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

MicroPython nos ofrece muchas imágenes integradas para mostrar por pantalla y podemos crear efectos interesantes. Mediante la característica de autocompletar se nos van a mostrar todas las definidas que están listadas en la documentación oficial. Ya hemos visto como cargar una imagen, lo que puedo aconsejar en este momento es realizar el ejercicio de mostrar cada una de las

disponibles para familiarizarnos con ellas.

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

Imágenes DIY

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.

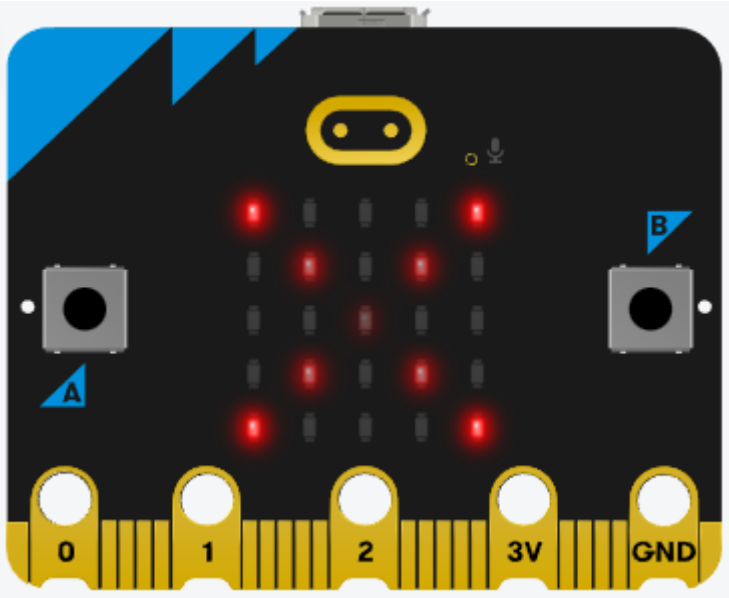

Imagen de una X en relieve

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*

Revision #18
Created 15 September 2024 09:32:33 by Javier Quintana
Updated 27 September 2024 12:24:19 by Javier Quintana