Micropython

Preparar ALVIK para MicroPython: Actualizar el firmware de Alvik

Actualizar el firmware significa que nuestro Alvik le instalamos el interpretador de micropython y por lo tanto podremos:
- Programar en código con **Python**
- Programar en bloques con **mBlock**

Si ya tiene el firmware instalado en el Alvik **puedes saltarte esta página**

Con este firmware **no** podemos programar con Arduino IDE

Antes de nada enciende el Alvik con esta precaución :

Primero **nos aseguramos que el Alvik este APAGADO antes de conectarlo con el PC en caso contrario se puede perjudicar la batería**robot-off.png

Licencia CC-BY-NC-SA origen https://courses.arduino.cc/explore-robotics-micropython/lessons/getting-started/

Luego lo **conectamos** por cableconnecting-final.gif

Licencia CC-BY-NC-SA origen https://courses.arduino.cc/explore-robotics-micropython/lessons/getting-started/

Y ahora ya podemos **encender** nuestro Alvik
alvik-on.png

Licencia CC-BY-NC-SA origen https://courses.arduino.cc/explore-robotics-micropython/lessons/getting-started/

Actualizamos el FIRMWARE

Esto lo hacemos una vez, entramos en https://alvikupdate.arduino.cc/ damos a **conectar** y luego **updated** *(si no sabes qué puerto es, desconecta y conecta y te fijas cual aparece)*

> Puede tardar varios ciclos, **paciencia**

2025-03-28 10_18_52-Arduino® Alvik Updater □□.png

> Asegúrate de tener el ALVIK **encendido**

Hasta que sale esta pantalla de éxito

2025-03-26 11_59_36-Arduino® Alvik Updater □□.png

ATENCIÓN ¿Y SI DA PROBLEMAS?

Por ejemplo se ha quedado enganchado, lo has desconectado antes de hora... entonces la solución pasa por utilizar un flasheador más potente

MicroPython Installer

Descargamos el programa y ejecutamos teniendo conectado el ESP32 del Alvik, (no hace falta encender el robot, pues sólo trabajamos con el ESP32) lo detecta y simplemente le damos a Instalar Micropython dentro del chip

Descargable en https://labs.arduino.cc/en/labs/micropython-installer

2024-07-04 19_17_33-MicroPython Installer.png

> Si sigue puñetero y no detecta el Arduino Nano ESP32 tendrás que ponerlo en modo Bootoloader, haz los pasos 1, 2 y 3 de https://libros.catedu.es/books/arduino-alvik/page/preparar-alvik-para-arduino-ide-modo-bootloader y vuelve a intentarlo con el MicroPython Installer

Al acabar de instalar, sale este mensaje :

2025-03-28 10_15_55-MicroPython Installer.png

Aconsejamos **apagar y desconectar totalmente** y volver a conectar (acuérdate que no hay que conectar el ALVIK en el PC con el ALVIK encendido, lo conectas con el PC apagado y luego lo enciendes, tal y como dice arriba del todo)

Entramos en https://alvikupdate.arduino.cc/ damos a **conectar** y luego **updated** *(si no sabes qué puerto es, desconecta y conecta y te fijas cual aparece)*

> AQUÍ VA A TARDAR VARIOS, VARIOS CICLOS, **paciencia, paciencia**

2025-03-28 10_18_52-Arduino® Alvik Updater .png

> Asegúrate de tener el ALVIK **encendido**

Hasta que sale esta pantalla de éxito

2025-03-26 11_59_36-Arduino® Alvik Updater .png

Instalar Micropython

Conceptos previos:
- Los **lenguajes de alto nivel**, es decir el **código**, que es entendible por los humanos (C++, Java, Python...) son textos que se tienen que traducir al lenguaje entendible por el procesador **MCU** (Micro Controler Unit). Este **lenguaje de bajo nivel** que está escrito en **binario** es difícil de entender para los humanos
- El **Compilador** es un programa que **Interpreta** este texto de lenguaje de alto nivel, y lo convierte en lenguaje de bajo nivel
- El Arduino Alvik se puede programar con Arduino IDE como con Micropytno, los dos son de alto nivel

Tanto Micropython como Arduino IDE son lenguajes de tipo CODIGO por lo tanto sólo se aconseja EN SECUNDARIA
Cuando permita lenguaje tipo BLOQUES como Scratch, ya será adecuado para PRIMARIA

¿Dónde se compila Micropython?

Como puedes ver en este vídeo en 21:20 Python se compila **dentro del microcontrolador** es decir, dentro del ESP32. A diferencia con otros lenguajes, como el C++, el ordenador tiene el compilador, y se lo da ya en binario.
2024-07-04 18_44_27-(1) Exploring the Arduino Nano ESP32 _ MicroPython & IoT Cloud - YouTube.pı

Fuente vídeo Exploring the Arduino Nano ESP32 | MicroPython & IoT

¿Y a mi qué más me da?

Pues sí que importa....

Si programas ESP32 con Arduino IDE o Arduino Cloud o con Steamakersblock (que está basado en C++) **te has cargado el compilador Python que has puesto en** "Actualizar firmware" del ESP32 luego si quieres programar en Python, tienes que volver a "Actualizar firmware"
O sea, si pasas de [ArduiIDE] o [Arduino Cloud] o [Steamakersblok] a Micropytho **tienes que volver a instalar el compilador Micropython**

¿Y con esto ya puedo crear mis programas con Micropython?

No, con esto tienes el compilador interpretador dentro del chip, pero necesitas un editor en tu PC y que se comunique con el Micropython del chip

Arduino Lab for Micropython

Tal y como dice la página https://docs.arduino.cc/micropython/ hay dos editores para cargar MicroPython en el Arduino Alvik

- **Arduino Lab for Micropython** https://labs.arduino.cc/en/labs/micropython
- **OpenMW** https://openmv.io/pages/download

Nosotros en este curso elegimos **Arduino Lab for Micropython** por su sencillez y adaptación al Arduino Alvik

Tal y como dice aquí **ES UN PROGRAMA PORTABLE**, es decir, no hay que instalarlo, simplemente descomprimir y ejecutar

2024-07-04 09_56_30-Arduino Labs.png

> Ejecutamos el programa en el lugar donde lo hemos descomprimido (o donde quieras llevarte la carpeta) :

2024-07-04 10_00_12-Arduino.Lab.for.MicroPython-win_x64.png

> Primero **nos aseguramos que el Alvik este APAGADO antes de conectarlo con el PC en caso contrario se puede perjudicar la batería**robot-off.png
> Licencia CC-BY-NC-SA origen https://courses.arduino.cc/explore-robotics-micropython/lessons/getting-started/
>
> Luego lo **conectamos** por cableconnecting-final.gif
> Licencia CC-BY-NC-SA origen https://courses.arduino.cc/explore-robotics-micropython/lessons/getting-started/
>
> Y ahora ya podemos **encender** nuestro Alvik

> alvik-on.png
>
> Licencia CC-BY-NC-SA origen https://courses.arduino.cc/explore-robotics-micropython/lessons/getting-started/

Al ejecutar el programa, vemos:

1. Botón para conectar la placa
2. Ejecutar o para el programa
3. Gestor de fichero
4. Donde escribimos el programa
5. Ventana de estado

IDEOverview.png

Licencia CC-BYSA Fuente https://docs.arduino.cc/micropython/environment/code-editor/

En el gestor de ficheros encontramos

1. **Los archivos que hay en la placa**: Puedes ver y administrar scripts o datos guardados directamente en la placa.
2. **Los archivos que hay en tu equipo**: lo que le permite seleccionar y administrar archivos para cargar o descargar.
3. D**escargar/Subir archivos**: Utiliza esta opción para transferir archivos entre su equipo y la placa. Puedea cargar nuevos scripts o descargar registros de datos de su placa.
4. **Crear archivo/carpeta:** Esta opción le permite crear nuevos archivos o carpetas directamente en la placa o en el directorio de su proyecto, lo que facilita la organización de su código y recursos.

IDEFileManager.png*Licencia CC-BYSA Fuente*

https://docs.arduino.cc/micropython/environment/code-editor/

Hola mundo

Vamos a comenzar con nuestro primer programa en Arduino Lab for MicroPython, el clásico Hola mundo ponemos este programa:

```
from time import sleep

print("Hola mundo, soy un robot que me gusta chatear, ¿cual es tu nombre? ")
student_name = input("Tu nombre : ")
print("Mucho gusto , " + student_name + "! ¿ Cómo quieres llamarme?")
robot_name = input("Mi nombre ? : ")

print(f"{robot_name} es un fantástico nombre. Ya me siento un poco más humano.")

sleep(2) # Use sleep() to make interaction feel more natural
print(f"Okay, {student_name}, voy a ponerte a prueba:")
sleep(2)
print("¿ Has oido hablar que puedo nadar ?")
sleep(4)
print("Je je, es broma..... :D")
sleep(5)
```

Adaptado de https://courses.arduino.cc/explore-robotics-micropython/lessons/getting-started/

Pulsamos a conectar, nos pregunta por el puerto

2024-06-14 23_20_41-Arduino Lab for MicroPython.png

Runeamos y vamos contestando a sus preguntas

2024-06-14 23_22_58-Arduino Lab for MicroPython.png

> **ATENCIÓN** si quieres que se ejecute en el Alvik SIN necesidad de darle al "play" del programa, entonces lo tienes que grabar como main.py en el Alvik entonces se ejecuta automáticamente

Empezando MicroPython de Alvik

> Inspirado en el esquema del tutorial MicroPython Basics autora Francesca Sanfilippo & Karl Söderby

Hemos visto la función **print** visualiza un mensaje en la cónsola :

```
print('Hola mundo !')
```

Podemos introducir una **variable**, frase que contenga el texto, la función **time.sleep(**segundos) que hace una pausa, (para utilizar esta función se necesita importar la librería time con **import time**) y dentro de un **bucle while** que se ejecuta mientras sea verdadero lo que le sigue, en este caso while True se ejecutará siempre:

```
import time
frase = "Hola mundo !!"

while True:
   print(frase)
   time.sleep(1)
```

Aquí se utiliza

- una **función** con **def** una variable contador que en la función se declara **global** de esta manera se puede utilizar dentro de cualquier función del programa (en este caso el programa principal la funcion_contar().
- Vemos la típica operación de cuenta contador = contador + 1
- print visualiza dos cosas, la frase y el contador

```
import time

frase = "Hola mundo "
contador = 0
```

```
def funcion_contar():
    global contador
    contador = contador + 1

while True:
    funcion_contar()
    print(frase, contador)
    time.sleep(1)
```

El resultado:

2024-06-15 07_32_00-Arduino Lab for MicroPython.png

Aquí utilizamos el **condicional if** con su auxiliar **else** y la función **exit** para acabar el programa:

```
import time

frase = "Hola mundo "
contador = 0
maximo = 20

def funcion_contar():
    global contador
    contador = contador + 1

while True:
    funcion_contar()
    if contador>20 :
      exit
    else :
      print(frase, contador)
      time.sleep(1)
```

Lo que provoca que a los 20 finalice

2024-06-15 07_44_37-Arduino Lab for MicroPython.png

Podemos usar en vez de variables numéricas, variables tipo **array** para los bucles :

```
Catedu = ['Javier', 'Santiago', 'Silvia', 'Berta', 'Cristina', 'Nacho', 'Arturo', 'Chefo',
'Vladi', 'Ruben', 'Pablo', 'JuanFran']

def printCatedus():
    for persona in Catedu:
        print(persona)

printCatedus()
```

2024-06-15 07_51_00-Arduino Lab for MicroPython.png

Con esto ya podemos avanzar, pero si quieres

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.

En la siguiente ventana puedes dar al botón *play* y ver el resultado

https://trinket.io/embed/python/47afa56dd668

> 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"

2025-04-05 09_29_39-Editing Page 6 Cadenas _ Librería CATEDU.png

fuente 'Python for Everybody' por Charles R. Severance

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

https://trinket.io/embed/python/5e023b136db8

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`
/=	División y asignación	`x/=3` equivale a `x = x / 3`

Operador	Descripción	Ejemplo
%=	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
&	AND bit a bit	`5&6 # 101 & 110 = 110 = 4`

Operador	Descripción	Ejemplo
|	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`

> Prueba, juega con este código:

https://trinket.io/embed/python/502063b6b44b

> **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, esto es mejor experimentarlo que leerlo :

https://trinket.io/embed/python/34653253eb52

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

https://trinket.io/embed/python3/5dbec1550b

> **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*

https://trinket.io/embed/python/cc1aa3f917a7

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

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

Venga pruébalo !!!

https://trinket.io/embed/python/f797a1eaea48

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

https://trinket.io/embed/python/900fd133a2a9

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. 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

Arduino Alvik API

> Estas instrucciones son específicas del ARDUINO ALVIK

Para acceder a las funciones de Arduino Alvik API tenemos que ejecutar las instrucciones:

```
alvik = ArduinoAlvik()
alvik.begin()
```

Entonces ya podemos usar las siguientes: (extraido de https://docs.arduino.cc/tutorials/alvik/api-overview/)

> Luego veremos en el apartado de programación del Arduino Alvik con código Arduino IDE que utilizaremos una biblioteca #include "Arduino_Alvik.h" que importa prácticamente las mismas funciones, ver https://libros.catedu.es/books/arduino-alvik/page/programas-arduino-ide-sin-iot

FUNCION con sus Inputs	**Outputs**
stop()	para todas las funciones Alvik
is_on()	true si esta encendido false si esta apagado
is_target_reached()	true si ha enviado M o R en el mensaje
get_ack()	last_ack: el valor del último mensaje
stop()	para todas las funciones Alvik
get_orientation()	r: valor de balanceo p: valor de cabeceo y: valor de guiñada

get_accelerations() ver uso en https://libros.catedu.es/books/arduino-alvik/page/programas-de-ejemplo	ax ay az
get_gyros() ver uso en https://libros.catedu.es/books/arduino-alvik/page/programas-de-ejemplo	gx by gz
get_imu()	las 6 anteriores
get_line_sensors()	left center right
brake()	Frena el robot
get_battery_charge()	battery_soc: el % de la batería
get_touch_any()	touch_any es true si se ha apretado cualquier botón
get_touch_ok() get_touch_cancel() get_touch_center() get_touch_up() get_touch_left() get_touch_down() get_touch_right()	touch_ok es true si se ha apretado ok etc... ver ejemplos en https://libros.catedu.es/books/arduino-alvik/page/robotica-para-infantil y en https://libros.catedu.es/books/arduino-alvik/page/mensajes-a-telegram
get_color_raw() get_color_label()	color
get_version() print_status()	versión del firmware para actualizarlo ver https://docs.arduino.cc/tutorials/alvik/user-manual/#how-to-upload-firmware
set_behaviour(behaviour: int)	
rotate(angle: float, unit: str = 'deg', blocking: bool = True)	
move(distance: float, unit: str = 'cm', blocking: bool = True)	

get_wheels_speed(unit: str = 'rpm')	left_wheel_speed: the speed value right_wheel_speed: the speed value
set_wheels_speed(left_speed: float, right_speed: float, unit: str = 'rpm')	
set_wheels_position(left_angle: float, right_angle: float, unit: str = 'deg')	
get_wheels_position(unit: str = 'deg')	angular_velocity
drive(linear_velocity: float, angular_velocity: float, linear_unit: str = 'cm/s',angular_unit: str = 'deg/s')	
get_drive_speed(linear_unit: str = 'cm/s', angular_unit: str = 'deg/s')	linear_velocity: speed of the robot. angular_velocity: speed of the wheels.
reset_pose(x: float, y: float, theta: float, distance_unit: str = 'cm', angle_unit: str = 'deg')	
get_pose(distance_unit: str = 'cm', angle_unit: str = 'deg')	x y theta
set_servo_positions(a_position: int, b_position: int)	
set_builtin_led(value: bool)	
set_illuminator(value: bool)	
color_calibration(background: str = 'white')	
rgb2hsv(r: float, g: float, b: float)	h: hue value s: saturation value v: brightness value
get_color(color_format: str = 'rgb')	r or h g or s b or v
hsv2label(h, s, v)	color label: like "BLACK" or "GREEN", if possible, otherwise return "UNDEFINED"
get_distance(unit: str = 'cm')	lee la distancia del sensor TOF: ver ejemplo en https://libros.catedu.es/books/arduino-alvik/page/evita-obstaculos left_tof: 45° to the left object distance center_left_tof: 22° to the left object distance center_tof: center object distance center_right_tof: 22° to the right object distance right_tof: 45° to the right object distance
get_distance_top(unit: str = 'cm')	top_tof: 45° to the top object distance

get_distance_bottom(unit: str = 'cm')	bottom_tof: 45° to the bottom object distance
on_touch_ok_pressed(callback: callable, args: tuple = ()) on_touch_cancel_pressed(callback: callable, args: tuple = ()) on_touch_center_pressed(callback: callable, args: tuple = ()) on_touch_up_pressed(callback: callable, args: tuple = ()) on_touch_left_pressed(callback: callable, args: tuple = ()) on_touch_down_pressed(callback: callable, args: tuple = ()) on_touch_right_pressed(callback: callable, args: tuple = ())	He intentado hacer programas con estas instrucciones, pero una vez pulsado la tecla, sigue llamando a callback continuamente No veo su utilidad teniendo get_touch

Unidades

- m: centimeters
  mm: millimeters
  m: meters
  inch: inch, 2.54 cm
  in: inch, 2.54 cm

- deg: degrees, example: 1.0 as reference for the other unit. 1 degree is 1/360 of a circle.
  rad: radiant, example: 1 radiant is 180/pi deg.
  rev: revolution, example: 1 rev is 360 deg.
  revolution: same as rev
  perc: percentage, example 1 perc is 3.6 deg.
  %: same as perc

- 'cm/s': centimeters per second
  'mm/s': millimeters per second
  'm/s': meters per second
  'inch/s': inch per second
  'in/s': inch per second

- 'rpm': revolutions per minute, example: 1.0 as reference for the other unit.
  'deg/s': degrees per second, example: 1.0 deg/s is 60.0 deg/min that is 1/6 rpm.
  'rad/s': radiant per second, example: 1.0 rad/s is 60.0 rad/min that is 9.55 rpm.
  'rev/s': revolution per second, example: 1.0 rev/s is 60.0 rev/min that is 60.0 rpm.

> ¿Qué es eso de **bloking**?
> Por ejemplo en rotate(angle: float, unit: str = 'deg', blocking: bool = True)

Si es true, todos los eventos no influyen, es decir el microprocesador esta centrado en esa instrucción
Si es falso, el microprocesador es libre de hacer otra cosa a la vez

Utiliza true si quieres precisión o no quieres que nada interaccione con la acción que estas ejecutando