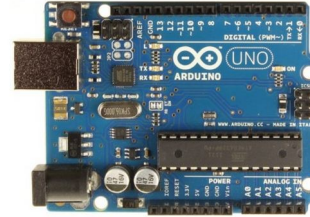


# Hardware IoT

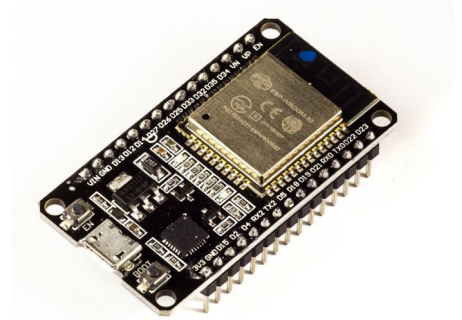
# Hardware IoT

- **Microcontroladores**

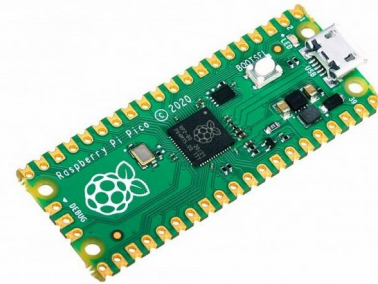
- ATMEL (Arduino)
- Espressif
- RP2040



**Arduino UNO**



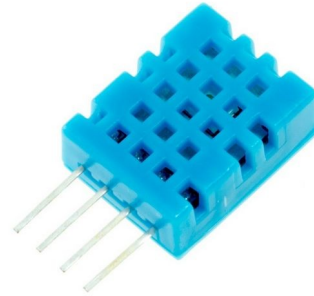
**ESP32**



**Raspberry Pi Pico**

# Hardware IoT

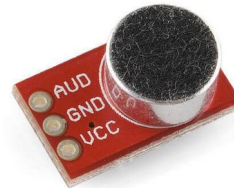
- **Sensores**
  - Recopilar datos del entorno
  - Procesar información
  - Determinar cómo actuar



**DHT11**  
Temperatura y humedad



**HC-SR04**  
Distancia



**Micrófono**  
Sonido



**MQ-5**  
Medidor de gas metano

# Hardware IoT

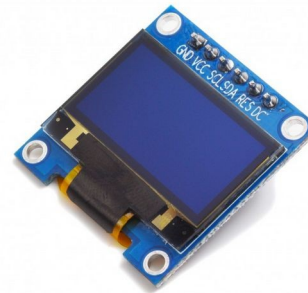
- Actuadores
  - Relé
  - Servomotor
- Periféricos
  - Indicadores
  - Pulsadores



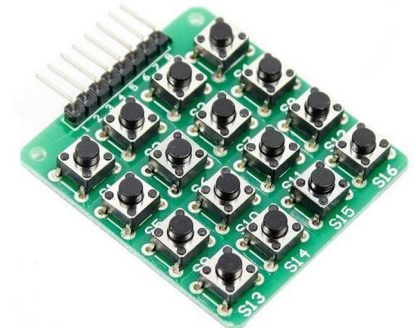
Relé



Servomotor



Pantalla OLED



Matriz de botones

# Hardware IoT

- Gateway
  - Raspberry Pi
  - PC de escritorio



**Raspberry Pi**

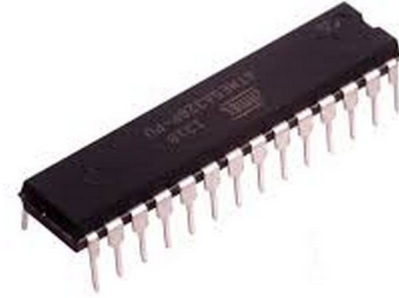


**PC Servidor**

# Microcontroladores

# Microcontroladores

- Circuitos integrados
- Placas de desarrollo



# Microcontroladores

- **ATMEL**
  - Gran variedad de microcontroladores de diferentes especificaciones
  - Usado en placas oficiales de Arduino
  - Diferentes placas de desarrollo orientadas a diferentes aplicaciones
  - Módulos directamente compatibles



# Microcontroladores

- **Espressif**
  - Poca variedad de modelos (ESP8266, ESP32, ESP32-S, ESP32-C)
  - WiFi y Bluetooth integrado
  - Variedad de protocolos soportados de forma oficial (HTTP, MQTT, ...)
  - Compatible con el ecosistema Arduino
  - Compatible con el ecosistema MicroPython

# Microcontroladores

- **RP2040**
  - Desarrollado por Raspberry Pi y pensado para trabajar en conjunto
  - Único modelo, económico y mejores especificaciones en comparación a Arduino de similar tamaño
  - Compatible con el ecosistema Arduino
  - Compatible con el ecosistema MicroPython
  - Desarrollo reciente pero prometedor

# **Kit de Desarrollo de Software IoT**

# Kit de Desarrollo de Software IoT

- **Arduino**
  - Ampliamente utilizado
  - Ecosistema de periféricos y sensores
  - Variada gama de microcontroladores (8 y 32 bits)
  - Fácil de utilizar
  - Lenguaje: C++

# Kit de Desarrollo de Software IoT

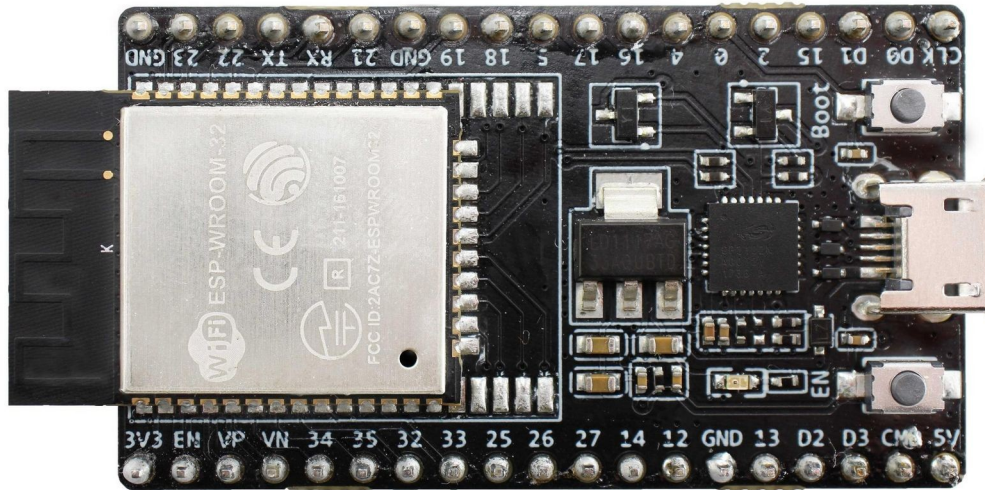
- **ESP-IDF**
  - Robusto y muy documentado
  - No es directamente compatible con bibliotecas de Arduino
  - Sólo dispositivos de Espressif
  - Bajo nivel. Expone complejidades al programador
  - Lenguaje: C/C++

# Kit de Desarrollo de Software IoT

- **MicroPython**
  - Creciente popularidad y excelente documentación
  - Fácilmente extendible y amplia biblioteca estándar
  - Dispositivos modernos (ESP32, RP Pico, ...)
  - Alto nivel e interpretado: no requiere compilar el código
  - Lenguaje: Python

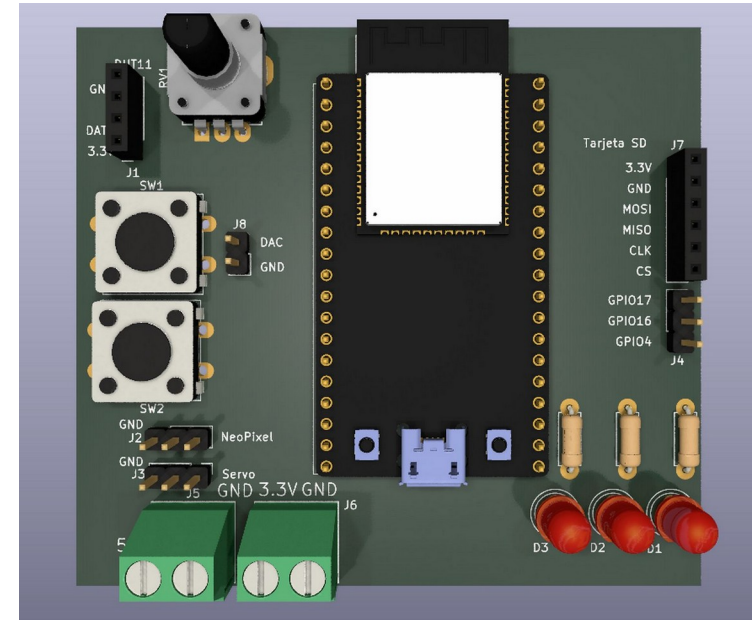
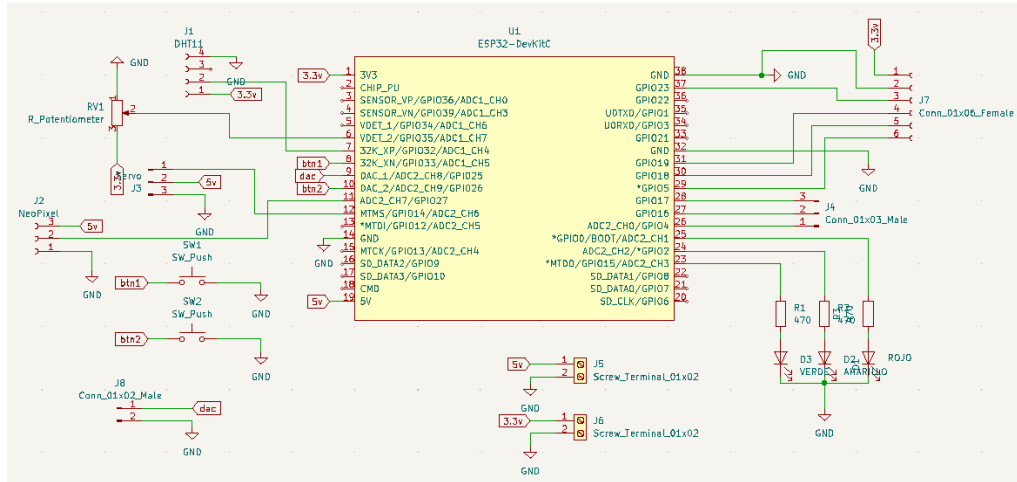
# Kit de Desarrollo de Software IoT

- Plataforma seleccionada:
  - ESP32 con MicroPython



# Kit de Desarrollo de Software IoT

- Placa de desarrollo





# Kit de Desarrollo de Software IoT

- Puertos relevantes
  - LED verde: GPIO15
  - LED amarillo: GPIO2
  - LED rojo: GPIO0
  - Botón superior: GPIO33
  - Botón inferior: GPIO26

# Introducción a MicroPython

# Introducción a MicroPython

- ¿Qué es?
  - Implementación liviana y eficiente de Python 3
  - Incluye parte de su biblioteca estándar
  - Optimizado para microcontroladores

# Introducción a MicroPython

- Instalación
  - <Video de instalación/demostración de instalación>

# Introducción a MicroPython

- Entorno Thonny

Archivos locales

Archivos en ESP32

Editor de código

Intérprete de comandos

# Introducción a MicroPython

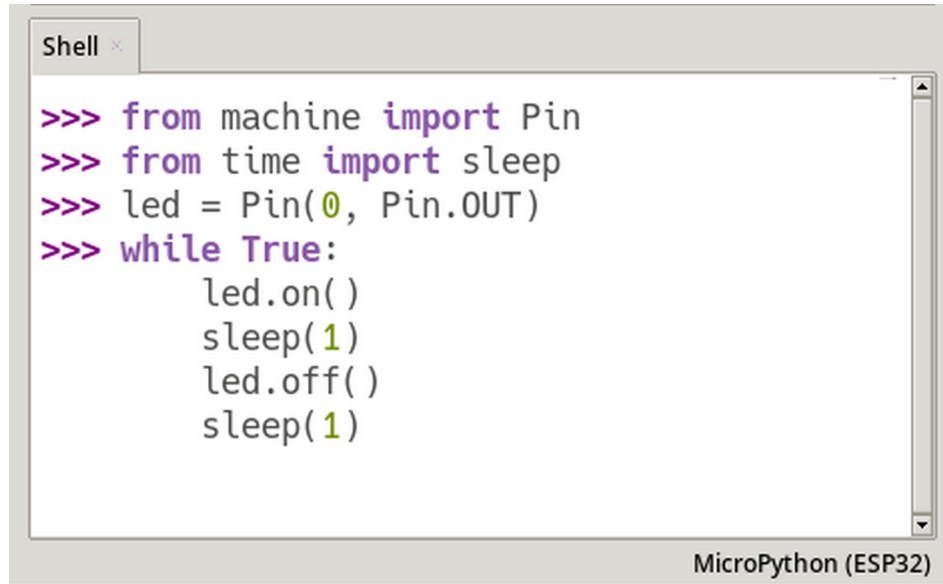
- **Módulos estándar**
  - `machine`: contiene funciones relacionadas con el hardware
    - `Pin`: permite controlar las entradas y salidas
  - `time`: contiene funciones relacionadas con el control del tiempo
    - `sleep`: demora la ejecución un tiempo determinado
    - `ticks_ms`: determina la cantidad de milisegundos desde el inicio del dispositivo

# Introducción a MicroPython

- `machine.Pin`
  - Parámetros:
    - Número de GPIO
    - Modo (`Pin.IN` para entrada, `Pin.OUT` para salida)
      - En caso de entrada, se puede configurar una resistencia de pull-up con `Pin.PULL_UP`
    - Devuelve un objeto correspondiente al pin que se puede encender o apagar con los métodos `.on()` y `.off()`
    - También puede usarse el método `.value()` para leer el estado actual o asignarle un estado nuevo como por ejemplo `.value(1)`

# Introducción a MicroPython

- Ejemplo



```
Shell x
>>> from machine import Pin
>>> from time import sleep
>>> led = Pin(0, Pin.OUT)
>>> while True:
    led.on()
    sleep(1)
    led.off()
    sleep(1)
```

MicroPython (ESP32)

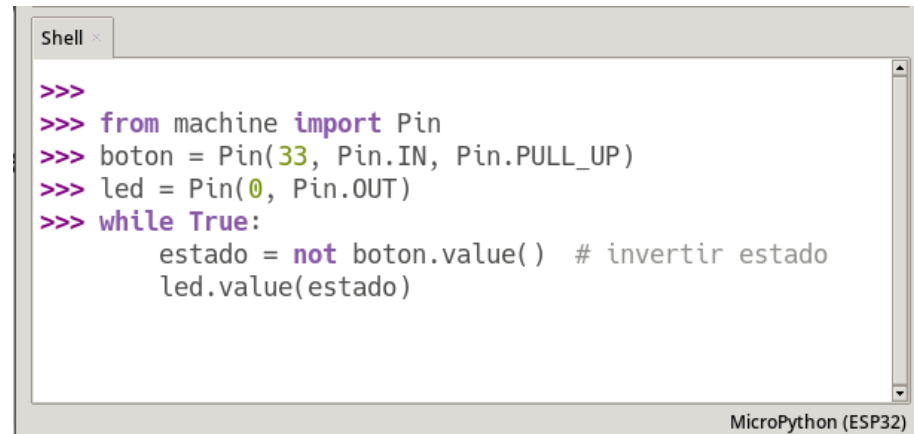


# Introducción a MicroPython

- Ejercicio 1

- Encender un LED según el estado de un botón. Si el botón no está pulsado, el LED debe estar apagado.

- Solución

A screenshot of a MicroPython shell window. The window has a title bar with a tab labeled 'Shell'. The code is as follows:

```
>>>
>>> from machine import Pin
>>> boton = Pin(33, Pin.IN, Pin.PULL_UP)
>>> led = Pin(0, Pin.OUT)
>>> while True:
    estado = not boton.value() # invertir estado
    led.value(estado)
```

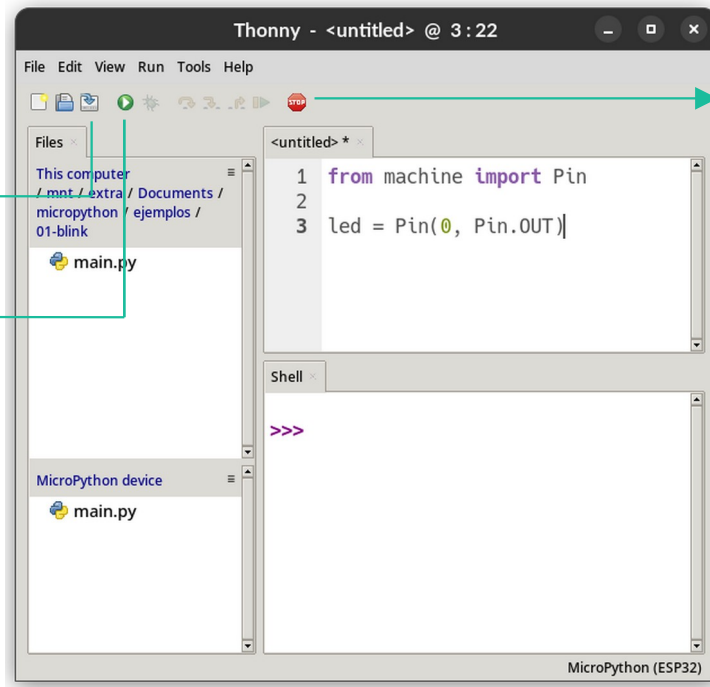
The bottom right corner of the window displays 'MicroPython (ESP32)'.

```
Shell x
>>>
>>> from machine import Pin
>>> boton = Pin(33, Pin.IN, Pin.PULL_UP)
>>> led = Pin(0, Pin.OUT)
>>> while True:
    estado = not boton.value() # invertir estado
    led.value(estado)
MicroPython (ESP32)
```

# Introducción a MicroPython

- **Ejercicio 2**
  - Incrementar un contador al pulsar un botón. Decrementarlo al pulsar el otro.
  - Nota: usar el editor de código para hacer pruebas

# Introducción a MicroPython



Detener ejecución

Guardar código actual

Ejecutar

# Introducción a MicroPython

- **Ejercicio 3**
  - Construir la secuencia de un semáforo avanzando cada estado al pulsar un botón