

## **Kit de programación Código Pi**

### **El juego del pong**



## Autoridades

### **Presidente de la Nación**

Mauricio Macri

### **Jefe de Gabinete de Ministros**

Marcos Peña

### **Ministro de Educación, Cultura, Ciencia y Tecnología**

Alejandro Finocchiaro

### **Secretario de Gobierno de Cultura**

Pablo Avelluto

### **Secretario de Gobierno de Ciencia, Tecnología e Innovación Productiva**

Lino Barañao

### **Titular de la Unidad de Coordinación General del Ministerio de Educación, Cultura, Ciencia y Tecnología**

Manuel Vidal

### **Secretaria de Innovación y Calidad Educativa**

Mercedes Miguel

### **Subsecretario de Coordinación Administrativa**

Javier Mezzamico

### **Directora Nacional de Innovación Educativa**

María Florencia Ripani

ISBN en trámite



Este material fue producido por el Ministerio de Educación, Cultura, Ciencia y Tecnología en base a contenidos provistos sin cargo por la Fundación Raspberry Pi mediante licencias Creative Commons y han sido desarrollados en función de los Núcleos de Aprendizajes Prioritarios de educación digital, programación y robótica y los recursos tecnológicos propuestos en el marco del Plan Aprender Conectados.

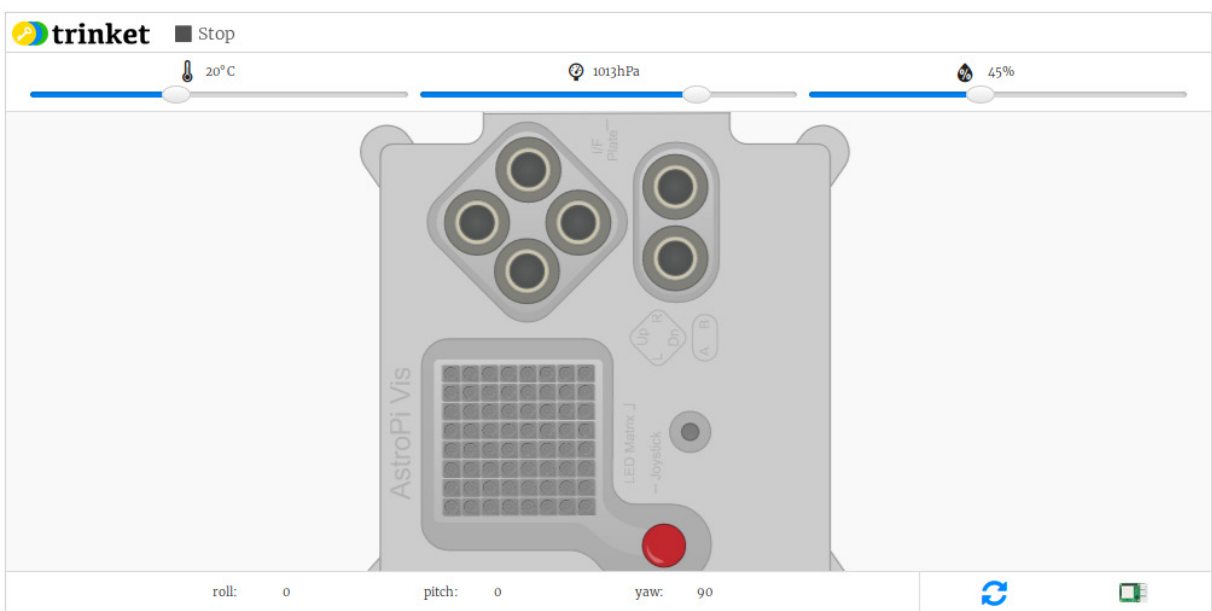
# Índice

Programemos el juego .....	5
Usando el emulador de Sense HAT .....	9
Mostrar un color en el Sense HAT .....	13
Desafío: agrega más funcionalidad .....	36

## Programemos el juego

### Introducción

En esta actividad crearemos un juego de Pong usando nuestro Raspberry Pi, un Sense HAT, y un poco de código de Python. Pong es uno de los videojuegos más antiguos, y [originalmente se jugaba con un osciloscopio](#).



### ¿Qué vamos a hacer?

Abrí este [Trinket](#) para probar el juego. Usando las flechas del teclado puedes controlar la paleta. ¡No dejes que la bola toque el borde o perderás el juego!

### ¿Qué vamos a aprender?

Programando un juego de Pong con tu Raspberry Pi y Sense HAT, aprenderás a:

- Iluminar los pixeles del Sense HAT
- Detectar movimientos del joystick de Sense HAT

## Lo que necesitás

### Hardware

- Raspberry Pi
- Sense HAT (Si no tenés un Sense HAT, podés crear el proyecto en un navegador web usando el [emulador de Sense HAT](#), o podés usar el software emulador en tu Raspi.)

### Software

Necesitarás la última versión de Raspbian que ya incluye los siguientes paquetes de software:

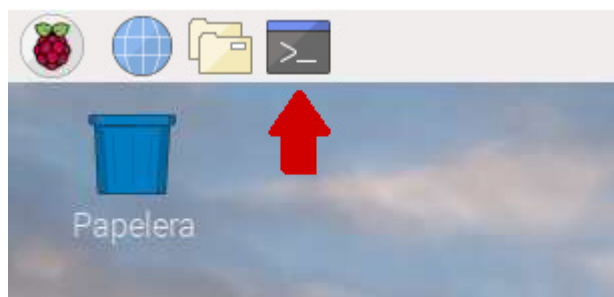
- Python 3
- Sense HAT para Python 3

Si por algún motivo necesitás instalar algún paquete de manera manual, seguí las siguientes instrucciones:

Instalar un paquete de software en la Raspberry Pi

Tu Raspberry Pi deberá estar conectado a internet para poder instalar paquetes de software. Antes de instalar cualquier paquete, actualizá a la última versión a Raspbian, el sistema operativo de tu Raspberry Pi.

- Para hacer esto, abrí una ventana de terminal e ingresá los siguientes comandos:



```
sudo apt-get update  
sudo apt-get upgrade
```

- Ahora podés instalar los paquetes que necesitas por medio de comandos `install` en tu ventana de terminal. Por ejemplo, así se instala el software para Sense HAT:

```
sudo apt-get install sense-hat
```

Escribí este comando en la terminal para instalar el paquete de Sense HAT:

```
sudo apt-get install sense-hat
```

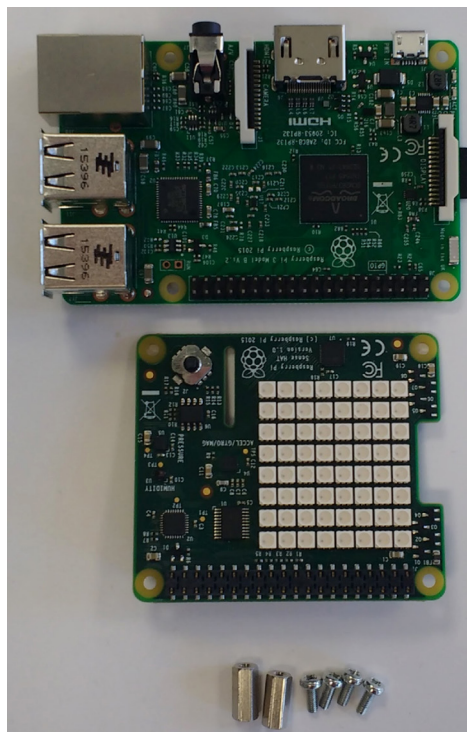
### Conectá el Sense HAT

- Si tenés un Sense HAT, conéctalo a tu Raspberry Pi.

Conectando un Sense HAT

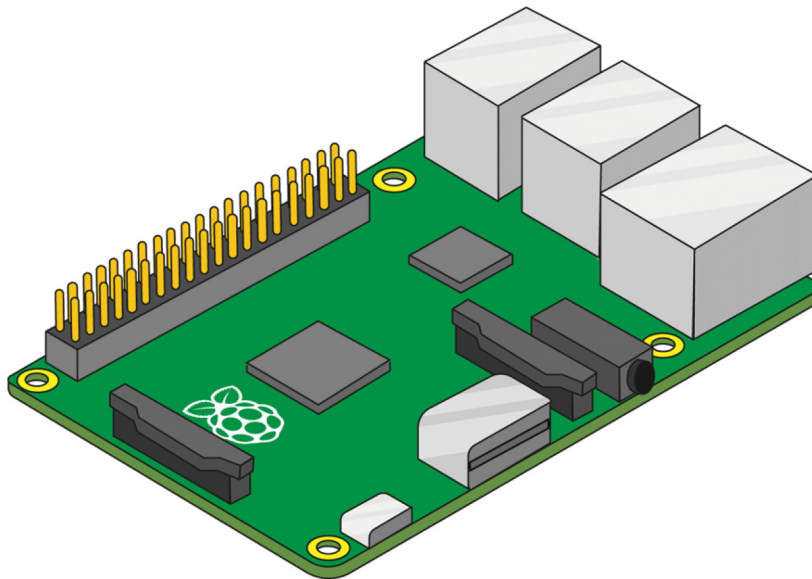
Antes de conectar cualquier HAT a tu Raspberry Pi, asegurate que esté apagado.

- Quitá el Sense HAT y sus partes del envoltorio.
- Usá dos de los tornillos provistos para unir los espaciadores a tu Raspberry Pi, como se muestra más abajo.



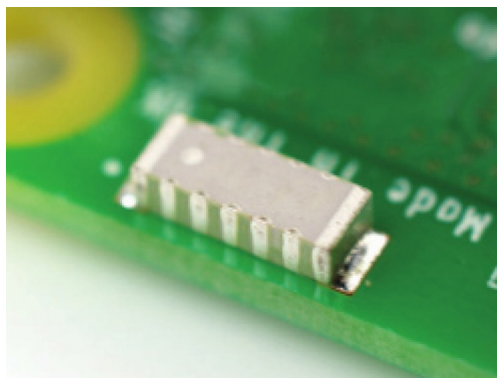
**Nota:** el paso anterior es opcional — no es necesario agregar los espaciadores al Sense HAT para que funcione.

- Luego insertá el Sense HAT cuidadosamente en los pines de tu Raspberry Pi, y aseguralo con los tornillos restantes.



**Nota:** usar un espaciador metálico cerca de la antena inalámbrica de tu Raspberry Pi 3's puede degradar su rendimiento o rango. Podés omitir este espaciador, o usar espaciadores y tornillos de nylon.

**Truco:** tené cuidado cuando intentes desconectar tu Sense HAT, ya que el conector de 40 pines puede quedar trabado.



- Si no tenés acceso a un Sense HAT real, podés usar un emulador.

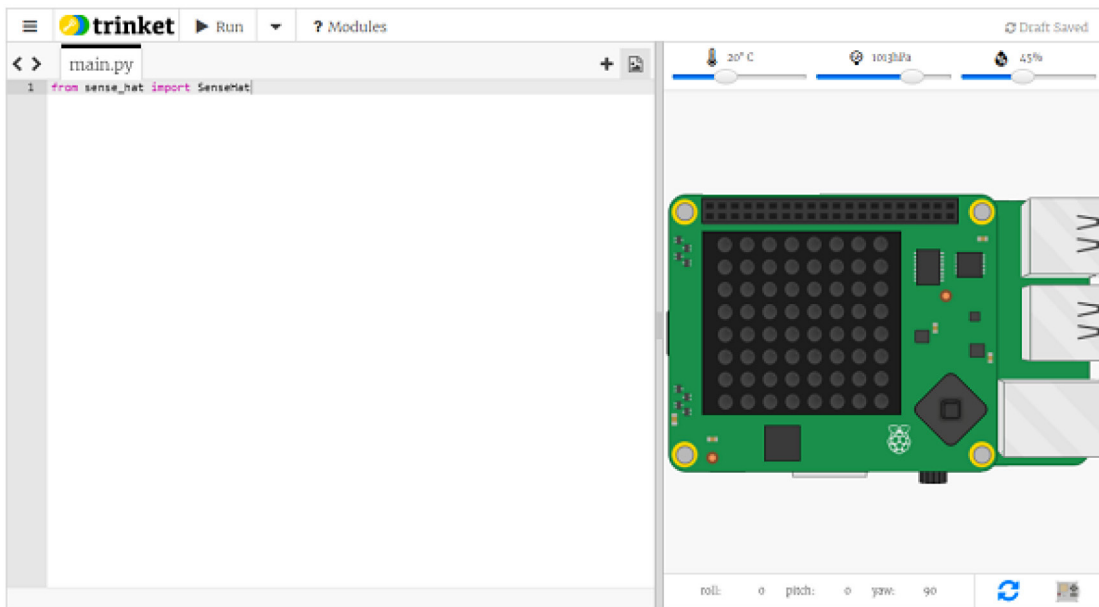


## Usando el emulador de Sense HAT

Si no tenés acceso a un Sense HAT real, podés usar un emulador.

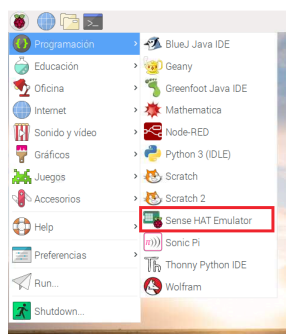
Emulador online de Sense HAT

Hay un emulador online que podés usar en tu navegador para escribir y probar código para Sense HAT.



- Abrí un navegador de internet, entrá a <https://trinket.io/sense-hat> y borrrá el código de muestra del editor.
- Para poder guardar tu trabajo, debés [crear una cuenta gratuita](#) en el sitio web de Trinket.

## Emulador de Sense HAT en tu Raspberry Pi



Tu Raspberry Pi incluye un emulador de Sense HAT en el sistema operativo Raspbian.

- Desde el menú principal, seleccioná **Programación > Sense HAT Emulator** para abrir una ventana que contiene al emulador.
- Si estás usando esta versión del emulador, tu programa debe importar desde `sense_emu` en vez de `sense_hat`:

```
from sense_emu import SenseHat
```

Si luego querés ejecutar tu código en un Sense HAT real, sólo debés cambiar la línea import como se ve debajo. El resto del código se mantiene exactamente igual.

```
from sense_hat import SenseHat
```

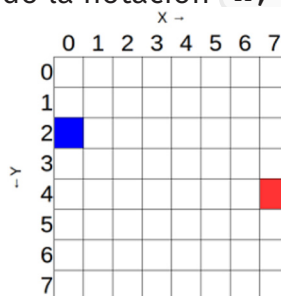
## Encendé un LED

Muchas veces los videojuegos usan las coordenadas `x` e `y` para determinar donde se encuentra un objeto en la pantalla. `x` se utiliza para fijar la posición horizontal de un objeto, e `y` se usa para fijar la posición vertical.

Podemos hacer lo mismo con los LEDs del Sense HAT.

### Coordenadas de la pantalla LCD del Sense HAT

La pantalla LED del The Sense HAT usa un sistema de coordenadas con un eje X y un eje Y. La numeración de ambos ejes inicia en 0 (no en 1) en la esquina superior izquierda. Cada LED puede ser usado como un pixel de una imagen, y puede ser referenciado usando la notación `x, y`.



El pixel azul está en las coordenadas `0, 2`. El pixel rojo está en las coordenadas `7, 4`.

Podés encender los pixeles (LEDs) individualmente utilizando el método `set_pixel()`.

Para replicar el diagrama de más arriba, podemos escribir un programa como este:

```
from sense_hat import SenseHat
sense = SenseHat()

blue = (0, 0, 255)
red = (255, 0, 0)

sense.set_pixel(0, 2, blue)
sense.set_pixel(7, 4, red)
```

Iniciemos nuestro juego de Pong iluminando un solo LED para crear una bola, y luego agreguemos algunos más para crear una paleta.

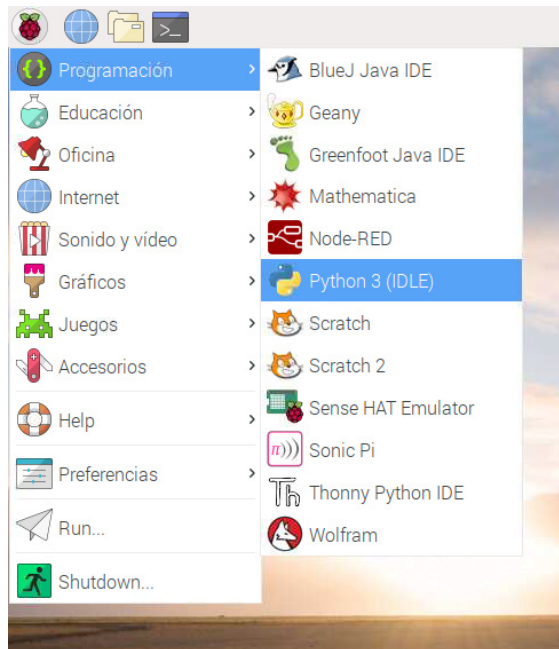
- Abrí IDLE si estás usando un Sense HAT real, o un nuevo trinket si estás usando el [emulador](#).

## Abriendo IDLE3

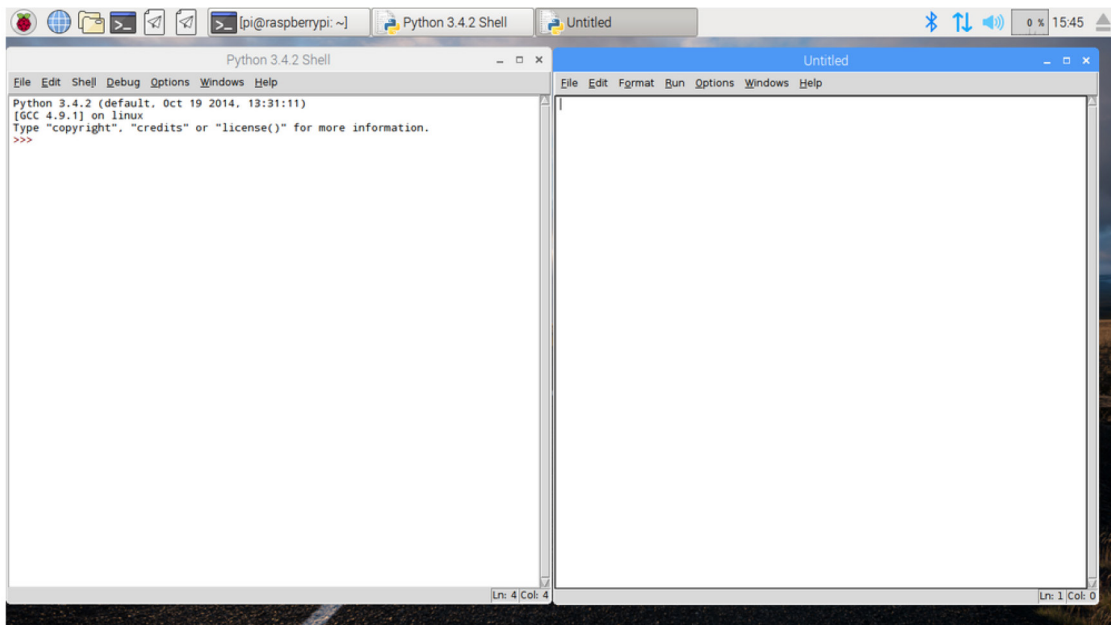
IDLE es el Entorno de Desarrollo Integrado de Python, que podés usar para escribir y ejecutar código.

Para abrir IDLE, debés ir al menú principal y seleccionar `programación`. Deberías ver dos versiones de IDLE - asegurate de seleccionar la que se llama `Python 3 (IDLE)`.

## APRENDER CONECTADOS



Para crear un nuevo archivo en IDLE, debés seleccionar el menú **Archivo**, y luego **Nuevo archivo**. Esto abrirá una segunda ventana donde podés escribir tu programa.



- Agregá estas líneas de código al inicio de tu archivo para importar el módulo `sense_hat` y conectarte al Sense HAT.

```
from sense_hat import SenseHat  
sense = SenseHat()
```

- La paleta será blanca. Definí una variable llamada `white`, y fija su valor en `255, 255, 255`, que es la representación RGB del color blanco.

## Mostrar un color en el Sense HAT

- En un archivo de Python, escribí el siguiente programa:

```
from sense_hat import SenseHat  
  
sense = SenseHat()  
  
r = 255  
g = 255  
b = 255  
  
sense.clear((r, g, b))
```

- Guardá y ejecutá tu código. La matriz LCD debería iluminarse de blanco brillante.
- Las variables `r`, `g`, y `b` representan los colores rojo, verde y azul respectivamente. Sus valores especifican qué tan brillante debería ser cada color, cada valor puede ir de `0` y `255`. En el programa de más arriba, el valor máximo de cada color ha sido usado, obteniendo el color blanco como resultado.
- También podés definir los tres valores RGB de un color usando una sola línea de código:

```
red = (255,0,0)
from sense_hat import SenseHat

sense = SenseHat()

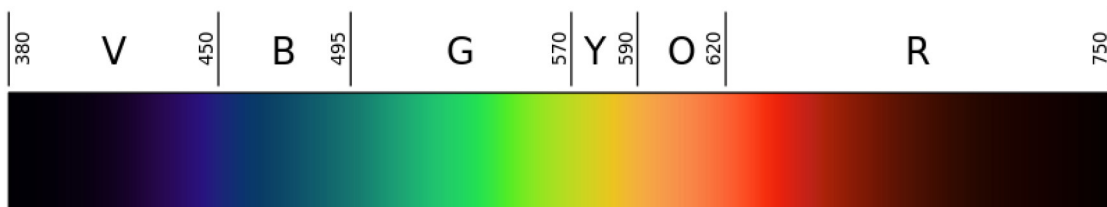
r = 255
g = 255
b = 255

sense.clear((r, g, b))
```

- Cambiá el valor de uno de los colores, y corre el programa de nuevo. ¿Qué ves?
- ¿Qué otros colores podés hacer?

## Representando colores con números

El color de un objeto depende del color de la luz que refleja o emite. La luz puede tener distintas longitudes de onda, y el color de la luz depende de su longitud de onda. El color de la luz de acuerdo a la longitud de onda puede verse en el diagrama de más abajo. Podés reconocer esto como los colores del arco iris.



Los humanos vemos el color gracias a células especiales en nuestros ojos. Estas células se llaman conos. Tenemos tres tipos de conos, y cada tipo detecta luz roja, azul o verde. Por lo tanto todos los colores que vemos son simplemente mezclas de los colores rojo, azul y verde.



En la mezcla aditiva de colores, se usan tres colores primarios (rojo, verde y azul) para hacer los demás colores. En la imagen de arriba, hay tres reflectores de igual brillo, uno por cada color. en la ausencia de cualquier color, el resultado es negro. Si los tres colores son mezclados, el resultado es blanco. Cuando combinamos rojo y verde, el resultado es amarillo. Cuando combinamos rojo y azul, el resultado es magenta. Cuando combinamos azul y verde, el resultado es cian. Es posible hacer muchos más colores variando el brillo de los tres colores primarios.

Las computadores almacenan todo en unos y ceros. Estos unos y ceros se organizan en conjuntos de ocho, llamados **bytes**.

Un byte puede representar cualquier valor de 0 a 255.

Cuando queremos representar un color en un programa informático, podemos hacerlo definiendo las cantidades de rojo, verde y azul que generan este color. Dichas cantidades se almacenan en un byte cada una y por lo tanto en números entre 0 y 255.

Aquí hay una tabla mostrando algunos valores de color:

Rojo	Verde	Azul	Color
255	00		Rojo
0	255	0	Verde
00		255	Azul
255	255	0A	marillo
255	0	255	Magenta
0	255	255	Cian

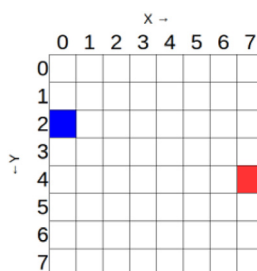
Podés encontrar un bonito selector de colores en [w3schools](#).

La paleta siempre estará en la columna izquierda de pixeles, por lo que su valor de `x` será siempre igual a `0`, pero su valor de `y` cambiará cuando muevas el bate de arriba a abajo.

- Creá otra variable llamada `bat_y` y fija su valor a `4`.
- Fijá el LED en la posición `0, bat_y` a white a blanco usando el método `set_pixel`.

## Cambiando un solo pixel en el Sense HAT

Podés usar el comando `set_pixel` para controlar LEDs individuales en el Sense HAT. Para hacer esto, debés fijar las variables `x` e `y` que el comando `set_pixel` recibe. `x` indica el eje horizontal del HAT, y puede tener un valor que va de `0` (izquierda) y `7` (derecha). `y` indica el eje vertical del HAT, y puede tener un valor que va de `0` (arriba) a `7` (abajo). Por lo tanto, las coordenadas `x, y = 0, 0` refieren al LED superior izquierdo, y las coordenadas `x, y = 7, 7` refieren al LED inferior derecho.



La grilla de más arriba corresponde con esta orientación de tu Raspberry Pi:





Vamos a probar este ejemplo, para fijar un color distinto en cada esquina de la pantalla de LED del Sense HAT. Necesitás usar el comando `set_pixel` varias veces en el código, de esta manera:

```
from sense_hat import SenseHat

sense = SenseHat()

sense.clear() # Esto borra cualquier pixel que haya quedado
prendido en el Sense HAT. Puede que no necesites hacerlo y
convenga hacerlo en otro momento
sense.set_pixel(0, 0, 255, 0, 0)
sense.set_pixel(0, 7, 0, 255, 0)
sense.set_pixel(7, 0, 0, 0, 255)
sense.set_pixel(7, 7, 255, 0, 255)
```

Probá de distintos colores usando el emulador de Sense HAT:

```
from sense_hat import SenseHat

sense = SenseHat()

sense.clear()
sense.set_pixel(0, 0, 255, 0, 0)
sense.set_pixel(0, 7, 0, 255, 0)
sense.set_pixel(7, 0, 0, 0, 255)
sense.set_pixel(7, 7, 255, 0, 255)
```

## Necesito una pista

Primero, créa una variable llamada `white` con un valor de `255, 255, 255` así:

```
white = (255, 255, 255)
```

En la siguiente línea, créa otra variable como la de color `white`, pero llamada `bat_y` con un valor de `4`.

El código debería verse así:

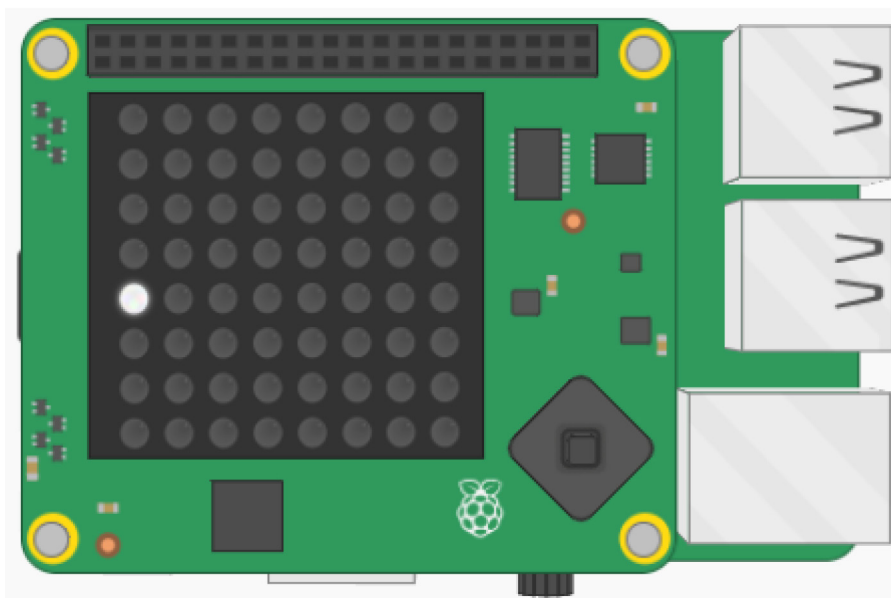
```
from sense_hat import SenseHat
sense = SenseHat()

white = (255, 255, 255)

bat_y = 4

sense.set_pixel(0, bat_y, white)
```

- Guardá y ejecutá tu programa. Un solo LED debería iluminarse en blanco, en el lado izquierdo de la pantalla LED.



## Hacé una paleta

Vamos a hacer el resto de la paleta, iluminando los LED inmediatamente arriba y abajo del que está iluminado en este momento. Para ello, crearemos una **función**.

### Creando e invocando funciones en Python

Cuando programás, a veces necesitás usar la misma línea de código muchas veces dentro de tu programa. Alternativamente, quizá quieras que las mismas líneas de código se ejecuten cada vez que ocurre un evento determinado, por ejemplo, presionar una tecla o escribir una frase particular. Para acciones como estas, podés considerar usar una **función**.

Las funciones son bloques de código con **nombre** que realizan una tarea determinada. La función más simple que puedes crear en Python se ve así:

```
def hola():
    print('¡Hola Mundo!')
```

Con la palabra clave `def` le decís a Python que estás creando una función con un nombre determinado, en este caso `hola`. los paréntesis posteriores al nombre son importantes.

Los dos puntos al final de la línea indican que el código dentro de la función estará indentado a partir de la próxima línea, como en los bucles `for` o `while` o los condicionales `if` / `elif` / `else`.

Podés hacer una **llamada** a una función escribiendo su nombre con los paréntesis incluidos. Entonces, para llamar a la función del ejemplo, debés escribir `hola()`. Aquí está el programa completo:

```
def hola():
    print('¡Hola Mundo!')

hola()
```

- Indentá la línea `sense.set_pixel(0, bat_y, white)` poniendo el cursor al inicio de la línea y pulsando la tecla `tab`.
- En la línea inmediatamente superior a ésta, iniciá una función llamada `draw_bat`:

```
< > main.py
1 from sense_hat import SenseHat
2 sense = SenseHat()
3
4 white = (255, 255, 255)
5
6 bat_y = 4
7
8 def draw_bat():
9     sense.set_pixel(0, bat_y, white)|
```

Las líneas que siguen al inicio de una función deben ser indentadas para indicar que están **dentro** de dicha función.

Podés agregar un **comentario** justo arriba del comienzo de tu función para mostrar que la siguiente sección del programa contendrá tus funciones - escribiremos otras más adelante.

```
# Functions -----
```

- Agregá dos líneas más de código dentro de la función para iluminar los LED en las posiciones `bat_y + 1`, y `bat_y - 1`.

## Necesito una pista

Las líneas que necesitás son muy similares a las que ya tenés. ¿Qué debés cambiar en esta línea para hacer que se ilumine `bat_y + 1` en vez de `bat_y`?

```
sense.set_pixel(0, bat_y, white)
```

No olvides indentar las nuevas líneas de código para que estén dentro de tu función.

Tu función debería verse así:

```
# Functions -----  
def draw_bat():  
    sense.set_pixel(0, bat_y, white)  
    sense.set_pixel(0, bat_y + 1, white)  
    sense.set_pixel(0, bat_y - 1, white)
```

Si corrés tu programa ahora, no pasará nada. El código que escribiste dentro de la función no hará nada hasta que hagas una **llamada** a dicha función.

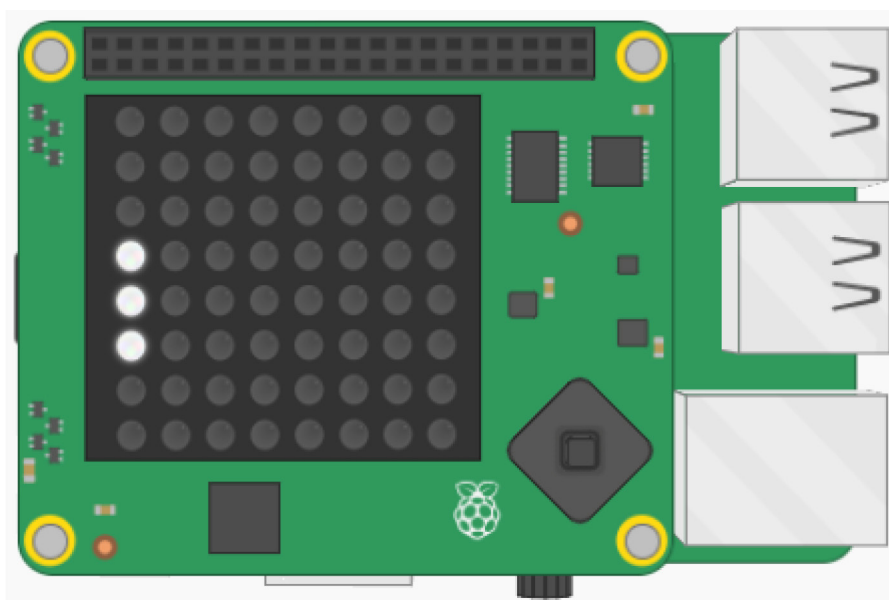
- Agregá un nuevo comentario debajo de la función para mostrar la sección donde el programa principal comienza. Asegurate que este comentario no esté indentado.

```
# Main program -----
```

- Agregá esta línea de código en el programa principal para llamar a la función:

```
draw_bat()
```

- Ejecutá el programa y corroborá que ahora se iluminan los tres LED.



## Mover la paleta

Hagamos que la paleta suba y baje cuando movés el joystick del Sense HAT.

- En tu sección de funciones, definí una nueva que se llame `move_up(event)`.

A esta función se le pasará algo de información por el argumento `event`. La información que esta función recibirá tiene que ver con lo que esté pasando con el joystick de Sense HAT. Esto incluye el momento en el que fue usado, la dirección, y si fue presionado, mantenido o soltado.

- Dentro de la función `move_up`, agregá un condicional `if` para probar si el `event.action` fue `'pressed'` (en otras palabras, si el joystick fue movido).

```
if event.action == 'pressed':
```

Si la condición se cumple, queremos que la paleta vaya hacia arriba. Hacia arriba en el sistema de coordenadas de nuestra pantalla LED significa hacer que la coordenada `y` sea más pequeña - recordá que la coordenada superior de `y` es igual a `0`.

- Si `event.action` es `'pressed'`, restale `1` a la coordenada `bat_y`. Esto nos permitirá redibujar la paleta en una posición diferente. Nota: como la variable `bat_y` se definió fuera de esta función, debemos decirle a Python que use la versión global de esta variable así se nos permite cambiarla desde dentro de la función.

```
< > main.py
8 # Functions -----
9 def draw_bat():
10     sense.set_pixel(0, bat_y, white)
11     sense.set_pixel(0, bat_y + 1, white)
12     sense.set_pixel(0, bat_y - 1, white)
13
14 def move_up(event):
15     global bat_y
16     if event.action == 'pressed':
17         bat_y -= 1
```

Recordá que al igual que con nuestra función `draw_bat`, ésta no hará nada hasta que se la **llame**.

- En nuestro programa principal, agregá esta línea de código arriba de la llamada a la función `draw_bat`. Esta línea dice "cuando el joystick del Sense HAT sea empujado hacia arriba, llama a la función `move_up`."

```
sense.stick.direction_up = move_up
```

Si ejecutás el programa en este momento, no sucederá nada. Esto es porque por ahora, sólo estamos detectando movimientos del joystick una sola vez, cuando la función es ejecutada. Para poder hacer que esta función sea útil a nuestro juego, debemos detectar continuamente si el joystick ha sido movido.

- En tu programa principal, debés poner la llamada a la función `draw_bat` dentro de un bucle infinito.

### Bucle While True en Python

El uso de un bucle **while** es el de repetir bloques de código una y otra vez siempre y cuando una condición sea `True`. Es por eso que los bucles while también son conocidos como **repeticiones condicionales**.

El ejemplo de aquí abajo es un bucle que se ejecutará por siempre - un bucle infinito o repetición incondicional. El bucle se ejecutará por siempre porque la condición es siempre `True`.

```
while True:
    print("Hola Mundo")
```

**Note:** La línea de código `while` declara la condición del bucle. La línea de código `print` que se encuentra debajo está levemente desplazada hacia la derecha. Esto se llama indentación - la línea está indentada para mostrar que está dentro del bucle. Todo el código dentro del bucle será repetido.

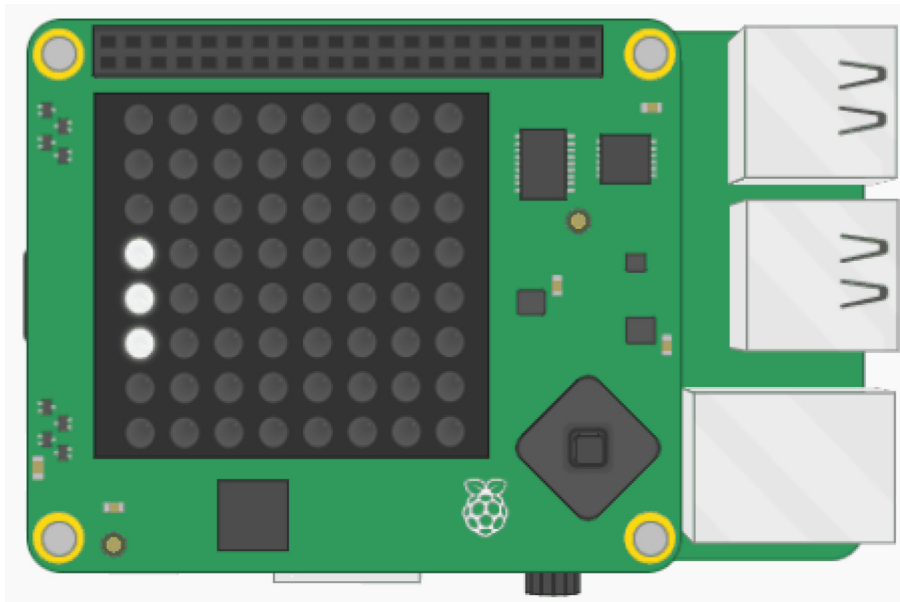
Los bucles infinitos son útiles en situaciones donde queremos realizar las mismas acciones una y otra vez, por ejemplo revisar el valor de un sensor. Un bucle infinito como este va a continuar funcionando por siempre, lo que significa que cualquier línea de código escrita luego del bucle no se ejecutará nunca. Esto se conoce como

**bloqueo** - porque el programa **bloquea** la ejecución del resto del código.

Si utilizaste Scratch alguna vez, esto será familiar, ya que es lo mismo que usar un bucle "por siempre".



- Guardá y ejecutá tu programa. Pulsa el joystick del Sense HAT hacia arriba (o usa las flechas del teclado en el emulador).



¿Qué sucedió? El resultado parece un poco como si estuviéramos pintando con la paleta hacia arriba en vez de moviéndonos. Debemos limpiar la pantalla y esperar un instante cada vez que vamos a dibujar la paleta en el bucle infinito.

- Agregá esta línea en el bucle infinito para limpiar la pantalla LED cada vez, antes de dibujar la paleta.

```
sense.clear(0, 0, 0)
```

- Para hacer que el programa espere un instante, agregá una línea dentro del bucle y después de `draw_bat` para `sleep` por 0.25 segundos.



## Usando el comando sleep de Python

Podés usar la función `sleep` para pausar temporalmente tu programa de Python.

- Agregá esta línea al principio de tu programa para importar la función `sleep`.

```
from time import sleep
```

- Cada vez que quieras una pausa en tu programa, llamá a la función `sleep`. El número entre paréntesis indica cuántos segundos queremos que dure la pausa.

```
sleep(2)
```

También podés pausar tu programa por fracciones de segundo.

```
sleep(0.5)
```

- Guardá y corré tu programa de nuevo. Intentá mover la paleta y corroborá que ahora se mueve hacia arriba como esperamos.

Si intentás mover la paleta demasiado arriba, tu programa intentará dibujarla fuera de la pantalla LED, y se colgará. Debés asegurarte que el valor de la variable `bat_y` nunca sea menor a `1`, para que la paleta se mantenga dentro de la pantalla todo el tiempo.

- Agrega código a tu función `move_up` para asegurarte que el valor de la variable `bat_y` nunca puede ser menor a `1`.

```
< > main.py
15 def move_up(event):
16     global bat_y
17     if event.action == 'pressed' and bat_y > 1:
18         bat_y -= 1
```

- Ahora seguí los pasos de nuevo, haciendo algunos cambios que permitan que la paleta se pueda mover hacia abajo en la pantalla además de hacia arriba.

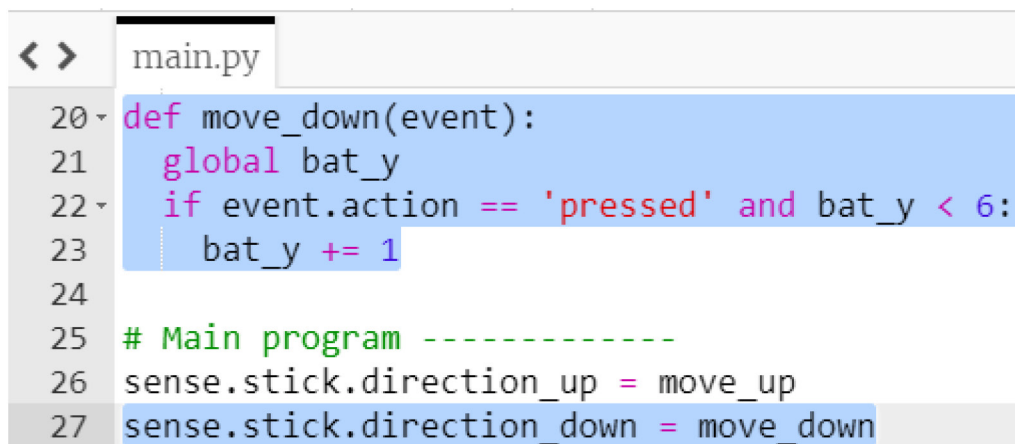
## Necesito una pista

Empezá por escribir una función `move_down(event)` que contenga instrucciones para cuando la paleta debe ser movida hacia abajo. Esta vez, deberías agregar `1` a `bat_y`, pero sólo si el valor de `bat_y` es menor a `6`, para que la paleta se mantenga en pantalla.

Necesitás usar otra línea de código en tu programa principal para llamar a la función `move_down` cuando el joystick es movido hacia abajo.

```
sense.stick.direction_down = move_down
```

El programa debería verse así:



```
< > main.py
20 def move_down(event):
21     global bat_y
22     if event.action == 'pressed' and bat_y < 6:
23         bat_y += 1
24
25 # Main program -----
26 sense.stick.direction_up = move_up
27 sense.stick.direction_down = move_down
```

## Crear una bola

El próximo paso es crear una bola. ¡Pero primero, un poco de matemática!

Una bola que se mueve en dos dimensiones tiene dos propiedades esenciales que debés considerar:

**Posición** - como la paleta, la bola tiene una coordenada horizontal y una vertical en la pantalla.

**Velocidad** - la velocidad de la bola en línea recta. Esto también puede ser descrito con dos números: qué tan rápido se mueve en la dimensión `x` y qué tan rápido se mueve en la dimensión `y`.

- Encontrá la variable `ball_y` en tu programa, y debajo de ella, agregá dos listas para describir las propiedades de la bola:

```
ball_position = [3, 3]
ball_velocity = [1, 1]
```

- Elegí un color para tu bola y, debajo de la variable `white`, definí una variable con el color elegido. Aquí usamos `(0, 0, 255)`, que es azul.
- En la sección de funciones, creá una función llamada `draw_ball`:

```
def draw_ball():
```

- Agregá una línea de código a la función `draw_ball:` que ilumine un LED en la `ball_position:`.

### Indexar una lista en Python

Una lista de Python es un tipo de estructura de datos. Puede almacenar grupos de cualquier tipo de dato, e inclusive una mezcla de tipos de dato.

- Aquí hay un ejemplo de una lista de cadenas de caracteres (strings) en Python:

```
band = ['paul', 'john', 'ringo', 'george']
```

- En Python las listas se indexan desde `0`. Esto significa que podemos referenciar al elemento número cero en una lista. En nuestro ejemplo, el elemento número cero es `'paul'`.
- Para encontrar el valor de un elemento en una lista, simplemente debés escribir el nombre de la lista seguido por el índice (número de elemento).

```
>>> band[0]
'paul'
>>> band[2]
'ringo'
```

- Para encontrar el valor del último elemento en una lista de Python, podés usar el índice `-1`.

```
>>> band[3]
'george'
>>> band[-1]
'george'
```

- Además, podés encontrar el valor del anteúltimo elemento usando el índice `-2`, y así sucesivamente.
- Puede que a veces necesites usar una lista de dos dimensiones, o lista de listas. Para encontrar un elemento determinado, debés proveer dos índices. Aquí hay una lista representando un ta-te-ti:

```
board = [['X', 'O', 'X'],
          ['O', 'X', 'O'],
          ['O', 'O', 'X']]
```

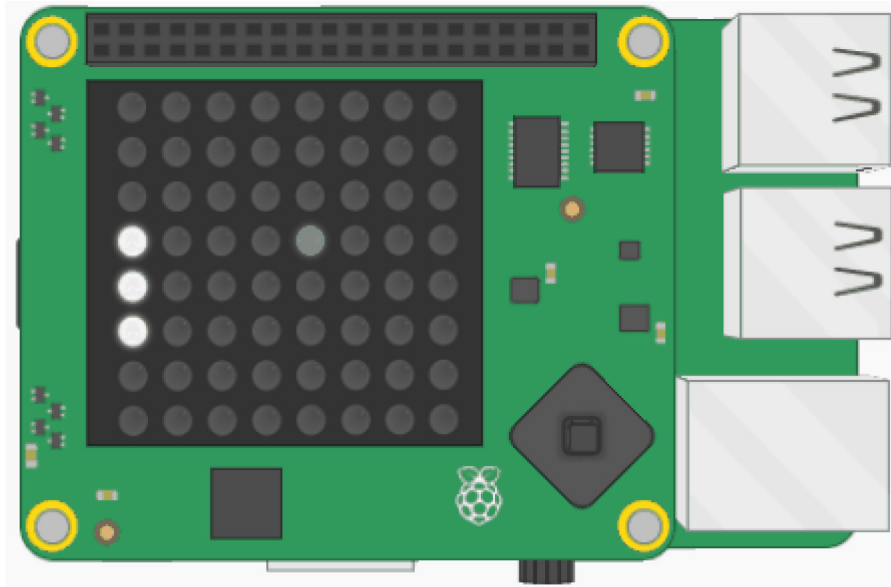
- Para encontrar el caracter central en esta lista de listas, deberías usar `board[1][1]`.

### Necesito una pista

La posición en el eje `x` será el item cero en la lista `ball_position` list. La posición en `y` será el item uno en la lista `ball_position`.

```
def draw_ball():
    sense.set_pixel(ball_position[0], ball_position[1], blue)
```

- En tu bucle `while`, hacé una llamada a la función `draw_ball`.
- Guardá y ejecutá tu programa, y corroborá que la bola aparece en la pantalla LED.



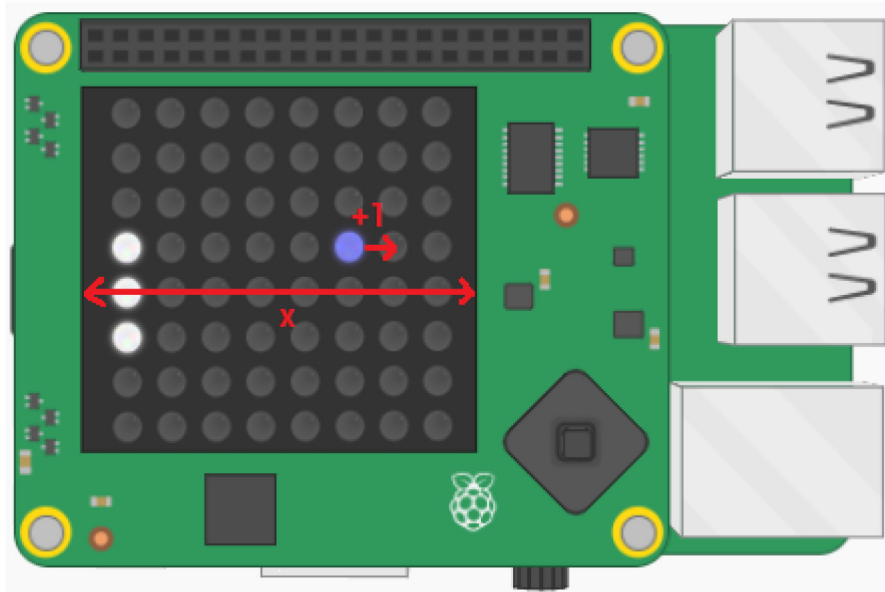
## Mové la bola

Para que la bola se mueva, debés cambiar su posición en `x` por su velocidad en `x`, y su posición en `y` por su velocidad en `y`.

La primer coordenada en cada una de las listas que has creado representa la propiedad de la bola en `x` - entonces `ball_position[0]` es su coordenada actual en `x`, y `ball_velocity[0]` es cuán rápido debe moverse en `x`.

- Dentro de tu función `draw_ball`, agregá esta línea de código para agregar la velocidad de la bola (en este momento `1`) a la posición actual de la bola en el eje `x`.

```
ball_position[0] += ball_velocity[0]
```



- Guardá y ejecutá tu programa. Ahora la bola se moverá a lo largo de la pantalla hasta llegar al borde, y luego se colgará tu programa. ¿Por qué creés que esto sucede?

### Respuesta

Seguramente hayas visto el mismo error cuando intentabas mover la paleta. La bola se mueve a lo largo de la pantalla hasta que el programa se cuelga con el error `ValueError: x position must be between 0 and 7`.

La bola se movió a una posición en `x` mayor a 7, lo que es fuera de la pantalla LED.

- Inmediatamente después de la línea de código que mueve la bola, agregá un condicional que cuando evalúe que la `ball_position[0]` llega a `7`, la velocidad se invierta así se desplaza en la otra dirección:

```
if ball_position[0] == 7:
    ball_velocity[0] = -ball_velocity[0]
```

- Guardá y corré tu programa de nuevo. La bola debería rebotar en el borde derecho de la pantalla - ¡pero cuando llegue al borde izquierdo recibirás otro error porque la bola sigue intentando salirse de la pantalla del lado izquierdo!
- Modificá tu condicional para que la bola invierta su dirección si la posición es igual a `7` or igual a `0`.

## Necesito una pista

Agregá tu condición adicional en el lugar resaltado en azul:

```
< > main.py
28 def draw_ball():
29     sense.set_pixel(ball_position[0], ball_position[1], blue)
30     ball_position[0] += ball_velocity[0]
31     if ball_position[0] == 7 or ???:
32         ball_velocity[0] = -ball_velocity[0]
```

Tu código debería verse así:

```
if ball_position[0] == 7 or ball_position[0] == 0:
    ball_velocity[0] = -ball_velocity[0]
```

## ¿Por qué funciona?

La velocidad de la bola empieza en 1. Si su posición en x llega a 7, cambiamos la velocidad en x a -1 para hacer que cambie de dirección. Entonces el programa agregará -1 a la posición de la bola en x para moverla hacia la izquierda en la pantalla.

¿Pero por qué funciona cuando la bola llega al extremo izquierdo? Mirá el código:

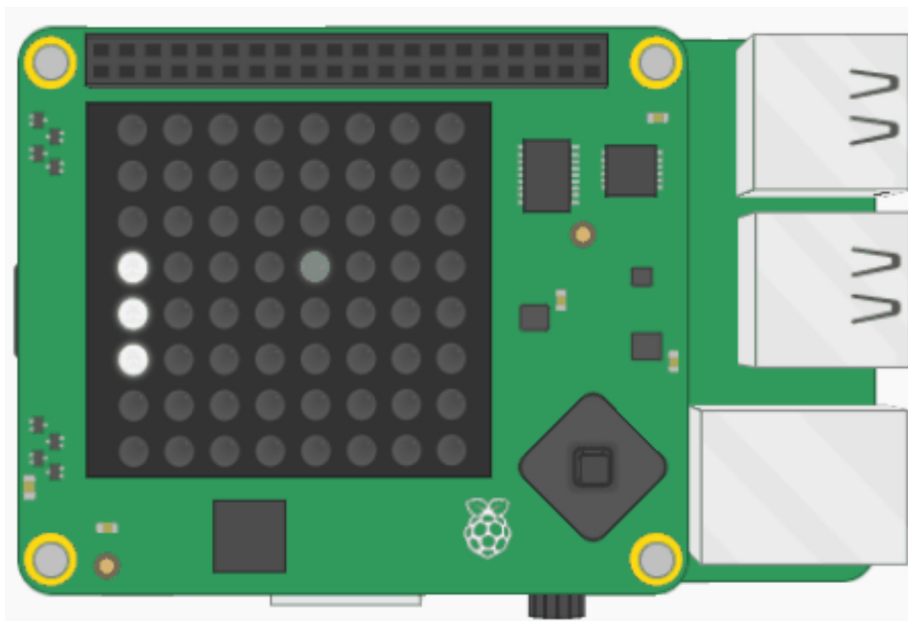
```
ball_velocity[0] = -ball_velocity[0]
```

Cuando la bola se traslada hacia la izquierda, su velocidad en x es de -1. Cuando usamos este valor en la línea de código, obtenemos lo siguiente:

```
ball_velocity[0] = -(-1)
```

Menos (menos uno) es igual a...¡uno positivo! Entonces ahora la velocidad es 1, y la bola comienza a viajar en el otro sentido.

- Guardá y ejecutá tu programa para comprobar que la bola ahora rebota felizmente entre los bordes izquierdo y derecho.



- Ahora hacé que la bola se mueva de acuerdo tanto a su velocidad en `y` como a su velocidad en `x` siguiendo estos pasos de nuevo con algunos cambios.

### Necesito una pista

Comenzá por agregar una línea de código al final de la función `draw_ball` para hacer que la bola se mueva de acuerdo a `ball_position[1]` y `ball_velocity[1]`. Esta línea es casi idéntica a la que utilizaste para cambiar la coordenada en `x` de la bola.

Luego, agregá un condicional que invierta la dirección de la bola si la posición en `y` llega a `0` o `7`. De nuevo, para hacer esto podés usar el código que agregaste para la posición en `x` con algunos cambios.



El código resaltado es la parte que debés agregar:

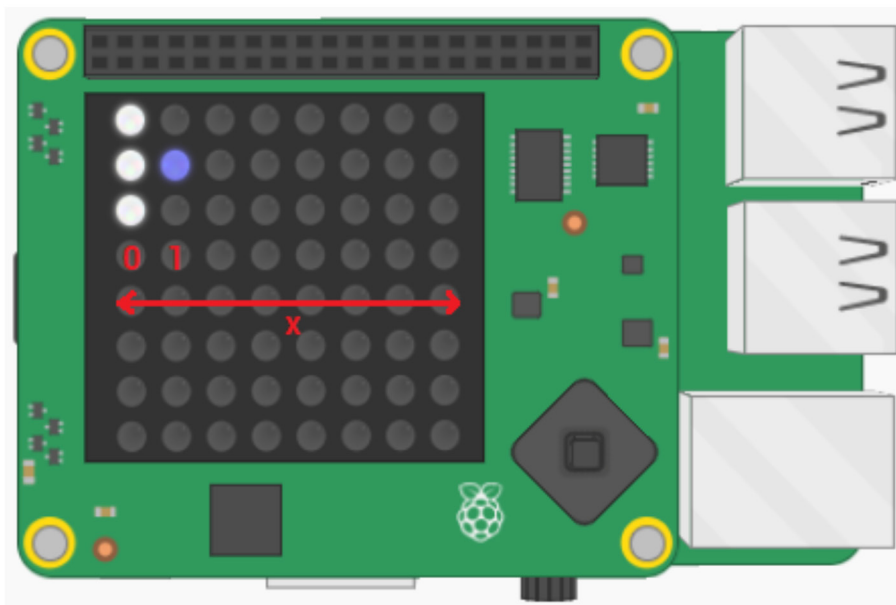
```
< > main.py
28 def draw_ball():
29     sense.set_pixel(ball_position[0], ball_position[1], blue)
30     ball_position[0] += ball_velocity[0]
31     if ball_position[0] == 7 or ball_position[0] == 0:
32         ball_velocity[0] = -ball_velocity[0]
33     ball_position[1] += ball_velocity[1]
34     if ball_position[1] == 7 or ball_position[1] == 0:
35         ball_velocity[1] = -ball_velocity[1]
```

## Colisión con la paleta

Ahora que la bola rebota en todas las direcciones, hagamos que rebote cuando se encuentra con la paleta.

La paleta siempre está en la columna izquierda de la pantalla, por lo que su coordenada en `x` es siempre `0`.

La bola va a rebotar en la paleta cuando esté en la columna siguiente a la paleta - esto es, si su posición en `x` es igual a `1`.



- Agregá este código al final de la función `draw_ball` :

```
if ball_position[0] == 1:  
    ball_velocity[0] = -ball_velocity[0]
```

Este bloque de código hará que la bola cambie de dirección si llega a una coordenada en `x` igual a `1`. ¡Pero ahora la bola rebota sin importar que la paleta esté allí o no!

- Agregá una condición que requiera que la posición en `y` también (**and**) esté entre la parte superior e inferior de la paleta.

Recordá que la paleta está hecha con tres pixeles. Entonces para que la bola "rebote" en la paleta, la coordenada en `y` de la bola puede tomar cualquier valor entre la parte superior `(bat_y - 1)` y la parte inferior `(bat_y + 1)` de la paleta.

## Necesito una pista

Agregá tu condición adicional en la zona resaltada en azul:

```
< > main.py  
28 def draw_ball():  
29     sense.set_pixel(ball_position[0], ball_position[1], blue)  
30     ball_position[0] += ball_velocity[0]  
31     if ball_position[0] == 7 or ???:  
32         ball_velocity[0] = -ball_velocity[0]
```

Para revisar si un valor se encuentra entre otros dos, podemos escribir una condición así:

```
1 <= x <= 10
```

Esta condición revisa si `x` se encuentra entre `1` y `10` (inclusive) preguntándose primero si `1` es menor o igual a `x`, y luego si `x` es menor o igual a `10`. Usá una línea de código parecida para determinar si la coordenada en `y` de la bola se encuentra entre `(bat_y - 1)` y `(bat_y + 1)`.

Así debería verse el código terminado. La parte a agregar está resaltada en azul:

```
< > main.py
28 def draw_ball():
29     sense.set_pixel(ball_position[0], ball_position[1], blue)
30     ball_position[0] += ball_velocity[0]
31     if ball_position[0] == 7 or ball_position[0] == 0:
32         ball_velocity[0] = -ball_velocity[0]
33     ball_position[1] += ball_velocity[1]
34     if ball_position[1] == 7 or ball_position[1] == 0:
35         ball_velocity[1] = -ball_velocity[1]
36     if ball_position[0] == 1 and (bat_y - 1) <= ball_position[1] <= (bat_y + 1):
37         ball_velocity[0] = -ball_velocity[0]
```

- Guardá y ejecutá tu programa. ¡Corroborá que la bola sólo rebota en la paleta cuando ella está en la posición correcta!

## Has perdido

En este momento, cuando no logramos golpear la bola con la paleta, la primera sólo rebota en la pared izquierda. Cambiemos el código para que el juego termine si el jugador no golpea la bola.

- Agregá otro condicional al final de la función `draw_ball` para revisar si la posición de la bola en `x` es igual a `0`, lo que significa que la bola ha llegado a la esquina izquierda de la pantalla.
- Si esta condición es verdadera, mostrará el mensaje “You lose”.

## Necesito una pista

Tu nuevo condicional se verá muy similar a los anteriores. Agregalo aquí:

```
< > main.py
28 def draw_ball():
29     sense.set_pixel(ball_position[0], ball_position[1], blue)
30     ball_position[0] += ball_velocity[0]
31     if ball_position[0] == 7 or ball_position[0] == 0:
32         ball_velocity[0] = -ball_velocity[0]
33     ball_position[1] += ball_velocity[1]
34     if ball_position[1] == 7 or ball_position[1] == 0:
35         ball_velocity[1] = -ball_velocity[1]
36     if ball_position[0] == 1 and (bat_y - 1) <= ball_position[1] <= (bat_y + 1):
37         ball_velocity[0] = -ball_velocity[0]
38     # Add your code here
39
```

Así se debería ver tu código. La parte agregada está resaltada en azul:

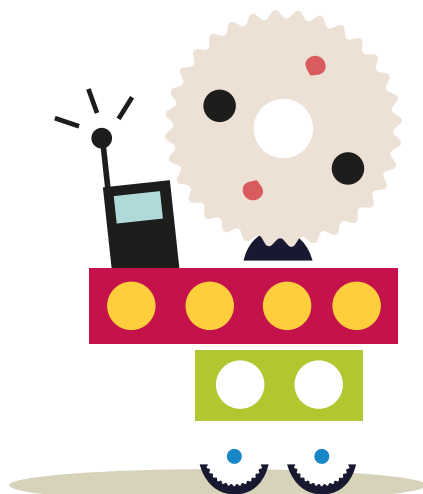
```
< > main.py
28 def draw_ball():
29     sense.set_pixel(ball_position[0], ball_position[1], blue)
30     ball_position[0] += ball_velocity[0]
31     if ball_position[0] == 7 or ball_position[0] == 0:
32         ball_velocity[0] = -ball_velocity[0]
33     ball_position[1] += ball_velocity[1]
34     if ball_position[1] == 7 or ball_position[1] == 0:
35         ball_velocity[1] = -ball_velocity[1]
36     if ball_position[0] == 1 and (bat_y - 1) <= ball_position[1] <= (bat_y + 1):
37         ball_velocity[0] = -ball_velocity[0]
38     if ball_position[0] == 0:
39         sense.show_message("You lose")
```

- Guardá y ejecutá tu programa. Corroborá que si le errás a la bola, aparezca el mensaje "You lose". El juego reiniciará luego de mostrar el mensaje.

## Desafío: agrega más funcionalidad

- Agregá un puntaje que aumente cada vez que la bola rebota en la paleta. Muestra el puntaje al final del juego.
- Hacé el juego más fácil o difícil cambiando el tiempo del período `sleep` en el bucle.
- Dale al jugador tres vidas: si erran, pierden una vida y la bola se reestablece. Una vez que se pierden las tres vidas, el juego termina.

La fundación Raspberry Pi suministra contenidos de aprendizaje de programación sin cargo. Encuentre más información en <https://projects.raspberrypi.org/en/> (inglés)



**APRENDER  
CONECTADOS**



Ministerio de Educación,  
Cultura, Ciencia y Tecnología  
**Presidencia de la Nación**