Traducción y resumen de API de extensión boards

[**POCKET SHIELD 2**](#_a2hd7xbrwm7y)

[Driving Motors 2](#_dtpfs1yap92h)

[Driving Servos 5](#_bwdhknjcor24)

[Driving LED strips 7](#_l6hyoqz7jaqo)

[PIN EXTENSION 11](#_p9o05a74rqc1)

[**mBot2 Shield 13**](#_o5xp5gez5lcs)

[Motion 13](#_unac6f6pav2)

[Driving encoder motors. 17](#_b2k2bwg06mb)

[Driving DC motors 21](#_syyfxztbwscr)

[Driving servos 23](#_6visa5yr2st)

[Driving LED strips 25](#_y80a8w8ghmpq)

[Pin extension 29](#_r9ep88ocue2p)

[**TABLAS RESUMEN DE API 32**](#_hzdlmeio1amv)

[API Pocket Shield 32](#_1e78afdvtxk4)

[Resumen de API Driving motors de Pocket Shield 32](#_w25fkugbvnn)

[Resumen de API Driving servos de Pocket Shield 32](#_1nj06g1v82x2)

[Resumen de API Driving LEDs strips de Pocket Shield 33](#_1wl5k2nfif3k)

[Resumen de API Pin extension de Pocket Shield 33](#_3lemx1w46gou)

[API mBot2 Shield 34](#_so8i1mn05r1l)

[Resumen de API Motion mBot2 Shield 35](#_yio0dqzcgzd4)

[Resumen de API Driving encoder motors mBot2 Shield 36](#_uewyxop7bysh)

[Resumen de API Driving DC motors de mBot2 Shield 37](#_gzihnphuz7ar)

[Resumen de API Driving servos mBot2 Shield 37](#_aoepl4kom2zb)

[Resumen de API Driving LED strips de mBot2 Shield 38](#_qpecr46qnp9q)

[Resumen de API Pin extension mBot2 Shield 39](#_y630ohk581br)

link de la API:

<https://education.makeblock.com/help/mblock-python-editor-apis-for-extension-boards>

raíz web del artículo :

* [Help Hub](https://education.makeblock.com/help/)
* [mBlock Python Editor](https://education.makeblock.com/help/category/mblock-python/)
* [Python API Documentation for Devices](https://education.makeblock.com/help/category/mblock-python/mblock-python-editor-python-api-documentation-for-devices/)
* [Python API Documentation for CyberPi](https://education.makeblock.com/help/category/mblock-python/mblock-python-editor-python-api-documentation-for-devices/mblock-python-editor-python-api-documentation-for-cyberpi/)
* APIs for Extension Boards

**Traducción de la API**

APIs for Extension Boards

Last Updated OnMay 1, 2022

Nota: Las piezas de extensión no se incluyen en el paquete de CyberPi. Para usar sus funciones se debe comprar el kit correspondiente.

Para ayudar a usar la API de manera más eficiente, los escenarios de aplicación se indican de la siguiente manera:

:Este símbolo indica que puede ser usado en modo directo en el editor de Python de mBlock

:este símbolo indica que la API funciona con programación de microPython y puede ser usado en modo carga en el editor de python de mBlock

:este símbolo indica que la API soporta tanto programación en Python 3 como Micro Python y puede ser usado en modo directo y carga en el editor de python de mBlock.

# POCKET SHIELD

Pocket shield suministra potencia a CyberPi permitiendo conectarlo a motores, servomotores, tiras LED. Además, cuenta con puertos analogicos y digitales para el control de módulos Arduino.



## Driving Motors

Los motores de conducción permiten el movimiento de mBot por medio del giro de ruedas controladas por el suministro de voltaje a través de su conexión por puerto al POCKET SHIELD.

Estos cuentan con múltiples funciones en la API para su uso.

**cyberpi.pocket.motor\_add(*potencia*, *puerto*)**

Este método permite modificar la potencia de motor, incrementando o reduciendo, al indicar el puerto donde se encuentra conectado el motor y la potencia a suministrar.

Parámetros:

***potencia***: float, porcentaje de modificación de potencia del motor;

rango de valores: -200–+200%

***puerto***: int or str, representa el puerto donde se encuentra conectado el motor objetivo

rango valores:

tipo str: “all”,”m1”,”m2”,”M1”,”M2”

tipo int: 1 y 2

**cyberpi.pocket.motor\_set(*potencia*, *puerto*)**

Este método permite asignar un valor de potencia específico a un motor indicando el puerto al que se encuentra conectado.

parámetros:

***potencia***: float, representa la potencia que se asignará al motor.

rango de valores: -100 a +100%

Si el número asignado como potencia es positivo, el motor rotara en contra del sentido del reloj

Si el número asignado como potencia es negativo, el motor rotará en el sentido del reloj

Es posible que el motor no gire si se utiliza una potencia muy pequeña

***puerto***: int or str, este valor representa el puerto donde está conectado el motor.

rango valores:

tipo str: “all”,”m1”,”m2”,”M1”,”M2”

tipo int: 1 y 2

**cyberpi.pocket.motor\_get(*puerto*)**

Este método permite consultar la potencia que se está usando en el motor conectado a un puerto específico.

parámetros:

***puerto***: int or str, representa el puerto donde está conectado el motor

rango valores:

tipo str: “all”,”m1”,”m2”,”M1”,”M2”

tipo int: 1 y 2

Retorno:

Esta función retorna un valor, este es de tipo float y su rango se encuentra entre -100 y 100, esto representa el porcentaje de potencia al que está trabajando el motor de ese puerto.

**cyberpi.pocket.motor\_drive(*potencia1*, *potencia2*)**

Este método permite asignar de manera individual a los motores conectados en el puerto M1 y M2.

Parámetros:

***potencia1***: float, este parámetro representa la potencia que se asignará al motor conectado al puerto M1; rango de valores: –100 a +100, representando porcentaje.

***potencia2***: float, este parámetro representa la potencia que se asignará al motor conectado al puerto M2; rango de valores: –100 a +100, representando porcentaje

**cyberpi.pocket.motor\_stop(*puerto*)**

Este método permite asignar la potencia a 0 del motor conectado al puerto indicado en el parámetro.

parámetro:

***puerto***: float or str, este parámetro representa el puerto donde está conectado el motor objetivo

rango valores:

tipo str: “all”,”m1”,”m2”,”M1”,”M2”

tipo int: 1 y 2

Resumen de métodos de Driving motors

| Resumen de métodos API Driving motors de Pocket Shield | | | | |
| --- | --- | --- | --- | --- |
| ref | Nombre | efecto | parámetros | tipo dato  potencia |
| 1 | motor\_add() | modificar potencia a motor | potencia puerto | potencia: float: -200 a 200  puerto: int: 1 y 2  puerto: str:”all,m1,m2,  M1,M2,1,2 |
| 2 | motor\_set() | asignar potencia a motor | potencia puerto | potencia: float: -100 a 100  puerto: int: 1 y 2  puerto: str:all,m1,m2,  M1,M2,1,2 |
| 3 | motor\_get() | leer potencia de motor | puerto | puerto: int: 1 y 2  puerto: str:all,m1,m2,  M1,M2,1,2 |
| 4 | motor\_drive() | asignar potencia a motor en puerto M1 y M2 | potencia1 potencia2 | potencia1: -100 a 100  potencia2: -100 a 100 |
| 5 | motor\_stop() | detener motor | puerto | puerto: int: 1 y 2  puerto: str:all,m1,m2,  M1,M2,1,2 |

## Driving Servos

**cyberpi.pocket.servo\_add(*ángulo*, *puerto*)**

Modificar el ángulo del servo conectado al puerto indicado

Si en el momento de usar este método no se ha usado el método servo\_set(), cuando se ejecute el código, el servo rotará a 20 grados y luego realizará la instrucción de modificar el ángulo.

parámetros:

***angulo***: int, representa la cantidad de grados que el servo girará;

rango de valores: -180 a +180, en grados

***puerto***: int or str, este representa el puerto al que se conecta el servo que se desea controlar

rango valores:

tipo str: “all”,”s1”,”s2”,”S1”,”S2”

tipo int: 1 y 2

**cyberpi.pocket.servo\_set(*ángulo*, *puerto*)**

Este método permite asignar un ángulo al servo conectado en el puerto especificado

parámetros:

***angulo***: int, representa el ángulo que tomara el servo

rango de valores: 0 a 180, en ángulos

***puerto***: int or str, este representa el puerto al que se conecta el servo que se desea controlar

rango valores:

tipo str: “all”,”s1”,”s2”,”S1”,”S2”

tipo int: 1 y 2

**cyberpi.pocket.servo\_get(*puerto*)**

Este método permite consultar al servo del puerto indicado, el ángulo que tiene asignado.

Observación, el servo no detecta su ángulo, por esto, si el método servo \_release se ha ejecutado o el servo ha sido rotado manualmente, el ángulo indicado por el método puede ser inexacto.

parámetro:

***puerto***: int or str, este representa el puerto al que se conecta el servo que se desea controlar

rango valores:

tipo str: “all”,”s1”,”s2”,”S1”,”S2”

tipo int: 1 y 2

retorno

Este método, tras ser ejecutado, retorna un dato tipo entero que va desde 0 a 180 y representa grados.

**cyberpi.pocket.servo\_release(*puerto*)**

Este método permite liberar el ángulo del servo que se encuentra conectado al puerto indicado.

La ejecución de este método significa que el servo liberara su bloqueo de rotación al eje. Esto se mantendrá hasta que se ejecute los métodos servo\_set() o servo\_add()

parámetros:

***puerto***: int or str, este representa el puerto al que se conecta el servo que se desea controlar

rango valores:

tipo str: ”s1”,”s2”,”S1”,”S2”

tipo int: 1 y 2

**cyberpi.pocket.servo\_drive(*angulo1*, *angulo2*)**

Este método permite asignar un ángulo individual a los servos de los puertos S1 y S2

Parameters:

***angulo1***: int, representa el ángulo del servo conectado al puerto S1; rango de valores: 0 a 180, in ángulos

***angulo2***: int, representa el ángulo del servo conectado al puerto S2; rango de valores: 0 a 180, in ángulos

| Resumen de métodos API Servos de Pocket Shield | | | | |
| --- | --- | --- | --- | --- |
| ref | Nombre | efecto | parámetros | tipo dato  angulo |
| 1 | servo\_add() | modificar ángulo de servo | ángulo puerto | angulo: int: -180 a 180  puerto: int: 1 y 2  puerto: str: “all”,”s1”,”s2”,”S1”,”S2” |
| 2 | servo\_set() | asignar ángulo a servo | ángulo y puerto | angulo: int: 0 a 180  puerto: int: 1 y 2  puerto: str: “all”,”s1”,”s2”,”S1”,”S2 |
| 3 | servo\_get() | leer ángulo de servo | puerto | puerto: int: 1 y 2  puerto: str: ”s1”,”s2”,”S1”,”S2 |
| 4 | servo\_release() | desbloquea el eje de rotación | puerto | puerto: int: 1 y 2  puerto: str: ”s1”,”s2”,”S1”,”S2 |
| 5 | servo\_drive() | asignar ángulo a dos servos | angulo1 ángulo2 | angulo1:int: 0 a 180  angulo2:int: 0 a 180 |

## Driving LED strips

**cyberpi.pocket.led\_on(*r*,*g*,*b*, *id*, *port*)**

Este método permite activar, con un color específico, los LED de la tira LED o anillo que se encuentre conectado al puerto indicado.

parámetros:

***r***: int or str

***r***: int , representa la intensidad del color rojo; setting range: 0 a 255

***r***: str, representa el nombre del color o su abreviación; se adjunta los valores posibles

| ref | nombre | abreviación |
| --- | --- | --- |
| 1 | red | r |
| 2 | orange | o |
| 3 | yellow | y |
| 4 | green | g |
| 5 | cyan | c |
| 6 | blue | b |
| 7 | purple | p |
| 8 | white | w |
| 9 | black | b |

***g***: int, representa la intensidad del color verde; rango de valores: 0 a 255

***b***: int, representa la intensidad del color azul; rango de valores: 0 a 255

***id***: int or str; su valor por defecto es all

***id***: str, el único valor válido es all

***id***: int, rango de valores: 1 a 36, representa la posición del LED a iluminar

***puerto***: int or str, representa el puerto donde esta conectada la tira LED

rango valores:

tipo str: “all”,”s1”,”s2”,”S1”,”S2”

tipo int: 1 y 2

**cyberpi.pocket.led\_show(*color*, *puerto* )**

Este método permite asignar el color a cada LED de la tira en un solo comando.

parámetros:

***color***: str, representa la secuencia de LEDs y su respectivo color por asignar.

Los colores de los LEDs se deben entregar en una string con el siguiente formato:

”color1 color2 color3 color4 color5...color36"

Si ingresas más de 36 nombres de colores o abreviaciones, el programa solo considerará los primeros 36

los colores disponibles son los siguientes:

| ref | nombre | abreviación |
| --- | --- | --- |
| 1 | red | r |
| 2 | orange | o |
| 3 | yellow | y |
| 4 | green | g |
| 5 | cyan | c |
| 6 | blue | b |
| 7 | purple | p |
| 8 | white | w |
| 9 | black | b |

***puerto***: int or str, representa el puerto donde esta conectada la tira LED

rango valores:

tipo str: “all”,”s1”,”s2”,”S1”,”S2”

tipo int: 1 y 2

**cyberpi.pocket.led\_move( *pasó*, *ciclo*, *puerto* )**

Este método permite desplazar hacia la derecha los colores de los leds de la tira led, desplazándose cierta cantidad de pasos.

parámetros:

***paso***: int, representa la cantidad de posiciones que los colores se moverán.

rango de valores: -36 a +36.

***ciclo***: int, rango de LEDs moviéndose; rango de valores: 1 a 36

***puerto***: int or str, representa el puerto donde esta conectada la tira LED

rango valores:

tipo str: “all”,”s1”,”s2”,”S1”,”S2”

tipo int: 1 y 2

**cyberpi.pocket.led\_off(*id* = "all", *puerto*)**

Este método permite apagar un led en específico de la tira led o todos.

parámetros:

***id***: int or str; su valor por defecto es all

***id***: int, rango de valores: 1 a 36, representa la posición del LED por apagar

***id***: str, solo permite el valor all

***puerto***: int or str, representa el puerto donde esta conectada la tira LED

rango valores:

tipo str: “all”,”s1”,”s2”,”S1”,”S2”

tipo int: 1 y 2

**cyberpi.pocket.led\_add\_bri( *brillo*, *puerto*)**

Este método permite modificar el brillo de la tira led.

parámetros:

***brillo***: int, representa el brillo que la tira led cambiará; rango de valores: -100 a +100, representa porcentaje.

***puerto***: int or str, representa el puerto donde esta conectada la tira LED

rango valores:

tipo str: “all”,”s1”,”s2”,”S1”,”S2”

tipo int: 1 y 2

**cyberpi.pocket.led\_set\_bri( *brillo, puerto*)**

Este método permite asignar el brillo a la tira LED.

parámetros:

***brillo***: int, representa el brillo de la tira LED; rango de valores: 0 a 100, representa porcentaje

***puerto***: int or str, representa el puerto donde esta conectada la tira LED

rango valores:

tipo str: “all”,”s1”,”s2”,”S1”,”S2”

tipo int: 1 y 2

**cyberpi.pocket.led\_get\_bri( *puerto*)**

Este método permite consultar el brillo que tiene la tira LED conectada en el puerto especificado.

parámetros:

***puerto***: int or str, representa el puerto donde esta conectada la tira LED

rango valores:

tipo str: “all”,”s1”,”s2”,”S1”,”S2”

tipo int: 1 y 2

return

no especifica el tipo de dato de retorno

**resumen de métodos de Driving LEDs strips**

| Resumen de métodos API de Driving LEDs strips de Pocket Shield | | | | |
| --- | --- | --- | --- | --- |
| ref | Nombre | efecto | parámetros | tipo dato  rgb |
| 1 | led\_on() | enciende un led | r  g  b  id  puerto | r: int: 0 a 255  r: str:  g: int: 0 a 255  b: int: 0 a 255  puerto: int: 1 y 2  puerto: str:“all”,”s1”,”s2”,”S1”,”S2” |
| 2 | led\_show() | asigna un color a cada led | color  puerto | color: str:“r”,”g”,”b”, “y”, “c”, “p”, “w”, “o”, “k”  puerto: int: 1 y 2  puerto: str:“all”,”s1”,”s2”,”S1”,”S2” |
| 3 | led\_move() | rota los colores de los LED | paso  ciclo  puerto | paso: int: -36 a 36  ciclo: int: 1 a 36  puerto: int: 1 y 2  puerto: str:“all”,”s1”,”s2”,”S1”,”S2” |
| 4 | led\_off() | apaga uno o todos los LEDs | id  puerto | id: int: 1 a 36  id: str: “all”  puerto: int: 1 y 2  puerto: str:“all”,”s1”,”s2”,”S1”,”S2” |
| 5 | led\_add\_bri() | modifica el brillo de la tira LED | brillo  puerto | brillo: int: -100 a 100  puerto: int: 1 y 2  puerto: str:“all”,”s1”,”s2”,”S1”,”S2” |
| 6 | let\_set\_bri() | asigna brillo a la tira LED | brillo  puerto | brillo: int: 0 a 100  puerto: int: 1 y 2  puerto: str:“all”,”s1”,”s2”,”S1”,”S2” |
| 7 | led\_get\_bri() | obtiene el brillo de la tira LED | puerto | puerto: int: 1 y 2  puerto: str:“all”,”s1”,”s2”,”S1”,”S2” |

## PIN EXTENSION

**cyberpi.pocket.write\_digital( *valor*, *puerto*)**

Este método permite asignar un valor a un puerto indicado o todos

parámetros:

***valor***: representa el valor a escribir en el puerto

***puerto***: int or str, representa el puerto de interés.

rango valores:

tipo str: “all”,”s1”,”s2”,”S1”,”S2”

tipo int: 1 y 2

**cyberpi.pocket.read\_digital(*puerto*)**

Este método permite leer la entrada digital de un puerto.

parámetros:

***puerto***: int or str, representa el puerto de interés.

rango valores:

tipo str: ”s1”,”s2”,”S1”,”S2”

tipo int: 1 y 2

return

Este método recibe de retorno un dato tipo entero que puede ser 1 o 0, representando alto voltaje y bajo voltaje respectivamente.

**cyberpi.pocket.set\_pwm( *trabajo*, *frecuencia,* *puerto*)**

Este método permite simular una salida analógica con señal digital indicando la frecuencia, ciclo de trabajo y puerto.

parámetros:

***ciclo*** de trabajo: int, representa cuánto tiempo la señal está activa por ciclo put; rango de valores: 0 a 100, en porcentaje

***frecuencia***: int, representa la cantidad de veces que se ejecuta la modulación en un segundo; rango de valores: 1 a 2000, en Hz

***puerto***: int or str, representa el puerto de interés.

rango valores:

tipo str: “all”,”s1”,”s2”,”S1”,”S2”

tipo int: 1 y 2

**cyberpi.pocket.read\_analog(*puerto*)**

Este metodo permite leer el voltaje analogico presente en un pin indicado

parámetros:

***puerto***: int or str, representa el puerto de interés.

rango valores:

tipo str: ”s1”,”s2”,”S1”,”S2”

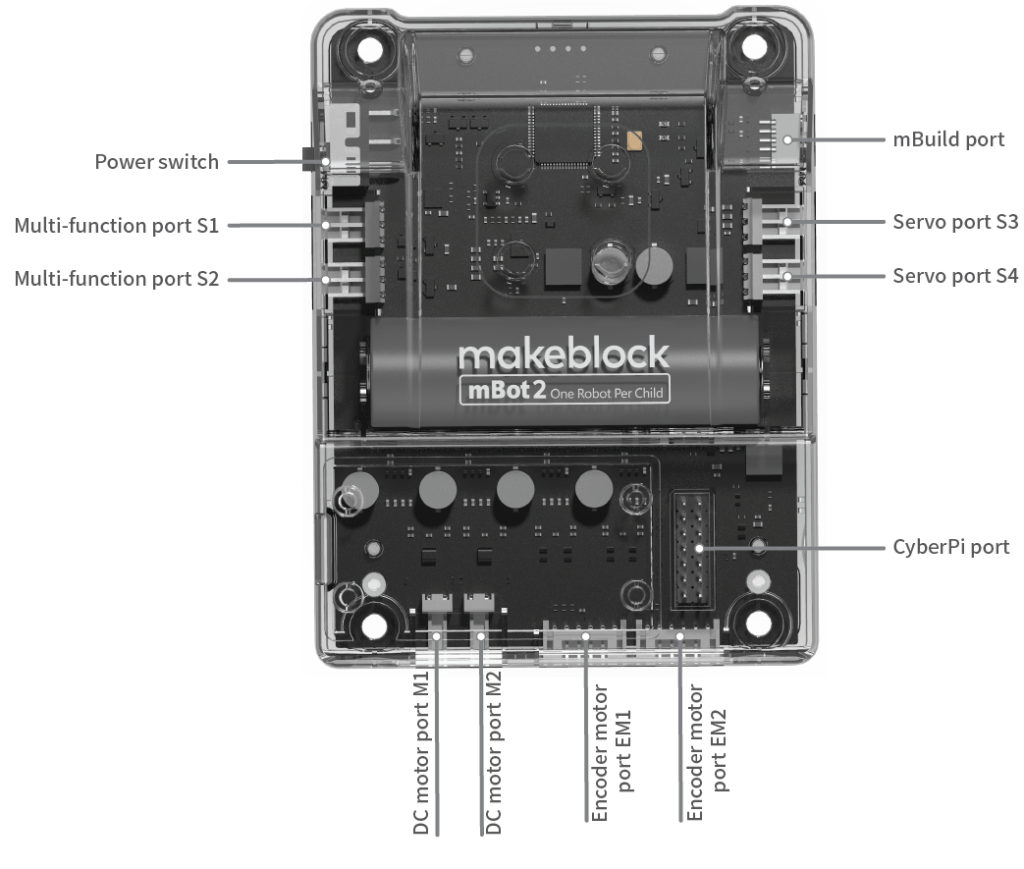
tipo int: 1 y 2

retorno

El retorno de este método es un dato tipo float con rango de 0 a 5 volts

| Resumen de métodos de API Pin extension de Pocket Shield | | | | | |
| --- | --- | --- | --- | --- | --- |
| ref | Nombre | efecto | parámetros | tipo dato  rgb | return |
| 1 | write\_digital() | escribe un dato digital en un puerto | valor  puerto | valor:  puerto: int: 1 y 2  puerto: str:“all”,”s1”,”s2”,”S1”,”S2” |  |
| 2 | read\_digital() | lee un dato digital en un puerto | puerto | puerto: int: 1 y 2  puerto: str:”s1”,”s2”,”S1”,”S2” | retorno: int: 0 o 1 |
| 3 | set\_pwm() | escribe un dato analogico en un puerto | ciclo\_trabajo  frecuencia  puerto | ciclo\_trabajo: int: 0 a 100, porcentaje  frecuencia: int: 1 a 2000, hertz  puerto: int: 1 y 2  puerto: str:“all”,”s1”,”s2”,”S1”,”S2” |  |
| 4 | read\_analog() | lee un dato analogico en un puerto | puerto | puerto: int: 1 y 2  puerto: str:”s1”,”s2”,”S1”,”S2” | retorno: float: 0 a 5, volt |

# mBot2 Shield

****

mBot2 Shield tiene 8 puertos para el control de servomotores y sensores. Tiene dos puertos multifuncionales denominados S1 y S2, también trae dos puertos para conectar motores de corriente directa con los nombres de M1 y M2, dos puertos para la conexión de servomotores S3 y S4, y finalmente, dos puertos para motores tipo encoder EM1 y EM2.

## Motion

La documentación de esta sección permite implementar un control de movimiento sencillo.

Los siguientes métodos le permiten controlar el movimiento de mBot2 solo si se han ensamblado sus motores, ruedas y llantas.

**mbot2.forward(*velocidad* = 50, *tiempo*)**

Este método permite hacer que mBot 2 avance a cierta velocidad por un tiempo determinado

parámetros:

***velocidad***: float, representa la velocidad de giro de los motores encoder; rango de valores –200 a +200 RPM

Si el valor es negativo, mBot 2 retroceda.

***tiempo***: float or str; tiene el valor null por defecto

Si se entrega un dato tipo float al parámetro t, este representa el tiempo por el cual se moverá, con un rango de valores desde 0 al infinito en segundos.

Si no se entrega un dato a este parámetro, asume el valor “null”, y se moverá a la velocidad indicada hasta que se ejecute un comando de parada.

**mbot2.backward(*velocidad* = 50, *tiempo*)**

Este método permite hacer retroceder a mBot 2 a cierta velocidad por cierto tiempo

parámetros:

***velocidad***: float, representa la velocidad de giro del motor encoder; rango de valor: –200 a +200 RPM

Si se entrega un entero negativo, mBot 2 avanzara.

t: float or str, su valor por defecto es “null”.

Si se entrega un float, este representa el tiempo en segundos que mBot 2 se moverá, con rango desde 0 al infinito.

Si se omite o se entrega “null”, mBot 2 se moverá indefinidamente hasta recibir un comando de detención.

**mbot2.turn\_left(*velocidad* = 50, *tiempo*)**

Este método permite hacer girar a cierta velocidad

parámetros:

***velocidad***: float, representa la velocidad a la cual los motores encoder giraran; rango valores: –200 a +200 RPM

Si se entre un número negativo, girara a la derecha.

***tiempo***: float or str, valor por defecto null

Si se entrega un valor float a mBot 2 girara tal cantidad de segundos, rango de valores desde 0 al infinito.

Si se omite el parámetro o se entrega una string “null”, indica que mBot 2 girara indefinidamente hasta recibir un comando de detención.

**mbot2.turn\_right(*velocidad* = 50, *tiempo*)**

Este método permite a mBot 2 girar a la derecha a cierta velocidad por determinado tiempo

parámetros:

***velocidad***: float, representa la velocidad de giro de las ruedas de los motores encoder; rango de valores desde -200 a 200 RPM.

Si se entrega un float negativo, mBot 2 girará a la izquierda.

***tiempo***: float or str, su valor por defecto es “null”

Si t es un float, representa la duración del giro, rango de valores desde 0 al infinito

Si se omite o se entrega una string “null”, girará indefinidamente hasta la llegada de un comando de parada.

**mbot2.straight(*distancia*, *velocidad*= 50)**

Este método permite avanzar cierta cantidad de distancia a una determinada velocidad

parámetros:

***distancia***: float, representa la distancia que avanzara en centímetros.

Si el float es negativo, mBot 2 retrocederá.

***velocidad***: float, representa la velocidad a la que giran las ruedas del motor encoder; rango de valor de 0 a 200 RPM. valor por defecto “50”.

Si se entrega un float negativo, se usará su valor absoluto.

**mbot2.turn(*angulo*, *velocidad*= 50)**

Este método permite girar una cantidad de grados a cierta velocidad.

parámetros:

***angulo***: int, representa la cantidad de grados que girará en sentido del reloj.

Si se entrega un int negativo, girara en sentido contra horario.

***velocidad***: float, representa la velocidad de giro de los motores encoder, rango de valores desde -200 a 200. tiene valor por defecto 50.

Si se entrega un float negativo, girará en sentido anti horario.

**mbot2.drive\_power( *potenciaEM1*, *potenciaEM2* )**

Este método permite asignar potencia a ambos motores encoders por separado (M1 rueda izq y M2 rueda derecha). esto asegura la sincronización de estos motores.

parámetros

***potenciaEM1***: float, representa la potencia del motor encoder M1, rango de valores -100 a +100. representa un porcentaje.

Si es positivo el valor, la rueda rotará en sentido contra horario.

***potenciaEM2*** : float, representa la potencia del motor encoder M2( conectado a la rueda derecha), rango de valores 100 a +100, en porcentaje.

Si es positivo el valor, la rueda rotará en sentido contra horario.

**mbot2.drive\_speed( *EM1\_velocidad*, *EM2\_velocidad*)**

Este método permite asignar la velocidad de rotación de las ruedas de ambos motores encoder, esto permite su sincronización.

parámetros:

***EM1\_velocidad***: float, representa la velocidad de giro de la rueda del motor encoder EM1, rango de valores desde –200 a +200 RPM

Si el valor es positivo, rotará en contra el sentido del reloj.

***EM2\_velocidad***: float, representa la velocidad de giro de la rueda del motor encoder EM2, rango de valores desde -200 a +200 RPM

Si el valor es positivo, rotará en contra del sentido del reloj.

**mbot2.EM\_stop(*puerto* = "all")**

Este método permite detener uno o todos los motores encoder indicando su puerto.

parámetro:

***puerto***: float or str, representa el puerto al cual se encuentra conectado el motor encoder

El valor por defecto es “all”, rango de valores:

rango valores:

tipo str: “all”,”ALL”,”em1”,”em2”,”EM1”,”EM2”

tipo int: 1 y 2

| Resumen de métodos API Motion mBot2 Shield | | | | |
| --- | --- | --- | --- | --- |
| ref | nombre | efecto | parámetros | tipo dato para 1 |
| 1 | forward() | avanzar x tiempo | velocidad  tiempo | velocidad:float: -200 a 200, RPM  tiempo: float: 0 a inf, segundos  tiempo: str: “null”, default = “null” |
| 2 | backward() | retroceder | velocidad  tiempo | velocidad:float: -200 a 200, RPM  tiempo: float: 0 a inf, segundos  tiempo: str: “null”, default = “null” |
| 3 | turn\_left() | giro izquierda | velocidad  tiempo | velocidad:float: -200 a 200, RPM  tiempo: float: 0 a inf, segundos  tiempo: str: “null”, default = “null” |
| 4 | turn\_right() | giro derecha | velocidad  tiempo | velocidad: float: -200 a 200, RPM  tiempo: float: 0 a inf, segundos  tiempo: str: “null”, default = “null” |
| 5 | straight() | avanzar x distancia | distancia  velocidad | distancia: float: 0 a inf, cm  velocidad: float: 0 a 200, RPM, default = 50 |
| 6 | turn() | girar | ángulo  velocidad | angulo: int:  velocidad: float: -200 a 200, default= 50 |
| 7 | drive\_power() | asignar potencia | potencia  potencia | potencia: float: -100 a 100, porcentaje  potencia: float: -100 a 100, porcentaje |
| 8 | drive\_speed() | asignar velocidad | velocidad  velocidad | velocidad: float: -200 a 200, RPM  velocidad: float: -200 a 200, RPM |
| 9 | stop() | detener | puerto | puerto: int: 1 y 2  puerto: str: "all","em1","em2","ALL","EM1","EM2" |

## Driving encoder motors.

mBot 2 está equipado con dos motores codificadores, que proporcionan una potencia de salida fuerte y precisa para mBot 2. Tenga en cuenta que la velocidad de rotación de los motores del codificador puede ser inferior a la establecida si su carga es demasiado pesada.

**mbot2.EM\_set\_power( *potencia*, *puerto*)**

Establece la potencia del motor encoder conectado al puerto especificado

Parámetros:

***potencia***: float , potencia del motor encoder EM1 , que va de –100 a +100, en porcentaje.

Si es un valor positivo, el eje gira en sentido contrario a las agujas del reloj.

***puerto***: float o str, representa el puerto al que está conectado el motor encoder.

El valor predeterminado es "all".

rango valores:

tipo str: “all”,”ALL”,”em1”,”em2”,”EM1”,”EM2”

tipo int: 1 y 2

**mbot2.EM\_set\_speed(*velocidad*, *puerto*)**

Este método permite asignar una velocidad de rotación al motor encoder conectado al puerto especificado.

Parámetros:

***velocidad***: float , representa la velocidad de rotación del motor codificador; rango de valores: –200 a +200 RPM

Si se da un valor positivo, el eje de salida gira en sentido contrario a las agujas del reloj.

***puerto***: float o str, representa el puerto al que está conectado el motor encoder.

El valor predeterminado es "all".

rango valores:

tipo str: “all”,”ALL”,”em1”,”em2”,”EM1”,”EM2”

tipo int: 1 y 2

**mbot2.EM\_turn(*ángulo*, *velocidad*, *puerto*)**

Este método permite girar una cantidad específica de grados en contra del sentido horario la rueda del motor encoder conectado al puerto indicado.

Parámetros:

***ángulo***: int , representa el número de grados que el motor encoder conectado al puerto gira en sentido contrario a las agujas del reloj.

***velocidad***: float , representa la velocidad de rotación del motor codificador especificado, rango de valores –200a +200 RPM.

***puerto***: float o str, representa el puerto al que está conectado el motor encoder,

su valor por defecto es “all”.

rango valores:

tipo str: “all”,”ALL”,”em1”,”em2”,”EM1”,”EM2”

tipo int: 1 y 2

**mbot2.EM\_get\_angle(*puerto*)**

Este método lee el ángulo que se encuentra el motor encoder en sentido contrario a las agujas del reloj.

El ángulo de los motores encoder se reinicia al encender el equipo. El número de grados aumenta cuando el motor encoder gira en sentido contrario a las agujas del reloj y disminuye cuando el motor del codificador gira en el sentido de las agujas del reloj.

Parámetro:

***puerto***: float o str.

rango valores:

tipo str: ”em1”,”em2”,”EM1”,”EM2”

tipo int: 1 y 2

retorno:

Se devuelve un valor int con un rango de valores de -∞a +∞, en grados.

**mbot2.EM\_get\_speed(*puerto*)**

Este método permite leer , en tiempo real, la velocidad de rotación del motor encoder conectado al puerto especificado cuando gira en sentido antihorario

La velocidad de rotación del motor encoder puede ser inferior a la velocidad establecida cuando la carga es demasiado pesada o superior debido a la acción de una fuerza externa.

Parámetro:

***puerto***: float o str.

rango valores:

tipo str: ”em1”,”em2”,”EM1”,”EM2”

tipo int: 1 y 2

return

Este método retorna un entero con rango de valor entre -∞a +∞, en RPM.

**mbot2.EM\_get\_power(*puerto*)**

Este método permite consultar , en tiempo real, la potencia del motor encoder que se encuentra conectado al puerto indicado.

Parámetro:

***puerto***: float o str.

rango valores:

tipo str: ”em1”,”em2”,”EM1”,”EM2”

tipo int: 1 y 2

retorno

Este método retorna un valor tipo int con rango desde -100 a +100, representado porcentaje.

**mbot2.EM\_reset\_angle(*puerto*)**

Este método permite establecer el ángulo de los motores encoder conectados a los puertos indicados

Parámetro:

***puerto***: float o str, representa el puerto al que está conectado el motor encoder. su valor predeterminado es "all".

rango valores:

tipo str: “all”,”ALL”,”em1”,”em2”,”EM1”,”EM2”

tipo int: 1 y 2

**mbot2.EM\_lock(*is\_lock* = False, *puerto*)**

Este método permite habilitar la función de autobloqueo del motor codificador conectado al puerto especificado.

Cuando la función de autobloqueo está habilitada, el motor encoder permanecerá en la posición actual después de su movimiento. Esta función está deshabilitada por defecto.

Parámetros:

***is\_lock***: bool , ya sea para habilitar la función de autobloqueo para el motor encoder especificado, su valor premeditado es “False”.

***puerto***: float o str, representa el puerto al que está conectado el motor del codificador, su valor premeditado es “all”.

rango valores:

tipo str: “all”,”ALL”,”em1”,”em2”,”EM1”,”EM2”

tipo int: 1 y 2

Resumen de métodos de Driving encoder motors.

| Resumen de métodos API Driving encoder motors mBot2 Shield | | | | | |
| --- | --- | --- | --- | --- | --- |
| ref | nombre | efecto | parámetros | tipo par1 | return |
| 1 | EM\_set\_power() | asignar potencia | potencia  puerto | potencia: float: -100 a 100, porcentaje  puerto: float: 1 y 2  puerto: str: “all”,”em1”,”em2”,”ALL”,”EM1”,”EM2” |  |
| 2 | EM\_set\_speed() | asignar velocidad | velocidad  puerto | velocidad: float: -200 a 200, RPM  puerto: float: 1 y 2  puerto: str: “all”,”em1”,”em2”,”ALL”,”EM1”,”EM2” |  |
| 3 | EM\_turn() | asignar rotación | ángulo velocidad  puerto | velocidad: -200 a 200 RPM  puerto: float: 1 y 2  puerto: str: “all”,”em1”,”em2”,”ALL”,”EM1”,”EM2” |  |
| 4 | EM\_get\_angle() | leer ángulo de motor | puerto | puerto: float: 1 y 2  puerto: str: ”em1”,”em2”,”EM1”,”EM2” | retorno: int: -inf a +inf, grados |
| 5 | EM\_get\_speed() | leer velocidad de motor | puerto | puerto: float: 1 y 2  puerto: str: ”em1”,”em2”,”EM1”,”EM2” | retorno: int: -inf a + inf, RPM |
| 6 | EM\_get\_power() | leer potencia de motor | puerto | puerto: float: 1 y 2  puerto: str: ”em1”,”em2”,”EM1”,”EM2” | retorno: int: -100 a +100, porcentaje |
| 7 | EM\_reset\_angle() | reiniciar ángulo de motor | puerto | puerto: float: 1 y 2  puerto: str: “all”,”em1”,”em2”,”ALL”,”EM1”,”EM2” |  |
| 8 | EM\_block() | bloquea el motor | is\_lock  puerto | puerto: float: 1 y 2  puerto: str: “all”,”em1”,”em2”,”ALL”,”EM1”,”EM2” |  |

## Driving DC motors

**mbot2.motor\_add( *potencia*, *puerto*)**

Este método permite cambiar la potencia del motor conectado al puerto indicado

Parámetros:

***potencia***: float , representa la potencia a cambiar del motor, rango de valores: -200 a +200, representa porcentaje.

***puerto***: int o str, representa el puerto al que está conectado el motor.

rango valores:

tipo str: “all”,”ALL”,”em1”,”em2”,”EM1”,”EM2”

tipo int: 1 y 2

**mbot2.motor\_set( *potencia*, *puerto*)**

Permite asignar una velocidad de rotación del motor conectado al puerto especificado

Parámetros:

***potencia***: float , representa la potencia del motor; rango valores: -100 a +100%

Si el valor es positivo, el motor gira en sentido antihorario; si es negativo, rota en sentido horario. Es posible que el motor no funcione si se le da una potencia muy baja.

***puerto***: int o str, representa el puerto al que está conectado el motor.

rango valores:

tipo str: “all”,”ALL”,”em1”,”em2”,”EM1”,”EM2”

tipo int: 1 y 2

**mbot2.motor\_get(*puerto*)**

Este método permite leer la potencia del motor conectado al puerto indicado.

Parámetro:

***puerto***: float o str, representa el puerto al que está conectado el motor,

rango valores:

tipo str: ”em1”,”em2”,”EM1”,”EM2”

tipo int: 1 y 2

return

Este método devuelve un valor tipo float del rango -100 a 100, representa porcentaje.

**mbot2.motor\_drive(*potencia1*, *potencia2*)**

Este método permite asignar potencia a cada motor conectado al puerto M1 y M2, permitiendo su sincronización.

Parámetros:

***potencia1***: float , representa la potencia de salida del motor conectado al puerto M1; rango valores: –100 a +100, representa porcentaje.

***potencia2***: float , representa la potencia de salida del motor conectado al puerto M2; rango de ajuste: –100 a +100, representa porcentaje.

**mbot2.motor\_stop(*puerto*)**

Este método permite detener uno o todos los motores.

Parámetros:

***puerto***: float o str, representa el puerto al que está conectado el motor.

rango valores:

tipo str: “all”,”ALL”,”em1”,”em2”,”EM1”,”EM2”

tipo int: 1 y 2

Resumen de métodos de Driving DC Motors.

| Resumen de métodos API Driving DC motors de mBot2 Shield | | | | | |
| --- | --- | --- | --- | --- | --- |
| ref | nombre | efecto | parámetros | tipo par1 | return |
| 1 | motor\_add() | modifica potencia del motor | potencia  puerto | potencia: float: -200 a 200, porcentaje  puerto: float: 1 y 2  puerto: str:"all","m1","m2","M1","M2" |  |
| 2 | motor\_set() | asigna potencia del motor | potencia  puerto | potencia: float: -100 a 100, porcentaje  puerto: float: 1 y 2  puerto: str:"all","m1","m2","M1","M2" |  |
| 3 | motor\_get() | leer potencia del motor | puerto | puerto: float: 1 y 2  puerto: str:"all","m1","m2","M1","M2" | float: -100 a 100, porcentaje |
| 4 | motor\_drive() | asigna potencia a ambos motores | potencia1  potencia2 | potencia1: float: -100 a 100, porcentaje  potencia2: float: -100 a 100, porcentaje |  |
| 5 | motor\_stop() | detiene motores | puerto | puerto: float: 1 y 2  puerto: str:"all","m1","m2","M1","M2" |  |

## Driving servos

**mbot2.servo\_add(*ángulo*, *puerto*)**

Cambia el ángulo del servo que se encuentra conectado al puerto indicado

Si la API mbot2.servo\_set(angle, port) no se ha utilizado antes de la ejecución de este comando, el servo gira primero a la posición de 20 grados y luego ejecuta el comando.

Parámetros:

***ángulo***: int , representa la cantidad de grados de giro del servo; rango de valores: -180 a +180, en grados

***puerto***: int o str, representa el puerto al que está conectado el servo

rango valores:

tipo str: “all”,”s1”,”s2”,”s3”,”s4”,”S1”,”S2”,”S3”,”S4”

tipo int: 1,2,3 y 4

**mbot2.servo\_set(*ángulo*, *puerto*)**

Este método permite asignar un ángulo al servo conectado que está conectado al puerto indicado.

Parámetros:

***ángulo***: int , representa la posición en grados del servo, desde 0 a 180, en grados

***puerto***: int o str, puerto al que está conectado el servo.

rango valores:

tipo str: “all”,”s1”,”s2”,”s3”,”s4”,”S1”,”S2”,”S3”,”S4”

tipo int: 1,2,3 y 4

**mbot2.servo\_get(*puerto*)**

Este método permite leer el ángulo en que se encuentra el servo que está conectado al puerto indicado.

Si se ha ejecutado mbo2.servo\_release() o se giró manualmente, la lectura del ángulo del servo no sería exacta.

Parámetro:

***puerto***: int o str, representa el puerto al que está conectado el servo.

rango valores:

tipo str: “all”,”s1”,”s2”,”s3”,”s4”,”S1”,”S2”,”S3”,”S4”

tipo int: 1,2,3 y 4

return

El retorno de este método es un dato tipo int con rango de valor desde 0 a 180, en grados.

**mbot2.servo\_release(*puerto*)**

Este método permite liberar el bloqueo al eje de rotación del servo.

Este método dejará el eje liberado hasta que se ejecute mbot2.servo\_add() o mbot2.servo\_set().

Parámetros:

***puerto***: int o str, representa el puerto al que está conectado el servo

rango valores:

tipo str: “all”,”s1”,”s2”,”s3”,”s4”,”S1”,”S2”,”S3”,”S4”

tipo int: 1,2,3 y 4

**mbot2.servo\_drive(*angulo1*, *angulo2*, *angulo3*, *angulo4*)**

Este método permite asignar angulo a múltiples servos a la vez que se encuentren conectados a los puertos S1, S2, S3 y S4, esto permite sincronizar sus movimientos.

Parámetros:

***angulo1***: int , ángulo del servo conectado al puerto S1; rango de ajuste: 0 a 180, en grados

***angulo2***: int , ángulo del servo conectado al puerto S2; rango de ajuste: 0 a 180, en grados

***angulo3***: int , ángulo del servo conectado al puerto S3; rango de ajuste: 0 a 180, en grados

***angulo4***: int , ángulo del servo conectado al puerto S4; rango de ajuste: 0 a 180, en grados

Resumen métodos de Driving servos.

| Resumen de métodos API Driving servos mBot2 Shield | | | | | |
| --- | --- | --- | --- | --- | --- |
| ref | nombre | efecto | parámetros | tipo par1 | return |
| 1 | servo\_add() | modifica ángulo del servo | ángulo, puerto | angulo: int: -180 a 180, grados  puerto: int: 1, 2, 3, 4  puerto: str:“all”,“s1”,“s2”,“s3”,“s4”,“S1”,“S2”,“S3”,“S4” |  |
| 2 | servo\_set() | asigna ángulo al servo | ángulo, puerto | angulo: int: 0 a 180, grados  puerto: int: 1, 2, 3, 4  puerto: str:“all”,“s1”,“s2”,“s3”,“s4”,“S1”,“S2”,“S3”,“S4” |  |
| 3 | servo\_get() | leer ángulo del servo | puerto | puerto: int: 1, 2, 3, 4  puerto: str: “s1”,“s2”,“s3”,“s4”,“S1”,“S2”,“S3”,“S4” | retorno: int: 0 a 180, grados |
| 4 | servo\_release() | libera el bloqueo del eje de rotación | puerto | puerto: int: 1, 2, 3, 4  puerto: str: “s1”,“s2”,“s3”,“s4”,“S1”,“S2”,“S3”,“S4” |  |
| 5 | servo\_drive() | asigna ángulo a multiples servos | angulo1  angulo2  angulo3  angulo4 | angulo1: int: 0 a 180, grados  angulo2: int: 0 a 180, grados  angulo3: int: 0 a 180, grados  angulo4: int: 0 a 180, grados |  |

## Driving LED strips

**mbot2.led\_on(*r*,*g*,*b*, *id* = "all", *port*)**

Este método permite encender los LED en la tira o anillo de LED conectado al puerto indicado con los colores especificados.

Parámetros:

***r***: int o str

***r***: int, representa la intensidad del color rojo; rango de ajuste:0 a 255

***r***: str, representa el nombre completo o abreviatura de un color; a continuación se describen los colores y sus abreviaturas:

r roja

o naranjo

y amarillo

g verde

c cian

b azul

p púrpura

blanco

k negro

***g***: int , representa la intensidad del color verde; rango de ajuste: 0 a 255

***b***: int , representa la intensidad del color azul; rango de ajuste: 0 a 255

***id***: int o str; El valor por defecto es “all”

***id***: str, solo el valor “all” es válido

***id***: int, rango de configuración: 1 a 36, que indica la posición del LED que se encenderá

***puerto***: int o str, representa el puerto al que se conecta la tira o anillo de LED

rango valores:

tipo str: “all”,”s1”,”s2”,”S1”,”S2”

tipo int: 1,2,3 y 4

**mbot2.led\_show(*color*, *puerto*)**

Este método permite asignar color individualmente a cada LED de la tira conectada al puerto indicado.

Parámetros:

***color***: str , representa el color del LED o la secuencia de Colores para cada LED.

Para asignar una secuencia de colores, se debe entregar una string con una secuencia de nombre o abreviatura de colores desde el 1 al 36, separados por espacio.

Si la string que se entrega trae más de 36 colores, se utilizarán solo los primeros 36. Los colores posibles son los siguiente:

red,r

green,g

blue,b

yellow,y

cyan,c

purple,p

white,w

orange,o

black,k

***puerto***: int o str, representa el puerto donde esta conectada la tira o anillo de LED

rango valores:

tipo str: “all”,”s1”,”s2”,”S1”,”S2”

tipo int: 1,2,3 y 4

**mbot2.led\_move( *pasó*, *ciclo*, *puerto* )**

Este método permite trasladar la secuencia de colores de los LED desde izquierda a derecha según las posiciones indicadas.

Parámetros:

***paso***: int , representa el número de posiciones de traslado de los colores de los LEDs; rango de valores: -36 a +36; el valor predeterminado es 1.

ciclo: int , representa el rango del balanceo de color del LED; rango de ajuste: 1 a 36

***puerto***: int o str, representa el puerto donde esta conectada la tira o anillo de LED

rango valores:

tipo str: “all”,”s1”,”s2”,”S1”,”S2”

tipo int: 1,2,3 y 4

**mbot2.led\_off(*id* = "all", *puerto*)**

Este método permite apagar los LED especificados en la tira de LED o el anillo conectado al puerto indicado.

Parámetros:

***id***: int o str; El valor por defecto es “all”

***id***: int, rango valores: 1 a 36, representa la posición del LED por apagar.

***id***: str, solo permite el valor “all”

***puerto***: int o str, representa el puerto al que se conecta la tira o anillo de LED.

rango valores:

tipo str: “all”,”s1”,”s2”,”S1”,”S2”

tipo int: 1,2,3 y 4

**mbot2.led\_add\_bri(*brillo*, *puerto*)**

Este método permite cambiar el brillo de la tira o anillo de LED conectado al puerto indicado.

Parámetros:

***brillo***: int , representa el porcentaje de cambio del brillo de la tira o anillo de LED; rango valores: -100 a +100, en porcentaje

***puerto***: int o str, representa el puerto al que se conecta la tira o anillo de LED.

rango valores:

tipo str: “all”,”s1”,”s2”,”S1”,”S2”

tipo int: 1,2,3 y 4

**mbot2.led\_set\_bri(*brillo*, *puerto*)**

Este método permite asignar el brillo de la tira o anillo de LED conectado al puerto indicado

Parámetros:

***brillo***: int , representa el brillo de la tira o anillo de LED; rango valores: -100 a +100, en porcentaje.

***puerto***: int o str, representa el puerto al que se conecta la tira o anillo de LED

rango valores:

tipo str: “all”,”s1”,”s2”,”S1”,”S2”

tipo int: 1,2,3 y 4

**mbot2.led\_get\_bri(*puerto*)**

Este método permite leer el brillo de la tira o anillo de LED conectado al puerto indicado.

Parámetro:

***puerto***: int o str, puerto al que se conecta la tira o anillo de LED

rango valores:

tipo str: ”s1”,”s2”,”S1”,”S2”

tipo int: 1,2,3 y 4

Resumen de métodos Driving LED strips

| Resumen de métodos Driving LED strips de mBot2 Shield | | | | |
| --- | --- | --- | --- | --- |
| ref | nombre | efecto | parámetros | tipo par1 |
| 1 | led\_on() | enciende LEDs | r  g  b  id  puerto | r: int: 0 a 255  r: str: “r”,”o”,”y”,”g”,”c”,”b”,”p”,”w”,”k”  g: int: 0 a 255  b: int: 0 a 255  id: str: “all”  id: int: 1 a 36  puerto: int: 1 y 2  puerto: str:”all”,”s1”,”s2”,”S1”,”S2” |
| 2 | led\_show() | asigna color individualmente a cada LED | color  puerto | r: int: 0 a 255  r: str: “r”,”o”,”y”,”g”,”c”,”b”,”p”,”w”,”k”  g: int: 0 a 255  b: int: 0 a 255  id: str: “all”  id: int: 1 a 36  puerto: int: 1 y 2  puerto: str:”all”,”s1”,”s2”,”S1”,”S2” |
| 3 | led\_move() | mueve los colores LED | step  cycle  puerto | step: int: -36 a 36  cycle: int: 1 a 36  puerto: int: 1 y 2  puerto: str:”all”,”s1”,”s2”,”S1”,”S2” |
| 4 | led\_off() | apaga uno o varios led o todos | id  puerto | id: str: “all”  id: int: 1 a 36  puerto: int: 1 y 2  puerto: str:”all”,”s1”,”s2”,”S1”,”S2” |
| 5 | led\_add\_bri() | modifica brillo de tira led | brillo  puerto | brillo: int: -100 a 100, porcentaje  puerto: int: 1 y 2  puerto: str:”all”,”s1”,”s2”,”S1”,”S2” |
| 6 | led\_set\_bri() | asigna brillo a tira led | brillo  puerto | brillo: int: -100 a 100, porcentaje  puerto: int: 1 y 2  puerto: str:”all”,”s1”,”s2”,”S1”,”S2” |
| 7 | led\_get\_bri() | consulta el brillo de la tira LED | puerto | puerto: str:”all”,”s1”,”s2”,”S1”,”S2” |

## Pin extension

**mbot2.write\_digital( *valor*, *puerto* )**

Este método permite escribir dato digital de salida por un puerto indicado

Parámetros:

***valor***: int o bool

Rango valores:

True/1: entrada de alto nivel

False/0: entrada de bajo nivel

***puerto***: int o str, representa el puerto donde se encuentran los pines

rango valores:

tipo str: “all”,”s1”,”s2”,”S1”,”S2”

tipo int: 1,2,3 y 4

**mbot2.read\_digital(*puerto*)**

Este método permite consultar el valor digital de un puerto indicado.

Parámetro:

***puerto***: int o str, representa el puerto donde se encuentran los pines

rango valores:

tipo str: “all”,”s1”,”s2”,”S1”,”S2”

tipo int: 1,2,3 y 4

return:

Este método retorna un dato tipo int 0 o 1, donde 0 indica un nivel eléctrico bajo e 1 indica un nivel eléctrico alto.

**mbot2.set\_pwm( *ciclo\_trabajo*, *frecuencia*, *puerto*)**

Este método permite simular una señal analógica en un puerto indicado por medio de una señal pwm con ciclo de trabajo y frecuencia.

Parámetros:

***ciclo\_trabajo***: int , representa el ciclo de trabajo de las señales PWM que se van a emitir; rango de valores: 0 a 100, en porcentaje

***frecuencia***: int , representa la frecuencia de la señal PWM a emitir; rango valores: 1 a 2000, en Hz.

***puerto***: int o str, representa el puerto donde se encuentran los pines.

rango valores:

tipo str: “all”,”s1”,”s2”,”S1”,”S2”

tipo int: 1,2,3 y 4

**mbot2.read\_analog(*puerto*)**

Este método permite leer el voltaje en un pin analógico conectado a un puerto indicado.

Parámetro:

***puerto***: int o str, representa el puerto donde se encuentra conectado el pin.

rango valores:

tipo str: ”s1”,”s2”,”S1”,”S2”

tipo int: 1,2,3 y 4

Retorno:

Este método devuelve un dato tipo float con rango de 0 a 5 volt

Resumen de métodos Pin extension

| Resumen de métodos API Pin extension mBot2 Shield | | | | | |
| --- | --- | --- | --- | --- | --- |
| ref | nombre | efecto | parámetros | tipo par1 | return |
| 1 | write\_digital() | asignar valor digital a puerto | valor  puerto | valor: int: 1 y 2  valor: bool: True y False  puerto: int: 1 y 2  puerto: str:”all”,”s1”,”s2”,”S1”,”S2” |  |
| 2 | read\_digital() | lee el valor digital de un puerto | puerto | puerto: int: 1 y 2  puerto: str:”all”,”s1”,”s2”,”S1”,”S2” | retorno: int: 0 y 1 |
| 3 | set\_pwm() | escribe una señal análoga simulada con pwm | duty  frecuencia  puerto | duty:int: 0 a 100, porcentaje  frecuencia:int:1 a 2000, hertz  puerto: int: 1 y 2  puerto: str:”all”,”s1”,”s2”,”S1”,”S2” |  |
| 4 | read\_analog() | lee una señal análoga de un puerto | puerto | puerto: int: 1 y 2  puerto: str:”all”,”s1”,”s2”,”S1”,”S2” | retorno: float: 0 a 5, volt |

# TABLAS RESUMEN DE API

## API Pocket Shield

| Resumen de API Driving motors de Pocket Shield | | | | | |
| --- | --- | --- | --- | --- | --- |
| ref | Nombre | Efecto | Parámetros | Argumentos | Return |
| 1 | motor\_add() | modificar potencia a motor | potencia puerto | potencia: float: -200 a 200  puerto: int: 1 y 2  puerto: str:”all”,”m1”,”m2”,”M1”,”M2” |  |
| 2 | motor\_set() | asignar potencia a motor | potencia puerto | potencia: float: -100 a 100  puerto: int: 1 y 2  puerto: str:”all”,”m1”,”m2”,”M1”,”M2” |  |
| 3 | motor\_get() | leer potencia de motor | puerto | puerto: int: 1 y 2  puerto: str:”m1”,”m2”,”M1”,”M2” | retorno: float: -100 a 100, porcentaje |
| 4 | motor\_drive() | asignar potencia a motor en puerto M1 y M2 | potencia1 potencia2 | potencia1: -100 a 100  potencia2: -100 a 100 |  |
| 5 | motor\_stop() | detener motor | puerto | puerto: int: 1 y 2  puerto: str:”all”,”m1”,”m2”,”M1”,”M2” |  |

| Resumen de API Driving servos de Pocket Shield | | | | | |
| --- | --- | --- | --- | --- | --- |
| ref | Nombre | Efecto | Parámetros | Argumentos | Return |
| 1 | servo\_add() | modificar ángulo de servo | ángulo puerto | angulo: int: -180 a 180  puerto: int: 1 y 2  puerto: str: “all”,”s1”,”s2”,”S1”,”S2” |  |
| 2 | servo\_set() | asignar ángulo a servo | ángulo y puerto | angulo: int: 0 a 180  puerto: int: 1 y 2  puerto: str: “all”,”s1”,”s2”,”S1”,”S2 |  |
| 3 | servo\_get() | leer ángulo de servo | puerto | puerto: int: 1 y 2  puerto: str: ”s1”,”s2”,”S1”,”S2 | retorno: int: 0 a 180, grados |
| 4 | servo\_release() | desbloquear el eje de rotación | puerto | puerto: int: 1 y 2  puerto: str: ”s1”,”s2”,”S1”,”S2 |  |
| 5 | servo\_drive() | asignar ángulo a dos servos | angulo1 ángulo2 | angulo1:int: 0 a 180  angulo2:int: 0 a 180 |  |

| Resumen de API Driving LEDs strips de Pocket Shield | | | | |
| --- | --- | --- | --- | --- |
| ref | Nombre | Efecto | Parámetros | Argumentos |
| 1 | led\_on() | encender un o todos los LEDs | r  g  b  id  puerto | r: int: 0 a 255  r: str:  g: int: 0 a 255  b: int: 0 a 255  puerto: int: 1 y 2  puerto: str:“all”,”s1”,”s2”,”S1”,”S2” |
| 2 | led\_show() | asignar color a cada led | color  puerto | color: str:“r”,”g”,”b”, “y”, “c”, “p”, “w”, “o”, “k”  puerto: int: 1 y 2  puerto: str:“all”,”s1”,”s2”,”S1”,”S2” |
| 3 | led\_move() | rotar colores de tira LED | paso  ciclo  puerto | paso: int: -36 a 36  ciclo: int: 1 a 36  puerto: int: 1 y 2  puerto: str:“all”,”s1”,”s2”,”S1”,”S2” |
| 4 | led\_off() | apagar uno, varios o todos los LEDs | id  puerto | id: int: 1 a 36  id: str: “all”  puerto: int: 1 y 2  puerto: str:“all”,”s1”,”s2”,”S1”,”S2” |
| 5 | led\_add\_bri() | modificar brillo de la tira LED | brillo  puerto | brillo: int: -100 a 100  puerto: int: 1 y 2  puerto: str:“all”,”s1”,”s2”,”S1”,”S2” |
| 6 | let\_set\_bri() | asignar brillo a tira LED | brillo  puerto | brillo: int: 0 a 100  puerto: int: 1 y 2  puerto: str:“all”,”s1”,”s2”,”S1”,”S2” |
| 7 | led\_get\_bri() | leer brillo de tira LED | puerto | puerto: int: 1 y 2  puerto: str:“all”,”s1”,”s2”,”S1”,”S2” |

| Resumen de API Pin extension de Pocket Shield | | | | | |
| --- | --- | --- | --- | --- | --- |
| ref | Nombre | Efecto | Parámetros | Argumentos | Return |
| 1 | write\_digital() | escribir un dato digital en un puerto | valor  puerto | valor:  puerto: int: 1 y 2  puerto: str:“all”,”s1”,”s2”,”S1”,”S2” |  |
| 2 | read\_digital() | leer un dato digital en un puerto | puerto | puerto: int: 1 y 2  puerto: str:”s1”,”s2”,”S1”,”S2” | retorno: int: 0 o 1 |
| 3 | set\_pwm() | escribir un dato analogico en un puerto | ciclo\_trabajo  frecuencia  puerto | ciclo\_trabajo: int: 0 a 100, porcentaje  frecuencia: int: 1 a 2000, hertz  puerto: int: 1 y 2  puerto: str:“all”,”s1”,”s2”,”S1”,”S2” |  |
| 4 | read\_analog() | leer un dato analogico en un puerto | puerto | puerto: int: 1 y 2  puerto: str:”s1”,”s2”,”S1”,”S2” | retorno: float: 0 a 5, volt |

## API mBot2 Shield

| Resumen de API Motion mBot2 Shield | | | | |
| --- | --- | --- | --- | --- |
| ref | Nombre | Efecto | Parámetros | Argumentos |
| 1 | forward() | avanzar cierto tiempo | velocidad  tiempo | velocidad:float: -200 a 200, RPM  tiempo: float: 0 a inf, segundos  tiempo: str: “null”, default = “null” |
| 2 | backward() | retroceder cierto tiempo | velocidad  tiempo | velocidad:float: -200 a 200, RPM  tiempo: float: 0 a inf, segundos  tiempo: str: “null”, default = “null” |
| 3 | turn\_left() | girar a la izquierda | velocidad  tiempo | velocidad:float: -200 a 200, RPM  tiempo: float: 0 a inf, segundos  tiempo: str: “null”, default = “null” |
| 4 | turn\_right() | girar a la derecha | velocidad  tiempo | velocidad: float: -200 a 200, RPM  tiempo: float: 0 a inf, segundos  tiempo: str: “null”, default = “null” |
| 5 | straight() | avanzar cierta distancia | distancia  velocidad | distancia: float: 0 a inf, cm  velocidad: float: 0 a 200, RPM, default = 50 |
| 6 | turn() | girar | ángulo  velocidad | angulo: int:  velocidad: float: -200 a 200, default= 50 |
| 7 | drive\_power() | asignar potencia a motor | potencia  potencia | potencia: float: -100 a 100, porcentaje  potencia: float: -100 a 100, porcentaje |
| 8 | drive\_speed() | asignar velocidad a motor | velocidad  velocidad | velocidad: float: -200 a 200, RPM  velocidad: float: -200 a 200, RPM |
| 9 | stop() | detener motor | puerto | puerto: int: 1 y 2  puerto: str: "all","em1","em2","ALL","EM1","EM2" |

| Resumen de API Driving encoder motors mBot2 Shield | | | | | |
| --- | --- | --- | --- | --- | --- |
| ref | Nombre | Efecto | Parámetros | Argumentos | Return |
| 1 | EM\_set\_power() | asignar potencia | potencia  puerto | potencia: float: -100 a 100, porcentaje  puerto: float: 1 y 2  puerto: str: “all”,”em1”,”em2”,”ALL”,”EM1”,”EM2” |  |
| 2 | EM\_set\_speed() | asignar velocidad | velocidad  puerto | velocidad: float: -200 a 200, RPM  puerto: float: 1 y 2  puerto: str: “all”,”em1”,”em2”,”ALL”,”EM1”,”EM2” |  |
| 3 | EM\_turn() | asignar rotación | ángulo velocidad  puerto | velocidad: -200 a 200 RPM  puerto: float: 1 y 2  puerto: str: “all”,”em1”,”em2”,”ALL”,”EM1”,”EM2” |  |
| 4 | EM\_get\_angle() | leer ángulo de motor | puerto | puerto: float: 1 y 2  puerto: str: ”em1”,”em2”,”EM1”,”EM2” | retorno: int: -inf a +inf, grados |
| 5 | EM\_get\_speed() | leer velocidad de motor | puerto | puerto: float: 1 y 2  puerto: str: ”em1”,”em2”,”EM1”,”EM2” | retorno: int: -inf a + inf, RPM |
| 6 | EM\_get\_power() | leer potencia de motor | puerto | puerto: float: 1 y 2  puerto: str: ”em1”,”em2”,”EM1”,”EM2” | retorno: int: -100 a +100, porcentaje |
| 7 | EM\_reset\_angle() | reiniciar ángulo de motor | puerto | puerto: float: 1 y 2  puerto: str: “all”,”em1”,”em2”,”ALL”,”EM1”,”EM2” |  |
| 8 | EM\_block() | bloquear motor | is\_lock  puerto | puerto: float: 1 y 2  puerto: str: “all”,”em1”,”em2”,”ALL”,”EM1”,”EM2” |  |

| Resumen de API Driving DC motors de mBot2 Shield | | | | | |
| --- | --- | --- | --- | --- | --- |
| ref | Nombre | Efecto | Parámetros | Argumentos | Return |
| 1 | motor\_add() | modificar potencia a motor | potencia  puerto | potencia: float: -200 a 200, porcentaje  puerto: float: 1 y 2  puerto: str:"all","m1","m2","M1","M2" |  |
| 2 | motor\_set() | asignar potencia a motor | potencia  puerto | potencia: float: -100 a 100, porcentaje  puerto: float: 1 y 2  puerto: str:"all","m1","m2","M1","M2" |  |
| 3 | motor\_get() | leer potencia del motor | puerto | puerto: float: 1 y 2  puerto: str:"all","m1","m2","M1","M2" | float: -100 a 100, porcentaje |
| 4 | motor\_drive() | asignar potencia a ambos motores | potencia1  potencia2 | potencia1: float: -100 a 100, porcentaje  potencia2: float: -100 a 100, porcentaje |  |
| 5 | motor\_stop() | detener motores | puerto | puerto: float: 1 y 2  puerto: str:"all","m1","m2","M1","M2" |  |

| Resumen de API Driving servos mBot2 Shield | | | | | |
| --- | --- | --- | --- | --- | --- |
| ref | Nombre | Efecto | Parámetros | Argumentos | Return |
| 1 | servo\_add() | modificar ángulo del servomotor | ángulo, puerto | angulo: int: -180 a 180, grados  puerto: int: 1, 2, 3, 4  puerto: str:“all”,“s1”,“s2”,“s3”,“s4”,“S1”,“S2”,“S3”,“S4” |  |
| 2 | servo\_set() | asignar ángulo a servomotor | ángulo, puerto | angulo: int: 0 a 180, grados  puerto: int: 1, 2, 3, 4  puerto: str:“all”,“s1”,“s2”,“s3”,“s4”,“S1”,“S2”,“S3”,“S4” |  |
| 3 | servo\_get() | leer ángulo del servomotor | puerto | puerto: int: 1, 2, 3, 4  puerto: str: “s1”,“s2”,“s3”,“s4”,“S1”,“S2”,“S3”,“S4” | retorno: int: 0 a 180, grados |
| 4 | servo\_release() | liberar bloqueo del eje de rotación | puerto | puerto: int: 1, 2, 3, 4  puerto: str: “s1”,“s2”,“s3”,“s4”,“S1”,“S2”,“S3”,“S4” |  |
| 5 | servo\_drive() | asignar ángulo a multiples servomotor | angulo1  angulo2  angulo3  angulo4 | angulo1: int: 0 a 180, grados  angulo2: int: 0 a 180, grados  angulo3: int: 0 a 180, grados  angulo4: int: 0 a 180, grados |  |

| Resumen de API Driving LED strips de mBot2 Shield | | | | |
| --- | --- | --- | --- | --- |
| ref | Nombre | Efecto | Parámetros | Argumentos |
| 1 | led\_on() | encender uno, varios o todos los LEDs | r  g  b  id  puerto | r: int: 0 a 255  r: str: “r”,”o”,”y”,”g”,”c”,”b”,”p”,”w”,”k”  g: int: 0 a 255  b: int: 0 a 255  id: str: “all”  id: int: 1 a 36  puerto: int: 1 y 2  puerto: str:”all”,”s1”,”s2”,”S1”,”S2” |
| 2 | led\_show() | asignar color individualmente a cada LED | color  puerto | r: int: 0 a 255  r: str: “r”,”o”,”y”,”g”,”c”,”b”,”p”,”w”,”k”  g: int: 0 a 255  b: int: 0 a 255  id: str: “all”  id: int: 1 a 36  puerto: int: 1 y 2  puerto: str:”all”,”s1”,”s2”,”S1”,”S2” |
| 3 | led\_move() | mover los colores LED | step  cycle  puerto | step: int: -36 a 36  cycle: int: 1 a 36  puerto: int: 1 y 2  puerto: str:”all”,”s1”,”s2”,”S1”,”S2” |
| 4 | led\_off() | apagar uno o varios led o todos | id  puerto | id: str: “all”  id: int: 1 a 36  puerto: int: 1 y 2  puerto: str:”all”,”s1”,”s2”,”S1”,”S2” |
| 5 | led\_add\_bri() | modificar brillo de tira led | brillo  puerto | brillo: int: -100 a 100, porcentaje  puerto: int: 1 y 2  puerto: str:”all”,”s1”,”s2”,”S1”,”S2” |
| 6 | led\_set\_bri() | asignar brillo de tira led | brillo  puerto | brillo: int: -100 a 100, porcentaje  puerto: int: 1 y 2  puerto: str:”all”,”s1”,”s2”,”S1”,”S2” |
| 7 | led\_get\_bri() | consultar brillo de la tira LED | puerto | puerto: str:”all”,”s1”,”s2”,”S1”,”S2” |

| Resumen de API Pin extension mBot2 Shield | | | | | |
| --- | --- | --- | --- | --- | --- |
| ref | Nombre | Efecto | Parámetros | Argumentos | Retorno |
| 1 | write\_digital() | escribir un valor digital a un puerto | valor  puerto | valor: int: 1 y 2  valor: bool: True y False  puerto: int: 1 y 2  puerto: str: ”all”,”s1”,”s2”,”S1”,”S2” |  |
| 2 | read\_digital() | leer el valor digital de un puerto | puerto | puerto: int: 1 y 2  puerto: str: ”all”,”s1”,”s2”,”S1”,”S2” | retorno: int: 0 y 1 |
| 3 | set\_pwm() | escribir una señal análoga simulada con pwm | duty  frecuencia  puerto | duty: int: 0 a 100, porcentaje  frecuencia:int: 1 a 2000, hertz  puerto: int: 1 y 2  puerto: str: ”all”,”s1”,”s2”,”S1”,”S2” |  |
| 4 | read\_analog() | leer una señal análoga de un puerto | puerto | puerto: int: 1 y 2  puerto: str: ”all”,”s1”,”s2”,”S1”,”S2” | retorno: float: 0 a 5, volt |