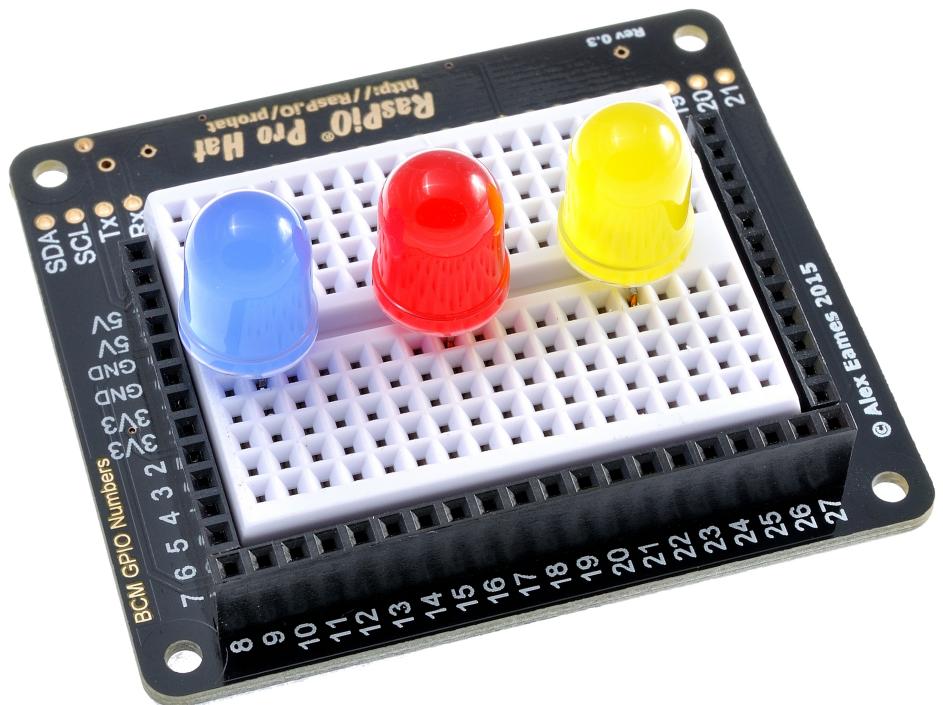


# GPIO Zero Programming With



# RasPiO® Pro Hat

by Alex Eames

# Introduction

## Protected Ports Perfectly Positioned

The RasPiO®<sup>1</sup> Pro Hat was developed out of the perceived need for a HAT which puts the *Raspberry Pi's GPIO ports in numerical order* and clearly labelled. You don't have to count pins or wonder which port you're connecting to. Each port has a female socket to plug your wires or components into. The ports are arranged, along with plenty of power and GND sockets, around a 72-point breadboard.

If you want to do some electronics, it's made a lot easier for you. LEDs *need no current-limiting resistors* because they are already built-in.

Pro Hat also has a *protection circuit on each GPIO port*, which means you won't damage your Pi's ports by wiring something up incorrectly. (But it is still possible to cause damage by directly shorting 3V3 or 5V to GND.)

Additionally, if you want to bypass the 330 Ohm resistor on a GPIO port, you can connect directly to the unprotected side where all the ports<sup>2</sup> are broken out as through-holes. This is particularly useful for buzzers, which usually require slightly over the 10mA limit imposed by the resistors.

Ben Nuttall and Dave Jones have created GPIO Zero as the ideal way into Python GPIO programming. Using it with the Pro HAT means there is *nothing to install* before you can start playing.

Also, by keeping the board inexpensive, I hope it's realistic for individuals, schools and jams to be able to get hold of some and discover the joys of controlling the world with the Raspberry Pi and GPIO Zero.

---

<sup>1</sup> RasPiO is a trademark of Alex Eames. Raspberry Pi is a trademark of the Raspberry Pi Foundation

<sup>2</sup> Apart from GPIO26, which is used for the HAT EEPROM

# RasPiO® Pro Hat Instructions

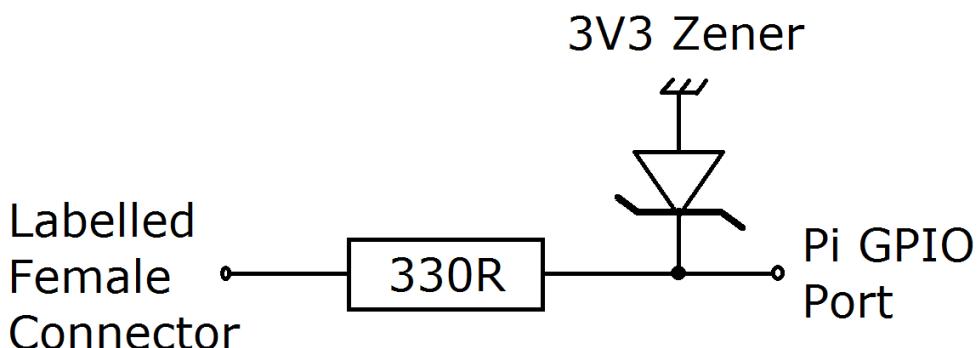
The RasPiO Pro Hat uses the BCM GPIO port numbering scheme. This is a perfect match for GPIO Zero.

## Hardware Technical Overview

This page is mainly for the technically minded. If you just want to get on with experimenting, you can skip to the next section.

### Port Protection

The port protection is via a 3V3 Zener diode and 330 Ohm resistor on each port. The Zener diode clips over-voltage down to a safe 3V3. The 330 Ohm resistor limits the current into or out of a port to 10 mA. This is enough to prevent port damage in most situations.



*Schematic of port protection circuit*

### Hardware Pull-ups

GPIOs 2, 3 and 26 all have hardware pull-ups. GPIOs 2 & 3 (the i<sup>2</sup>c ports) have 2k pull-up resistors on them. GPIO 26 on the Pro Hat is also connected to the EEPROM write-protect pin, which has a 1k pull-up. This means that the default state for these pins is HIGH unless brought LOW in software. So if you connect an LED to any of these ports it will

be (dimly) lit by default.

## SPI Limitations

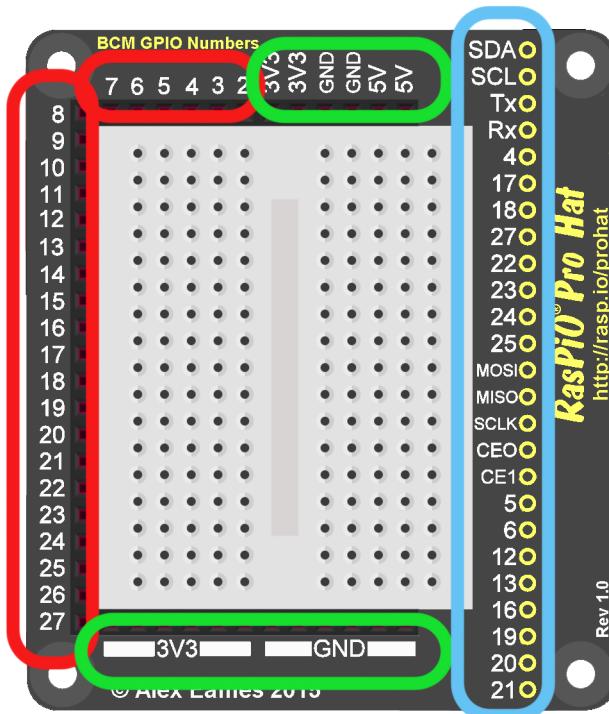
I've found that SPI devices (e.g. MCP3008) work fine, even through the protected ports. In Beta-testing, it was discovered that some high-speed SPI devices, like the PiTFT and other small SPI LCD colour screens do not play nicely with the protection circuitry. This is not seen as much of a problem as it falls outside the expected use of the Pro Hat. But that's what Beta testing is for. Thanks [Ton van Overbeek](#) for discovering that one.

If SPI is enabled on the Pi, the default state is HIGH for GPIOs 7 & 8 (CE0 & CE1). LEDs connected to these ports will be lit unless brought LOW in software.

## Know Your RasPiO Pro Hat

The RasPiO Pro Hat has been designed to fit any 40-pin consumer model of Raspberry Pi and make it as easy as possible for people to get into GPIO Zero programming on the Pi.

BCM  
GPIO  
Numbers



Alternative  
Unprotected  
GPIO access

Power & Ground

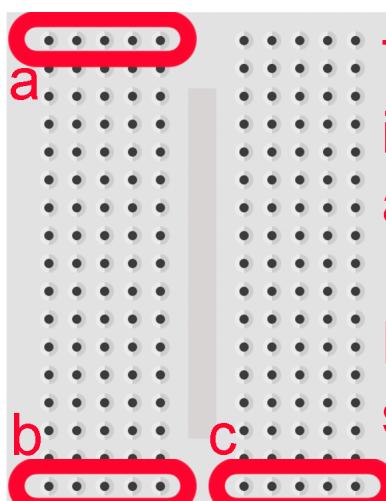
*Anatomy of RasPiO Pro Hat*

The RasPiO Pro Hat has female header sockets to plug in wires and components. In the wiring diagrams we'll remove these to make things clearer.

Breadboards give an easy way to make connections.

The five points in each row are connected to each other.

But each of the rows (a, b, c) are completely separate from each other.



The 5 holes in each group are connected

Each group is separate

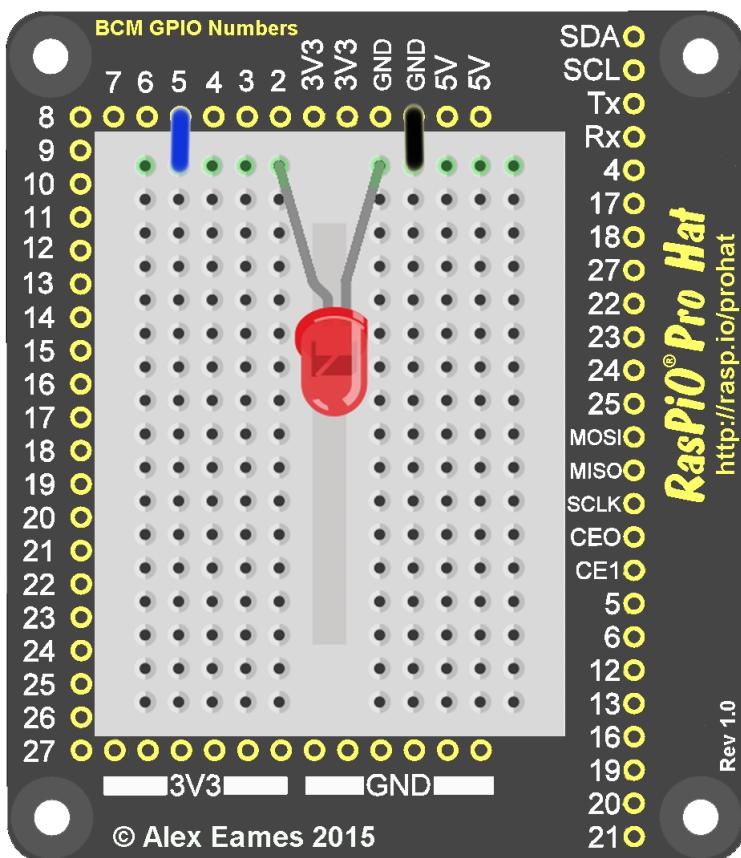
# Controlling An LED

The very simplest GPIO experiment is to control an LED. So that's where we'll start.

A light emitting diode (LED) is a small electronic component that gives out light when electricity is passed through it. Usually when you wire up an LED, you put a resistor in series with it to limit the current. This prevents the LED from burning out. But with the RasPiO Pro Hat you don't need one because all the ports from 2-27 have a 330 Ohm resistor already on the underside of the board.

## LED Circuit

So let's take an LED and a wire or two and make up the following circuit...



*Circuit for led control*

The longer leg (+) on the LED connects to GPIO5. The shorter leg (-) connects to GND (ground).

What we now need to do is control GPIO5 to make the LED light up.

When we switch on GPIO5, the LED lights. When we switch off GPIO5 it goes off. First let's switch it on. Probably the best way to experience this for the first time is in a live Python environment.

## LED In Python Live Environment



Click the terminal icon or choose Menu > Accessories > Terminal  
This should open up a fresh new terminal window...

A screenshot of a terminal window on a Raspberry Pi. The window has a title bar with "File Edit Tabs Help" and a command prompt "pi@raspberrypi:~ \$". The main area of the window is entirely black, indicating that no output has been displayed yet.

*Terminal Window*

Type **python3** and press <ENTER>

You are now in a live Python3 environment.

A screenshot of a terminal window showing a live Python 3 environment. The window title bar includes "File Edit Tabs Help". The command prompt is "pi@raspberrypi:~ \$ python3". The terminal displays the following Python session:

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> from gpiozero import LED
>>> red = LED(5)
>>> red.on()
>>> red.off()
```

*Python 3 live environment*

If you need to get out of it at any time, hit <CTRL> + D

Now let's get coding. Type the following three lines of code, hitting <ENTER> at the end of each line...

```
from gpiozero import LED  
red = LED(5)  
red.on()
```

...as soon as you hit <ENTER> after red.on() the LED should light up.

Now if you type...

```
red.off()
```

...it will switch off. Congratulations. You are now ready to begin taking over the world. Now try...

```
red.blink()
```

...and the LED should blink on and off every second. You can stop it with...

```
red.off()
```

There's another useful function we can use in our programs to change the state of the LED. If it's ON, it gets turned OFF and vice versa. This is called toggle(). Try it now a few times...

```
red.toggle()
```

...each time you type this, the LED's state should toggle between ON and OFF.

There are a couple of other useful tricks you can use with LEDs in GPIOZero...

```
red.pin
```

...should return the GPIO number that red is attached to. In our case it's 5. Also...

```
red.is_lit
```

...is a way your program can tell if your red LED is lit or not. If lit, it returns True if not, False. Try this...

```
red.on()  
red.is_lit  
red.toggle()  
red.is_lit
```

You should see something like this...

```
>>> red.on()  
>>> red.is_lit  
True  
>>> red.toggle()  
>>> red.is_lit  
False  
>>> █
```

You've now had a good overview of each of the parts of gpiozero.LED

Now exit the Python live session by hitting <CTRL> + D

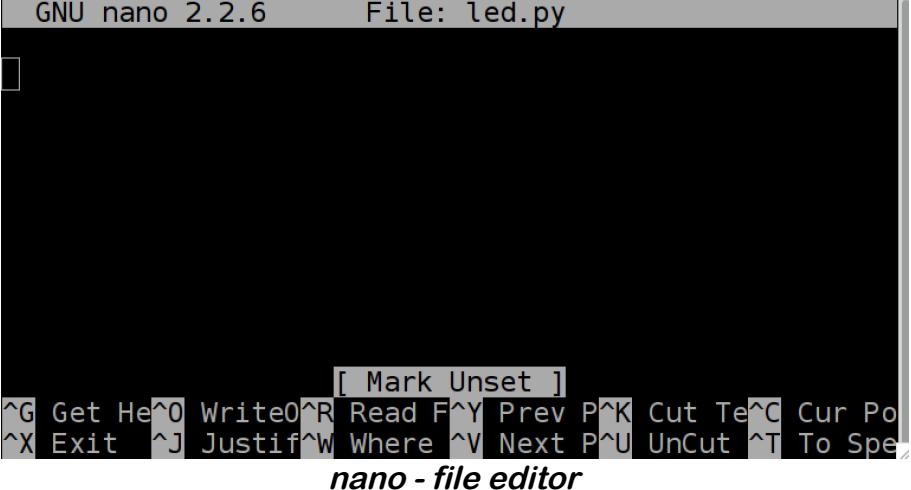
## Incorporating LED In A Program

What we just did is fine in a live Python session, and great for trying things out in real time. But if you want to store your program to use again, we'll need to learn a couple more tricks.

Now we're going to write a little program and save it. Type...

```
nano led.py
```

...and this screen should appear...

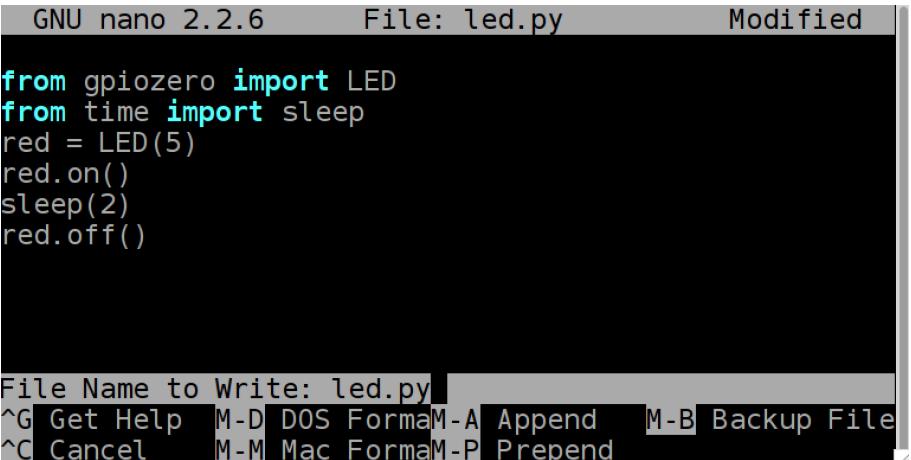


The screenshot shows a terminal window titled "GNU nano 2.2.6" with the file name "File: led.py". The editor is currently empty, displaying a single blank line. At the bottom of the screen, there is a menu bar with various keyboard shortcuts for file operations like Get Help, Write, Read, Prev, Cut, Copy, Paste, Exit, Justify, Where, Next, UnCut, To Spec, and Mark Unset. Below the menu bar, the text "nano - file editor" is displayed.

Enter the following code...

```
from gpiozero import LED
from time import sleep
red = LED(5)
red.on()
sleep(2)
red.off()
```

...then press <CTRL> + 0 (letter o, not zero) to save it. Then <ENTER> to confirm the filename led.py



The screenshot shows the same terminal window as before, but now it contains the Python code for a blinking LED. The code is identical to the one above. At the bottom of the screen, a message box appears asking "File Name to Write: led.py". Below this message box, there is a standard nano file menu with options like Get Help, M-D, DOS Form, M-A, Append, M-B, Backup File, Cancel, M-M, Mac Form, M-P, Prepend, and M-C.

Then press <CTRL> + X to exit nano. Now let's try running the program. Type...

```
python3 led.py
```

What should happen is the LED should switch on for 2 seconds, then it switches off and the program exits. What we learnt from this was how to use `sleep()` to control the timing of the switching.

In line 2 of our code we imported the `sleep()` function from the `time` module.

In line 5 we used `sleep(2)` to make the program wait for 2 seconds. You can change this to any figure you like. It can be an integer (whole number) or a decimal like 3.14 or 0.005, but either way it represents an amount of time in seconds.

During the `sleep(2)` the program literally does nothing but wait until the set time period is over.

The suggested next step is to change the `sleep()` time to a number of your choice. You could even add some more `red.on()`, `red.off()` and `sleep()` lines to make a sequence. To do this you need to edit, save and run the file as we did before.

If you ever need to stop a python program when it's running, press `<CTRL> + C`

## Make It Run Forever

If we change our code to this, our LED will flash on and off every 2s...

```
from gpiozero import LED
from time import sleep
red = LED(5)
while True:
    red.on()
    sleep(2)
    red.off()
    sleep(2)
```

This will run forever. To break out of this program press <CTRL> + C

The new thing here is the `while True:` loop. This means “repeat all the indented lines immediately below this one forever”.

The four lines after `while True:` are all indented with 4 spaces. This is important in Python. So be sure to get that exactly right or you'll see an error message. Python uses the indent to determine which code belongs in which block.

The program will flash the LED on and off every 2 seconds until...

- you hit <CTRL> + C
- an error occurs and the program crashes out
- there's a power outage

## Another Way To Pause And Blink

You can do the same thing as the above program another way, with less code if you use the `pause()` function from the `signal` module

```
from gpiozero import LED
from signal import pause
red = LED(5)
red.blink(on_time=2, off_time=2)
pause()
```

If you use `blink()` the default `on_time` and `off_time` are both set to 1s. If you want to change them, you need to specify them as in line 4. The value can be a decimal if you want. You can also omit the keyword arguments and just use the numbers, in which case `on_time` is first, `off_time` second, giving `red.blink(2, 2)`

Both work in exactly the same way.

## PWMLED

So far we've covered simple switching of an LED on and off. But it is possible to vary the brightness of an LED by switching it on and off very fast and repeatedly. This is called pulse-width modulation (PWM). Here's how to use it with an LED.

### PWMLED Live Session

Let's go into a live Python session to try it out. Type `python3`

```
from gpiozero import PWMLED  
red = PWMLED(5)  
red.value = 0.1
```

```
pi@raspberrypi:~/prohat $ python3  
Python 3.4.2 (default, Oct 19 2014,  
[GCC 4.9.1] on linux  
Type "help", "copyright", "credits"  
>>> from gpiozero import PWMLED  
>>> red = PWMLED(5)  
>>> red.value = 0.1  
>>> █
```

The red LED should now be dimly lit. It is switched ON and OFF 100 times per second, but is only ON for 10% of the time. Because the switching is faster than your eye can detect, it looks like it's on all the time, but not very brightly.

`red.value` can be 0.0 to 1.0. Anything outside that range will throw an error.

To save retyping in a live session, you can use the up arrow ↑ on your keyboard to bring up the last command then tweak it. Now experiment with different values for `red.value` and see how low and how high you can set it and still see a difference in LED brightness. It will vary, depending on the LED, your eyesight and lighting conditions.

0.005 to 0.7 makes a visible difference to my eyes with the LED I'm using. Anything between 0.7-1.0 looks equally bright to me and anything 0.005 and below looks equally dim.

## PWMLED In A Program

Now let's write a little program to cycle through brightening and dimming the LED in turn. <CTRL> + D to exit python3, then nano ledpwm.py to open nano. You can use other text editors/IDEs too.

```
from gpiozero import PWMLED
from time import sleep
red = PWMLED(5)
while True:
    for x in range(101):
        red.value = x * 0.01
        sleep(0.02)
    for x in range(100,-1,-1):
        red.value = x * 0.01
        sleep(0.02)
```

The first four lines should be familiar to you. We're just importing the required functions, setting up our red LED on GPIO5 for PWM and then starting a loop that goes on forever.

Let's have a closer look at the next bit...

```
for x in range(101):
    red.value = x * 0.01
    sleep(0.02)
```

for x in range(101): starts a loop counting up from x=0 to x=100 and stopping when it reaches 101.

red.value = x \* 0.01 sets our LED's PWM value to a value between 0.0 and 1.0 and then sleep(0.02) pauses for 0.02s. This means our LED will change brightness setting 50 times per second, which should give us a fairly smooth visual effect.

## The second block...

```
for x in range(100,-1,-1):
    red.value = x * 0.01
    sleep(0.02)
```

...does pretty much the same, but in reverse. It counts down from 100 to 0 in steps of -1.

for x in range(100,-1,-1):

literally means count down values of x starting at 100 and stop when you get to -1 in steps of -1. So the last value used will be 0.

## Alternative Way To PWMLED

As before, there is another way to achieve exactly the same thing with GPIOZero's built-in functions...

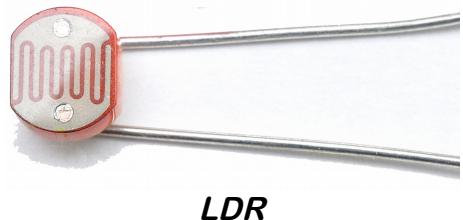
```
from gpiozero import PWMLED
from signal import pause
red = PWMLED(5)
red.blink(on_time=1, off_time=0, fade_in_time=1,
          fade_out_time=1)
pause()
```

On/off times and fade times are all in seconds.

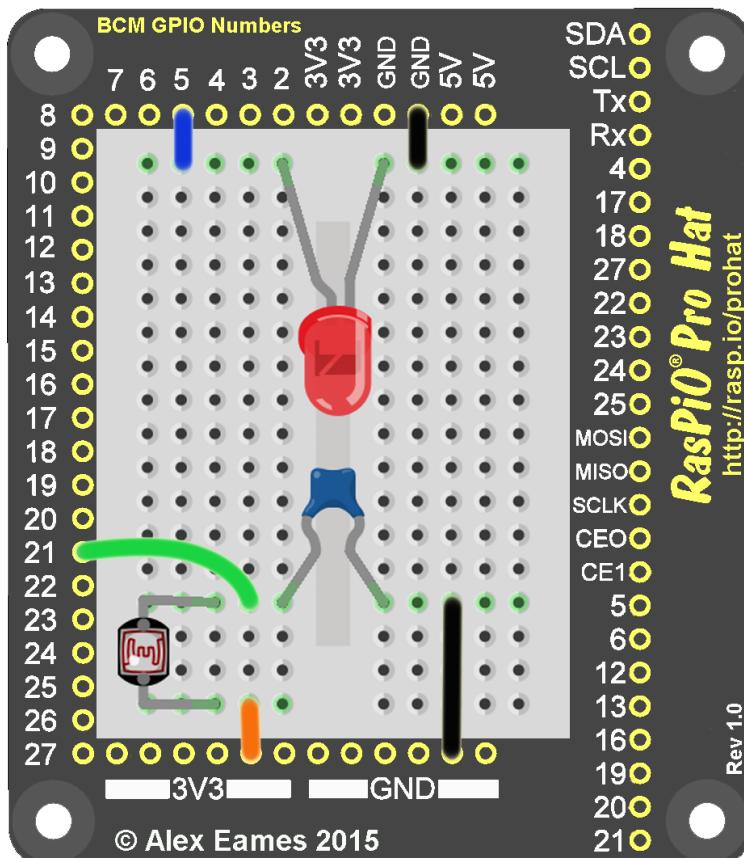
# LDR Light Sensor

A light dependent resistor (LDR) is a really useful component for detecting light and dark. I use a pen lid over the LDR to simulate darkness, when needed.

LDRs are often used in household security lights to stop them switching on when it's light. We're going to use GPIOZero to read an LDR and report the results on the screen. Later on, we'll use it to control our LED. First we need to add a couple of components and wires to our circuit...



LDR



Adding LDR & 100nF capacitor to our circuit

One end of the LDR connects to 3V3 and the other to GPIO21. One end of the 100nF capacitor connects to GPIO21 and the other to GND.

## LDR Code

There's a dedicated function called `LightSensor()` which we're going to use. Usage is similar to the `LED()` function. We set up our LDR on GPIO21. The simple code below will read `ldr` ten times per second and display the status on the screen...

```
from gpiozero import LightSensor  
from time import sleep  
ldr = LightSensor(21)  
while True:  
    if ldr.light_detected:  
        print("Light")  
    else:  
        print("Dark")  
    sleep(0.1)
```

```
Dark  
Dark  
Dark  
Dark  
Dark  
Dark  
Dark  
Light  
Light  
Light  
Light  
Light  
Light  
Light
```

*Screen output of ldr code*

You might want to use the output of `ldr.light_detected` to make a decision in your program instead.

## Using LDR For Switching

Since we still have the LED hooked up to GPIO5, let's tweak the above code to control the LED as well. Essentially, we're combining the LED and LDR code into one program...

```
from gpiozero import LightSensor, LED      # added LED  
from time import sleep  
red = LED(5)  
ldr = LightSensor(21)  
while True:  
    if ldr.light_detected:  
        print("Light")  
        red.off()                  # switch OFF LED if light
```

```
else:  
    print("Dark")  
    red.on()                      # switch ON LED if dark  
    sleep(0.1)
```

**When you run this code, it will switch OFF the LED if it detects light and ON if it doesn't detect light.**

## Variable Brightness LED/LDR

**To be a bit more ambitious, we can borrow this example from the GPIOZero documentation. This code varies the LED brightness using PWMLED() depending on how much light there is.**

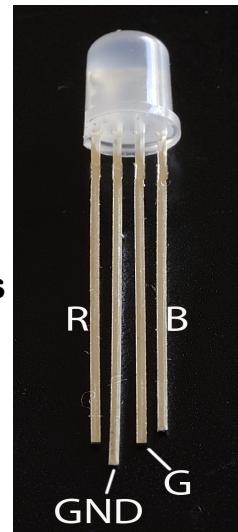
```
from gpiozero import LightSensor, PWMLED  
from signal import pause  
sensor = LightSensor(21)  
led = PWMLED(5)  
led.source = sensor.values  
pause()
```

## RGB LED

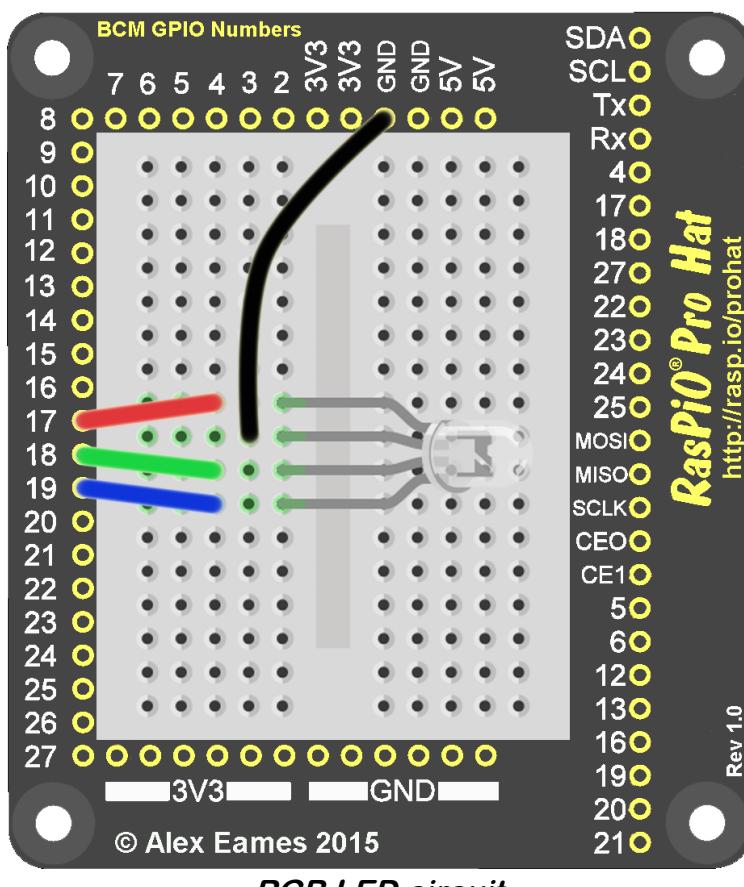
An RGB LED is actually three LEDs in one package; Red; Green; Blue. You can switch each colour independently. You can even PWM them independently to make pretty much any colour.

An RGB LED has four pins. The longest one (in our case) is GND. We're using a common cathode LED.

Red +ve is the pin on its own next to to the long GND pin.  
Blue +ve is the other outer pin and Green +ve is between GND and Blue +ve. Let's wire it up...



## RGBLED Circuit



## RGBLED Live Session

Let's explore this with a live session, as usual, python3  
then type the following two lines of code to set things up...

```
from gpiozero import RGBLED  
led = RGBLED(red=17, green=18, blue=19)
```

In line 2 we're setting up the led object and setting which GPIOs are red, green and blue. With that done, it's now time to experiment a bit...

```
led.red = 1  
led.red = 0.5  
led.red = 0  
led.green = 1  
led.green = 0.7  
led.green = 0  
led.blue = 1  
led.blue = 0.8  
led.blue = 0.2  
led.blue = 0
```

That is how we can control a single colour, setting led.red = anything between 0 and 1.0. We can also set all three colours in one line using led.color = () with 0 to 1.0 values for red, green and blue. Let's try it. (No need to type the comment after the # or the # itself)...

```
led.color = (1, 1, 1) # white  
led.color = (1, 1, 0) # yellow  
led.color = (1, 0, 1) # magenta  
led.color = (1, 0, 0) # red  
led.color = (0, 1, 1) # cyan  
led.color = (0, 1, 0) # green  
led.color = (0, 0, 1) # blue  
led.color = (0, 0, 0) # off (black)
```

This is what my live python3 session looked like...

```
>>> from gpiozero import RGBLED
>>> led = RGBLED(red=17, green=18, blue=19)
>>> led.red = 1
>>> led.red = 0.5
>>> led.red = 0
>>> led.green = 1
>>> led.green = 0.7
>>> led.green = 0
>>> led.blue = 1
>>> led.blue = 0.8
>>> led.blue = 0.2
>>> led.blue = 0
>>> led.color = (1, 1, 1)
>>> led.color = (1, 1, 0)
>>> led.color = (1, 0, 1)
>>> led.color = (1, 0, 0)
>>> led.color = (0, 1, 1)
>>> led.color = (0, 1, 0)
>>> led.color = (0, 0, 1)
>>> led.color = (0, 0, 0)
```

*RGB LED live session*

But don't forget you can use anything between 0 and 1.0 for these values. e.g. `led.color = (0.7, 0.2, 0.4)` is perfectly valid.

Now let's incorporate this into a program...

## RGB LED in a Program

The following program will fade each of the colours, red, green and blue in and out over a 4 second period. It will keep going until you close it with <CTRL> + C You can adjust the time delay in line 4 to speed up or slow down the fading.

```
from gpiozero import RGBLED
from time import sleep
led = RGBLED(red=17, green=18, blue=19)
delay = 0.02

while True:
    for x in range(100):
        led.red = x/100
        sleep(delay)
    for x in range(100,-1,-1):
        led.red = x/100
        sleep(delay)
    for x in range(100):
        led.green = x/100
        sleep(delay)
    for x in range(100,-1,-1):
        led.green = x/100
        sleep(delay)
    for x in range(100):
        led.blue = x/100
        sleep(delay)
    for x in range(100,-1,-1):
        led.blue = x/100
        sleep(delay)
```

The `for x in range(100):` loop cycles through 100 values and the `led.red = x/100` line determines the PWM value (0-1.0) of the led.

By now, you won't be overly surprised to hear that you can also do this with the built-in `blink()` function of `GPIOZero`...

## Fade RGB LED using built-in `blink()`

```
from gpiozero import RGBLED
from time import sleep
led = RGBLED(red=17, green=18, blue=19)

while True:
    led.blink(on_time=0, off_time=0, fade_in_time=2,
              fade_out_time=2, on_color=(1, 0, 0),
              off_color=(0, 0, 0))
    sleep(4)
    led.blink(on_time=0, off_time=0, fade_in_time=2,
              fade_out_time=2, on_color=(0, 1, 0),
              off_color=(0, 0, 0))
    sleep(4)
    led.blink(on_time=0, off_time=0, fade_in_time=2,
              fade_out_time=2, on_color=(0, 0, 1),
              off_color=(0, 0, 0))
    sleep(4)
```

## Final Word

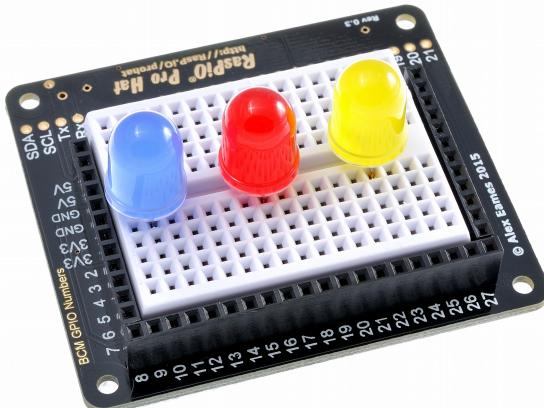
You should now have a fairly good overview of the basics of GPIOZero and using the RasPiO Pro Hat.

I hope you've had a lot of fun with it. There is always more to learn and further to go. As time goes by, I intend to add more to this guide to cover more aspects of GPIOZero and more components.

You can check for the latest version at <http://rasp.io/prohat>

And if you haven't yet got yourself a RasPiO Pro Hat, or need another, you can get that from here as well...

<http://rasp.io/prohat>



**RasPiO® Pro Hat**