Lo esencial

Una vez más, permítanme enfatizar que este sitio no pretende enseñarle la programación básica de Python. Hay cientos de cursos gratuitos para ayudarlo a aprender sobre Python estándar y algunos están listados en la página Learn Python . Este sitio existe para ayudar a las personas que ya tienen un conocimiento básico de Python estándar a dar el paso extra de aprender a usar las funciones robóticas que EV3 Python pone a disposición.

Asegúrese de tener una versión de núcleo que incluya -10-ev3dev o superior (un número mayor). Puede verificar la versión del núcleo 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 . Recomiendo encarecidamente que elija la opción 1 en esa página: vuelva a flashear su tarjeta SD. Sin embargo, tenga en cuenta que perderá todos los archivos que se almacenaron en su instalación ev3dev, incluidos el código y los programas instalados.

Inicie el intérprete de Python 3 desde la línea de comando

Si su PC se está comunicando con el EV3 a través de SSH (Secure SHell), entonces puede iniciar el intérprete de Python 3 escribiendo python3 en el indicador de línea de comando. Luego, cuando vea el indicador de Python '>>>', puede ejecutar los comandos de Python uno por uno. Para salir de la Python 3 Tipo de intérprete Ctrl + D .

Lanzar programas de Python desde la línea de comando

Para ejecutar un programa llamado myprogram.py en la carpeta actual en Python 3 (la única versión utilizada en este sitio), escriba python3 myprogram.py en el indicador de línea de comando. Si necesita forzar un programa para detener la línea de comandos y escriba Ctrl + C . Tenga en cuenta que a veces los motores continuarán girando incluso después de que se haya detenido su programa. Tengo un programa especial ' stop\_motors.py ' listo para funcionar en tales casos: consulte la página Uso de motores . Si su secuencia de comandos escribe en la pantalla LCD del EV3, debe ejecutar un comando adicional sudo chvt 6 antes de ejecutar su secuencia de comandos de Python. Esto evita que Brickman pelee con su script para controlar la pantalla LCD. Una vez que su programa ha terminado, escriba sudo chvt 1para devolver el control de la pantalla LCD a Brickman. Consulte la parte inferior de la página de la pantalla LCD para obtener más información al respecto.

Detener un programa que se inició desde la línea de comando

La mayoría de los programas, por supuesto, se detendrán solos. Si lanzó un programa que no se detiene solo, tal vez porque deliberadamente incluyó un bucle infinito, por ejemplo, o porque algo salió mal, forzar al programa a detenerse con Ctrl-C .

Lanzar programas de Python desde el menú de Brickman

Antes de que pueda iniciar un programa de Python desde la interfaz de Brickman en el ladrillo, debe hacer dos cosas (una por programa de Python) :

Hacer el archivo de programa ejecutable

Incluye una instrucción especial llamada 'shebang' como la primera línea de tu programa

Tenga en cuenta que ninguno de estos pasos es necesario si solo desea iniciar su programa desde la línea de comandos, pero es una buena idea hacer los dos pasos para cada archivo de todos modos en caso de que usted (u otra persona) más tarde quiera ejecutar el programa. desde la interfaz de Brickman. Para convertir un archivo de programa llamado myprogram.py en la carpeta actual en un archivo ejecutable, escriba esto en el indicador de línea de comando:

chmod + x myprogram.py

chmod es el comando de Linux 'modo de cambio', y + x es una opción o 'interruptor' que cambia el modo para hacer que el archivo de programa sea ejecutable.

También debe incluir una instrucción especial llamada ' shebang ' como la primera línea de su programa (DEBE ser la primera línea para poder funcionar). El shebang le dice al compilador qué programa usar para abrir el archivo del programa. Dado que todos los programas en este sitio están escritos en Python 3 (y debe hacer lo mismo, ya que Python 2 es obsoleto), necesitamos el shebang para Python 3 que es este: #! / Usr / bin / env python3

He incluido ese shebang como la primera línea en cada programa en este sitio. Como la línea comienza con un carácter hash, se puede suponer que esto es simplemente un comentario que el compilador ignorará cuando se ejecute el programa. ¡Esta sería una suposición falsa porque de hecho las líneas comienzan con #! en realidad son instrucciones especiales llamadas 'shebangs' que SÍ afectan cómo se ejecuta el programa, ya que le dicen al compilador qué programa utilizar para abrir el archivo. También puede encontrar programas que comiencen con #! / Usr / bin / python, que es una instrucción para usar Python 2 para abrir el programa, pero debe favorecer a Python 3, ya que Python 2 es ahora obsoleto. Todos los programas de Python en este sitio están escritos en Python 3 y usan el sheng de Python 3.

Un shebang es comparable a una asociación de archivos en Windows. En Windows puede usar 'Abrir con ...' para especificar que todos los archivos con una cierta extensión de archivo se deben abrir dentro de un programa determinado. Por ejemplo, puede especificar que los programas con la extensión .txt se abran en MS Word. Shebangs tiene un efecto similar al de una asociación de archivos, pero el shebang debe incluirse en cada archivo que debe abrir un determinado programa .

Detener un programa que se inició desde Brickman

La mayoría de los programas, por supuesto, se detendrán solos. Si lanzó un programa que no se detiene solo, tal vez porque deliberadamente incluyó un bucle infinito, por ejemplo, o porque algo salió mal, forzará al programa a detenerse presionando durante mucho tiempo el botón Atrás .

ev3 o auto? Cuando te encuentres con los programas EV3 Python en Internet, notarás que algunos de ellos comienzan con la importación ev3dev. ev3 como ev3 y otros comienzan con ev3dev. importación automática \*

Ambas declaraciones tienen objetivos similares: hacer que las funciones robóticas contenidas en la biblioteca ev3dev.ev3 estén disponibles para su script. La diferencia entre auto y ev3 (en negrita arriba) es que ev3 funciona solo en el ev3 mientras que auto DEBERÍA detectar automáticamente si el código se está ejecutando en el ev3 o en BrickPi (una interfaz que permite que Raspberry Pi funcione con hardware EV3) y utiliza automáticamente la biblioteca correcta. Por lo tanto, es tentador recomendar que siempre use auto en lugar de ev3por lo que es muy probable que su código sea compatible tanto con el BrickPi como con el ev3. SIN EMBARGO, David Lechner, uno de los principales contribuidores al proyecto ev3dev, dice (desde septiembre de 2016) que 'auto' está realmente roto y solo funciona para EV3 y BrickPi en el modelo 1 de Raspberry Pi. No funcionará en ninguna otra combinación de dispositivos. Además, usar auto significa que la función autocompletar no funcionará en PyCharm, el IDE de Python que recomiendo, por lo que hasta que se corrija el auto (si es que lo hago ) debo recomendar que uses ev3 en lugar de auto . Los usuarios de hardware que no sea EV3 (PiStorms, BrickPi, EVB) tendrán que cambiar la línea de importación o de lo contrario el programa no funcionará. Los usuarios de BrickPI usarán

de la importación ev3dev.brickpi \*

El principal colaborador de ev3dev, Denis Demidov, agrega que otros usuarios (PiStorms, EVB ...) estarán restringidos a la funcionalidad central (universal) y deberían usar

de la importación ev3dev.core \*

Otra diferencia entre las formas de importación ev3dev. ev3 como ev3

y de ev3dev. auto import \* it que una declaración usa la forma import ???? como ???? mientras que el otro usa desde ????? importar \* . En este sitio, dirigido a principiantes, uso y recomiendo el formulario de ????? import \*, ya que permite escribir código más corto y más legible. Por ejemplo, le permite escribir ts = TouchSensor () en lugar de tener que escribir ts = ev3 .TouchSensor () . Los codificadores expertos que trabajan en programas complejos a menudo preferirán la otra forma, por razones que no explicaré aquí. Para resumir, le recomiendo que use la declaración de ev3dev.ev3 import \* en sus programas, tal como lo hago siempre en este sitio.

Basics

Once again, let me stress that this site is not intended to teach you basic Python programming. There are hundreds of free courses to help you learn about standard Python and some are listed on the Learn Python page. This site exists to help people who already have a basic knowledge of standard Python to make the extra step of learning how to use the robotic functions that EV3 Python makes available.

Make sure that you have a kernel version that includes -10-ev3dev or higher (a larger number). 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. I strongly recommend that you choose option 1 on that page: re-flash your SD card. However, be aware that you will lose any files that were stored in your ev3dev installation, including code and installed programs.

Start the Python 3 interpreter from the command line

If your PC is communicating with the EV3 via SSH (Secure SHell) then you can start up the Python 3 interpreter by typing python3 at the command line prompt. Then, when you see the Python prompt '>>>' you can run Python commands one by one. To exit the Python 3 interpreter type Ctrl+D.

Launch Python programs from the command line

To run a program called myprogram.py in the current folder in Python 3 (the only version used on this site) type python3 myprogram.py at the command line prompt. If you need to force a program to stop from the command line then type Ctrl+C. Note that sometimes motors will continue to turn even after your program has stopped - I keep a special program 'stop\_motors.py' ready to run in such cases - see the Using Motors page. If your script writes to the EV3's LCD screen you need to run an additional command sudo chvt 6 before you run your Python script. This stops Brickman from fighting with your script for control of the LCD screen. Once your program has terminated, type sudo chvt 1 to give control of the LCD back to Brickman. See the bottom of the LCD screen page for more on this.

Stopping a program that was launched from the command line

Most programs will of course stop by themselves. If you launched a program that does not stop by itself, perhaps because you deliberately included an infinite loop, for example, or because something went wrong, force the program to stop with Ctrl-C.

Launch Python programs from the Brickman menu

Before you can launch a Python program from the Brickman interface on the brick you must do two things (once per Python program):

Make the program file executable

Include a special instruction called a 'shebang' as the first line of your program

Note that neither of these steps is necessary if you only ever want to launch your program from the command line, but it is a good idea to do both steps for each file anyway in case you (or someone else) later wants to run the program from the Brickman interface. To make a program file called myprogram.py in the current folder into an executable file type this at the command line prompt:

chmod +x myprogram.py

chmod is the Linux 'change mode' command, and +x is an option or 'switch' that changes the mode to make the program file executable.

You also need to include a special instruction called a 'shebang' as the first line of your program (it MUST be the first line in order to work). The shebang tells the compiler what program to use to open the program file. Since all the programs on this site are written in Python 3 (and you should do the same since Python 2 is obsolescent), we need the shebang for Python 3 which is this: #!/usr/bin/env python3

I have included that shebang as the first line in every program on this site. Since the line begins with a hash character you could assume that this is simply a comment that will be ignored by the compiler when the program is run. This would be a false assumption because in fact lines beginning with #! are actually special instructions called 'shebangs' that DO effect how the program runs since they tell the compiler which program to use to open the file. You may also come across programs beginning with #!/usr/bin/python which is an instruction to use Python 2 to open the program but you should favour Python 3 since Python 2 is now obsolescent. All the Python programs on this site are written in Python 3 and use the Python 3 shebang.

A shebang is comparable to a file association in Windows. In Windows you can use 'Open with...' to specify that all files with a certain file extension are to be opened within a certain program. For example, you could specify that programs with the .txt extension are to be opened in MS Word. Shebangs have a similar effect as a file association but the shebang must be included in every file that is to be opened by a certain program.

Stopping a program that was launched from Brickman

Most programs will of course stop by themselves. If you launched a program that does not stop by itself, perhaps because you deliberately included an infinite loop, for example, or because something went wrong, force the program to stop by long-pressing the Back button.

ev3 or auto? As you come across EV3 Python programs on the internet, you will notice that some of them start with import ev3dev.ev3 as ev3 and others start with from ev3dev.auto import \*

Both these statements have similar aims: to make the robotic functions contained in the ev3dev.ev3 library available for use in your script. The difference between auto and ev3 (in bold above) is that ev3 works only on the ev3 whereas auto SHOULD automatically detect whether the code is running on the ev3 or on BrickPi (an interface that allows the Raspberry Pi to work with EV3 hardware) and automatically use the correct library. Therefore it's tempting to recommend that you always use auto rather than ev3 so that there is a good chance that your code will be compatible with the BrickPi as well as the ev3. HOWEVER David Lechner, a top contributor to the ev3dev project, says (as of September 2016) that 'auto' is actually broken and only works for EV3 and BrickPi on Raspberry Pi model 1. It will not work on any other combination of devices. Also, using auto means that autocomplete will not work in PyCharm, the Python IDE that I recommend, so until auto is fixed (if ever) I have to recommend that you use ev3 rather than auto. Users of non-EV3 hardware (PiStorms, BrickPi, EVB) will have to change the import line or else the program won't work. BrickPI users will use

from ev3dev.brickpi import \*

Top ev3dev contributor Denis Demidov adds that other users (PiStorms, EVB...) will be restricted to core (universal) functionality and should use

from ev3dev.core import \*

Another difference between the forms import ev3dev.ev3 as ev3

and from ev3dev.auto import \* it that one statement uses the form import ???? as ???? while the other uses from ????? import \*. In this site, aimed at beginners, I use and recommend the form from ????? import \* since it allows one to write shorter, more legible code. For example, it allows you to write ts=TouchSensor() instead of having to write ts=ev3.TouchSensor(). Expert coders working on complex programs would often prefer the other form, for reasons that I will not explain here. To summarise, I recommend that you use the statement from ev3dev.ev3 import \* in your programs, just as I always do on this site.

**Basics**

|  |
| --- |
| Once again, let me stress that this site is not intended to teach you basic Python programming. There are hundreds of free courses to help you learn about standard Python and some are listed on the [**Learn Python page**](https://sites.google.com/site/ev3python/learn_python). This site exists to help people who already have a basic knowledge of standard Python to make the extra step of learning how to use the robotic functions that EV3 Python makes available.  Make sure that you have a kernel version that includes -10-ev3dev or higher (a larger number). 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/)**.**I strongly recommend that you choose option 1 on that page: re-flash your SD card. However, be aware that you will lose any files that were stored in your ev3dev installation, including code and installed programs.  **Start the Python 3 interpreter from the command line**  If your PC is communicating with the EV3 via SSH (Secure SHell) then you can start up the Python 3 interpreter by typing **python3** at the command line prompt. Then, when you see the Python prompt '>>>' you can run Python commands one by one. To exit the Python 3 interpreter type Ctrl+D.  **Launch Python programs from the command line**  To run a program called myprogram.py in the current folder in Python 3 (the only version used on this site) type **python3 myprogram.py** at the command line prompt. If you need to force a program to stop from the command line then type Ctrl+C. Note that sometimes motors will continue to turn even after your program has stopped - I keep a special program '**stop\_motors.py**' ready to run in such cases - see the [**Using Motors page**](https://sites.google.com/site/ev3python/learn_ev3_python/using-motors). If your script writes to the EV3's LCD screen you need to run an additional command **sudo chvt 6** before you run your Python script. This stops Brickman from fighting with your script for control of the LCD screen. Once your program has terminated, type **sudo chvt 1** to give control of the LCD back to Brickman. See the bottom of the [**LCD screen page**](https://sites.google.com/site/ev3python/learn_ev3_python/screen) for more on this.  **Stopping a program that was launched from the command line**  Most programs will of course stop by themselves. If you launched a program that does not stop by itself, perhaps because you deliberately included an infinite loop, for example, or because something went wrong, force the program to stop with Ctrl-C.  **Launch Python programs from the Brickman menu**  Before you can launch a Python program from the Brickman interface on the brick you must do two things (once per Python program):   * Make the program file executable * Include a special instruction called a 'shebang' as the first line of your program   Note that neither of these steps is necessary if you only ever want to launch your program from the command line, but it is a good idea to do both steps for each file anyway in case you (or someone else) later wants to run the program from the Brickman interface. To make a program file called myprogram.py in the current folder into an executable file type this at the command line prompt:  **chmod +x myprogram.py**  **chmod** is the Linux 'change mode' command, and **+x** is an option or 'switch' that changes the mode to make the program file executable.  You also need to include a special instruction called a '**shebang**' as the first line of your program (it MUST be the first line in order to work). The shebang tells the compiler what program to use to open the program file. Since all the programs on this site are written in Python 3 (and you should do the same since Python 2 is obsolescent), we need the shebang for Python 3 which is this:  **#!/usr/bin/env python3**  I have included that shebang as the first line in every program on this site. Since the line begins with a hash character you could assume that this is simply a comment that will be ignored by the compiler when the program is run. This would be a false assumption because in fact lines beginning with #! are actually special instructions called 'shebangs' that DO effect how the program runs since they tell the compiler which program to use to open the file. You may also come across programs beginning with #!/usr/bin/python which is an instruction to use Python 2 to open the program but you should favour Python 3 since Python 2 is now obsolescent. All the Python programs on this site are written in Python 3 and use the Python 3 shebang.  A shebang is comparable to a file association in Windows. In Windows you can use 'Open with...' to specify that all files with a certain file extension are to be opened within a certain program. For example, you could specify that programs with the .txt extension are to be opened in MS Word. Shebangs have a similar effect as a file association but *the shebang must be included in every file that is to be opened by a certain program*.    **Stopping a program that was launched from Brickman**  Most programs will of course stop by themselves. If you launched a program that does not stop by itself, perhaps because you deliberately included an infinite loop, for example, or because something went wrong, force the program to stop by long-pressing the Back button.  **ev3 or auto?** As you come across EV3 Python programs on the internet, you will notice that some of them start with import ev3dev.**ev3** as ev3 and others start withfrom ev3dev.**auto**import \*  Both these statements have similar aims: to make the robotic functions contained in the ev3dev.ev3 library available for use in your script. The difference between auto and ev3 (in bold above) is that ev3 works only on the ev3 whereas auto SHOULD automatically detect whether the code is running on the ev3 or on BrickPi (an interface that allows the Raspberry Pi to work with EV3 hardware) and automatically use the correct library. Therefore it's tempting to recommend that you always use auto rather than ev3 so that there is a good chance that your code will be compatible with the BrickPi as well as the ev3. HOWEVER David Lechner, a top contributor to the ev3dev project, says (as of September 2016) that 'auto' is actually broken and only works for EV3 and BrickPi on Raspberry Pi model 1. It will not work on any other combination of devices. Also, using auto means that autocomplete will not work in PyCharm, the Python IDE that I recommend, so until auto is fixed (if ever) **I have to recommend that you use ev3 rather than auto**. Users of non-EV3 hardware (PiStorms, BrickPi, EVB) will have to change the import line or else the program won't work. BrickPI users will use  from ev3dev.brickpi import \*  Top ev3dev contributor Denis Demidov adds that other users (PiStorms, EVB...) will be restricted to core (universal) functionality and should use  from ev3dev.core import \*  Another difference between the forms import ev3dev.**ev3** as ev3  and from ev3dev.**auto**import \* it that one statement uses the form import ???? as ???? while the other uses from ????? import \*. In this site, aimed at beginners, I use and recommend the form from ????? import \* since it allows one to write shorter, more legible code. For example, it allows you to write ts=TouchSensor() instead of having to write ts=**ev3**.TouchSensor(). Expert coders working on complex programs would often prefer the other form, for reasons that I will not explain here. To summarise, I recommend that you use the statement **from ev3dev.ev3 import \*** in your programs, just as I always do on this site. |