**Using Motors**

|  |
| --- |
| **Be sure to be using version 17 or later of ev3dev because significant changes were made in versions 10 and 17.**You can check the kernel version by selecting "About" in Brickman and scrolling down to the "kernel version". If you don't have a compatible version, [**upgrade the kernel before continuing**](http://www.ev3dev.org/docs/tutorials/upgrading-ev3dev/)**.**  The most important change in version 10 is that speed regulation is now always on (except for the run\_direct command) which means **motors are now always controlled by speed\_sp parameter** (except for the run\_direct command). **Basically then, you should avoid using run\_direct and thus never use duty\_cycle\_sp and always use speed\_sp**. When using run\_forever in place of run\_direct you need to be aware that changing speed\_sp while run\_forever is running does NOT have immediate effect whereas changing duty\_cycle\_sp while run\_direct was running DID have immediate effect.  In version 17 a bug was fixed which had caused the motors to behave unreasonably with large values of speed\_sp.  For the EV3 motors, speed\_sp is in degrees per second so if you set speed\_sp to 360 the motor will try to rotate 360° per second or one rotation per second. **For the standard EV3 large motors, a speed\_sp value of 1000 (degrees per second) is roughly equivalent to a duty\_cycle\_sp value of 100 and to a power value of 100 in the standard Lego EV3 software (EV3-G). Therefore with the large motors you shouldalways use speed\_sp values in the range -1000 to +1000.** (Prior to February 2017 I was recommending the use of values in the range -900 to +900 because a bug was causing unpredictable behaviour with values outside this range but that bug was solved with version 17 of ev3dev so as long as you have version 17 or later you can safely use values -1000 to +1000.)  With the standard EV3 **medium**motor it should be safe to use values of speed\_sp up to**1400**but **it is very convenient to assume that the maximum advisable value is 1000**so that the medium motor can be considered to behave just like the large motor.  Note that speed\_sp represents the TARGET speed in degrees per second but motors are of course subject to the laws of physics so the real speed may sometimes not correspond to the requested speed. For example, if you run this code mB.run\_timed(time\_sp=600, speed\_sp=600) then you might expect the motor to turn 0.6s\*600°/s=360°=1 rotation but in reality it will turn significantly less (maybe 15% less) because the inertia of the motor stops the motor from reaching its target speed instantly and therefore for a short period at the beginning of the motion the motor turns less fast than requested. Also, be aware that most EV3 large motors are not quite capable of reaching a speed of 1000 degrees per second so for values of speed\_spabove 900 you might get an actual speed that is a few percent less than that requested.  When *low*values of speed\_sp are used the movements can again differ from what was requested but in this case the motor tends to move *faster*than requested, for reasons that are not yet well understood. For example, when using speed\_sp=100 the motor may turn about 23% faster than requested.  For the official motor documentation click [**HERE**](http://python-ev3dev.readthedocs.io/en/latest/motors.html).  You may want to make one or more motors turn at a given speed   * through a given angle or number of rotations * for a given time * 'forever' (until the motor is stopped by a stop() command later during program execution)   **Turn motor through a given angle**  Use run\_to\_rel\_pos(position\_sp=<angle in degrees>, speed\_sp=<value>). If you want to motor to run backwards then use a negative value for position\_sp rather than for speed\_sp. **Using a negative value for speed\_sp will not work because the sign of speed\_sp is ignored by this command (but not by others)**. Use speed\_sp values between 0 and 1000.  **Example**  To make a large motor on port B turn through 360° at speed 900 and optionally apply a 'hold' (like a strong brake - see later):  #!/usr/bin/env python3  # so that script can be run from Brickman  from ev3dev.ev3 import \*  from time import sleep  m = LargeMotor('outB')  m.run\_to\_rel\_pos(position\_sp=360, speed\_sp=900, stop\_action="hold")  sleep(5)   # Give the motor time to move  **Turn motor for a given time**  Use  run\_timed(time\_sp=<time in milliseconds>, speed\_sp=<value>)  Note that any negative sign for time\_sp is ignored.  **Example**  This example runs a large motor attached to port B backwards for 3 seconds with a 'speed setpoint' set to -750 (equivalent to a power setting of -75 in the standard EV3 software). It will work even if a second large motor is also plugged in to another motor port, since the port letter is specified. Without the last line, the program would end as soon as the motor starts to turn, so you probably wouldn't see it move at all. Waiting for 5 seconds (1+4) ensures that the motor can turn for 3 seconds before the program ends.  #!/usr/bin/env python3  # So program can be run from Brickman  from ev3dev.ev3 import \*  from time import sleep  m = LargeMotor('outB')  **m.run\_timed(time\_sp=3000, speed\_sp=-750)**  print("set speed (speed\_sp) = " + str(m.speed\_sp))  sleep(1)  # it takes a moment for the motor to start moving  print("actual speed = " + str(m.speed))  sleep(4)  The above example prints the value of speed\_sp to the terminal window, or console. I've used a '+' operator to concatenate (join) the text string on the left with the speed\_sp value on the right, but the speed\_sp value is an integer so I had to convert it to a text string with the str() function before the concatenation could take place. I think it is reasonable for newbies to use the '+' operator in this way, but there is another method which is often preferred by more advanced Python programmers - see [**this discussion**](http://programmers.stackexchange.com/questions/304445/why-is-s-better-than-for-concatenation) and [**this page**](http://www.pythonforbeginners.com/concatenation/string-concatenation-and-formatting-in-python). Note as you run the above program that the print statement is run as soon as the motor STARTS to move - the program does not wait until the motor has finished moving before running the next command.  The example also gives a value (m.speed) for the actual speed of the motor, one second after the motor was told to start turning. The one second delay is needed because it takes a moment for the motor to get going - without the delay the actual speed would probably be given as zero because the motor would not yet have begun turning.  Interestingly, the above program also works if the third line is changed to  m = Motor('outB')  but does not work (with a large motor still attached to port B) if this line is used:  m = MediumMotor('outB')  **Turn motor 'forever'**  Use      run\_forever(speed\_sp=<value>)   and stop the motor with stop() or stop(stop\_action=<value>) where <value> can be "coast", "brake" or"hold".  **Example**  #!/usr/bin/env python3  # So program can be run from Brickman  from ev3dev.ev3 import \*  from time import sleep  m = LargeMotor('outB')  m.run\_forever(speed\_sp=900)  sleep(5)  m.stop(stop\_action="hold")  sleep(5)  I show this code because it is brief - in reality you would never use this code because the motor will run for a given time so it would be better to use run\_timed. Real use of run\_forever could be, for example, to start a robot moving then interrupt the motion when an obstacle is detected.  In my example the  stop\_action  is set to  'hold'  which makes an *active*, forceful effort to stop the motor turning. You can alternatively use  stop\_action="brake"  which *passively* and less forcefully tries to stop the motor turning, or you can use  stop\_action="coast" which lets the motor continue to turn until stopped by friction. If you want, you can also specify a value for stop\_action when running other motor commands like  run\_timed()  or  run\_to\_rel\_pos(). In the standard Lego EV3 icon-based software the only options are 'coast' and 'brake'.  The value of stop\_action will be remembered so in the following code, for example, the second stop() command will use stop\_action="coast" . Also, the second run\_forever command will use the remembered value for speed\_sp since none was specified.  #!/usr/bin/env python3  # So program can be run from Brickman  from ev3dev.ev3 import \*  from time   import sleep  m = LargeMotor('outB')  m.run\_forever(speed\_sp=200)   # equivalent to power=20 in EV3-G  sleep(5)  m.stop(stop\_action="coast")  sleep(4)  m.run\_forever()  sleep(5)  m.stop()  sleep(4)  **Wait for completion**  When a command causes a motor to begin a movement program execution does not pause for the movement to complete before continuing. Often we DO want program execution to pause. It is usually possible to use a sleep() command for this but this is not ideal - the script below shows a better way to make the program pause until the movement completes.  #!/usr/bin/env python3  from ev3dev.ev3 import \*  motor = LargeMotor('outC')  motor.run\_timed(time\_sp=3000, speed\_sp=-750, stop\_action='brake')  motor.wait\_while('running')  Sound.beep()  Note that the above method works with all three possible stop\_action values, 'coast', 'brake' and 'hold'. The beep() command is included so that you can hear that program execution was indeed paused until the motor's movement was complete before playing the beep. The wait\_while() command only works in EV3 Python v0.8.0 or later, so be sure to have the latest version. Version 0.8.0 was released in October 2016.  You may sometimes see the following code fragment:  while any(motor.state):      sleep(0.1)  this code is also designed to make program execution pause until the motor completes its movement but it is less satisfactory, largely because it does not work with stop\_action='hold'. This code fragment above should therefore be considered obsolete.  **Stopping the Motors**  You already know that motors can be stopped with the stop() command and that stopping can be done in three ways: coast, brake or hold (for example, to stop a motor called mB: mB.stop(stop\_action='brake') . I think the stop\_action='coast' is probably rarely used. I also think that it would quite rare to change to stop\_actionmode during a script e.g. from brake to hold. Therefore I think that in programs where the stop\_action mode will not change the stop\_action mode should be set just once and as early in the script as possible. However, so not set the stop\_action mode in the same line that assigns the motor to a motor class since that will not work as expected. In other words, do not do this (I've used strikethrough to emphasise that this is wrong). ~~mB=LargeMotor('outB', stop\_action='brake')~~.  Sometimes the motors will continue to run after a (poorly-written?) script terminates. How would you stop the motors in such a case? You could start up the Python3 interpreter and issue a few commands or you could do what I do: keep a special script 'stop-motors.py' to hand for just such cases. My script only works for two large motors called mB and mC plugged into ports B and C, but for me that is often the case. Modify the script if necessary to match your situation.  #!/usr/bin/env python3  from ev3dev.ev3 import \*  mB = LargeMotor('outB')  mC = LargeMotor('outC')  mB.stop()  mC.stop()  # to make extra sure the motors have stopped:  mB.run\_forever(speed\_sp=0)  mC.run\_forever(speed\_sp=0)  **Speed Regulation**  EV3 Python usually uses a feature called 'speed regulation' when using motors. When speed regulation is on, the motor controller will vary the power supplied to the motor to try to maintain the speed specified in speed\_sp which is in degrees per second. It's comparable to the way your car's cruise control will try to maintain a constant speed even if your car goes up or down hills. If speed regulation is off, which since kernel version 10 and later is only the case for run\_direct(), the controller will use the power specified in duty\_cycle\_sp. Don't forget that for the EV3 large motors a power value of 100 in the standard Lego EV3 software (EV3-G) is roughly equivalent to a speed\_sp value of 1000.  **Other Notes**  Note that if you have only one large motor plugged in to the EV3 then you do not need to specify which motor port it is plugged into. The same is true for the medium motor, and even for each type of sensor.  This page has introduced some but not all of the motor commands. For example, it introduced run\_to\_rel\_pos() but not run\_to\_abs\_pos(). To learn about other commands, see the [**official documentation**](http://python-ev3dev.readthedocs.io/en/latest/motors.html). |

**Usando motores**

|  |
| --- |
| **Asegúrese de utilizar la versión 17 o posterior de ev3dev porque se hicieron cambios significativos en las versiones 10 y 17.**Puede verificar la versión del kernel seleccionando "Acerca de" en Brickman y desplazándose hacia abajo a la "versión del kernel". Si no tiene una versión compatible,  [**actualice el kernel antes de continuar**](http://www.ev3dev.org/docs/tutorials/upgrading-ev3dev/)**.**  El cambio más importante en la versión 10 es que la regulación de velocidad ahora está activada (excepto para el comando run\_direct) lo que significa que los  **motores ahora están controlados por el parámetro speed\_sp**(excepto por el comando run\_direct).**Básicamente, entonces, debes evitar usar run\_direct y nunca usar duty\_cycle\_sp y siempre usar speed\_sp**. Cuando se usa run\_forever en lugar de run\_direct, se debe tener en cuenta que cambiar speed\_sp mientras se ejecuta run\_forever NO tiene efecto inmediato, mientras que cambiar duty\_cycle\_sp mientras se ejecutó run\_direct, DID tiene efecto inmediato.  En la versión 17, se corrigió un error que había provocado que los motores se comportaran de manera irrazonable con grandes valores de speed\_sp.  Para los motores EV3, speed\_sp está en grados por segundo, por lo que si configura speed\_sp en 360, el motor intentará girar 360 ° por segundo o una rotación por segundo.**Para los motores grandes estándar EV3, un valor de velocidad\_sp de 1000 (grados por segundo) es aproximadamente equivalente a un valor de duty\_cycle\_sp de 100 y a un valor de potencia de 100 en el software Lego EV3 estándar (EV3-G). Por lo tanto, con los motores grandes, siempre debe usar valores de velocidad\_sp en el rango de -1000 a +1000.**(Antes de febrero de 2017, estaba recomendando el uso de valores en el rango de -900 a +900 porque un error estaba causando un comportamiento impredecible con valores fuera de este rango, pero ese error se solucionó con la versión 17 de ev3dev, siempre y cuando tengas la versión 17 o más tarde, puede usar de manera segura los valores de -1000 a +1000).  Con el motor **mediano** EV3 estándar , debería ser seguro usar valores de speed\_sp de hasta **1400,** pero **es muy conveniente suponer que el valor máximo recomendable es 1000,** por lo que se puede considerar que el motor mediano se comporta como el motor grande .  Tenga en cuenta que speed\_sp representa la velocidad TARGET en grados por segundo, pero los motores están, por supuesto, sujetos a las leyes de la física, por lo que la velocidad real puede no corresponder a la velocidad solicitada. Por ejemplo, si ejecuta este código mB.run\_timed (time\_sp = 600, speed\_sp = 600),entonces puede esperar que el motor gire 0.6s \* 600 ° / s = 360 ° = 1 rotación, pero en realidad se volverá significativamente menor ( tal vez un 15% menos) porque la inercia del motor impide que el motor alcance su velocidad objetivo instantáneamente y, por lo tanto, durante un breve período al comienzo del movimiento, el motor gira menos rápido de lo solicitado. Además, tenga en cuenta que la mayoría de los motores grandes EV3 no son capaces de alcanzar una velocidad de 1000 grados por segundo, por lo que para los valores de speed\_sp por encima de 900, es posible que obtenga una velocidad real que es un poco menor que la solicitada.  Cuando se usan valores *bajos* de speed\_sp , los movimientos pueden diferir de los solicitados, pero en este caso el motor tiende a moverse *más rápido* de lo solicitado, por razones que todavía no se comprenden bien.  Por ejemplo, cuando se usa  speed\_sp = 100, el motor puede volverse un 23% más rápido que lo solicitado.  Para la documentación oficial del motor, haga clic  [**AQUÍ**](http://python-ev3dev.readthedocs.io/en/latest/motors.html) .  Es posible que desee hacer que uno o más motores giren a una velocidad determinada   * a través de un ángulo dado o número de rotaciones * por un tiempo dado * 'para siempre' (hasta que el motor sea detenido por un comando stop () más tarde durante la ejecución del programa)   **Gire el motor a través de un ángulo determinado**  Utilice  run\_to\_rel\_pos (position\_sp = <angle in degrees>, speed\_sp = <value>) . Si desea que el motor funcione hacia atrás, utilice un valor negativo para position\_sp en  lugar de speed\_sp . **El uso de un valor negativo para  speed\_sp  no funcionará porque este comando ignora el signo de  velocidad \_sp(pero no el de otros)** . Use  valores spe ed\_sp entre 0 y 1000 .  **Ejemplo**  Para hacer un motor grande en el puerto B, gire 360 ​​° a una velocidad de 900 y, opcionalmente, aplique una "retención" (como un freno fuerte, véase más adelante):  #! / usr / bin / env python3  # para que el script pueda ejecutarse desde Brickman  de la importación ev3dev.ev3 \*  desde el tiempo de importación de sueño  m = LargeMotor ('outB' )  m.run\_to\_rel\_pos (position\_sp = 360, speed\_sp = 900 , stop\_action = "hold" )  dormir (5) # Darle tiempo al motor para moverse  **Gire el motor por un tiempo determinado**  Utilice   run\_timed (time\_sp = <tiempo en milisegundos>, speed\_sp = <valor>)  Tenga en cuenta que cualquier signo negativo para time\_sp se ignora.  **Ejemplo**  Este ejemplo ejecuta un motor grande conectado al puerto B hacia atrás durante 3 segundos con un "punto de ajuste de velocidad" establecido en -750 (equivalente a una configuración de potencia de -75 en el software estándar EV3). Funcionará incluso si un segundo motor grande también está enchufado a otro puerto del motor, ya que se especifica la letra del puerto. Sin la última línea, el programa terminaría tan pronto como el motor comience a girar, por lo que probablemente no lo verías moverse en absoluto. Esperar 5 segundos (1 + 4) asegura que el motor pueda girar durante 3 segundos antes de que el programa finalice .  #! / usr / bin / env python3  # Entonces el programa puede ejecutarse desde Brickman  de la importación ev3dev.ev3 \*  desde el tiempo de importación de sueño  m = LargeMotor ('outB')  **m.run\_timed (time\_sp = 3000, speed\_sp = -750)**  print ("establecer la velocidad (speed\_sp) =" + str (m.speed\_sp))  sleep (1) # se necesita un momento para que el motor comience a moverse  print ("velocidad real =" + str (m.speed))  dormir (4)  El ejemplo anterior imprime el valor de speed\_sp en la ventana del terminal o consola. He usado un operador '+' para concatenar (unir) la cadena de texto de la izquierda con el  valor de speed\_sp a la derecha, pero el valor de speed\_sp es un entero, así que tuve que convertirlo en una cadena de texto con str () función antes de que la concatenación pudiera tener lugar. Creo que es razonable para los novatos usar el operador '+' de esta manera, pero hay otro método que a menudo es preferido por los programadores más avanzados de Python: mira [**esta discusión**](http://programmers.stackexchange.com/questions/304445/why-is-s-better-than-for-concatenation) y [**esta página**](http://www.pythonforbeginners.com/concatenation/string-concatenation-and-formatting-in-python) . Cuando ejecute el programa anterior, tenga en cuenta que la impresiónLa instrucción se ejecuta tan pronto como el motor COMIENZA a moverse: el programa no espera hasta que el motor haya terminado de moverse antes de ejecutar el siguiente comando.  El ejemplo también da un valor ( m.speed ) para la velocidad real del motor, un segundo después de que se le dijo al motor que comenzara a girar. La demora de un segundo es necesaria porque lleva un momento para que el motor arranque: sin el retraso, la velocidad real probablemente se daría como cero porque el motor aún no habría comenzado a girar.  Curiosamente, el programa anterior también funciona si la tercera línea se cambia a  m = Motor ('outB')  pero no funciona (con un motor grande todavía conectado al puerto B) si se usa esta línea:  m = MediumMotor ('outB')  **Gire el motor 'para siempre'**  Utilice       run\_forever ( speed\_sp = <value> )   y pare el motor con stop () o  stop ( stop\_action = <value>) donde <value> puede ser "coast", "brake" o "hold".  **Ejemplo**  #! / usr / bin / env python3  # Entonces el programa puede ejecutarse desde Brickman  de la importación ev3dev.ev3 \*  desde el tiempo de importación de sueño  m = LargeMotor ('outB')  m.run\_forever (speed\_sp = 900)  dormir (5)  m.stop (stop\_action = "mantener")  dormir (5)  Muestro este código porque es breve; en realidad, nunca usarías este código porque el motor funcionará durante un tiempo dado, por lo que sería mejor usar run\_timed . El uso real de run\_forever podría ser, por ejemplo, iniciar un robot en movimiento y luego interrumpir el movimiento cuando se detecta un obstáculo.  En mi ejemplo,   stop\_action   está configurado como   'hold', lo   que hace un esfuerzo *activo*y contundente para detener el giro del motor. Alternativamente, puede usar   stop\_action = "brake",   que de forma *pasiva* y con menos fuerza intenta detener el giro del motor, o puede usar   stop\_action = "coast", que permite que el motor siga girando hasta que se detenga por fricción. Si lo desea, también puede especificar un valor para stop\_action cuando ejecuta otros comandos del motor como   run\_timed ()   o   run\_to\_rel\_pos () . En el software estándar Lego EV3 basado en iconos, las únicas opciones son 'costa' y 'freno'.  El valor de  stop\_action  se recordará así que en el siguiente código, por ejemplo, el segundo comando stop () usará stop\_action = "coast" . Además, el segundo comando run\_forever usará el valor recordado para  speed\_sp  ya que no se especificó ninguno.  #! / usr / bin / env python3  # Entonces el programa puede ejecutarse desde Brickman  de la importación ev3dev.ev3 \*  desde el tiempo de importación de sueño  m = LargeMotor ('outB')  m.run\_forever (speed\_sp = 200) # equivalente a la potencia = 20 en EV3-G  dormir (5)  m.stop (stop\_action = "costa")  dormir (4)  m.run\_forever ()  dormir (5)  m.stop ()  dormir (4)  **Espera a completar**  Cuando un comando hace que un motor comience, la ejecución del programa de movimiento no se pausa para que el movimiento se complete antes de continuar. A menudo QUEREMOS que pause la ejecución del programa. Por lo general, es posible utilizar un comando sleep () para esto, pero esto no es ideal  : el siguiente script muestra una mejor manera de hacer que el programa se detenga hasta que el movimiento se complete.  #!/usr/bin/env python3  from ev3dev.ev3 import \*  motor = LargeMotor('outC')  motor.run\_timed(time\_sp=3000, speed\_sp=-750, stop\_action = 'freno' )  motor.wait\_while('running')  Sound.beep()  Tenga en cuenta que el método anterior funciona con los tres posibles valores de stop\_action , 'coast', 'brake' y 'hold'. El comando beep () se incluye para que pueda escuchar que la ejecución del programa se pausó hasta que se completó el movimiento del motor antes de reproducir el pitido. El comando wait\_while () solo funciona en EV3 Python  v0.8.0 o posterior, así que asegúrese de tener la última versión. La versión 0.8.0 fue lanzada en octubre de 2016.  A veces puede ver el siguiente fragmento de código:  while any(motor.state):      sleep(0.1)  este código también está diseñado para hacer que la ejecución del programa se detenga hasta que el motor complete su movimiento pero es menos satisfactorio, en gran parte porque no funciona con  stop\_action = 'hold' . Este fragmento de código anterior debe considerarse obsoleto.  **Detener los motores**  Usted ya sabe que los motores pueden detenerse con el comando stop () y que la detención se puede hacer de tres maneras: costa, freno o mantener (por ejemplo, para detener un motor llamado mB: mB.stop (stop\_action = 'brake') Creo que el stop\_action = 'coast' probablemente se use muy poco. También creo que sería bastante raro cambiar al modo stop\_action durante un script, por ejemplo, desde el freno a mantener.  Por lo tanto, creo que en los programas donde el modo stop\_action no cambiará el El modo stop\_action debe establecerse solo una vez y tan pronto como sea posible en el guión. Sin embargo, no configure el stop\_action.modo en la misma línea que asigna el motor a una clase de motor, ya que no funcionará como se esperaba. En otras palabras, no hagas esto (he usado el tachado para enfatizar que esto está mal). ~~mB = LargeMotor ('outB', stop\_action = 'freno')~~ .  A veces, los motores continuarán funcionando después de que finalice un script (¿mal escrito?). ¿Cómo detendría los motores en tal caso? Podría iniciar el intérprete de Python3 y emitir algunos comandos o podría hacer lo que hago: mantener un script especial 'stop-motors.py' a mano para tales casos. Mi script solo funciona para dos motores grandes llamados mB y mC enchufados en los puertos B y C, pero para mí ese es el caso a menudo. Modifique la secuencia de comandos si es necesario para que coincida con su situación.  #!/usr/bin/env python3  from ev3dev.ev3 import \*  mB = LargeMotor('outB')  mC = LargeMotor('outC')  mB.stop()  mC.stop()  # to make extra sure the motors have stopped:  mB.run\_forever(speed\_sp=0)  mC.run\_forever(speed\_sp=0)  **Regulación de velocidad**  EV3 Python usualmente usa una característica llamada 'regulación de velocidad' cuando usa motores.  Cuando la regulación de velocidad está activada , el controlador del motor variará la potencia suministrada al motor para tratar de mantener la velocidad especificada en  speed\_sp, que está en grados por segundo. Es comparable a la forma en que el control de crucero de tu auto intentará mantener una velocidad constante incluso si tu auto sube o baja las montañas. Si la regulación de velocidad está desactivada, lo cual desde kernel versión 10 y posterior es solo el caso de run\_direct () , el controlador usará la potencia especificada en duty\_cycle\_sp . No olvide que para los motores grandes EV3, un valor de potencia de 100 en el software  Lego EV3 estándar (EV3-G) es más o menos equivalente a un speed\_sp valor de 1000  **Otras notas**  Tenga en cuenta que si solo tiene un motor grande enchufado al EV3, entonces no necesita especificar a qué puerto del motor está enchufado. Lo mismo es cierto para el motor mediano, e incluso para cada tipo de sensor.  Esta página ha introducido algunos, pero no todos, los comandos del motor. Por ejemplo, introdujo  run\_to\_rel\_pos ()  pero no  run\_to\_abs\_pos () . Para conocer otros comandos, consulte la [**documentación oficial**](http://python-ev3dev.readthedocs.io/en/latest/motors.html) . |