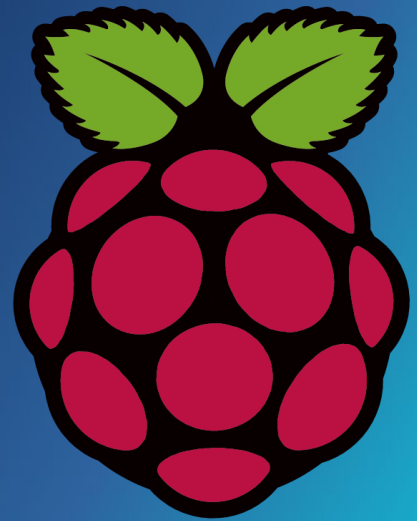
Raspberry Pi: usando GPIO con Python

y gpiozero



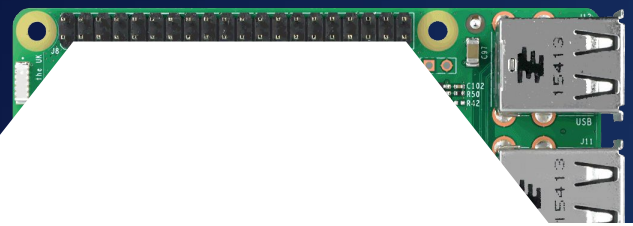
¿Qué son los **GPIO**?

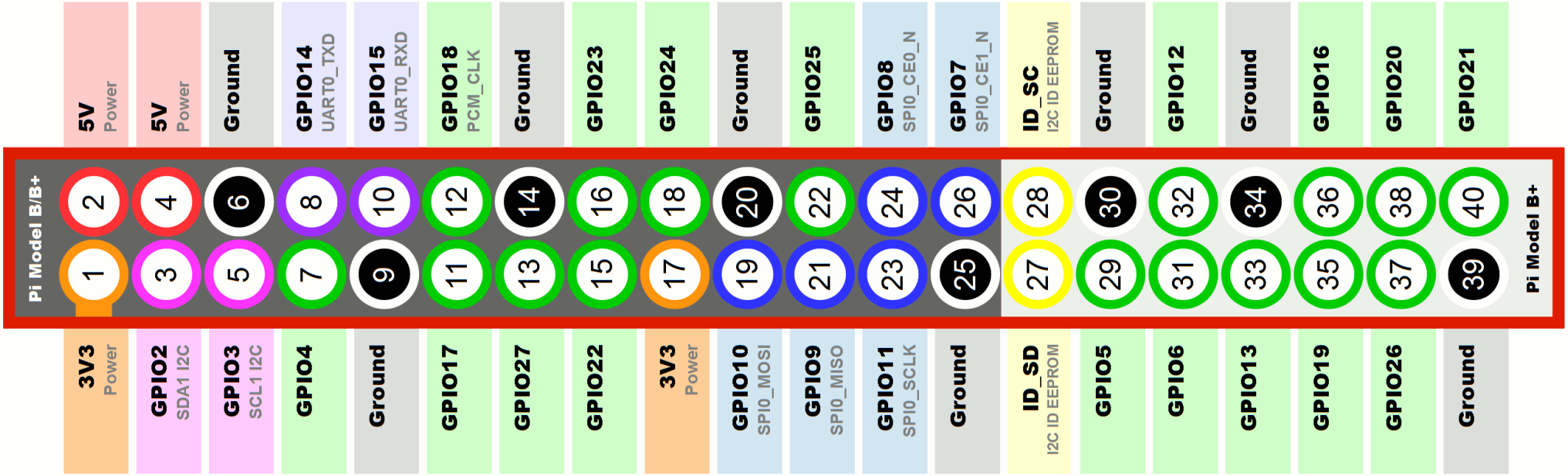
GPIO: General Purpose Input-Output

(pines de entrada-salida de propósito general)

Nos permiten ‘leer’ y ‘escribir’ señales digitales (apagado/encendido, 0V/5V) en cada uno de sus pines.

GPIO en Raspberry Pi 3 y 2





Fuente:
https://www.raspberrypi-spy.co.uk/2012/06/simple-guide-to-the-rpi-gpio-header-and-pins/

Instalando *gpiozero* en Python

- Partimos de una Raspberry Pi con Raspbian instalado
- Usaremos la librería *gpiozero* para Python
- Viene preinstalada en las últimas versiones de Raspbian Desktop
- Si no la tuviésemos, se instala con el comando:

  `sudo apt install python3-gpiozero`
- Probar si está instalado escribiendo los comandos:
  - `python3` *(en Raspbian)*
  - `import gpiozero` *(dentro de Python)*
    - Si no da ningún error: está instalado
- Documentación: https://gpiozero.readthedocs.io/en/stable/

github.com/pythoneiro

Blinking LED: el “Hello World” de los LED

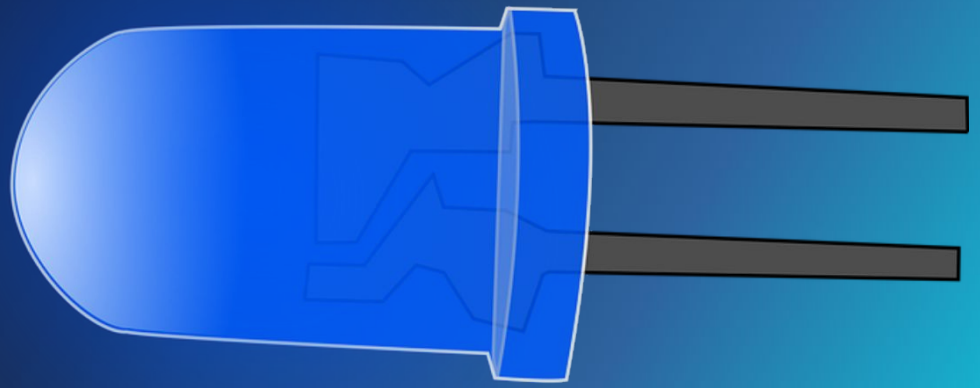
El ejemplo más sencillo que podemos hacer con un LED y los GPIO de Raspberry es hacer parpadear un LED

Accedemos al intérprete de Python ejecutando python3 en terminal.

Escribimos las siguientes líneas:

```
from gpiozero import LED
from time import sleep

led = LED(17)

while True:
    led.on()
    sleep(1)
    led.off()
    sleep(1)
```

Parar la ejecución con Ctrl+C



1-LED/0_blink_manual.py

github.com/pythoneiro

Blinking LED desde *gpiozero*

Este ejemplo está bien didácticamente, pero *gpiozero* ya incluye un método para hacer parpadear el LED las veces que queramos.

```
from gpiozero import LED

led = LED(17)

# El LED parpadea, en segundo plano
led.blink()

# Finalizar el parpadeo cuando queramos:
led.off()

# Personalizar el tiempo que está encendido y apagado
# 0.5 segundos encendido, 0.25 segundos apagado
led.blink(on_time=0.5, off_time=0.25)

# Hacer que sólo parpadee 3 veces
led.blink(n=4)
```

1-LED/1_blink_gpiozero.py
1-LED/2_blink_tiempos.py
1-LED/3_blink_n_veces.py

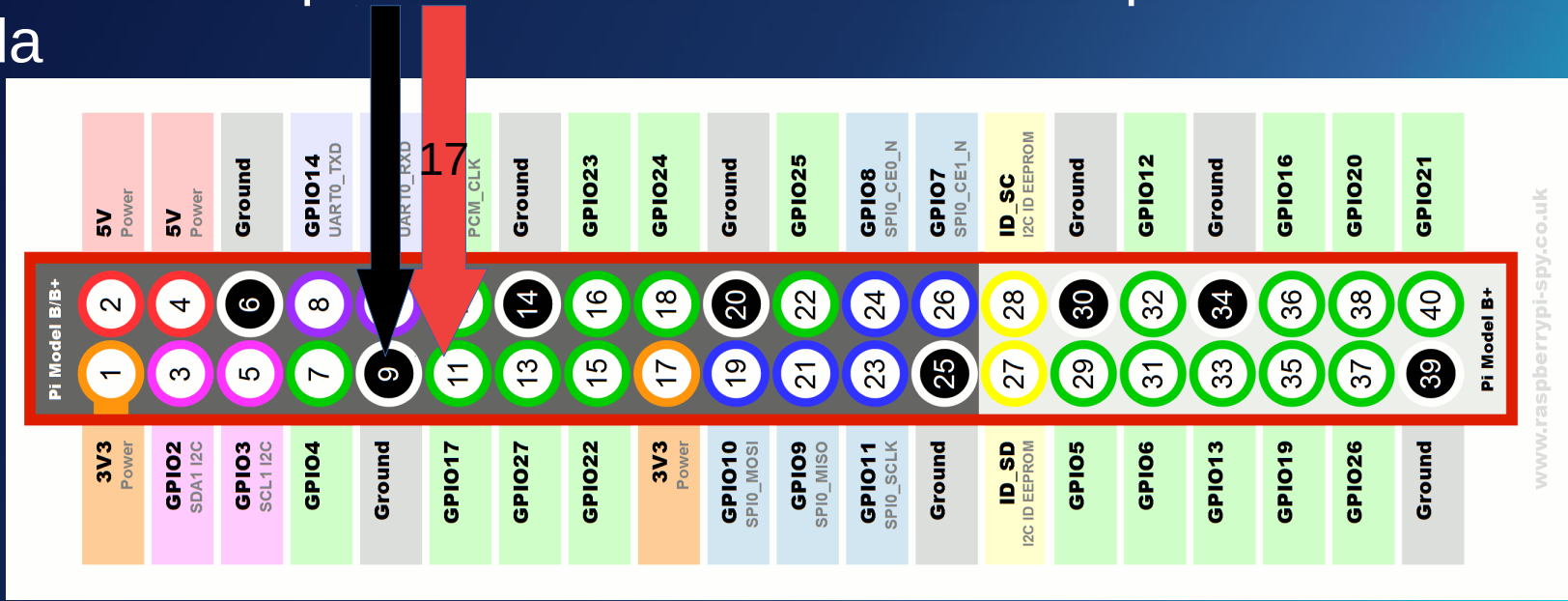
Los números de pines GPIO

En el ejemplo anterior declaramos el Pin que vamos a usar con:

```
led = LED(17)
```

El número 17 corresponde al pin marcado como GPIO17.

Debemos usar sólo los pines marcados como GPIOx para entrada/salida



github.com/pythoneiro

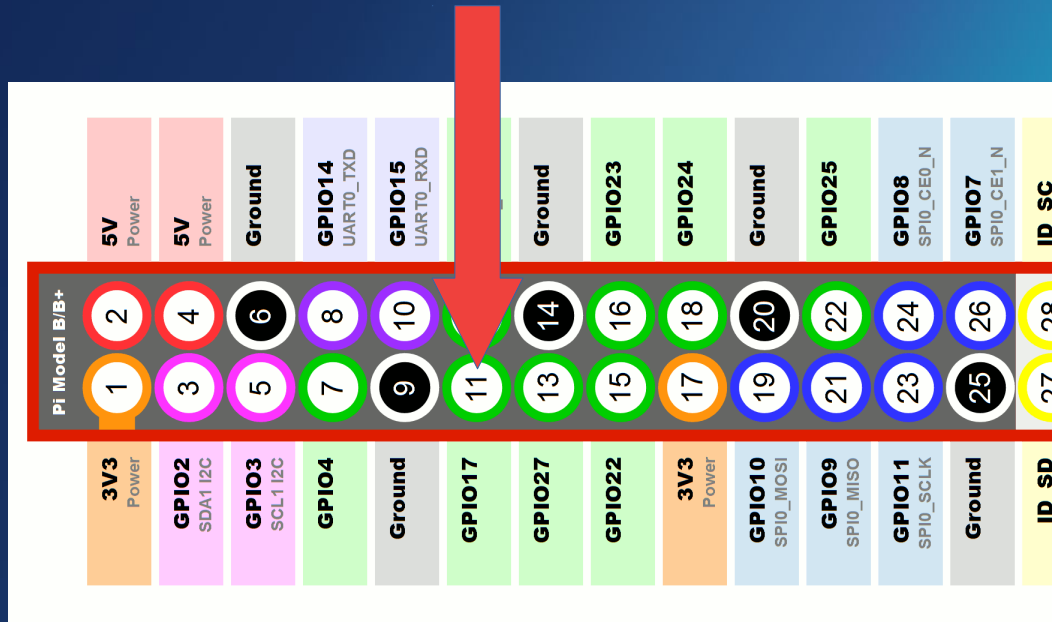
Usar los números de pines secuenciales

Si quisiéramos usar los números de pines en orden secuencial (los números dentro de los agujeros), podemos usar **`"BOARDxx"`** como número de pin.

En el mismo ejemplo: el pin GPIO17 es el pin número 11 en la placa.

Si queremos usar el número 11,
lo inicializaríamos así:

```
led = LED("BOARD11")
```



github.com/pythoneiro

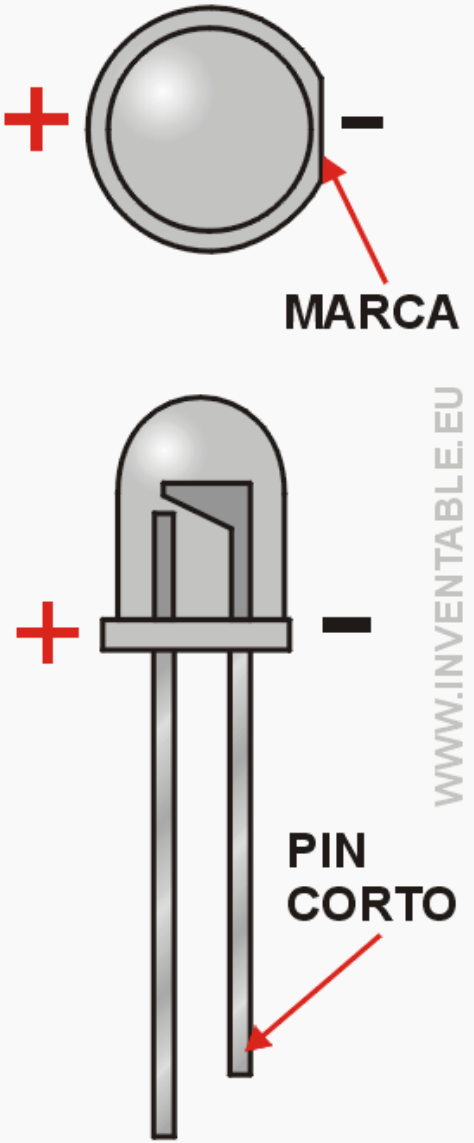
1-LED/4_blink_board_pin.py

Precauciones

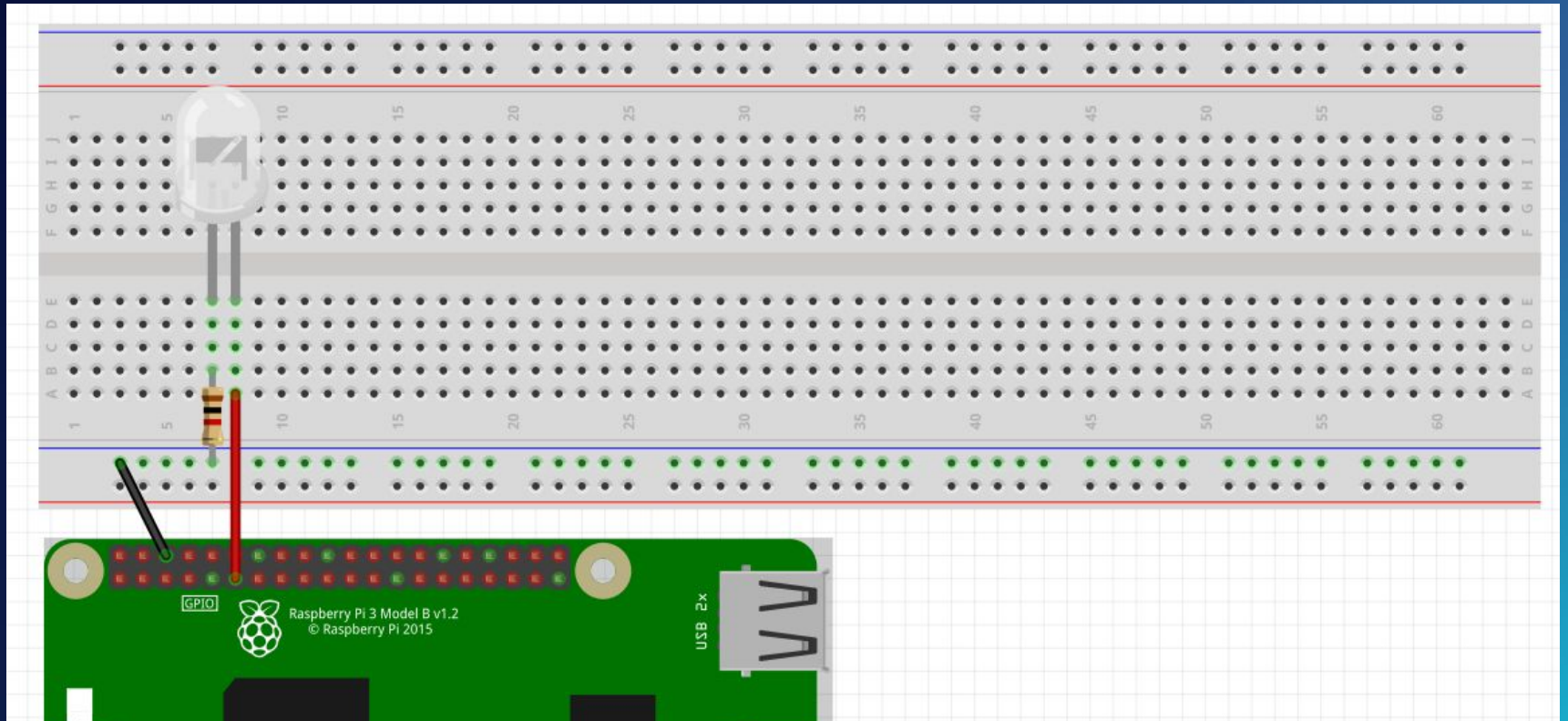
- **No cortocircuitar los pines** (por lo general, no juntar nunca dos cables conectados a pines distintos)
- Usar resistencias para **limitar la corriente** en las salidas (no siempre es necesario, pero más vale prevenir que tener que comprar otra Raspberry porque la nuestra se ha quemado)
- **No conectar cargas pesadas directamente** en los pines (p.ej. bombillas, LEDs potentes, relés, motores…)
  En estos casos, usar placas de relés preparadas, transistores MOSFET...
- **Respetar la polaridad** de los componentes y conexiones
- Tener en cuenta que **los GPIO de Raspberry trabajan a 3.3V**

LED
Esquema de conexión

- Componentes necesarios (usando Breadboard + resistencia):
  - 1 LED
  - 1 Resistencia ~ 1K (opcional)
  - 2 cables Dupont macho-hembra
  - 1 placa Breadboard/protoboard
- Componentes necesarios (sin usar Breadboard ni resistencia):
  - 1 LED
  - 2 cables Dupont hembra-hembra
- **Prestar atención a la polaridad del LED:**
  - Ánodo (+) a GPIO7
  - Cátodo (-) a GND



LED
Esquema de conexión



github.com/pythoneiro

Liberar un pin

- Cuando inicializamos un pin, éste queda ocupado
- Cuando dejamos de trabajar con él, conviene cerrarlo:

```
led = LED(17)

# aquí está todo lo que hacemos con el LED
# y cuando queremos liberar su pin:

led.close()
```

- Cuando cerramos Python, los pines se liberan automáticamente

Cambiar luminosidad del LED

Usando PWM, podemos ajustar la luminosidad del LED.

```
from gpiozero import PWMLED
from time import sleep

led = PWMLED(17)

while True:
    led.value = 0  # apagado
    sleep(0.5)
    led.value = 0.25
    sleep(0.5)
    led.value = 0.5  # 50%
    sleep(0.5)
    led.value = 0.75  # 75%
    sleep(0.5)
    led.value = 1  # 100%
    sleep(0.5)
```

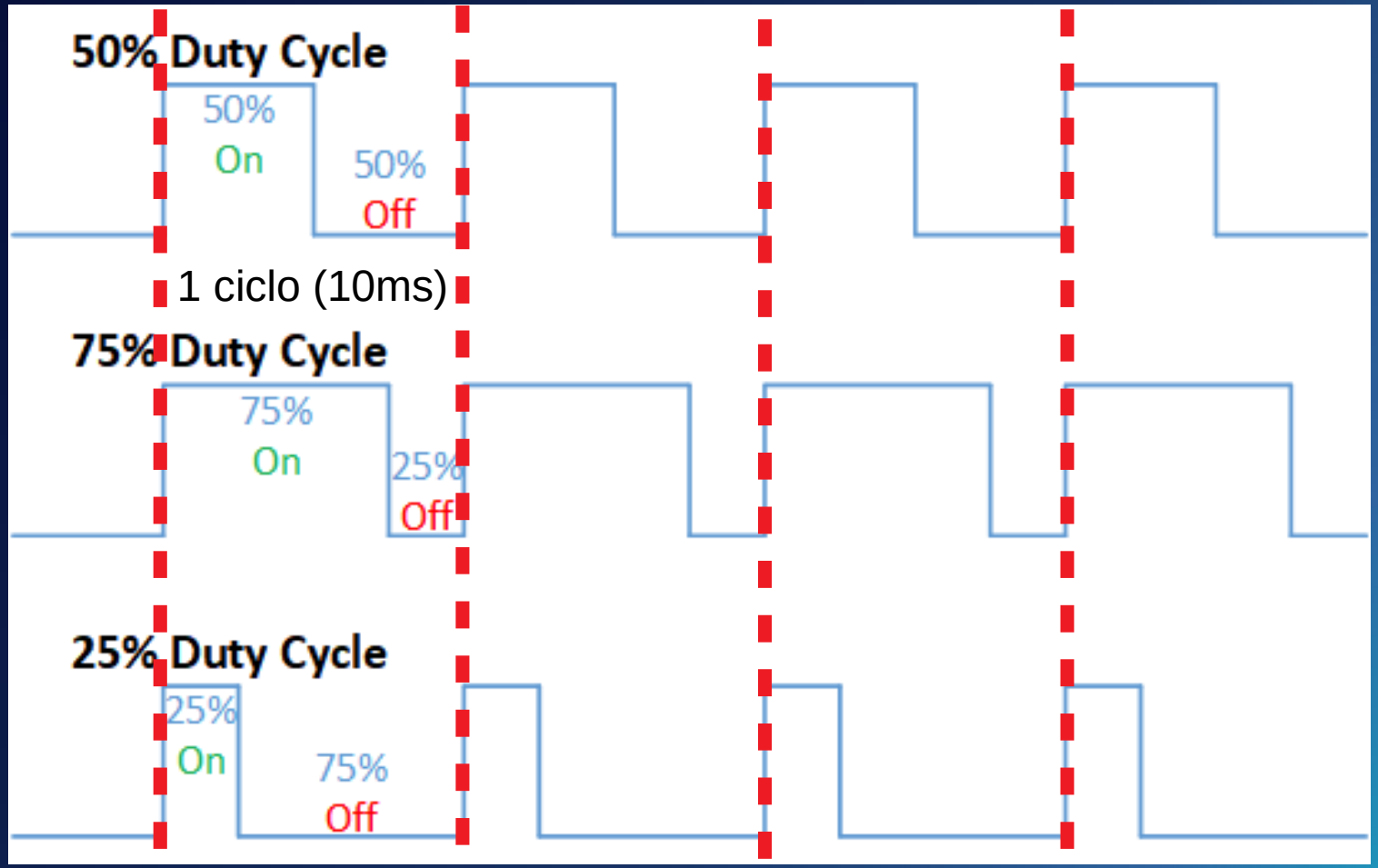
Parar la ejecución con Ctrl+C

1-LED/5_blink_pwm.py

github.com/pythoneiro

La modulación PWM

Por defecto en *gpiozero*: 100Hz (10ms por ciclo)



github.com/pythoneiro

Probando a cambiar la frecuencia PWM

Gpiozero nos permite personalizar la frecuencia del funcionamiento del PWM.

Vamos a ver qué pasa si usamos una frecuencia de 1Hz (1 ciclo por segundo).

```
from gpiozero import PWMLED
from time import sleep

led = PWMLED(17, frequency=1)
led.value=0.5
```

Con led.value=0.5 especificamos un duty cycle del 50%.

Como cada ciclo dura 1 segundo, el LED permanece:

- Encendido el 50% de 1 segundo (0.5 segundos)
- Apagado el 50% restante de 1 segundo (0.5 segundos)

github.com/pythoneiro

Probando a cambiar la frecuencia PWM

Con un duty cycle del 75%:

```
from gpiozero import PWMLED
from time import sleep

led = PWMLED(17, frequency=1)
led.value=0.75
```

Con led.value=0.75 especificamos un duty cycle del 75%.

Como cada ciclo dura 1 segundo, el LED permanece:

- Encendido el 75% de 1 segundo (0.75 segundos)
- Apagado el 25% restante de 1 segundo (0.25 segundos)

Frecuencia predeterminada de PWM en gpiozero: 100Hz

Función Pulse sobre un LED

Con Pulse logramos un efecto “respiración/latido” (como un blink, pero el LED se enciende y apaga suavemente, no de golpe). Es necesario usar PWMLED.

```python
from gpiozero import PWMLED

led = PWMLED(17)

led.pulse(
    fade_in_time=0.5,  # tiempo hasta encenderse
    fade_out_time=2,  # tiempo hasta apagarse
    background=False  # ejecutar en primer plano
)
```

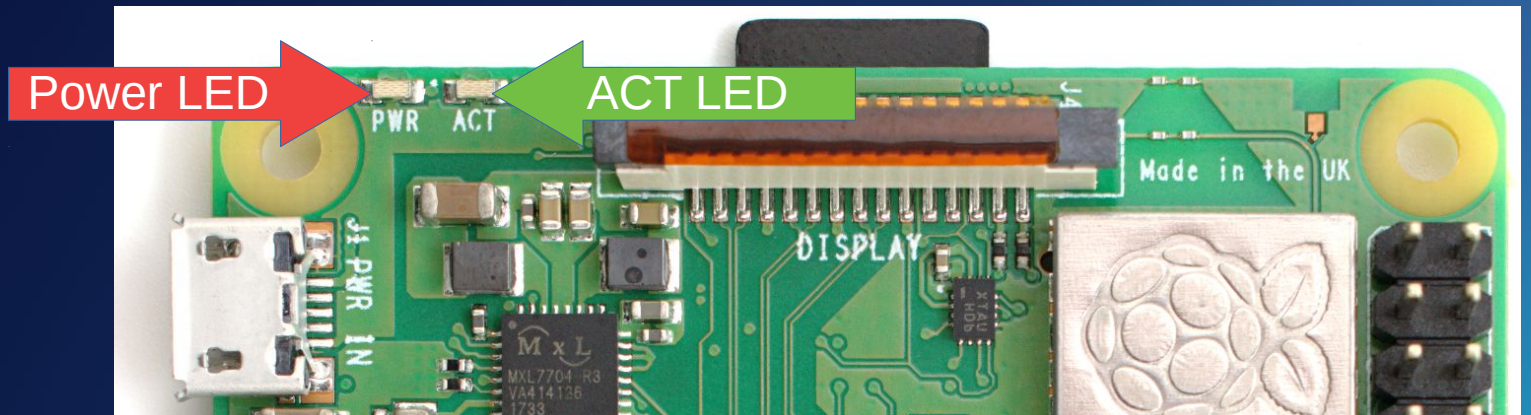
Parar la ejecución con Ctrl+C

Controlando los LED de Raspberry Pi

Raspberry tiene en placa dos LEDs:

- Rojo (POWER), encendido cuando la placa está encendida
- Verde (ACTIVITY), parpadea cuando se lee/escribe en disco

Es posible controlar a voluntad estos LED desde *gpiozero*.



Controlando los LED de Raspberry Pi

En primer lugar, hay que desactivarlos en Raspbian para que el sistema deje de usarlos.

```
echo none | sudo tee /sys/class/leds/led0/trigger
echo gpio | sudo tee /sys/class/leds/led1/trigger
```

Fuente:
https://gpiozero.readthedocs.io/en/stable/recipes_advanced.html#controlling-the-pi-s-own-leds

Para volver a dejarlos como estaban, reiniciar la Raspberry o ejecutar:

```
echo mmc0 | sudo tee /sys/class/leds/led0/trigger
echo input | sudo tee /sys/class/leds/led1/trigger
```

github.com/pythoneiro

Controlando los LED de Raspberry Pi

Ahora ya podemos instanciarlos y usarlos desde Python, como cualquier otro LED, incluso en modo PWM:

```
from gpiozero import LED, PWMLED

power = LED(35)
activity = PWMLED(47)

# Los LED parpadean, en segundo plano
power.blink(on_time=0.5, off_time=0.5)
activity.pulse(fade_in_time=1, fade_out_time=2)
```

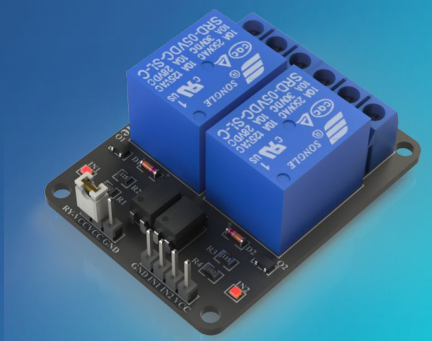
Al instanciar los LED, posiblemente salga un error, pero funcionan igualmente.

1-LED/7_leds_placa_raspberry.py

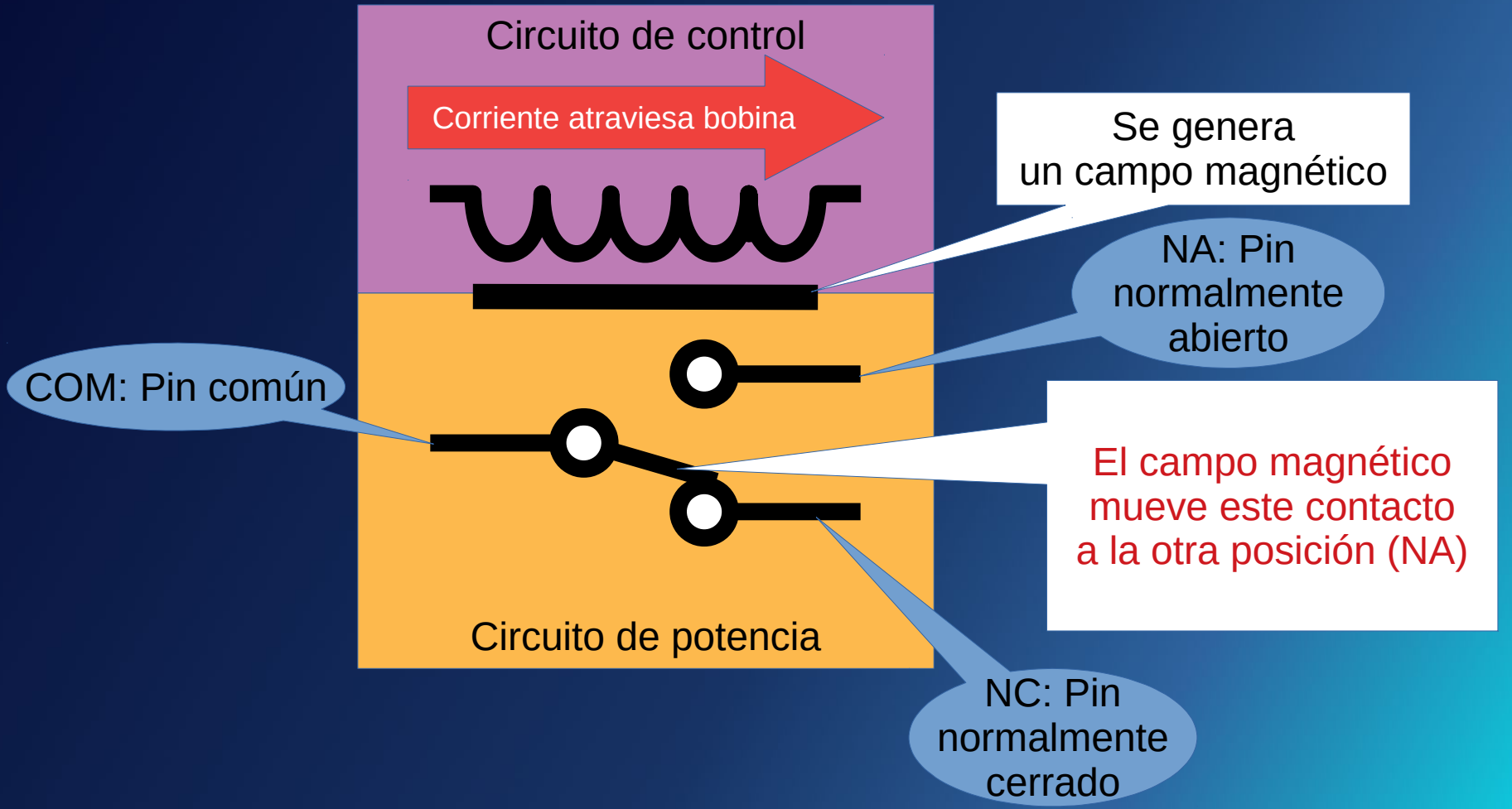
github.com/pythoneiro

Controlando relés

- Los relés son como interruptores que se abren o cierran cuando una corriente circula por su bobina.
- El circuito de control y la carga que controlamos están aislados.
- Esto nos puede servir, por ejemplo:
  - para controlar una luz de casa (230V AC) con nuestra Raspberry (3.3V DC de los GPIO)
  - para controlar un motor, una tira de LEDs, o cualquier carga que requiera más corriente de la que los GPIO de Raspberry pueden suministrar



Funcionamiento de un relé



github.com/pythoneiro

Partes del circuito de control de un módulo relé

- Vamos a suponer que utilizamos una placa de relé preparada para ser controlada con señales digitales de control.
  - Esto nos facilita mucho las cosas, porque para controlar un relé desde Raspberry, Arduino y otros microcontroladores, no se puede utilizando directamente sus salidas digitales, ya que la bobina consume demasiada corriente.
- El circuito de control dispone de dos conexiones distintas:
  - La alimentación propia de la bobina (normalmente 5V);
    Esta alimentación permanece siempre conectada
  - La señal de control (GPIO de Raspberry)

El problema de los relés y los 3.3V de Raspberry

- La mayoría de estos módulos de relés se controlan con señales de 5V…
- ...pero Raspberry utiliza lógica de 3.3V
- Estas placas pueden controlarse de dos formas:
  - Nivel bajo: el relé se activa cuando la señal de control es 0 (GND)
  - Nivel alto: el relé se activa cuando la señal de control es 1 (5V)
  - Las placas más comunes son las de nivel bajo, pero...
  - ...hay placas que permiten personalizar para cada relé si se activan a nivel bajo o alto

El problema de los relés y los 3.3V de Raspberry

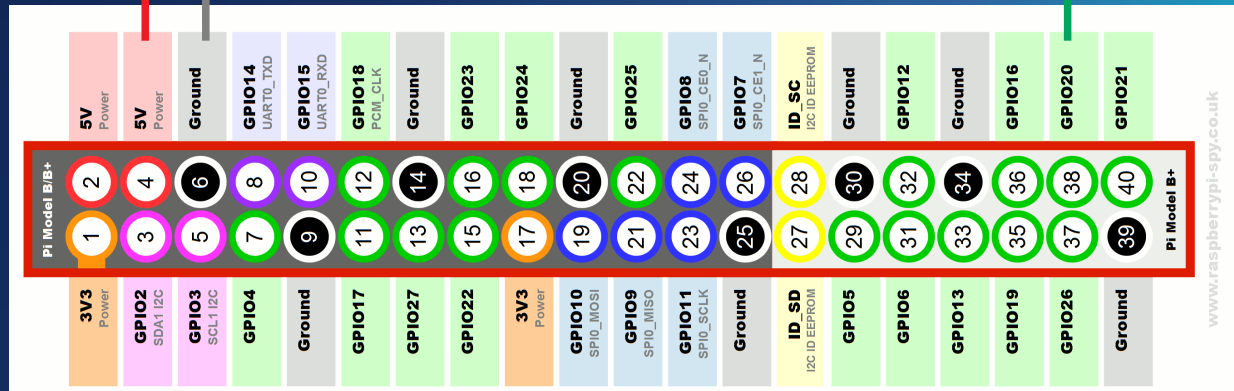
- Soluciones:
  - Buscar un módulo de relé/s compatible con lógica 3.3V…
  - ...o un módulo que pueda controlarse adecuadamente con 3.3V
- Ejemplo de un módulo probado que a mí me funciona:
  https://es.aliexpress.com/item/One-1-Channel-5V-Relay-Module-Board-Shield-with-Optocoupler-Support-High-and-Low-Level-Trigger/32742580606.html
  - (Variante 5V)
  - Permite cambiar para cada relé si se activa en nivel bajo o alto
  - Funciona configurándolo como activo en nivel alto (H)

WORKING VOLTAGE	5V	12V	24V
STATIC CIRCUIT	5mA		
MAXIMUM CURRENT	190mA	80mA	50mA
TRIGGER VOLTAGE	low: 0-1.5V	low: 0-1.4V	low: 0-8V
	high: 3-5V	high: 5-12V	high: 9-24V
TRIGGER CURRENT	2-4mA		
MAXIMUM LOAD	AC250V/10A,DC30V/10A		

WAVGAT

Esquema de conexión
Módulo de relés

Jumper entre medio y H



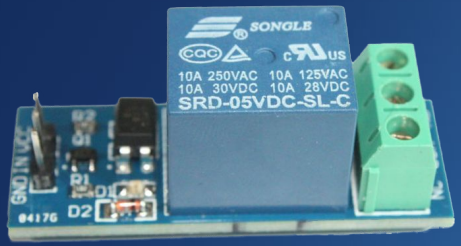
github.com/pythoneiro

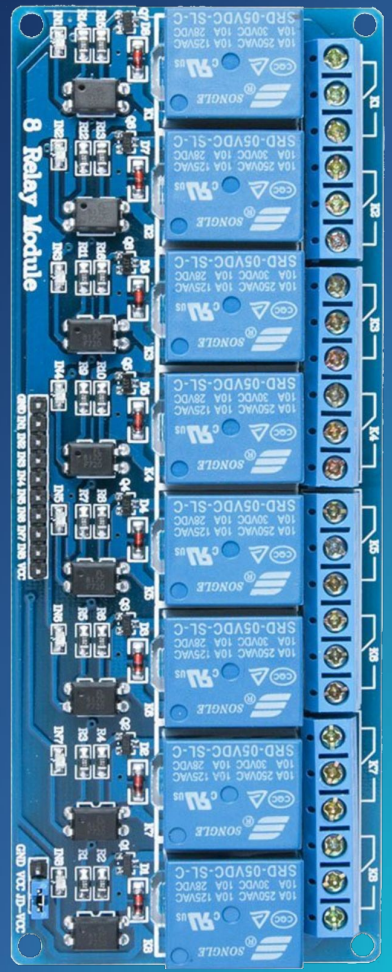
www.raspberrypi-spy.co.uk

Controlando el relé desde gpiozero

- La metodología es muy parecida a la de controlar un LED: es un dispositivo de salida que encendemos o apagamos.
- Vamos a instanciar un OutputDevice genérico para ello.

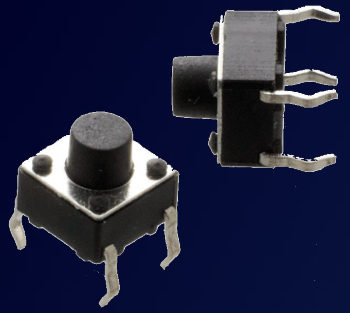
```
from gpiozero import OutputDevice

rele = OutputDevice(20)

# Encender relé
rele.on()

# Apagar relé
rele.off()

# El relé se apaga automáticamente
# cuando salimos del programa
```





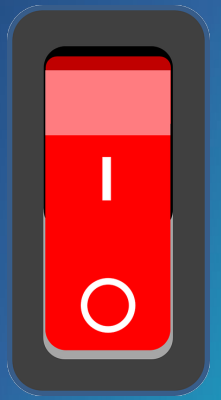
github.com/pythoneiro

1-LED/8-rele.py



Usando botones y pulsadores

- Al contrario que los LED, los botones/pulsadores utilizan pines configurados como Entrada.
- Los dos cables de un pulsador se conectan a:
  - Un pin GPIO
  - Según si queremos detectar “Pulsado” a nivel alto o bajo:
    - Activo a nivel Alto (1/True/verdadero): 3V3
    - Activo a nivel Bajo (0/False/falso): GND



Para simplificar las cosas, vamos a utilizar “activo a nivel bajo”:
Conectaremos un cable a GPIO y otro a GND.
Este es el modo que usa por defecto *gpiozero*

Detectando pulsaciones en Python

Este ejemplo muestra por consola un mensaje cuando se pulsa o se suelta el botón.

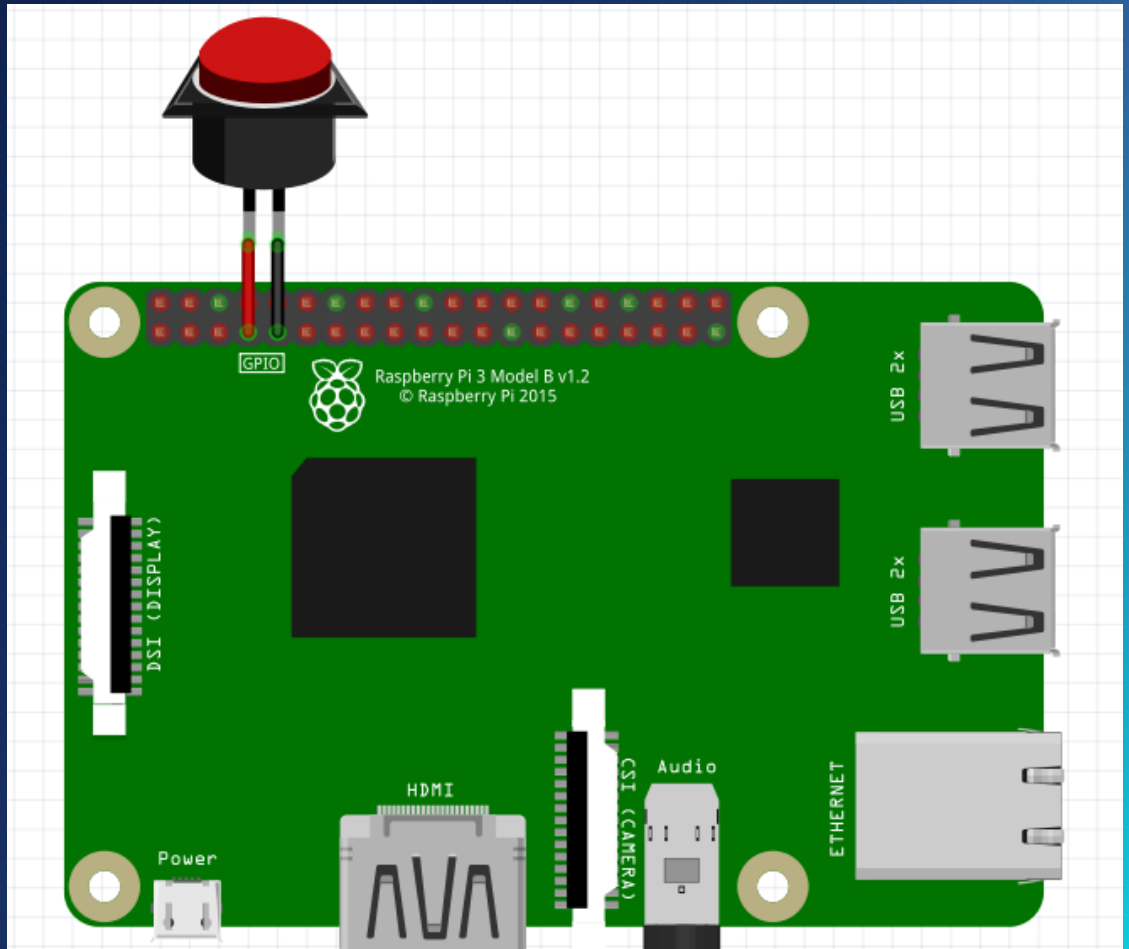
```python
from gpiozero import Button

button = Button(4)

# Función que se ejecuta cuando el botón se pulsa
def btn_pulsado():
    print("El botón se ha pulsado")

# Función que se ejecuta cuando el botón se suelta
def btn_soltado():
    print("El botón se ha soltado")

# Asignamos las funciones a las acciones del botón
button.when_pressed = btn_pulsado
button.when_released = btn_soltado
```

github.com/pythoneiro

2-Pulsadores/0_pulsador.py

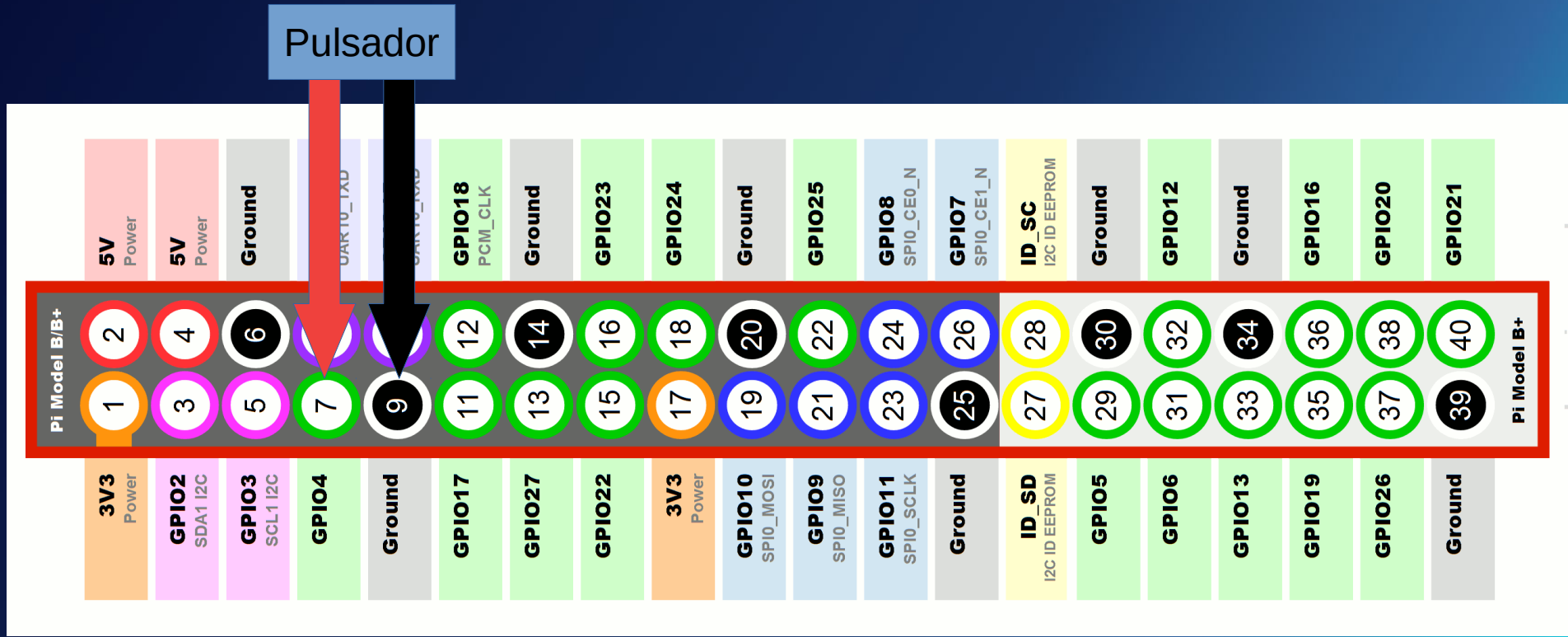
Botón pulsador
Esquema de conexión

- Componentes necesarios:
  - 1 botón pulsador
  - 2 cables, dependiendo del pulsador
- La propia Raspberry incorpora resistencias internas para limitar la corriente que pasa por el pulsador.
- Los pulsadores no tienen polaridad.



github.com/pythoneiro

Botón pulsador
Esquema de conexión



github.com/pythoneiro

Ejemplo “práctico”: Contador

Este ejemplo tiene un contador que se incrementa cada vez que pulsamos el botón. Usa los mismos componentes que el anterior ejemplo.


```python
from gpiozero import Button

button = Button(4)
contador = 0

# Función que se ejecuta cuando el botón se pulsa
def btn_pulsado():
    # contador como variable global para que se actualice
    # desde dentro de la función
    global contador
    # sumamos +1 al contador
    contador += 1
    # y lo mostramos
    print("Contador:", contador)

# Asignamos las funciones a las acciones del botón
button.when_pressed = btn_pulsado
```

2-Pulsadores/1_contador.py

github.com/pythoneiro

¿Al pulsar o al soltar?

- Los botones tienen los métodos:
  - **`when_pressed`**: cuando se **pulsa** el botón
  - **`when_released`**: cuando se **suelta** el botón
- Está de nuestra mano pensar qué método usar en cada situación.

Ejemplo: encender y apagar un LED con un botón

En este caso, vamos a encender o apagar un LED cada vez que pulsamos el botón, según el estado anterior (si estaba apagado se enciende, si estaba encendido se apaga).

gpiozero ya incorpora una función para cambiar el estado del LED (de apagado a encendido y viceversa)

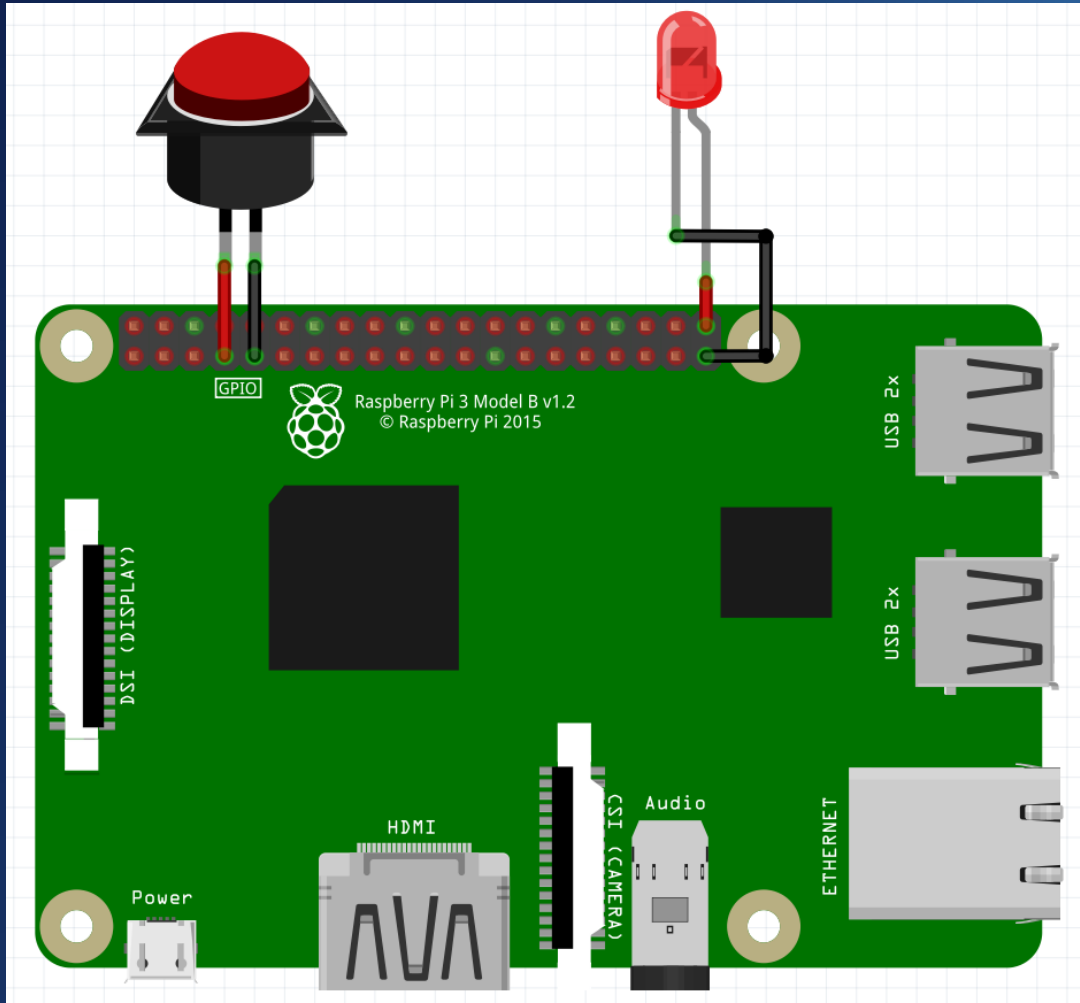
```
from gpiozero import LED, Button
from signal import pause

led = LED(21)
button = Button(4)

# Asignamos las funciones a las acciones del botón
# El método toggle sobre LED cambia el estado del led
button.when_pressed = led.toggle

# Mantenemos el programa pausado
# Las acciones del botón se procesan de fondo
# No necesario si lo ejecutamos desde intérprete
pause()
```

2-Pulsadores/2_pulsador_led.py

github.com/pythoneiro

Ejemplo: encender y apagar un LED con un botón
Esquema de conexión

- Para simplificar este ejemplo, vamos a usar el LED sin resistencia, aunque no es lo recomendable.
- Componentes necesarios:
  - 1 botón pulsador
  - 1 LED
  - 2 cables hembra-hembra para el LED
  - 2 cables o ninguno, dependiendo del pulsador



Ejemplo: encender y apagar un LED con un botón
Esquema de conexión

github.com/pythoneiro

Ejemplo: encender y apagar un LED con un botón

Si quisiéramos hacer manualmente la acción de encender/apagar el LED con el botón, se haría así:

```
from gpiozero import LED, Button
from signal import pause

led = LED(21)
button = Button(4)

def cambiar_led():
    if led.value == 1:
        # LED encendido -> apagarlo
        led.value = 0
    else:
        # LED apagado -> encenderlo
        led.value = 1

# Asignamos las funciones a las acciones del botón
# El método toggle sobre LED cambia el estado del led
button.when_released = cambiar_led

# Mantenemos el programa pausado
# Las acciones del botón se procesan de fondo
# No necesario si lo ejecutamos desde intérprete
pause()
```

github.com/pythoneiro

2-Pulsadores/3_pulsador_led_manual.py

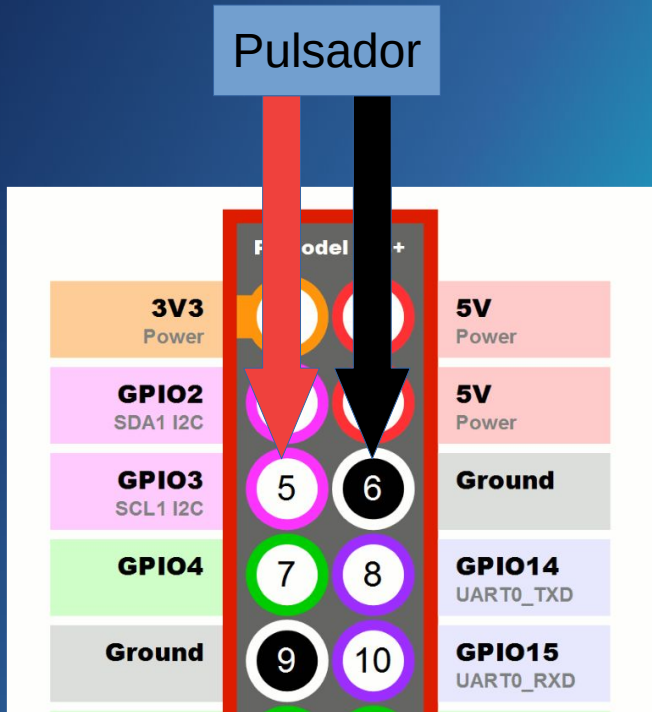
Esperar a que se pulse un botón

El método wait_for_press paraliza nuestro programa hasta que el botón no sea pulsado.

```
from gpiozero import Button

button = Button(4)

print("Esperando a que pulses el botón...")

# El programa queda parado en este punto:
button.wait_for_press()
# Cuando se pulsa el botón, se continúa...

print("Botón pulsado! Hasta luego!")
```

Ejemplo útil: Botón de Encendido/Apagado

- Las Raspberry carecen de un botón físico para poder encender y apagar, pero es posible añadir uno, teniendo en cuenta que:
  - Para apagar la Raspberry, podemos ejecutar el comando
    `sudo shutdown -h now`
  - Para encender la Raspberry, una vez está apagada, tenemos que unir los pines GPIO3 y GND (con un pulsador, por ejemplo)



Ejemplo útil: Botón de Encendido/Apagado

- Podemos programar un script con Python y *gpiozero* que apague la Raspberry al pulsar un botón conectado entre GPIO3 y GND.
- Este botón servirá luego para encender nuevamente la Raspberry.
- Para evitar apagados accidentales, tendremos que mantener pulsado el botón unos segundos para apagar la Raspberry.
- Añadiremos además un LED para saber el estado del script.

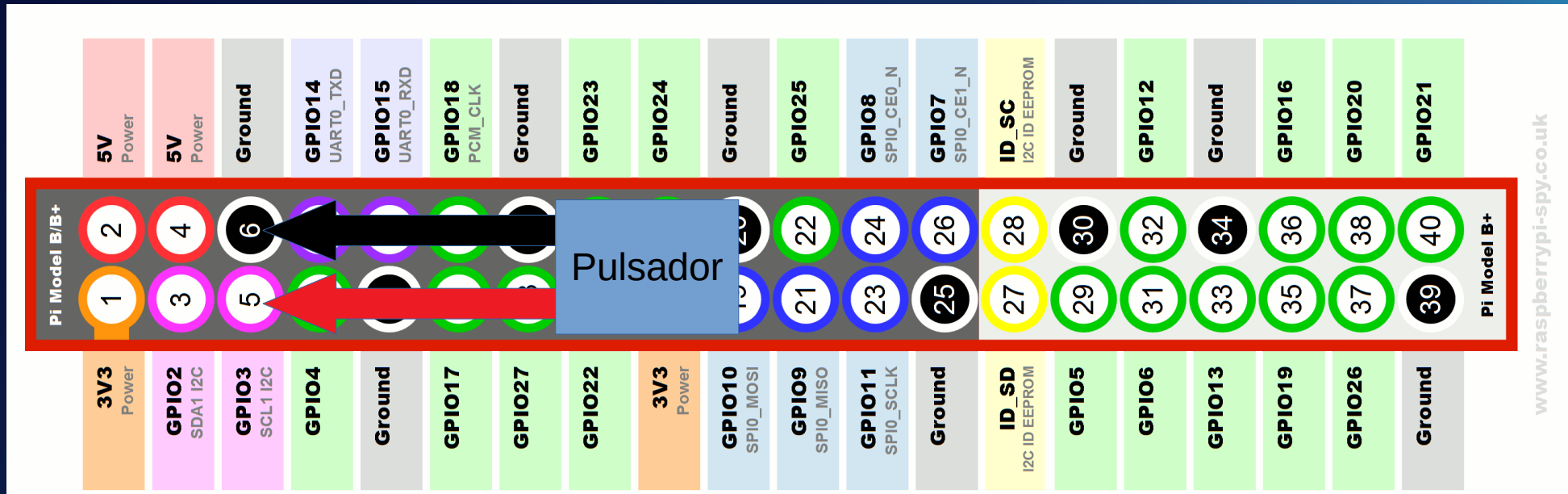
Ejemplo útil: Botón de Encendido/Apagado

```
# Ejecutar como sudo

from gpiozero import Button, LED
from subprocess import check_call
from signal import pause

# Boton de apagado; debe mantenerse pulsado 3 segundos
shutdown_btn = Button(3, hold_time=3)
# LED opcional para saber que todo va bien
led = LED(21)

# Funcion que se ejecuta cuando pulsamos el boton
def shutdown():
    # Parpadear el LED mas rapido mientras se esta apagando
    led.blink(on_time=0.1, off_time=0.1)
    check_call(['sudo', 'poweroff'])

# Parpadear el LED opcional para saber que
# todo esta funcionando bien
led.blink(on_time=0.5, off_time=0.5)
# Asignar la funcion shutdown al mantener el boton pulsado
shutdown_btn.when_held = shutdown

# Mantener el programa pausado
pause()
```

github.com/pythoneiro

3-Utiles/apagar.py

Ejemplo útil: Botón de Encendido/Apagado
Esquema de conexión



github.com/pythoneiro

Basic Recipes con *gpiozero*

En la documentación de la librería podemos encontrarnos con multitud de pequeños proyectos que podemos realizar, así como componentes adicionales que conectar.

Además, existen numerosas clases para realizar algunos de los ejemplos propuestos (por ejemplo, para montar un semáforo con LEDs, controlar RGB o hacer *pings* a equipos de nuestra red).

https://gpiozero.readthedocs.io/en/stable/recipes.html