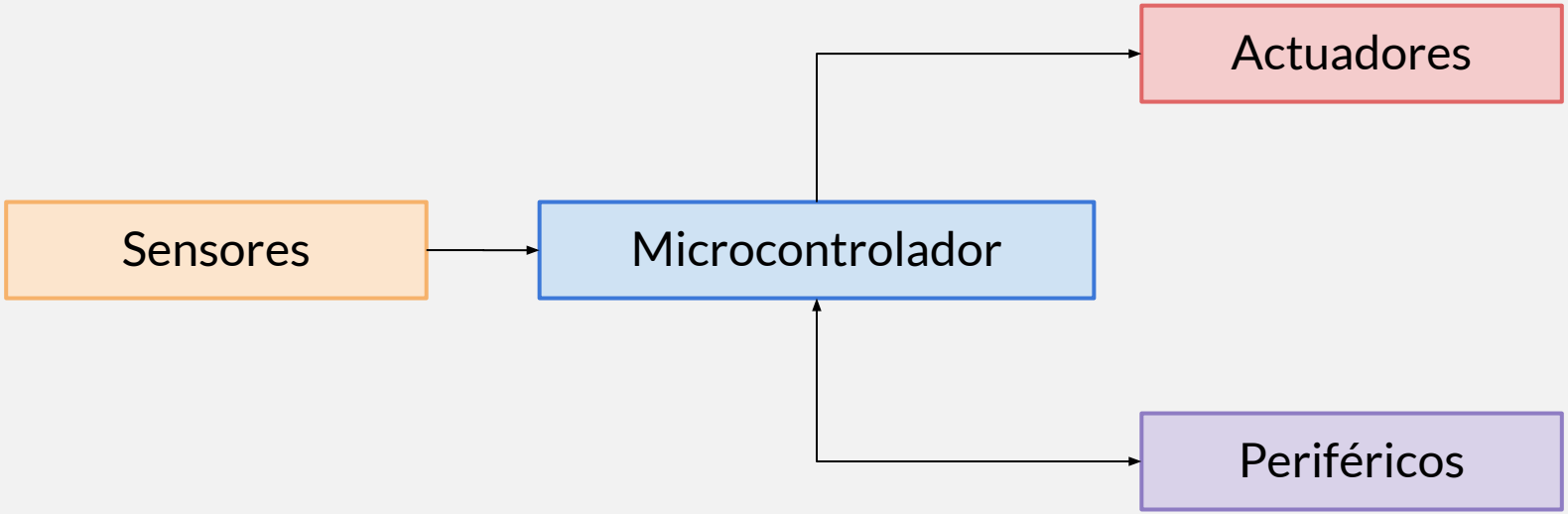
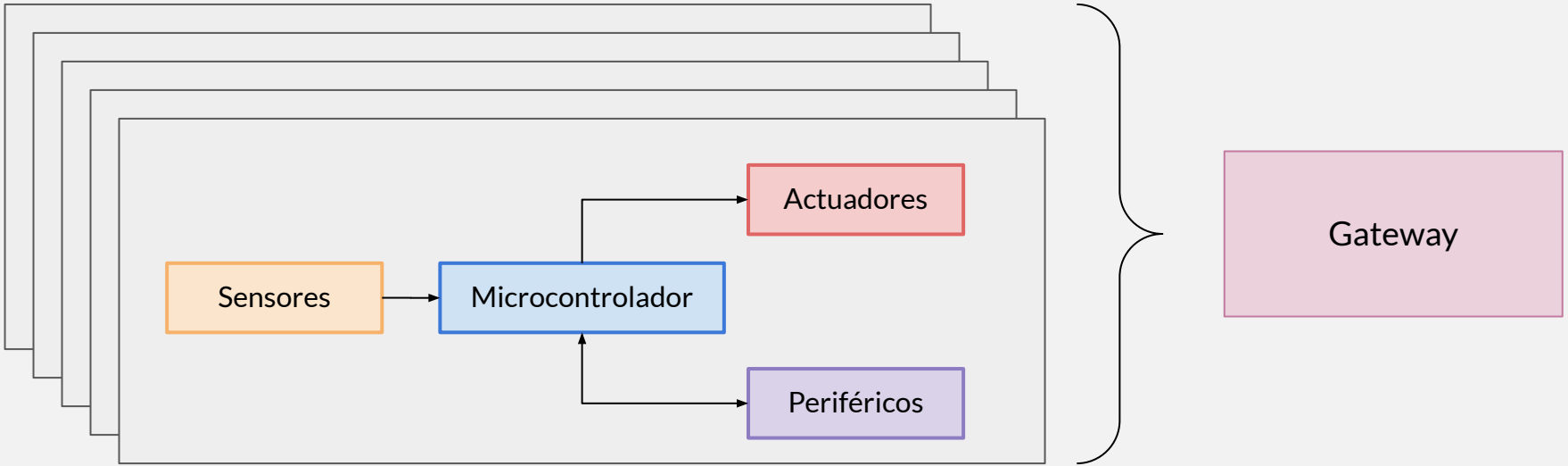
Hardware IoT

Hardware IoT

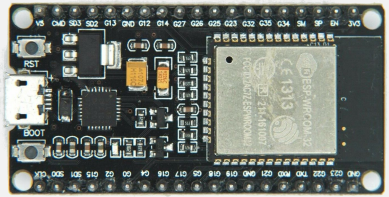


Hardware IoT



Placas de desarrollo

Arduino UNO

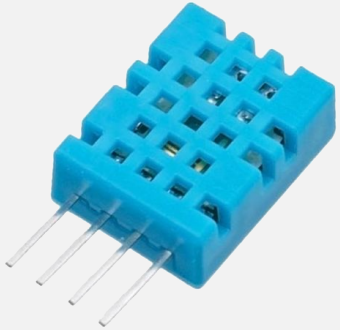


ESP32

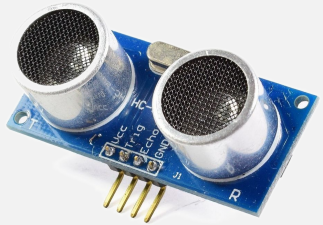
Raspberry Pi Pico

Sensores

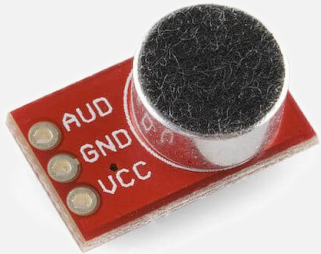
- Recopilar datos del entorno
- Procesar información
- Determinar cómo actuar



DHT11
Temperatura y humedad



HC-SR04
Distancia



Micrófono
Sonido

MQ-5
Medidor de gas metano

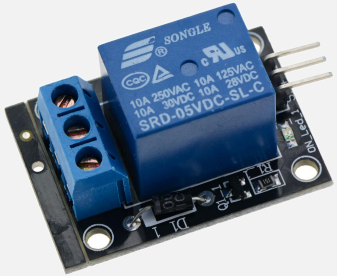
Actuadores y periféricos

Actuadores

- Relé
- Servomotor

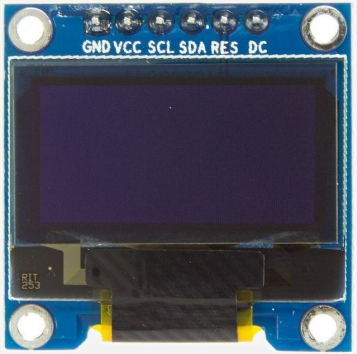
Periféricos

- Indicadores
- Pulsadores

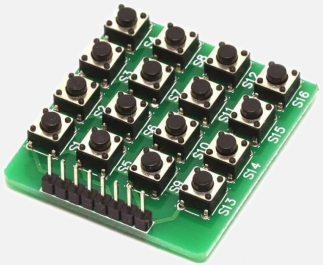


Relé

Servomotor



Pantalla OLED



Matriz de botones

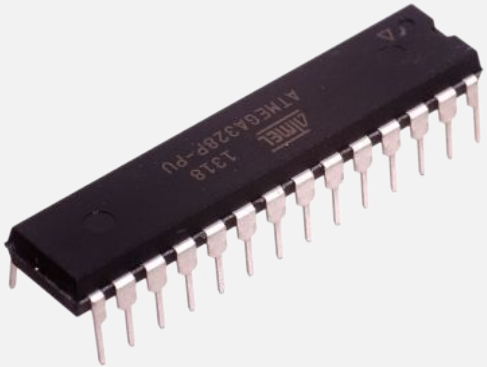
Gateway

Raspberry Pi

PC Servidor

Microcontroladores

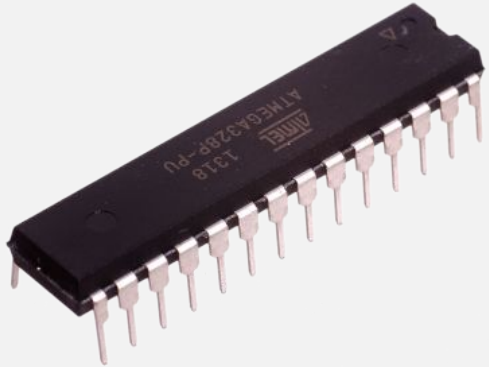
Microcontroladores



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

NEON

Facultad de Ingeniería

Universidad Nacional de Mar del Plata

Kit de Desarrollo Arduino

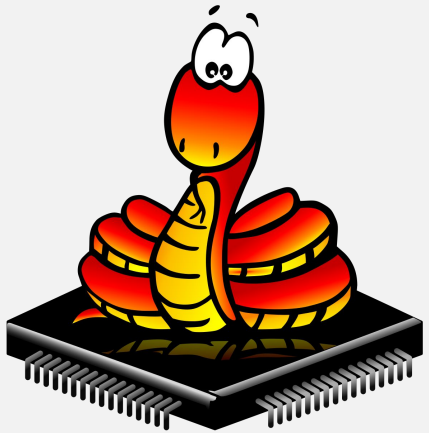
- Ampliamente utilizado
- Ecosistema de periféricos y sensores
- Variada gama de microcontroladores (8 y 32 bits)
- Fácil de utilizar
- Lenguaje: C++

Kit de Desarrollo 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++

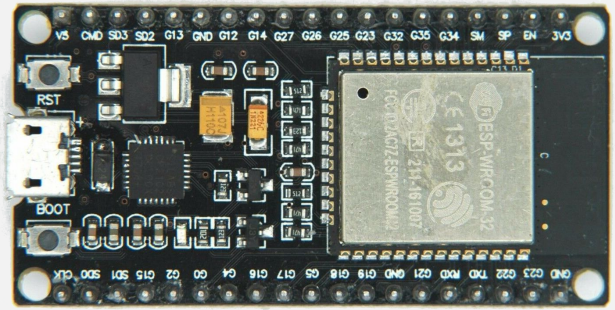
Kit de Desarrollo MicroPython

- Creciente popularidad
- Fácilmente extensible
- Amplia biblioteca estándar
- Alto nivel e interpretado
- Lenguaje: Python

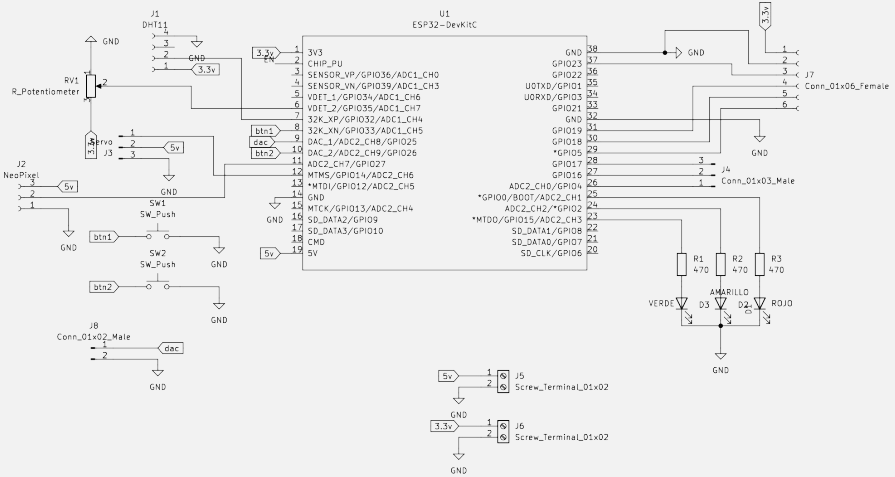


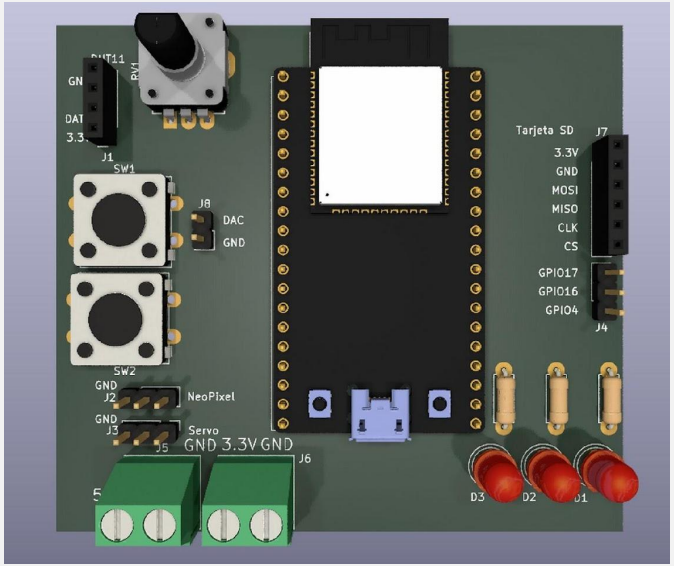
Kit de Desarrollo Seleccionado

- ESP32
- MicroPython



Placa de Desarrollo

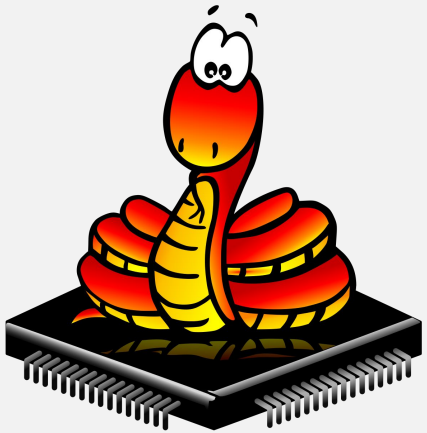




Introducción a MicroPython

MicroPython

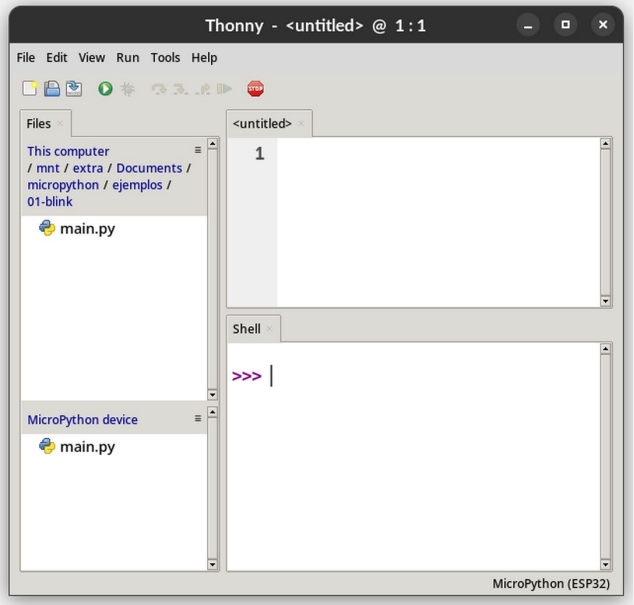
- Implementación liviana y eficiente de Python 3
- Incluye parte de su biblioteca estándar
- Optimizado para microcontroladores



Entorno de desarrollo Thonny

Archivos locales

Archivos en ESP32



Editor de código

Intérprete de Python

Módulos estándar de MicroPython

`machine`

Contiene funciones relacionadas con el hardware

- `Pin`: permite controlar entradas y salidas digitales
  - Parámetros:
    - Número de GPIO
    - Modo (Pin.IN para entrada, `Pin.OUT` para salida)
    - Resistencia Pull-up o Pull-down (por defecto deshabilitado)
  - 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)`

Módulos estándar de MicroPython

`time`

Contiene funciones relacionadas con el control del tiempo

- `sleep_ms`: demora la ejecución un tiempo determinado
- `ticks_ms`: determina la cantidad de milisegundos desde el inicio del dispositivo

Ejemplo

Shell

```
>>> 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)

Ejemplos

Ejemplo 1

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

Solución

Shell

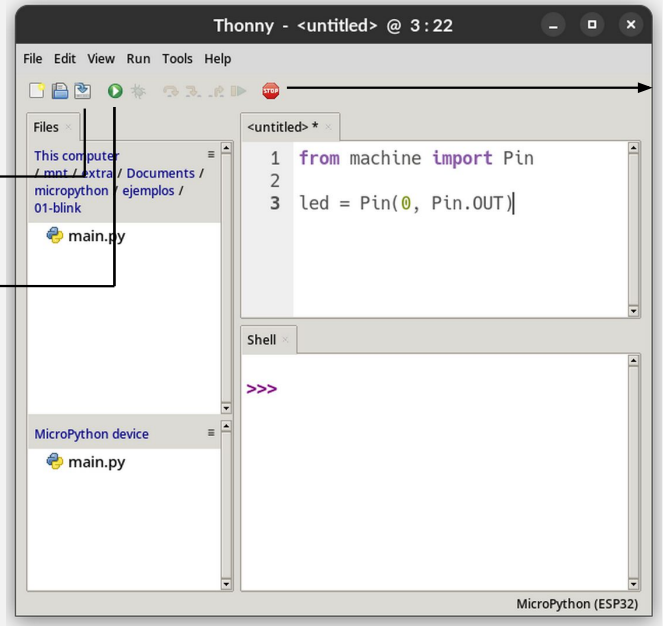
```
>>>
>>> from machine import Pin
>>> boton = Pin(33, Pin.IN, Pin.PULL_UP)
>>> led = Pin(0, Pin.OUT)
>>> while True:
        estado = not boton.value()  # invertir estado
        led.value(estado)
```

MicroPython (ESP32)

Ejemplos

Guardar código actual

Ejecutar

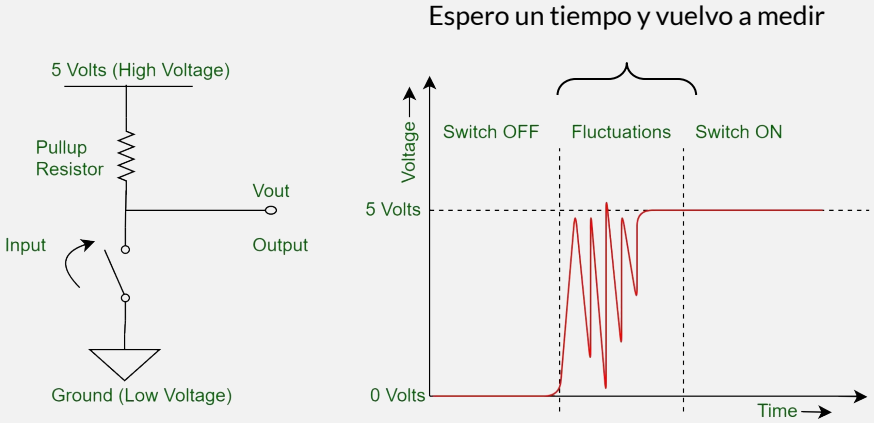


Detener ejecución

Ejemplo 2

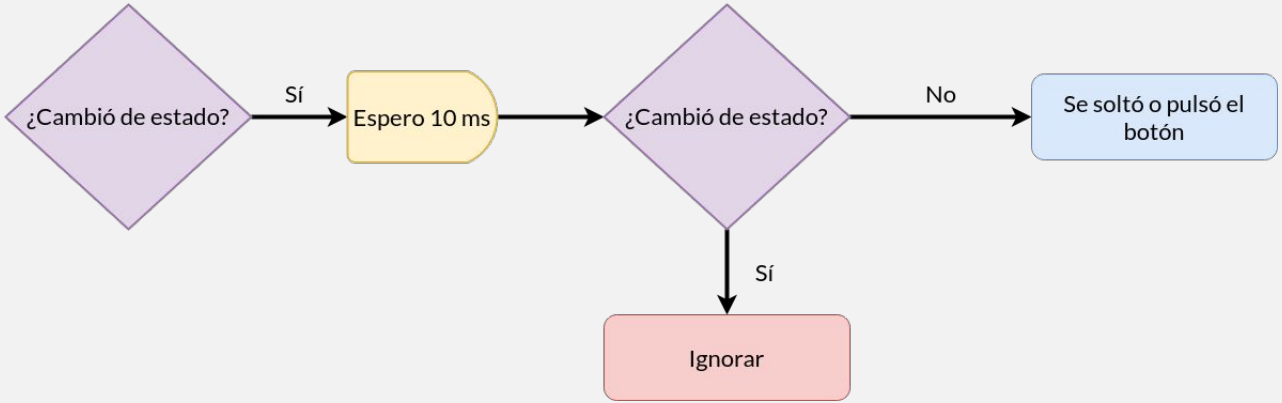
- Incrementar un contador al pulsar un botón. Decrementarlo al pulsar el otro. Mostrar el valor del contador por cada pulsación

Rebote de botones



Solución para el rebote de un botón

- Hardware
  - Filtro RC para obtener una transición suave
- Software
  - Implementar un retardo y sensar dos veces el estado



Solución para el rebote de un botón

```
1  from machine import Pin
2  from time import sleep_ms
3
4  boton = Pin(33, Pin.IN, Pin.PULL_UP)
5
6  ultimo_estado_boton = False
7
8  while True:
9      estado_boton = not boton.value()
10
11     if estado_boton != ultimo_estado_boton:
12         sleep_ms(10)
13         nuevo_estado_boton = not boton.value()
14         if nuevo_estado_boton == estado_boton and estado_boton:
15             print("Pulsado")
16
17     ultimo_estado_boton = estado_boton
```

Solución para el rebote de un botón

- Uso de interrupciones
  - Código más legible al incorporar varios comportamientos

```
boton.irq(handler=irq_boton, trigger=Pin.IRQ_FALLING)
```

Función a invocar al momento de producirse la interrupción

Disparador de la interrupción
`IRQ_FALLING`: flanco descendente
`IRQ_RISING`: flanco ascendente

Solución para el rebote de un botón

```
1  from machine import Pin, disable_irq, enable_irq
2  from time import sleep_ms
3
4  def irq_boton(pin):
5      s = disable_irq()
6      sleep_ms(10)
7      if pin.value() == False:
8          print("Pulsado")
9      enable_irq(s)
10
11 boton = Pin(33, Pin.IN, Pin.PULL_UP)
12
13 boton.irq(handler=irq_boton, trigger=Pin.IRQ_FALLING)
```

Deshabilito interrupciones mientras proceso

Habilito interrupciones al finalizar

Ejercicios

1. Construir la secuencia de un semáforo avanzando cada estado al pulsar un botón
2. Modificar el programa del parpadeo del LED para que al pulsar un botón, parpadee más rápido y al pulsar el otro botón parpadee más lento