# Microbit y Python


- Objetivos y contenidos
- Lenguajes de programación
- Introducción
  - Microbit v2 vs v1
  - Ventajas y desventajas Python
  - Editores
  - Introducción al Python
  - Micropython de microbit
  - Hoja de Ruta
- Empezando
  - Hola Mundo
  - Imágenes
  - Imágenes estáticas y animadas
  - Eventos para los botones
  - Botones
- Jugando
  - Nivel de luz
  - Temperatura
  - Magnetómetro
  - Acelerómetro
  - Micrófono


Copyright 2026

    - Radio

- Hackeando
    - Pines de Entrada/salida
    - Input output
    - Música predefinida o crea tu música
    - Musica
    - Putty
    - UART
    - Registro de datos

- Créditos

# Objetivos y contenidos

## Objetivos

- Conocer las posibilidades de la placa microbit sobre todo las versiones más actuales
- Saber utilizar el lenguaje de programación en código frente al gráfico con fines didácticos
- Realizar programas sencillos con Python utilizando las posibilidades de la placa Microbit
- Conocer estrategias de electrónica básica dentro del campo de entradas y salidas para realizar proyectos didácticos.

## Contenidos

- **Introducción**
  - Microbit v2 vs v1
  - Ventajas y desventajas de Python
  - Editores
  - Introducción al Python
  - Micropython de Microbit
  - Para saber más de Python
- **Empezando**
  - Hola Mundo
  - Imágenes
  - Imágenes estáticas y animadas
  - Eventos para los botones
  - Botones
- **Jugando**
  - Nivel de luz
  - Temperatura
  - Magnetómetro
  - Acelerómetro
  - Micrófono
  - Radio
- **Hackeando**
  - Pines de entrada/salida
  - Input/Output
  - Música predefinida o crea tu música
  - Música
  - Putty

Copyright 2026

- UART
- Registro de datos

Copyright 2026

# Lenguajes de programación

Micro:bit se puede programar :

| Tipo de lenguaje | Lenguaje | Ventajas | Inconvenientes | En AULARAGON |
|---|---|---|---|---|
| **BLOQUES** | **MAKECODE** https://makecode.microbit.org/ | -Muy extendido<br>-Existen en Internet muchos recursos y ejemplos<br>-Con CretateAI podemos usar IA | | -micro:bit<br>-micro:bit+car<br>-Smart Home<br>-Smart Agriculture |
| **BLOQUES** | **SCRATCH+LearningML** https://learningml.org/lml-scratch/ | -Se puede usar IA<br>-En vivo | | Es el que proponemos en el curso de micro:bit en LearningML |
| **BLOQUES** | **Classroom Microbit** https://classroom.microbit.org/ | - Se puede crear una clase, enviar proyectos a los alumnos, corregir.... | | |
| **BLOQUES** | **Microblocks** https://microblocks.fun/ | -Código abierto | | |
| **BLOQUES** | **mBlock** | -Permite usar multitud de extensiones | Recomendamos usar la versión descargada no la online | |
| **CÓDIGO** | **PYTHON** https://python.microbit.org/ | - Se usa todo el potencial del código | - Curva de aprendizaje alta, no recomendado en primaria. | Micro:bit y Python |

Guía de los lenguajes de programación para robots

https://docs.google.com/presentation/d/e/2PACX-1vQb1Dv9wN9QK-F6V7yvwDoyzquqwWlGvIyVJr83Yk56kAoYD7bXLnYDm_tCQkeAgg/pubembed?start=falseloop=false

Copyright 2026

&delayms=3000

> Tenemos un **grupo Telegram Robótica Educativa en Aragón**,
> https://t.me/roboticaeducativaaragon



# Introducción

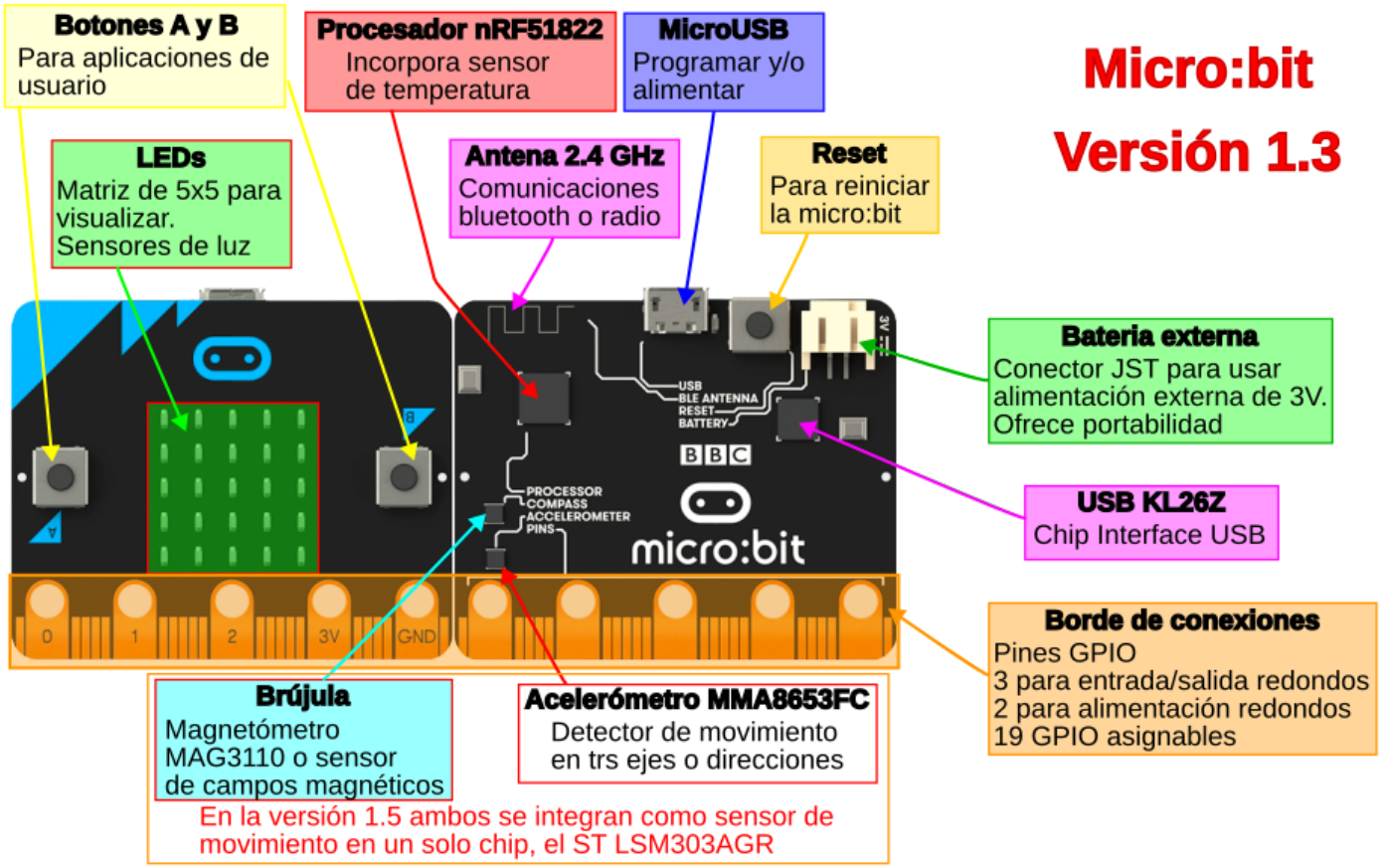
Copyright 2026

Introducción

# Microbit v2 vs v1

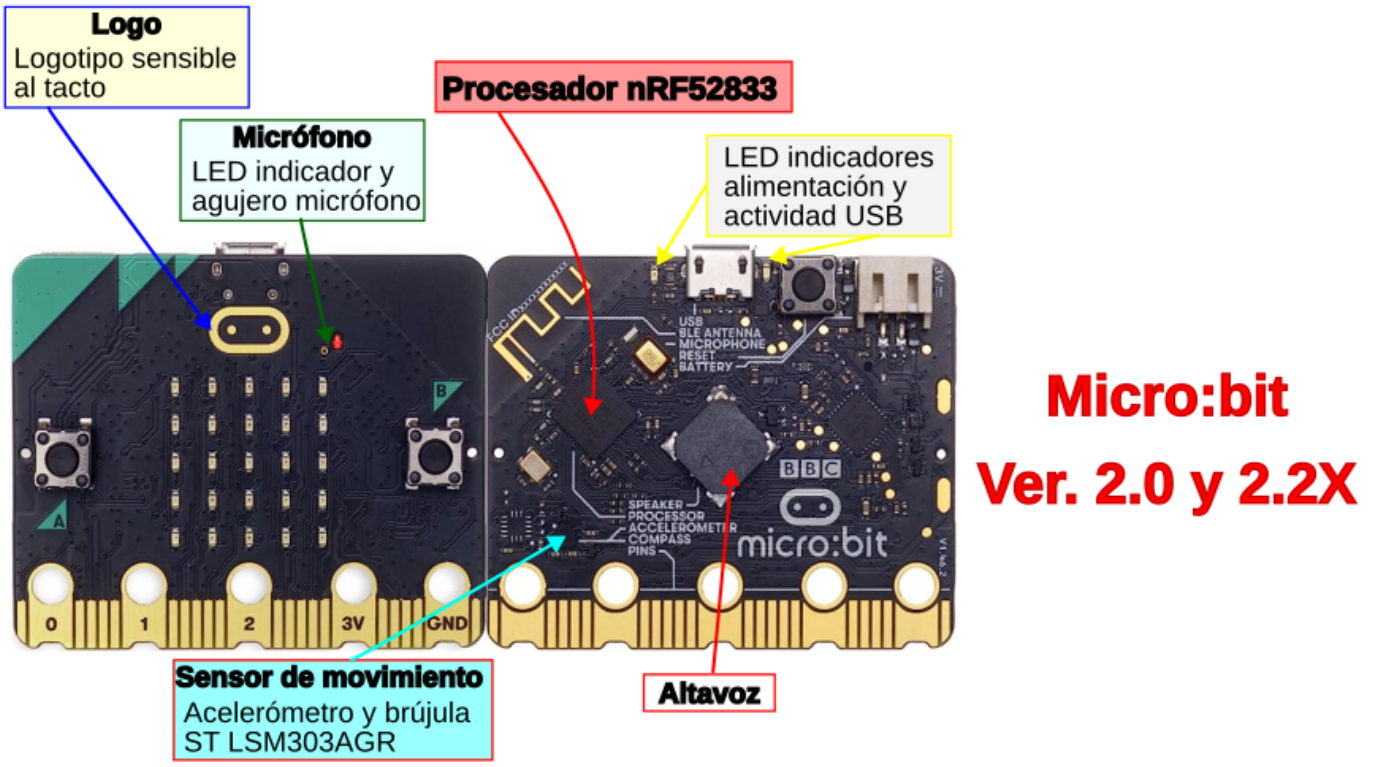
> Página extraída de Federico Coca Guia de Trabajo de Microbit CC-BY-SA

La microbit fué lanzada en el 2015  con la versión 1 que tiene los siguientes sensores y actuadores




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

En el 2020 se lanzó la versión 2 que tiene además incorporó estas características




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

Además de estas que no se ven:

- Modo ahorro de energía. Esta nueva función de ahorro o modo de espera detendrá el programa que se esté ejecutando en la micro:bit hasta que se pulse el botón de reinicio.
- Microprocesador cuatro veces más potente que la que tenía la v1
- Ocho veces más memoria que la v1 (512 KB de memoria Flash y 128 KB de memoria RAM)

La alimentación es a través de los 5V del puerto USB o 3V a través del conector JST. También es posible alimentar a la micro:bit desde los anillos 3V/GND en el conector de borde.

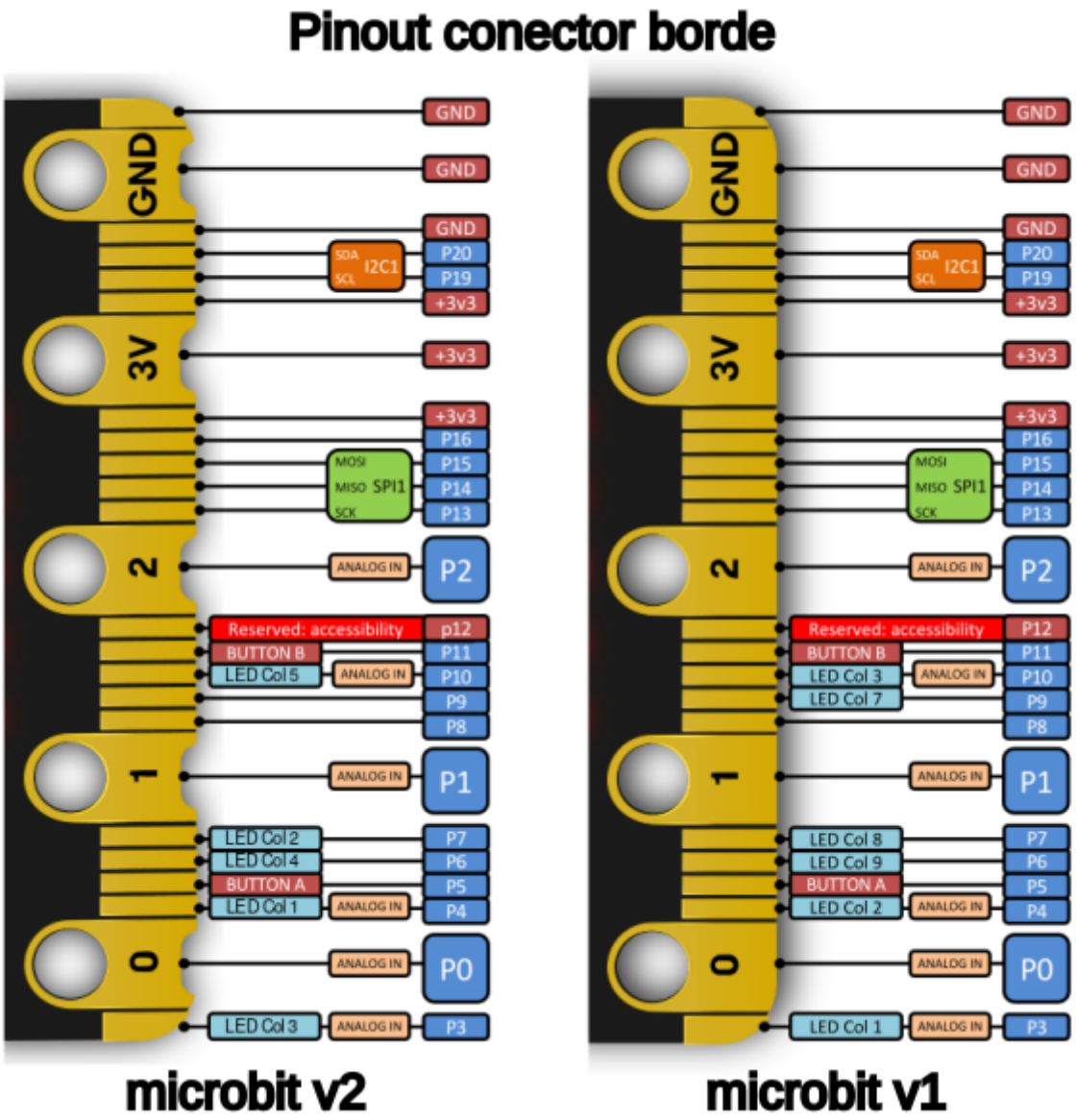
## Pinout

Hay 25 pistas/patillas que incluyen 5 agujeros para usar con clavijas tipo banana de 4 mm o pinzas de cocodrilo. Tres de estos anillos son para entrada y salida de propósito general (GPIO) y también sirven para detección analógica, PWM y táctil, y dos están conectados a la alimentación de la micro:bit.

Solamente tienen conexión las pistas frontales, las posteriores están sin conexión y los anillos posteriores están conectados a los delanteros. Las pistas mas finas están separadas 1,27 mm, algunas son utilizadas por micro:bit y otras están disponibles para su uso mediante cualquiera de los conectores externos existentes, lo que permite un amplio mercado de accesorios externos. En

en enlace tenemos una guia de accesorios para micro:bit

En la imagen siguiente tenemos la descripción de pines de la micro:bit v2 a la izquierda y de la v1 a la derecha para poder comparar y establecer las diferencias de una forma sencilla.




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

En microbit.pinout.xyz tenemos un fantástico recurso para obtener más información sobre los pines de la micro:bit y de cómo los utilizan algunos accesorios.

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

Copyright 2026

Introducción

# Ventajas y desventajas Python

## Ventajas

Python es un lenguaje de desarrollo y curva de aprendizaje rápido. Tiene una comunidad amplia con muchas librerías, ejemplos, tutoriales... que para casi todos los problemas, seguro que encuentras una solución escrita en Python

Es un lenguaje de alto nivel, es decir, que se programa igual que los programadores, pero interpretable para los humanos !. Comparándolo con otros lenguajes (Java, C++, etc..) es el más "*humanizado*".

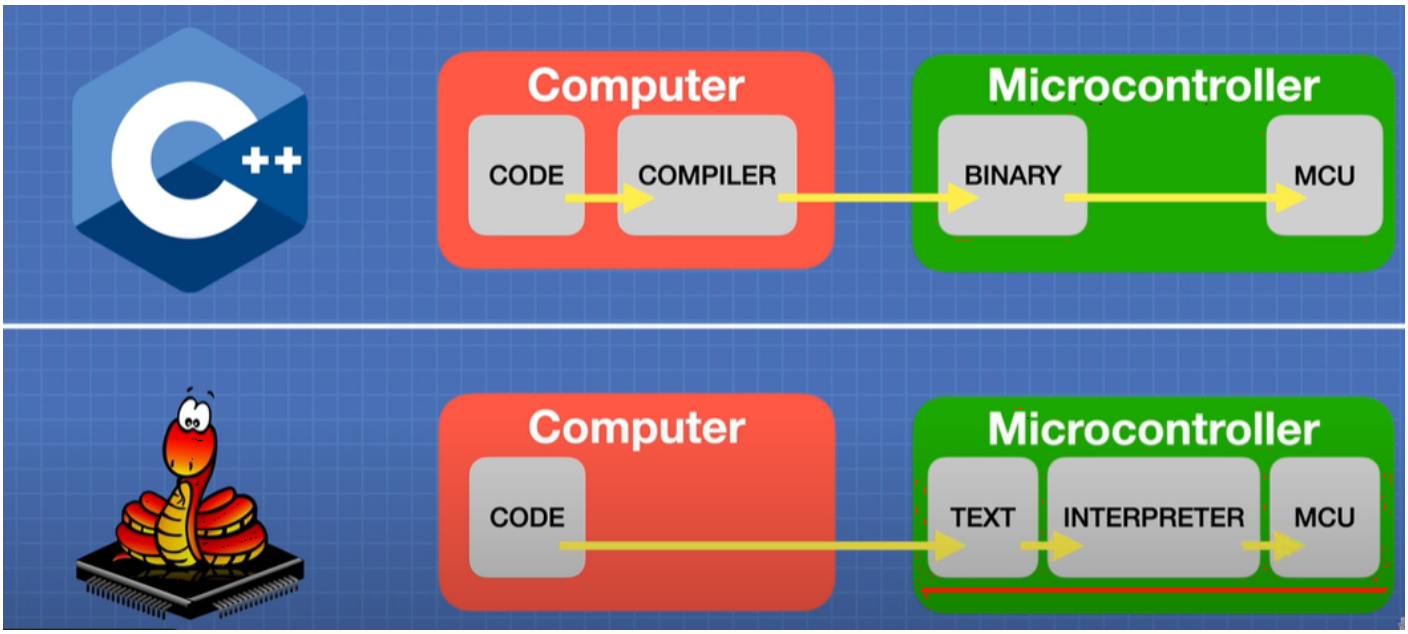
También gestiona la memoria, por ejemplo, si programas en C++, tú eres el responsable de limpiar la memoria de datos que ya no usas, o corres el peligro de quedarte sin memoria. En Python ya lo hace por ti.

## Desventajas

La gestión de memoria que antes se mencionaba tiene un precio; bajada de velocidad y paradójicamente coste de memoria.

En otros programas, el compilador esta en tu PC, pero en Python está en el dispositivo (por eso se llama lenguaje **Interpretado**), esto hace que ocupa memoria, y en microbit por ejemplo esto hace que no puedes usar Python y código Bluetooh pues no hay suficiente memoria RAM.

Copyright 2026




Fuente  vídeo Exploring the Arduino Nano ESP32 | MicroPython & IoT

También hay que tener en cuenta que si Python es un lenguaje **interpretado**, siempre será más lento que un lenguaje **compilado** por ejemplo el C++, pues para ejecutarlo el dispositivo, lo ejecuta, pues lo tiene en binario y en paz, pero en Python cada instrucción necesita ser interpretado, decodificado, en binario antes de ejecutarse.

Copyright 2026

Introducción

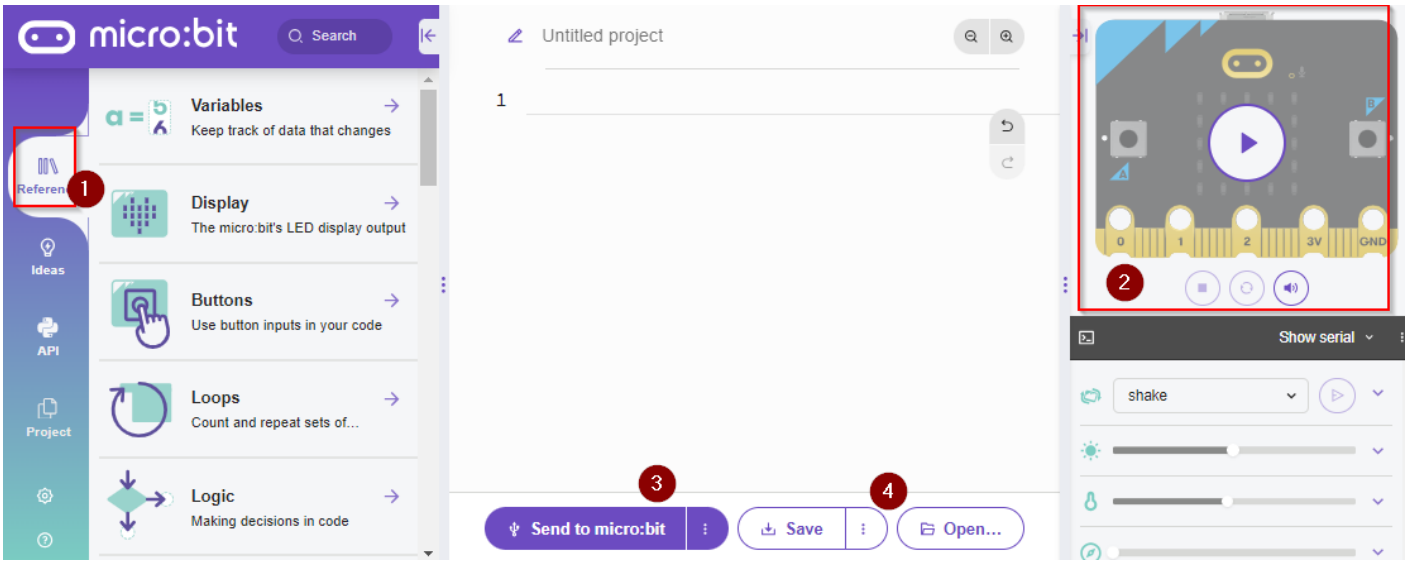
# Editores

Tienes dos opciones, online o local :

## Programar online con https://python.microbit.org (recomendado)

Entramos en https://python.microbit.org/ y el editor online nos permite trabajar ;

1. Una biblioteca de códigos que nos permitirá seleccionar y usar para programar de forma guiada
2. Un simulador para ver cómo se ejecutaría nuestro código
3. Un botón para enviar a la microbit real
4. Botones para guardar nuestro código de forma local y abrir los existentes.



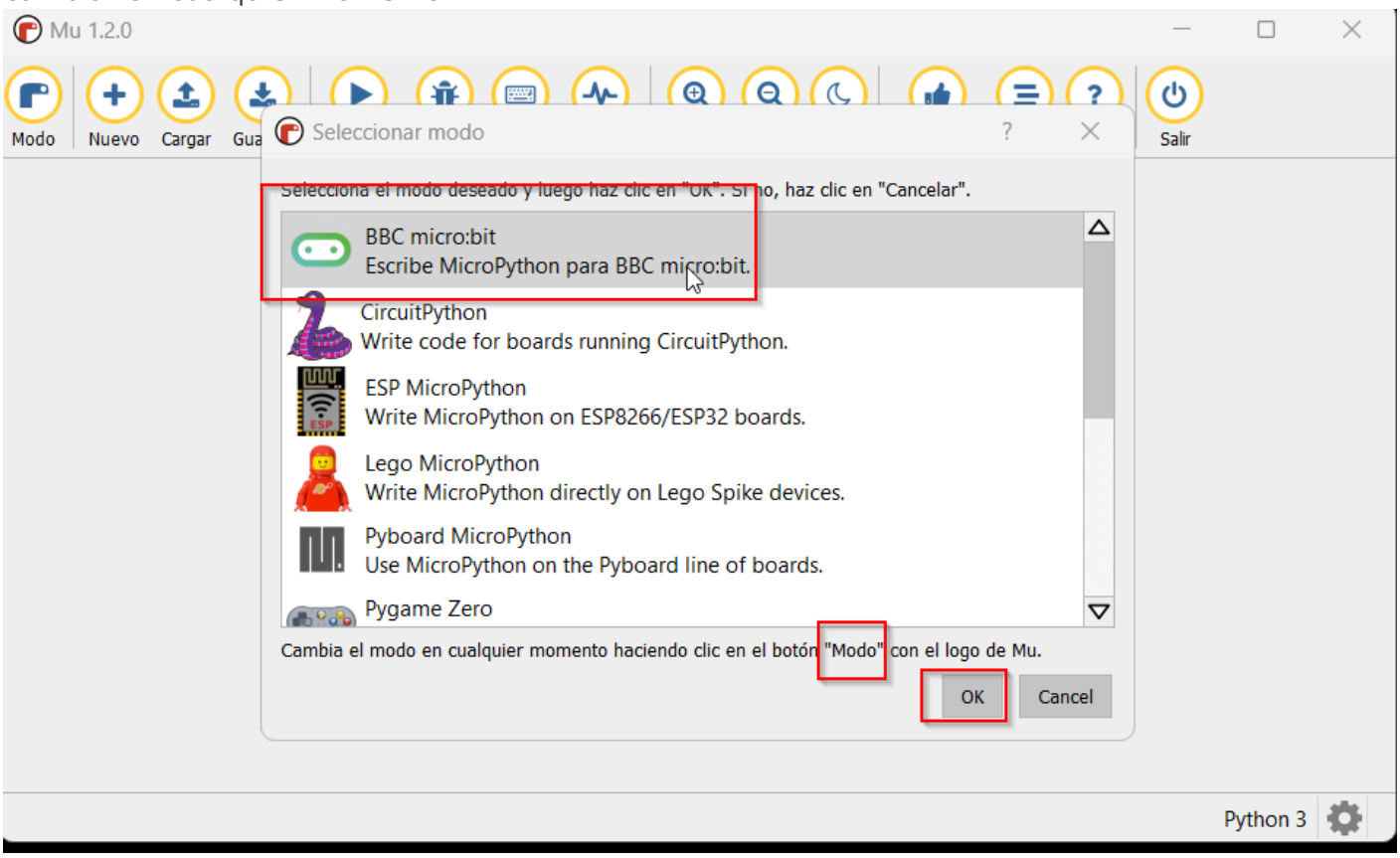

> En este curso utilizaremos el editor online microbit.org

## Programar en local con MU

Es un editor muy sencillo, se descarga en https://codewith.mu/ y permite su instalación en Windows, Linux y Apple.
2024-07-04 18_44_27-(1) Exploring the Arduino Nano ESP32 _ MicroPython & IoT Cloud - YouTube.pr

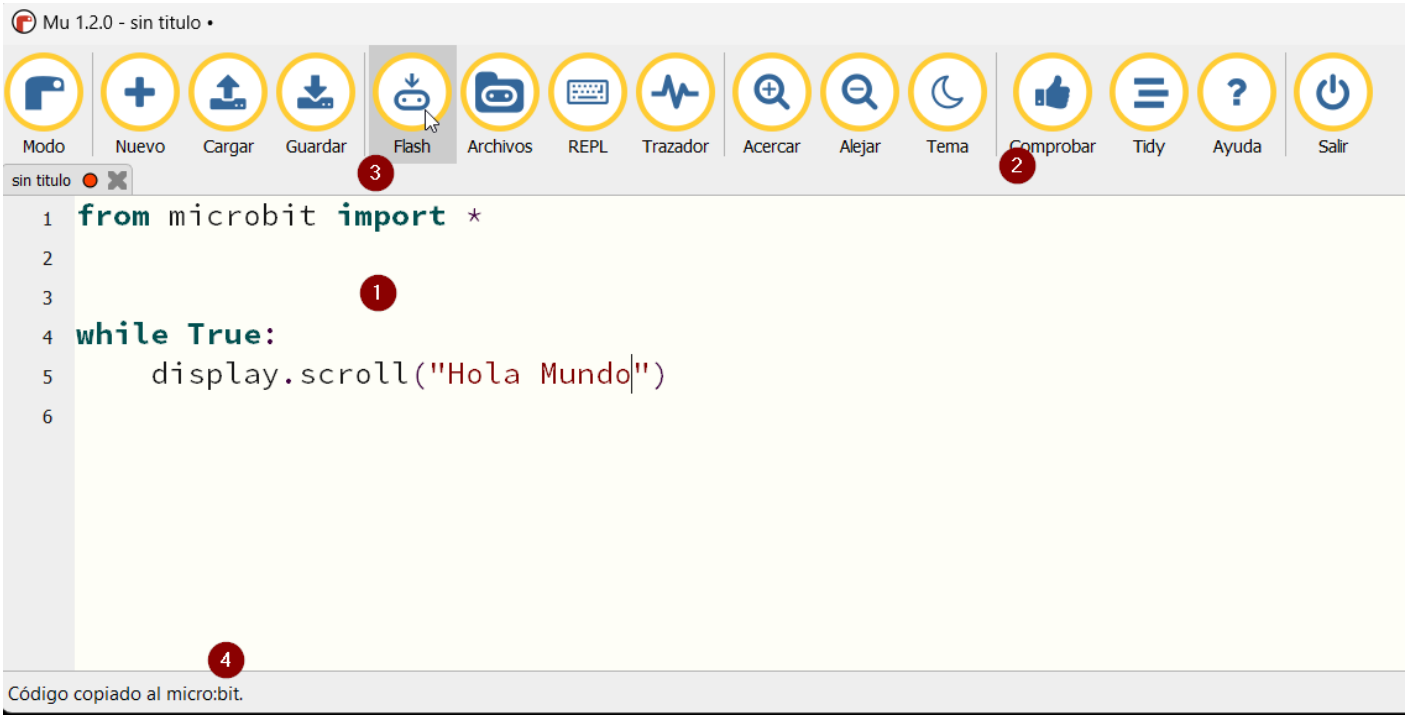
Fuente https://codewith.mu/ CC-BY-NC-SA

La primera vez que lo ejecutamos (tarda algo la primera vez) nos pide el **modo** que se puede cambiar en cualquier momento:




1 Escribimos el código
2 Lo comprobamos
3 Flasheamos, es decir enviamos el código al Microbit (conectarlo previamente)
4 Cuando sale el mensaje *Código copiado al microbit* **procedemos a resetearlo** para que la placa ejecute el programa.

> ATENCIÓN ES IMPORTANTE **RESETEAR LA MICRO:BIT** tienes un botón de ***reset*** al lado del conector de USB para no estar desconectando y conectando. Una vez reseteado tu programa funcionará.




```
from microbit import *


while True:
  display.scroll("Hola Mundo")
```

## OTROS EDITORES DE PYTHON QUE NO SON COMPATIBLES CON PYTHON MICROBIT

Vamos a ver este programa escribo en https://python.microbit.org/

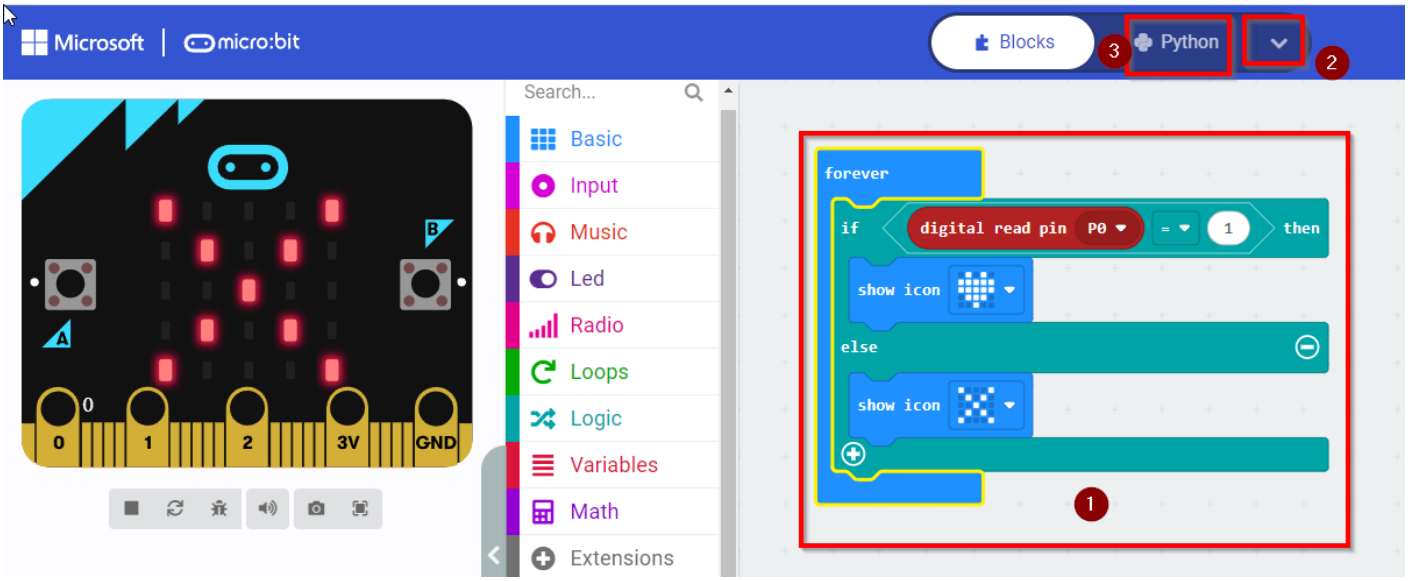
```
# Imports go at the top
from microbit import *
while True:
    if pin0.is_touched():
        display.show(Image.HEART)
    else:
        display.show(Image.NO)
```

Lo que hace es :

https://www.youtube.com/embed/uI2p9HazV1Y

## EL MISMO CÓDIGO EN MAKECODE-PYTHON

Makecode a pesar de que esta orientado a programar con bloques, t**iene su sección de Python**



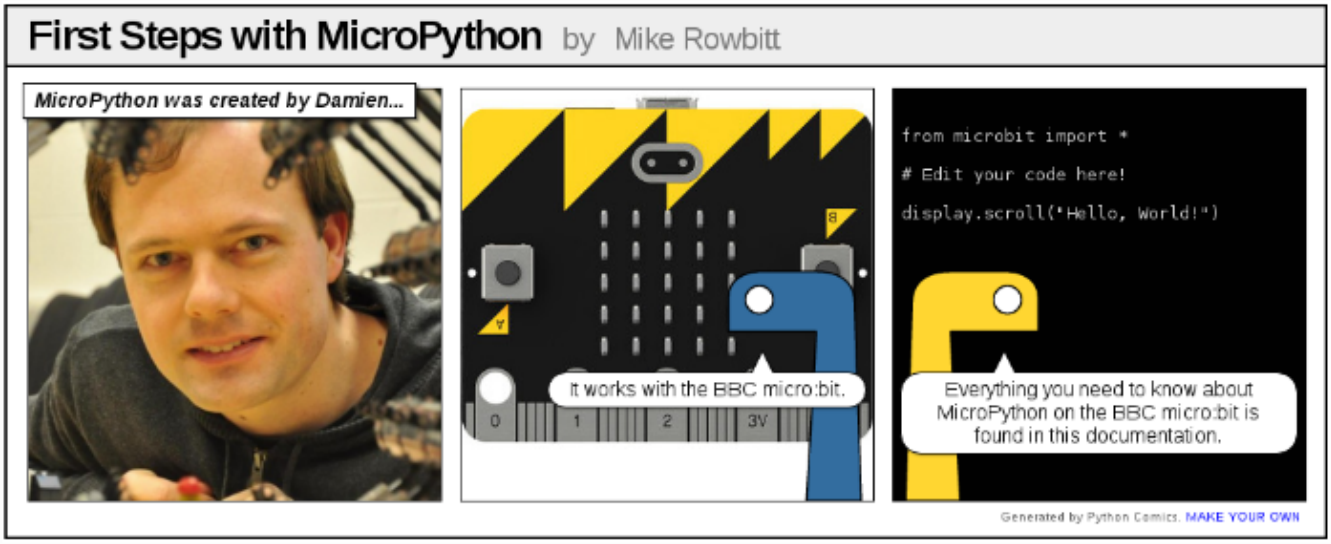

Al darle en Python (arriba a la derecha), muestra este código

```
def on_forever():
    if pins.digital_read_pin(DigitalPin.P0) == 1:
        basic.show_icon(IconNames.HEART)
    else:
        basic.show_icon(IconNames.NO)
basic.forever(on_forever)
```

Como se puede ver **makecode python no es compatible con https://python.microbit.org/** ya lo dice en su tutorial https://microbit-micropython.readthedocs.io/en/v2-docs/

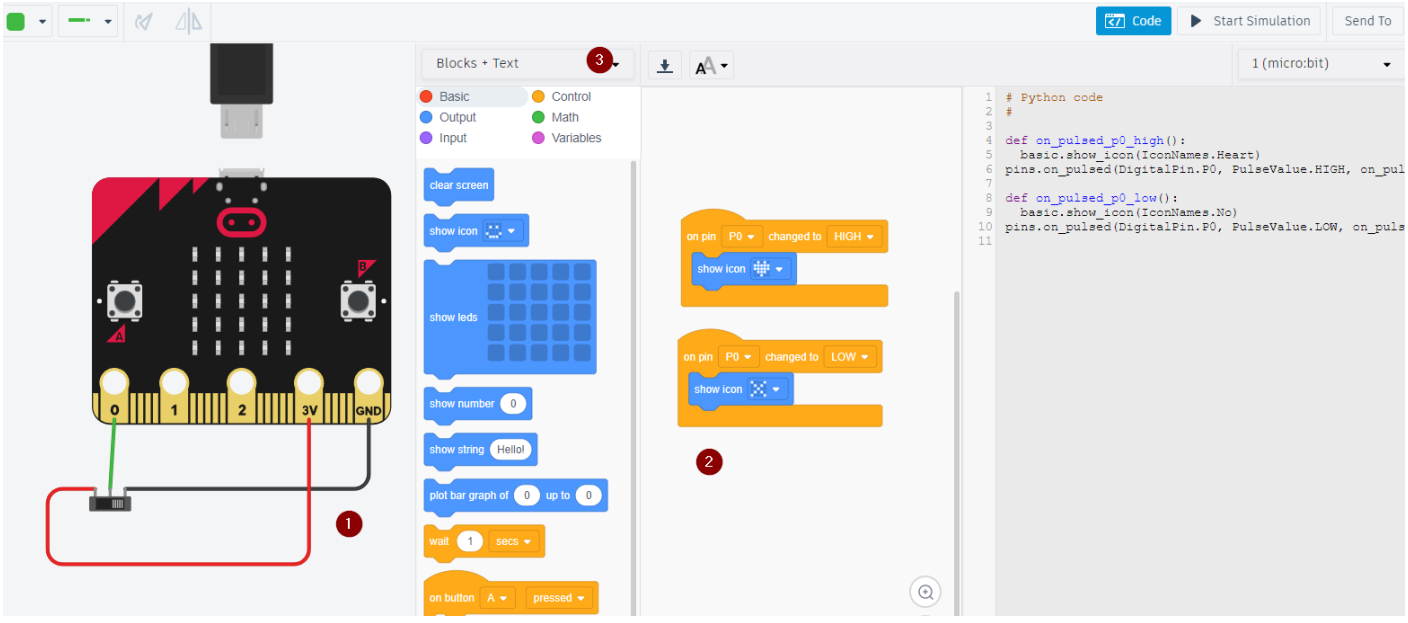
**Note**

The MicroPython API will not work in the MakeCode editor, as this uses a different version of Python.




## EL MISMO CÓDIGO CON PYTHON DE TINKERCAD

Tinkercad https://www.tinkercad.com/ es una herramienta estupenda de simulación pues es muy realístico, igual que Maquecode, este muy orientado a la programación en bloques pero también tiene su sección de código python



Si le das la opción de bloque+código intenta muestra los bloques traducidos a código, pero si le das la opción sólo código **pierdes** la programación en bloques, Esto ya lo vimos en https://libros.catedu.es/books/programa-arduino-mediante-codigo/page/software en los párrafos escritos en naranja.

El código generado vemos que **no es compatible con Python microbit**

```
# Python code
#

def on_pulsed_p0_high():
  basic.show_icon(IconNames.Heart)
pins.on_pulsed(DigitalPin.P0, PulseValue.HIGH, on_pulsed_p0_high)

def on_pulsed_p0_low():
  basic.show_icon(IconNames.No)
pins.on_pulsed(DigitalPin.P0, PulseValue.LOW, on_pulsed_p0_low)
```

Copyright 2026

Introducción

# Introducción al Python

> Esta es **una muy breve introducción al Python** como recordatorio de algunas instrucciones si ya has utilizado este lenguaje.
>
> Si es la primera vez, te recomendamos que visites nuestro curso **PYTHON PARA TODOS**
>
> **Python for everybody** por Charles R. Severance licencia CC-BY-NCSA que empieza desde cero.

## Lenguajes, intérpretes y compiladores

Python es un lenguaje **de alto nivel** destinado a ser relativamente sencillo para que los humanos lean y escriban y para que los ordenadores lean y procesen. Otros lenguajes de alto nivel incluyen Java, C ++, PHP, Ruby, Basic, Perl, JavaScript y muchos más. El hardware real dentro de la Unidad Central de Procesamiento (CPU) no comprende ninguno de estos lenguajes de alto nivel.

La CPU entiende un idioma que llamamos **lenguaje de máquina**. El lenguaje de máquina es muy simple y francamente muy tedioso de escribir porque está representado en ceros y unos:

El lenguaje de máquina parece bastante simple en la superficie, dado que solo hay ceros y unos, pero su sintaxis es aún más compleja y mucho más compleja que Python. Muy pocos programadores escriben lenguaje de máquina. En su lugar, creamos varios traductores para permitir que los programadores escriban en lenguajes de alto nivel como Python o JavaScript y estos traductores convierten los programas al lenguaje de máquina para su ejecución real por parte de la CPU.

Estos traductores de lenguaje de programación se dividen en dos categorías generales: (1) intérpretes y (2) compiladores.

Un **intérprete** lee el código fuente del programa como está escrito por el programador, analiza el código fuente e interpreta las instrucciones sobre la marcha. Python es un intérprete y cuando ejecutamos Python de forma interactiva, podemos escribir una línea de Python (una oración) y Python la procesa de inmediato y está lista para que escribamos otra línea de Python.

```
>>> x = 6
>>> print(x)
```

```
6
>>> y = x * 7
>>> print(y)
42
>>>
```

Está en la naturaleza de un **intérprete** poder tener una conversación interactiva como se muestra arriba. A un **compilador** debemos entregarle todo el programa en un archivo, y luego ejecuta un proceso para traducir el código fuente de alto nivel al lenguaje de máquina y luego el compilador coloca el lenguaje de máquina resultante en un archivo para su posterior ejecución.

## Variables

Las variables son como cajas que puedes meter valores. Y los valores pueden ser de varios **tipos** :

- **int** si son enteros
- **float** si tienen decimales
- **binario**Deben comenzar por 0b. Por ejemplo: 0b110, 0b11
- **string** son frases, son "***cadenas***" de caracteres entre "
- **bool** Solamente hay dos literales booleanos True o False
- **lista** Se pueden declarar variables que son conjuntos por ejemplo Colores = ["verde", "rojo", "naranja" ]

Para crear una variable puedes usar cualquier palabra, x, y, z o Nombre_alumno ... pero algunas palabras no puedes usar, ver

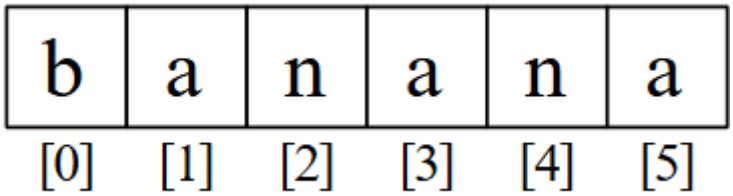
Para visualizar variables puedes usar la **instrucción print** poniendo entre paréntesis el valor o variable que quieres visualizar.

> Modifica los valores como quieras, es un **intérprete**, juega y dale al play para ver el resultado

Como puedes ver se ha introducido un operador el + que realiza la suma del valor de x original (43) y se le incrementa una unidad resultando en la impresión un 44.

## Cadenas

Cadenas son secuencias de caracteres, por ejemplo la palabra "banana"




fuente 'Python for Everybody' por Charles R. Severance

Se puede obtener su longitud con la función len, o obtener un carácter ...

## Operadores

*Este apartado de operadores es adaptado de Federico Coca Guia de Trabajo de Microbit CC-BY-SA*

Los **operadores aritméticos** se utilizan para realizar operaciones matemáticas como sumas, restas, multiplicaciones, etc.

| Operador | Descripción | Ejemplo |
|---|---|---|
| + | Suma o concatenación en textos | `5+3=8`, `"Hola" + "Mundo" = "Hola Mundo"` |
| - | Diferencia | `6-3=3` |
| * | Multiplicación | `3*3=9` |
| / | División | `6/2=3` |
| // | Parte entera de un cociente | `10//3=3` |
| % | Resto de un cociente | `10%3=1` |
| ** | Potenciación | `5**2=25` |

Los **operadores de asignación** se utilizan para asignar valores a variables.

| Operador | Descripción | Ejemplo |
|---|---|---|
| = | Asignación | `x=4`, `a = a + 1` |
| += | Suma y asignación | `x+=1` equivale a `x = x + 1` |
| -= | Diferencia y asignación | `x-=1` equivale a `x = x - 1` |
| *= | Multiplicación y asignación | `x*=3` equivale a `x = x * 3` |

| Operador | Descripción | Ejemplo |
|---|---|---|
| /= | División y asignación | `x/=3` equivale a `x = x / 3` |
| %= | Asignación de restos | `x%=3` equivale a `x = x % 3` |
| **= | Asignación de exponentes | `x**=3` equivale a `x = x ** 3` |

Los **operadores de comparación** comparan dos valores/variables y devuelven un resultado booleano: Verdadero o Falso `True` o `False`.

| Operador | Descripción | Ejemplo |
|---|---|---|
| == | Igual a | `2==3` retorna `False` |
| != | Distinto de | `2!=3` retorna `True` |
| < | Menor que | `2<3` retorna `True` |
| > | Mayor que | `2>3` retorna `False` |
| <= | Menor o igual que | `2<=3` retorna `True` |
| >= | Mayor o igual que | `2>=3` retorna `False` |

Los **operadores lógicos** se utilizan para comprobar si una expresión es Verdadera o Falsa. Se utilizan en la toma de decisiones.

| Operador | Descripción | Ejemplo |
|---|---|---|
| and | AND lógica | `a and b #True si a y b son ciertos` |
| or | OR lógica | `a or b #True si a o b son ciertos` |
| not | NOT lógica | `not a #True si el operador a es falso` |
| in | pertenencia | Devuelve True si pertenece |
| no int | no pertenencia | Devuelve True si no pertenece |
| is | identidad | Devuelve True si son iguales |
| is not | no identidad | Devuelve True si no son inguales |

Los **operadores bit a bit** o bitwise actúan sobre los operandos como si fueran cadenas de dígitos binarios. Operan bit a bit:

| Operador | Descripción | Ejemplo |
|---|---|---|

Copyright 2026

| | | |
|---|---|---|
| & | AND bit a bit | `5&6 # 101 & 110 = 110 = 4` |
| \| | OR bit a bit | `5 \| 6 # 101 \| 110 = 111 = 7` |
| ~ | NOT bit a bit | `~3 # ~011 = 100 = -4` |
| ^ | XOR bit a bit | `5^3 # 101^011 = 110 = 6` |
| << | Desplazamiento izquierda | `4<<1 # 100 << 1 = 1000 = 8` |
| >> | Desplazamiento derecha | `4 >> 1 # 100 >> 1 = 010 = 2` |

**Comentarios en Python**
Una sola línea : Escribiendo el símbolo almohadilla (#) delante del comentario.
Multilínea: Escribiendo triple comillas dobles (“””) al principio y al final del comentario.

## Entradas de teclado

Ya hemos visto salidas por pantalla con **print**, pero ahora con input puede leer variables del teclado.
Fíjate que hay que poner las líneas **x = float (x)** e **y = float(y)** para convertirlos a números decimales, en caso contrario las interpreta string y no puede multiplicar en Resultado, pero en el siguiente ejemplo **no es necesario en la variable cel** (celsius) pues se multiplica por números decimales 32.0 5.0 y 9.0

**try** y **except** son dos funciones que son *un seguro para el programador* por si el usuario en vez de teclear un número, mete un string o carácter

**La sangría es importante en Python**
La sangría se refiere a los espacios al comienzo de una línea de código. Mientras que en otros lenguajes de programación la sangría en el código es solo para facilitar la lectura, la sangría en Python es muy importante ya que se usa para indicar un bloque de código.

## Condicionales

Las instrucciones **if: else:** son las que nos permiten realizar operaciones según las condiciones puestas. *Ojo con la sangría*

**\n** es un carácter especial que significa "Salto de página"

Copyright 2026

## Bucles

- **while** ejecuta lo contenido en la sangría mientras sea verdadero la condición
- **for** ejecuta lo contenido en la sangría mientras y va recorriendo la variable dentro del rango creado

Para verlo mejor vamos a ver estos ejemplos

- EJEMPLO BUCLE WHILE
  - mientras n sea positivo va ejecutando : imprime n y lo decrementa
  - al decrementar llega un momento que deja de ser positivo y finaliza el bucle
- EJEMPLO BUCLE WHILE INFINITO
  - Es muy típico en robótica, todo el rato hace el bucle (en robótica para que lea los sensores y realice cosas en los actuadores) pero este ejemplo no esta en un robot sino en tu pc y no queremos que se quede "colgado" luego al teclear "fin" acaba gracias a la instrucción **break**
  - Fíjate que hay una instrucción **continue** para que pase a la siguiente iteración provocando que no imprime lo tecleado
- EJEMPLO BUCLE FOR FRIENDS
  - Va recorriendo la variable friend dentro del cojunto lista friends
  - como puedes ver la diferencia entre for y while es que for además recorre la variable
- EJEMPLO BUCLE FOR
  - mientras n este en el rango de 0 a 5 se ejecuta

## Funciones

No vamos a entrar en detalle, pero observa el siguiente código

- **FUNCIONES PREDEFINIDAS** Si observas, la primera línea llama a importar una librería externa, **import math** donde math es un fichero que tienen funciones predefinidas, vamos a utilizar una de ellas, la raiz cuadrada **sqrt** luego para llamar a esa función que esta definida dentro de math se hace con la instrucción **math.sqrt**
- **FUNCIONES DEFINIDAS POR TI** em este caso, se utiliza la palabra **def** para crear una función, que le vamos a pasar tres argumentos a, b y c y para finalizar la función usamos **return** para devolver el valor que queremos obtener

# Para saber más de Python

| | |
|---|---|
| CURSO PYTHON FOR EVERYBODY en español | ver |
| Curso completo de Python 222pag pdf (*) | Descargar |

| | |
|---|---|
| Curso completo de Python 422pag (*) | Descargar |
| Curso completo de Python desde 0 (*) | Ver |
| Curso de Python desde 0 (*) | Ver |
| Manual de referencia Python (*) | Ver |
| Programación en Python (*) | Ver |
| Trabajando con ficheros en Python (*) | Ver |
| Programación orientada a objeto en Python (*) | Ver |
| un manual para aquellos usuarios con previos conocimientos de Python, como la programación modular y orientada a objetos.<br>También algunos conocimientos de las librerías **tkinter** (Para crear interfaces gráficos y **SQlite3** (para gestionar bases de datos). (*) | Descargar |

(*) Agradecimientos a Pere Manel http://peremanelv.com

Introducción

# Micropython de microbit

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

## API: El módulo microbit

Todo lo necesario para interactuar con el hardware de la micro:bit está en el módulo *microbit* y se recomienda su uso escribiendo al principio del programa:

```
from microbit import *
```

Las funciones disponibles directamente son:

```
sleep(ms) #1
running_time() #2
temperature() #3
scale(valor_a_convertir, from_=(min, max), to=(min, max)) #4
panic(error_code) #5
reset() #6
set_volume(valor) #7 (V2)
'''
1 Esperar el número de milisegundos indicado
2 Devuelve el tiempo en ms desde la última vez que se encendió la micro:bit
3 Devuelve la temperatura en Celcius
4 Convierte un número de una escala de valores a otra
5 La micro:bit entra en modo pánico por falta de memoria y se dibuja una
cara triste en la pantalla. El valor de error_code puede ser cualquier entero.
6 Resetea la micro:bit
7 Estable el volumen de salida con un *valor* entre 0 y 255
'''
```

# Estructuras de datos en Python

## Las listas (list)

Se trata de un tipo de dato que permite almacenar series de datos de cualquier tipo bajo su estructura. Se suelen asociar a las matrices o arrays de otros lenguajes de programación.

En Python las listas son muy versatiles permitiendo almacenar un conjunto arbitrario de datos. Es decir, podemos guardar en ellas lo que sea.

Una lista se crea con `[]` y sus elementos se separan por comas. Una gran ventaja es que pueden tener datos de diferentes tipos.

```
lista = [1, "Hola", 3.141592, [1 , 2, 3], Image.HAPPY]
```

Las de principales propiedades de las listas:

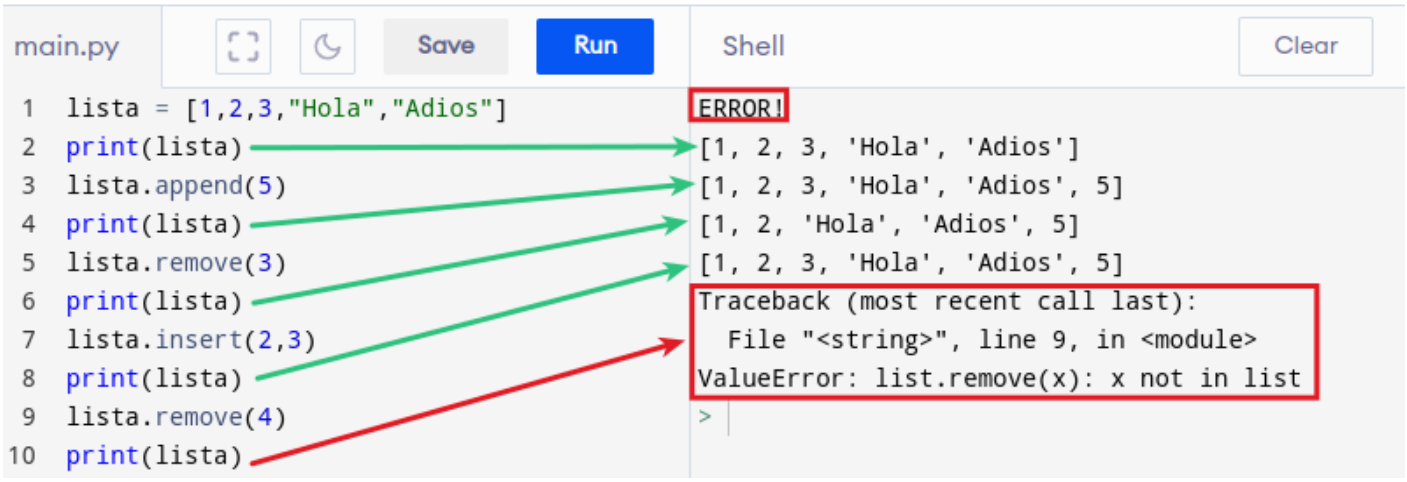
- Son ordenadas, mantienen el orden en el que han sido definidas
- Pueden ser formadas por tipos arbitrarios de datos
- Pueden ser indexadas con [i]
- Se pueden anidar, es decir, meter una lista dentro de otra
- Son mutables, ya que sus elementos pueden ser modificados
- Son dinámicas, ya que se pueden añadir o eliminar elementos

Hay dos métodos aplicables:

- **`append`**. Permite agregar elementos a la lista.
- **`remove`**. Elimina elementos de la lista.
- **`insert(pos,elem)`**. Inserta el elemento `elem` en la posición `pos` indicada.

En el ejemplo vemos el funcionamiento.

Copyright 2026




*Federico Coca Guia de Trabajo de Microbit CC-BY-SA*

Con estos conocimientos tendremos suficiente para hacer lo que pretendemos, que no es otra cosa que animar imágenes.

## Las tuplas (tuple)

Son muy similares a las listas con una diferencia principal con las mismas y es que las tuplas no pueden ser modificadas directamente, lo que implica que no dispone de los métodos vistos para listas. Una tupla permite tener agrupados un número inmutable de elementos.

Una tupla se crea con `()` y sus elementos se separan por comas.

```
tupla = (1, 2, 3)
```

Principales propiedades:

- Se pueden declarar sin usar los paréntesis, pero no se recomienda. No usarlos puede llevarnos a ambigüedades del tipo print(1, 2, 3) y print((1, 2, 3)).
- Si la tupla tiene un solo elemento esta debe finalizar con coma.
- Se pueden anidar tuplas, por ejemplo `tupla2 = tupla1, 4, 5, 6, 7`.
- Se pueden declarar tuplas vacias, por ejemplo `tupla3 = ()`.
- Las tuplas son *iterables* por lo que sus elementos pueden ser accesados mediante la notación de índice del elemento entre corchetes. Si se quiere acceder a un rango de indices se separan por ":" ambos índices.
- Es posible convertir listas en tuplas simplemente poniendo la lista dentro de los paréntesis de la tupla, por ejemplo, `tupla_lista = ([1, "Hola", 3.141592, [1 , 2, 3], Image.HAPPY])`

A continuación vemos un ejemplo.




*Federico Coca Guia de Trabajo de Microbit CC-BY-SA*

## Diccionarios (dict)

Estas estructuras contienen la colección de elementos con la forma `clave:valor` separados por comas y encerrados entre `{}`. Las claves son objetos inmutables y los valores pueden ser de cualquier tipo. Sus principales características son:

- En lugar de por índice como en listas y tuplas, en diccionarios se acceder al valor por su clave.
- Permiten eliminar cualquier entrada.
- Al igual que las listas, el diccionario permite modificar los valores.
- El método `dicc.get()` accede a un valor por la clave del mismo.
- El método `dicc.items()` devuelve una lista de tuplas `clave:valor`.
- El método `dicc.keys()` devuelve una lista de las claves.
- El método `dicc.values()` devuelve una lista de los valores.
- El método `dicc.update()` añade elemento `clave:valor` al diccionario.
- El método `del dicc` borra el par `clave:valor`.
- El método `dicc.pop()` borra el par `clave:valor`.

A continuación vemos un ejemplo




*Federico Coca Guia de Trabajo de Microbit CC-BY-SA*

# Bucles

Los **Bucles** son un tipo de estructura de control muy útil cuando queremos repetir un bloque de código varias veces. En Python existen dos tipos de bloques, el bucle ***for*** para contar la cantidad de veces que se ejecuta un bloque de código, y el bucle ***while*** que realiza la acción hasta que la condición especificada no sea cierta.

- While
- for
- Bucle for decontando
- Sentencias break y continue

## While

### La sintaxis de while es la siguiente:

```
while condicion:
    bloque de codigo
```

donde "*condicion*", que se evalúa en cada iteración, puede ser cualquier expresión realizado con operadores condicionales que devuelva como resultado un valor True o False. Mientra que "bloque de codigo" es el conjunto de instrucciones que se estarán ejecutando mientras la condición sea verdadera (True o '1'). Es lo mismo poner `while true:` que poner `while 1:`.

Para recorrer los bucles se utilizan variables que forman parte de la condición, estableciendose en esta lo que deben cumplir.

Un ejemplo sencillo podría ser el siguiente, controlar el riego de una planta en función del valor de la humedad de la tierra en la que está.

```
from microbit import *

while (humedad() < 45):
    display.scroll(Image.SAD)
    sleep(1000)

display.show(Image.HAPPY)
```

que hará que si la humedad baja por debajo de 45 se muestre una carita triste indicando que hay que regar y si es mayor mostrará una carita feliz. Evidentemente hay que resolver el tema de como obtener la humedad, pero esa es una historia que veremos mas adelante.

El bucle `while` puede tener de manera opcional un bloque `else` cuyas sentencias se ejecutan cuando se han realizado todas las iteraciones del bucle. Un ejemplo lo vemos a continuación:

```
cuenta = 0
while cuenta < 5:
    print("Iteración del bucle")
    cuenta = cuenta + 1
else:
    print("bucle finalizado")
```

## for

Son también bucles pero su acción está dirigida a contar el número de veces que ocurre algo o realizar una acción un determinado número de veces. Es especialmente útil para recorrer los datos de una lista, tupla o diccionario.

La sintaxis de este tipo de bucles en Python es:

```
for variable in secuencia:
    declaracion
```

Copyright 2026

Siendo "variable" la variable que se va a recorrer en el bucle de forma que cuando se alcance el valor establecido se sale del bucle.

La variable puede ser una cadena, un rango de valores que se expresa con `range(n)`, siendo n el número de valores del rango que se inicia en 0 y que pueden ser iterados con una variable. Mas ampliamente, la sintaxis de `range()` es `range(start, stop, step)` siendo `start` y `stop` opcionales.

Veamos un primer ejemplo en el que vamos a utilizar un bucle para encender uno a uno por filas los LEDs de la primera y última columna.

```
from microbit import *
for var in range(5): # var puede tomar 5 valores, del 0 al 4
    display.set_pixel(0, var, 9) # Se ilumina el LED de la fila 0 y el valor de var para
columna
    sleep(300)
    display.set_pixel(4, var, 9) # Se ilumina el LED de la fila 4 y el valor de var para
columna
    sleep(300)
```

Los bucles se pueden anidar, es decir se puede crear un bucle dentro de otro del mismo o diferente tipo, de forma que por cada iteración del bucle mas externo se tienen que producir todas las iteraciones del bucle mas interno. Veamos como ejemplo el de encender todos los LEDs de uno en uno, de izquierda a derecha, utilizando el valor de sus coordenadas x,y. El programa sería:

```
from microbit import *

display.clear()
for y in range(0, 5): # Valor de columna
    for x in range(0, 5): # Valor de fila
        display.set_pixel(x, y, 9) # Encender LED x,y
        sleep(100)
```

En la animación siguiente vemos el programa en funcionamiento.

ejem_dicc.png

*Federico Coca Guia de Trabajo de Microbit CC-BY-SA*

El bucle `for` puede tener de manera opcional un bloque `else` cuyas sentencias se ejecutan cuando se han realizado todas las iteraciones del bucle. Un ejemplo lo vemos a continuación:

```
for var in range(5):
    print(var)
else:
    print("bucle finalizado")
```

## Bucle for decontando

Se trata del mismo bucle `for` pero ahora la cuenta la realizamos hacia atrás. Hay dos formas sencillas de hacerlo:

- Utilizando la función `range()`. Si queremos darle un enfoque Pythonic simplemente configuramos los argumentos de la función de manera que se indique el principio, el final y el incremento, que será logicamente negativo.

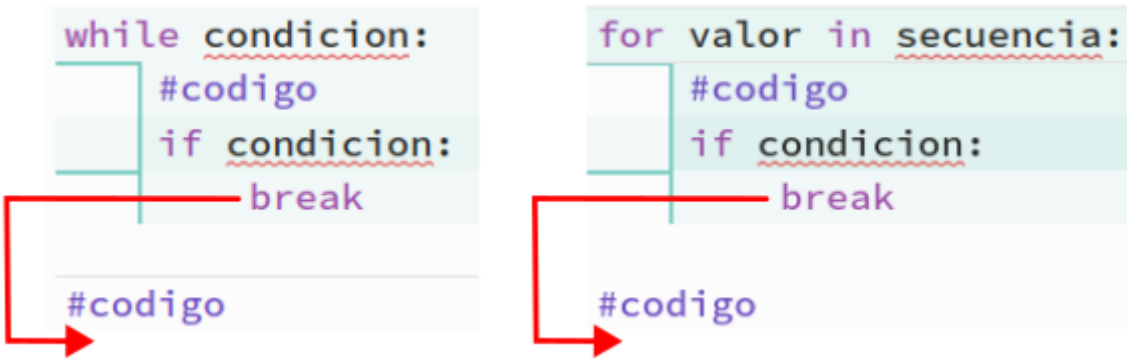
```
for i in range(20, 0, -2): #imprimere 20, 18, 16, ... 0
```

- Utilizando la función `reversed()`. Es una función incorporada en la que hay que indicar como primer argumento el final de la cuenta, como segundo el principio, teniendo en cuenta que se omite, y como tercero el decremento si es ditintos de 1, pero se especifica en módulo. Se utiliza así:

```
for i in reversed(range(0,21,2)): #imprimere 20, 18, 16, ... 0
```

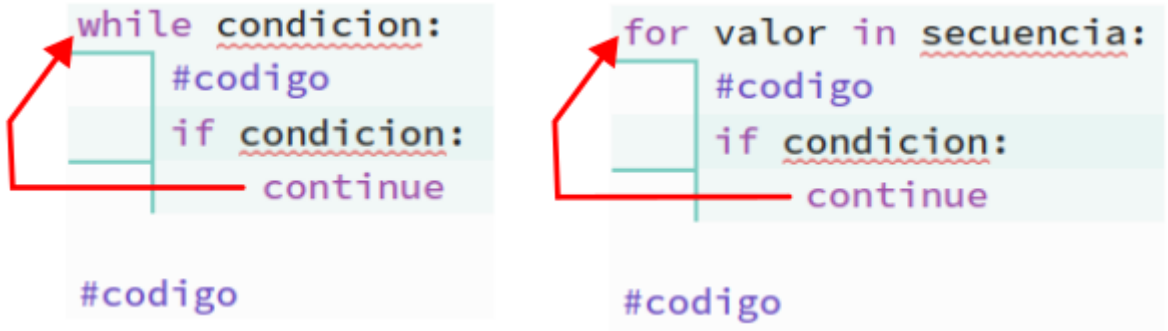
## Sentencias `break` y `continue`

La sentencia `break` se utiliza para terminar un bucle de forma inmediata al ser encontrada. En la imagen vemos la sintaxis de la sentencia `break` y su funcionamiento.



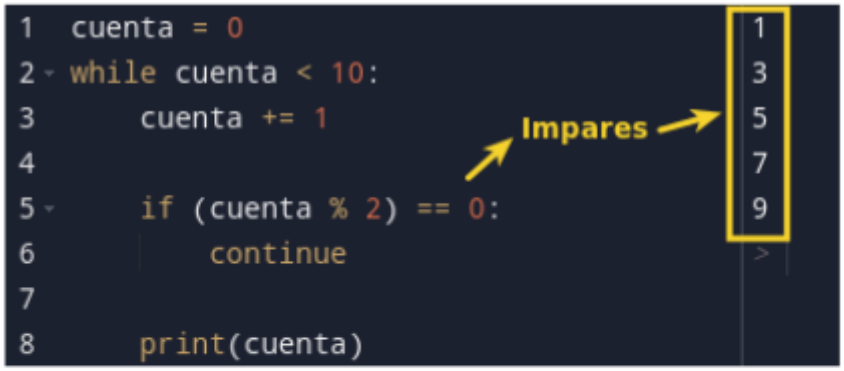

*Federico Coca Guia de Trabajo de Microbit CC-BY-SA*

La sentencia `continue` se utiliza para saltar la iteración actual del bucle y el flujo de control del programa pasa a la siguiente iteración. En la imagen vemos la sintaxis de la sentencia `continue` y su funcionamiento.




*Federico Coca Guia de Trabajo de Microbit CC-BY-SA*

En la figura siguiente vemos dos ejemplos de esta sentencia







*Federico Coca Guia de Trabajo de Microbit CC-BY-SA*

## Sentencia condicional `if...else`

En Python hay tres formas de declaración de `if...else`

> 1. Declaración `if`
> 2. Declaración `if...else`
> 3. Declaración `if...elif...else`

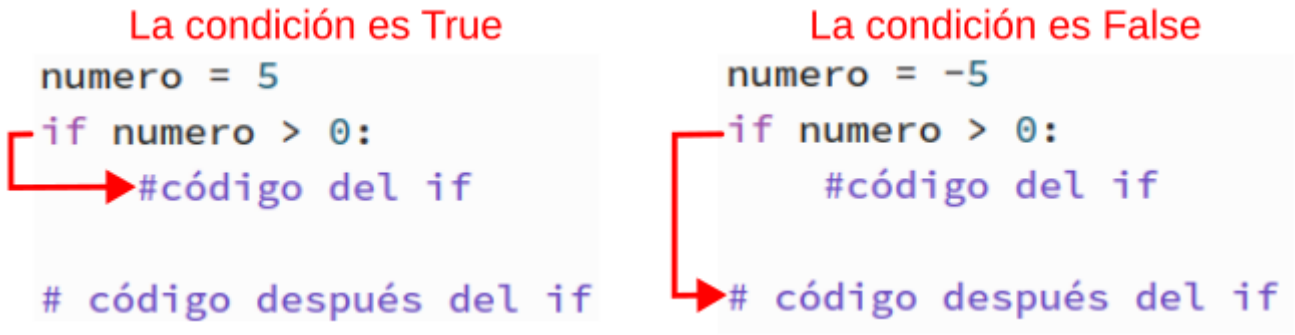
1. Declaración `if`. La sintaxix de esta declaración en Python tiene la forma siguiente:

```
if condicion:
    # Cuerpo de la sentencia if


# Código después del if
```

Si el resultado de evaluar la condición es cierto (True o 1), el código en "Cuerpo de la sentencia if" y lo estará haciendo mientras se cumpla la condición.

En el momento que la condición sea evaluada como falsa (False o 0) el código en "Cuerpo de la sentencia if" se omite y continua la ejecución del programa por "Código después del if". En la figura siguiente vemos la explicación de forma gráfica.




*Funcionamiento de la sentencia if*

*Federico Coca Guia de Trabajo de Microbit CC-BY-SA*

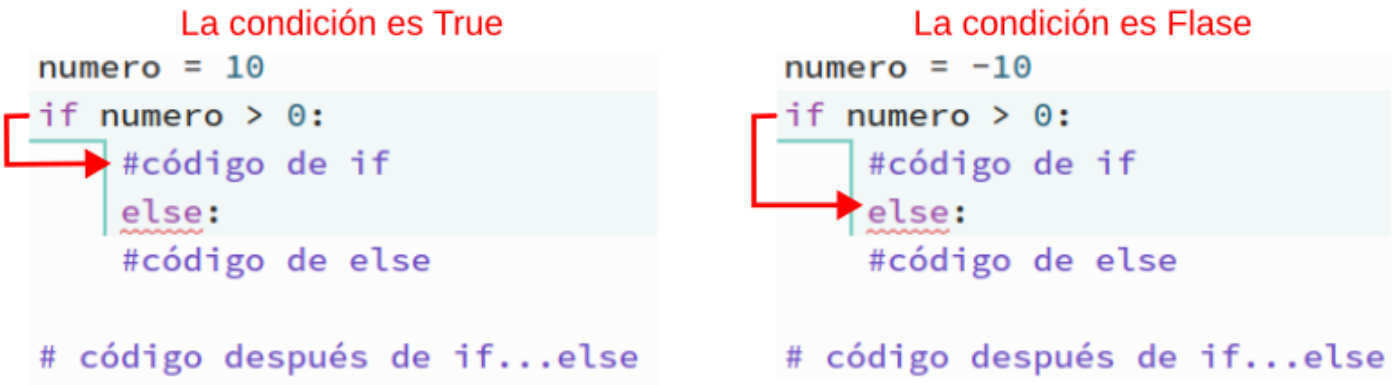
1. Declaración `if...else`. Una sentencia `if` puede tener de manera opcional una clausula `else`. La sintaxix de esta declaración en Python tiene la forma siguiente:

```
if condicion:
    # Bloque de sentencias si condicion es True


    else:
    # Bloque de sentencias si condicion es False
```

La sentencia se evalúa de la siguiente forma: Si `condición` es `True` se ejecuta el código dentro del `if` y el código dentro del `else` se omite. Si `condición` es `False` se ejecuta el código dentro del `else` y el código dentro del `if` se omite. Cuando finaliza bien la parte del `if` o bien la del `else` el programa continua con la siguiente sentencia.

En la figura siguiente vemos la explicación de forma gráfica.




*Funcionamiento de la sentencia `if...else` Federico Coca Guia de Trabajo de Microbit CC-BY-SA*

1. Declaración `if...elif...else`. La sentencia `if...else` se utiliza para ejecutar un bloque de código entre dos alternativas posibles. Sin embargo, si necesitamos elegir entre más

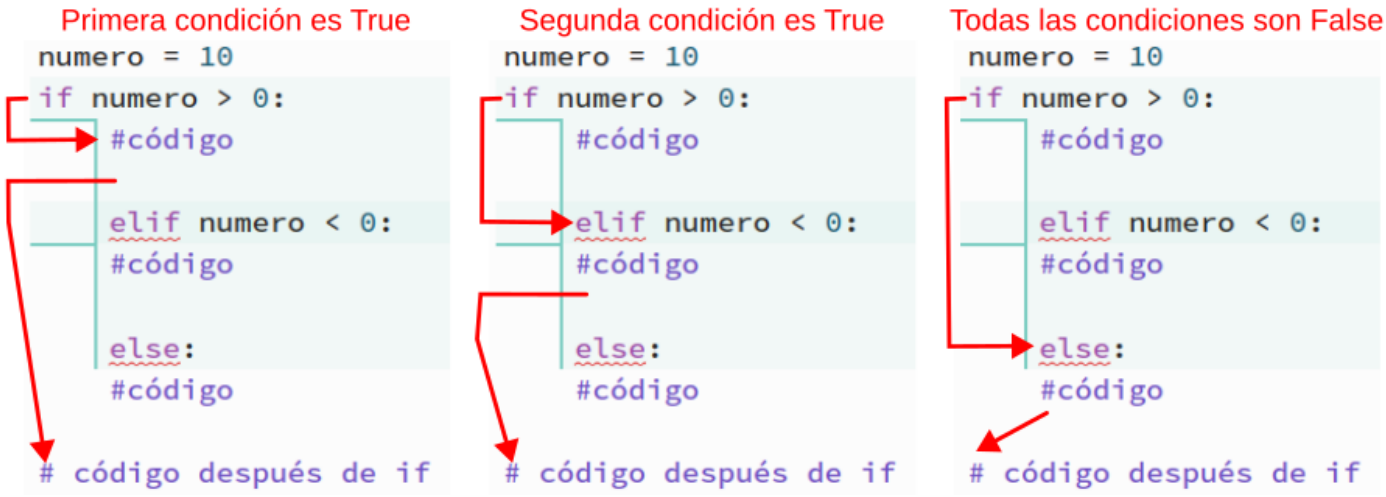
de dos alternativas, entonces utilizamos la sentencia `if...elif...else`. La sintaxis de la sentencia `if...elif...else` es:

```
if condicion_1:
    # Bloque 1
elif condicion_2:
    #Bloque 2

    else:
    # Bloque 3
```

Se evalúa así: Si `condicion_1` es `True`, se ejecuta Bloque 1. Si `condicion_1` es `False`, se evalúa `condicion_2`. Si `condicion_2` es `True`, se ejecuta Bloque 2. Si `condicion_2` es `False`, se ejecuta Bloque 3.

En la figura siguiente vemos la explicación de forma gráfica.




*Federico Coca Guia de Trabajo de Microbit CC-BY-SA*

## Funciones en Python

En esta sección vamos a dar solamente una breve introducción a lo que son las funciones y los módulos en Python para estudiar dos funciones concretas definidas en MicroPhyton para micro:bit.

Una función es un bloque de código que realiza una tarea específica.

Supongamos que necesitas crear un programa para crear un círculo y colorearlo. Puedes crear dos funciones para resolver este problema:

- crear una función de círculo
- crear una función de color

Dividir un problema complejo en trozos más pequeños hace que nuestro programa sea fácil de entender y reutilizar.

Existen dos tipos de funciones en Python:

- **Standard library functions (Funciones de biblioteca estándar)**. Son funciones incorporadas en Python que están disponibles para su uso.
- **User-defined functions (Funciones definidas por el usuario)**. Podemos crear nuestras propias funciones para que cumplan con nuestros requisitos.

La sintaxis de una función es la siguiente:

```
def nombre_funcion(argumentos):
    #Cuerpo de la función


    return
```

Donde,

- `def` es la palabra reservada para declarar una función
- `nombre_funcion` es el nombre que le damos a la función
- `argumentos` es el valor o valores pasados a la función
- `return` retorna un valor desde la función. Es opcional

Veamos un ejemplo sencillo que no manda parametros ni retorna nada.

```
def saludo():
    print("Hola Mundo!")

saludo() #Llama a la función
print("Programa")
saludo()
print("Otra vez programa")
```

*Va a generar como salida la cadena "Hola Mundo!" seguida de la cadena "Programa" seguida otra vez de "Hola Mundo!" y finaliza con "Otra vez programa".*

Cuando se llama a la función, el control del programa pasa a la definición de la función, se ejecuta todo el código dentro de la función y despés el control del programa salta a la siguiente sentencia después de la llamada a la función.

Como ya se ha mencionado, una función también puede tener argumentos. Un argumento es un valor aceptado por una función. Cuando creamos una función con argumentos necesitamos pasar los correspondientes valores cuando la llamamos.

De forma genérica una función con argumentos tiene la siguiente sintaxis:

```
def funcion(arg1, arg2, ar3,...):
    #Código

#Llamada a la función
funcion(valor1, valor2, valor3, ...)
#Código
```

Cuando llamamos a la función le pasamos los valores correspondiendo valor1 a arg1, valor2 a arg2 y así sucesivamente.

La llamada a la función se puede hacer mencionando el nombre del argumento, que es lo que se conoce como 'argumentos con nombre', siendo el código totalmente equivalente al anterior.

```
funcion(arg1=valor1, arg2=valor2, arg3=valor3, ...)
```

Una función Python puede o no devolver un valor. Si queremos que nuestra función devuelva algún valor a una llamada realizada a función, utilizamos la sentencia `return`.

En el ejemplo siguiente se llama a la función cuatro veces con valores diferentes.

```
def cal_potencia(base, exponente):
    resultado = base ** exponente
    return resultado

#Llamadas a la función
print('Potencia =', cal_potencia(2,8))
print('Potencia =', cal_potencia(3,3))
```

```
print('Potencia =', cal_potencia(4,5))
print('Potencia =', cal_potencia(9,6))
```

El resultado es:

```
Potencia = 256
Potencia = 27
Potencia = 1024
Potencia = 531441
```

En Python, las funciones de la biblioteca estándar son las funciones incorporadas que se pueden utilizar directamente en nuestro programa. Por ejemplo,

- `print()`, imprime la cadena entre comillas
- `sqrt()`, devuelve la raíz cuadrada de un número
- `pow()`, devuelve la potencia de un número

Estas funciones están definidas dentro de un módulo. Y, para utilizarlas debemos incluir dicho módulo en nuestro programa. Por ejemplo, `sqrt()` y `pow()` están definidos en el módulo `math`. Para usar las funciones podemos hacer como en el ejemplo siguiente:

```
import math #Carga el módulo math

raiz = math.sqrt(25)
print("La raiz cuadrada de 25 es ", raiz)

potencia = pow(2, 8)
print("2^8 =", potencia)
```

En el ejemplo la variable raiz contendrá el cálculo de la raiz cuadrada y se define por defecto como variable real o decimal y potencia contendrá el resultado de elevar a 8 el número 2. Los resultados obtenidos son:

```
La raiz cuadrada de 25 es 5.0
2^8 = 256
```

Las principales ventajas de utilizar funciones son:

- **Código reutilizable**. Podemos llamar a la misma función tantas veces en nuestro programa como necesitemos, lo que hace que nuestro código sea reutilizable.
- **Código legible**. Las funciones nos ayudan a dividir nuestro código en trozos para que nuestro programa sea mas legible y fácil de entender.

## Módulos en Python

A medida que nuestro programa crece, puede contener muchas líneas de código. En lugar de poner todo en un solo archivo, podemos utilizar módulos para separar por funcionalidad los códigos en varios archivos. Esto hace que nuestro código quede organizado y sea más fácil de mantener.

Un módulo es un archivo que contiene código para realizar una tarea específica. Un módulo puede contener variables, funciones, clases, etc. Veamos un ejemplo, vamos a crear un módulo escribiendo algo como lo siguiente:

```
#Definición del módulo suma


def sumar(a, b):


    resultado = a + b
    return resultado
```

Guardamos este programa en un archivo, por ejemplo `modulo_sumar.py` y tendremos definida una función de nombre `sumar` en ese módulo. La función recibe dos valores y devuelve la suma.

Cuando, en un programa diferente, queramos sumar dos números podemos importar la definición creada utilizando la palabra reservada `import`. Para acceder a la función definida en el módulo tenemos que utilizar el operador `.` (punto). Se parece mucho a que el módulo es una clase y la función una instancia de esa clase.

```
# Programa de sumas
import modulo_sumar


modulo_sumar.sumar(4, 5) #devolverá 9
```

Python tiene mas de 200 módulos estándar que pueden ser importados de la misma manera que importamos los módulos definidos por nosotros. En la documentación de Python en español encontramos la referencia a La biblioteca estándar de Python.

## Números aleatorios

Este módulo está basado en el módulo `random` de la librería estándar de **Python**. Contiene funciones para generar comportamientos aleatorios.

Para acceder a este módulo es necesario:

```
import random
```

Vamos a ver sus funciones a continuación.

- **`.getrandbits(n)`**. Retorna un entero con "n" bits aleatorios. La función generadora devuelve como máximo 30 bits, por lo tanto "n" tiene que estar comprendido entre 1 y 30.

```
random.getrandbits(n)
```

* ***`.seed(n)`***. *Inicializa el generador de números aleatorios con un número entero conocido "n". Esto le proporcionará una aleatoriedad determinista reproducible a partir de un estado inicial dado (n).*

```
random.seed(n)
```

- **`.randint(a, b)`**. Devuelve un entero aleatorio **N** tal que $a \leq N \leq b$ $a \leq N \leq b$.

```
random.randint(a, b)
```

- **`.randrange(stop)`**. Devuelve un número entero seleccionado aleatoriamente entre cero y stop, que no está incluido.

```
random.randrange(stop)
```

- **`.randrange(start, stop)`**. Devuelve un número entero seleccionado aleatoriamente comprendido entre start y stop. El límite stop no está incluido.

```
random.randrange(start, stop)
```

- **`.randrange(start, stop, step)`**. Devuelve un número entero aleatorio entre start y stop separando los valores posibles entre si la distancia establecida por step. Por ejemplo

`randrange(3, 30, 5)` devolverá un valor aleatorio de los siguientes posibles: 3, 8, 13, 18, 23, 28.

```
random.randrange(start, stop, step)
```

- **`.choice(secuencia)`**. Devuelve un elemento aleatorio de 'secuencia' que no puede estar vacía. Si 'secuencia' está vacía, genera in `IndexError`.

```
random.choice(secuencia)
```

- **`.random()`**. Devuelve un número aleatorio en coma flotante en el rango [0.0, 1.0).

```
random.random()
```

- **`.uniform(a, b)`**. Devuelve un número aleatorio de coma flotante **N** tal que a≤N≤ba≤N≤b para a≤ba≤b y b≤N≤ab≤N≤a para b<ab<a.

```
random.uniform(a, b)
```

En la imagen vemos ejemplos ejecutados en la shell.

Copyright 2026

main.py | Save | Run | Shell | Clear

```
1  import random
2  r1 = random.getrandbits(15)
3  r2 = random.randint(10, 40)
4  r3 = random.randrange(3)
5  r4 = random.randrange(3, 5)
6  r5 = random.randrange(3, 30, 5)
7  frutas = ['pera', 'manzana', 'plátano',
       'ciruelas', 'sandia', 'melon']
8  r6 = random.choice(frutas)
9  r7 = random.random()
10 r8 = random.uniform(5, 15)
11 r9 = random.uniform(30, 20)
12 print('.getrandbits(n):', r1)
13 print('.randint(a, b):',r2)
14 print('.randrange(stop):', r3)
15 print('.randrange(start, stop):' ,r4)
16 print('.randrange(start, stop, step):'
       ,r5)
17 print('.choice(secuencia):' ,r6)
18 print('.random():' ,r7)
19 print('.uniform(a, b), (a<=b):' ,r8)
20 print('.uniform(a, b), (b<a):' ,r9)
```

```
.getrandbits(n): 25299
.randint(a, b): 17
.randrange(stop): 1
.randrange(start, stop): 4
.randrange(start, stop, step): 8
.choice(secuencia): ciruelas
.random(): 0.7110020050094659
.uniform(a, b), (a<=b): 11.866913706801071
.uniform(a, b), (b<a): 28.918703017544235
>
```

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

Introducción

# Hoja de Ruta

¿Dónde encajaría usar Microbit con Python en las etapas educativas?  ¿Comparativa con otros robots?

Guía orientativa

https://docs.google.com/presentation/d/e/2PACX-1vQHiZvv1cGHet7eXVy-QcECY4Lj0k0l7ntDi8MevRWHQX-9myA0bfR5IofMeuGZkWD0Hw-Ob-MGoco_/embed?start=trueloop=true&delayms=3000

> Tenemos un **grupo Telegram Robótica Educativa en Aragón**, https://t.me/roboticaeducativaaragon



# Empezando

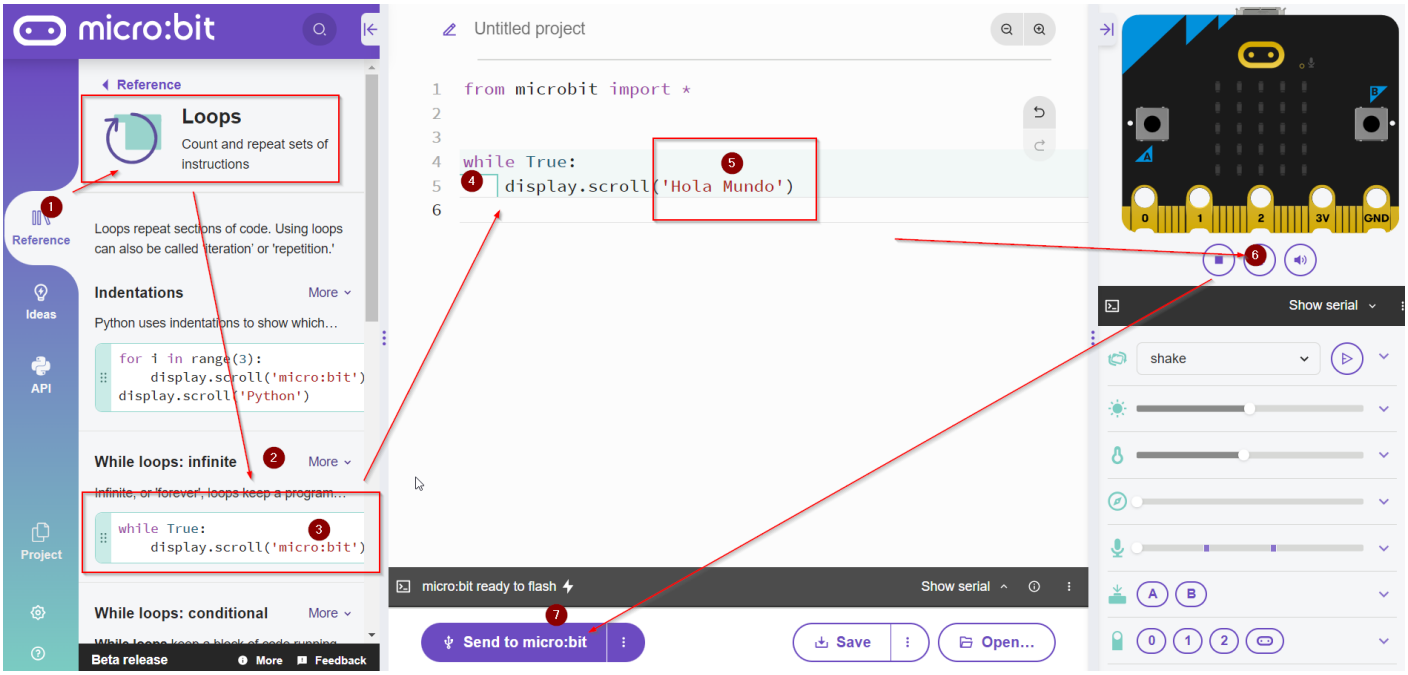
Copyright 2026

Empezando

# Hola Mundo

No hay mejor manera para empezar que este sencillo programa

1. Entramos en https://python.microbit.org/
2. Nos vamos a la pestaña de **Reference - Loops** y arrastramos el código de **While loops infinite**
3. Cambiamos el texto por "**Hola Mundo**" y lo simulamos en el microbit virtual de la izquierda
4. ¿Lo ha hecho bien? pues conecta tu microbit a tu ordenador, y **Sent to Microbit** te saldrá un diálogo pidiendo vincular tu microbit, acepta y ya esta !!!!




El código Pytho que ha subido a Microbit es el siguiente, la primera línea importa las librerías para manejar microbit, la segunda es el bucle While y al poner la condición true, se ejecutará siempre, y la instrucción que ejecuta es display.scroll donde visualiza en forma de marquesina el texto que pongamos, también puede ser un número.

```
from microbit import *



while True:
    display.scroll('Hola Mundo')
```

https://www.youtube.com/embed/sCpnOzJnutY

Copyright 2026

Empezando

# Imágenes

> *Extraído de Federico Coca Guia de Trabajo de Microbit CC-BY-SA*

## API: *Display*

Control de la matriz de 5x5 LEDs que en micro:bit se conoce como pantalla. Los métodos de la clase son:

```
display.get_pixel(x, y) #1
display.set_pixel(x, y, val) #2
display.clear() #3
display.show(image, delay=0, wait=True, loop=False, clear=False) #4
display.scroll(string, delay=400) #5
'''
1 Obtiene el brillo [0 (apagado) a 9 (máx))] del pixel (x,y)
2 Establece el brillo [0 (apagado) a 9 (máx))] del pixel (x,y)
3 Borra (apaga) la pantalla
4 Muestra la imagen
5 Desplaza una cadena por la pantalla a la velocidad en ms del *delay*
'''
```

En ambos casos de la API existen otras muchas opciones no incluidas. La funcionalidad de autocompletar nos ayudará para no tener que recordar la sintaxis y conocer las que no aparece aquí. En la animación siguiente vemos un ejemplo de ambos casos.

logicos.png

*Federico Coca Guia de Trabajo de Microbit CC-BY-SA*

## Imágenes prediseñadas

MicroPython nos ofrece muchas imágenes integradas para mostrar por pantalla y podemos crear efectos interesantes.

```
from microbit import *

while True:
  display.show(Image.HEART)
  sleep(500)
  display.show(Image.HEART_SMALL)
  sleep(500)
```

Las opciones son múltiples:

```
Image.HEART
Image.HEART_SMALL
Image.HAPPY
Image.SMILE
Image.SAD
Image.CONFUSED
Image.ANGRY
Image.ASLEEP
Image.SURPRISED
Image.SILLY
Image.FABULOUS
Image.MEH
Image.YES
Image.NO
Image.CLOCK12,Image.CLOCK11,Image.CLOCK10,Image.CLOCK9,Image.CLOCK8,Image.CLOCK7,Image.CLOCK6,
Image.CLOCK5,Image.CLOCK4,Image.CLOCK3,Image.CLOCK2,Image.CLOCK1
Image.ARROW_N,Image.ARROW_NE,Image.ARROW_E,Image.ARROW_SE,Image.ARROW_S,Image.ARROW_SW,Image.A
RROW_W,Image.ARROW_NW
Image.TRIANGLE
Image.TRIANGLE_LEFT
Image.CHESSBOARD
Image.DIAMOND
Image.DIAMOND_SMALL
Image.SQUARE
```

```
Image.SQUARE_SMALL
Image.RABBIT
Image.COW
Image.MUSIC_CROTCHET
Image.MUSIC_QUAVER
Image.MUSIC_QUAVERS
Image.PITCHFORK
Image.PACMAN
Image.TARGET
Image.TSHIRT
Image.ROLLERSKATE
Image.DUCK
Image.HOUSE
Image.TORTOISE
Image.BUTTERFLY
Image.STICKFIGURE
Image.GHOST
Image.SWORD
Image.GIRAFFE
Image.SKULL
Image.UMBRELLA
Image.SNAKE Image.ALL_CLOCKS Image.ALL_ARROW
```

## Imágenes DIY

Es perfectamente posible crear nuestras propias imágenes configurando cada Pixel o LED de la pantalla. También es posible crear animaciones con imágenes.

Crear nuestras propias imágenes va a resultar una tarea sencilla cuando conozcamos la información para hacerlo. Cada pixel (LED) de la pantalla se puede configurar con diez valores que pueden tomar un valor entre 0 (cero) y 9 (nueve). Cuando le damos valor 0 (cero) es decirle literalmente que el brillo es nulo y sin embargo cuando le damos el valor 9 (nueve) lo ponemos al máximo de brillo posible. Podemos jugar con todos los valores intermedios para crear niveles de brillo.

La forma mas sencilla de definir una imagen consiste en utilizar la *clase microbit.Image* para crearla a partir de una cadena o string que devuelva el pictograma. Es decir utilizando el comando *Image(string)* teniendo que constar de dígitos con los valores 0 a 9 indicados. Para verlo rápidamente hacemos el ejemplos de dibujar una X en relieve asignándola a una variable.

```
mi_imagen_X = Image("90009:"
                    "06060:"
                    "00300:"
                    "06060:"
                    "90009")
```

Los dos puntos indican un salto de línea por lo que se puede usar el ASCII no imprimible "\n" que es precisamente eso, un salto de línea.

```
mi_imagen_X = Image("90009\n"
                    "06060\n"
                    "00300\n"
                    "06060\n"
                    "90009")
```

Los valores de brillo dan la sensación de relieve de profundidas a la X.

En cualquier caso esto no se escribe normalmente así, salvo para hacer mas o menos un gráfico del pixelado, sino en una sola línea.

```
mi_imagen_X = Image("90009\n06060\n00300\n06060\n90009")
```

Ahora parece mas elegante utilizar los dos puntos como indicador de salto de línea.

```
mi_imagen_X = Image("90009:06060:00300:06060:90009")
```

En la imagen vemos el resultado de lo explicado.

Copyright 2026

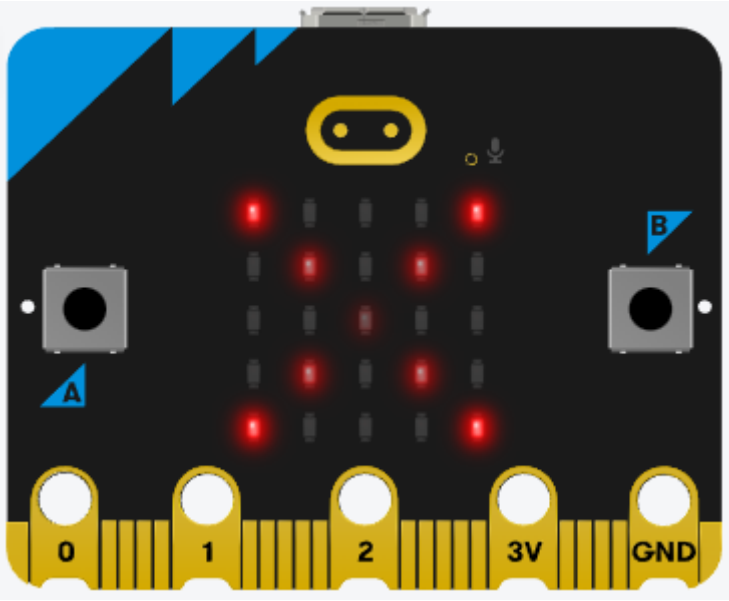



*Imagen de una X en relieve*

*Federico Coca Guia de Trabajo de Microbit CC-BY-SA*

Este es el código creado:

```
from microbit import *
"""mi_imagen_X = Image("90009\n"
                       "06060\n"
                       "00300\n"
                       "06060\n"
                       "90009")"""
#mi_imagen_X = Image("90009\n06060\n00300\n06060\n90009")
mi_imagen_X = Image("90009:06060:00300:06060:90009")
display.show(mi_imagen_X)
```

## Animar imágenes

En micro:bit Python ya disponemos de un par de listas de imágenes incorporadas que se llaman

```
Image.ALL_Clocks
Image.ALL_ARROWS
```

Estas dos ordenes hacen que MicroPython entienda que necesita mostrar cada imagen de la lista, una tras otra.

Cuando queremos mostrar en la pantalla una imagen se nos muestra la siguiente ayuda contextual:




*Ayuda contextual para display.show()*

*Federico Coca Guia de Trabajo de Microbit CC-BY-SA*

donde nos indica claramente que **image** puede ser una cadena, un número, una imagen o una lista de imágenes. Además aparecen las opciones que podemos configurar.

Con esta información crear un "reloj" que esté continuamente marcando cada hora es bastante sencillo, basta con poner el siguiente código y darle a simular.

```
# Imports go at the top
from microbit import *
display.show(Image.ALL_CLOCKS, delay=400, loop=True)
```

En la animación vemos el funcionamiento de este "reloj".

ayuda_disp_show.png

*Federico Coca Guia de Trabajo de Microbit CC-BY-SA*

Si cambiamos el reloj por las flechas veremos como van rotando flechas en ángulos de 45 grados.

*ayuda_disp_show.png*

*Federico Coca Guia de Trabajo de Microbit CC-BY-SA*

Para animar nuestras propias imágenes tendremos que crear cada una sobre un lienzo de 5x5 pixeles y establecer las diferencias para crear la animación. Podemos crear tantas imágenes como creamos oportuno. Creamos una lista con todas las imágenes en el orden que se tienen que reproducir y ya podemos mostrar nuestra lista en la pantalla.

En la animación siguiente vemos un efecto creado de esta forma.

ayuda_disp_show.png

*Federico Coca Guia de Trabajo de Microbit CC-BY-SA*

Este es el código para crear la animación.

```
# Imports go at the top
from microbit import *
display.clear()
cor1=Image("90000:90000:90000:90000:90000")
cor2=Image("79000:79000:79000:79000:79000")
cor3=Image("57900:57900:57900:57900:57900")
cor4=Image("35790:35790:35790:35790:35790")
cor5=Image("13579:13579:13579:13579:13579")
cor6=Image("01357:01357:01357:01357:01357")
cor7=Image("00135:00135:00135:00135:00135")
cor8=Image("00013:00013:00013:00013:00013")
cor9=Image("00001:00001:00001:00001:00001")
cor10=Image("00000:00000:00000:00000:00000")
todas_las_cortinas=[cor1,cor2,cor3,cor4,cor5,cor6,cor7,cor8,cor9,cor10]
display.show(todas_las_cortinas, delay=100, loop=True)
```

# Funciones para la pantalla

- `microbit.display.get_pixel(x, y)`. Devuelve el brillo del LED en la columna x y la fila y como un número entero entre 0 (apagado) y 9 (brillante).
- `microbit.display.set_pixel(x, y, value)`. Establece el brillo del LED en la columna x y la fila y como un número entero entre 0 y 9.
- `microbit.display.clear()`. Apaga (pone el brillo a 0) todos los LEDs.
- `microbit.display.show(image)`. Muestra la imagen.
- `microbit.display.show(image, delay=400, *, wait=True, loop=False, clear=False)`. Si `image` es una cadena, un real o un entero, muestra las letras/dígitos en secuencia. De lo contrario, si `image` es una secuencia iterable de imágenes, muestra estas imágenes en secuencia. Cada letra, dígito o imagen se muestra con un `delay` de milisegundos entre ellos.

Si `wait` es `True`, esta función se bloqueará hasta que la animación termine, de lo contrario la animación ocurrirá en segundo plano.

Si `loop` es `True`, la animación se repetirá para siempre.

Si `clear` es `True`, la pantalla se borrará después de que las iteraciones hayan terminado.

Los argumentos `wait`, `loop` y `clear` deben especificarse utilizando su palabra clave.

- `microbit.display.scroll(text, delay=150, *, wait=True, loop=False, monospace=False)`. Desplaza el texto horizontalmente en la pantalla. Si el texto es un número entero o flotante, se convierte primero en una cadena mediante str(). El parámetro delay controla la velocidad de desplazamiento del texto.

Si `wait` es `True`, esta función se bloqueará hasta que la animación termine, de lo contrario la animación ocurrirá en segundo plano.

Si `loop` es `True`, la animación se repetirá para siempre.

Si `monospace` es `True`, todos los caracteres ocuparán 5 columnas de píxeles de ancho, de lo contrario habrá exactamente 1 columna de píxeles en blanco entre cada carácter mientras se desplazan.

Los argumentos `wait`, `loop` y `monospace` deben especificarse utilizando su palabra clave.

- `microbit.display.on()`. Enciende la pantalla.
- `microbit.display.off()`. Apaga la pantalla. Esto permitirá reutilizar los pines GPIO asociados a la pantalla para otros fines.
- `microbit.display.is_on()`. Devuelve `True` si la pantalla está encendida, en caso contrario devuelve `False`.
- `microbit.display.read_light_level()`. Utiliza los LEDs de la pantalla en modo de polarización inversa para detectar la cantidad de luz que incide sobre la pantalla. Devuelve un número entero entre 0 (oscuridad) y 255 (máximo brillo) que representa el nivel de luz.

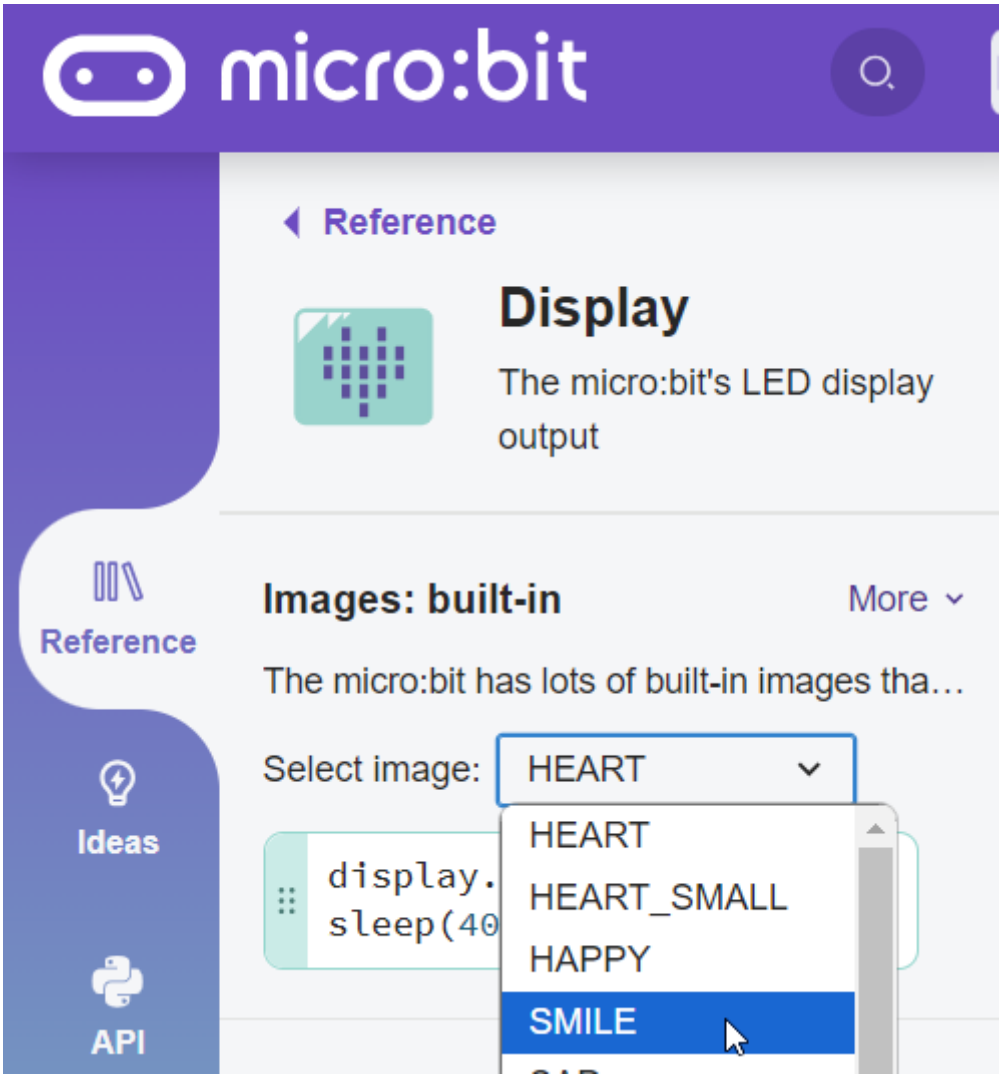
> *Extraído de Federico Coca Guia de Trabajo de Microbit CC-BY-SA*

Empezando

# Imágenes estáticas y animadas

## Imágenes estáticas

Sin tocar el código anterior, vamos ahora a **Display** y arrastramos el código de **sonrisa**



Vamos a tocar el código para que quede de esta forma, de esta manera practicamos la edición de código

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



# Code in a 'while True:' loop repeats forever
while True:
    display.show(Image.SMILE)
    sleep(1000)
    display.scroll('Hola mundo')
```

La sonrisa se ve un segundo = 1.000 mseg y luego ejecuta el scroll

> ¿¿ Qué esperas para experimentar otras imágenes ?

## Imágenes animadas

Podemos poner las imágenes prediseñadas en la variable **Image** pero también podemos crearlas fácilmente. En el siguiente programa se define qué led de la matriz 5x5 se enciende a la máxima intensidad (0-9)

> Avanzando en la programación, se definen 5 variables **catedux** tipo imagen, y se define una variable **all_catedus** que es tipo array que contienen todas.

```
from microbit import *

catedu1 =  Image("00900:"
                 "09000:"
                 "90000:"
                 "09000:"
                 "00900")

catedu2 =  Image("09000:"
```

```
                  "90000:"
                  "09000:"
                  "00900:"
                  "00090")

catedu3 =  Image("90000:"
                  "09000:"
                  "00900:"
                  "00090:"
                  "00009")

catedu4 =  Image("00009:"
                  "00090:"
                  "00900:"
                  "09000:"
                  "90000")

catedu5 =  Image("00090:"
                  "00900:"
                  "09000:"
                  "90000:"
                  "09000")

all_catedus = [catedu1,catedu2,catedu3,catedu2,catedu1,catedu5,catedu4]
while(True):
    display.show(all_catedus, delay=200)
```

https://www.youtube.com/embed/yLjKgy2NaPI

O jugar con las intensidades: En este juego de **luces del coche fantástico** se utiliza la intensidad media 5 :

> Este ejemplo de regular la intensidad del led es imposible de realizar en programación por bloques.

Copyright 2026

```
from microbit import *


catedu1 =  Image("00005:"
                 "00000:"
                 "00000:"
                 "00000:"
                 "00000")

catedu2 =  Image("00009:"
                 "00050:"
                 "00000:"
                 "00000:"
                 "00000")

catedu3 =  Image("00005:"
                 "00090:"
                 "00500:"
                 "00000:"
                 "00000")

catedu4 =  Image("00000:"
                 "00050:"
                 "00900:"
                 "05000:"
                 "00000")

catedu5 =  Image("00000:"
                 "00000:"
                 "00500:"
                 "09000:"
                 "50000")

catedu6 =  Image("00000:"
                 "00000:"
                 "00000:"
```

Copyright 2026

```
                  "05000:"
                  "90000")

catedu7 =  Image("00000:"
                  "00000:"
                  "00000:"
                  "00000:"
                  "50000")

all_catedus =
[catedu1,catedu2,catedu3,catedu4,catedu5,catedu6,catedu7,catedu6,catedu5,catedu4,catedu3,cated
u2]
while(True):
    display.show(all_catedus, delay=100)
```

https://www.youtube.com/embed/jko4xAxbm2I

¿No sabes lo que es el coche fantástico? eso es que no tienes la edad adecuada para la robótica □□

https://www.youtube.com/embed/oNeQi8-PXAU

También podemos hacerlo pixel a pixel y no utilizar variables tipo array

```
from microbit import *

display.clear()
while(True):
    for n in range(0, 5):
        display.set_pixel(n, 3, 9)
        if (n<4):
            display.set_pixel(n+1, 3, 5)
        if (1<n):
            display.set_pixel(n-1, 3, 5)
```

```
        if (1<n):
            display.set_pixel(n-2, 3, 0)
        sleep(200)
    for n in reversed(range(0, 5)):
        display.set_pixel(n, 3, 9)
        if (n<4):
            display.set_pixel(n+1, 3, 5)
        if (1<n):
            display.set_pixel(n-1, 3, 5)
        if (n<3):
            display.set_pixel(n+2, 3, 0)
        sleep(200)
```

https://www.youtube.com/embed/a5J1793GCdg

Empezando

# Eventos para los botones

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

Si trabajamos con versiones anteriores a V2 solamente disponemos de los botones A, B y A+B, pero si tenemos una versión V2 también disponemos del botón táctil incorporado en el logo, aunque a todos los efectos este se considera un pin de entrada.

El logo no es tratado exactamente como un botón, sino como un pin de nombre logo. En el borde existen otros tres pines, los 0, 1 y 2. Por ello la forma de trabajar con el logo va a ser un poco diferente, como veremos en la actividad A04.

La diferencia fundamental, ademas de la forma, es que el logo es un sensor capacitivo y los pines son sensores resistivos. En la práctica esto significa que el logo funciona simplemente tocandolo y los pines necesitan cerrar el circuito con GND, por lo que para que funcionen como pulsador debemos tocar tanto el pinto como GND.

Si queremos que MicroPython reaccione a los eventos de pulsación de los botones, debemos ponerlo en un bucle infinito y comprobar si el botón `is_pressed`.

- **Función `is_pressed()`**

Para trabajar con los botones de la micro:bit tenemos disponibles funciones que se han cargado al importar el módulo `microbit`. Estas funciones están basadas en la función genérica `is_pressed()` pensada para saber que tecla de un teclado se ha pulsado. Sin embargo, en el caso de MicroPython a para micro:bit a estos botones se les ha asignado un nombre a cada uno, `button_a` para el A y `button_b` para el B, de manera que para usarlos se llama al botón y con el operador `.` a la función `is_pressed()`. Por ejemplo, `button_a.is_pressed()` es el código encargado de saber si estamos pulsando el botón A y `button_b.is_pressed()` si lo es el B.

- **Función `get_pressed()`**

Esta función retorna el total acumulado de pulsaciones de botones y restablece este total a cero antes de volver. Es decir, podemos capturar el número de veces que hemos pulsado un botón. El valor de retorno es un número, por lo que, para mostrarlo en la pantalla de LEDs hay que convertirlo en cadena con la función `str()`.

- **Función** `was_pressed()`

Devuelve `True` o `False` para indicar si se ha presionado el botón desde la última vez que se inicio el dispositivo o se llamó a este método. Llamar a este método borra el estado de que ha sido pulsado, de modo que el botón debe pulsarse de nuevo antes de que este método vuelva a retornar `True`.

Vamos a hacer un ejemplo que aclarará mejor lo explicado. Se trata de crear un programa (le podremos de nombre Caritas_X) en el que mientras mantegamos pulsado el botón A se muestra una cara sonriente, si no se pulsa ningún botón se muestra una cara triste y si se pulsa el botón B la cara desaparece (se apagan todos los LEDs) y tras 2 segundos aparece una X que se va haciendo cada vez mas grande partiendo del punto central. Finalmente pasados otros 2 segundos el programa vuelve a empezar. El código es:

```
from microbit import *
while True:
    while True:
        if button_a.is_pressed():
            display.show(Image.HAPPY)
        elif button_b.is_pressed():
            break
        else:
            display.show(Image.SAD)

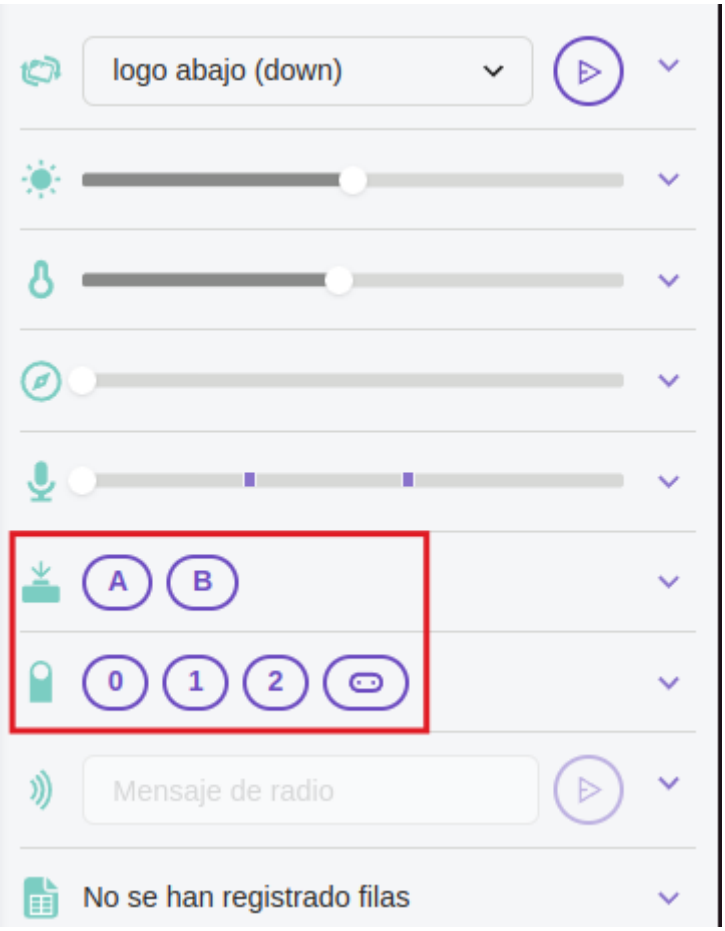
    display.clear()
    sleep(2000)
    mi_X_peque = Image("00000:00000:00900:00000:0000")
    display.show(mi_X_peque)
    sleep(200)
    mi_X_media = Image("00000:09090:00900:09090:0000")
    display.show(mi_X_media)
    sleep(200)
    mi_X_grande = Image("90009:09090:00900:09090:90009")
    display.show(mi_X_grande)
    sleep(2000)
```

En la animación siguiente vemos como funciona

f_ifelifelse.png

*Federico Coca Guia de Trabajo de Microbit CC-BY-SA*

Si observamos con cuidado apreciaremos que en algún momento se accionan los botones A y B pero los que aparecen en la parte inferior, debajo de la pantalla de simulación. Están al lado de un logotipo que indica que se pulsen con una flechita. Justo debajo de estos aparecen los citados del borde de placa y el logo junto a ellos, pues es tratado asi, como un pin, y además a su izquierda hay un candado cerrado indicativo de que no se está usando ninguno de ellos. En la imagen siguiente se ve mejor lo indicado.




*Federico Coca Guia de Trabajo de Microbit CC-BY-SA*

Vamos a crear otro ejemplo en el que se cuenten las veces que pulsamos el botón A o el botón B durante un tiempo de 3 segundo. El programa es el siguiente:

```
from microbit import *


sleep(3000) #Espera de 3 segundos
```

```
#Convertimos número a cadena con str()
pulsado = str(button_b.get_presses())

display.show(pulsado)

# Por si hemos pulsado mas de 9 veces
display.scroll(pulsado)
```

En la 'Referencia' del compilador, dentro de Botones tenemos un ejemplo que nos indica el botón que hemos pulsado con cuatro opciones posibles, el A, el B, A o B y finalmente A y B. Animamos a cargarlos y probarlos para familiarizarnos todo lo posible con ellos.

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

Empezando

# Botones

Los botones pueden dar juego, combinándolos con la instrucción if --- else

```
from microbit import *

while True:
    if button_a.is_pressed():
        display.show(Image.HAPPY)
    elif button_b.is_pressed():
        break
    else:
        display.show(Image.SAD)

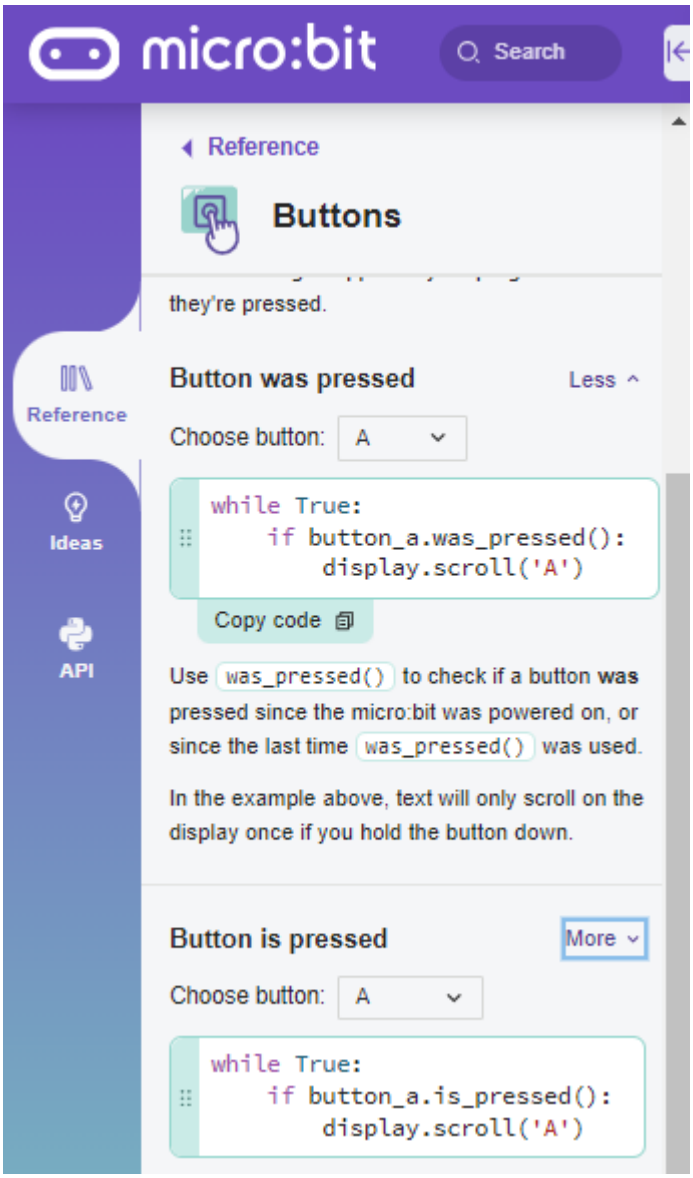
display.clear()
```

Extraido de tutorial https://microbit-micropython.readthedocs.io/en/v2-docs/tutorials/buttons.html

https://www.youtube.com/embed/VgimuhHlRgQ

> ¿Qué pasa si pulsamos el botón B ?

En el apartado **Reference** podemos ir a **Buttons** tenemos diferentes muestras de código :




La diferencia entre este código

```
while True:
    if button_a.was_pressed():
        display.scroll('A')
```

y este otro código

```
while True:
    if button_a.is_pressed():
```

```
        display.scroll('A')
```

es muy sutil, no hay diferencia si apretamos el botón A *excepto si lo mantenemos pulsado*

El siguiente código, visualiza el número de veces que pulsas el botón A durante 3 segundos :

```
from microbit import *
display.scroll('Press A')
sleep(3000)
display.scroll(button_a.get_presses())
```

# Jugando

Copyright 2026

Jugando

# 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_6uIkbaShE

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

Jugando

# Temperatura

La función en micropython para leer la temperatura de la placa en ºC 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/uczf6IvUBH4

Copyright 2026

Jugando

# 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 calibtrar
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:

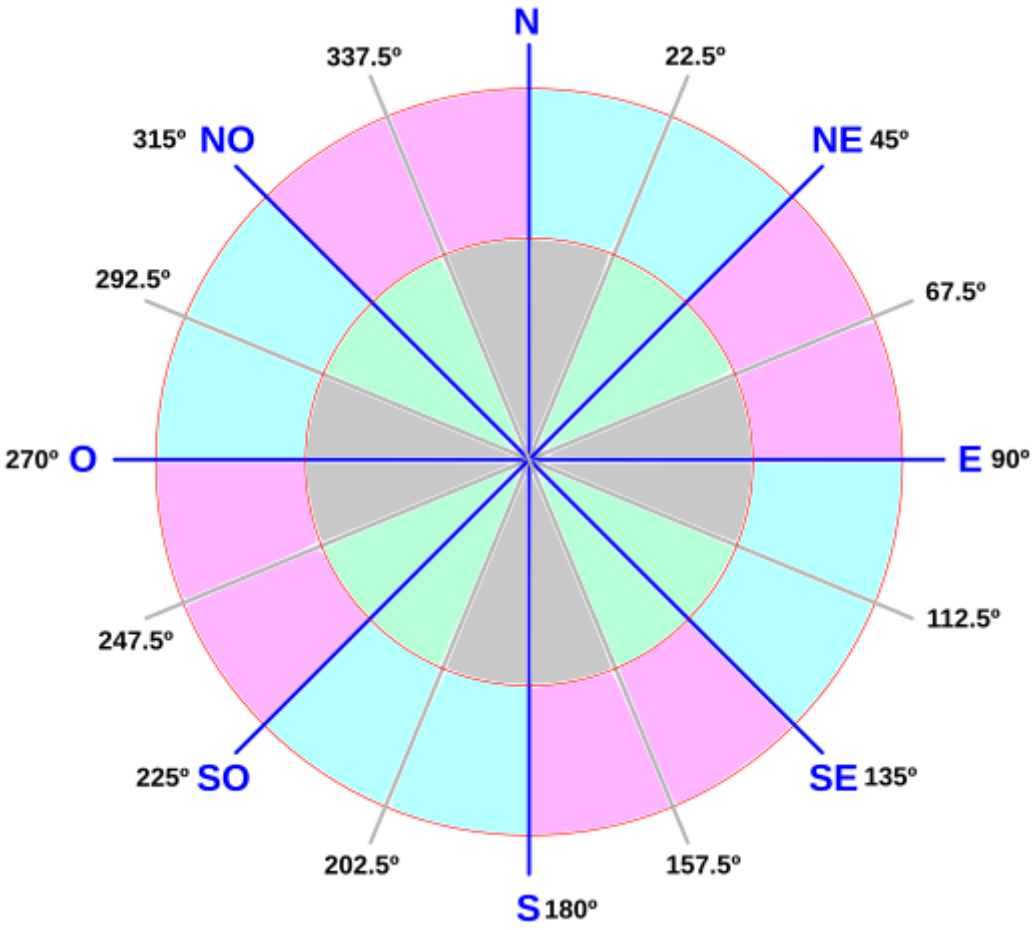
```
from microbit import *

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 calibtrar
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])
```

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

Jugando

# 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 ± 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",
    "Si",
    "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 consistentente 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)
```

Copyright 2026

```
            # 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

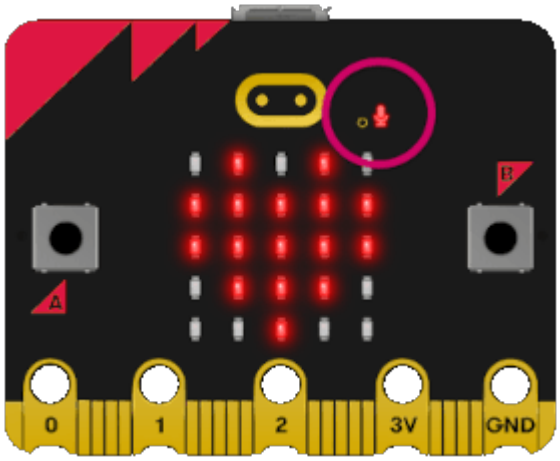
Jugando

# Micrófono

> Página extraída de Federico Coca Guia 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 (>= 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 los imagenes para el grafico de barras
grafico5 = Image("99999:"
                 "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 leido
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:
```

Copyright 2026

```
            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

```
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)
```

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

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

Jugando

# 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 energia 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 contantes definidos en el módulo de radio : `RATE_1MBIT` o `RATE_2MBIT`.

Copyright 2026

> **Nota** Una velocidad de datos menor de 250 kbit/seg es compatible con micro:bit V1, ypuede ser posible con micro:bit V2, pero no se garantiza que funcione en todos losdispositivos. 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 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

# Hackeando

Copyright 2026

Hackeando

# Pines de Entrada/salida

> Página extraída de Federico Coca Guia 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 sonria y si no se toca ningúno 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
```

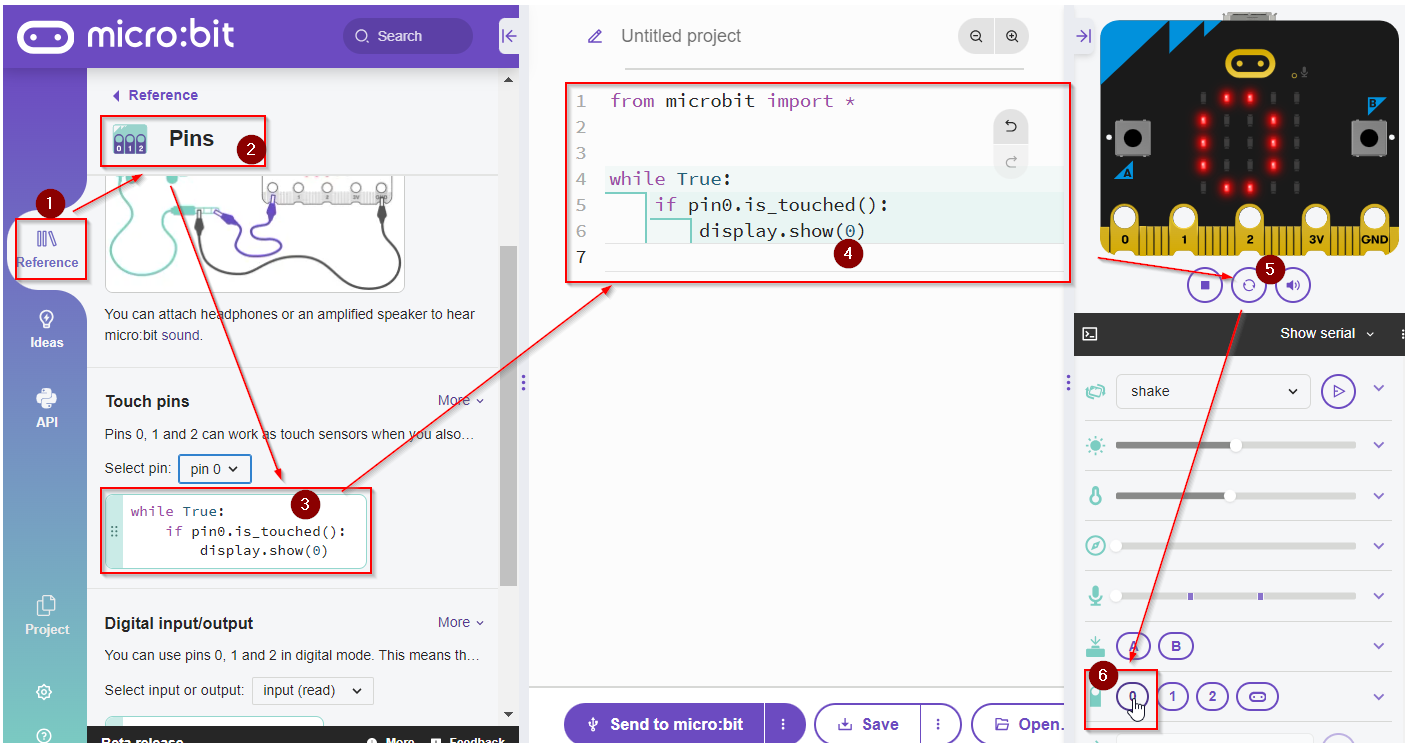
Página extraída de Federico Coca Guia de Trabajo de Microbit CC-BY-SA

Hackeando

# 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




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/uI2p9HazV1Y

Copyright 2026

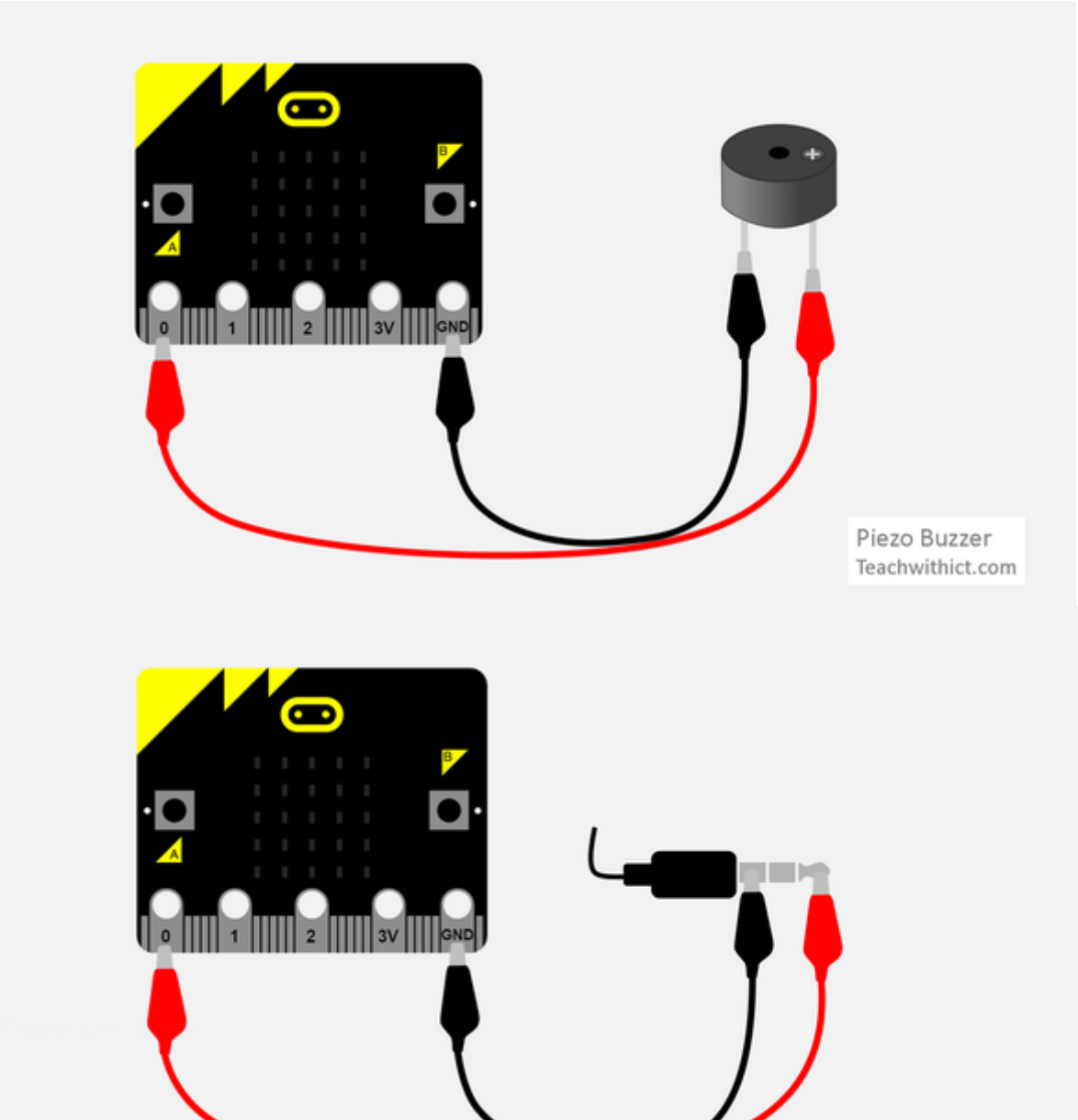
Hackeando

# 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:



*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 señan analógica en sonido (o sea un altavoz normal y corriente)

> si quieres conectar un **buzzer activo**, reproduce un tono (prederminado) 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.

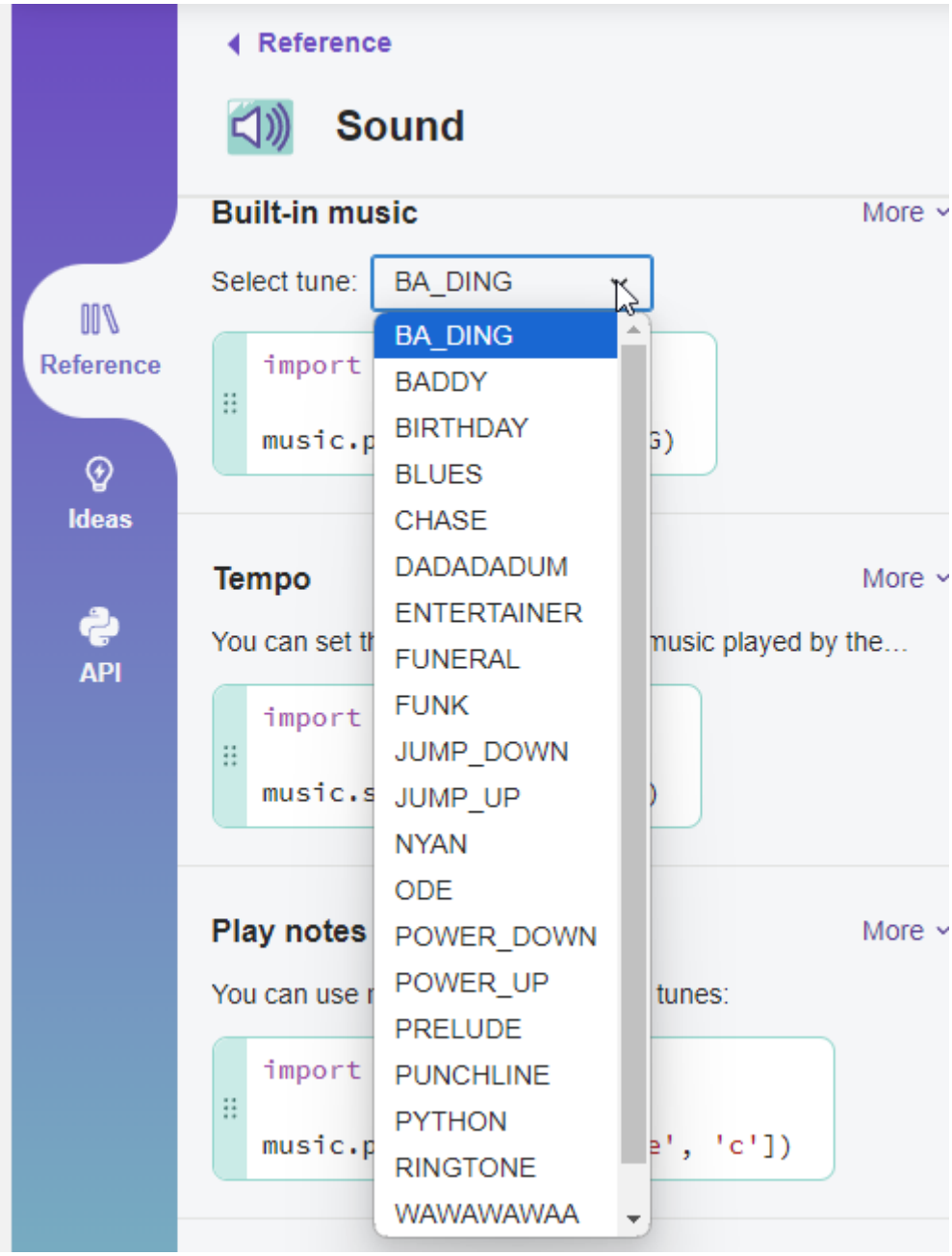
```
import music
from microbit import *
display.show(Image.GHOST)
while True:
    for freq in range(880, 1760, 16):
        music.pitch(freq, 6)
    for freq in range(1760, 880, -16):
        music.pitch(freq, 6)
```

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 "E**xpressive sounds**"  como audio.play(Sound.TWINKLE)

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

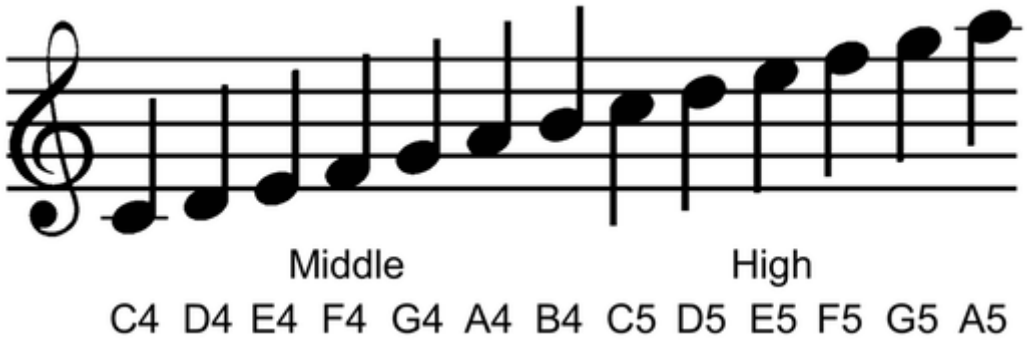
```
import speech
from microbit import *
display.show(Image.FABULOUS)
speaker.on()
set_volume(255)
```

```
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 |
|---|---|---|
| 𝅝 | **Semibreve** Whole note | 16 |
| 𝅗𝅥 | **Minim** Half note | 8 |
| ♩ | **Crotchet** Quarter note | 4 |
| ♪♪ | **Quaver** Eighth note | 2 |
| 𝅘𝅥𝅯 | **Semiquaver** Sixteenth note | 1 |

*Fuente [https://www.teachwithict.com/microbit-music-python.html](https://www.teachwithict.com/microbit-music-python.html)*
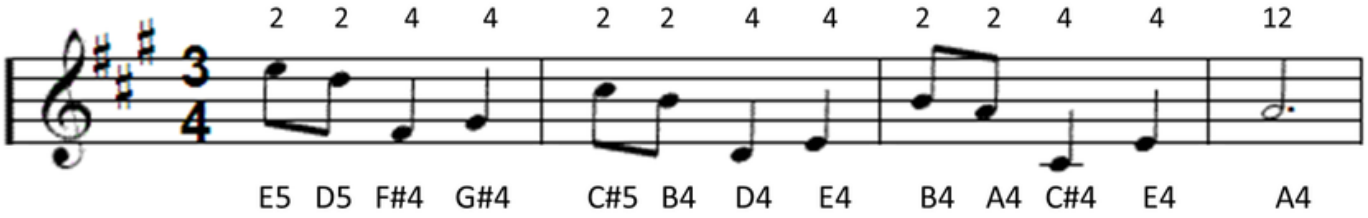
*BBC micro:bit "Magic 8-Ball" lesson is licenced under a [Creative Commons Attribution 4.0 International License](#).*

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

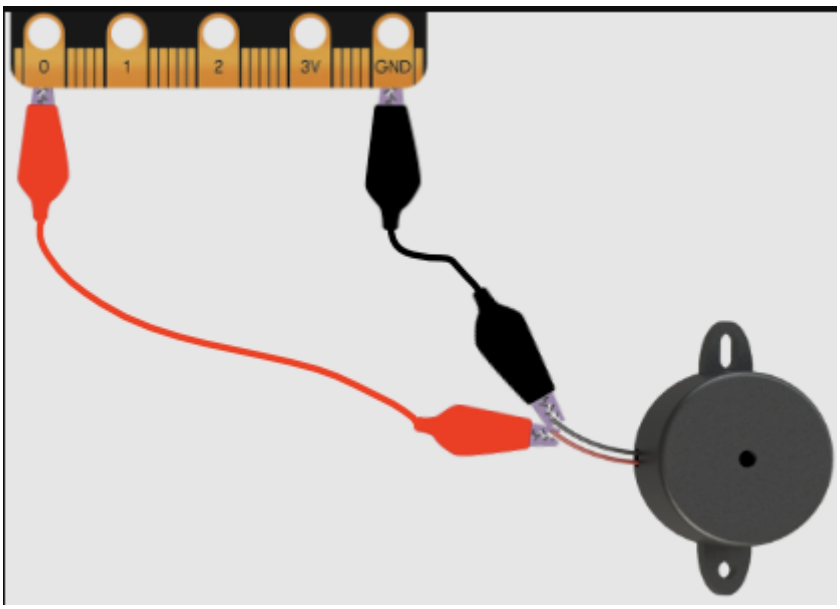
Hackeando

# 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*

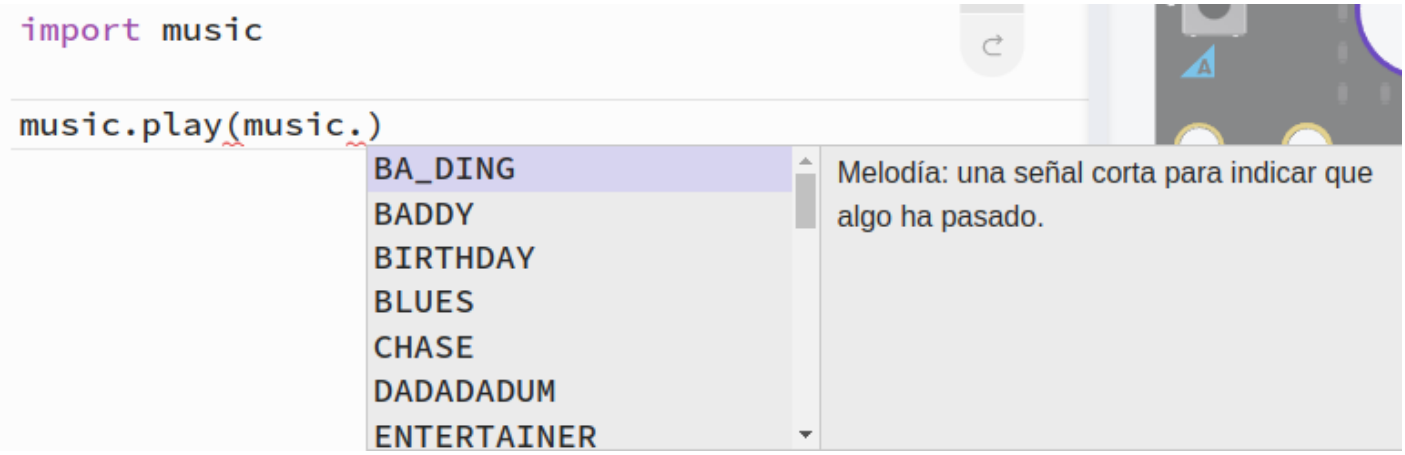
> **Nota:** Debemos asegurarnos de que el zumbador es pasivo y no activo, que tan solo puede reproducir un tono. En el apartado 'Componentes discretos' de Conceptos técnicos podemos encontrar como distinguirlos.

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 Guia de Trabajo de Microbit CC-BY-SA*

En la referencia de la API podemos encontrar mas inforamción 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 melodia 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 rpiten 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 en 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

Hackeando

# 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

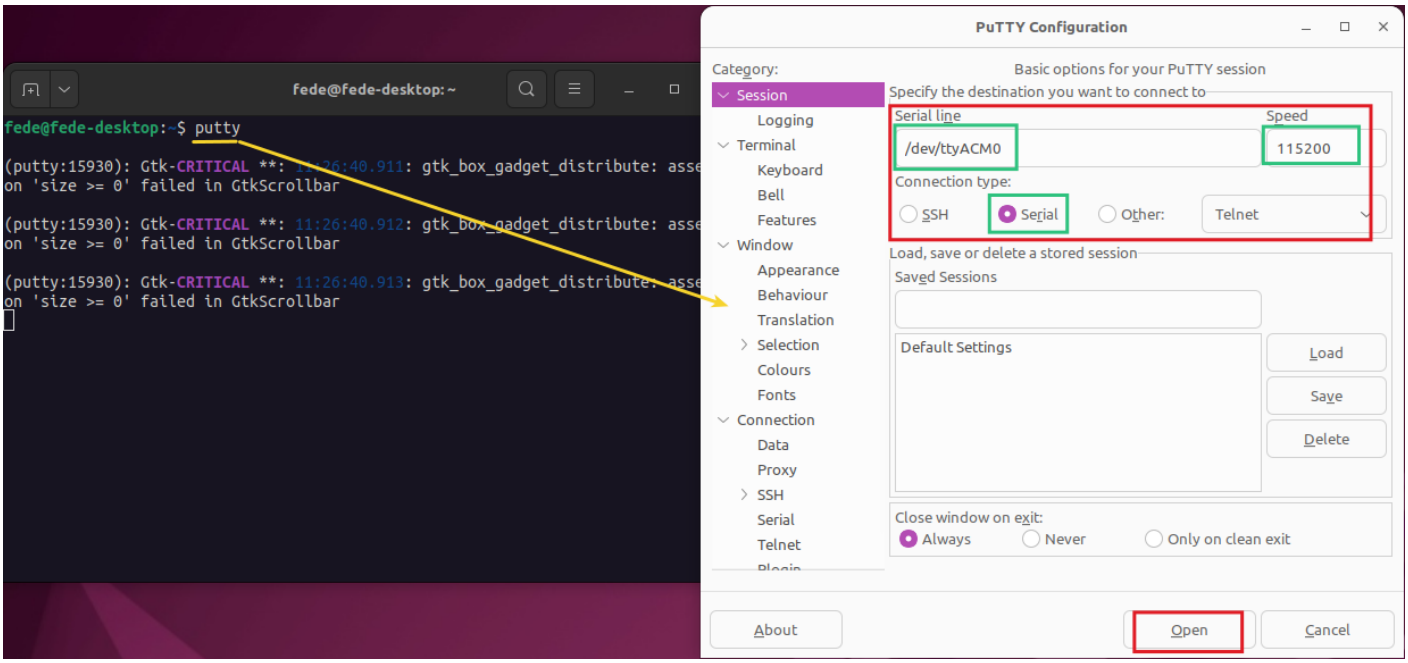
Entramos en https://putty.org/ y en Downloads descarga y ejecuta el fichero correspondiente, si es un Intel x86 64bits al menos que sea un equipo viejo 32 bit. Si es un AMD insala el arm

> **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** :

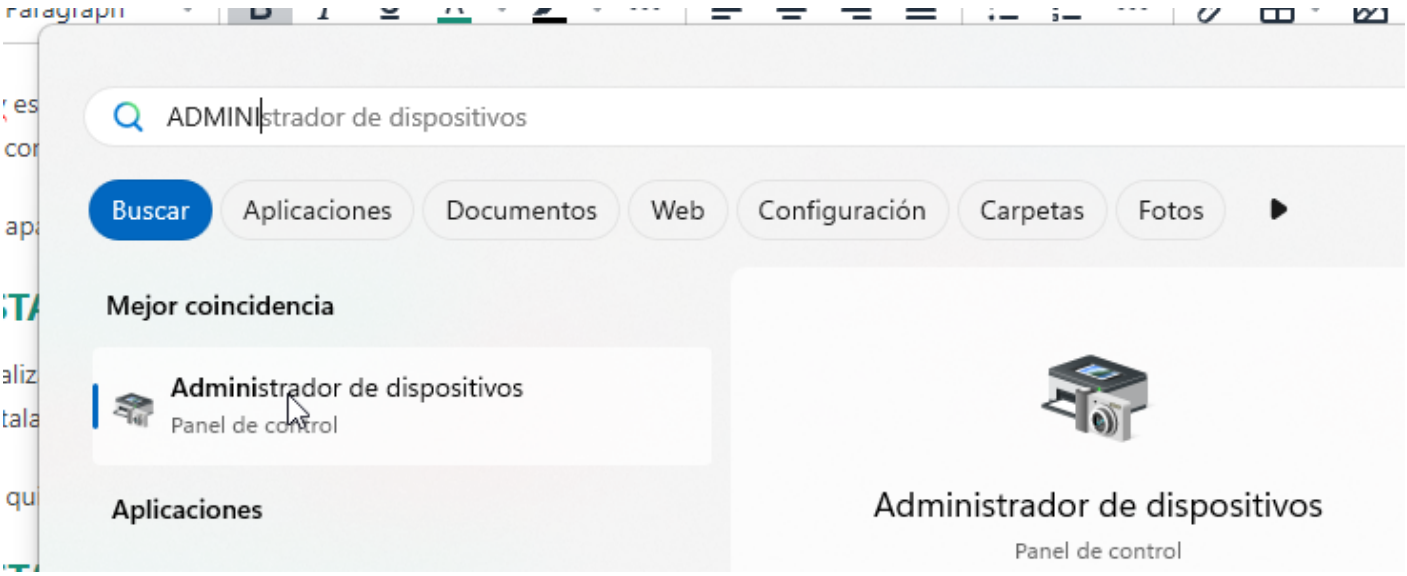
Copyright 2026



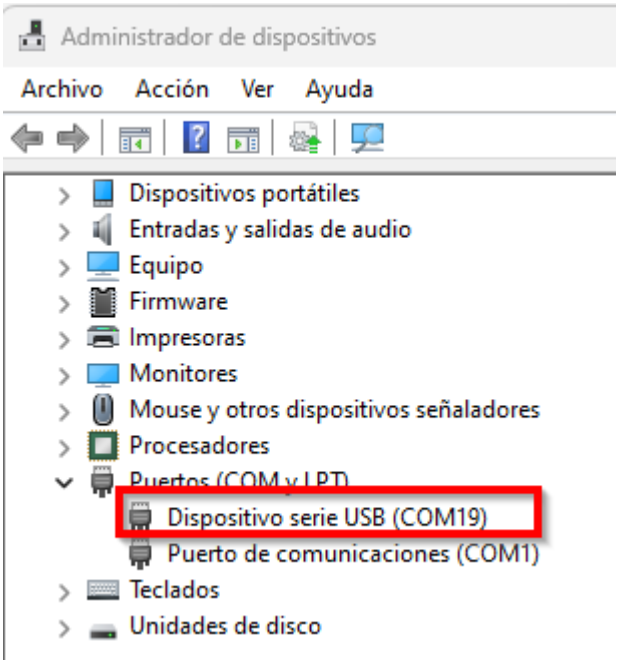

Autor Federico Coca de https://fgcoca.github.io/Guia-de-trabajo-para-microbit/conceptos/serie/ licencia CC-BY-SA

## CONFIGURAR PUTTY PUERTO SERIE CON MICROBIT WINDOWS

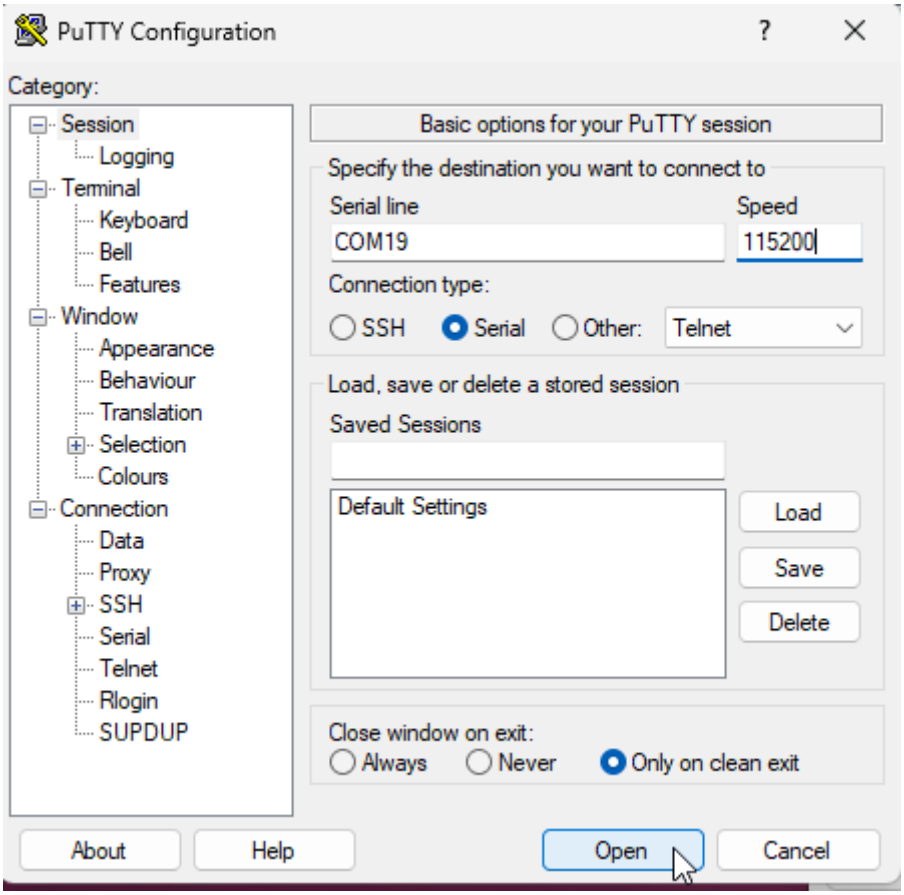
Conectamos nuestra microbit por el puerto serie, y ejecutamos el **Administrador de dispositivos**




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 PYTHONhttps://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)

Copyright 2026

Hackeando

# UART

> Página extraída de Federico Coca Guia de Trabajo de Microbit 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 dispositovos 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 transmisitido 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.

---

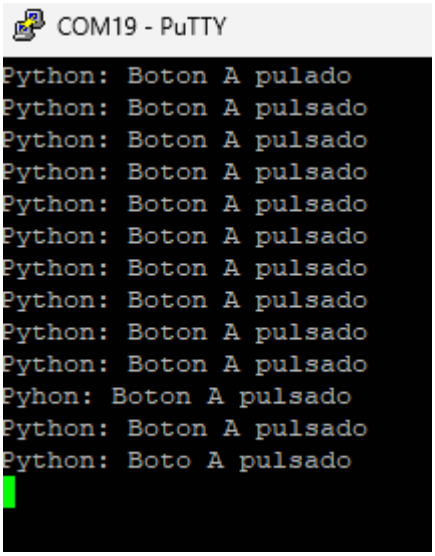
El ejemplo siguiente nos va a permitir comprobar la comunicación serie desde MicroPython. Grabamos el programa en la micro:bit.

```
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/ Licencia CC-BY-SA

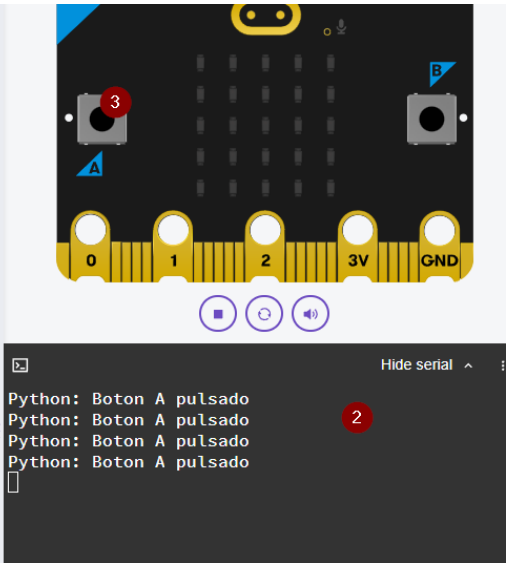



https://www.youtube.com/embed/44kSyEAdF6Q

En el simulador de Micropython también aparece

```
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")
```



Hackeando

# 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
())

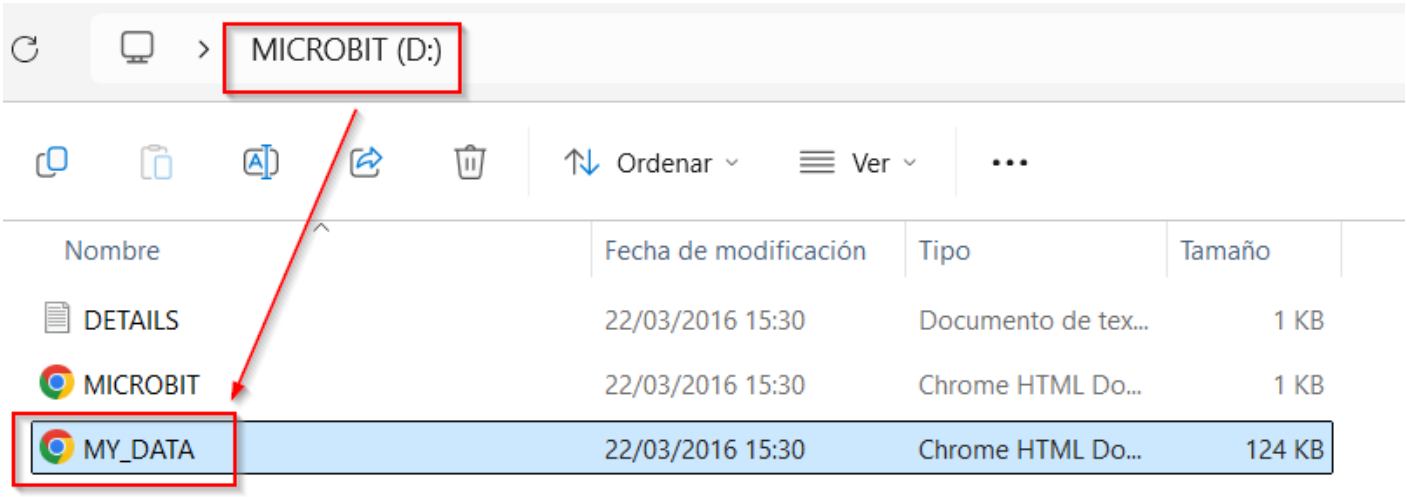
while True:
   sleep(500)
```

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

# micro:bit data log

Download | Copy | Update data... | Clear log... | Visual preview

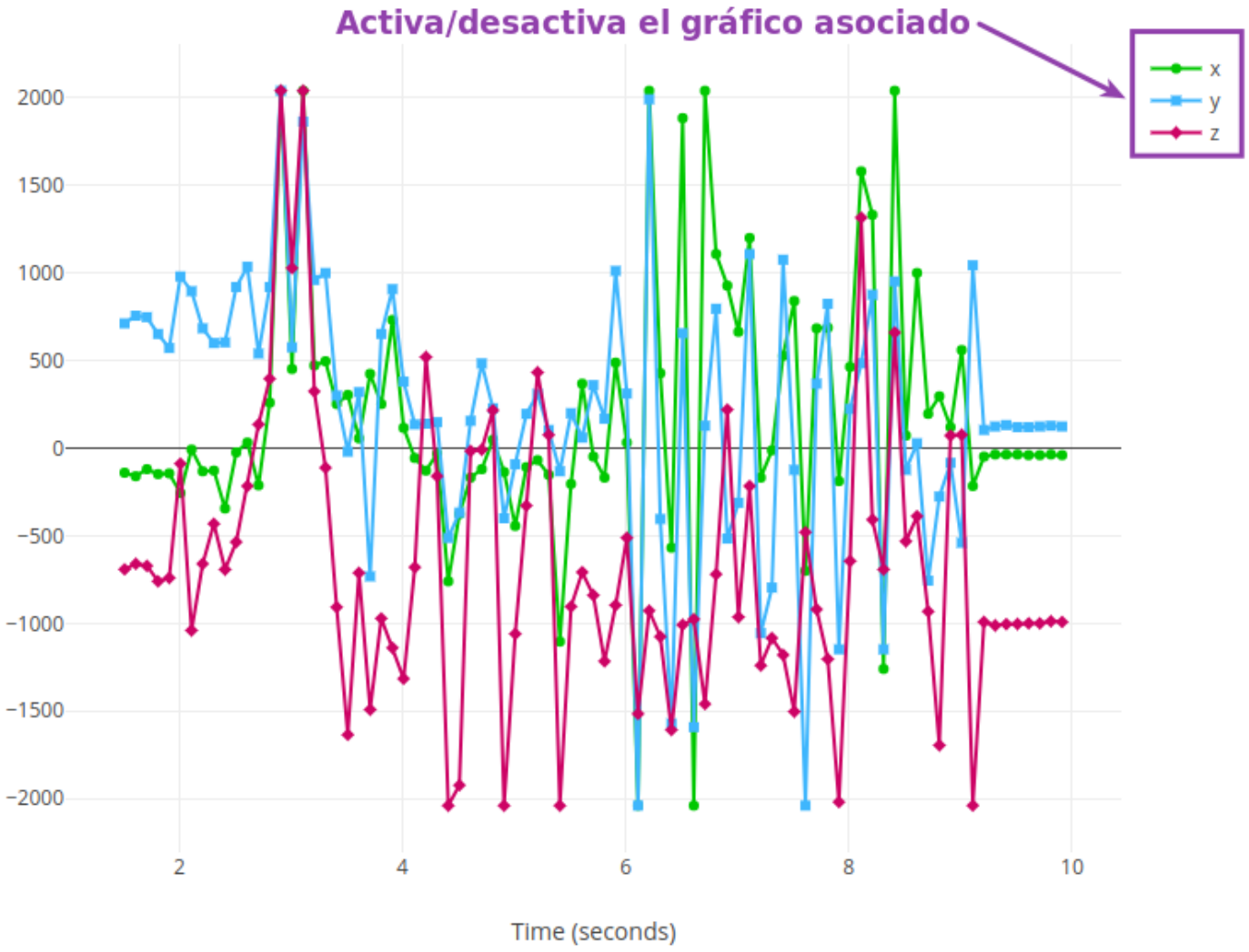
This is the data on your micro:bit. To analyse it and create your own graphs, transfer it to your computer. You can copy and paste your data, or download it as a CSV file which you can import into a spreadsheet or graphing tool. Learn more about micro:bit data logging.

| 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.

Copyright 2026




*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)

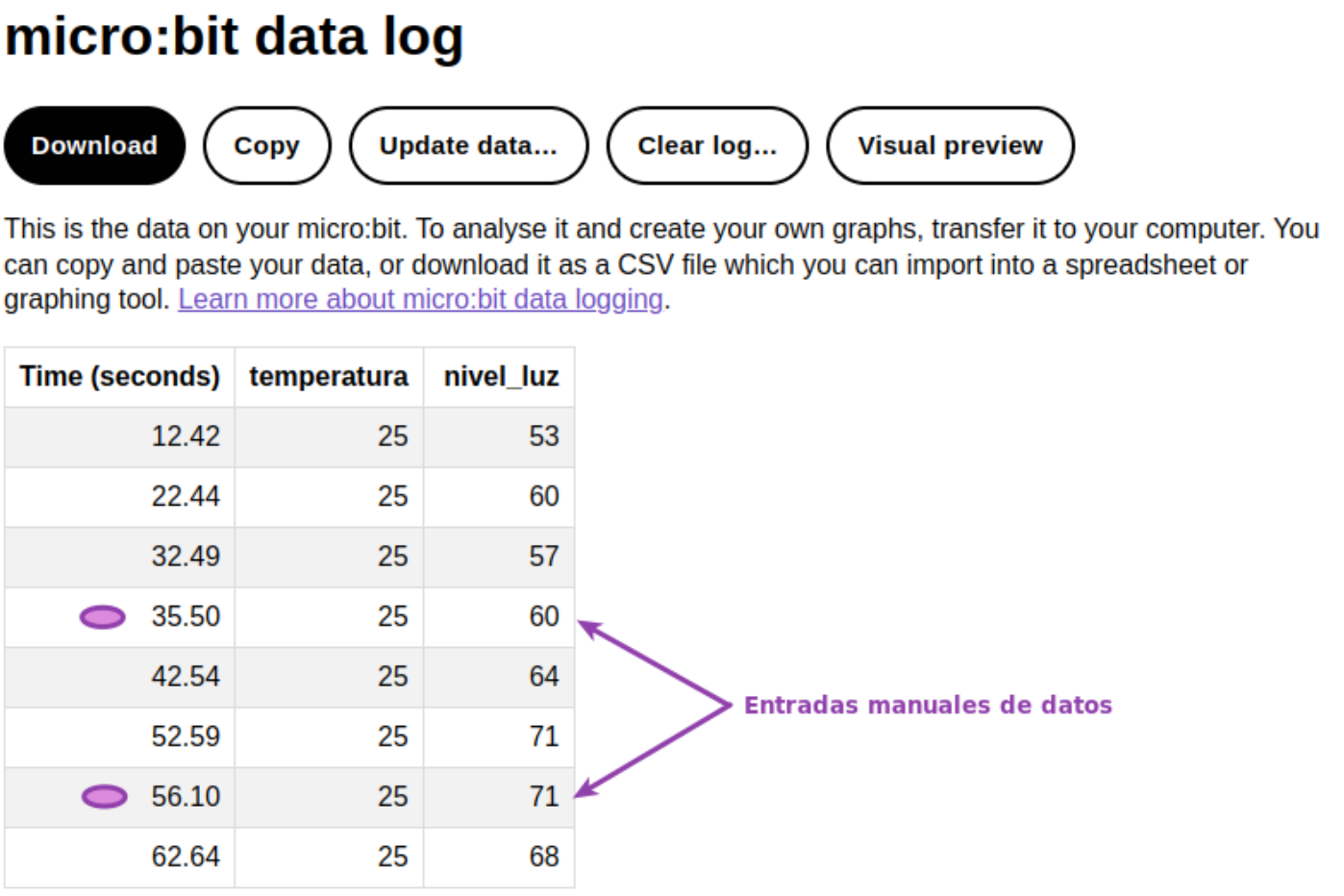
continue_registro = True

# 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.NO)
    sleep(500)
```

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




*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*

# Créditos

## Autoría y licencias

- **Federico Coca** GitHub Twitter Instagram Mas información.
  - Autor de los siguientes capítulos :
    - Introducción
      - **Microbit v2 vs v1**
      - **Introducción al Python**
      - **Micropython de microbit**
    - Empezando
      - **Imágenes**
      - **Eventos para los botones**
    - Jugando
      - **Nivel de luz**
      - **Acelerómetro**
      - **Magnetómetro**
      - **Micrófono**
      - **Radio**
    - Hackeando
      - **Pines de Entrada/salida**
      - **Música**
      - **UART**
      - **Registro de datos**
  - Licencia Creative Commons Atribución-CompartirIgual 4.0 International ("CC BY-SA "). Ver: Guia de trabajo para micro:bit - Miscelanea-Licencias https://fgcoca.github.io/Guia-de-trabajo-para-microbit/Miscelanea/lic/
- **Javier Quintana CATEDU 2024**
  - Maquetador
  - Autor del resto de páginas
  - Licencia
    - CC-BY-SA

Cualquier observación o detección de error en soporte.catedu.es



GOBIERNO DE ARAGON

Departamento de Educación, Cultura y Deporte

CATEDU
CENTRO ARAGONÉS de TECNOLOGÍAS para la EDUCACIÓN

Copyright 2026