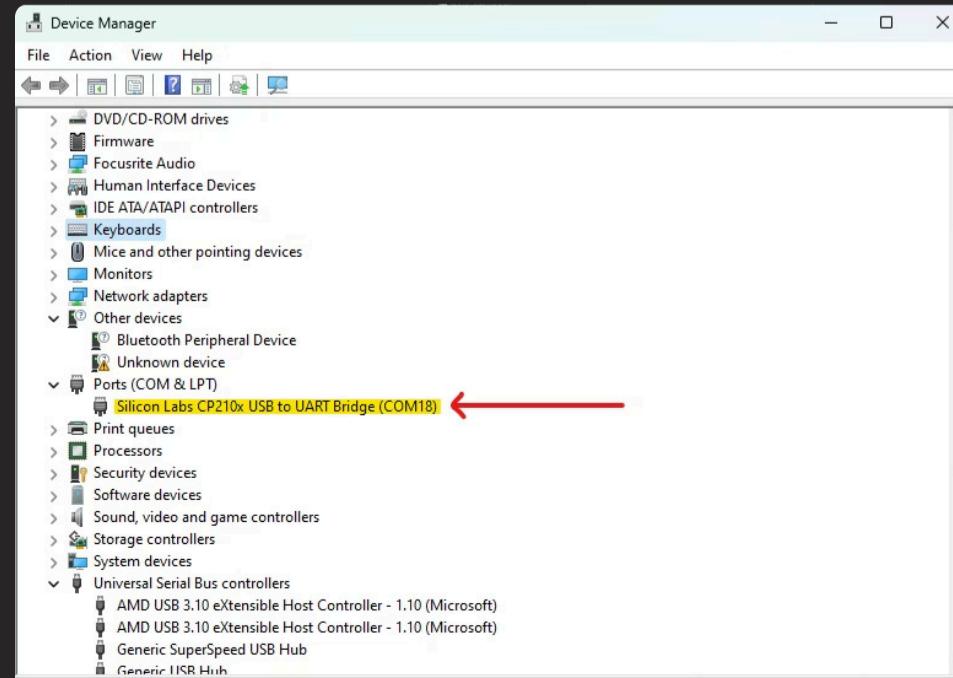
- Al dar click en el enlace del driver → [link](#) se les descargará la carpeta llamada "CP210x\_Windows\_Drivers.zip"
- Van a darle click derecho y extraer todo.



- Dentro de la carpeta deberán buscar la opción que corresponda a su sistema, x86 para 32 bits y x64 para 64 bits. En nuestro caso utilizaremos la versión para 64 bits:

CP210xVCPInstaller_x64.exe	2/25/2024 8:16 PM	Application	1,026 KB
CP210xVCPInstaller_x86.exe	2/25/2024 8:16 PM	Application	903 KB
dpinst.xml	2/25/2024 8:16 PM	xmlfile	12 KB
SLAB_License_Agreement_VCP_Windows...	2/25/2024 8:16 PM	Text Document	9 KB
slabvcp.cat	2/25/2024 8:16 PM	Security Catalog	11 KB
slabvcp.inf	2/25/2024 8:16 PM	Setup Information	8 KB
v6-7-6-driver-release-notes.txt	2/25/2024 8:16 PM	Text Document	16 KB
x64	2/25/2024 8:16 PM	File folder	
x86	2/25/2024 8:16 PM	File folder	

- Sigan los pasos de instalación y al terminar el dispositivo debería ser reconocido de la siguiente manera:

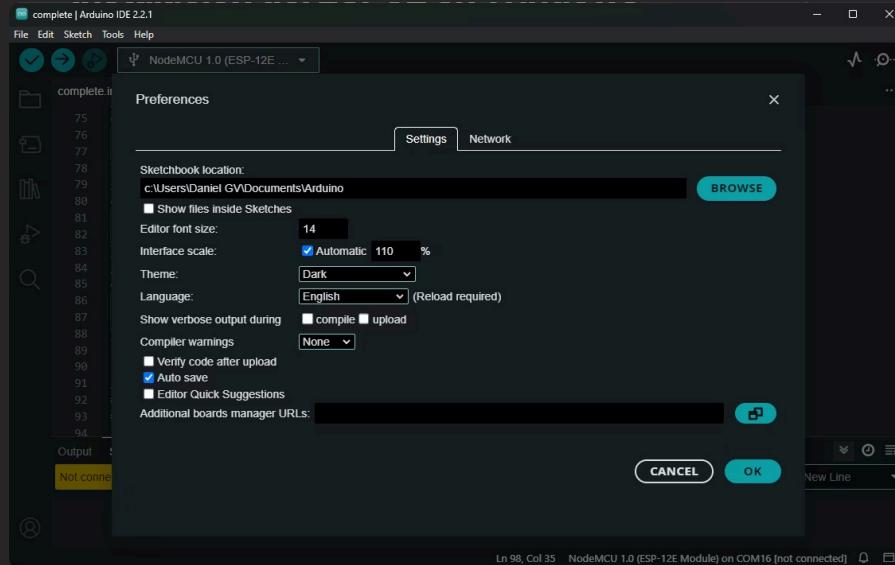


# Agregar Board Managers al IDE de Arduino

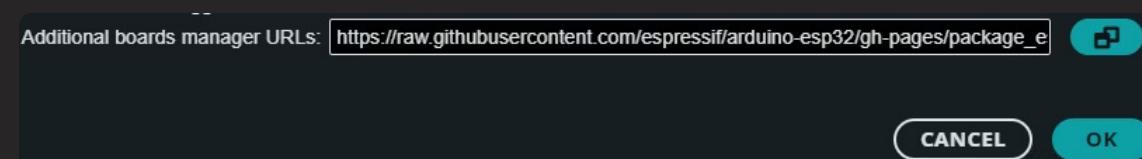


El entorno de desarrollo que utilizaremos para programar la ESP32 será el IDE de Arduino. Para que esta pueda reconocer la placa primero debemos de instalarle la paquetería de placas de espressif.

1. Empezaremos copiando el link de la paquetería → [https://raw.githubusercontent.com/espressif/arduino-esp32/gh-pages/package\\_esp32\\_index.json](https://raw.githubusercontent.com/espressif/arduino-esp32/gh-pages/package_esp32_index.json) ←
2. Abrimos la IDE de Arduino y nos vamos a → Archivo → Preferencias.



3. En la sección que dice "Additional boards manager URLs" insertaremos el link de la paquetería y damos click en OK:

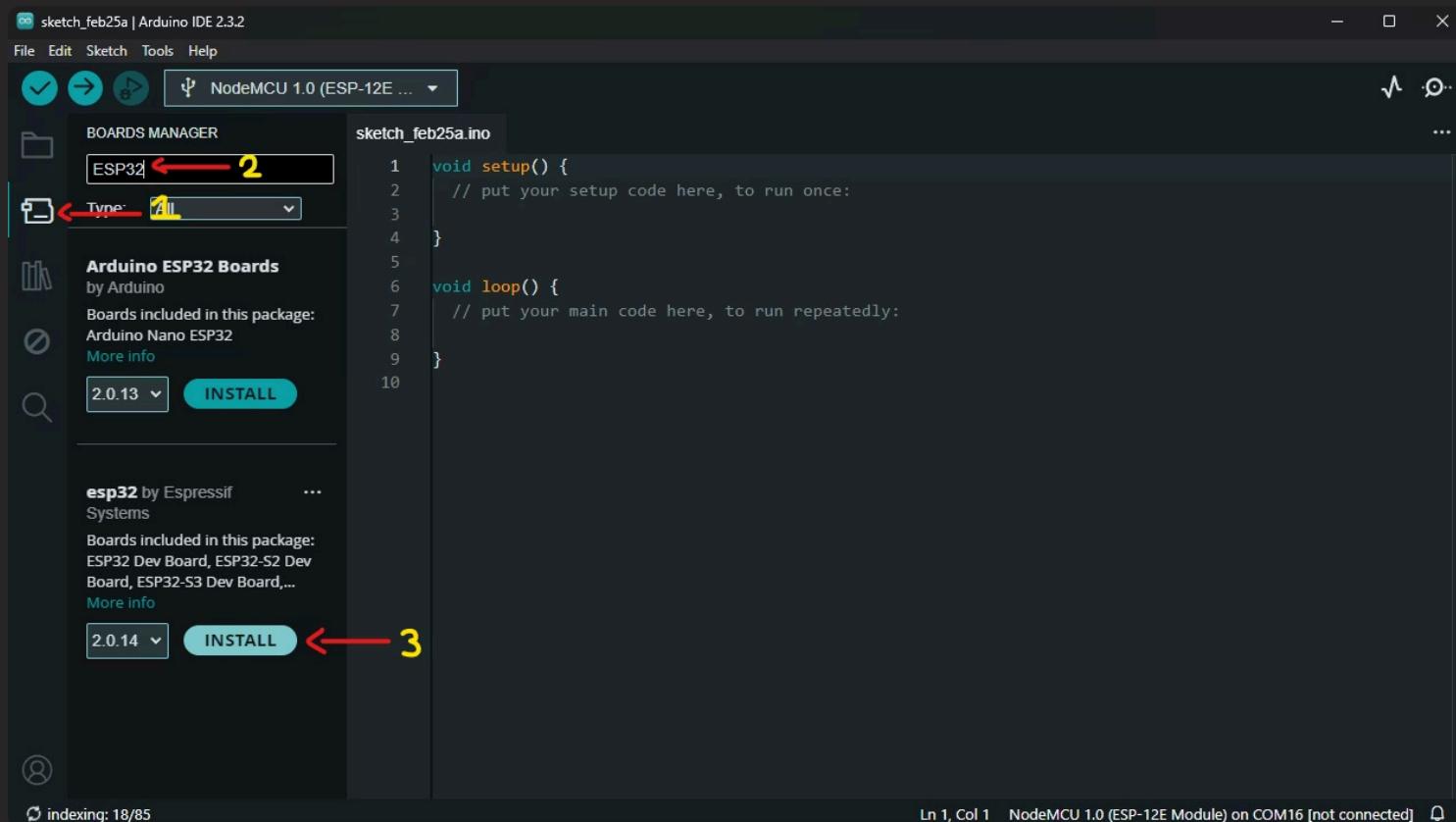


# Agregar Tarjeta ESP32 de ESPRESSIF



Ya como último paso para que el IDE de Arduino pueda cargar el programa a nuestra tarjeta debemos instalarla.

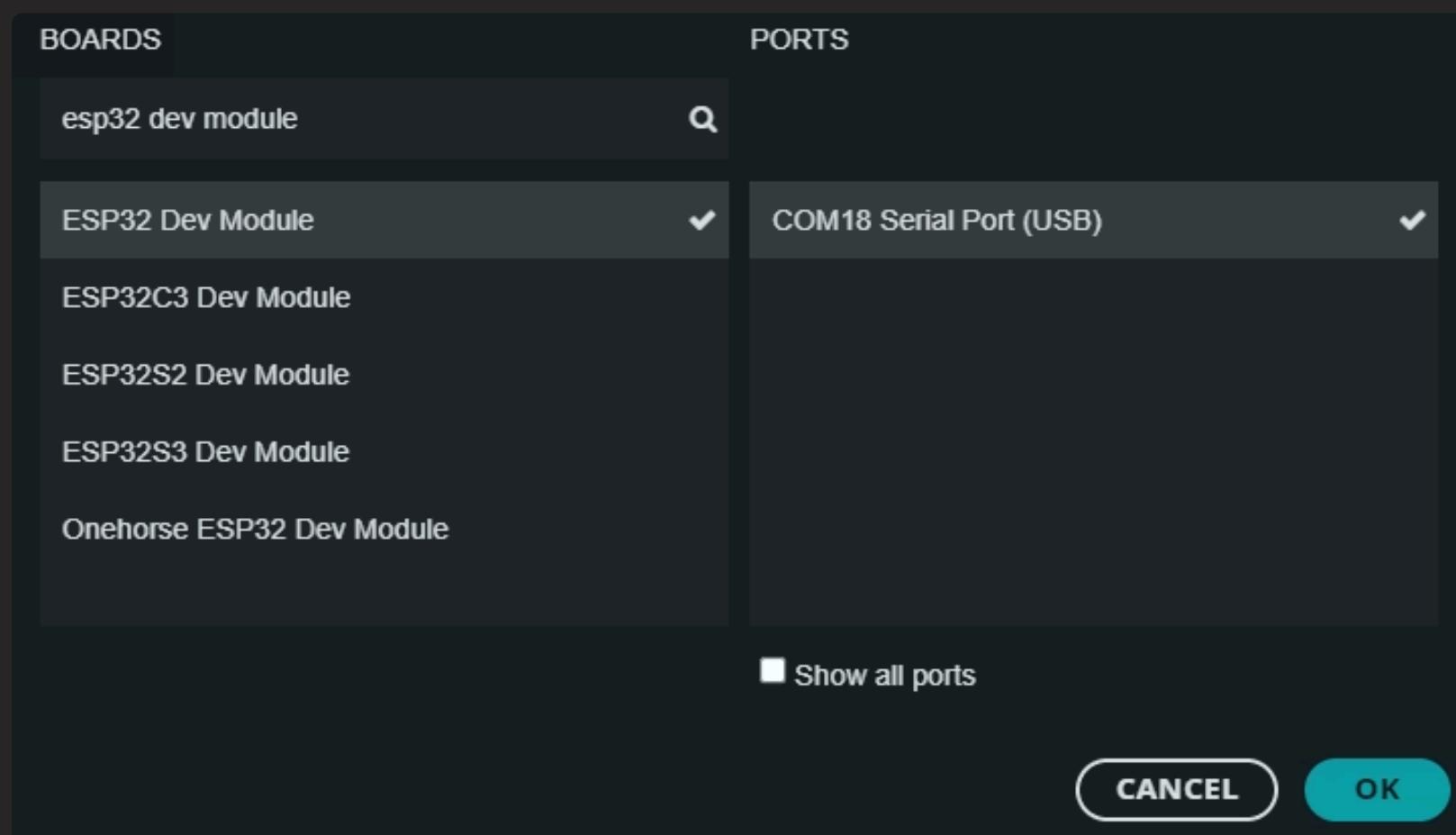
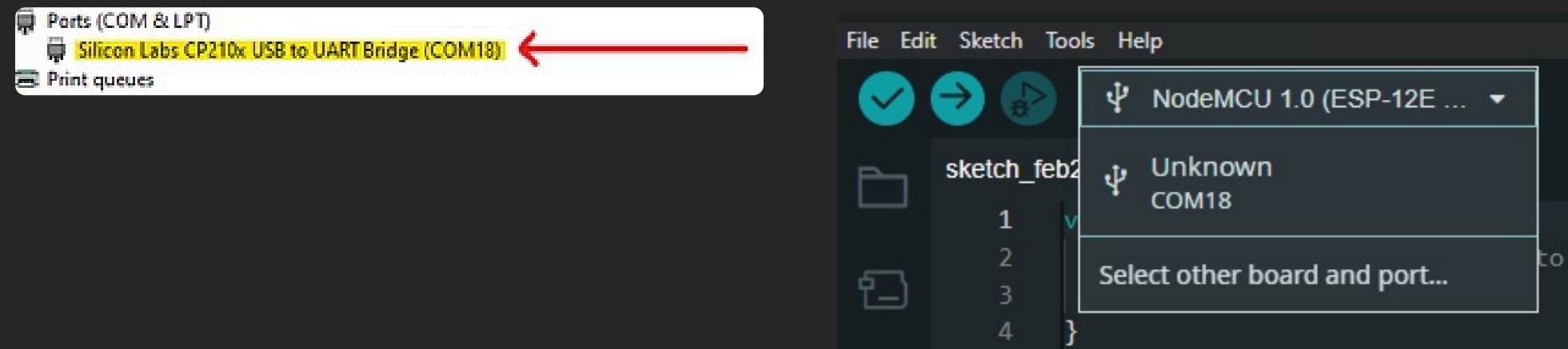
1. Abriremos el Boards Manager
2. Buscamos la ESP32
3. Instalamos la versión de ESPRESSIF



# Seleccionar Puerto y Placa

Una vez preparado nuestro entorno de desarrollo ahora si podemos empezar a trabajar con el ESP32.

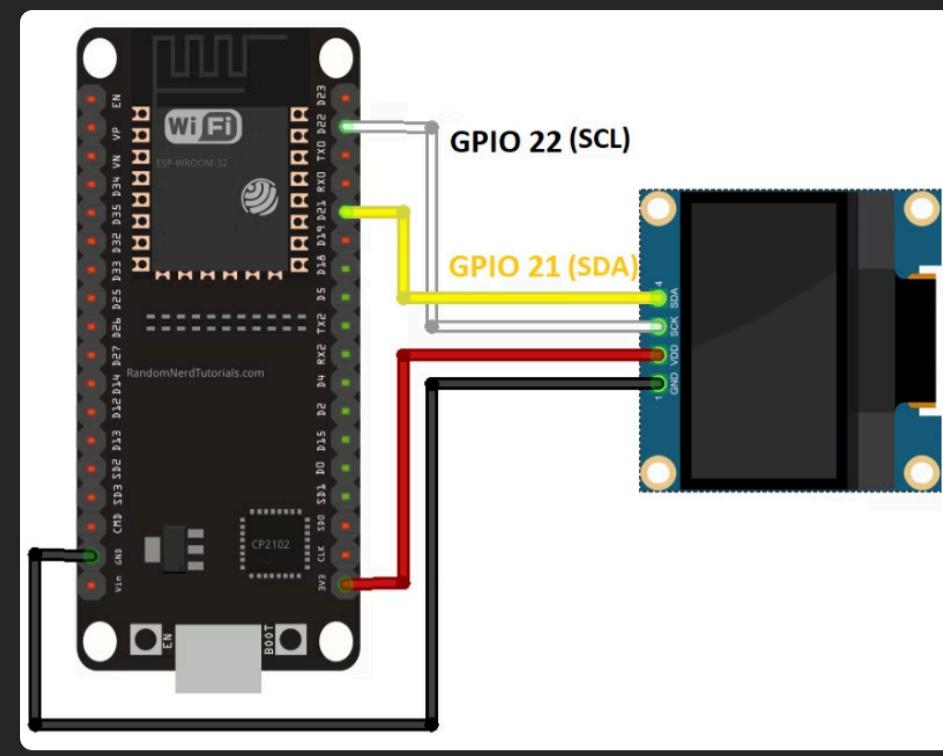
Para hacerlo debemos seleccionar el puerto y placa correctos:



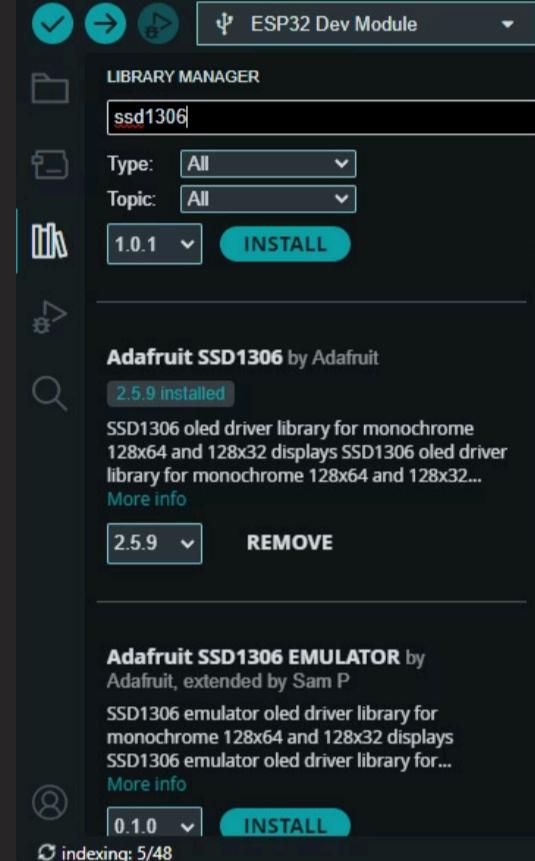
# Conexión OLED 128x64 con la librería SSD1306

## Configuración de la Conexión

El display OLED 128x64 es un componente útil para mostrar información en nuestro proyecto de domótica. Para poder utilizarlo, necesitamos establecer la conexión adecuada y configurar la librería SSD1306 en nuestro código.



Para instalar la librería SSD1306, debemos buscarla en el manager de librerías del IDE:



El código base para hacer funcionar la pantalla OLED 128x64 es el [siguiente](#):

```
//----- DISPLAY>INICIO
#include <SPI.h>
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>

Adafruit_SSD1306 display = Adafruit_SSD1306(128, 64, &Wire);

// 'luz', 13x15px
const unsigned char icon_luz [] PROGMEM = {
    0xff, 0xe0, 0x40, 0x20, 0x20, 0x20, 0x20, 0x40, 0x20, 0x40, 0x80, 0x41,
    0xf8, 0x80, 0x08,
    0xc0, 0x10, 0x38, 0x20, 0x06, 0x40, 0x03, 0x80, 0x06, 0x00, 0x04, 0x00, 0x08, 0x00
};

//----- DISPLAY>FIN

void setup()
{
//----- DISPLAY>INICIO
    display.begin(SSD1306_SWITCHCAPVCC, 0x3C);
    // Display the splash screen (we're legally required to do so)
    display.display();
    display.clearDisplay();
//----- DISPLAY>FIN
}

void loop()
{
//----- DISPLAY>INICIO
    drawScreen();
//----- DISPLAY>FIN
}

//----- DISPLAY>INICIO
void drawScreen(){
    display.clearDisplay();

    display.drawBitmap(28,12, icon_luz, 13,15, WHITE); // Position, bitmap, size,
color

    display.display();
}
//----- DISPLAY>FIN
```

# Conexión a WIFI utilizando la librería Wifi.h

## Importancia de la Conexión a Internet

La conexión a internet es fundamental en un sistema de domótica para poder controlar y monitorear dispositivos de forma remota. Utilizaremos la librería WiFi.h para conectar nuestro ESP32 a una red WIFI disponible.

El código base para lograr una conexión WIFI con la ESP32 es el [siguiente](#):

```
----- WIFI>INICIO
#include <HTTPClient.h>
#include <Wifi.h>

WiFiClient wifiClient;

const char* ssid = "nombreDelRouter";
const char* password = "contrasena";
----- WIFI>FIN
void setup() {
    Serial.begin(115200);
----- WIFI>INICIO
    delay(1000); WiFi.begin(ssid, password);

    Serial.print("Conectando...");
    while (WiFi.status() != WL_CONNECTED) { //Buscando la conexión
        delay(500);
        Serial.print(".");
    }

    Serial.print("Conectado con éxito, mi IP es: ");
    Serial.println(WiFi.localIP());
----- WIFI>FIN
}

void loop() {
----- WIFI>INICIO
    if(WiFi.status()== WL_CONNECTED){ //Revisa el estado de la conexión Wifi

    }else{
        Serial.println("Error en la conexión WIFI");
    }
----- WIFI>FIN
    delay(10);
}
```

# Conexión a WIFI - Inicializando la librería

```
#include <HTTPClient.h>
```

Incluimos la librería HTTPClient.h, que nos permite realizar llamadas HTTP.

---

```
#include <Wifi.h>
```

Incluimos la librería WiFi.h, que nos permite conectarnos a redes WiFi.

---

```
WiFiClient wifiClient;
```

Creamos un objeto WiFiClient, que utilizaremos para interactuar con la red WiFi.

---

```
const char* ssid = "nombreDelRouter";
const char* password = "contrasena";
```

Definimos el nombre de la red WiFi (SSID) y la contraseña.

```
...
//----- WIFI>INICIO
#include <HTTPClient.h>
#include <Wifi.h>

WiFiClient wifiClient;

const char* ssid = "nombreDelRouter";
const char* password = "contrasena";
//----- WIFI>FIN
```

# Conexión a WIFI - Realizando la conexión WiFi

```
WiFi.begin(ssid, password);
```

Comenzamos la conexión con el SSID y la contraseña definidos.

---

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

Esperamos hasta que el ESP32 se conecte a la red WiFi. Mientras no estemos conectados:

1. Esperamos medio segundo.
  2. Mostramos un punto para indicar que seguimos intentándolo.
- 

```
Serial.print("Conectado con éxito, mi IP es: ");
Serial.println(WiFi.localIP());
```

Una vez conectados, imprimimos en el monitor serial la dirección IP asignada al ESP32.



# Conexión a WIFI - Trabajando en la conexión

```
if(WiFi.status()== WL_CONNECTED){
```

Verificamos si estamos conectados a la red WiFi.

---

```
Serial.println("Error en la conexión WIFI");
```

Si no estamos conectados, mostramos un mensaje de error.

```
void loop() {
//----- WIFI>INICIO
  if(WiFi.status()== WL_CONNECTED){ //Revisa el estado de la conexión Wifi

  }else{
    Serial.println("Error en la conexión WIFI");
  }
//----- WIFI>FIN
  delay(10);
}
```



# Llamadas HTTP utilizando la librería HTTPClient.h

1

## Interacción con Servicios Web

Para interactuar con servicios web y enviar/recibir datos, utilizaremos llamadas HTTP. La librería HTTPClient.h nos permitirá realizar estas operaciones de manera sencilla desde nuestro ESP32.

2

## Envío y Recepción de Datos

Las llamadas HTTP nos permitirán enviar y recibir datos de forma eficiente y segura, lo que es crucial para el funcionamiento de un sistema de domótica.

# Llamadas HTTP utilizando el método POST

El código base para lograr una conexión WiFi con la ESP32 es el [siguiente](#):

```
● ● ●

//-----> WIFI>INICIO
#include <HTTPClient.h>
#include <Wifi.h>

WiFiClient wifiClient;

const char* ssid = "nombreDelRouter";
const char* password = "contrasena";
//-----<< WIFI>FIN

//*****>> HTTP>INICIO
HTTPClient http;
int amount_of_devices = 13;
String devices[13] = {"0", "0", "0", "0", "0", "0", "0", "0", "0", "0", "0", "0", "0"};
float gas, agua, elect, tinaco, solar, heolico;
String message, datos_a_enviar, cuerpo_respuesta;
int codigo_respuesta;
//*****<< HTTP>FIN

void setup() {
    Serial.begin(115200);
    //----->> WIFI>INICIO
    delay(1000); WiFi.begin(ssid, password);

    Serial.print("Conectando...");
    while (WiFi.status() != WL_CONNECTED) { //Buscando la conexión
        delay(500);
        Serial.print(".");
    }

    Serial.print("Conectado con éxito, mi IP es: ");
    Serial.println(WiFi.localIP());
    //-----<< WIFI>FIN
}

void loop() {
    //----->> WIFI>INICIO
    if(WiFi.status()== WL_CONNECTED){ //Revisa el estado de la conexión WiFi
    //*****>> HTTP>INICIO
        httpCall();
    //*****<< HTTP>FIN
    }else{
        Serial.println("Error en la conexión WiFi");
    }
    //-----<< WIFI>FIN
    delay(10);
}

//*****>> HTTP>INICIO
void httpCall(){
    //Indicamos el destino
    http.begin(wifiClient, "http://3d-pop.com/DanielGV/assets/php/esp32.php");
    //Preparamos el header text/plain si solo vamos a enviar texto plano
    http.addHeader("Content-Type", "application/x-www-form-urlencoded");

    gas += 5;
    agua += 2;
    elect += 1;
    tinaco = random(10, 4000);
    solar = random(3, 5);
    heolico = random(1, 3);

    message = String(gas)+"$"+String(agua)+"$"
    +String(elect)+"$"+String(tinaco)
    +"$"+String(solar)+"$"+String(heolico);
    datos_a_enviar = "device_states=" + message;
    codigo_respuesta = http.POST(datos_a_enviar); //Enviamos el post pasándole, los datos que queremos enviar. (esta función nos devuelve un código que guardamos en un int)

    if(codigo_respuesta>0){
        Serial.println("Código HTTP - " + String(codigo_respuesta)); //Print return code

        if(codigo_respuesta == 200){
            cuerpo_respuesta = http.getString();
            Serial.println("El servidor respondió ▼ ");
            explode(cuerpo_respuesta);
            Serial.println(cuerpo_respuesta);
        }
    }else{
        Serial.print("Error enviando POST, código: ");
        Serial.println(codigo_respuesta);
    }

    http.end(); //libero recursos
}
void explode(String data){
    int i;
    char delimiter[] = "$";
    char *p;
    char string[128];

    data.toCharArray(string, sizeof(string));
    i = 0;
    p = strtok(string, delimiter);
    while(p && i < amount_of_devices)
    {
        devices[i] = p;
        p = strtok(NULL, delimiter);
        ++i;
    }
}
//*****<< HTTP>FIN
```

# Llamadas HTTP - Inicializando valores

## Inicializando la librería

```
HTTPClient http;
```

Creamos un objeto de tipo HTTPClient.

---

```
int amount_of_devices = 13;
```

Número total de dispositivos.

---

```
String devices[13] = {"0", "0", "0", "0", "0", "0", "0", "0",  
                      "0", "0", "0", "0", "0",};
```

Arreglo para almacenar los estados de los dispositivos una vez recibidos desde la base de datos.

---

```
float gas, agua, elect, tinaco, solar, heolico;
```

Variables para almacenar datos de sensores y actuadores.

---

```
String message, datos_a_enviar, cuerpo_respuesta;
```

Variables para almacenar datos de envío y datos recibidos de la base de datos.

---

```
int codigo_respuesta;
```

Variable para almacenar el código de respuesta HTTP.

---

```
...  
//*****> HTTP>INICIO  
HTTPClient http;  
int amount_of_devices = 13;  
String devices[13] = {"0", "0", "0", "0", "0", "0", "0", "0",  
                      "0", "0", "0", "0", "0",};  
float gas, agua, elect, tinaco, solar, heolico;  
String message, datos_a_enviar, cuerpo_respuesta;  
int codigo_respuesta;  
//*****<< HTTP>FIN
```

# Llamadas HTTP - Función httpCall()

## Llamando a la comunicación HTTP

```
httpCall();
```

Desde el loop llamamos a la función httpCall() que se encargará de comunicarse con la base de datos.

```
void loop() {
    //-----> WIFI>INICIO
    if(WiFi.status()== WL_CONNECTED){ //Revisa el estado de la conexión Wifi
        //*****><*****> HTTP>INICIO
        httpCall();
    }else{
        Serial.println("Error en la conexión WIFI");
    }
    //-----<< WIFI>FIN
    delay(10);
}
```

## Función httpCall()

```
http.begin(wifiClient, "http://3d-
pop.com/DanielGV/assets/php/esp32.php");
```

Indicamos el destino de la llamada HTTP.

```
http.addHeader("Content-Type", "application/x-www-
form-urlencoded");
```

Preparamos el header text/plain si solo vamos a enviar texto plano.

```
gas += 5;
agua += 2;
elect += 1;
tinaco = random(10, 4000);
solar = random(3, 5);
heolico = random(1, 3);
```

Simulamos datos de sensores y actuadores.

```
message = String(gas)+"$"+String(agua)
+"$"+String(elect)+"$"+String(tinaco)
+"$"+String(solar)+"$"+String(heolico);
```

Creamos un mensaje con los datos de los sensores y los valores de consumo simulados.

```
datos_a_enviar = "device_states=" + message;
```

Concatenamos el mensaje con la clave "device\_states". Esto para cumplir con el formato de la llamada

```
"device_states=0$0$0$0$0$0"
```

```
codigo_respuesta = http.POST(datos_a_enviar);
```

Enviamos el mensaje utilizando el método POST y almacenamos el código de respuesta.

```
if(codigo_respuesta > 0){
```

Si recibimos una respuesta...

```
Serial.println("Código HTTP ► " +
String(codigo_respuesta));
```

Imprimimos el código de respuesta.

```
if(codigo_respuesta == 200){
```

Si el código de respuesta es 200 (OK)...

```
cuerpo_respuesta = http.getString();
```

Almacenamos el cuerpo de la respuesta HTTP. (Esta es la respuesta del servidor, incluye la información de los dispositivos en formato de cadena de texto separado por \$).

```
explode(cuerpo_respuesta);
```

Llamamos a la función "explode" para procesar la respuesta.

```
Serial.println(cuerpo_respuesta);
```

Si no recibimos respuesta, mostramos un mensaje de error.

```
//*****><*****> HTTP>INICIO
void httpcall(){
    //Indicamos el destino
    http.begin(wifiClient, "http://3d-pop.com/DanielGV/assets/php/esp32.php");
    //Preparamos el header text/plain si solo vamos a enviar texto plano
    http.addHeader("Content-Type", "application/x-www-form-urlencoded");

    gas += 5;
    agua += 2;
    elect += 1;
    tinaco = random(10, 4000);
    solar = random(3, 5);
    heolico = random(1, 3);

    message = String(gas)+"$"+String(agua)+"$"
    +"$"+String(elect)+"$"+String(tinaco)
    +"$"+String(solar)+"$"+String(heolico);
    datos_a_enviar = "device_states=" + message;
    codigo_respuesta = http.POST(datos_a_enviar); //Enviamos el post pasándole, los datos que queremos enviar. (esta función nos devuelve un código que guardamos en un int)

    if(codigo_respuesta > 0){
        Serial.println("Código HTTP ► " + String(codigo_respuesta)); //Print return code

        if(codigo_respuesta == 200){
            cuerpo_respuesta = http.getString();
            Serial.println("El servidor respondió ▶ ");
            explode(cuerpo_respuesta);
            Serial.println(cuerpo_respuesta);
        }
    }else{
        Serial.print("Error enviando POST, código: ");
        Serial.println(codigo_respuesta);
    }

    http.end(); //libero recursos
}
```

```
Serial.print("Error enviando POST, código: ");
Serial.println(codigo_respuesta);
```

Si no recibimos respuesta, mostramos un mensaje de error.

```
http.end();
```

Liberamos recursos.

# Llamadas HTTP - Función explode()

## Separando valores utilizando el carácter \$

```
explode(String data);
```

Función para procesar la respuesta HTTP y extraer los datos de los dispositivos. Separa la cadena entrante utilizando el carácter \$ como identificador y guarda cada uno de los valores en un índice del arreglo devices

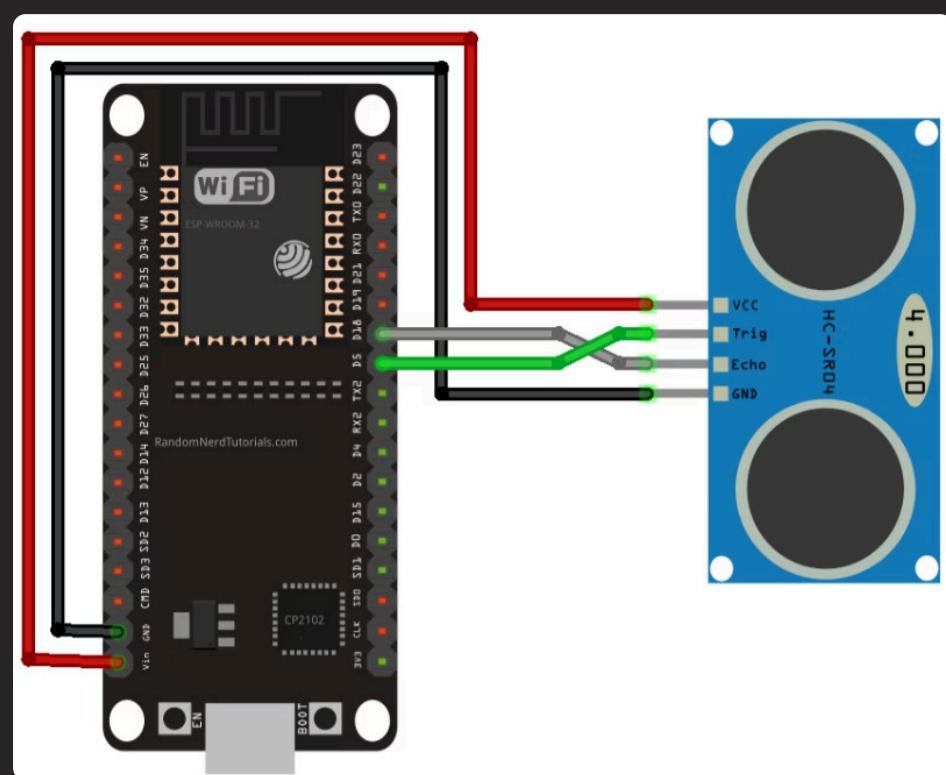
```
...
void explode(String data){
    int i;
    char delimiter[] = "$";
    char *p;
    char string[128];

    data.toCharArray(string, sizeof(string));
    i = 0;
    p = strtok(string, delimiter);
    while(p && i < amount_of_devices)
    {
        devices[i] = p;
        p = strtok(NULL, delimiter);
        ++i;
    }
}
//*****<< HTTP>FIN
```

# Medir distancia con el sensor HC-SR04

## Sensor de distancia como nivel de agua digital

El sensor ultrasónico HC-SR04 es un dispositivo que utiliza ondas ultrasónicas para medir distancias de forma precisa y no invasiva. Funciona emitiendo pulsos ultrasónicos a través de un transductor y luego detectando el eco generado al rebotar en un objeto. Midiendo el tiempo que tarda en regresar el eco, el sensor puede calcular la distancia entre él y el objeto.



El código base para lograr una lectura de distancia con el sensor hc-sr04 es el [siguiente](#):

```
•••  
//oooooooooooooooooooo>> DISTANCIA>INICIO  
const int trigPin = 5;  
const int echoPin = 18;  
  
//Define la velocidad del sonido en cm/uS  
#define SOUND_VELOCITY 0.034  
#define CM_TO_INCH 0.393701  
  
long duration;  
float distanceCm;  
//oooooooooooooooooooo>> DISTANCIA>FIN  
void setup() {  
    Serial.begin(115200);  
    //oooooooooooo>> DISTANCIA>INICIO  
    pinMode(trigPin, OUTPUT); // Establece trigPin como Output  
    pinMode(echoPin, INPUT); // Establece echoPin como Input  
    //oooooooooooo>> DISTANCIA>FIN  
}  
  
void loop() {  
    //oooooooooooo>> DISTANCIA>INICIO  
    readDistance();  
    //oooooooooooo>> DISTANCIA>FIN  
    delay(10);  
}  
  
//oooooooooooo>> DISTANCIA>INICIO  
void readDistance(){  
    // Libera el pin trigger  
    digitalWrite(trigPin, LOW);  
    delayMicroseconds(2);  
    // Establece el trigPin al estado HIGH por 10 micro segundos  
    digitalWrite(trigPin, HIGH);  
    delayMicroseconds(10);  
    digitalWrite(trigPin, LOW);  
  
    // Lee el echoPin, regresa el tiempo que tardo en regresar la onda de sonido en  
    // microsegundos  
    duration = pulseIn(echoPin, HIGH);  
  
    // Calcula la distancia en centímetros  
    distanceCm = duration * SOUND_VELOCITY/2;  
}  
//oooooooooooo>> DISTANCIA>FIN
```

# Sensor HC-SR04 - Inicialización de valores

```
const int trigPin = 5;
```

Pin de salida para enviar el pulso ultrasónico

---

```
const int echoPin = 18;
```

Pin de entrada para recibir el eco del pulso ultrasónico

---

```
#define SOUND_VELOCITY 0.034
```

Velocidad del sonido en centímetros por microsegundo  
(cm/uS)

---

```
#define CM_TO_INCH 0.393701
```

Factor de conversión de centímetros a pulgadas

---

```
long duration;
```

Duración del eco en microsegundos

---

```
float distanceCm;
```

Distancia medida en centímetros

---

```
•••  
//oooooooooooooooooooo>> DISTANCIA>INICIO  
const int trigPin = 5;  
const int echoPin = 18;  
  
//Define la velocidad del sonido en cm/uS  
#define SOUND_VELOCITY 0.034  
#define CM_TO_INCH 0.393701  
  
long duration;  
float distanceCm;  
//oooooooooooooooooooo<< DISTANCIA>FIN
```

# Sensor HC-SR04 - Configuración de pines

```
pinMode(trigPin, OUTPUT);
```

Configura el pin trigPin como salida (para enviar el pulso ultrasónico)

---

```
pinMode(echoPin, INPUT);
```

Configura el pin echoPin como entrada (para recibir el eco del pulso ultrasónico)

```
...
void setup() {
    Serial.begin(115200);
//oooooooooooooooooooo>> DISTANCIA>INICIO
    pinMode(trigPin, OUTPUT); // Establece trigPin como Output
    pinMode(echoPin, INPUT); // Establece echoPin como Input
//oooooooooooooooooooo<< DISTANCIA>FIN
}
```

# Sensor HC-SR04 - función readDistance()

## Llamando a la función readDistance

```
readDistance();
```

Llama a la función readDistance() para leer la distancia

```
void loop() {  
    //>>> DISTANCIA>INICIO  
    readDistance();  
    //<< DISTANCIA>FIN  
    delay(10);  
}
```

```
digitalWrite(trigPin, LOW);
```

Establece el pin trigPin en estado bajo (LOW) para asegurar un inicio limpio

```
delayMicroseconds(2);
```

Espera 2 microsegundos para asegurar el estado bajo

```
digitalWrite(trigPin, HIGH);  
delayMicroseconds(10);
```

Establece el pin trigPin en estado alto (HIGH) por 10 microsegundos para enviar el pulso ultrasónico

```
digitalWrite(trigPin, LOW);
```

Regresa el pin trigPin a estado bajo

```
//>>> DISTANCIA>INICIO  
void readDistance(){  
    // Libera el pin trigger  
    digitalWrite(trigPin, LOW);  
    delayMicroseconds(2);  
    // Establece el trigPin al estado HIGH por 10 micro segundos  
    digitalWrite(trigPin, HIGH);  
    delayMicroseconds(10);  
    digitalWrite(trigPin, LOW);  
  
    // Lee el echoPin, regresa el tiempo que tarde en regresar la onda de sonido en  
    // microsegundos  
    duration = pulseIn(echoPin, HIGH);  
  
    // Calcula la distancia en centímetros  
    distanceCm = duration * SOUND_VELOCITY/2;  
}  
//<< DISTANCIA>FIN
```

```
duration = pulseIn(echoPin, HIGH);
```

Lee el tiempo que tarda en llegar el eco

```
distanceCm = duration * SOUND_VELOCITY / 2;
```

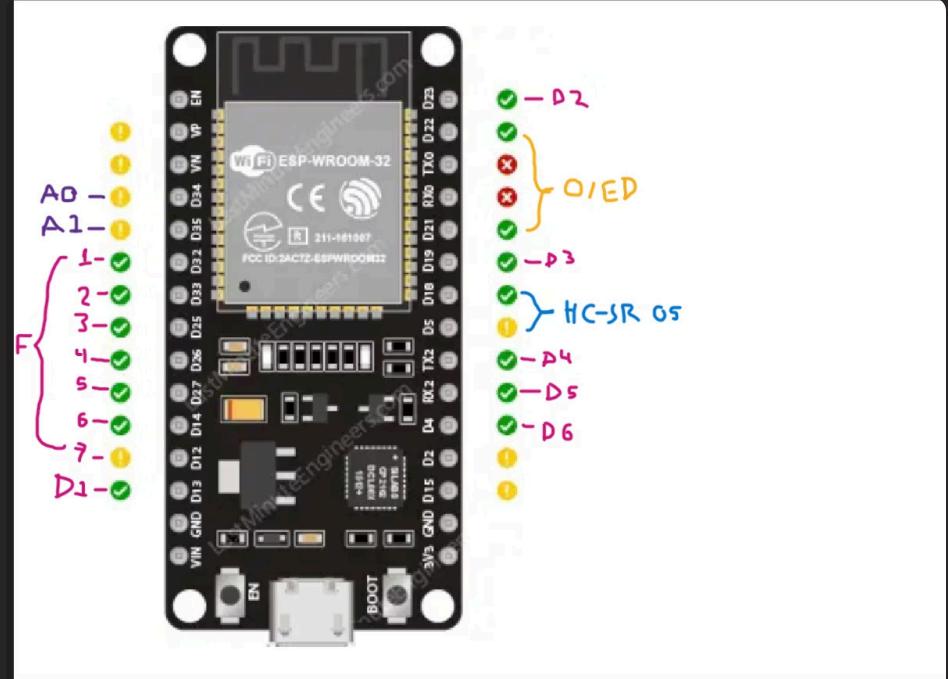
Calcula la distancia en centímetros dividiendo la duración por la mitad

# Sistema de Control de Dispositivos

Para lograr mostrar el comportamiento de nuestros dispositivos en la pantalla debemos aprender a interpretar la información del servidor.

Una vez recibidos los datos del servidor y procesarlos por la función `explode()` los valores de los dispositivos se guardarán en el arreglo `devices[]`, este arreglo tiene la información de los dispositivos en el siguiente orden:

`devices[0]` → foco 1 - Medio Baño  
`devices[1]` → foco 2 - Baño  
`devices[2]` → foco 3 - Cuarto 1  
`devices[3]` → foco 4 - Cuarto 2  
`devices[4]` → foco 5 - Sala/Comedor  
`devices[5]` → foco 6 - Cocina  
`devices[6]` → foco 7 - Garage  
`devices[7]` → computadora - D1  
`devices[8]` → tv - D2  
`devices[9]` → bocinas - D3  
`devices[10]` → regadera - D4  
`devices[11]` → estufa - D5  
`devices[12]` → boiler - D6  
`devices[13]` → portón



Cada uno de los indices del arreglo tendrá el valor "0" o "1" que representará encendido o apagado.

En tu loop deberás evaluar si el dispositivo está encendido o no e igualar la variable booleana que lo representa a `true` o `false` respectivamente.

Ejemplo:

```
if(devices[0] == "1"){
    foco1 = true;
} else{
    foco1 = false;
}
```

En este ejemplo evaluamos el primer dispositivo guardado en `devices[0]` y dependiendo de su valor, cambiamos el estado de la variable que representa a ese dispositivo en nuestro loop.