Empezando

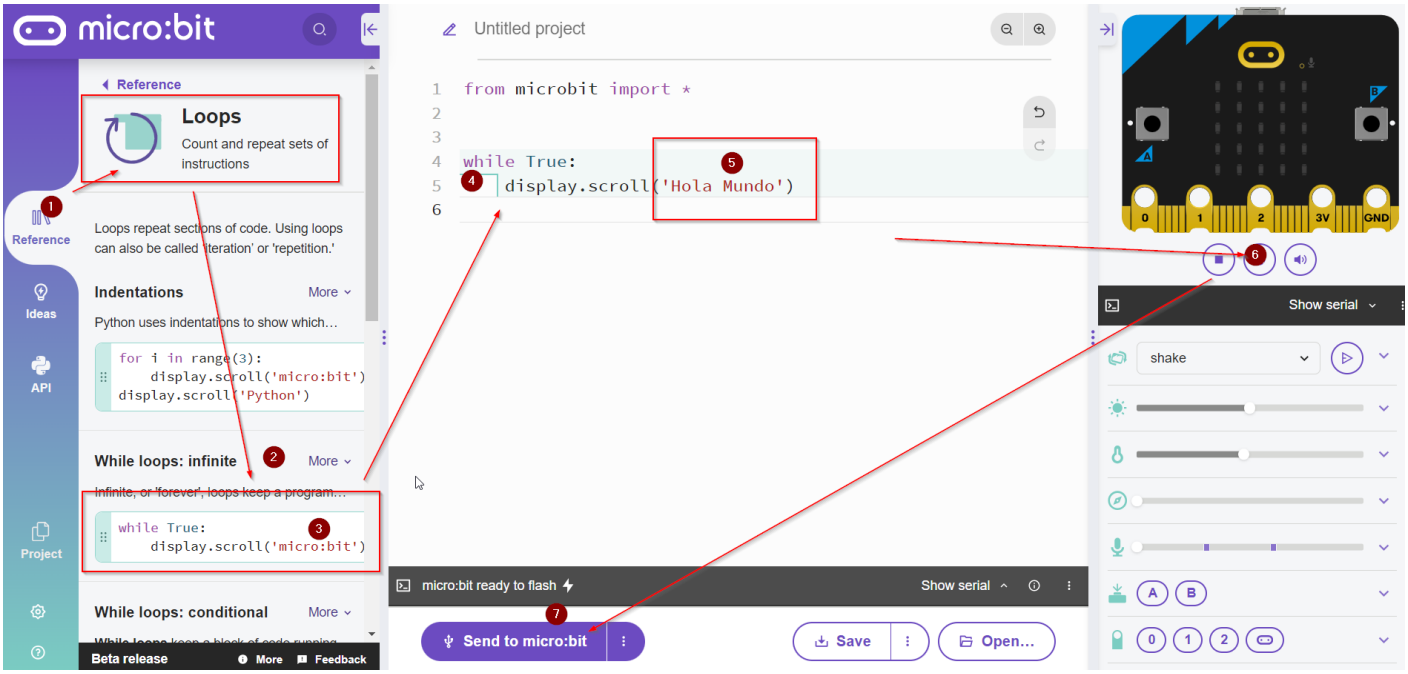
- Hola Mundo
- Imágenes
- Imágenes estáticas y animadas
- Eventos para los botones
- Botones

Copyright 2026

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

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.

Copyright 2026

```
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
```

Copyright 2026

```
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.

Copyright 2026

```
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

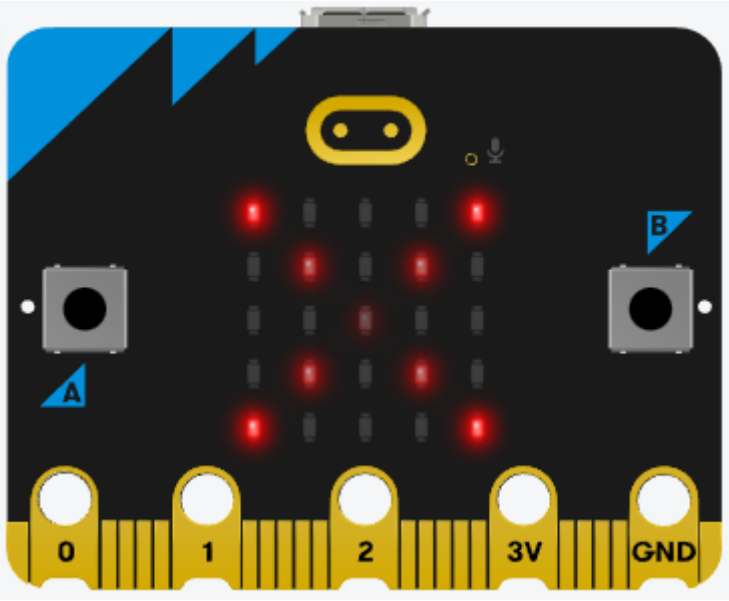


Imagen de una X en relieve

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.

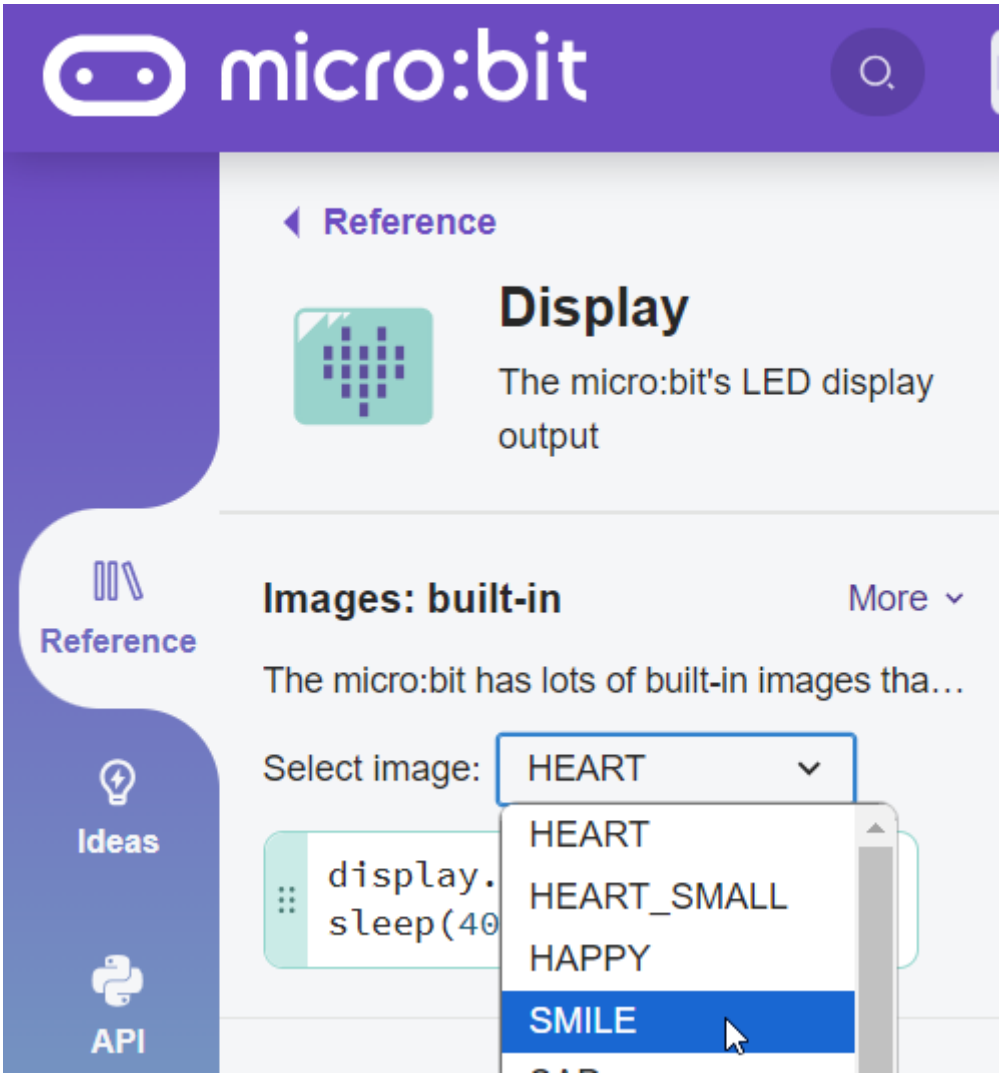
- `microbit.display.on()`. Enciende la pantalla.
- `microbit.display.off()`. Apaga la pantalla. Esto permitirá reutilizar los pines GPIO asociados a la pantalla para otros fines.
- `microbit.display.is_on()`. Devuelve `True` si la pantalla está encendida, en caso contrario devuelve `False`.
- `microbit.display.read_light_level()`. 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*

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)
```

Copyright 2026

```
        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

Copyright 2026

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)

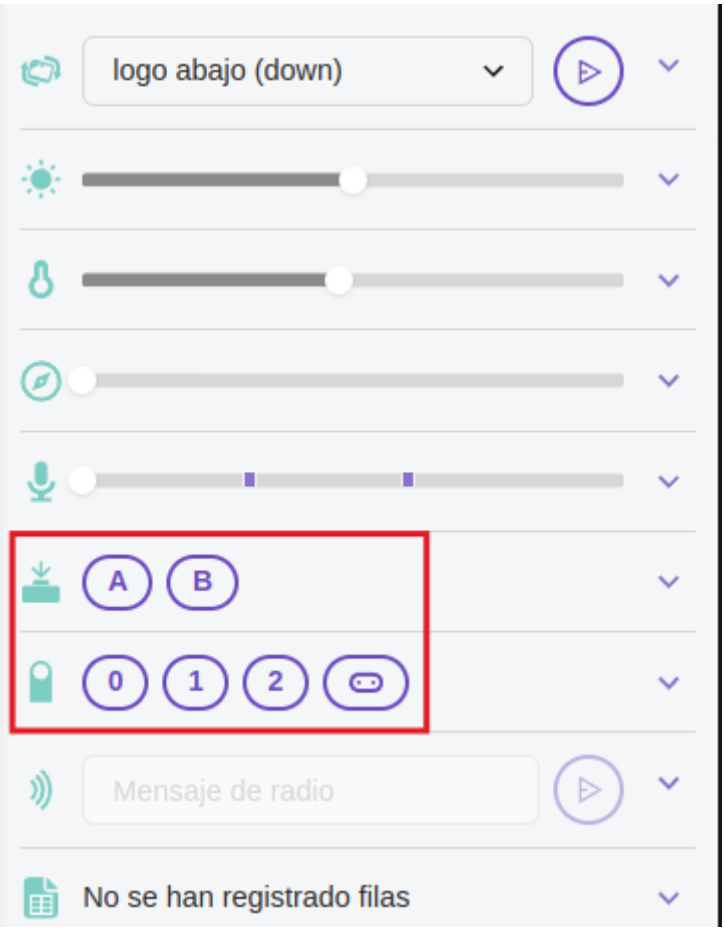
    display.clear()
    sleep(2000)
    mi_X_peque = Image("00000:00000:00900:00000:0000")
    display.show(mi_X_peque)
    sleep(200)
    mi_X_media = Image("00000:09090:00900:09090:0000")
    display.show(mi_X_media)
    sleep(200)
    mi_X_grande = Image("90009:09090:00900:09090:90009")
    display.show(mi_X_grande)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

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)

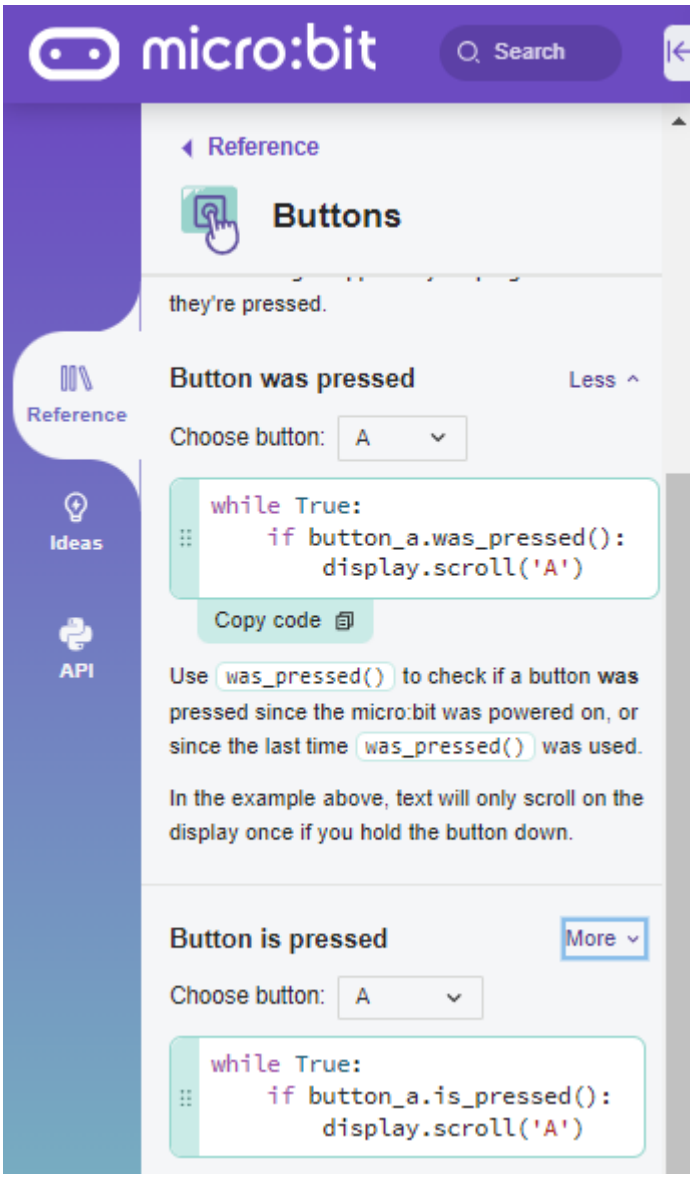
display.clear()
```

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())
```