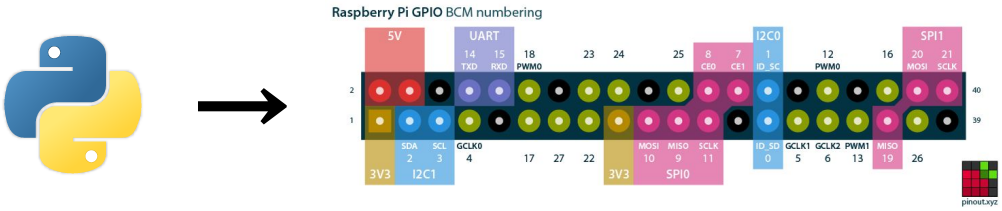
Name: _____ Date: _____

Python and the gpiozero Library



This guide has been modified from the original text -- https://gpiozero.readthedocs.io/en/stable/

gpiozero is a simple interface to everyday GPIO components used with Raspberry Pi.

Created by Ben Nuttall of the Raspberry Pi Foundation, Dave Jones, and other contributors.

Generic Examples

Blinks an LED on GPIO pin 17. 1 second on and 1 second off.

```
1  from gpiozero import LED
2  from time import sleep
3
4  led = LED(17)
5
6  while True:
7      led.on()
8      sleep(1)
9      led.off()
10     sleep(1)
```

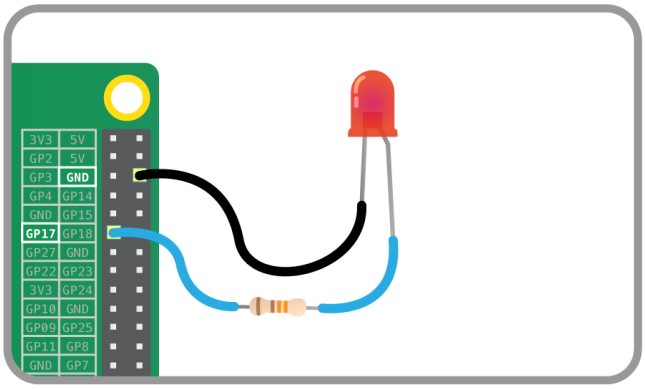
Turns on the LED on GPIO pin 17 when the button is pressed. Assigns the `.on()` method to the `.when_pressed` event and the `.off()` method to the `.when_released` event

```
1  from gpiozero import LED, Button
2  from signal import pause
3
4  led = LED(17)
5  button = Button(3)
6
7  button.when_pressed = led.on
8  button.when_released = led.off
9
10 pause()
```

Reference: https://gpiozero.readthedocs.io/en/stable/

Components / Objects

LED



Turn an LED on and off repeatedly:

```
1  from gpiozero import LED
2  from time import sleep
3
4  red = LED(17)
5
6  while True:
7      red.on()
8      sleep(1)
9      red.off()
10     sleep(1)
```

Alternatively:

```
1  from gpiozero import LED
2  from signal import pause
3
4  red = LED(17)
5
6  red.blink()
7
8  pause()
```

Note

Reaching the end of a Python script will terminate the process and GPIOs may be reset. Keep your script alive with signal.pause(). See Keep your script running for more information.

PWMLED -- LED with variable brightness

Any regular LED can have its brightness value set using PWM (pulse-width-modulation). Basically, the RPi will vary the duty cycle on the pin to affect the brightness level. In GPIO Zero, this can be achieved using PWMLED using **values** between 0 and 1:

```
from gpiozero import PWMLED
from time import sleep

led = PWMLED(17)

while True:
    led.value = 0  # off
    sleep(1)
    led.value = 0.5  # half brightness
    sleep(1)
    led.value = 1  # full brightness
    sleep(1)
```

Similarly to blinking on and off continuously, a PWMLED can **pulse** (fade in and out continuously):

```
from gpiozero import PWMLED
from signal import pause

led = PWMLED(17)

led.pulse()

pause()
```

LEDBoard

A collection of LEDs can be accessed using LEDBoard. This simplifies the control of multiple LEDs together as a single 'board' or matrix.

```
from gpiozero import LEDBoard
from time import sleep
from signal import pause

leds = LEDBoard(5, 6, 13, 19, 26)

leds.on()
sleep(1)
leds.off()
sleep(1)
leds.value = (1, 0, 1, 0, 1)
sleep(1)
leds.blink()

pause()
```

Using **LEDBoard** with `pwm=True` allows each LED's brightness to be controlled:

```
from gpiozero import LEDBoard

leds = LEDBoard(5, 6, 13, 19, 26, pwm=True)

leds.value = (0.2, 0.4, 0.6, 0.8, 1.0)
```

LEDBarGraph

A collection of LEDs can be treated like a bar graph using **LEDBarGraph**:

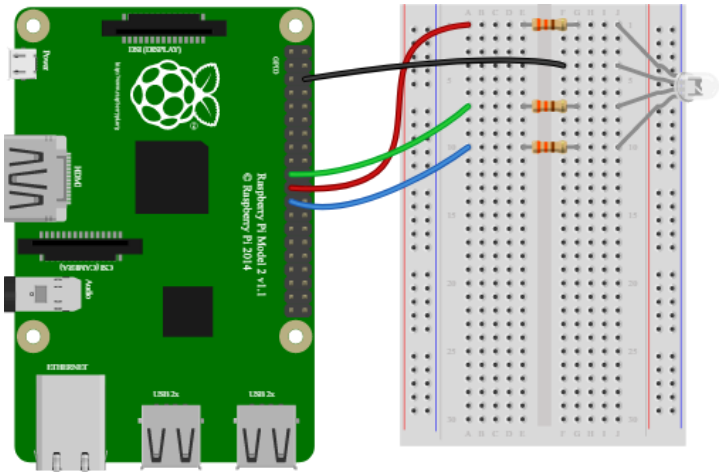
```
from gpiozero import LEDBarGraph
from time import sleep
graph = LEDBarGraph(5, 6, 13, 19, 26, pwm=True)

graph.value = 1/10  # (0.5, 0, 0, 0, 0)
sleep(1)
graph.value = 3/10  # (1, 0.5, 0, 0, 0)
sleep(1)
graph.value = -3/10  # (0, 0, 0, 0.5, 1)
sleep(1)
graph.value = 9/10  # (1, 1, 1, 1, 0.5)
sleep(1)
graph.value = 95/100  # (1, 1, 1, 1, 0.75)
sleep(1)
```

Note values are essentially rounded to account for the fact LEDs can only be on or off when `pwm=False` (the default).

However, using **LEDBarGraph** with `pwm=True` allows more precise values using LED brightness:

```
from gpiozero import LEDBarGraph
from time import sleep

graph = LEDBarGraph(5, 6, 13, 19, 26, pwm=True)

graph.value = 1/10  # (0.5, 0, 0, 0, 0)
sleep(1)
graph.value = 3/10  # (1, 0.5, 0, 0, 0)
sleep(1)
graph.value = -3/10  # (0, 0, 0, 0.5, 1)
sleep(1)
graph.value = 9/10  # (1, 1, 1, 1, 0.5)
sleep(1)
graph.value = 95/100  # (1, 1, 1, 1, 0.75)
sleep(1)
```

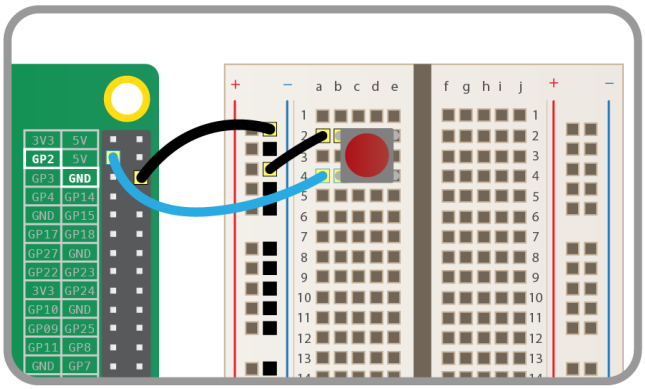
Full color LED (RGB)



Making colours with an RGBLED:

```
from gpiozero import RGBLED
from time import sleep

led = RGBLED(red=9, green=10, blue=11)

led.red = 1  # full red
sleep(1)
led.red = 0.5  # half red
sleep(1)

led.color = (0, 1, 0)  # full green
sleep(1)
led.color = (1, 0, 1)  # magenta
sleep(1)
led.color = (1, 1, 0)  # yellow
sleep(1)
led.color = (0, 1, 1)  # cyan
sleep(1)
led.color = (1, 1, 1)  # white
sleep(1)

led.color = (0, 0, 0)  # off
sleep(1)

# slowly increase intensity of blue
for n in range(100):
    led.blue = n/100
    sleep(0.1)
```

Button



Check if a **Button** is pressed:

```
from gpiozero import Button

button = Button(2)

while True:
    if button.is_pressed:
        print("Button is pressed")
    else:
        print("Button is not pressed")
```

Wait for a button to be pressed before continuing:

```
from gpiozero import Button

button = Button(2)

button.wait_for_press()
print("Button was pressed")
```

Run a function every time the button is pressed:

```
from gpiozero import Button
from signal import pause

def say_hello():
    print("Hello!")

button = Button(2)

button.when_pressed = say_hello

pause()
```

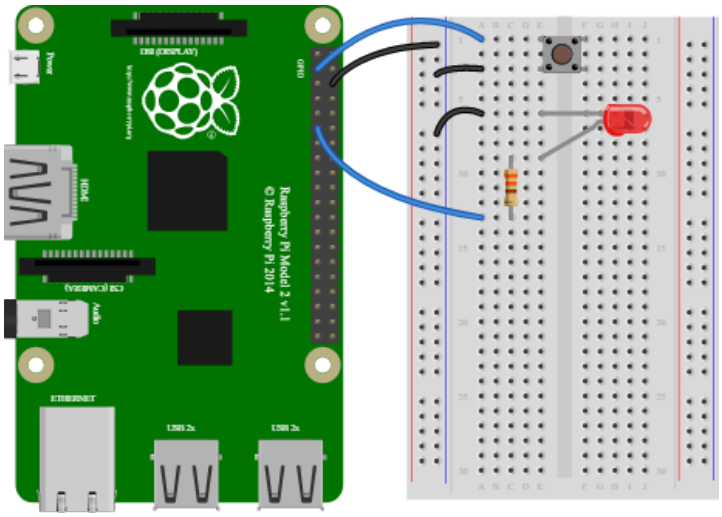
Note

Note that the line `button.when_pressed = say_hello` does not run the function `say_hello`, rather it creates a reference to the function to be called when the button is pressed. Accidental use of `button.when_pressed = say_hello()` would set the `when_pressed` action to `None` (the return value of this function) which would mean nothing happens when the button is pressed.

Similarly, functions can be attached to button releases.

```
1  from gpiozero import Button
2  from signal import pause
3
4  def say_hello():
5      print("Hello!")
6
7  def say_goodbye():
8      print("Goodbye!")
9
10 button = Button(2)
11
12 button.when_pressed = say_hello
13 button.when_released = say_goodbye
14
15 pause()
```

Button controlled LED



Turn on an `LED` when a `Button` is pressed:

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

led = LED(17)
button = Button(2)

button.when_pressed = led.on
button.when_released = led.off

pause()
```

Alternatively:

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

led = LED(17)
button = Button(2)

led.source = button.values

pause()
```