

# Jugando

- Nivel de luz
- Temperatura
- Magnetómetro
- Acelerómetro
- Micrófono
- Radio

# Nivel de luz

Página extraída de Federico Coca [Guia de Trabajo de Microbit](#) CC-BY-SA

Vamos a ver como utilizar la micro:bit como detector de luz, cosa que es bastante sencilla por la pantalla de LEDs que puede actuar como sensor.

El programa es:

```
from microbit import *  
  
while True:  
    nivel_luz = display.read_light_level()  
    uart.write(str(nivel_luz) + "\r\n")  
    sleep(1000)
```

Página extraída de Federico Coca [Guia de Trabajo de Microbit](#) CC-BY-SA

[https://www.youtube.com/embed/H\\_6ulkbaShE](https://www.youtube.com/embed/H_6ulkbaShE)

Si quieres realizarlo con la microbit real, tendrás que visualizar el puerto serial, para ello necesitas instalar el programa Putty ver <https://libros.catedu.es/books/microbit-y-python/page/putty>

# Temperatura

La función en micropython para leer la temperatura de la placa en °C interna y devuelve un valor entero es la siguiente

```
microbit.temperature()
```

Un programa ejemplo sería :

```
from microbit import *  
  
while True:  
    Temperatura = temperature()  
    Frase = "T= "+str(Temperatura)+"*C "  
    display.scroll(Frase)  
    sleep(1000)
```

<https://www.youtube.com/embed/uczf6lvUBH4>

# Magnetómetro

Página extraída de Federico Coca [Guia de Trabajo de Microbit](#) CC-BY-SA

Este módulo permite acceder a la brújula electrónica incorporada. Antes de utilizarla, la brújula debe estar calibrada; de lo contrario, las lecturas pueden ser erróneas.

**Advertencia.** Calibrar la brújula hará que su programa se detenga hasta que se complete la calibración. La calibración consiste en un pequeño juego para dibujar un círculo en la pantalla LED girando el dispositivo.

Las funciones son:

- `microbit.compass.calibrate()`

Inicia el proceso de calibración. El mensaje Tilt to Fill Screen (Inclinar para llenar la pantalla) se desplazará en la pantalla que el usuario debe rellenar completamente moviendo el dispositivo.

- `microbit.compass.is_calibrated()`

Devuelve `True` si la brújula se ha calibrado correctamente, y devuelve `False` en caso contrario.

- `microbit.compass.clear_calibration()`

Deshace la calibración, haciendo que la brújula vuelva a estar descalibrada.

- `microbit.compass.get_x()`

- `microbit.compass.get_y()`

- `microbit.compass.get_z()`

Da la lectura, como un número entero positivo o negativo, de la intensidad del campo magnético en el eje especificado dependiendo de la dirección del campo. La medida se da en nano teslas.

- `microbit.compass.heading()`

Da el rumbo de la brújula, calculado a partir de las lecturas anteriores, como un número entero en el rango de 0 a 360, representando el ángulo en grados, en el sentido de las agujas del reloj, con el norte como 0.

- `microbit.compass.get_field_strength()`



Devuelve un valor entero de la magnitud del campo magnético alrededor del dispositivo expresada en nano tesla.

## Ejemplos de actividades con la brújula

Antes de nada debemos calibrar la brújula para después mostrar la posición de la micro:bit utilizando las flechas predefinidas.

### Brújula indicando el norte

```
from microbit import *

# Antes de nada calibrar
compass.calibrate()

# Mantener la aguja apuntando aproximadamente en la dirección correcta.
while True:
    sleep(100)
    aguja = ((15 - compass.heading()) // 30) % 12
    display.show(Image.ALL_CLOCKS[aguja])
```

### Brújula N,S,E,O

El programa, de muy poca precisión es el siguiente:

```
from microbit import *

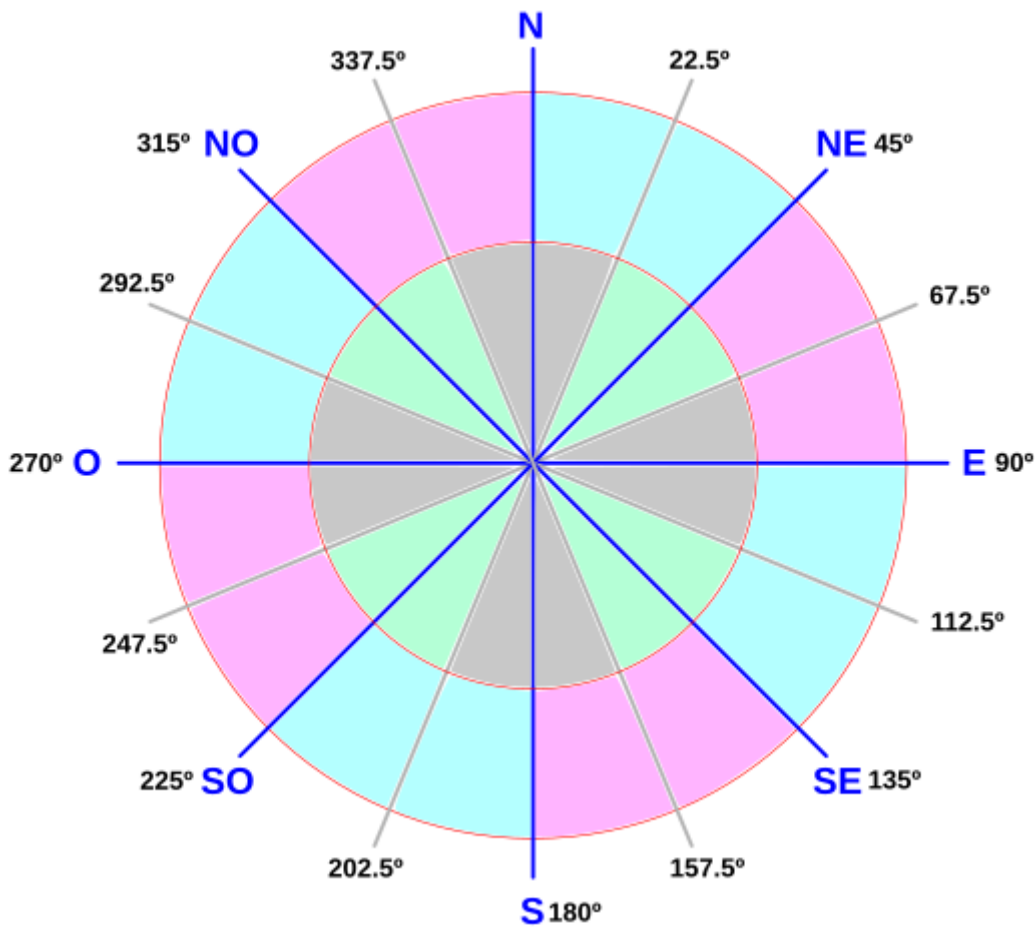
compass.calibrate()
while True:
    angulo = compass.heading()
    if angulo < 45:
        display.show("N")
    elif angulo < 135:
        display.show("E")
    elif angulo < 225:
        display.show("S")
    elif angulo < 315:
        display.show("O")
```

else:

display.show("N")

## Brújula a 45º

Lo que vamos a hacer ahora es localizar las siguientes ocho direcciones, separadas entre si un ángulo de 45º: noroeste (NE), oeste (O), suroeste (SO), sur (S), sureste (SE), este (E), noreste (NE), norte (N). El margen o ángulo que va a englobar cada dirección estará ajustado dentro de otro ángulo de 45º dividido en dos iguales de 22.5º respecto a la dirección principal. Un gráfico nos aclara mejor la idea



En el editor Mu también son válidas las definiciones predefinidas para las flechas.

El programa es:

```
from microbit import *
```

```
compass.calibrate()
```

```
while True:
    angulo = compass.heading()
    if angulo > 22.5 and angulo <= 67.5:
        display.show(Image.ARROW_NE)
    elif angulo > 67.5 and angulo <= 112.5:
        display.show(Image.ARROW_E)
    elif angulo > 112.5 and angulo <= 157.5:
        display.show(Image.ARROW_SE)
    elif angulo > 157.5 and angulo <= 202.5:
        display.show(Image.ARROW_S)
    elif angulo > 202.5 and angulo <= 247.5:
        display.show(Image.ARROW_SW)
    elif angulo > 247.5 and angulo <= 292.5:
        display.show(Image.ARROW_W)
    elif angulo > 292.5 and angulo <= 337.5:
        display.show(Image.ARROW_NW)
    elif angulo > 337.5 and angulo <= 22.5:
        display.show(Image.ARROW_N)
```

## Brújula indicando el norte

En este se intenta simplemente visualizar una aguja que apunte al norte

```
from microbit import *

# Antes de nada calibrar
compass.calibrate()

# Mantener la aguja apuntando aproximadamente en la dirección correcta.
while True:
    sleep(100)
    aguja = ((15 - compass.heading()) // 30) % 12
    display.show(Image.ALL_CLOCKS[aguja])
```

# Acelerómetro

Página extraída de Federico Coca [Guia de Trabajo de Microbit](#) CC-BY-SA

Este objeto permite acceder al acelerómetro de la placa.

Por defecto MicroPython establece el rango del acelerómetro en  $\pm 2000$  mg (siendo g una unidad de aceleración basada en la gravedad estándar), que configura los valores máximo y mínimo devueltos por las funciones del acelerómetro. El rango puede cambiarse mediante `microbit.accelerometer.set_range()`.

El acelerómetro también proporciona funciones de conveniencia para detectar gestos. Los gestos reconocidos se representan como cadenas: arriba (up), abajo (down), izquierda (left), derecha (right), boca arriba (face up), boca abajo (face down), caída libre (freefall), 3g, 6g, 8g, sacudida (shake).

**Nota:** Los gestos no se actualizan en segundo plano por lo que es necesario realizar llamadas constantes a algún método del acelerómetro para realizar la detección de gestos. Normalmente los gestos pueden ser detectados usando un bucle con un pequeño retardo `microbit.sleep()`.

Sus funciones son:

- `microbit.accelerometer.get_x()`
- `microbit.accelerometer.get_y()`
- `microbit.accelerometer.get_z()`

Retorna como un entero positivo o negativo la aceleración medida en el eje correspondiente en mili-g.

- `microbit.accelerometer.get_values()`

Devuelve las medidas de aceleración en todos los ejes a la vez, como una tupla de tres elementos de enteros ordenados como X, Y, Z.

- `microbit.accelerometer.get_strength()`

Obtiene la medida de la aceleración de todos los ejes combinados, como un entero positivo. Es la suma pitagórica de los ejes X, Y y Z. Devuelve la fuerza de aceleración



combinada de todos los ejes, en mili-g.

- `microbit.accelerometer.current_gesture()`

Devuelve una cadena con el nombre del gesto actual.

- `microbit.accelerometer.is_gesture(name)`

El parámetro `name` es una cadena con el nombre del gesto a comprobar. Devuelve un valor booleano que indica si el gesto nombrado está activo actualmente.

- `microbit.accelerometer.was_gesture(name)`

El parámetro `name` es una cadena con el nombre del gesto a comprobar. Devuelve un valor booleano que indica si el gesto nombrado ha estado activo desde la última vez.

- `microbit.accelerometer.get_gestures()`

Se usa para obtener una lista histórica de los gestos registrados. Al llamar a esta función se borra el histórico de gestos antes de devolver el valor. Devuelve una tupla del historial de gestos, el más reciente aparece en último lugar.

- `microbit.accelerometer.set_range(value)`

Ajusta el rango de sensibilidad del acelerómetro, en g (gravedad estándar), a los valores más cercanos soportados por el hardware, de forma que redondee a 2, 4 u 8 g. El parámetro `value` establece el nuevo rango para el acelerómetro, un entero en g.

A continuación vamos a ver los ejemplos que aparecen en la documentación oficial traducidos.

## Trazador gráfico de aceleraciones

El programa lo vamos a hacer en el editor Mu para aprovechar su trazador gráfico.

```
from microbit import *

while True:
    uart.write(str(accelerometer.get_values()) + "\r\n")
    sleep(1000)
```

Ponemos en marcha el trazador, y la comunicación serie REPL después de flashear el programa. Hacemos reset de la micro:bit y podemos ver la evolución de las lecturas con los movimientos de la placa.

A11\_brujula\_MC.png

## Gradiometro

En este programa se ve como si la luz pesara:



<https://www.youtube.com/embed/JR8h-O11QbY>

El programa es:

```
from microbit import *
# El brillo estara entre 0 y 9
brillo = 9
# La funcion mapear ajusta los valores
# leidos y los lleva al rango 0-4.
def mapear(valor):
    if valor < -500:
        valor=-500
    elif valor > 500:
        valor=500
    valor=(valor+500)/250
    return int(valor)

while True:
    # Lee la aceleración en x e y con un
    #rango que va de -2000 a 2000.
    roll_x = accelerometer.get_x()
    pitch_y = accelerometer.get_y()
    # No necesitamos un rango tan amplio por
    # eso lo bajamos de -500 a 500
    x=mapear(roll_x)
    y=mapear(pitch_y)
    display.clear()
    display.set_pixel(x, y, brillo)
    sleep(500)
```

## Bola mágica

Una bola 8 mágica que adivina el futuro. Haz una pregunta y agita el dispositivo para obtener una respuesta.

```
# Magic 8 ball by Nicholas Tollervey. February 2016.  
# Bola mágica 8 por Nicholas Tollervey. Febrero 2016.  
# Ask a question then shake.  
# Haz una pregunta y agita la micro:bit  
# This program has been placed into the public domain.  
# Este programa es de dominio público.
```

```
from microbit import *  
import random
```

```
respuestas = [  
    "Es cierto",  
    "Es decididamente así",  
    "Sin duda alguna",  
    "Sí, definitivamente.",  
    "Puedes confiar en ello",  
    "Como yo lo veo, sí",  
    "Lo más probable",  
    "Buenas perspectivas",  
    "Sí",  
    "Los indicios apuntan a que sí",  
    "Respuesta dudosa inténtalo de nuevo",  
    "Vuelve a preguntar más tarde",  
    "Mejor no te lo digo ahora",  
    "No se puede predecir ahora",  
    "Concéntrate y vuelve a preguntar",  
    "No cuentes con ello",  
    "Mi respuesta es no",  
    "Mis fuentes dicen que no",  
    "Perspectivas no tan buenas",  
    "Muy dudoso",  
]
```

```
while True:
```

```
display.show('8')  
if accelerometer.was_gesture('shake'):  
    display.clear()  
    sleep(1000)  
    display.scroll(random.choice(respuestas))  
sleep(10)
```

## Juego evita obstáculos

Un juego consistente en evitar obstáculos moviendo la micro:bit.

```
# Simple Slalom by Larry Hastings, September 2015  
# Eslalon simple de Larry Hastings, septiembre de 2015.  
# This program has been placed into the public domain.  
# Este programa es de dominio público.  
import microbit as m  
import random  
  
p = m.display.show  
min_x = -1024  
max_x = 1024  
range_x = max_x - min_x  
wall_min_speed = 400  
player_min_speed = 200  
wall_max_speed = 100  
player_max_speed = 50  
speed_max = 12  
  
while True:  
  
    i = m.Image('00000:~*5)  
    s = i.set_pixel  
    player_x = 2  
    wall_y = -1  
    hole = 0  
    score = 0  
    handled_this_wall = False
```

```

wall_speed = wall_min_speed
player_speed = player_min_speed
wall_next = 0
player_next = 0

while True:
    t = m.running_time()
    player_update = t >= player_next
    wall_update = t >= wall_next
    if not (player_update or wall_update):
        next_event = min(wall_next, player_next)
        delta = next_event - t
        m.sleep(delta)
        continue

    if wall_update:
        # calculate new speeds
        speed = min(score, speed_max)
        wall_speed = wall_min_speed + int((wall_max_speed - wall_min_speed) * speed / speed_max)
        player_speed = player_min_speed + int((player_max_speed - player_min_speed) * speed / speed_max)
        wall_next = t + wall_speed
        if wall_y < 5:
            # erase old wall
            use_wall_y = max(wall_y, 0)
            for wall_x in range(5):
                if wall_x != hole:
                    s(wall_x, use_wall_y, 0)
    wall_reached_player = (wall_y == 4)
    if player_update:
        player_next = t + player_speed
        # find new x coord
        x = m.accelerometer.get_x()
        x = min(max(min_x, x), max_x)
        # print("x accel", x)
        s(player_x, 4, 0) # turn off old pixel
        x = ((x - min_x) / range_x) * 5

```

```

x = min(max(0, x), 4)
x = int(x + 0.5)
# print("have", position, "want", x)
if not handled_this_wall:
    if player_x < x:
        player_x += 1
    elif player_x > x:
        player_x -= 1
# print("new", position)
# print()
if wall_update:
    # update wall position
    wall_y += 1
    if wall_y == 7:
        wall_y = -1
        hole = random.randrange(5)
        handled_this_wall = False
    if wall_y < 5:
        # draw new wall
        use_wall_y = max(wall_y, 0)
        for wall_x in range(5):
            if wall_x != hole:
                s(wall_x, use_wall_y, 6)
if wall_reached_player and not handled_this_wall:
    handled_this_wall = True
    if (player_x != hole):
        # collision! game over!
        break
    score += 1
if player_update:
    s(player_x, 4, 9) # turn on new pixel
p(i)
p(i.SAD)
m.sleep(1000)
m.display.scroll("Score:" + str(score))
while True:

```



```
if (m.button_a.is_pressed() and m.button_a.is_pressed()):  
    break  
m.sleep(100)
```

<https://www.youtube.com/embed/a0f75gyTm5U>

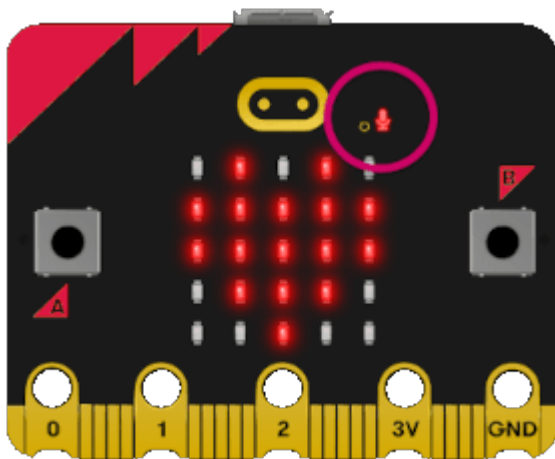
Página extraída de Federico Coca [Guia de Trabajo de Microbit](#) CC-BY-SA

# Micrófono

Página extraída de Federico Coca [Guía de Trabajo de Microbit](#) CC-BY-SA

ATENCIÓN SÓLO VÁLIDO PARA PLACAS V2

Este objeto permite acceder al micrófono integrado disponible en micro:bit V2. Se puede utilizar para responder al sonido. La entrada del micrófono se encuentra en la parte frontal de la placa junto a un LED de actividad del micrófono, que se ilumina cuando el micrófono está en uso.



Autor [Federico Coca](#) Fuente : [Guía de Trabajo de Microbit](#) Licencia [CC-BY-SA](#)

El micrófono puede responder a un conjunto predefinido de **eventos sonoros** que se basan en la amplitud y la longitud de onda del sonido. Están representados por instancias de la clase

`SoundEvent`, accesibles a través de variables en `microbit.SoundEvent`:

- `microbit.SoundEvent.QUIET`: Representa la transición de eventos de sonido, de fuerte (`loud`) a silencioso (`quiet`) como hablar tranquilo o música de fondo a bajo volumen.
- `microbit.SoundEvent.LOUD`: Representa la transición de eventos de sonido, de silencioso (`quiet`) a fuerte (`loud`) como aplausos o hablar a gritos.

Las funciones disponibles son:

- `microbit.microphone.current_event()`: Retorna el nombre del último evento sonoro grabado, `SoundEvent('loud')` o `SoundEvent('quiet')`.





- `microbit.microphone.was_event(event)`: donde `event` es un evento sonoro como `SoundEvent.LOUD` o `SoundEvent.QUIET`. Retorna `true` si el sonido se ha escuchado al menos una vez desde la última llamada, en caso contrario `false`. `was_event()` también borra el historial de eventos de sonido antes de retornar.
- `microbit.microphone.is_event(event)`: donde `event` es un evento sonoro como `SoundEvent.LOUD` o `SoundEvent.QUIET`. Retorna `true` si el evento sonoro es el más reciente desde la última llamada, en caso contrario `false`. No borra el historial de eventos de sonido.
- `microbit.microphone.get_events()`: Retorna una tupla del historial de eventos. El más reciente aparece en último lugar. `get_events()` también borra el historial de eventos de sonido antes de retornar.
- `microbit.microphone.set_threshold(event, value)`: donde `event` es un evento sonoro como `SoundEvent.LOUD` o `SoundEvent.QUIET`. `value` es el umbral en el rango 0-255. Por ejemplo `set_threshold(SoundEvent.LOUD, 250)` sólo se activará si el sonido es muy alto ( $\geq 250$ ).
- `microbit.microphone.sound_level()`: Retorna una representación del nivel de presión sonora en el intervalo de 0 a 255.

## Sonómetro

No estaría nada mal poner esto en clase, comedores...

```
from microbit import *

# definicion funcion mapea para cambiar un rango de valores a otro
def mapea(valor, deMin, deMax, aMin, aMax):
    deRango = deMax - deMin
    aRango = aMax - aMin
    valorEsc_de = float(valor - deMin)/float(deRango)
    valorEsc_a = aMin + (valorEsc_de * aRango)
    return valorEsc_a

# Creamos las imagenes para el grafico de barras
grafico5 = Image("99999:"
    "99999:"
    "99999:"
    "99999:")

grafico4 = Image("00000:"
    "99999:")
```



```
"99999:"
```

```
"99999:"
```

```
"99999")
```

```
grafico3 = Image("00000:"
```

```
"00000:"
```

```
"99999:"
```

```
"99999:"
```

```
"99999")
```

```
grafico2 = Image("00000:"
```

```
"00000:"
```

```
"00000:"
```

```
"99999:"
```

```
"99999")
```

```
grafico1 = Image("00000:"
```

```
"00000:"
```

```
"00000:"
```

```
"00000:"
```

```
"99999")
```

```
grafico0 = Image("00000:"
```

```
"00000:"
```

```
"00000:"
```

```
"00000:"
```

```
"00000")
```

```
graficos = [grafico0, grafico1, grafico2, grafico3, grafico4, grafico5]
```

```
# ignora el primer nivel de sonido leído
```

```
nivelSonido = microphone.sound_level()
```

```
sleep(200)
```

```
# establece un umbral para el nivel de sonido
```

```
umbral = microphone.set_threshold(SoundEvent.LOUD, 125)
```

```
while True:
```

```
# si el umbral es superado se muestra una carita triste
if microphone.sound_level() >= 125:
    display.show(Image.SAD)
    sleep(1000)
else:
    # mapear nivel de sonido de 0-255 a 0-5 para escoger gráfico
    nivelSonido = int(mapea(microphone.sound_level(), 0, 255, 0, 5))
    display.show(graficos[nivelSonido])
```

<https://www.youtube.com/embed/IGcbk1rR8FQ>

## Uso de las API

Un ejemplo que utiliza algunas de las funciones de la API del micrófono es:

```
'''Prueba básica del micrófono.
Boton A: actualizar pantalla cuando se escucha un sonido alto o bajo.
Botón B: actualizar la pantalla cuando se escucho un sonido alto o bajo.
Al agitarla: se muestran los últimos sonidos escuchados, para intentar esta prueba
se hace un sonido fuerte y uno silencioso antes de agitar.'''

from microbit import *

display.clear()
sound = microphone.current_event()

while True:
    if button_a.is_pressed():
        if microphone.current_event() == SoundEvent.LOUD:
            display.show(Image.SQUARE)
            uart.write('Es Fuerte\n')
        elif microphone.current_event() == SoundEvent.QUiet:
            display.show(Image.SQUARE_SMALL)
            uart.write('Es Silencio\n')
        sleep(500)
```

```
display.clear()
if button_b.is_pressed():
    if microphone.was_event(SoundEvent.LOUD):
        display.show(Image.SQUARE)
        uart.write('Fue Fuerte\n')
    elif microphone.was_event(SoundEvent.QUIET):
        display.show(Image.SQUARE_SMALL)
        uart.write('Fue silencioso\n')
    else:
        display.clear()
        sleep(500)
display.clear()
if accelerometer.was_gesture('shake'):
    sounds = microphone.get_events()
    soundLevel = microphone.sound_level()
    print(soundLevel)
    for sound in sounds:
        if sound == SoundEvent.LOUD:
            display.show(Image.SQUARE)
        elif sound == SoundEvent.QUIET:
            display.show(Image.SQUARE_SMALL)
        else:
            display.clear()
            print(sound)
            sleep(500)
```

En la consola serie vemos algunos resultados:

Ejemplo\_funciones\_microfono

```

37 for sound in sounds:
38     if sound == SoundEvent.LOUD:
39         display.show(Image.SQUARE)
40     elif sound == SoundEvent.QUIET:
41         display.show(Image.SQUARE_SMALL)
42     else:
43         display.clear()
44         print(sound)
45         sleep(500)

```

micro:bit ready to flash ⚡ Hide serial ⓘ ⋮

```

Es Silencio
    Es Silencio
        Es Silencio
            Es Silencio
                Es Silencio
                    Fue Fuerte
                        Fu
e silencioso
0
SoundEvent('loud')
SoundEvent('quiet')
SoundEvent('loud')
SoundEvent('quiet')
SoundEvent('loud')
SoundEvent('quiet')
SoundEvent('loud')

```

Autor Federico Coca Fuente : Guía de Trabajo de Microbit Licencia CC-BY-SA

Se ve mejor con un vídeo, pero con el simulador que nos muestra la cantidad de sonido:

- El botón A muestra el sonido presente
- El botón B muestra los sonidos pasados
- Sacudir la microbit nos muestra en mensaje y numero el sonido presente

<https://www.youtube.com/embed/EWnYdzFa1nM>



# Radio

Página extraída de Federico Coca [Guia de Trabajo de Microbit CC-BY-SA](#)

El módulo de `radio` permite que los dispositivos trabajen juntos a través de redes inalámbricas sencillas.

El módulo de radio es conceptualmente muy sencillo:

- Los mensajes broadcast o de difusión tienen una longitud configurable (hasta 251 bytes).
- Los mensajes recibidos se leen de una cola de tamaño configurable (cuanto mayor sea la cola, más memoria RAM se utilizará). Si la cola está llena, se ignoran los mensajes nuevos. La lectura de un mensaje lo elimina de la cola.
- Los mensajes se emiten y reciben en un canal preseleccionado (numerado de 0 a 83).
- Las emisiones tienen un determinado nivel de potencia: más potencia significa más alcance.
- Los mensajes se filtran por dirección (como un número de casa) y grupo (como un destinatario con nombre en la dirección especificada)
- La velocidad de transmisión puede ser una de las tres predeterminadas.
- Se envían y reciben bytes para trabajar con datos arbitrarios.
- Utilizando `receive_full` se obtiene todos los detalles sobre un mensaje entrante: los datos como tales, la intensidad de la señal de recepción y una marca de tiempo en microsegundos cuando llegó el mensaje.
- Es fácil enviar y recibir mensajes como cadenas.

Para acceder a este módulo se necesita:

- `import radio`

Las constantes son:

- `radio.RATE_1MBIT`. Es una constante utilizada para indicar un caudal de 1 Mbit por segundo.
- `radio.RATE_2MBIT`. Es una constante utilizada para indicar un caudal de 2 Mbit por segundo.

Las funciones disponibles son:



- `radio.on()`. Enciende el módulo de radio. Desde MicroPython-on-micro:bit v1.1 la radio se activa por defecto cuando se importa el módulo de radio. En versiones anteriores, para reducir el consumo de energía, esta función tenía que ser llamada explícitamente. Para esos casos `radio.off()` puede ser llamada después de la importación.
- `radio.off()`. Apaga la radio. Esto ahorra energía y memoria.
- `radio.config(**kwargs)`. Configura varios ajustes basados en palabras clave relacionados con la radio. A continuación se enumeran los ajustes disponibles y sus valores predeterminados.
- `length`. La longitud (por defecto=32) define la longitud máxima, en bytes, de un mensaje enviado por radio. Puede tener una longitud máxima de 251 bytes (254 - 3 bytes para los preámbulos S0, LENGTH y S1).
- `queue`. La cola (por defecto=3) especifica el número de mensajes que se pueden almacenar en la cola de mensajes entrantes. Si no hay espacio en la cola para mensajes entrantes, el mensaje entrante se descarta.
- `channel`. El canal (por defecto=7) puede ser un valor entero de 0 a 83 (inclusive) que define un "canal" arbitrario al que se sintoniza la radio. Los mensajes se enviarán a través de este canal y sólo los mensajes recibidos a través de este canal se pondrán en la cola de mensajes entrantes. Cada paso tiene un ancho de 1MHz, basado en 2400MHz.
- `power`. La potencia (por defecto=6) es un valor entero de 0 a 7 (ambos inclusive) que indica la intensidad de la señal utilizada al emitir un mensaje. Cuanto mayor sea el valor, más potente será la señal, pero más potencia consumirá el dispositivo. La numeración se traduce en posiciones en la siguiente lista de valores dBm (decibelios milivatio): -30, -20, -16, -12, -8, -4, 0, 4.
- `address`. La dirección (por defecto=0x75626974) es un nombre arbitrario, expresado como una dirección de 32 bits, que se utiliza para filtrar los paquetes entrantes a nivel de hardware, manteniendo sólo aquellos que coinciden con la dirección que establezca. El valor por defecto utilizado por otras plataformas relacionadas con micro:bit es el valor por defecto utilizado aquí.
- `group`. El grupo (por defecto=0) es un valor de 8 bits (0-255) que se utiliza con la dirección al filtrar los mensajes. Conceptualmente, "dirección" es como una dirección de casa/oficina y "grupo" es como la persona de esa dirección a la que se quiere enviar el mensaje.
- `data_rate`. La tasa\_de\_datos (por defecto=radio.RATE\_1MBIT) indica la velocidad a la que se produce el flujo de datos. Puede ser uno de los siguientes constantes definidos en el módulo de radio : `RATE_1MBIT` o `RATE_2MBIT`.

**Nota** Una velocidad de datos menor de 250 kbit/seg es compatible con micro:bit V1, y puede ser posible con micro:bit V2, pero no se garantiza que funcione en todos los dispositivos. Para acceder a esta característica oculta para la compatibilidad con V1 ponemos 2 en el argumento `data_rate`.



Si **no se llama** a `config` **se asumen** los valores por defecto descritos anteriormente.

- `radio.reset()`. Restablece los valores por defecto (como se indica en la documentación de la función de configuración). Ninguno de los siguientes métodos de envío o recepción funcionará hasta que la radio esté encendida.
- `radio.send_bytes(message)`. Envía `message` conteniendo bytes.
- `radio.receive_bytes()`. Recibe el siguiente mensaje entrante en la cola de mensajes. Devuelve `None` (Ninguno) si no hay mensajes pendientes. Los mensajes se devuelven como bytes.
- `radio.receive_bytes_into(buffer)`. Recibe el siguiente mensaje entrante en la cola de mensajes. Copia el mensaje en el búfer, recortando el final del mensaje si es necesario. Devuelve `None` si no hay mensajes pendientes; en caso contrario, devuelve la longitud del mensaje (que puede ser superior a la longitud del búfer).
- `radio.send(message)`. Envía una cadena de mensajes. Esto es el equivalente de `send_bytes(bytes(message, 'utf8'))` pero con `b'\x01\x00\x01'` antepuesto (para hacerlo compatible con otras plataformas que apuntan al micro:bit).
- `radio.receive()`. Funciona exactamente igual que `receive_bytes` pero devuelve lo que se haya enviado. Es equivalente a `str(receive_bytes(), 'utf8')` pero con una comprobación de que los tres primeros bytes son `b'\x01\x00\x01'` (para hacerlo compatible con otras plataformas que puedan tener como objetivo el micro:bit). Elimina los bytes añadidos antes de convertir a cadena. Se lanza una excepción `ValueError` si falla la conversión a cadena.
- `radio.receive_full()`. Devuelve una tupla que contiene tres valores que representan el siguiente mensaje entrante en la cola de mensajes. Si no hay mensajes pendientes se devuelve `None`.

Los tres valores de la tupla representan:

- el siguiente mensaje entrante en la cola de mensajes en bytes.
- el RSSI (intensidad de la señal): un valor entre 0 (más fuerte) y -255 (más débil) medido en dBm.
- una marca de tiempo en microsegundos: el valor devuelto por `time.ticks_us()` cuando se recibió el mensaje.

## Envío de la temperatura

Vamos a enviar la temperatura medida por la placa y en el valor recibido vamos a calcular la diferencia entre la temperatura de la placa que recibe y la que envía, estableciendo así la diferencia de temperaturas entre, por ejemplo, una zona al sol y otra a la sombra.

El código es el siguiente

El código del programa es el siguiente:

```
from microbit import *
import radio
radio.on()
radio.config(channel=50, group=90)

while True:
    if button_a.is_pressed():
        radio.send(str(temperature()))
    recibido = radio.receive()
    if recibido is not None:
        display.show(recibido)
        sleep(50)
    display.clear()
```

El resultado (se visualiza en Makecode que permite una simulación en dos placas que <https://python.microbit.org/> no permite)

A11\_brujula\_MC.png

Autor Federico Coca Fuente : Guía de Trabajo de Microbit Licencia CC-BY-SA

## Enviar "Agitar" cuando se agita

vamos a realizar una especie de juego en el que se envía el mensaje "agitar (shake)" a un grupo de micro:bits (podemos poner otras en un grupo diferente) cuando se realiza justamente ese gesto. Usaremos uno de los botones para salir del programa.

El código del programa es el siguiente:

```
from microbit import *
import radio
radio.on()
radio.config(channel=50, group=90)

while True:
    if accelerometer.is_gesture('shake'):
```



```
radio.send("Agitar")  
recibido = radio.receive()  
if recibido is not None:  
    display.show(recibido)  
    sleep(50)  
display.clear()
```

El resultado (se visualiza en Makecode que permite una simulación en dos placas que <https://python.microbit.org/> no permite)

A11\_brujula\_MC.png

Autor Federico Coca Fuente : Guía de Trabajo de Microbit Licencia CC-BY-SA

Página extraída de Federico Coca Guia de Trabajo de Microbit CC-BY-SA