

# Hackeando

- [Pines de Entrada/salida](#)
- [Input output](#)
- [Música predefinida o crea tu música](#)
- [Musica](#)
- [Putty](#)
- [UART](#)
- [Registro de datos](#)

# Pines de Entrada/salida

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

En MicroPython, cada pin en la BBC micro:bit está representado por un objeto llamado pinN, donde N es el número del pin.

Por ejemplo, para usar el pin etiquetado 0 (cero), puedes usar el objeto llamado pin0 en tu script. El pin del logo V2 utiliza pin\_logo.

Estos objetos tienen varios métodos asociados dependiendo de lo que el pin específico es capaz de hacer, por ejemplo, leer, escribir o tocar.

Quizá lo mas sencillo que podemos hacer es comprobar que los pines 0, 1 y 2 del borde de placa son táctiles. Haremos como ejemplo que al tocar cualquiera de ellos la micro:bit sonría y si no se toca ninguno que esté triste. Le hacemos cosquillas a la micro:bit. El programa es:

```
from microbit import *  
"""  
pin0.set_touch_mode(pin0.CAPACITIVE)  
pin1.set_touch_mode(pin0.CAPACITIVE)  
pin2.set_touch_mode(pin0.CAPACITIVE)  
"""  
while True:  
    if (pin0.is_touched() or pin1.is_touched() or pin2.is_touched()):  
        display.show(Image.HAPPY)  
    else:  
        display.show(Image.SAD)
```

En la animación vemos el funcionamiento del programa.

[pinout.png](#)

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

Si descargamos firmware en una placa para probar el programa debemos saber que no basta con tocar alguno de los pines con una mano, hay que tocarlo simultaneamente con la otra mano en

GND para cerrar el circuito eléctrico.

En la última versión de micro:bit V2 es posible cambiar el comportamiento predeterminado de la patilla, de modo que no sea necesario tocar GND. En los programas siguientes el código que hace esto está comentado por lo que si queremos probarlo debemos eliminar esos comentarios. Recordemos que por defecto los pines del conector de borde son sensores táctiles resistivos mientras que el pin logo V2 es capacitivo.

## Pines digitales

Podemos utilizar los pines 0, 1 y 2 del borde de placa en modo digital tanto para leer su valor como para escribir o establecer su valor. Esto se representa con un "1" lógico (sin las comillas) si están activados o los queremos activar y un "0" lógico si están desactivados o los queremos desactivar.

Si queremos escribir en ellos los pines estarán actuando como salidas y tenemos que invocar al método `write` para hacerlo. Las sentencias, para un pin genérico "N" son:

```
pinN.write_digital(1) #Salida en estado alto  
pinN.write_digital(0) #Salida en estado bajo
```

También podemos conectar, por ejemplo un interruptor o botón pulsador al pin (veremos como hacerlo en la siguiente actividad) y comprobar si el interruptor está abierto (0) o cerrado (1). En este caso los pines estarán configurados como entradas y la lectura de su estado se obtiene invocando el método `read`. Las sentencias, para un pin genérico "N" son:

```
pinN.read_digital() #Devuelve el estado 0 o 1 del pin N
```

***Nunca se conecta nada a los pines con un voltaje superior a 3v porque se puede dañar la micro:bit.***

## Pines analógicos

Podemos utilizar los pines 0, 1 y 2 del borde de placa en modo analógico tanto para leer su valor como para escribir o establecer su valor. Esto significa que en lugar de estar activos o inactivos (0 o 1), varían su valor entre 0 y 1023.

Si queremos escribir en ellos los pines estarán actuando como salidas y tenemos que invocar al método `write` para hacerlo. La sentencia, para un pin genérico "N" es:

```
pinN.write_analog(valor) #valor puede estar entre 0 y 1023
```



Si conectamos sensores o actuadores analógicos a los pines podemos leer su valor invocando a `read`. La sentencia, para un pin genérico "N" es:

```
pinN.read_analog(valor) #valor puede estar entre 0 y 1023
```

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

# Input output

Una manera rápida de probar las entradas y salidas de microbit es utilizar el código predefinido que hay en reference

Si entramos en simulación, al pulsar sobre el PIN0 se visualiza 0 en el display

The screenshot shows the micro:bit IDE interface. On the left, the 'Reference' panel is open, displaying the 'Pins' section (1) and the 'Touch pins' section (2). The 'Touch pins' section shows a code snippet (3) for pin 0. The main editor area (4) shows the same code snippet. On the right, the 'Simulation' panel is open, showing a virtual micro:bit board (5) and a 'Show serial' panel (6) with a 'shake' button. Red arrows indicate the flow from the reference code to the simulation and the serial panel.

Podemos ahora usar bloques lógicos para tener otras posibilidades:

```
# Imports go at the top
from microbit import *

while True:
    if pin0.is_touched():
        display.show(Image.HEART)
    else:
        display.show(Image.NO)
```



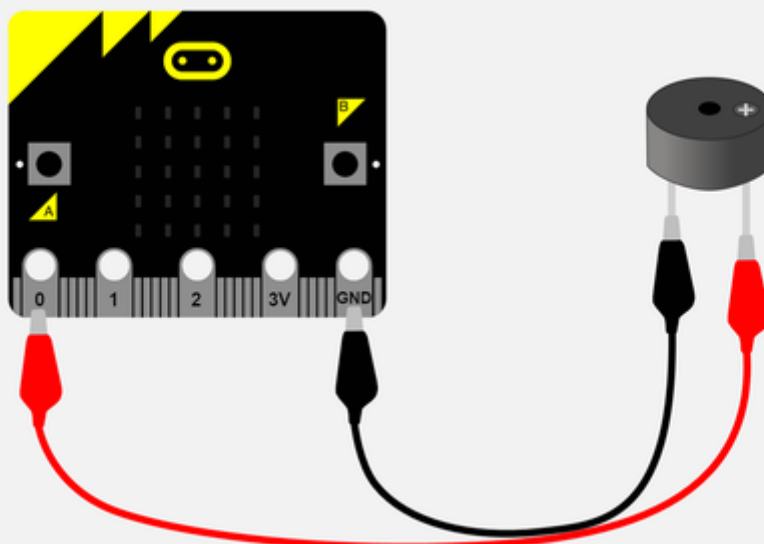
<https://www.youtube.com/embed/ul2p9HazV1Y>

# Música predefinida o crea tu música

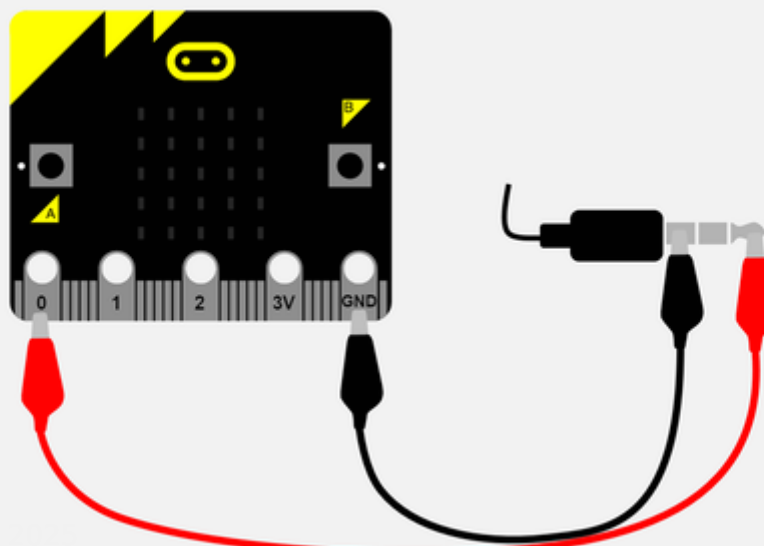
## SALIDAS DE AUDIO

La placa Microbit v2 tiene un altavoz incorporado que se puede anular o activar con la instrucción **speaker.on()** o **speaker.off()**

La salida de audio también sale por el **pin0**, de tal manera que si conectamos un altavoz o headphone, y tenemos **speaker.on()** se oirá por los sitios, si lo tienes en off sólo por el buzzer/headphones exterior:



Piezo Buzzer  
Teachwithict.com



Headphones  
Teachwithict.com

Fuente <https://www.teachwithict.com/microbit-music-python.html>

BBC micro:bit "Magic 8-Ball" lesson is licenced under a [Creative Commons Attribution 4.0 International License](#).

Ojo, hay que conectar un **buzzer pasivo**, es decir, que reproduce la seña analógica en sonido (o sea un altavoz normal y corriente)

si quieres conectar un **buzzer activo**, reproduce un tono (predeterminado) al suministrarle un 1, el siguiente código sonaría una alarma:

```
from microbit import *  
while True:  
    pin0.write_digital(1)  
    sleep(500)  
    pin0.write_digital(0)  
    sleep(500)
```

En los siguientes ejemplo usaremos siempre **buzzer pasivo**. Si no te queda claro lo que es un buzzer activo y un pasivo, mira [esta página](#)

## CREA TUS EFECTOS

Puedes crear los efectos utilizando rangos de frecuencias, aquí en el ejemplo la función pitch reproduce durante 6mseg las frecuencias de medio 880 , aguda 1760 y grave 16 y luego lo mismo pero en orden decreciente, y así sucesivamente para dar el efecto de sirena.

```
import music  
from microbit import *  
display.show(Image.GHOST)  
while True:  
    for freq in range(880, 1760, 16):  
        music.pitch(freq, 6)  
    for freq in range(1760, 880, -16):  
        music.pitch(freq, 6)
```

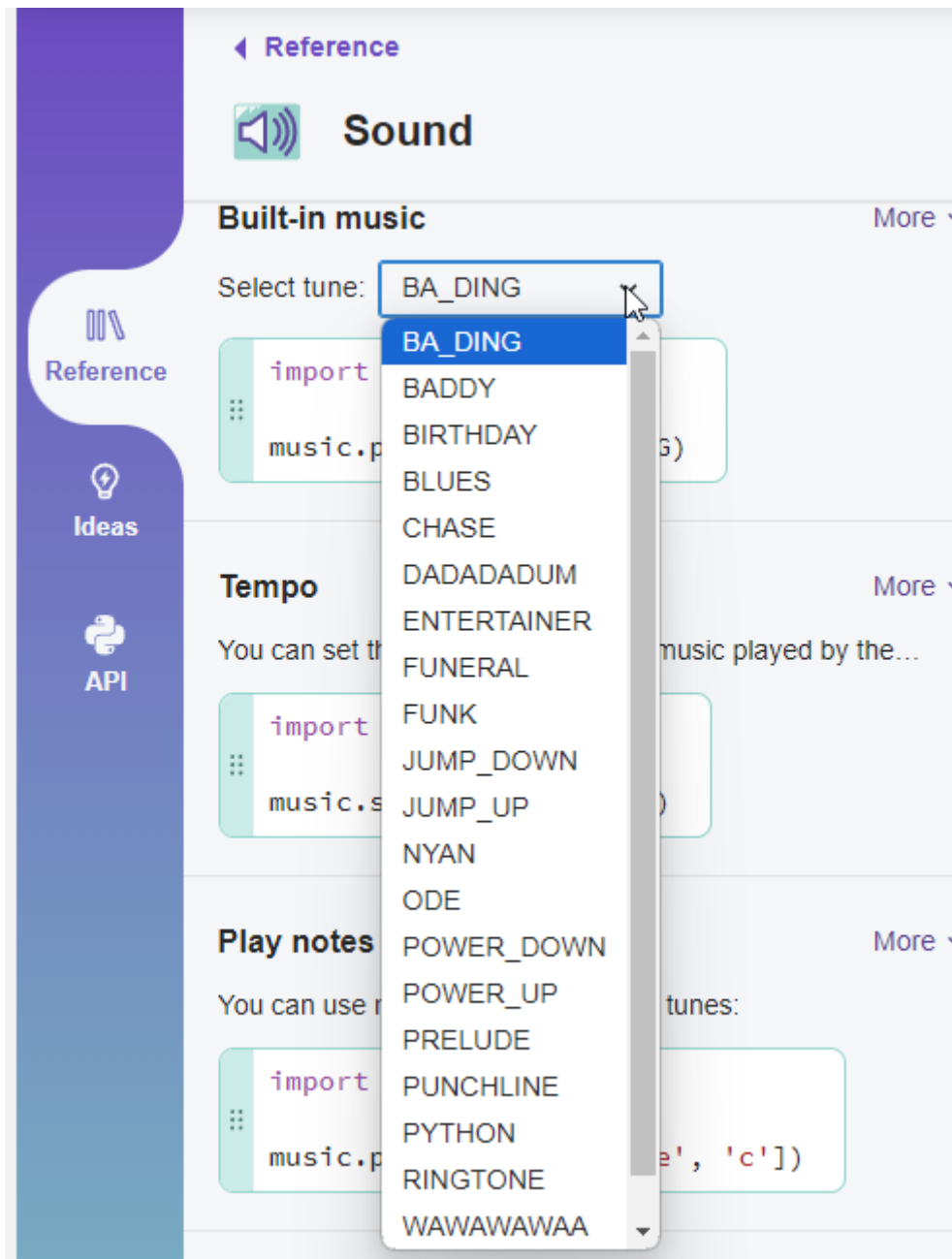


Extraído de <https://microbit-micropython.readthedocs.io/en/v2-docs/tutorials/music.html#sound-effects>

<https://www.youtube.com/embed/tSkmgffox2A>

## MUSICA PREDEFINIDA

En Reference- Sound tienes muchos tonos predefinidos para experimentar:



Si tienes la microbit v2 tienes otros en "**Expressive sounds**" como `audio.play(Sound.TWINKLE)`

También puede hablar, arrastra el código que tienes en Reference-Sound- Speech, pero no lo intentes en español, no se entiende nada

```
import speech
from microbit import *
display.show(Image.FABULOUS)
speaker.on()
set_volume(255)
```

```
speech.say('Hello, How are you? Do you sign up in online course in CATEDU.ES ?')
audio.play(Sound.TWINKLE)
```

<https://www.youtube.com/embed/EcV3aeYo6Vs>

## CONSTRUYENDO TU MÚSICA

Tienes que añadir la librería **music** y componer las notas según la notación americana :



Fuente <https://www.teachwithict.com/microbit-music-python.html>

*BBC micro:bit "Magic 8-Ball" lesson is licenced under a [Creative Commons Attribution 4.0 International License](#).*

La duración (si no se pone, sigue con la duración anterior)



Notes	Name	Duration
	<b>Semibreve</b> Whole note	16
	<b>Minim</b> Half note	8
	<b>Crotchet</b> Quarter note	4
	<b>Quaver</b> Eighth note	2
	<b>Semiquaver</b> Sixteenth note	1

Fuente <https://www.teachwithict.com/microbit-music-python.html>

BBC micro:bit "Magic 8-Ball" lesson is licenced under a [Creative Commons Attribution 4.0 International License](#).

Se puede poner incluso sostenidos, por ejemplo C#4:4 o f#5:4

Si quieres poner descansos es con la letra r seguido de su duración por ejemplo r:4 r:2

## Un ejemplo

Tono Nokia (arriba la duración)



El código sería:

```
from microbit import *
import music

tune = ["e5:2", "d5", "f#4:4", "g#4", "c#5:2", "b4", "d4:4", "e4", "b4:2", "a4", "c#4:4", "e4", "a4:12"]

music.play(tune)
```

Otro ejemplo, el código lo tienes en <https://microbit-micropython.readthedocs.io/en/v2-docs/music.html>

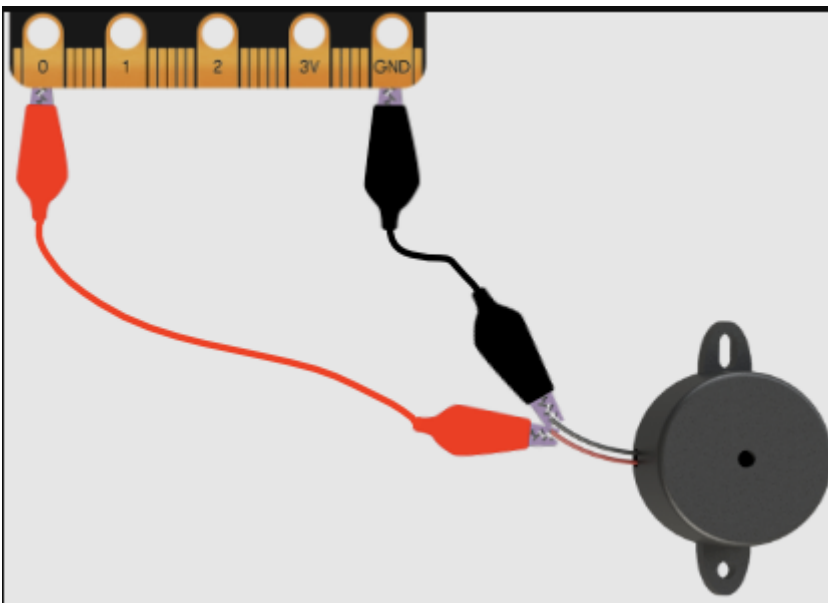
<https://www.youtube.com/embed/0u3hjXXEZYg>

# Musica

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

MicroPython de BBC micro:bit viene acompañado de un potente módulo de música y sonido. Es muy fácil generar pitidos y zumbidos desde el dispositivo conectando un altavoz o unos auriculares con cable, o utilizando el altavoz integrado si estamos con una versión V2.

La forma de conectar unos auriculares está descrita en el apartado de MakeCode. También se puede conectar un zumbador piezoeléctrico pasivo o un altavoz con pinzas de cocodrilo. Estos elementos pueden estar polarizados por lo que tendremos que comprobar si existe un terminal "+", y si es así conectar al pin0.



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

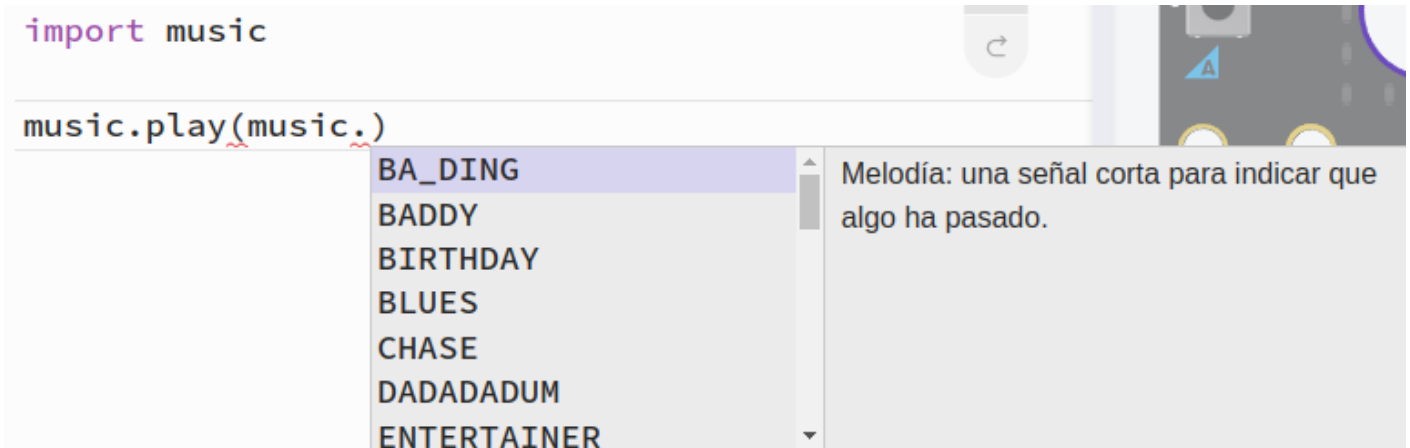
**Nota:** Debemos asegurarnos de que el zumbador es pasivo y no activo, que tan solo puede reproducir un tono. En el apartado 'Componentes discretos' de [Conceptos técnicos](#) podemos encontrar como distinguirlos.

Para trabajar con música hacemos:

```
import music
music.play(music.NYAN)
```

Tenemos que importar módulo `music` que contiene los métodos para crear y controlar el sonido.

La función de autocompletado de MicroPython nos muestra las melodías incorporadas.



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

En la referencia de la API podemos encontrar mas información en inglés sobre [Music](#)

Cada nota tiene un nombre como Do# (C#) o Fa (F), una octava, que indica lo alta o baja que debe tocarse y una duración. Las octavas se indican con un número, siendo 0 la octava más baja, 4 la del Do central y 8 es la más alta. Las duraciones también se expresan con números. Estos valores están relacionados entre sí: por ejemplo, una duración de 4 es el doble que una duración de 2 (y así sucesivamente). Si utilizamos como nombre de nota `R`, MicroPython reproducirá un silencio de la duración especificada.

Cada nota se expresa como una cadena de caracteres como ésta:

```
Nombre_nota[octave][:duration] #La1:4 (A1:4) es un La en la octava 1 con una duración de 4
```

Crear listas de notas para hacer una melodía es similar a crear una animación con una lista de imágenes. En el ejemplo vemos como sería la apertura de "Frere Jaques":

```
import music

frere_jaques_o = ["C4:4", "D4:4", "E4:4", "C4:4", "C4:4", "D4:4", "E4:4", "C4:4",
                  "E4:4", "F4:4", "G4:8", "E4:4", "F4:4", "G4:8"]
```

```
music.play(frere_jaques_o)
```

El ejemplo se puede re-escribir como vemos a continuación ya que los valores de octava y duración se repiten hasta que se indique un cambio.

```
import music

frere_jaques_o = ["C4:4", "D", "E", "C", "C", "D", "E", "C", "E", "F", "G:8",
                  "E:4", "F", "G:8"]

music.play(frere_jaques_o)
```

MicroPython nos permite crear tonos que no son notas musicales. Por ejemplo, este código crea un efecto de sirena de policía:

```
import music

while True:
    for frecuencia in range(880, 1760, 16):
        music.pitch(frecuencia, 6)
    for frecuencia in range(1760, 880, -16):
        music.pitch(frecuencia, 6)
```

El método `music.pitch` utiliza una frecuencia que puede ser la de una nota musical. En los rangos de `frecuencia` se especifican los tonos de los sonidos de una sirena como "valor inicial, valor final y paso". Cuando el paso es positivo sube el tono y cuando es negativo lo baja.

El ejemplo también nos muestra como anidar distintos tipos de bucle.

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



# Putty

Putty es un programa que nos permite realizar comunicaciones, normalmente se usa [en protocolo SSH](#), por ejemplo comunicarte con tu PC y con tu Raspberry con la ventana de comandos

Pero aparte del protocolo SSH también permite la comunicación serie, que con la Microbit es lo que vamos a utilizar:

## INSTALAR PUTTY EN LINUX

Actualizamos la lista de paquetes con **sudo apt update** podemos comprobar qué versión de Putty esta disponible con **apt show putty** e instalar Putty con el comando **sudo apt install putty -y**.

Si se quiere desinstalar **sudo apt purge putty -y**

## INSTALAR PUTTY EN WINDOWS

Entramos en <https://putty.org/> y en Downloads descarga y ejecuta el fichero correspondiente, si es un Intel x86 64bits al menos que sea un equipo viejo 32 bit. Si es un AMD insala el arm

### MSI ('Windows Installer')

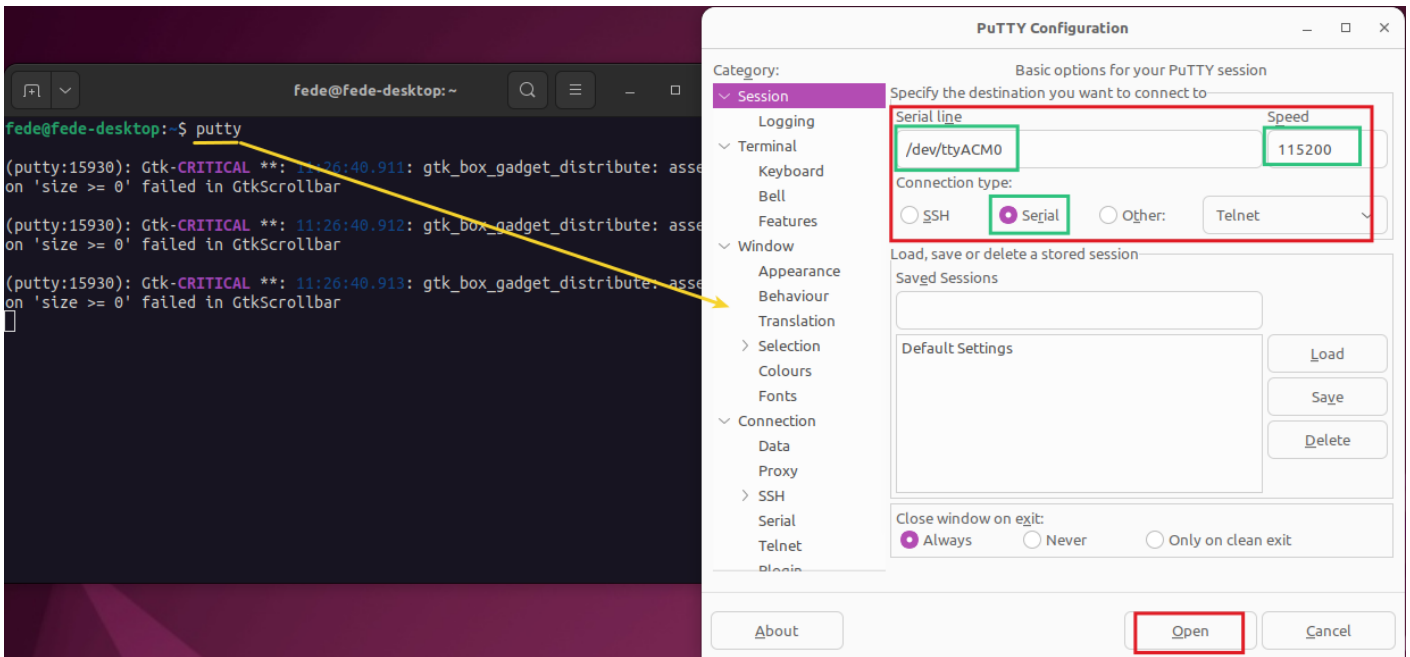
64-bit x86: [putty-64bit-0.81-installer.msi](#)

64-bit Arm: [putty-arm64-0.81-installer.msi](#)

32-bit x86: [putty-0.81-installer.msi](#)

## CONFIGURAR PUTTY PUERTO SERIE CON MICROBIT LINUX

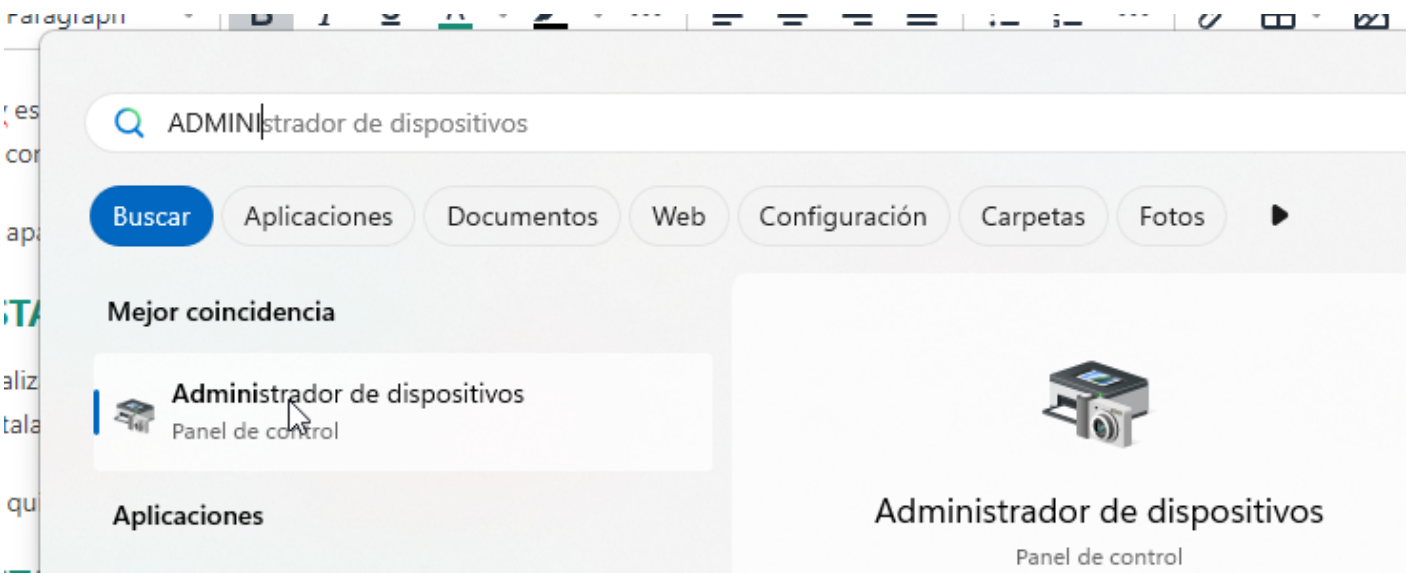
Conectamos nuestra microbit por el puerto serie, y ejecutamos la siguiente instrucción **ls /dev/ttyACM\*** normalmente será /dev/ttyACM0 por lo que en PUTTY ponemos este dato **con la velocidad 115200** :



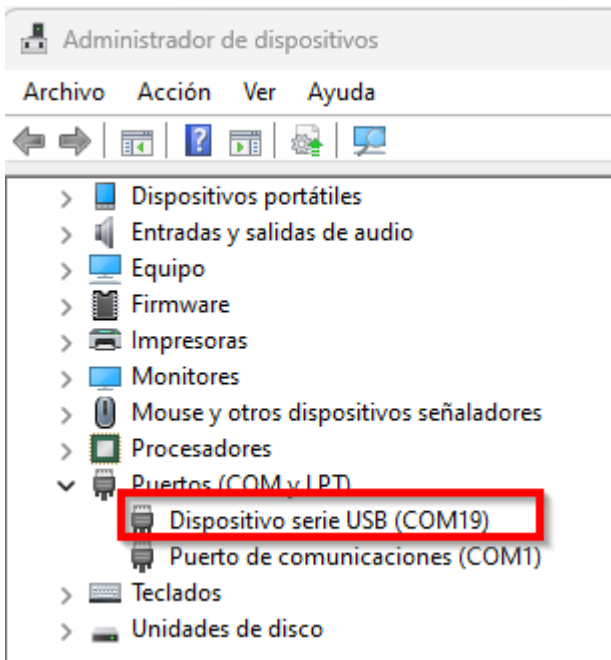
Autor Federico Coca de <https://fgcoca.github.io/Guia-de-trabajo-para-microbit/conceptos/serie/>  
licencia CC-BY-SA

## CONFIGURAR PUTTY PUERTO SERIE CON MICROBIT WINDOWS

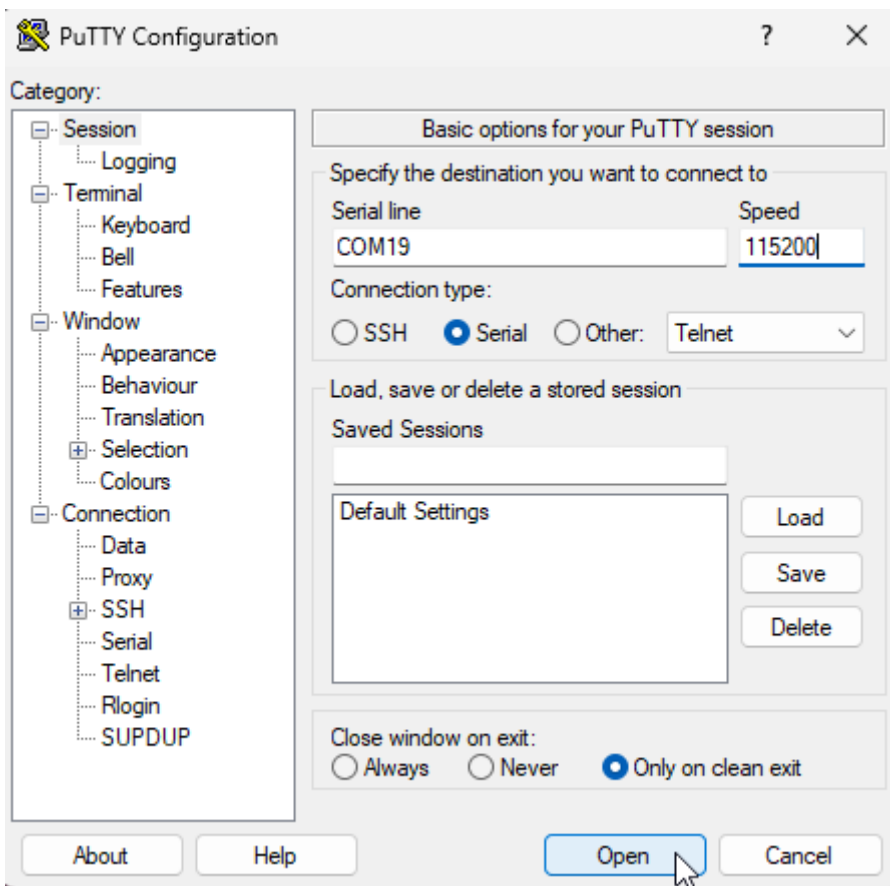
Conectamos nuestra microbit por el puerto serie, y ejecutamos el **Administrador de dispositivos**



Al ejecutarlo vemos que aparece un dispositivo nuevo conectado en los puertos COM (si tienes dudas, conecta y desconecta la microbit) en la imagen aparece el COM19



Por lo que en PUTTY ponemos este dato **con la velocidad 115200** :





**ATENCIÓN:** ESTAS OCUPANDO EL MISMO PUERTO SERIE QUE LA COMUNICACIÓN DEL EDITOR PYTHON <https://python.microbit.org/> o MU

SI QUIERES VOLVER A INSTALAR UN PROGRAMA **TIENES QUE CERRAR PUTTY** (sino, el editor da error al flashear pues no puede comunicarse con la placa microbit)

# UART

Página extraída de Federico Coca [Guia de Trabajo de Microbit](https://fgcoca.github.io/Guia-de-trabajo-para-microbit/conceptos/serie/) <https://fgcoca.github.io/Guia-de-trabajo-para-microbit/conceptos/serie/> Licencia CC-BY-SA

En MicroPython es el módulo `uart` el que permite comunicarse a través de la interfaz serie entre la `micro:bit` y el ordenador. El módulo tiene diferentes funciones encomendadas a realizar diferentes tareas de comunicación serie, pero sobre todo hay una fundamental que es la de inicializar la `micro:bit` para trabajar.

```
microbit.uart.init(baudrate=9600, bits=8, parity=None, stop=1, *, tx=None, rx=None)
```

Inicializa la comunicación serie con los parámetros especificados en los pines `Tx` y `Rx` especificados. Hay que tener en cuenta que, para que la comunicación sea correcta, los parámetros deben ser los mismos en los dispositivos que se comunican. En la función tenemos:

- **baudrate.** Define la velocidad de la comunicación en baudios y puede ser: 9600, 14400, 19200 (2x9600), 28800 (14400x2), 38400 (2x19200), 57600 (28800x2) o 115200 (57600x2).
- **bits.** El parámetro `bits` define el tamaño de los bytes que se transmiten. `Micro:bit` solamente soporta 8 bits.
- **parity.** El parámetro `paridad` define la forma de comprobación de la paridad pudiendo valer: `none`, `microbit.uart.ODD` o `microbit.uart.EVEN`, lo que indica: ninguna, impar o par respectivamente. La paridad es una forma de comprobar que el dato transmitido y el recibido coinciden.
- **stop.** Este parámetro indica el número de bits de parada, que en el caso de la `micro:bit` es uno.
- **Tx y Rx.** Son los pines de transmisión (`Tx`) y recepción (`Rx`) de la placa. Si no se especifican se utilizan los internos de USB/UART. Se puede especificar cualquier otro pin.

## Notas:

- Inicializar la UART en pines diferentes a los establecidos por defecto para USB puede originar que la consola de Python deje de ser accesible, ya que utiliza el mismo hardware. Para recuperar la consola hay que reinicializar la UART sin pasar nada por `Tx` o `Rx` (o

pasando None a estos argumentos). Es decir, llamar a `uart.init(115200)` es suficiente para restaurar la consola Python.

- Las conexiones de dispositivos mediante Tx y Rx requieren "cruzar" los cables, de forma que el pin TX de la placa esté conectado al Rx del otro dispositivo y el pin Rx, con el pin Tx. De esta forma lo que uno transmite el otro lo escucha. También es imprescindible que los dispositivos tengan un GND común.

El ejemplo siguiente nos va a permitir comprobar la comunicación serie desde MicroPython. Grabamos el programa en la micro:bit.

```
from microbit import *

uart.init(115200,8,None,1)
while True:
    if button_a.was_pressed():
        #"\n" indica un salto de linea
        #\r" indica un retorno de carro
        #Con los dos se salta una linea y se lleva
        #el curso al principio
        uart.write("Python: Boton A pulsado\n\r")
```

Abrimos y configuramos PuTTY y la consola nos muestra:

Página extraída de Federico Coca [Guia de Trabajo de Microbit](https://fgcoca.github.io/Guia-de-trabajo-para-microbit/conceptos/serie/) <https://fgcoca.github.io/Guia-de-trabajo-para-microbit/conceptos/serie/> Licencia CC-BY-SA



COM19 - PuTTY

```
Python: Boton A pulado
Python: Boton A pulsado
Python: Boton A pulsado
Python: Boton A pulsado
Python: Boton A pulsado
Python: Boton A pulsado
Python: Boton A pulsado
Python: Boton A pulsado
Python: Boton A pulsado
Python: Boton A pulsado
Pyhon: Boton A pulsado
Python: Boton A pulsado
Python: Boto A pulsado
```

<https://www.youtube.com/embed/44kSyEAdF6Q>

En el simulador de Micropython también aparece

```

1 from microbit import *
2
3 uart.init(115200,8,None,1)
4 while True:
5     if button_a.was_pressed():
6         #"\n" indica un salto de linea
7         #\r" indica un retorno de carro
8         #Con los dos se salta una linea y se lleva
9         #el curso al principio
10        uart.write("Python: Boton A pulsado\n\r")

```

# Registro de datos

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

## Grabar datos

Para utilizar el Registro de datos con micro:bit V2 tenemos disponibles en Python:

- `import log`. Importamos el módulo, como siempre al principio del código, para tener disponibles las funciones de registro de datos.
- `log.set_labels()`. Para configurar los encabezados de las columnas del registro de datos. Por ejemplo: `log.set_labels('temperatura', 'sonido', 'luz')`.
- `log.add()`. Añadir entradas al registro de datos. Por ejemplo:

```
log.add({
    'temperatura': temperature(),
    'sonido': microphone.sound_level(),
    'luz': display.read_light_level()
})
```

- `run_every()`. Programar entradas de registro en el intervalo especificado de tiempo. Puedes utilizar un programador para registrar datos automáticamente a intervalos regulares. `run_every` puede utilizarse de dos formas:
  - Como **Decorador** - se coloca encima de la función a programar. Por ejemplo:

```
@run_every(days=1, h=1, min=20, s=30, ms=50)
def mi_funcion():
    # Hacer lo que sea
```

- Como una **función** - pasando la llamada de retorno como argumento posicional. Por ejemplo:

```
def mi_funcion():
    # Hacer lo que sea
run_every(mi_funcion, s=30)
```





Cada argumento corresponde a una unidad de tiempo diferente y son aditivos. Así,

`run_every(min=1, s=30)` programa la llamada de retorno cada minuto y medio.

Cuando se lanza una excepción dentro de la función callback se desprograma la función. Para evitar esto puedes atrapar excepciones con `try/except`.

Los parámetros son:

- `callback` - Function to call at the provided interval.
- `days` - Establece la marca de días para la programación.
- `h` - Establece la marca de horas para la programación.
- `min` - Establece la marca de minutos para la programación.
- `s` - Establece la marca de segundos para la programación.
- `ms` - Establece la marca de milisegundos para la programación.

## Ejemplo grabación de registro simple

A continuación vemos un ejemplo de registro:

```
from microbit import *
import log

log.set_labels('temperatura', 'sonido', 'luz', timestamp=log.SECONDS)

@run_every(s=5)
def reg_dato():

    log.add(temperatura=temperature(),sonido=microphone.sound_level(),luz=display.read_light_level
    ())

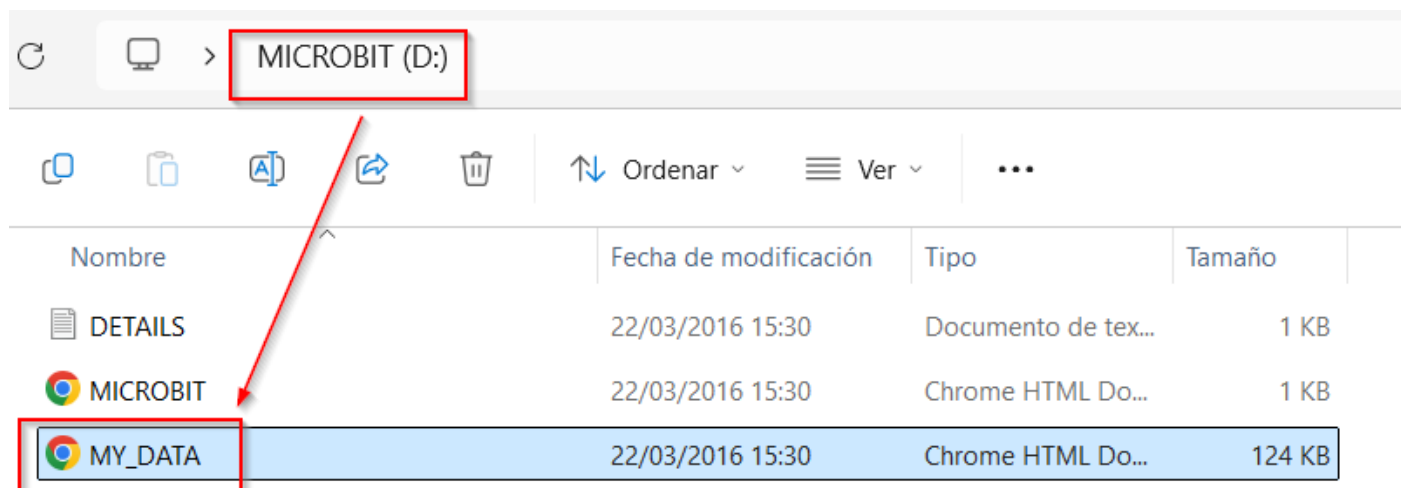
while True:
    sleep(500)
```

<https://www.youtube.com/embed/jBymplVqT-8>

## Lectura de datos

Para ver datos reales grabados en la micro:bit utilizaremos el ejemplo anterior de registro automáticos de aceleraciones.

Una vez que tenemos datos registrados en la micro:bit, la conectamos a un ordenador y dejamos que se monte como una unidad USB de nombre MICROBIT. Si abrimos esta unidad nos vamos a encontrar con tres archivos, uno de ellos es "MY\_DATA.HTM".



Si hacemos doble clic sobre el archivo "MY\_DATA.HTM" se nos abrirá en una ventana de nuestro navegador por defecto.



## micro:bit data log

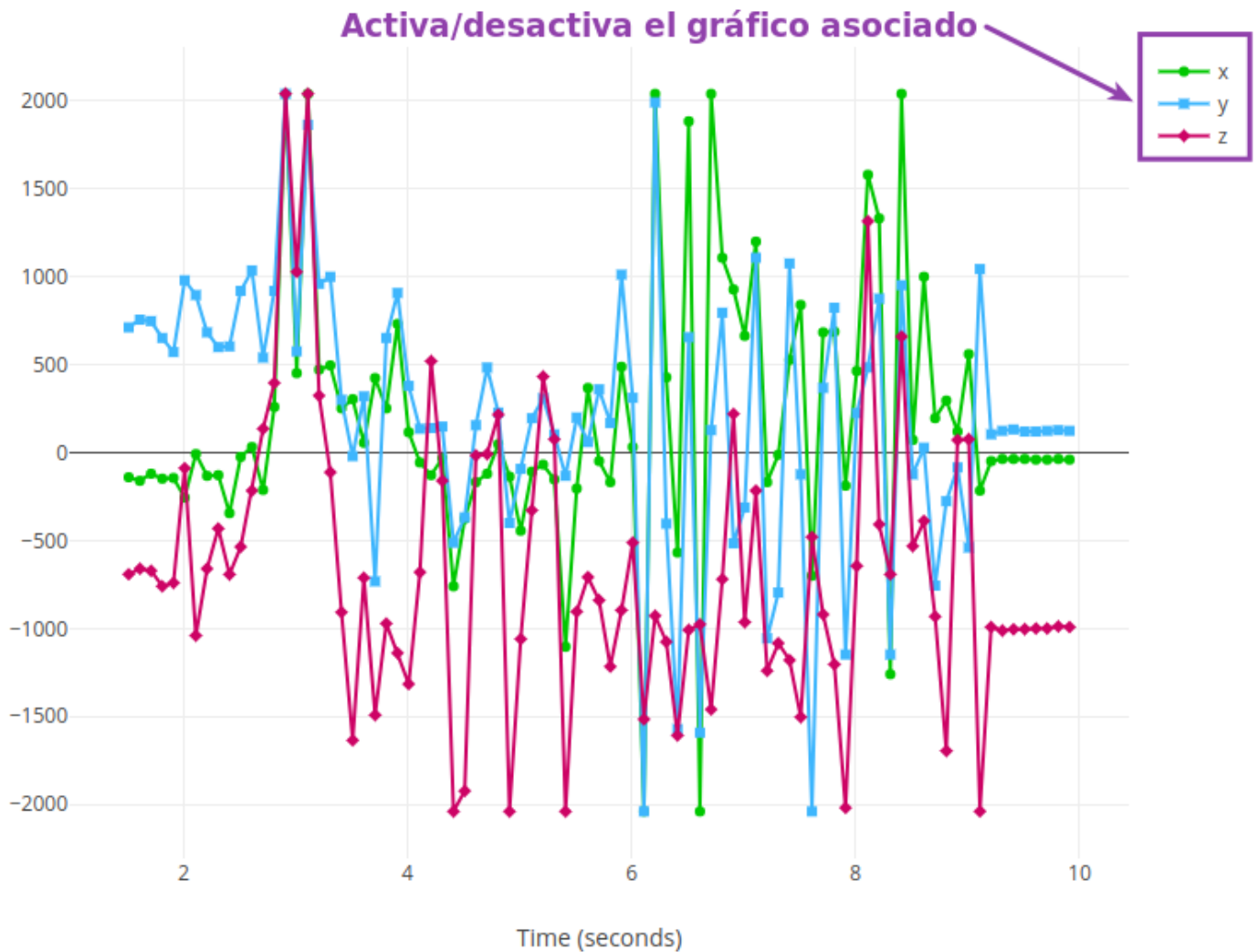
[Download](#)
[Copy](#)
[Update data...](#)
[Clear log...](#)
[Visual preview](#)

This is the data on your micro:bit. To analyse it and create your own graphs, transfer it to your computer. You can copy and paste your data, or download it as a CSV file which you can import into a spreadsheet or graphing tool. [Learn more about micro:bit data logging.](#)

Time (seconds)	temperatura	sonido	luz
6.22	21	0	0
11.19	22	0	0
16.19	22	0	0
21.19	22	72	0
26.19	23	0	0
31.19	23	0	0

Los botones nos muestran diferentes opciones que podemos realizar con estos datos:

- **Descargarlos (Download).** Se guardan los datos en formato CSV con los valores separados por comas. Estos datos se pueden importar a una hoja de cálculo y realizar todo tipo de análisis con los mismos.
- **Copiarlos (Copy).** Realiza una copia de los datos en el portapapeles para que podamos pegarlos donde queramos, como por ejemplo en una hoja de cálculo. De esta forma no tenemos que descargar el archivo CSV.
- **Actualizarlos (Update data).** Comprueba si los datos en la micro:bit han cambiado respecto a la lectura actual desconectando y conectando la micro:bit del puerto USB.
- **Borrar registro (Clear log).** Nos muestra un mensaje indicando que el registro se borra cuando regrabemos la micro:bit. El programa puede incluir código o bloques para borrar el registro cuando queramos, como es el caso del ejemplo. Por ahora este botón no borra los datos en la micro:bit.
- **Previsualización (Visual preview).** Muestra los datos obtenidos de forma gráfica. Se pueden mostrar y ocultar utilizando los iconos de la leyenda. En la imagen vemos estos gráficos.



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

## Ejemplo más extenso

Vamos a realizar una actividad en la que registraremos la temperatura y la luz ambiente en la misma micro:bit que contiene el programa

El programa que vemos a continuación realiza un registro automático cada 10 segundos o cuando pulsemos el botón A. Pulsando A+B se borran los datos registrados en la microbit.

```
from microbit import *
import log
```

```
# Configurar etiquetas y establecer la unidad de tiempo
log.set_labels("temperatura", "nivel_luz", timestamp=log.SECONDS)
display.show(Image.NO)
sleep(1000)
# Enviar cada fila de datos a la salida serie
log.set_mirroring(True)

continue_registro = True

# Decorador programado para que se ejecute cada 10s durante 50ms
@run_every(s=10, ms=50)
def reg_dato():
    # Registra cada 10s temperatura y nivel de luz y muestra un icono
    global continue_registro
    if continue_registro:
        display.show(Image.YES)
        try:
            log.add(temperatura=temperature(), nivel_luz=display.read_light_level())
        except OSError:
            continue_registro = False
            display.show(Image.CHESSBOARD)
            sleep(500)

while True:
    if button_a.is_pressed() and button_b.is_pressed():
        display.show(Image.GHOST)
        # Borra el archivo de registro con la opcion "full" lo
        # que asegura el borrado de datos aunque tarde mas tiempo.
        log.delete(full=True)
        continue_registro = True
    elif button_a.is_pressed():
        display.show(Image.YES)
        sleep(500)
        log.add(temperatura=temperature(), nivel_luz=display.read_light_level())
        display.show(Image.HEART)
```

```
else:
    display.show(Image.N0)
    sleep(500)
```

A continuación vemos el registro de datos tras unos segundos y un par de entradas manuales:

## micro:bit data log

Download



Copy

Update data...

Clear log...

Visual preview

This is the data on your micro:bit. To analyse it and create your own graphs, transfer it to your computer. You can copy and paste your data, or download it as a CSV file which you can import into a spreadsheet or graphing tool. [Learn more about micro:bit data logging.](#)

Time (seconds)	temperatura	nivel_luz
12.42	25	53
22.44	25	60
32.49	25	57
 35.50	25	60
42.54	25	64
52.59	25	71
 56.10	25	71
62.64	25	68

Entradas manuales de datos

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

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