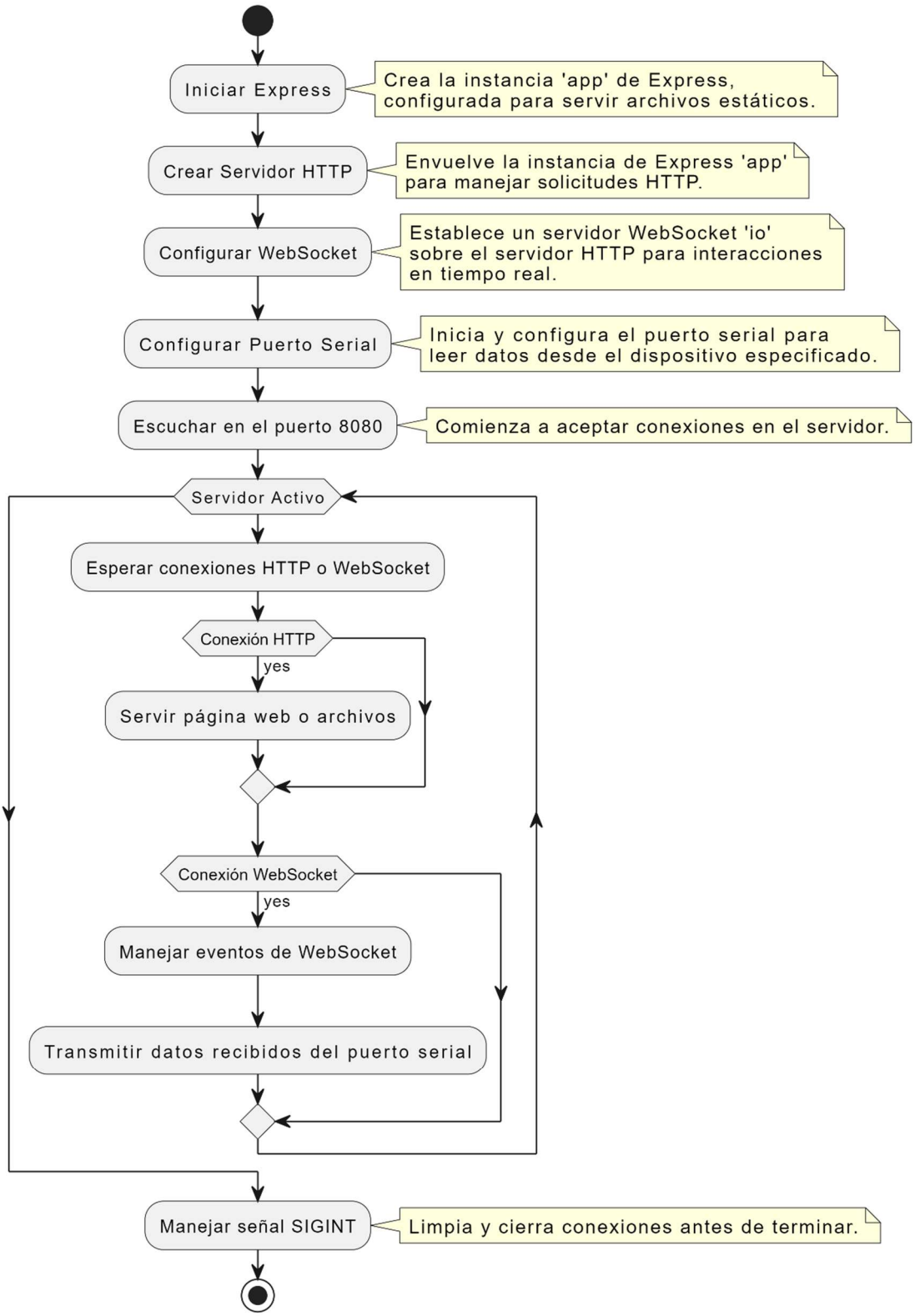
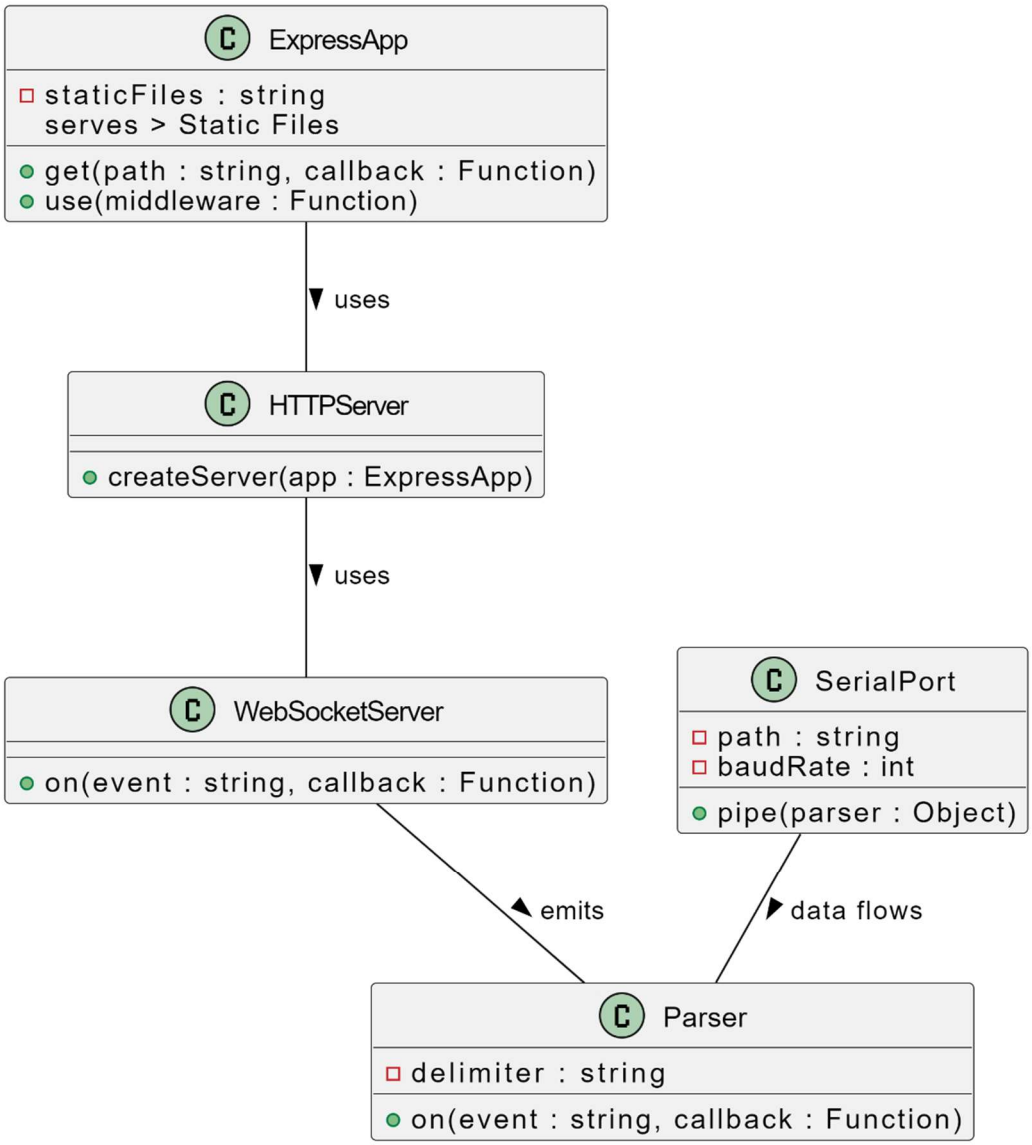
serverSerialRTC

- **server**

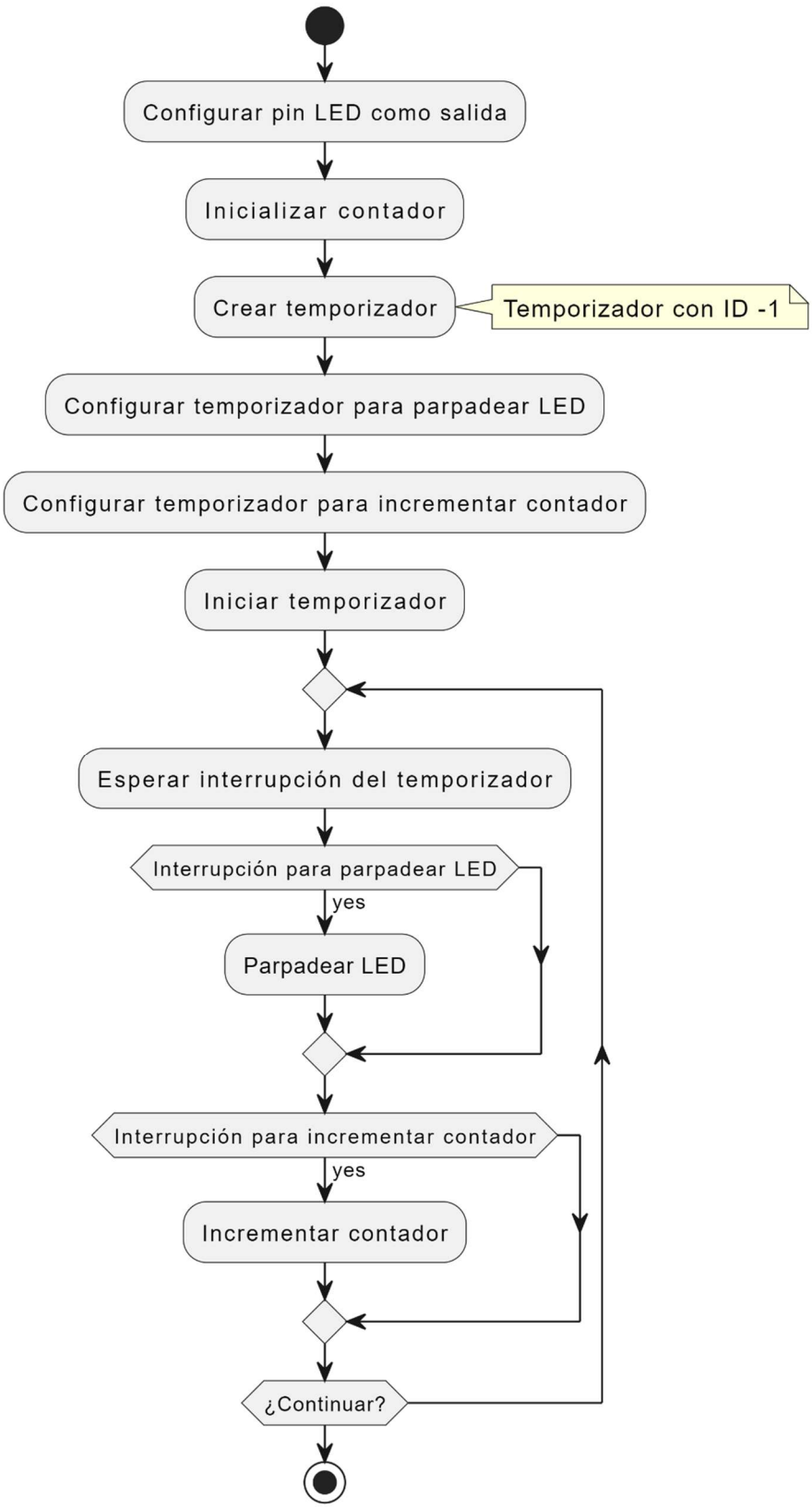
```
// Carga el módulo Express para manejar la lógica del servidor web.
const express = require('express');

// Crea una instancia de una aplicación Express.
const app = express();

// Carga el módulo HTTP y crea un servidor HTTP usando la instancia de Express.
const http = require('http');
const server = http.createServer(app);

// Carga el módulo Socket.IO y crea un servidor de WebSocket enlazado al servidor HTTP.
const { Server } = require("socket.io");
const io = new Server(server);

// Carga el módulo SerialPort y configura la conexión al puerto serial para recibir datos.
const { SerialPort, ReadlineParser } = require('serialport');
const port = new SerialPort({
  path: '/dev/ttyACM0',
  baudRate: 9600 });

// Crea un parser que procesará los datos del puerto serial basándose en delimitadores específicos.
const parser = port.pipe(new ReadlineParser({ delimiter: '\r\n' }));

// Configura la aplicación Express para servir archivos estáticos desde el directorio 'public'.
app.use(express.static('public'));

// Define una ruta para la raíz que simplemente envía "Hello World!" al navegador.
app.get('/', (req, res) => {
 res.send('Hello World!');
});

// Maneja eventos de conexión WebSocket.
io.on('connection', (socket) => {
 console.log('a user connected');

 // Establece un manejador para los datos recibidos a través del puerto serial.
 parser.on('data', (data) => {
  // Escribe los datos recibidos en la consola del servidor y los transmite a los clientes conectados.
  process.stdout.write(data + "\r");
  socket.emit('time', data); // Envia el estado recibido al cliente conectado.
 });
});

// Inicia el servidor en el puerto 8080 y registra un mensaje en la consola.
server.listen(8080, () => {
 console.log('listening on *:8080');
});
```

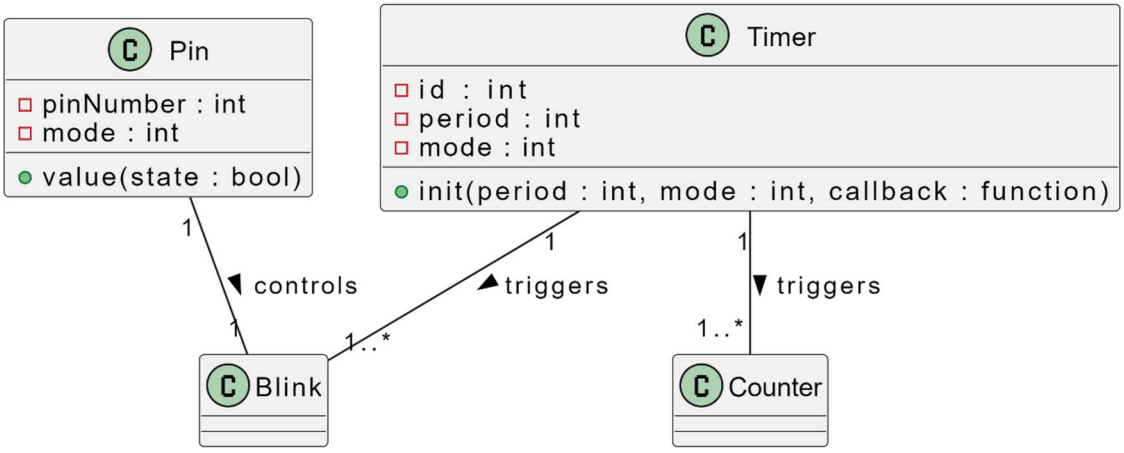

Iniciar Express
Crea la instancia 'app' de Express, configurada para servir archivos estáticos.
Crear Servidor HTTP
Envuelve la instancia de Express 'app' para manejar solicitudes HTTP.
Configurar WebSocket
Establece un servidor WebSocket 'io' sobre el servidor HTTP para interacciones en tiempo real.
Configurar Puerto Serial
Inicia y configura el puerto serial para leer datos desde el dispositivo especificado.
Escuchar en el puerto 8080
Comienza a aceptar conexiones en el servidor.
Servidor Activo
Esperar conexiones HTTP o WebSocket
Conexión HTTP
yes
Servir página web o archivos
Conexión WebSocket
yes
Manejar eventos de WebSocket
Transmitir datos recibidos del puerto serial
Manejar señal SIGINT
Limpia y cierra conexiones antes de terminar.


ExpressApp
staticFiles : string
serves > Static Files
get(path : string, callback : Function)
use(middleware : Function)
uses
HTTPServer
createServer(app : ExpressApp)
uses
WebSocketServer
on(event : string, callback : Function)
SerialPort
path : string
baudRate : int
pipe(parser : Object)
emits
data flows
Parser
delimiter : string
on(event : string, callback : Function)

- # rtc

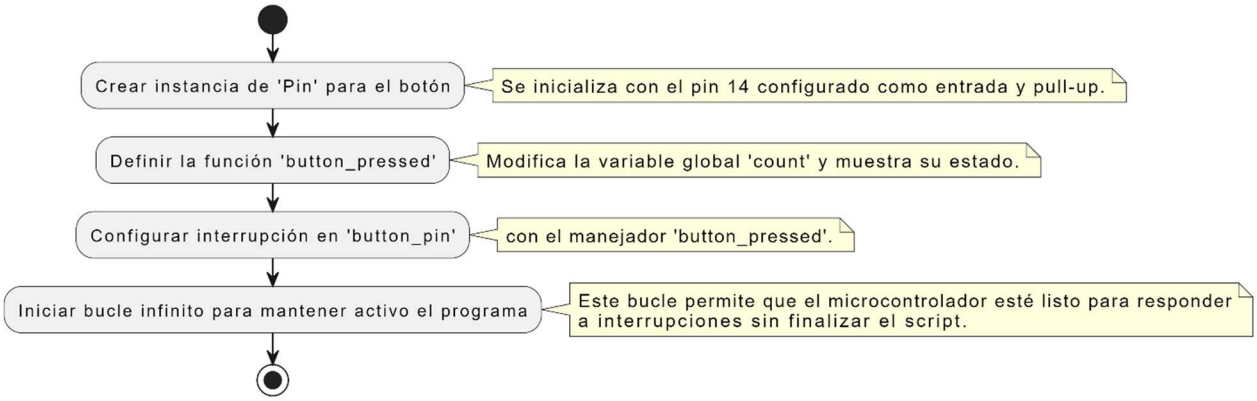
```
# Importación del módulo Timer y machine para controlar los GPIOs.
from machine import Timer
import machine

# Inicialización de una variable global 'count' que se usa como contador.
count = 0

# Configuración del pin del LED como salida.
# 'led_pin' es una instancia de la clase 'Pin' con el pin 25 configurado como salida.
led_pin = machine.Pin(25, machine.Pin.OUT)

# Definición de la función 'blink' que actúa como callback para el temporizador.
# Esta función toma un objeto 'timer' y cambia el estado del LED.
def blink(timer):
  # Cambia el valor del pin del LED al valor opuesto (encendido/apagado).
  led_pin.value(not led_pin.value())

# Definición de la función 'counter' que también actúa como callback para el temporizador.
# Incrementa la variable global 'count' y la imprime.
def counter(timer):
  global count
  # Incrementa 'count' y muestra su valor en la consola.
  print(count)
  count += 1

# Creación de una instancia de la clase 'Timer'.
# 'timer' es un objeto de la clase 'Timer' que se utiliza para crear temporizadores.
timer = Timer(-1)

# Configuración del temporizador para llamar a la función 'blink' cada 1000 milisegundos (1 segundo).
timer.init(period=1000, mode=Timer.PERIODIC, callback=blink)

# Configuración del temporizador para llamar a la función 'counter' cada 1000 milisegundos (1 segundo).
timer.init(period=1000, mode=Timer.PERIODIC, callback=counter)

# Comentarios sobre el modo de suspensión profunda no implementado en el código proporcionado.
```

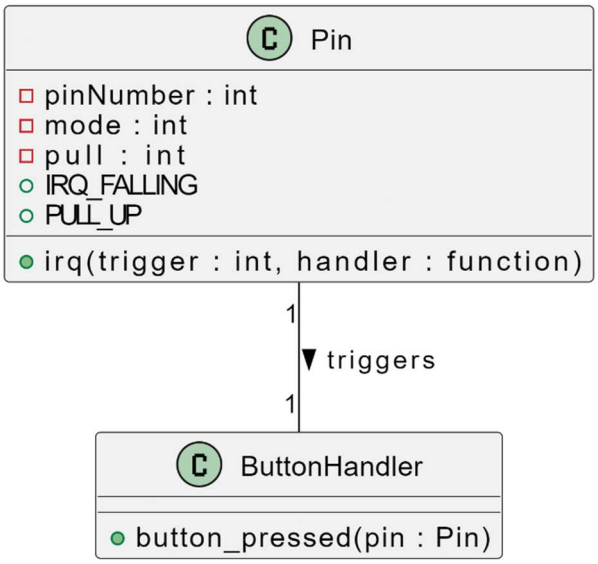

Configurar pin LED como salida
Inicializar contador
Crear temporizador
Temporizador con ID -1
Configurar temporizador para parpadear LED
Configurar temporizador para incrementar contador
Iniciar temporizador
Esperar interrupción del temporizador
Interrupción para parpadear LED
yes
Parpadear LED
Interrupción para incrementar contador
yes
Incrementar contador
¿Continuar?


Pin
pinNumber : int
mode : int
value(state : bool)
Timer
id : int
period : int
mode : int
init(period : int, mode : int, callback : function)
1
controls
1
1
triggers
1..*
1
triggers
1..*
Blink
Counter

- **rtc2**

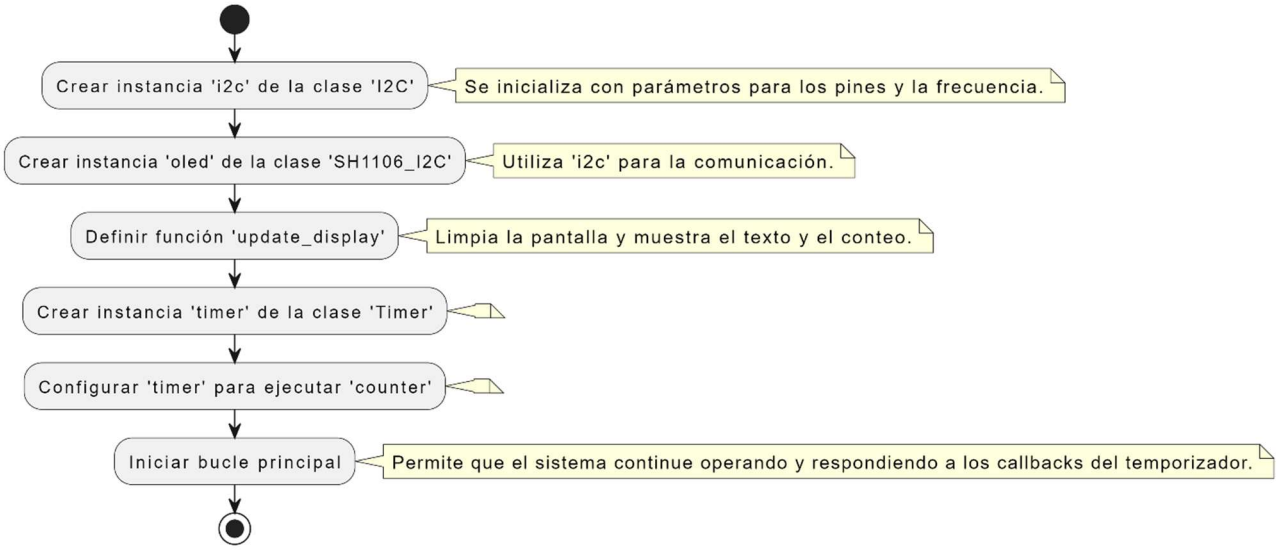
```
# Importa la clase Pin desde el módulo machine para control de GPIO.
from machine import Pin
import machine

# Inicializa una variable global para contar las pulsaciones del botón.
count = 0

# Configura el pin del botón como entrada con resistencia de pull-up.
# 'button_pin' es una instancia de la clase 'Pin', configurada para leer entradas.
button_pin = Pin(14, Pin.IN, Pin.PULL_UP)

# Define la función que se ejecutará cuando se detecte una pulsación del botón.
# Esta función modifica la variable global 'count' y muestra su valor.
def button_pressed(pin):
  global count
  count += 1  # Incrementa 'count' cada vez que se presiona el botón.
  print(count)  # Muestra el valor actual de 'count'.

# Configura una interrupción en el pin del botón para llamar a 'button_pressed' cuando el botón se presione.
button_pin.irq(trigger=Pin.IRQ_FALLING, handler=button_pressed)

# Ciclo infinito para mantener el programa en ejecución.
# Mantiene el programa activo para que pueda responder a interrupciones.
while True:
  pass  # Este lugar es adecuado para añadir más lógica si es necesario.
```

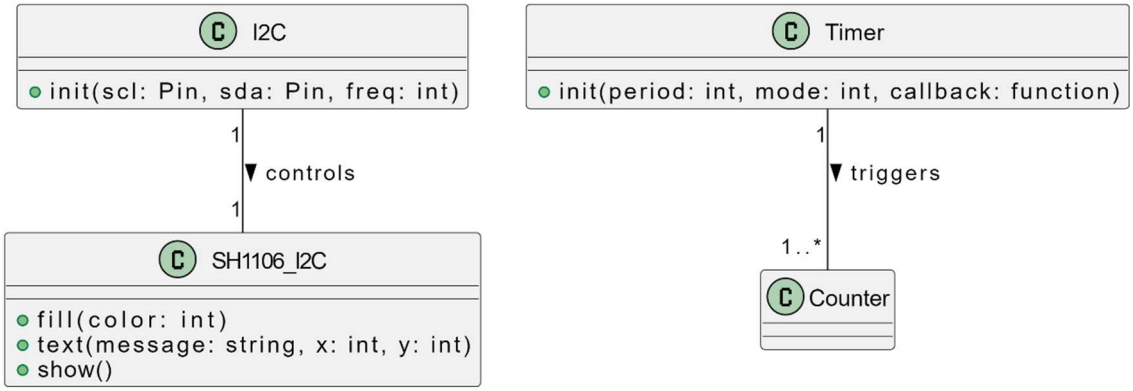



Pin
pinNumber : int
mode : int
pull : int
IRQ_FALLING
PULL_UP
irq(trigger : int, handler : function)
1
triggers
1
ButtonHandler
button_pressed(pin : Pin)

- # rtcoled1

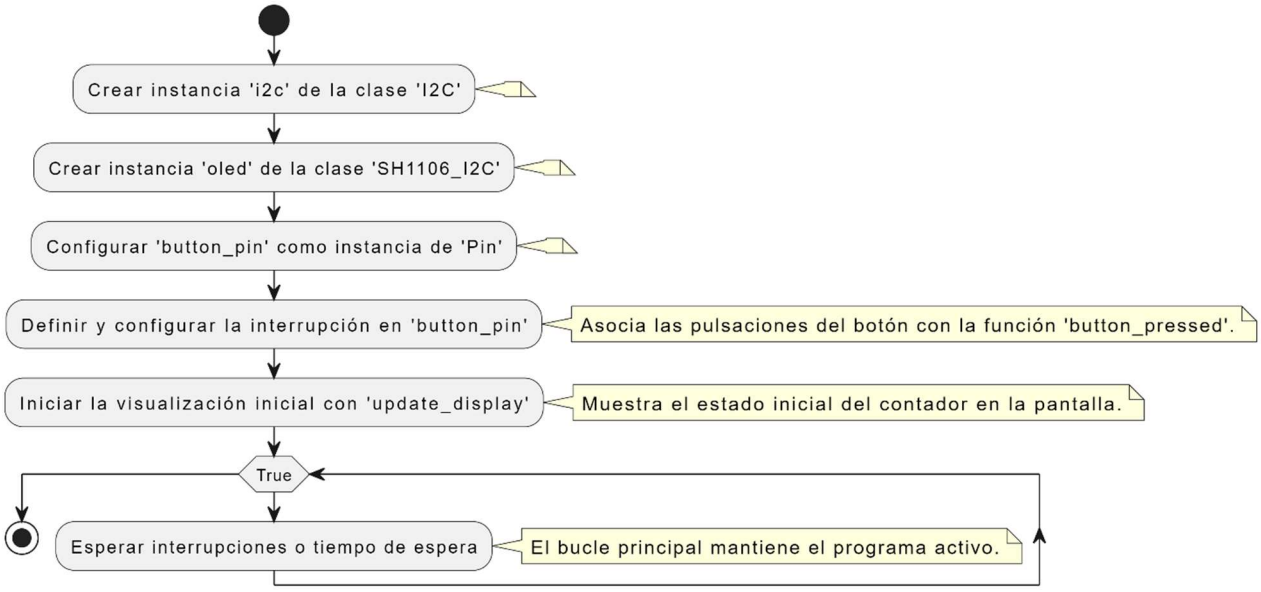
```
# Importa las clases Timer, I2C y Pin del módulo machine.
from machine import Timer, I2C, Pin
# Importa la biblioteca sh1106 para interactuar con la pantalla OLED.
import sh1106
# Importa la biblioteca time para manejar funciones relacionadas con el tiempo.
import time

# Inicializa una variable global para llevar un conteo.
count = 0

# Configura el bus I2C para la pantalla OLED.
# 'i2c' es una instancia de la clase 'I2C' configurada con pines específicos y frecuencia.
i2c = I2C(0, scl=Pin(1), sda=Pin(0), freq=400000)

# Crea una instancia de la pantalla OLED utilizando el bus I2C.
# 'oled' es una instancia de la clase 'SH1106_I2C' que representa la pantalla OLED.
oled = sh1106.SH1106_I2C(128, 64, i2c)

# Define una función para actualizar el contenido mostrado en la pantalla OLED.
def update_display():
  oled.fill(0)  # Limpia la pantalla.
  oled.text('Conteo:', 0, 0)  # Muestra el texto "Conteo:".
  oled.text(str(count), 0, 10)  # Muestra el valor actual del conteo.
  oled.show()  # Envía el buffer gráfico a la pantalla para actualizarla.

# Define una función que actúa como callback para un temporizador.
# Incrementa el contador y actualiza la pantalla.
def counter(timer):
  global count
  count += 1  # Incrementa el contador.
  print(count)  # Imprime el valor actual del contador en la consola.
  update_display()  # Llama a la función para actualizar la pantalla.

# Crea y configura un temporizador.
# 'timer' es un objeto de la clase 'Timer' que se utiliza para ejecutar periódicamente el callback 'counter'.
timer = Timer(-1)
timer.init(period=1000, mode=Timer.PERIODIC, callback=counter)

# El programa mantiene el temporizador activo para actualizar continuamente la pantalla OLED.
```

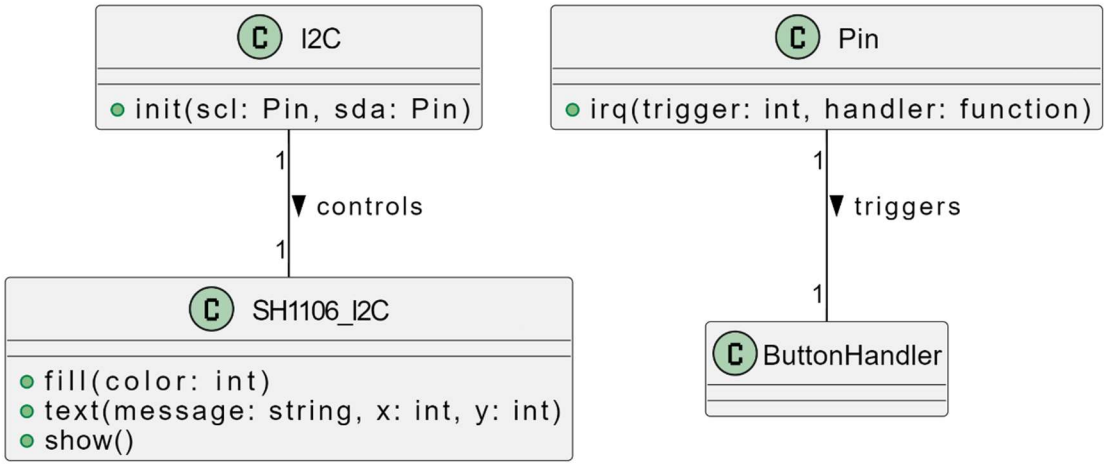

Crear instancia 'i2c' de la clase 'I2C'
Se inicializa con parámetros para los pines y la frecuencia.
Crear instancia 'oled' de la clase 'SH1106_I2C'
Utiliza 'i2c' para la comunicación.
Definir función 'update_display'
Limpia la pantalla y muestra el texto y el conteo.
Crear instancia 'timer' de la clase 'Timer'
Configurar 'timer' para ejecutar 'counter'
Iniciar bucle principal
Permite que el sistema continue operando y respondiendo a los callbacks del temporizador.


I2C
init(scl: Pin, sda: Pin, freq: int)
1
controls
1
SH1106_I2C
fill(color: int)
text(message: string, x: int, y: int)
show()
Timer
init(period: int, mode: int, callback: function)
1
triggers
1..*
Counter

- **rtcoled2**

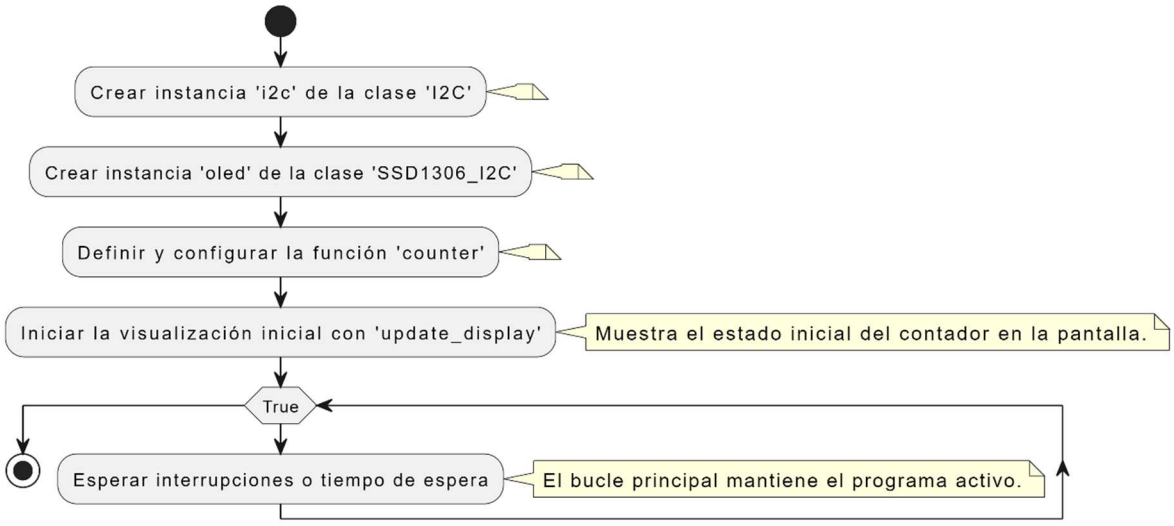
```
# Importa las clases necesarias para el control de pines e I2C.
from machine import Pin, I2C
# Importa la biblioteca sh1106 para la gestión de la pantalla OLED.
import sh1106
# Importa la biblioteca time para pausas y temporización.
import time

# Inicializa un contador global para rastrear la cantidad de presiones de botón.
count = 0

# Crea una instancia de la clase 'I2C' para comunicarse con la pantalla OLED.
i2c = I2C(0, scl=Pin(22), sda=Pin(21))  # Configuración de los pines SCL y SDA.
# Crea una instancia de la pantalla OLED a través de la clase 'SH1106_I2C'.
oled = sh1106.SH1106_I2C(128, 64, i2c)  # Dimensiones de la pantalla OLED.

# Configura un pin para el botón como una entrada con resistencia de pull-up.
button_pin = Pin(14, Pin.IN, Pin.PULL_UP)

# Define una función que se llama cuando el botón es presionado.
def button_pressed(pin):
    global count
    count += 1  # Incrementa el contador global.
    print(count)  # Imprime el contador en la consola.
    update_display()  # Actualiza la pantalla OLED con el nuevo contador.

# Define una función para actualizar el contenido mostrado en la pantalla OLED.
def update_display():
    oled.fill(0)  # Borra el contenido de la pantalla.
    oled.text('Count:', 0, 0)  # Muestra el texto "Count".
    oled.text(str(count), 0, 20)  # Muestra el contador.
    oled.show()  # Envía el contenido al buffer de la pantalla para su visualización.

# Configura una interrupción para detectar pulsaciones del botón.
button_pin.irq(trigger=Pin.IRQ_FALLING, handler=button_pressed)

# Actualiza la pantalla OLED con el estado inicial del contador.
update_display()

# Ciclo infinito para mantener el programa en ejecución y permitir actualizaciones continuas.
while True:
    time.sleep(1)  # Pausa de un segundo para reducir el uso del CPU.
```

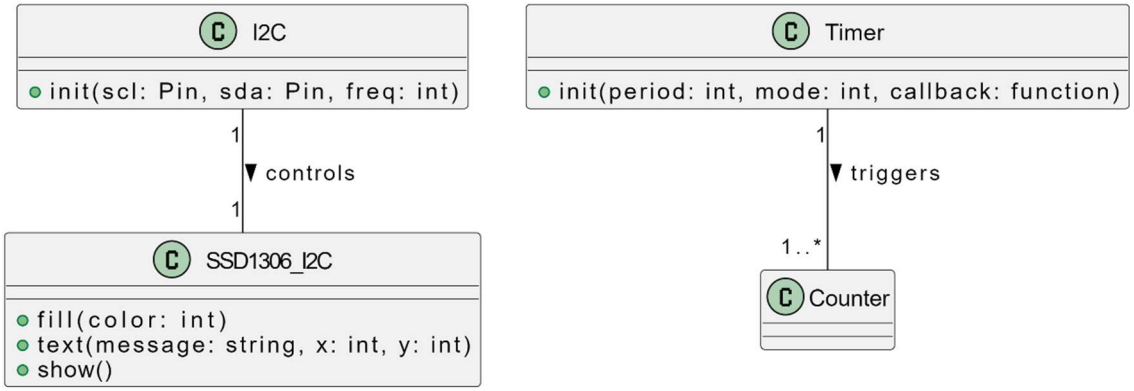

Crear instancia 'i2c' de la clase 'I2C'
Crear instancia 'oled' de la clase 'SH1106_I2C'
Configurar 'button_pin' como instancia de 'Pin'
Definir y configurar la interrupción en 'button_pin'
Asocia las pulsaciones del botón con la función 'button_pressed'.
Iniciar la visualización inicial con 'update_display'
Muestra el estado inicial del contador en la pantalla.
True
Esperar interrupciones o tiempo de espera
El bucle principal mantiene el programa activo.


I2C
init(scl: Pin, sda: Pin)
1
controls
1
SH1106_I2C
fill(color: int)
text(message: string, x: int, y: int)
show()
Pin
irq(trigger: int, handler: function)
1
triggers
1
ButtonHandler

- # rtcoled11

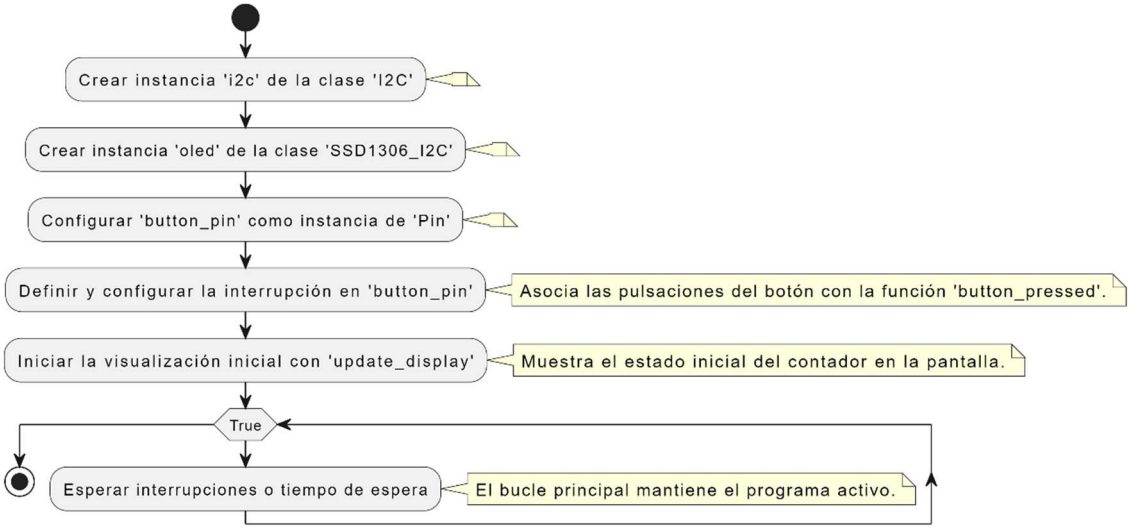
```
# Importación de clases necesarias para manejar el temporizador, I2C, y pines GPIO.
from machine import Timer, I2C, Pin
# Importación de la biblioteca para controlar la pantalla SSD1306.
import ssd1306
# Importación de time para controlar el tiempo de espera en el bucle.
import time

# Inicialización de una variable global para llevar el conteo.
count = 0

# Creación de una instancia de I2C para la comunicación con la pantalla OLED.
i2c = I2C(0, scl=Pin(1), sda=Pin(0), freq=400000)  # Configuración de pines y frecuencia.

# Instanciación de la pantalla OLED utilizando la clase SSD1306_I2C.
oled = ssd1306.SSD1306_I2C(128, 64, i2c)

# Definición de una función para actualizar la pantalla OLED.
def update_display():
    oled.fill(0)  # Borra el contenido de la pantalla.
    oled.text('Conteo:', 0, 0)  # Muestra texto estático.
    oled.text(str(count), 0, 10)  # Muestra el valor de 'count'.
    oled.show()  # Envía los datos al buffer de la pantalla para visualización.

# Definición de la función callback para el temporizador que incrementa y muestra el conteo.
def counter(timer):
    global count
    count += 1  # Incrementa el contador.
    print(count)  # Imprime el valor actualizado del contador.
    update_display()  # Llama a la función de actualización de la pantalla.

# Creación y configuración de un objeto temporizador.
timer = Timer(-1)
timer.init(period=1000, mode=Timer.PERIODIC, callback=counter)  # Configura para ejecutar 'counter' cada segundo.

# Actualización inicial de la pantalla para mostrar el estado inicial.
update_display()

# Bucle infinito para mantener el programa funcionando y permitir actualizaciones continuas.
while True:
    time.sleep(1)  # Reducción del uso de CPU.
```

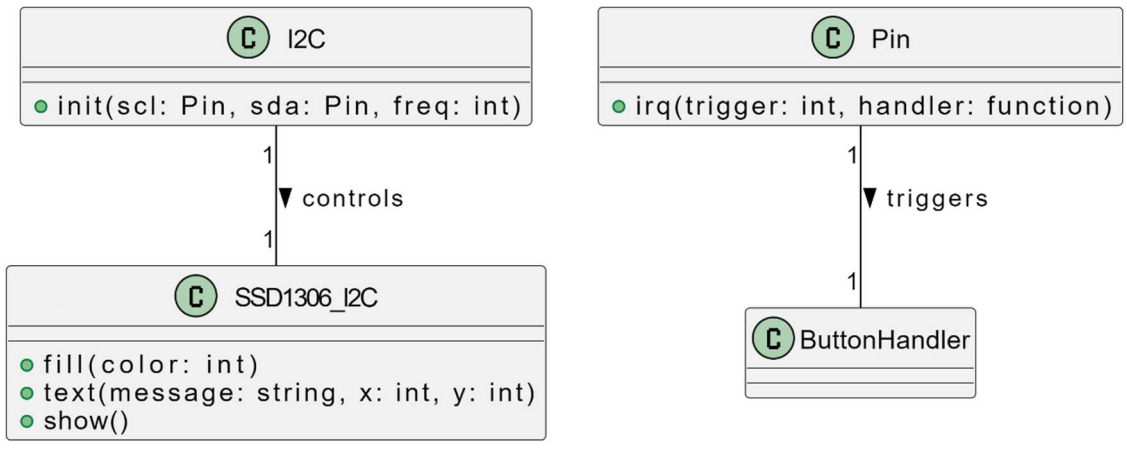

Crear instancia 'i2c' de la clase 'I2C'
Crear instancia 'oled' de la clase 'SSD1306_I2C'
Definir y configurar la función 'counter'
Iniciar la visualización inicial con 'update_display'
Muestra el estado inicial del contador en la pantalla.
True
Esperar interrupciones o tiempo de espera
El bucle principal mantiene el programa activo.


I2C
init(scl: Pin, sda: Pin, freq: int)
1
controls
1
SSD1306_I2C
fill(color: int)
text(message: string, x: int, y: int)
show()
Timer
init(period: int, mode: int, callback: function)
1
triggers
1..*
Counter

- **rtcoled22**

```
# Importa las clases necesarias para la gestión de pines e I2C.
from machine import Pin, I2C
# Importa la biblioteca SSD1306 para controlar la pantalla OLED.
import ssd1306
# Importa la biblioteca time para funciones relacionadas con el tiempo.
import time

# Inicializa un contador global para rastrear la cantidad de veces que se presiona el botón.
count = 0

# Crea una instancia de la clase 'I2C' para la comunicación con la pantalla OLED.
i2c = I2C(0, scl=Pin(22), sda=Pin(21))  # Configuración de los pines SCL y SDA.

# Crea una instancia de la pantalla OLED utilizando la clase 'SSD1306_I2C'.
oled = ssd1306.SSD1306_I2C(128, 64, i2c)  # Especificaciones de la pantalla OLED.

# Configura un pin para el botón como una entrada con resistencia de pull-up.
button_pin = Pin(14, Pin.IN, Pin.PULL_UP)

# Define una función que se ejecuta cuando el botón es presionado.
def button_pressed(pin):
    global count
    count += 1  # Incrementa el contador global.
    print(count)  # Imprime el contador actualizado en la consola.
    update_display()  # Actualiza la pantalla OLED con el nuevo contador.

# Define una función para actualizar el contenido mostrado en la pantalla OLED.
def update_display():
    oled.fill(0)  # Borra el contenido de la pantalla.
    oled.text('Count:', 0, 0)  # Muestra texto estático.
    oled.text(str(count), 0, 20)  # Muestra el valor actual de 'count'.
    oled.show()  # Envía los datos al buffer de la pantalla para su visualización.

# Configura una interrupción para detectar pulsaciones del botón.
button_pin.irq(trigger=Pin.IRQ_FALLING, handler=button_pressed)

# Realiza una actualización inicial de la pantalla para mostrar el estado inicial del contador.
update_display()

# Bucle infinito para mantener el programa funcionando y permitir actualizaciones continuas.
while True:
    time.sleep(1)  # Pausa para reducir el uso de CPU.
```


Crear instancia 'i2c' de la clase 'I2C'
Crear instancia 'oled' de la clase 'SSD1306_I2C'
Configurar 'button_pin' como instancia de 'Pin'
Definir y configurar la interrupción en 'button_pin'
Asocia las pulsaciones del botón con la función 'button_pressed'.
Iniciar la visualización inicial con 'update_display'
Muestra el estado inicial del contador en la pantalla.
True
Esperar interrupciones o tiempo de espera
El bucle principal mantiene el programa activo.


I2C
init(scl: Pin, sda: Pin, freq: int)
1
controls
1
SSD1306_I2C
fill(color: int)
text(message: string, x: int, y: int)
show()
Pin
irq(trigger: int, handler: function)
1
triggers
1
ButtonHandler