Fundamentos de la Programación
Proyecto 1: *micro:bit*

Ingeniería de Robótica SW
GSyC, ETSIT, URJC

29 de octubre de 2019

Resumen

Los *micro:bit* son una plataforma de hardware diseñada para programar dispositivos para la Internet de las cosas (*Internet of Things—IoT*) que pueden ser utilizados autónomamente para recibir señales a través de sus propios sensores o de otros conectados al *micro:bit*, para controlar robots, etc. El *micro:bit* es una placa de hardware Open Source, basada en un microcontrolador ARM Cortex-M0 que fue diseñada por la BBC (British Broadcasting Corporation) en el Reino Unido para utilizarla en entornos educativos como una plataforma IoT en los colegios ingleses en 2016. A fecha de hoy se utiliza en muchos otros países del mundo, en colegios, institutos y universidades. También en la industria: el Bluetooth SIG utiliza los *micro:bit* para formar desarrolladores en la tecnología Bluetooth Mesh[1] y AdaCore los utiliza para formar a desarrolladores en la programación sobre máquina desnuda con el lenguaje Ada[2].

Para realizar esta práctica es importante que hayas estudiado todos los temas hasta el 9 (Transformación de Secuencias) incluido.

En esta práctica tendrás que programar un *micro:bit* que se te prestará en clase. Se piden dos programas, `graf_acelerometros.py` y `cronometro.py`. Es opcional la entrega del tercer programa, `cronometro_alarmas.py`.

En primer lugar se describe qué es el *micro:bit* y cómo se programa. Después aparece la descripción de los dos programas que tienes que realizar, y del tercer programa opcional. Finalmente se describe cómo se evaluará la práctica, y cómo tienes que entregarla.

1. *micro:bit*

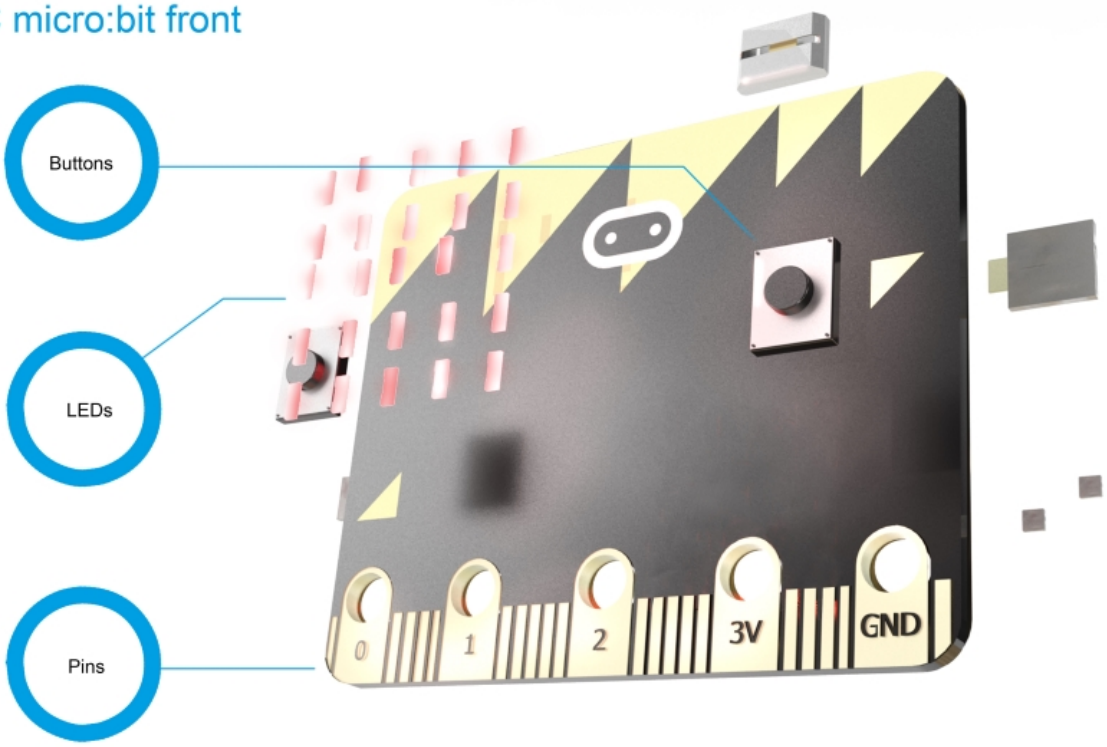
En la caja que te entregamos encontrarás un *micro:bit*, un cable USB-microUSB, un cargador de baterías, 2 baterías AAA y manuales. Por favor trata con sumo cuidado todos estos elementos.

- La placa aloja el **procesador ARM Cortex-M0** con 16kB de memoria RAM. Además contiene 256kB de memoria flash (no volátil) en la que se pueden almacenar programas y datos en ficheros.
- Posee también una pantalla de **25 LEDs**, dos **botones programables** y uno de reinicialización (*reset*). Además tiene **5 conectores** en forma de anilla, uno para alimentación, uno para tierra y 3 para entrada/salida (E/S) a los que se pueden conectar cables con pinzas tipo cocodrilo o conectores banna de 4mm. También se puede acceder a **23 pines** distintos usando un conector externo.
- En la placa *micro:bit* vienen integrados varios sensores:
  - Un **acelerómetro** que permite medir la posición del *micro:bit* en los ejes X, Y y Z
  - Un **magnetómetro** que permite utilizar el *micro:bit* como brújula y como detector de metales
  - Un sensor de **temperatura** del procesador
  - Los leds se pueden utilizar como sensores de intensidad lumínica
- La **radio Bluetooth** permite enviar y recibir mensajes vía radio a otros *micro:bit* o a otros dispositivos Bluetooth como teléfonos móviles.
- El *micro:bit* se puede conectar por **USB** a un ordenador para transferir datos y para recibir alimentación eléctrica, aunque también puede ser alimentado mediante baterías externas.

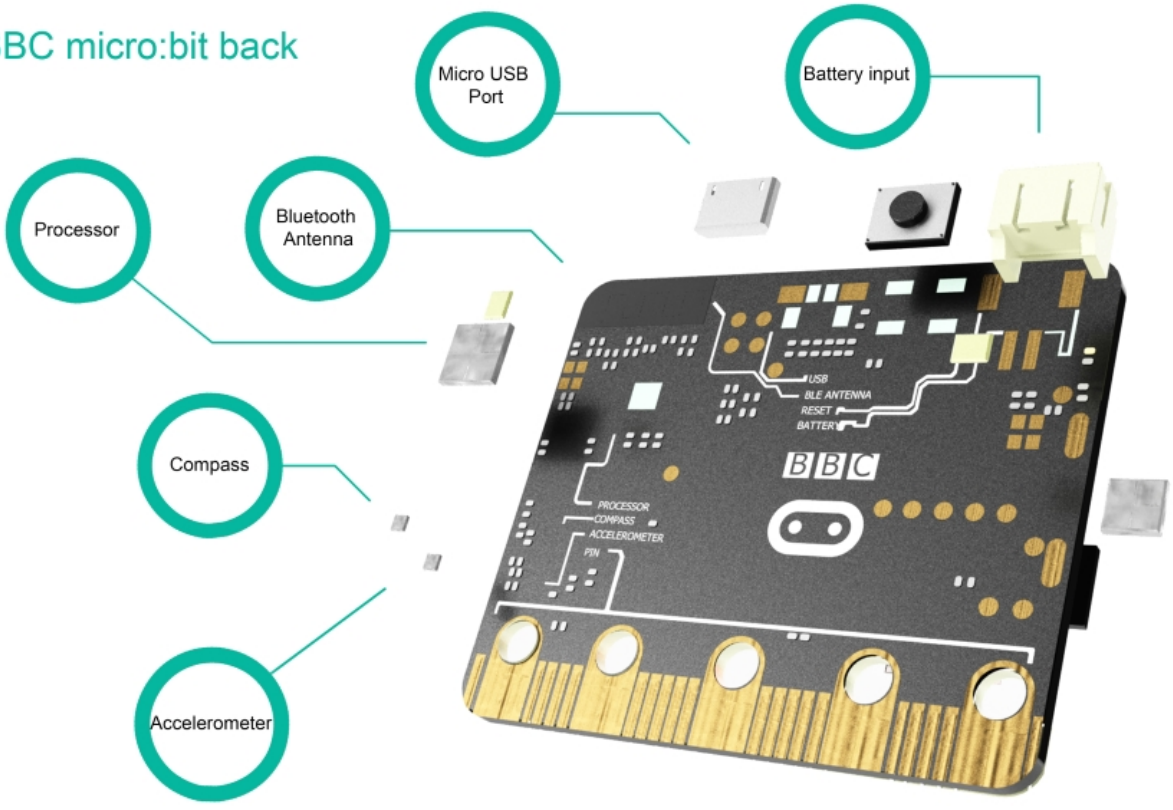
La siguiente figura muestra los componentes de un *micro:bit*:

[1] `https://www.bluetooth.com/develop-with-bluetooth/build/developer-kits/mesh-developer-study-guide`
[2] `https://blog.adacore.com/ada-on-the-microbit`

BBC micro:bit front



BBC micro:bit back



2. Entorno de programación

Cada *micro:bit* tiene instalado un *runtime* programado en C++ por la Universidad de Lancaster[3]. El *runtime* está compuesto del SW necesario para controlar el HW del *micro:bit* proporcionando una API de alto nivel para poder programar más cómodamente el dispositivo. Como alternativa, se puede instalar también Zephyr[4] o programar directamente la máquina desnuda, sin ningún *runtime*, en C/C++, Ada, etc.

En esta asignatura utilizaremos el *runtime* de la Universidad de Lancaster que viene preinstalado, pero no programaremos en C++. Encima de la API C++ se han desarrollado diferentes entornos de programación en múltiples lenguajes: Python, JavaScript, y varios entornos de programación con bloques gráficos: Scratch 3.0 y MakeCode, desarrollado por Microsoft para esta plataforma. Ver `https://microbit.org/code/`.

Cuando conectas mediante USB el *micro:bit* el sistema operativo de tu ordenador lo reconocerá como una unidad de disco denominada MICROBIT. Para poder ejecutar un programa hay que copiar su código en esta unidad de disco. El *runtime* del *micro:bit* sólo ejecuta un programa a la vez, dando comienzo su ejecución al terminar la recepción del programa. Puede reiniciarse un programa pulsando el botón *Reset* del *micro:bit*. Una vez descargado el programa en el *micro:bit* se almacena en la memoria flash, por lo que puede ejecutarse sin estar conectado al ordenador en caso de recibir alimentación de baterías.

En esta asignatura utilizaremos MicroPython, un intérprete de Python para pequeños procesadores (microcontroladores) como el ARM Cortex-M0 del *micro:bit*.

2.1. Programación con MicroPython

Hay dos entornos de programación que permiten programar el *micro:bit* usando MicroPython:

- Desde el navegador utilizando el entorno de desarrollo Web disponible en esta página: `https://python.microbit.org/v/1.1` Para descargar el programa en tu placa *micro:bit* haz click en el botón *Download* y arrastra el fichero descargado hasta la unidad de disco denominada MICROBIT.
- Instalando en tu máquina el editor *mu-editor* `https://codewith.mu/`. En Ubuntu GNU/Linux puedes instalarlo desde la línea de comandos con las órdenes (NOTA: sustituye USUARIO por tu nombre de usuario en tu ordenador):

  ```
  sudo apt install python3-pip
  pip3 install mu-editor
  sudo adduser USUARIO dialout
  ```

  Para arrancarlo ejecuta `mu-editor`. IMPORTANTE: asegúrate de tener conectado al puerto USB de tu ordenador el *micro:bit* antes de arrancar el editor *mu-editor*.

  El editor *mu-editor* tiene algunas ventajas frente al entorno en el navegador:

  - El botón *Flash* permite enviar directamente el programa al *micro:bit* sin tener que copiarlo al dispositivo MICROBIT.
  - El botón *REPL*[5] ofrece un intérprete dentro del propio editor que permite ejecutar código interactivamente en el procesador del *micro:bit*.
    - Cuando se activa REPL, el programa que se está ejecutando en el *micro:bit* se para.
    - Para volver a empezar a ejecutar el programa cargado es necesario pulsar CTRL-D
    - Para interrumpir el programa de nuevo es necesario pulsar CTR-C
    - Mientras el programa se está ejecutando con REPL activado, la salida generada con `print` en el programa se envía por el puerto USB y se muestra en el panel REPL de *mu-editor* en el ordenador al que está conectado el *micro:bit*. Esto es extremadamente útil para depurar el código: se pueden ir imprimiendo mensajes con `print` en cualquier momento: por ejemplo se puede hacer que se impriman cuando se pulse un botón, o cuando se realice un gesto al mover el *micro:bit*, etc. Imprimir el estado de variables del programa puede ayudar a depurar programas. Hay que tener cuidado: si se imprime con `print` continuamente se puede saturar la comunicación con el editor.
    - Mientras el programa está parado (con CTRL-C), puede consultarse el estado de las variables en el momento en que se interrumpió el programa.

[3] `https://lancaster-university.github.io/microbit-docs/`

[4] `https://www.zephyrproject.org/`

[5] El acrónimo REPL significa Read-Eval-Print-Loop). Se usa para denominar a la interfaz interactiva de intérpretes de lenguajes interpretados.

- El botón *Trazador* permite representar gráficamente en el editor las tuplas de datos numéricos que envía el *micro:bit* al editor por el cable USB. En el eje X se representa el tiempo y en el eje Y los valores de cada uno de los componentes de las tuplas recibidas. Estas tuplas se envían llamando a `print()`. Ejemplo: `print((5,10,12))`.
- Desde el editor se puede acceder al sistema de ficheros del *micro:bit*.

mu-editor guarda los ficheros en una carpeta de nombre `mu-code` dentro de tu carpeta hogar.

Consulta esta documentación para aprender a utilizar *mu-editor*: `https://codewith.mu/en/tutorials/1.0/microbit`

3. Ejemplos de programación con MicroPython

En esta sección recogemos algunos ejemplos que te permitirán comenzar a programar en pocos segundos tu *micro:bit*.

Trata de descubrir su comportamiento. Verás que en todos los programas utilizamos un bucle infinito programado con `while`, aún no visto en clase. Un bucle `while` hace que se repitan sus sentencias mientras la condición sea True. Por ello un bucle `while True:` es un bucle cuyas sentencias se repiten para siempre, sin terminar nunca.

3.1. Ejemplo 1

```
#############################################################################
#
# Agita el micro:bit y observa los cambios en los leds
#
# Arranca el REPL y observa lo que se muestra cuando pulsas el botón B
# en el micro:bit
#
# Cambia la llamada a button_b.is_pressed() por button_b.was_pressed()
# y observa la diferencia
#
# sleep(2000) suspende la ejecución del programa durante 2000ms. Prueba
# otros valores
#

from microbit import *

def main():
    number_sad = 0
    while True:
        display.show(Image.HAPPY)
        if accelerometer.was_gesture("shake"):
            display.show(Image.SAD)
            number_sad = number_sad + 1
            sleep(2000)
        if button_b.is_pressed():
            print ("number_sad: " + str(number_sad))

if __name__ == "__main__":
    main()
```

3.2. Ejemplo 2

```
#####################################################################
#
# En este otro caso las sentencias del bucle while se ejecutan si
# no se ha presionado el botón A. Si se ha presionado, la condición será
# falsa y no se ejecutarán las sentencias del bucle while,
# continuando la ejecución en la sentencia que viene después del bucle:
# display.clear()
#
from microbit import *

def main():
    while not button_a.is_pressed():
        display.show(Image.HAPPY)
        if accelerometer.was_gesture("shake"):
            display.show(Image.SAD)
            sleep(2000)

    display.clear()
    sleep(1000)
    display.show("FIN. Pulsa reset para reiniciar")

if __name__ == "__main__":
    main()
```

3.3. Ejemplo 3

```
#####################################################################
#
# Muestra en mu-editor el Trazador y el intérprete (REPL) y observa
# los valores.
#
# Comenta la última sentencia y observa lo que ocurre
#
from microbit import *

def main():
    while True:
        accel_x = accelerometer.get_x()
        accel_y = accelerometer.get_y()
        accel_z = accelerometer.get_z()

        print ( (accel_x, accel_y, accel_z) )
        sleep(200)

if __name__ == "__main__":
    main()
```

En esta URL tienes una guía de programación del *micro:bit* con MicroPython: `https://microbit.org/guide/python/`. Explora la documentación para que conozcas los métodos disponibles en la biblioteca de programación de MicroPython para *micro:bit*.

4. Ejercicio 1: media de valores de acelerómetros

Escribe un programa en un fichero llamado `graf_acelerometros.py`

El objetivo de esta práctica es que hagas un programa que muestre una gráfica con la media de las últimas medidas del acelerómetro en los ejes x, y, z.

Para hacer esta práctica tienes que utilizar el editor *mu-editor* para que se muestre en la herramienta Trazador o Plotter los valores que va enviando *micro:bit* a través del cable USB al editor.

El programa tendrá declaradas al principio una variable que almacenará un valor entero, de nombre `MAX`, inicializada a 10.

El programa deberá estar leyendo continuamente los valores del acelerómetro de *micro:bit* para cada uno de los 3 ejes x, y, z. Estos valores los irá almacenando en una lista de tuplas de tres elementos correspondientes a las medidas del acelerómetro para x,y,z. Esta lista no podrá almacenar más de `MAX` lecturas. Cuando se llene se eliminará la lectura más antigua para hacer hueco.

El programa calculará la media de los últimas `MAX` lecturas y enviará dicha media, para cada uno de los ejes, a *mu-editor*, para que el trazador los represente gráficamente. Recuerda que para poder enviar valores representables por el trazador por *mu-editor* hay que imprimir las tuplas con *print*. En este caso tuplas de tres números.

Para que el programa esté todo el tiempo realizando esta tarea puedes utilizar un bucle infinito con `while True:`, y dentro de él escribir todo el código de tu programa.

Deberás leer los tres valores del acelerómetro para los ejes x, y, z con `accelerometer.get_values()`.

Activa en *mu-editor* el trazador o plotter y verás una gráfica de los valores leídos.

Prueba a modificar el valor de `MAX` y observa las diferencias cuando mueves el *micro:bit*. Usa por ejemplo los siguientes valores: 1, 5, 10, 20, 50, 100.

5. Ejercicio 2: Cronómetro

Escribe un programa en un fichero llamado `cronometro.py`

Este programa no utiliza el trazador de *mu-editor*. Sin embargo te puede venir bien utilizar *mu-editor* mientras lo realizas para mostrar mensajes con `print` en el REPL que te ayuden a depurar tu código.

En esta práctica escribirás el código para un cronómetro con alarmas para *micro:bit*. Se irá contabilizando el tiempo que transcurre desde que comienza a ejecutarse la aplicación en *micro:bit*.

La pantalla del *micro:bit* mostrará los segundos, los minutos, o las horas transcurridas desde que comenzó a ejecutarse la aplicación.

- Si no hay ningún botón pulsado la pantalla del *micro:bit* mostrará los segundos.
- Si está pulsado el botón A se mostrarán las horas.
- Si está pulsado el botón B se mostrarán los minutos.

Por ejemplo, si hace 12 horas, 3 minutos y 13 segundos que se arrancó la aplicación, la pantalla mostrará el número 13 si no se pulsa ningún botón, el número 12 si se pulsa el botón A y el número 3 si se pulsa el botón B.

Al arrancar el programa se comenzará a contar el tiempo en tres variables, `horas`, `minutos` y `segundos`, que se inicializarán a 0.

El programa entrará en un bucle infinito (`while True:`). En cada vuelta del bucle, tras comprobar si se ha pulsado algún botón, se deberá mostrar los segundos (o las horas o los minutos), y dejar que pase un segundo de tiempo, llamando para ello a `sleep(1000)`. De esta forma, cada vuelta del bucle tardará un segundo al menos en ejecutarse [6]. En cada vuelta del bucle por tanto bastará con actualizar los contadores de segundos, minutos y horas.

Se representarán los dígitos en pantalla usando el siguiente sistema:

- Cuando el número a representar tenga un sólo dígito, éste se mostrará en las columnas 3 y 4 (coordenadas X 3 y 4), no mostrándose nada en el resto de columnas.
- Cuando el número a representar tenga dos dígitos, el primero se mostrará en las columnas 0 y 1 y el segundo en las columnas 3 y 4.

[6] tardará más pues el microprocesador tiene que ejecutar las instrucciones, sumándose el tiempo empleado en ello al segundo que pasa en `sleep(1000)`

Cada dígito por lo tanto constará de dos columnas definidas con las intensidades de cada led. La intensidad 9 es la máxima y 0 la mínima (apagado). Así, representaremos el número 3 mediante un *string* con el siguiente valor: `three = "90909:99999"`. El *string* contiene las intensidades de los leds del dígito 3. Si quisiéramos dibujar el dígito 3 sólo en la pantalla tendríamos que dibujarlo en las columnas 3 y 4, con la siguiente correspondencia de coordenadas de leds:

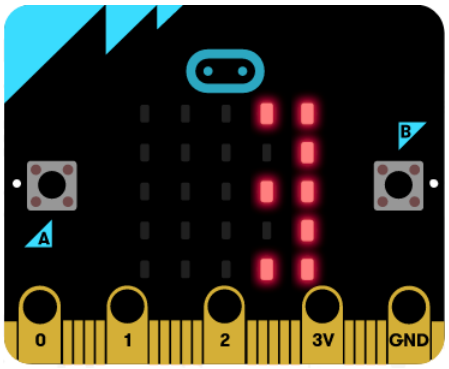
Columna 3:

columna (X)	fila (Y)	intensidad
3	0	**9**
3	1	**0**
3	2	**9**
3	3	**0**
3	4	**9**

Columna 4:

columna (X)	fila (Y)	intensidad
4	0	**9**
4	1	**9**
4	2	**9**
4	3	**9**
4	4	**9**

El resultado sería el siguiente:



De esta manera, utilizaremos las siguientes declaraciones para representar los dígitos en la pantalla:

```
zero  = "99999:99999"
one   = "00000:99999"
two   = "90999:99909"
three = "90909:99999"
four  = "99900:09999"
five  = "99909:90999"
six   = "99999:90999"
seven = "90900:99999"
eight = "99099:99099"
nine  = "99900:99999"

fonts = (zero, one, two, three, four, five, six, seven, eight, nine)
```

Para escribir en los leds de la pantalla de *micro:bit* habrá que utilizar el método `display.set_pixel()`. En esta práctica **no se puede usar para mostrar los dígitos ningún otro método de** `display`.

NOTA: si una variable `segundos` contiene un entero de dos dígitos, `segundos % 10` devuelve el dígito de las unidades y `segundos // 10` el dígito de las decenas.

6. Ejercicio 3 (opcional): Cronómetro con alarmas

Copia el programa del cronómetro en un fichero llamado `cronometro_alarmas.py`.

Cuando se agite el *micro:bit* se podrá introducir una nueva alarma. Aparecerá en la pantalla de leds del micro:bit el contador 0 para fijar la hora de la alarma. Pulsando el botón A se incrementará este valor. Cuando se quiera pasar a los minutos se pulsará el botón B. Entonces aparecerá de nuevo el dígito 0 para fijar los minutos de la alarma y con el botón A se podrá incrementar el contador de minutos. Tras pulsar el botón B se pasará a fijar los segundos del mismo modo. Finalmente se pulsará el botón B, almacenándose la nueva alarma en una lista de alarmas pendientes.

Al realizar esta parte de la práctica tendrás que modificar el código escrito para el cronómetro, pues ahora cuando se pulsan los botones A o B, habrá que hacer una cosa u otra en función de si se ha agitado el *micro:bit*. Si se ha agitado, las pulsaciones de los botones sirven para fijar los valores de horas, minutos y segundos de la alarma. Si no se ha agitado las pulsaciones de los botones sirven, como antes de añadir las alarmas, para mostrar las horas o los minutos del cronómetro.

Se podrán introducir como máximo 5 alarmas en una lista de alarmas pendientes, almacenándolas en el formato que consideres más oportuno, pero ordenadas temporalmente de forma que aparezca primero la de hora menor.

Llegado el momento indicado en la primera de las alarmas de la lista de alarmas pendientes, se mostrará alternativamente en la pantalla la imagen de una cara sonriente y una triste durante 5 segundos[7], y se eliminará la alarma de la lista de alarmas pendientes.

Después se volverá al modo normal, mostrándose el valor de los segundos.

Cuando se arranca la aplicación no hay ninguna alarma en lista de alarmas pendientes. Si cuando se intenta añadir una alarma agitando el *micro:bit* ya hay 5 alarmas, se mostrará el mensaje `TOO MANY ALARMS` con `display.show()` y se seguirá mostrando los segundos transcurridos del cronómetro.

La gestión de las alarmas, tanto para añadirlas como para eliminarlas, no debe alterar el funcionamiento del cronómetro, que seguirá contando el tiempo.

La estructura del programa del cronómetro se mantendrá, con un único bucle infinito (`while True:`), dentro del cuál en cada vuelta se realiza toda la funcionalidad descrita.

7. Rúbrica de evaluación

El programa que entregues deberá ser correcto, es decir, deberá cumplir los requisitos funcionales descritos en el enunciado.

Pero eso sólo no basta. El programa deberá estar bien realizado:

- Las variables que utilices deberán estar inicializadas al comienzo del programa.
- Los nombres de las variables han de ser elegidos cuidadosamente.
- No se deben utilizar valores literales en el código, sino variables que estén declaradas al principio del programa, almacenando el valor. Si es una constante, el nombre de la variable deberá estar en mayúsculas.
- El código deberá estar bien sangrado.
- Utiliza las líneas en blanco adecuadamente, para separar grupos de líneas de código que estén relacionadas de otras que no lo estén.
- Añade comentarios razonables que ayuden a entender el código. Los comentarios deberán ser precisos, sin circunloquios. No añadas un comentario si consideras que no es necesario. Suele ser una buena idea comentar grupos de líneas de código que te ha costado programar.
- Debes utilizar funciones para estructurar adecuadamente el código y evitar la repetición del mismo.
- Sólo se puede utilizar el código explicado en clase hasta ahora. No puede utilizarse ningún módulo ni función que no hayamos visto en clase. En caso de duda, pregunta en el foro si puedes utilizar algún elemento del lenguaje no visto.

8. Formato de entrega de la práctica

Entrega el código en la tarea que será habilitada en el Aula Virtual. Deberías tener tres programas: `graf_acelerometros.py` y `cronometro.py`, y `cronometro-alarmas.py` si has hecho la parte opcional.

[7] Ver `https://microbit-micropython.readthedocs.io/en/latest/tutorials/images.html`