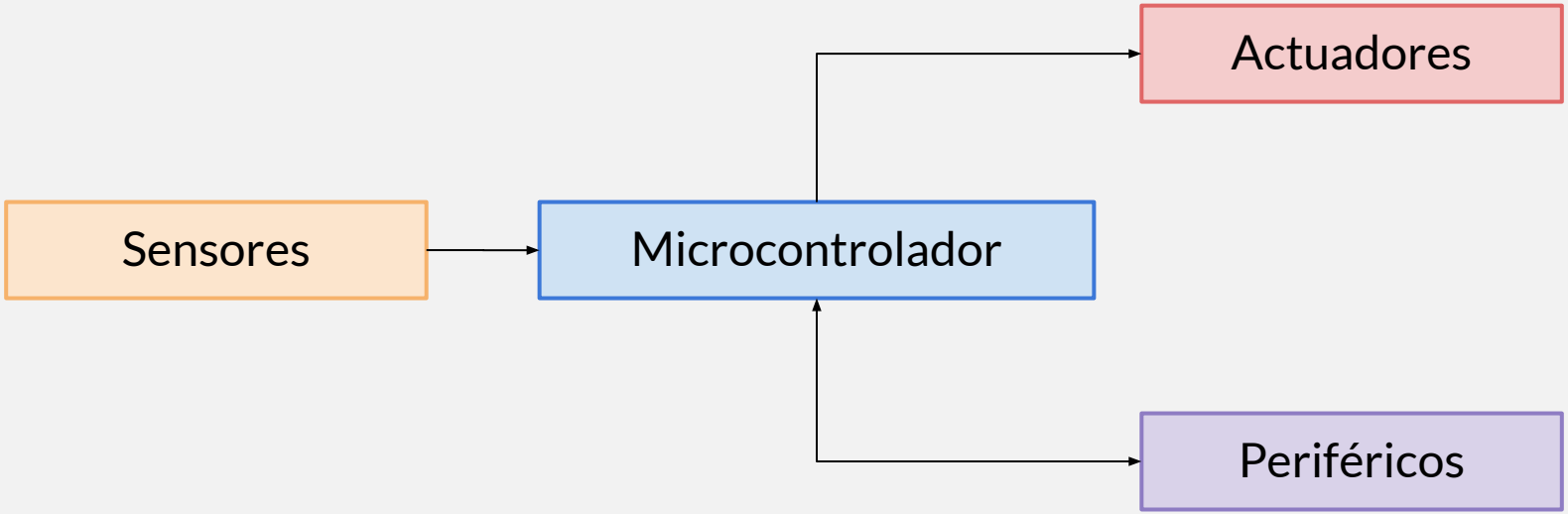
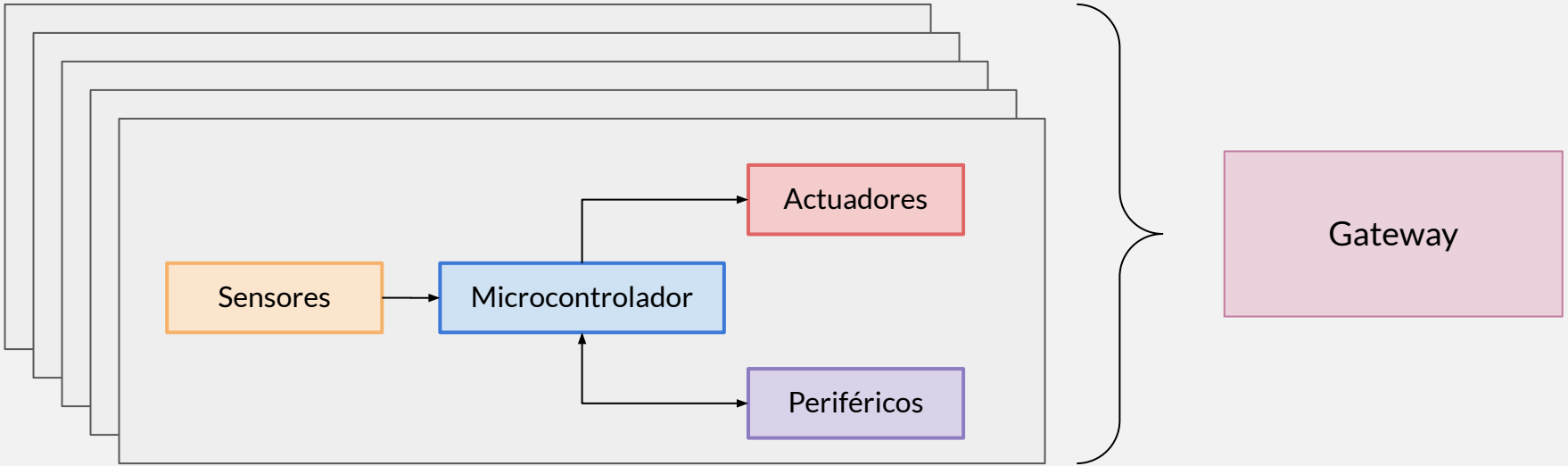
NEON
Network of Competence on Internet of Things
Co-funded by the Erasmus+ Programme of the European Union

Hardware IoT

Hardware IoT

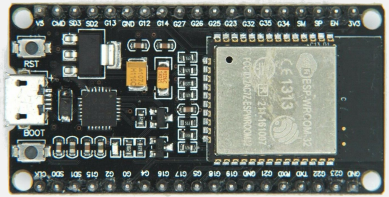


Hardware IoT



Placas de desarrollo

Arduino UNO

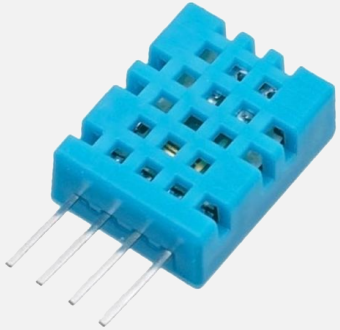


ESP32

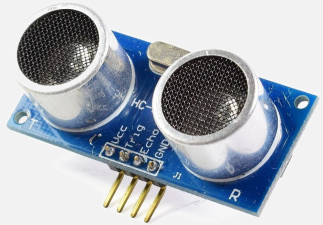
Raspberry Pi Pico

Sensores

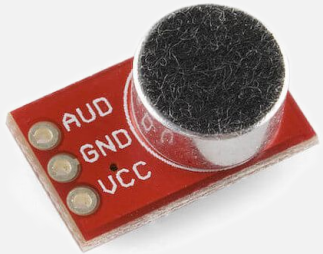
- Recopilar datos del entorno
- Procesar información
- Determinar cómo actuar



DHT11
Temperatura y humedad



HC-SR04
Distancia



Micrófono
Sonido

MQ-5
Medidor de gas metano

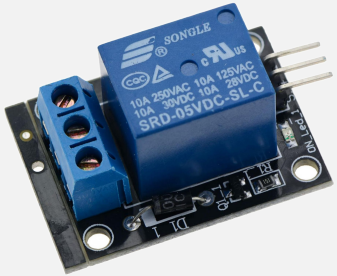
Actuadores y periféricos

Actuadores

- Relé
- Servomotor

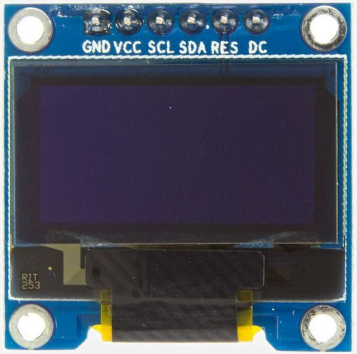
Periféricos

- Indicadores
- Pulsadores

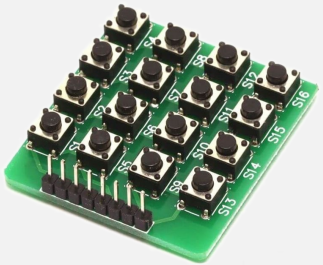


Relé

Servomotor



Pantalla OLED



Matriz de botones

Gateway

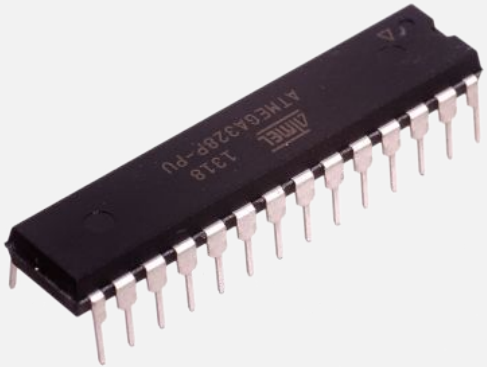
Raspberry Pi

PC Servidor

Gateway Industrial

Microcontroladores

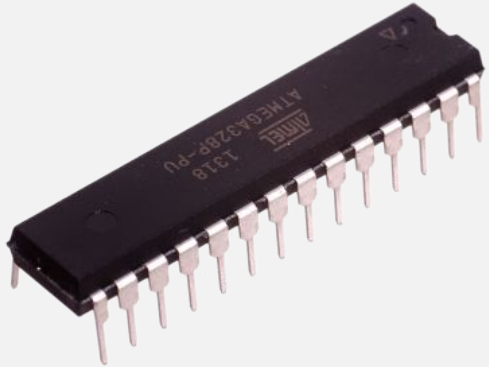
Microcontroladores



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

- Variedad de modelos para diferentes aplicaciones (ESP32, ESP32-S, ESP32-C, ESP32-H)
- Diferentes protocolos de conectividad según modelo (WiFi, BLE, ZigBee)
- Variedad de protocolos soportados de forma oficial (HTTP, MQTT, ...)
- Compatible con Arduino
- Compatible con MicroPython

Microcontroladores - RP2040

- Desarrollado por Raspberry Pi y pensado para trabajar en conjunto
- Único modelo
- Más económico en comparación a ESP32 y Arduino de similares prestaciones
- Compatible con Arduino
- Compatible con MicroPython

Kit de Desarrollo de Software IoT

NEON

Facultad de Ingeniería

Universidad Nacional de Mar del Plata

Kit de Desarrollo Arduino

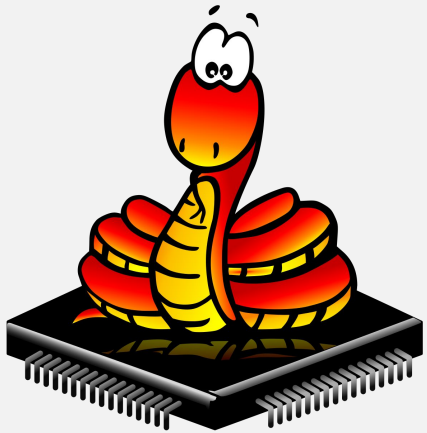
- Ampliamente utilizado
- Ecosistema de periféricos y sensores
- Variedad de microcontroladores (8 y 32 bits)
- Fácil de utilizar
- Lenguaje: C++

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++

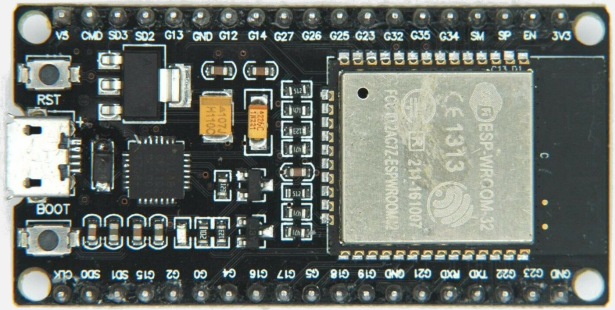
Kit de Desarrollo MicroPython

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

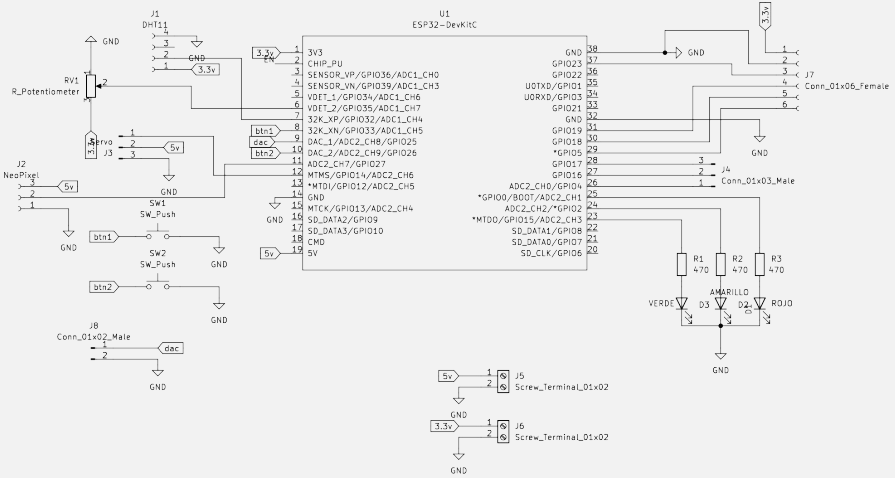


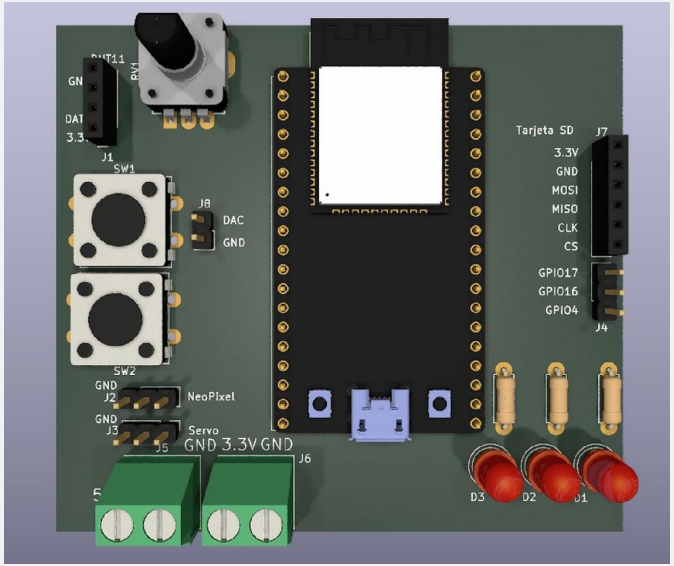
Kit de Desarrollo Seleccionado

- ESP32
- MicroPython



Placa de Desarrollo





Introducción a Python

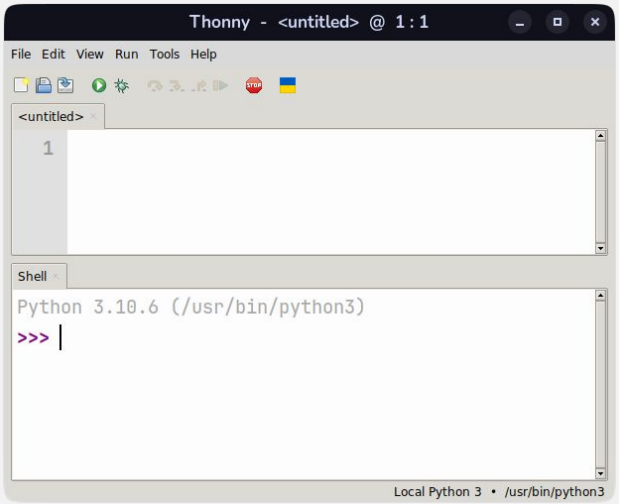
¿Qué es Python?

- Lenguaje de programación de alto nivel
- Orientado a objetos
- Tipos dinámicos
- Interpretado

Referencia: https://docs.python.org/es/3.7/tutorial/index.html

Intérprete de comandos

Por defecto, el IDE Thonny utiliza el intérprete de comandos de Python instalado.



Editor de código

Intérprete de Python

Variables

No requieren declaración. Se asignan de forma directa

```
>>> x = 10
>>> y = x * 2
```

Números

```
>>> x = 10      # entero
>>> x = 10.0    # punto flotante
>>> 10 / 3
3.3333333333333335
>>> 10 // 2
3
>>> int(10 / 3)
3
```

Cadenas de texto

```
>>> a = "abc"
>>> b = 'def'
>>> a + b
'abcdef'
>>> f"{a}{b}"
'abcdef'
```

Variables

Cadenas de texto

```
>>> a = "abc"
>>> a.endswith("bc")
True
>>> a.startswith("bc")
False
>>> a.find("b")
1
>>> a.find("d")
-1
```

```
>>> b = "abc      \n"
>>> b.strip()
'abc'
>>> c = "a b c"
>>> c.split(" ")
['a', 'b', 'c']
```

Listas

```
>>> lista = [1, 2, 3]
>>> lista[0]
1
>>> lista[0:2]
[1, 2]
>>> lista[-1]
3
```

Listas

```
>>> lista = [1, 2, 3]
>>> lista.append(4)
>>> lista
[1, 2, 3, 4]
>>> lista.insert(0, 5)
>>> lista
[5, 1, 2, 3, 4]
>>> del lista[2]
>>> lista
[5, 1, 3, 4]
```

Salida

```
>>> print("abc")
abc
>>> x = 123
>>> print(x)
123
>>> print(x, end="(fin linea)\n")
123(fin linea)
```

Operadores lógicos

```
>>> 1 == 1
True
>>> 1 != 1
False
>>> 1 <= 4
True
>>> 4 >= 4
True
```

```
>>> 1 < 2 and 2 < 3
True
>>> 1 == 1 or 2 == 3
True
>>> not 1 > 2
True
>>> 1 in [1, 2, 3]
True
```

Condicionales

Deben respetarse la misma cantidad de espacios

```
>>> a = 1
>>> if a == 1:
...     print("a es 1")
... else:
...     print("a no es 1")
...
a es 1
```

Ciclos

```
>>> for i in range(3):
...     print(i)
...
0
1
2
```

```
>>> i = 0
>>> while i < 3:
...     i += 1
...     print(i)
...
1
2
3
```

Ciclos

```
>>> for c in "ab":
...     print(c)
...
a
b
```

```
>>> for x in [20, 4]:
...     print(x)
...
20
4
```

Funciones

```
>>> def funcion(a, b, c):
...     return a + b * c
...
>>> funcion(1, 2, 3)
7
>>> funcion(1, 2, "a")
Traceback (most recent call last):
 File "<stdin>", line 1, in <module>
 File "<stdin>", line 2, in funcion
TypeError: unsupported operand type(s) for +: 'int' and
'str'
```

Excepciones

```
>>> a = [1, 2]
>>> a[3]
Traceback (most recent call last):
 File "<stdin>", line 1, in <module>
IndexError: list index out of range
>>> 1 + "b"
Traceback (most recent call last):
 File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for +: 'int' and
'str'
```

Try-Except

```
>>> try:
...     1 + "b"
... except TypeError:
...     print("Los tipos no coinciden")
...
Los tipos no coinciden
```

Módulos

modulo1.py

```
def funcion(a, b):
    if a > b:
        return 1
    elif a == b:
        return 0
    return -1
```

```
>>> from modulo1 import funcion
>>> funcion(2, 1)
1
>>> import modulo1
>>> modulo1.funcion(1, 1)
0
```

Introducción a MicroPython

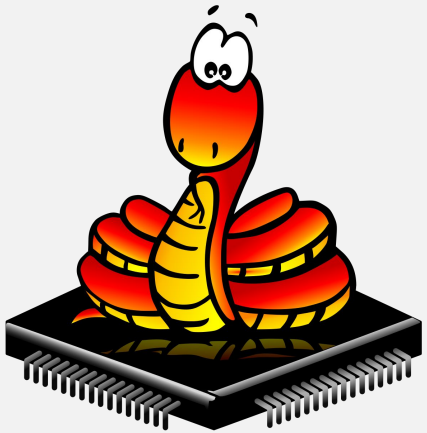
NEON

Facultad de Ingeniería

Universidad Nacional de Mar del Plata

MicroPython

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

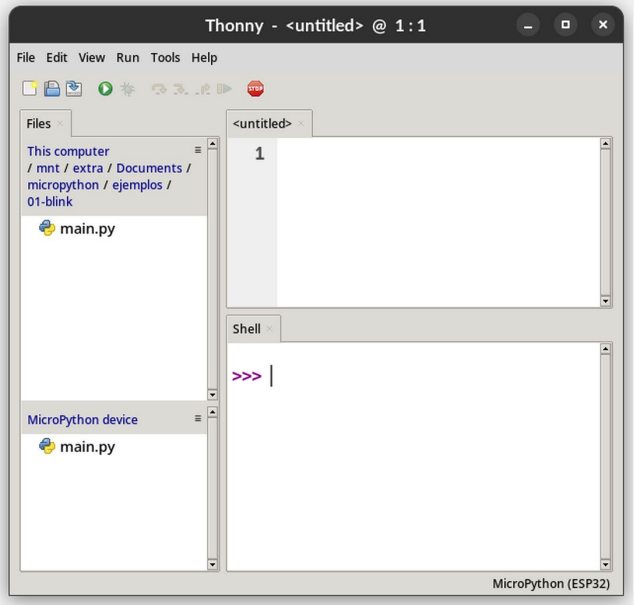


Referencia: https://docs.micropython.org/en/v1.19.1/esp32/quickref.html

Entorno de desarrollo Thonny

Archivos locales

Archivos en ESP32



Editor de código

Intérprete de Python

Intérprete seleccionado

`machine`

Contiene funciones relacionadas con el hardware

- Pin: permite controlar entradas y salidas digitales
  - Parámetros: GPIO, Modo, resistencia Pull-up
  - Establecer nivel alto o bajo para modo **OUT**
  - Determinar el nivel para modo **IN** o **OUT**

Ejemplos

```
>>> from machine import Pin
>>> salida = Pin(14, Pin.OUT)
>>> salida.on()  # equivalente a salida.value(1)
>>> salida.off() # equivalente a salida.value(0)
>>> entrada = Pin(15, Pin.IN, Pin.PULL_UP)
>>> entrada.value()
1
```

`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

Ejemplos

```
>>> from time import sleep_ms, ticks_ms
>>> inicio = ticks_ms()
>>> sleep_ms(1000)
>>> fin = ticks_ms()
>>> fin - inicio
1000
```

Ejemplos

Ejemplo 1

Encender y apagar un LED cada intervalo fijo de tiempo. El LED debe permanecer encendido el mismo tiempo que apagado.

Ejemplos

```
from machine import Pin
from time import sleep_ms

led = Pin(0, Pin.OUT)

while True:
  led.on()
  sleep_ms(500)
  led.off()
  sleep_ms(500)
```

Ejemplos

```
from machine import Pin
from time import sleep_ms

led = Pin(0, Pin.OUT)

while True:
  led.on()
  sleep_ms(500)
  led.off()
  sleep_ms(500)
```

Ejemplos

```
from machine import Pin
from time import sleep_ms

led = Pin(0, Pin.OUT)

while True:
  led.on()
  sleep_ms(500)
  led.off()
  sleep_ms(500)
```

Ejemplos - Alternativa

```
led = Pin(0, Pin.OUT)
ultimo_parpadeo = 0

while True:
    if ticks_ms() - ultimo_parpadeo > 1000:
        if led.value() == 0:
            led.on()
        else:
            led.off()
        ultimo_parpadeo = ticks_ms()
```

Ejemplos - Alternativa

```
led = Pin(0, Pin.OUT)
ultimo_parpadeo = 0

while True:
    if ticks_ms() - ultimo_parpadeo > 1000:
        if led.value() == 0:
            led.on()
        else:
            led.off()
        ultimo_parpadeo = ticks_ms()
```

Ejemplos - Alternativa

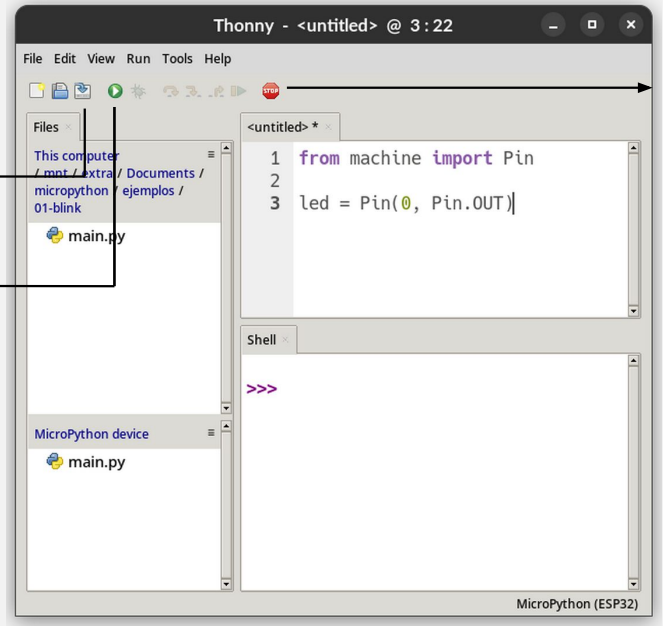
```
led = Pin(0, Pin.OUT)
ultimo_parpadeo = 0

while True:
    if ticks_ms() - ultimo_parpadeo > 1000:
        if led.value() == 0:
            led.on()
        else:
            led.off()
        ultimo_parpadeo = ticks_ms()
```

Ejemplos

Guardar código actual

Ejecutar

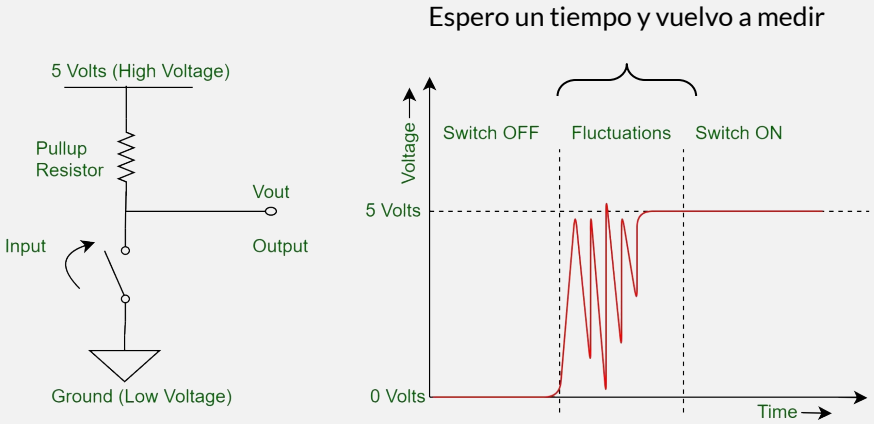


Detener ejecución

Ejemplo 2

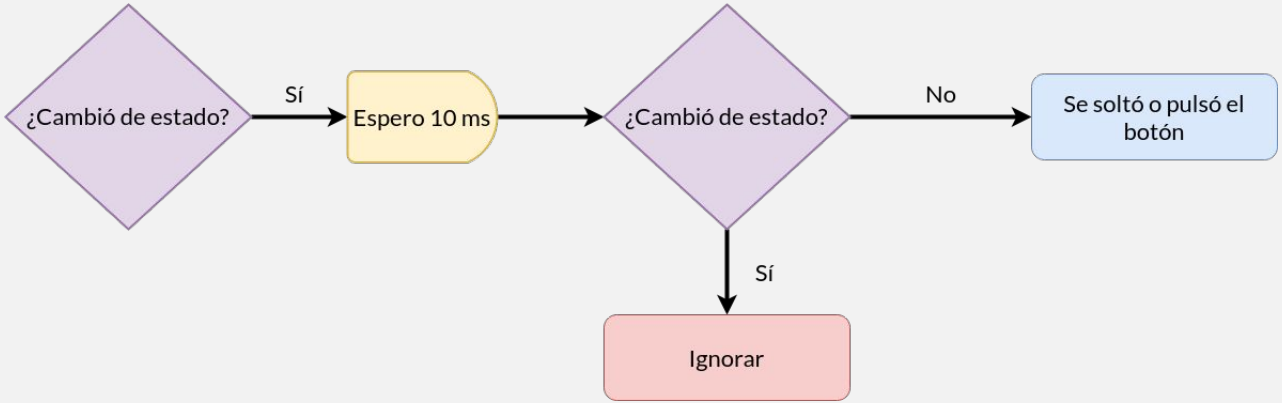
- Incrementar un contador al pulsar un botón. Decrementarlo al pulsar el otro. Mostrar el valor del contador por cada pulsación

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

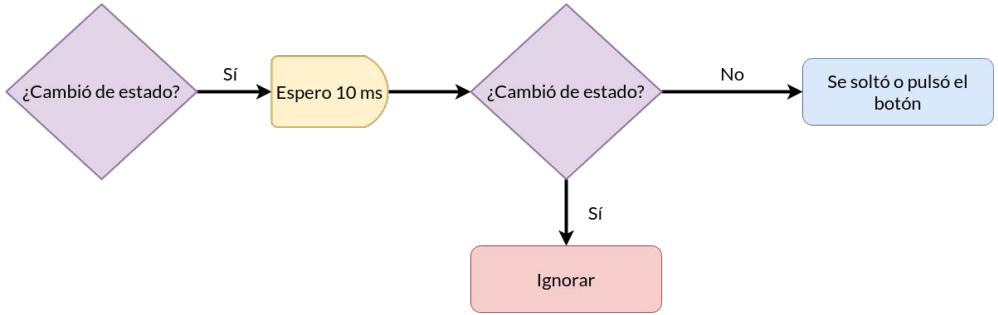


Solución para el rebote de un botón

```
ultimo_estado = False
contador = 0
while True:
    estado = not boton.value()
    if estado != ultimo_estado:
        sleep_ms(50)
        nuevo_estado = not boton.value()
        if nuevo_estado:
            contador += 1
            print(f"Pulsado {contador} veces")
    ultimo_estado = estado
```

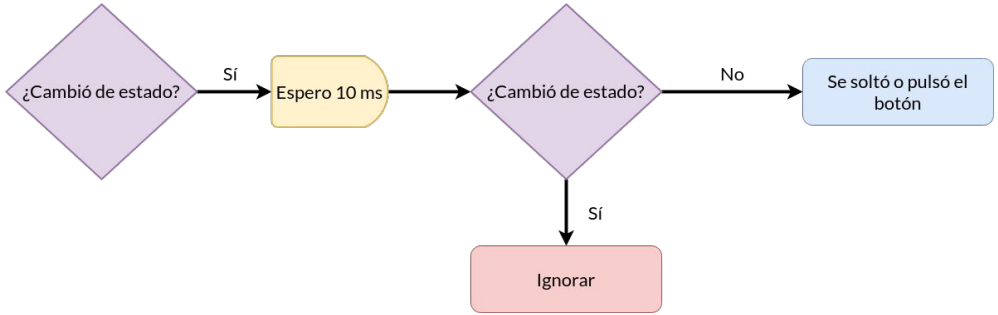
Solución para el rebote de un botón

```
ultimo_estado = False
contador = 0
while True:
    estado = not boton.value()
    if estado != ultimo_estado:
        sleep_ms(50)
        nuevo_estado = not boton.value()
        if nuevo_estado:
            contador += 1
            print(f"Pulsado {contador} veces")
    ultimo_estado = estado
```



Solución para el rebote de un botón

```
ultimo_estado = False
contador = 0
while True:
    estado = not boton.value()
    if estado != ultimo_estado:
        sleep_ms(50)
        nuevo_estado = not boton.value()
        if nuevo_estado:
            contador += 1
            print(f"Pulsado {contador} veces")
    ultimo_estado = estado
```



Solución para el rebote de un botón

```
ultimo_estado = False
contador = 0
while True:
    estado = not boton.value()
    if estado != ultimo_estado:
        sleep_ms(50)
        nuevo_estado = not boton.value()
        if nuevo_estado:
            contador += 1
            print(f"Pulsado {contador} veces")
    ultimo_estado = estado
```

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

- **Ejercicio 1**
  - Construir la secuencia de un semáforo avanzando cada estado al pulsar un botón
- **Ejercicio 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