**LCD Screen**

|  |
| --- |
| Working with the LCD screen is rather complicated and in any case the EV3's screen is small and hard to see (no backlight) so I would suggest that working with the screen should be a low priority for you as you learn EV3 Python.  The EV3 has a 178 x 128 pixels monochrome (black and white) LCD. The coordinates of the top-left pixel are (0, 0) and the coordinates of the bottom-right pixel are (177, 127).  To interact with the LCD you will mainly be using a powerful graphics library called **Pillow**which is part of standard Python rather than something unique to EV3 Python. Note that Pillow evolved out of the Python Image Library (**PIL**). There is [**some documentation on the EV3 Python bindings page**](http://python-ev3dev.readthedocs.io/en/latest/other.html#screen) but you will find much more detailed documentation in the official Pillow documentation [**HERE**](https://pillow.readthedocs.io/en/3.3.x/), especially the [**documentation on the ImageDraw module**](https://pillow.readthedocs.io/en/3.3.x/reference/ImageDraw.html). The ImageDraw module provides the following methods:   * arc(xy, start, end, fill=None) * bitmap(xy, bitmap, fill=None) * chord(xy, start, end, fill=None, outline=None) * ellipse(xy, fill=None, outline=None) * line(xy, fill=None, width=0) * pieslice(xy, start, end, fill=None, outline=None) * point(xy, fill=None) * polygon(xy, fill=None, outline=None) * rectangle(xy, fill=None, outline=None) * text(xy, text, fill=None, font=None, anchor=None, spacing=0, align="left") * ~~multiline\_text~~(xy, text, fill=None, font=None, anchor=None, spacing=0, align="left") * textsize(text, font=None, spacing=0) * multiline\_textsize(text, font=None, spacing=0)   You may find [**this tutorial**](https://pillow.readthedocs.io/en/3.3.x/handbook/tutorial.html)to be a good place to start.  Multiline\_text does not seem to be available currently in EV3 Python and there may be others that are not available.  **Warning**: EV3 Python scripts using the LCD screen can easily be run successfully from within the Brickman interface but if you want to run such scripts from within an SSH session (PuTTY or MobaXTerm, for example), you should follow the instructions at the bottom of this page. If you run code that addresses the screen from the SSH session without following the instructions at the bottom of this page then you may experience two problems:   * the Brickman display may reappear before your script stops running, even after a second or less * or you might have the opposite problem: when the script stops it may take a minute or two before the the screen reverts back to the Brickman interface   **Smiley / Grumpy**  This script draws two circles for eyes then an arc (in this case half an ellipse) which alternates between opening upwards (a smile) and opening downwards (grumpy). **Warning**: Do NOT run this script from within Brickman because it runs forever and therefore you need to stop it with a long press on the Back button but this does NOT properly stop the program, presumably due to some kind of bug in Brickman. You CAN successfully run this program from SSH, stopping it with Ctrl-C, but don't forget to read the note at the end of this page.  #!/usr/bin/env python3  from time import sleep  from ev3dev.ev3 import \*  lcd = Screen()  smile = True  while True:      lcd.clear()      # lcd.draw returns a PIL.ImageDraw handle      lcd.draw.ellipse(( 20, 20,  60, 60))      lcd.draw.ellipse((118, 20, 158, 60))      if smile:          lcd.draw.arc((20, 80, 158, 100), 0, 180)      else:          lcd.draw.arc((20, 80, 158, 100), 180, 360)      smile = not smile  # toggle between True and False      # Update lcd display      lcd.update() # Applies pending changes to the screen.      # Nothing will be drawn on the lcd screen      # until this function is called.      sleep(1)  Note that the screen.draw.ellipse() command has parameters that are the top-left corner and bottom-right corner of a bounding box (an imaginary box that would contain the ellipse). The  screen.draw.arc() command needs the same pair of coordinates and also needs a starting angle and an ending angle, both in degrees. Angles are measured from 3 o’clock, increasing clockwise.  **Display Text**  This script will display 'Hello, World' as small white text in a black rectangle with 'THIS TEXT IS BLACK' displayed in black in the lower half of the screen. The coordinates in the parameter of the text() function refer to the top left corner of the text.  #!/usr/bin/env python3  from ev3dev.ev3 import \*  from time import sleep  lcd = Screen()  **lcd.draw.rectangle((0,0,177,40), fill='black')**  **lcd.draw.text((48,13),'Hello, world.', fill='white')**  **lcd.draw.text((36,80),'THIS TEXT IS BLACK')**  lcd.update()  sleep(6)  The textsize() function is not used to set the size of text, it is used to get the size of a text string in pixels. Here is the same script including a line that prints the size of the second text string:  #!/usr/bin/env python3  from ev3dev.ev3 import \*  from time import sleep  lcd = Screen()  lcd.draw.rectangle((0,0,177,40), fill='black')  lcd.draw.text((48,13),'Hello, world.', fill='white')  **my\_string='THIS TEXT IS BLACK'**  **print(lcd.draw.textsize(my\_string))**  lcd.draw.text((36,80),**my\_string**)  lcd.update()  sleep(6)  **Use larger fonts**  The draw.text() command used above without any 'font' argument draws very small text. Beginning with EV3 Python v0.8.0 you can use larger fonts. Version 0.8.0 was released in October 2016 - always be sure to use the latest version. Specifying a 'font' argument allows you to choose from a wide range of fonts and sizes that you can visualise [**HERE**](http://python-ev3dev.readthedocs.io/en/latest/other.html#ev3dev.fonts.load). The capital letters in the names of the fonts have the following meanings: B= Bold, I=Italic, O=Oblique (similar to italic), S=?, R=? As an example I will use the luBS14 font which is therefore bold, size 14. As usual, before using SSH to run this script or any other script that draws on the LCD screen you should follow the instructions at the bottom of this page.  #!/usr/bin/env python3  from ev3dev.ev3 import \*  from time import sleep  import ev3dev.fonts as fonts  from PIL import Image, ImageDraw, ImageFont  lcd = Screen()  lcd.draw.text((10,10), 'Hello World!', font=fonts.load('luBS14'))  lcd.update()  sleep(8)  **Install fonts to use bigger text**  To use fonts and font sizes other than the ones included by default in version 0.8.0 and beyond you can install additional fonts. Denis D, a top contributor to the ev3dev project, suggested this procedure to install and use additional fonts:   1. Install the MS core fonts package: sudo apt-get install ttf-mscorefonts-installer 2. Run this to see what fonts have been installed: ls /usr/share/fonts/truetype/msttcorefonts/   Here's a sample script using one of the newly available fonts, a TrueType (scalable) font called Arial in size 75 (a.k.a. huge):  #!/usr/bin/env python3  from ev3dev.ev3 import \*  **from PIL import Image, ImageDraw, ImageFont**  from time import sleep  lcd = Screen()  **f = ImageFont.truetype('/usr/share/fonts/truetype/msttcorefonts/Arial.ttf', 75)**  **lcd.draw.text((3,0), 'Hello', font=f)**  **lcd.draw.text((2,55), 'world', font=f)**  lcd.update()  sleep(7)  # if run from Brickman, need time to see displayed image  Here is the result:  [https://sites.google.com/site/ev3python/_/rsrc/1475081735262/learn_ev3_python/screen/screenshot4.png?height=143&width=200](https://sites.google.com/site/ev3python/learn_ev3_python/screen/screenshot4.png?attredirects=0)  **Display an image file**  I thank Denis D, a top contributor to the ev3dev project, for helping me with this code.  Before you can display an image file, you have to have an image file to display! Most of the popular image formats should work (jpg, bmp, gif, png etc). I suggest you start by doing what I did: copy all the standard Lego EV3 BMP image files that are included with the free Lego EV3 software into a directory called 'pics' within your 'robot' directory. You can see the BMP files listed on [**this page**](https://sites.google.com/site/ev3python/learn_ev3_python/screen/bmp-image-collection) and at the bottom of that page is a link to a zip file (328KB) containing the 107 monochrome BMP files, each one perfectly dimensioned to fit the EV3 screen (178\*128 pixels). If you prefer, you can find the files on your PC in a location like this: **C:\Program Files (x86)\LEGO Software\LEGO MINDSTORMS EV3 Home Edition\Resources\BrickResources\Retail\Images\files**. The 'Program Files (x86)' folder may also be called simply 'Program Files'. If you are getting the files from your PC I suggest you copy all the BMP files (not the RGF files) into a single folder.  Decompress the zip file then create a directory inside your robot directory, name it 'pics' and upload the decompressed BMP files to the pics directory. This is very easy to do if you are using MobaXTerm as I recommend. Once you've done all of that you should be ready to run the following script, replacing 'Bomb.bmp' with the name of the image file of your choice.  #!/usr/bin/env python3  from ev3dev.ev3 import \*  from time import sleep  from PIL import Image  lcd = Screen()  **logo = Image.open('pics/Bomb.bmp')**  **lcd.image.paste(logo, (0,0))**  lcd.update()  sleep(5) # when running from Brickman, need time to admire image  Here is the result:  [https://sites.google.com/site/ev3python/_/rsrc/1475060394826/learn_ev3_python/screen/screenshot.png?height=143&width=200](https://sites.google.com/site/ev3python/learn_ev3_python/screen/screenshot.png?attredirects=0)  The script runs easily from Brickman (don't forget to make your program file executable). If you want to run your script via SSH then be sure to read the paragraph at the bottom of this page.  Note that the line from PIL import Image is not really needed since the Image module is imported already by from ev3dev.ev3 import \*. However, it is still a good idea to include from PIL import Image because it makes it more obvious that the Image module is being used.  **How to capture a screenshot of the EV3 screen**  The simplest way to do a screen capture is to run this at the command line:  fbgrab "screenshot.png"  This will save a black and white png image called 'screenshot.png' in the current folder.  If you are using MobaXTerm you can then click the 'Download selected files' button to download the file to the location of your choice on the PC.  WasabiFan, a top contributor to the ev3dev project, sent me this more advanced script below for taking a screenshot of the EV3 screen. The script creates an image file 'screenshot.png' in the directory from which the script is run. The image is (optionally) scaled to double its original size and given a pale green background that resembles the real background of the EV3 LCD screen. This script was used to create the 'Hello World' screenshot that you saw higher up this page. You will want to run this script from SSH and not from Brickman so see the paragraph at the bottom of this page.  #!/usr/bin/env python3  # Import system utilities  import sys  from subprocess import call  # Import imaging library to resize and recolor the screenshot  from PIL import Image  # Choose the name for the screenshot to save.  # It defaults to "screenshot.png",  # but you can supply a custom name as a command-line argument.  out\_name = sys.argv[1] if len(sys.argv) > 1 else "screenshot.png"  # Call the fbgrab utility to have it save a screenshot  call(["fbgrab", out\_name]);  # Load the screenshot that fbgrab saved so that we can modify it  image = Image.open(out\_name)  # Convert the black-and-white screenshot (where  # each pixel is just "black" or "white")  # to an image with red, green and blue channels.  # This lets us make it more colorful later.  image = image.convert("RGB")  # Resize the image to be twice the dimensions,  # making sure to preserve crisp edges  image = image.resize(tuple(i \* 2 for i in image.size), Image.NEAREST)  # Get access to the underlying pixel data so that we can modify it  pixel\_data = image.load()  # Loop through each pixel  for y in range(image.size[1]):      for x in range(image.size[0]):          # If the pixel is white, make it #adb578 (the color we          # use as an approximation of the LCD screen) instead          if pixel\_data[x, y] == (255, 255, 255):              pixel\_data[x, y] = (173, 181, 120)  # Save the image again  image.save(out\_name);  The actual screenshot is made by the fbgrab method in bold. The yellow-highlighted line scales the image to double its original size - you can comment out that line if you prefer the original size (178x128 pixels). The green-highlighted block changes each white pixel into a pale green pixel.  **How to run from the terminal a script that writes to the EV3's LCD display**  David L, a top contributor to the ev3dev project, gave me the following advice:  **Easy way**, recommended unless you are a perfectionist. Disadvantage: you will then see a little blinking cursor at the bottom of a small 'dead square' somewhere on the screen (you can see the cursor in the 'Hello World' image higher up this page). :   1. Run sudo chvt 6. This just switches the display to show you an empty terminal instead of Brickman, so that Brickman doesn't mess with you. You will need to enter the password for robot, which is 'maker'. 2. Run your program. You should see it show up on the EV3 display, as you'd expect. You can stop the script running with Ctrl+C if necessary. You can work from this terminal like normal, run your program multiple times, etc. 3. Run sudo chvt 1 to get Brickman back   **Perfectionist's way**. Disadvantage: More steps and you will be using a copy of the EV3's tiny display.   1. Run sudo chvt 6. This just switches the display to show you an empty terminal instead of Brickman, so that Brickman doesn't mess with you. You will need to enter the password for robot, which is 'maker'. 2. **Run sudo conspy. This will make your SSH session show the same terminal that is on the EV3's display. Log in to the prompt it shows you with login = robot and password = maker.** 3. Run your program. You should see it show up on the EV3 display, as you'd expect. You can stop the script running with Ctrl+C if necessary. You can work from this terminal like normal, run your program multiple times, etc. 4. **Press the ESC key on the PC 3 times to exit the display terminal** 5. Run sudo chvt 1 to get Brickman back |

**Pantalla LCD**

|  |
| --- |
| Trabajar con la pantalla LCD es bastante complicado  y, en cualquier caso , la pantalla del EV3 es pequeña y difícil de ver (sin luz de fondo), por lo que le sugiero que trabajar con la pantalla debería ser una prioridad baja para usted al aprender EV3 Python.  El EV3 tiene una pantalla LCD monocroma (blanco y negro) de 178 x 128 píxeles. Las coordenadas del píxel superior izquierdo son (0, 0) y las coordenadas del píxel inferior derecho son (177, 127).  Para interactuar con la pantalla LCD, utilizará principalmente una potente biblioteca de gráficos llamada **Pillow,** que forma  parte de Python estándar en lugar de ser algo exclusivo de EV3 Python. Tenga en cuenta que  Pillow salió de la biblioteca de imágenes de Python ( **PIL** ). Hay [**algún tipo de documentación en la página de enlaces de Python EV3**](http://python-ev3dev.readthedocs.io/en/latest/other.html#screen) pero se puede encontrar documentación más detallada en t que  la documentación oficial Almohada  [**AQUÍ**](https://pillow.readthedocs.io/en/3.3.x/), especialmente la [**documentación sobre el módulo de ImageDraw**](https://pillow.readthedocs.io/en/3.3.x/reference/ImageDraw.html) . El módulo ImageDraw proporciona los siguientes métodos:   * arco (xy, inicio, fin, relleno = Ninguno) * mapa de bits (xy, mapa de bits, relleno = Ninguno) * acorde (xy, inicio, final, relleno = Ninguno, esquema = Ninguno) * elipse (xy, fill = None, outline = None) * line (xy, fill = None, width = 0) * pieslice (xy, inicio, fin, relleno = Ninguno, esquema = Ninguno) * punto (xy, llenar = Ninguno) * polígono (xy, fill = None, outline = None) * rectángulo (xy, fill = None, outline = None) * text (xy, text, fill = None, font = None, anchor = None, spacing = 0, align = "left") * ~~multiline\_text~~ (xy, text, fill = None, font = None, anchor = None, spacing = 0, align = "left") * Tamaño de texto (texto, fuente = Ninguno, espaciado = 0) * multiline\_textsize (text, font = None, spacing = 0)   Y ou puede encontrar  [**este tutorial**](https://pillow.readthedocs.io/en/3.3.x/handbook/tutorial.html) para ser un buen lugar para comenzar.  Multiline\_text no parece estar disponible actualmente en EV3 Python y puede haber otros que no están disponibles.  **Advertencia** : las secuencias de comandos EV3 Python usando la pantalla LCD se pueden ejecutar con éxito desde la interfaz de Brickman, pero si desea ejecutar dichas secuencias de comandos desde una sesión SSH (por ejemplo, PuTTY o MobaXTerm), debe seguir las instrucciones en la parte inferior de esta página. Si ejecuta código que dirige la pantalla desde la sesión SSH sin seguir las instrucciones al pie de esta página, puede experimentar dos problemas:   * la pantalla de Brickman puede reaparecer antes de que la secuencia de comandos deje de ejecutarse, incluso después de un segundo o menos * o puede tener el problema opuesto: cuando el script se detiene, puede tomar uno o dos minutos antes de que la pantalla vuelva a la interfaz de Brickman   **Smiley / Grumpy**  Esta secuencia de comandos dibuja dos círculos para los ojos y luego un arco (en este caso la mitad de una elipse) que alterna entre abrir hacia arriba (una sonrisa) y abrir hacia abajo (gruñón). **Advertencia** : NO ejecute este script desde Brickman porque se ejecuta para siempre y, por lo tanto, debe detenerlo presionando prolongadamente el botón Atrás, pero esto NO detiene adecuadamente el programa, presumiblemente debido a algún tipo de error en Brickman. PUEDE ejecutar satisfactoriamente este programa desde SSH, deteniéndolo con Ctrl-C, pero no olvide leer la nota al final de esta página.  #! / usr / bin / env python3  desde el tiempo de importación de sueño  de la importación ev3dev.ev3 \*  lcd = Pantalla ()  smile = True  mientras que es cierto:      lcd.clear ()      # lcd.draw devuelve un identificador PIL.ImageDraw      lcd.draw.ellipse ((20, 20, 60, 60))      lcd.draw.ellipse ((118, 20, 158, 60))      si sonríe:          lcd.draw.arc ((20, 80, 158, 100), 0, 180)      más:          lcd.draw.arc ((20, 80, 158, 100), 180, 360)      sonreír = no sonreír # alternar entre verdadero y falso      # Actualizar pantalla lcd      lcd.update () # Aplica los cambios pendientes a la pantalla.      # No se dibujará nada en la pantalla lcd      # hasta que se llame a esta función.      dormir (1)  Tenga en cuenta que el comando screen.draw.ellipse () tiene parámetros que son la esquina superior izquierda y la esquina  inferior derecha  de un cuadro delimitador (un cuadro imaginario que contendría la elipse). El   screen.draw.arc ()  de comandos necesita el mismo par de coordenadas y también tiene como tarting ángulo y un ángulo final, tanto en grados. Los ángulos se miden desde las 3 en punto, aumentando en el sentido de las agujas del reloj.  **Mostrar texto**  Este script mostrará 'Hello, World' como un pequeño texto blanco en un rectángulo negro con 'THIS TEXT IS BLACK' en negro en la mitad inferior de la pantalla. Las coordenadas en el parámetro de la función de texto () se refieren a la esquina superior izquierda del texto.  #! / usr / bin / env python3  de la importación ev3dev.ev3 \*  desde el tiempo de importación de sueño  lcd = Pantalla ()  **lcd.draw.rectangle ((0,0,177,40), fill = 'black')**  **lcd.draw.text ((48,13), 'Hola, mundo.', fill = 'blanco')**  **lcd.draw.text ((36,80), 'ESTE TEXTO ES NEGRO')**  lcd.update ()  dormir (6)  La función textsize () no se usa para establecer el tamaño del texto, sino que se usa para obtener el tamaño de una cadena de texto en píxeles. Aquí está el mismo script que incluye una línea que imprime el tamaño de la segunda cadena de texto:  #! / usr / bin / env python3  de la importación ev3dev.ev3 \*  desde el tiempo de importación de sueño  lcd = Pantalla ()  lcd.draw.rectangle ((0,0,177,40), fill = 'black')  lcd.draw.text ((48,13), 'Hola, mundo.', fill = 'blanco')  **my\_string = 'ESTE TEXTO ES NEGRO'**  **print (lcd.draw.textsize (my\_string))**  lcd.draw.text ((36,80), **my\_string** )  lcd.update ()  dormir (6)  **Use fuentes más grandes**  El comando draw.text () utilizado anteriormente sin ningún argumento 'font' dibuja un texto muy pequeño. A partir de EV3 Python v0.8.0 puede usar fuentes más grandes. La versión 0.8.0 w fue lanzada en octubre de 2016 - siempre asegúrese de usar la última versión. Especificar un argumento de "fuente" le permite elegir entre una amplia gama de fuentes y tamaños que puede visualizar [**AQUÍ**](http://python-ev3dev.readthedocs.io/en/latest/other.html#ev3dev.fonts.load) . Las  letras mayúsculas en los nombres de las fuentes tienen los siguientes significados: B = Negrita, I = Cursiva,  O = Oblicua (similar a cursiva),  S =? R =? Como ejemplo, usaré la fuente luBS14 que, por lo tanto, es negrita, tamaño 14. Como de costumbre, antes de usar SSH para ejecutar este script o cualquier otro script que se dibuje en la pantalla LCD, debe seguir las instrucciones al pie de esta página.  #!/usr/bin/env python3  from ev3dev.ev3 import \*  from time import sleep  import ev3dev.fonts as fonts  from PIL import Image, ImageDraw, ImageFont  lcd = Screen()  lcd.draw.text((10,10), 'Hello World!', font=fonts.load('luBS14'))  lcd.update()  sleep(8)  **Instalar fuentes para usar texto más grande**  Para utilizar fuentes y tamaños de fuente distintos a los incluidos por defecto en la versión 0.8.0 y posteriores, puede instalar fuentes adicionales. Denis D, uno de los principales contribuidores al proyecto ev3dev, sugirió este procedimiento para instalar y usar fuentes adicionales:   1. Instale el paquete de fuentes de MS core:  sudo apt-get install ttf-mscorefonts-installer 2. Ejecute esto para ver qué fuentes se han instalado:  ls / usr / share / fonts / truetype / msttcorefonts /   Aquí hay una secuencia de comandos de muestra con una de las fuentes disponibles recientemente, una fuente TrueType (escalable) llamada Arial en tamaño 75 (también conocida como enorme):  #! / usr / bin / env python3  de la importación ev3dev.ev3 \*  **de PIL import Image, ImageDraw, ImageFont**  desde el tiempo de importación de sueño  lcd = Pantalla ()  **f = ImageFont.truetype ('/ usr / share / fonts / truetype / msttcorefonts / Arial.ttf', 75)**  **lcd.draw.text ((3,0), 'Hola', fuente = f)**  **lcd.draw.text ((2,55), 'mundo', fuente = f)**  lcd.update ()  sleep (7) # si se ejecuta desde Brickman, necesita tiempo para ver la imagen mostrada  Aquí está el resultado:  [https://sites.google.com/site/ev3python/_/rsrc/1475081735262/learn_ev3_python/screen/screenshot4.png?height=143&width=200](https://sites.google.com/site/ev3python/learn_ev3_python/screen/screenshot4.png?attredirects=0)  **Mostrar un archivo de imagen**  Agradezco a Denis D,  colaborador principal del proyecto ev3dev,  por ayudarme con este código.  ¡Antes de que pueda visualizar un archivo de imagen, debe tener un archivo de imagen para visualizar! La mayoría de los formatos de imagen populares deberían funcionar (jpg, bmp, gif, png, etc.). Le sugiero que empiece haciendo lo que hice: copie todos los archivos de imagen Lego EV3 BMP estándar que se incluyen con el software Lego EV3 gratuito en un directorio llamado 'fotos' dentro de su directorio 'robot'. Puede ver los archivos BMP enumerados en [**esta página**](https://sites.google.com/site/ev3python/learn_ev3_python/screen/bmp-image-collection) y en la parte inferior de esa página hay un enlace a un archivo zip ( 328 KB) que  contiene los 107 archivos BMP monocromáticos, cada uno perfectamente dimensionado para adaptarse a la pantalla EV3 (178 \* 128 píxeles). Si lo prefiere, puede encontrar los archivos en su PC en una ubicación como esta:  **C: \ Archivos de programa (x86) \ LEGO Software \ LEGO MINDSTORMS EV3 Home Edition \ Resources \ BrickResources \ Retail \ Images \ files** . TLa carpeta 'Archivos de programa (x86)' también se puede llamar simplemente ' Archivos de programa'. Si obtiene los archivos de su PC, le sugiero que copie todos los archivos BMP (no los archivos RGF) en una única carpeta.  Descomprime el archivo comprimido y luego crea un directorio dentro del directorio de tu robot, llámalo "fotos" y sube los archivos BMP descomprimidos al directorio de fotos. Esto es muy fácil de hacer si está usando MobaXTerm como lo recomiendo. Una vez que haya hecho todo eso, debe estar preparado para ejecutar el siguiente script, reemplazando 'Bomb.bmp' con el nombre del archivo de imagen que elija.  #! / usr / bin / env python3  de la importación ev3dev.ev3 \*  desde el tiempo de importación de sueño  de PIL import Imagen  lcd = Pantalla ()  **logo = Image.open ('fotos / Bomb.bmp')**  **lcd.image.paste (logo, (0,0))**  lcd.update ()  sleep (5) # cuando se ejecuta desde Brickman, necesita tiempo para admirar la imagen  Aquí está el resultado:  [https://sites.google.com/site/ev3python/_/rsrc/1475060394826/learn_ev3_python/screen/screenshot.png?height=143&width=200](https://sites.google.com/site/ev3python/learn_ev3_python/screen/screenshot.png?attredirects=0)  La secuencia de comandos se ejecuta fácilmente desde Brickman (no olvides hacer tu archivo de programa ejecutable). Si desea ejecutar su script a través de SSH, asegúrese de leer el párrafo en la parte inferior de esta página.  Tenga en cuenta que la línea  de PIL import Image no es realmente necesaria ya que el módulo Image ya se ha importado  desde ev3dev.ev3 import \* . Sin embargo, sigue siendo una buena idea incluir desde la importación de imágenes de PIL porque hace más obvio que se está utilizando el módulo de imágenes.  **Cómo capturar una captura de pantalla de la pantalla EV3**  La forma más sencilla de hacer una captura de pantalla es ejecutar esto en la línea de comando:  fbgrab "screenshot.png"  Esto guardará una imagen png en blanco y negro llamada 'screenshot.png' en la carpeta actual.  Si está utilizando MobaXTerm, puede hacer clic en el botón 'Descargar archivos seleccionados' para descargar el archivo a la ubicación que elija en la PC.  WasabiFan, uno de los principales contribuidores al proyecto ev3dev, me envió este script más avanzado a continuación para tomar una captura de pantalla de la pantalla EV3. El script crea un archivo de imagen 'screenshot.png' en el directorio desde el que se ejecuta el script. La imagen (opcionalmente) se escala para duplicar su tamaño original y se le da un fondo verde pálido que se asemeja al fondo real de la pantalla LCD del EV3. Este script se usó para crear la captura de pantalla de "Hello World" que usted vio más arriba en esta página. Querrá ejecutar este script desde SSH y no desde Brickman, de modo que vea el párrafo en la parte inferior de esta página.  #! / usr / bin / env python3  # Importar utilidades del sistema  import sys  de la llamada de importación del subproceso  # Importar la biblioteca de imágenes para redimensionar y volver a colorear la captura de pantalla  de PIL import Imagen  # Elija el nombre de la captura de pantalla para guardar.  # Se establece de forma predeterminada en "screenshot.png",  # pero puede proporcionar un nombre personalizado como un argumento de línea de comandos.  out\_name = sys.argv [1] si len (sys.argv)> 1 else "screenshot.png"  # Llame a la utilidad fbgrab para que guarde una captura de pantalla  call (["fbgrab", out\_name]);  # Cargar la captura de pantalla que fbgrab guardó para que podamos modificarla  image = Image.open (out\_name)  # Convertir la captura de pantalla en blanco y negro (donde  # cada píxel es solo "negro" o "blanco")  # a una imagen con canales rojos, verdes y azules.  # Esto nos permite hacerlo más colorido más tarde.  image = image.convert ("RGB")  # Cambiar el tamaño de la imagen para que sea el doble de las dimensiones,  # asegurándose de conservar los bordes nítidos  image = image.resize (tupla (i \* 2 para i en image.size), Image.NEAREST)  # Accede a los datos de píxeles subyacentes para que podamos modificarlo  pixel\_data = image.load ()  # Loop a través de cada píxel  para y dentro del rango (image.size [1]):      para x en rango (image.size [0]):          # Si el píxel es blanco, hágalo # adb578 (el color que          # uso como una aproximación de la pantalla LCD) en su lugar          si pixel\_data [x, y] == (255, 255, 255):              pixel\_data [x, y] = (173, 181, 120)  # Guardar la imagen de nuevo  image.save (out\_name);  La captura de pantalla real se realiza mediante el método fbgrab en negrita. La línea resaltada en amarillo amplía la imagen para duplicar su tamaño original; puede comentar esa línea si prefiere el tamaño original (178x128 píxeles). El bloque resaltado en verde cambia cada píxel blanco en un píxel verde pálido.  **Cómo  ejecutar desde el terminal un script que escribe en la pantalla LCD del EV3**  David L, uno de los principales contribuidores al proyecto ev3dev, me dio los siguientes consejos:  **De manera fácil** , recomendado a menos que seas un perfeccionista. Desventaja:  A continuación,  ver  un cursor parpadeante poco en la parte inferior de un pequeño 'muertos cuadrado' en algún lugar de la pantalla (se puede ver el cursor en la imagen de 'Hola Mundo' más arriba en esta página). :   1. Ejecute sudo chvt 6 . Esto simplemente cambia la pantalla para mostrarle una terminal vacía en lugar de Brickman, para que Brickman no se meta con usted. Tendrá que ingresar la contraseña para robot, que es 'fabricante'. 2. Ejecute su programa. Debería verlo aparecer en la pantalla del EV3, como era de esperar. Puede detener el script que se ejecuta con Ctrl + C si es necesario. Puede trabajar desde este terminal como de costumbre, ejecutar su programa varias veces, etc. 3. Ejecute sudo chvt 1 para recuperar Brickman   **El camino del perfeccionista** . Desventaja: Más pasos y usarás una copia de la pequeña pantalla del EV3 .   1. Ejecute  sudo chvt 6 . Esto simplemente cambia la pantalla para mostrarle una terminal vacía en lugar de Brickman, para que Brickman no se meta con usted. Tendrá que ingresar la contraseña para robot, que es 'fabricante'. 2. **Ejecute  sudo conspy . Esto hará que su sesión SSH muestre el mismo terminal que está en la pantalla del EV3. Inicie sesión en el mensaje que le muestra con login = robot y password = maker.** 3. Ejecute su programa. Debería verlo aparecer en la pantalla del EV3, como era de esperar. Puede detener el script que se ejecuta con Ctrl + C si es necesario. Puede trabajar desde este terminal como de costumbre, ejecutar su programa varias veces, etc. 4. **Presione la tecla ESC en la PC 3 veces para salir de la terminal de visualización** 5. Ejecute  sudo chvt 1  para recuperar Brickman |