

# Traffic Light Project Kit Project Guide

---

## Initial Setup

If you've purchased this kit, you're hopefully already fairly familiar with how a Raspberry Pi works, and how to set one up. However, if this is not the case, you will possibly need a little guidance on how to set up the software on your Raspberry Pi before you get started with the traffic light projects.

So, if you need a little assistance in getting started with the Raspberry Pi, the quick start guide at the Raspberry Pi foundation's web site is an excellent starting point.

<https://www.raspberrypi.org/help/quick-start-guide/>

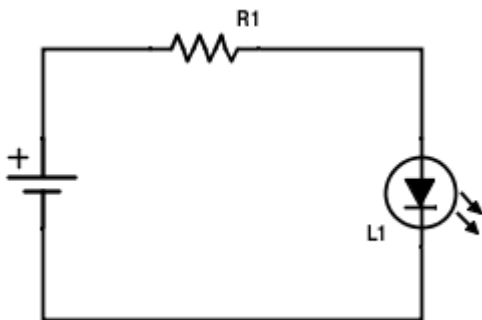
Following these instructions will allow you to set up your Raspberry Pi board with the other bits of hardware that you will require; and will also allow you to set up the software required to get your Raspberry Pi up and running.

## Project One – Light an LED

### Introduction

If you're reading this, we will assume that you've got your Raspberry Pi all set up. Everything is up and running and you've upgraded to the latest version of Raspbian etc. Now you've got that interesting looking bag of components burning a hole in your pocket, right? Let's use some of them and make the Pi control something in the real world. This first project is simple; it gets you used to using the breadboard and understanding how an LED works in a circuit.

This is the circuit we are going to build; it is a simple circuit designed to illuminate one of the LEDs in your component pack.



<http://www.instructables.com/id/Choosing-The-Resistor-To-Use-With-LEDs/?ALLSTEPS>

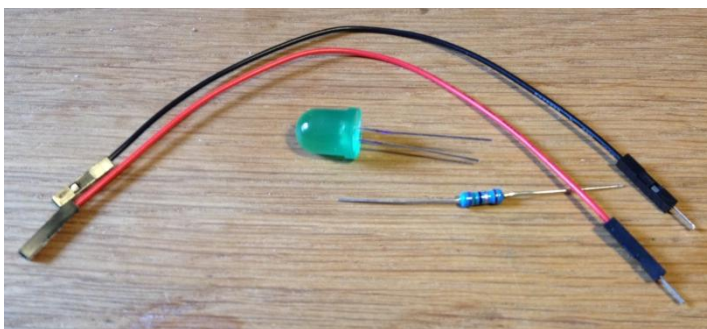
You will need:

- 1 LED (choose your favourite colour)
- 1 75R Resistor (these are the ones with the first three bands being Purple (7) Green (5) and Black (x1).  $75 \times 1 = 75\text{ohms}$ , for a full explanation of resistor colour codes see the links below)

<http://www.allaboutcircuits.com/textbook/reference/chpt-2/resistor-color-codes/>

[https://physics.ucsd.edu/neurophysics/courses/physics\\_120/resistorcharts.pdf](https://physics.ucsd.edu/neurophysics/courses/physics_120/resistorcharts.pdf)

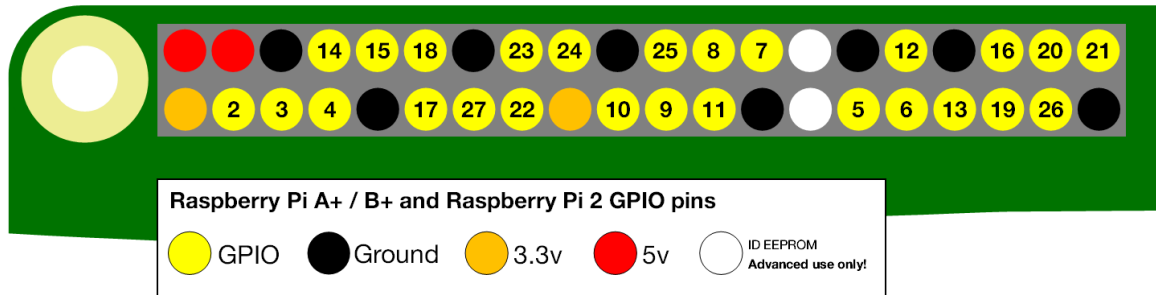
- 2 Jumper wires (black and red)



We are going to build this circuit on our breadboard. Breadboards are useful electronics prototyping tools and are wired like this -

<https://learn.sparkfun.com/tutorials/how-to-use-a-breadboard>

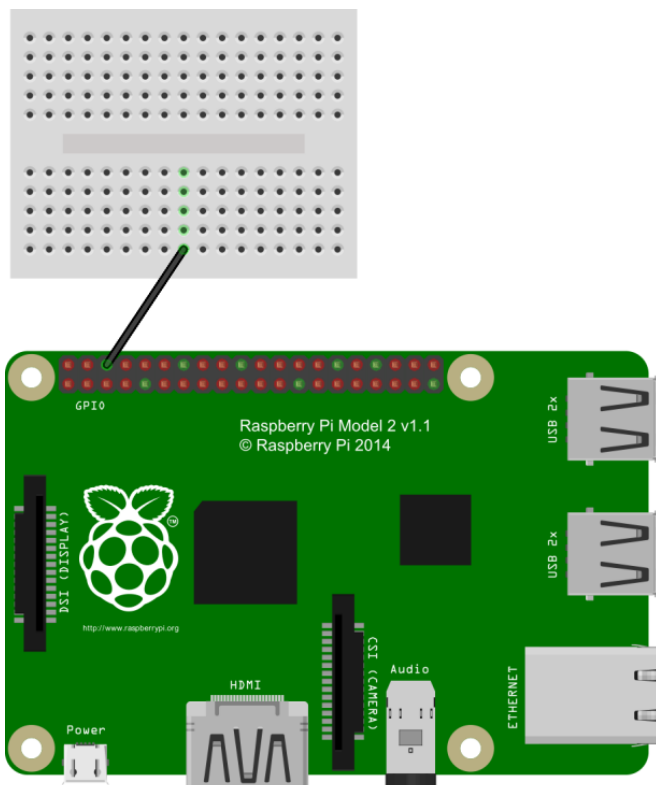
We are going to connect our circuit to the Pi using the 40-pin GPIO connector. The pinout for the GPIO connector looks like this:



If you want more information on the Raspberry Pi GPIO pins then this resource by our buddy @gadgetoid is fantastic [www.pinout.xyz](http://www.pinout.xyz)

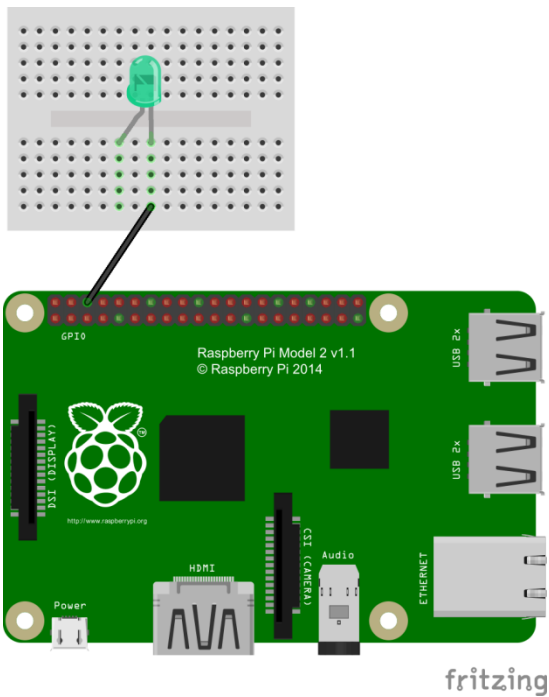
So let's get building our first circuit:

1. The first part of the circuit we will build is the link to the -ve side of our power source, in this instance we are using the Ground signal on the Pi GPIO. Take a black jumper wire and insert the pin into the breadboard next to the edge. Plug in the socket end to one of the Ground connectors on the GPIO. I used pin 6 as shown below:

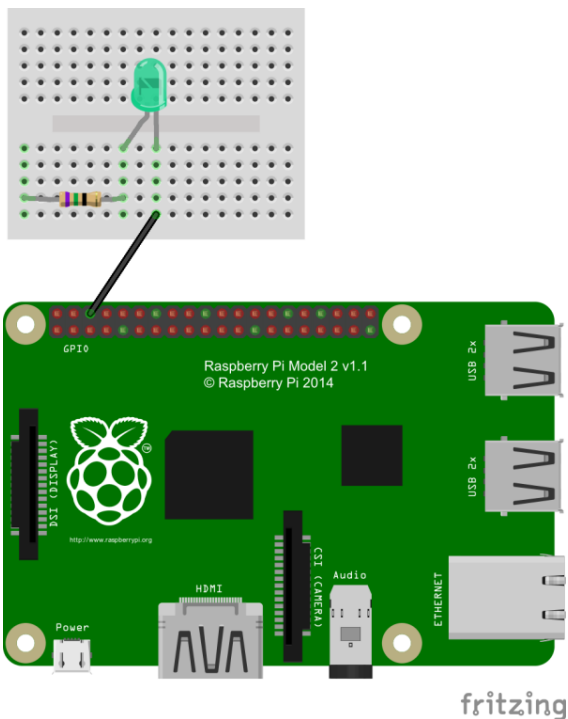


fritzing

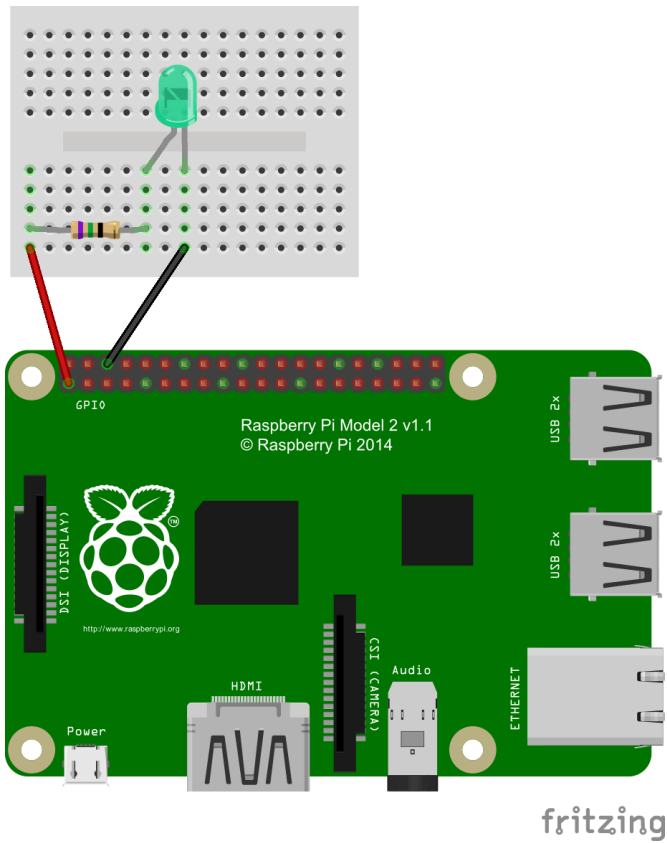
- Now let's add the LED into the circuit. Insert the shorter leg of the LED, remember this is the -ve side called the cathode, into the same row as the black jumper wire. Insert the other, longer, leg into a different row.



- Next we need to add the resistor into the circuit. Take the 75R resistor and insert one leg into the breadboard on the same row as the +ve anode of the LED (remember the anode of the LED is the one with the longer leg). Insert the other leg into a different row of the breadboard as below. The good news here is that, unlike LEDs, resistors are not polarised so you cannot plug it in the wrong way around!



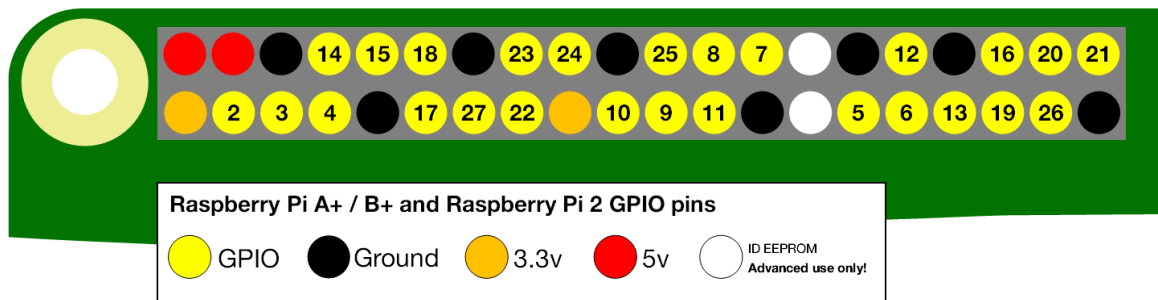
- Now to complete our first circuit. We need to connect the resistor to the +ve side of our power source. Take a red jumper lead and insert the pin into the same row as the second leg of the resistor. Plug the socket end to one of the +3.3V connectors on the Pi GPIO. I've plugged it into pin 1 below. And let there be light!



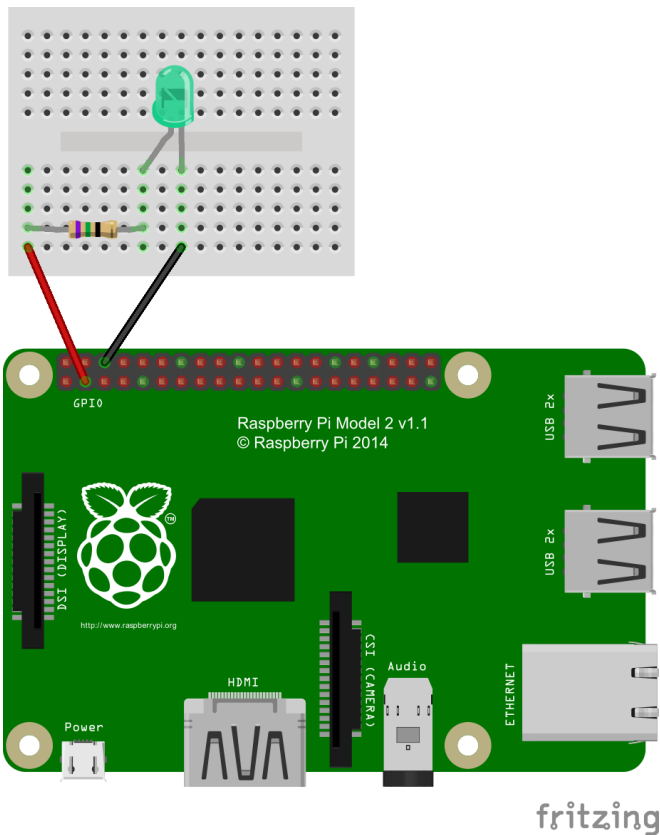
# Project Two – Controlling the LED with the Pi

## Introduction

This project builds directly on from what we learnt in Project One. We are going to use the same LED circuit but this time we will control the LED from the Raspberry Pi. That means that it is time to do some coding!



So that we can control the LED from the Pi we need to make one small change to our circuit. The Ground, 3.3V and 5V pins of the Pi are always at those voltages. The GPIO pins, however, we can control by writing a Python programme to run on the Pi. So let's move the red jumper wire from 3.3V on Pin 1 and connect it to GPIO2 on Pin 3.



In order to make it easier to control the GPIO pins and connect them to real world electronic components we are going to use a library of programming commands called GPIO Zero.

<https://pythonhosted.org/gpiozero/>

To install GPIO Zero type the following commands at the command prompt

```
sudo apt-get install python-pip python3-pip
sudo pip install gpiozero
sudo pip-3.2 install gpiozero
```

```
sudo pip install gpiozero --upgrade
sudo pip-3.2 install gpiozero --upgrade
```

Once installed type:

```
sudo idle3
```

This will open the Python programming shell. Use File->New File to open a new editor window and then type in the following code:

```
from gpiozero import LED

green = LED(2)
green.blink()
```

The first line imports the LED class into your code from the GPIO Zero library. The second line creates an LED object called “green” which is connected to GPIO2. The final line calls the blink method which as the name suggests makes our LED blink. By default the LED turns on for 1 second and then off for one second and then repeats forever. The blink method is quite powerful and actually allows you to pass arguments which control how long the LED is on, how long it is off and how often it blinks. That’s all a bit easy and we’re supposed to be learning some Python so let’s write our own code to control the blinking. Enter the following code:

```
from gpiozero import LED
from time import sleep

green = LED(2)

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

This programme will do exactly the same as our three line piece of code before. This time however we have imported the sleep class from the time library. We’ve then created a loop and used the .on and .off methods from the LED class to turn our LED on and off. Try adjusting the arguments of the two sleep commands and see how the blinking of the LED changes.

But what if you don't want the LED to blink forever? Let's try the same code but now replace the While loop with a For loop.

```
from gpiozero import LED
from time import sleep

green = LED(2)

for i in range (0, 3):
    green.on()
    sleep(1)
    green.off()
    sleep(1)
    print ("%d blink" % (i))
```

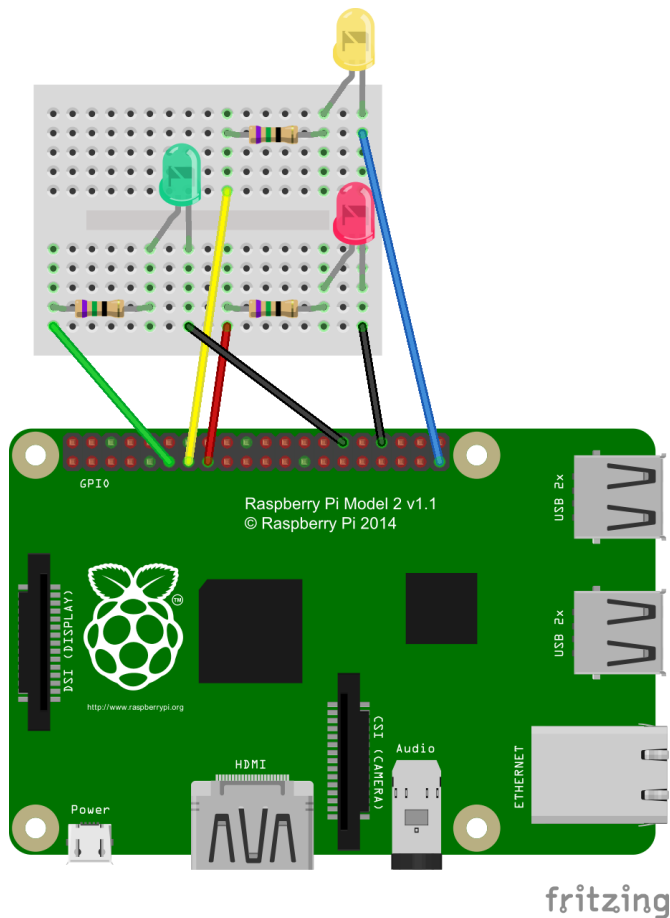


## Project Three – Basic Traffic Lights

### Introduction

So, having learned how to control a single LED, let's move on to something a little more complex. This time we're now going to have a go at switching 3 LEDs on and off, following a defined sequence; your very own set of traffic lights.

For this project, we will require 3 LEDs, 3 75R resistors (1 for each LED), and 6 jumper wires. The following diagram shows how we need to connect our components.



The jumper leads controlling the LEDs are connected to physical pins 11, 13 and 15 of the Pi. You will notice that these are actually labelled as pins 17, 27 and 22; and this is how we will need to refer to them in our Python code. Because we only have 2 black jumper leads in this kit, we'll use a blue lead as one of the ground leads. It doesn't matter which ground pins the black and blue jumper leads is connected to on the Raspberry Pi, but for this project we are going to use physical pins 30, 34 and 39, as this will make the wiring of our later projects a little bit easier to follow.

Having completed and checked the wiring, it's time to take a look at the code we will need to control our traffic lights. As with our previous projects, we're going to make use of the GPIO zero library, since this simplifies things for us. You'll notice that the first few lines of code for this project are almost identical to those from our previous project. The only difference is the pin number that is used for our green LED.

```

from gpiozero import LED
from time import sleep

green = LED(17)
yellow = LED(27)
red = LED(22)

while True:
    green.on()
    yellow.off()
    red.off()
    sleep(10)
    green.off()
    yellow.on()
    red.off()
    sleep(1)
    green.off()
    yellow.off()
    red.on()
    sleep(10)
    green.off()
    yellow.on()
    red.on()
    sleep(1)

```

Something else you might have noticed about this piece of code is that it is quite repetitive. When we write code, it is always a good idea to look at possible ways of simplifying the code, to make it easier to understand. One way that we can do this, is by making use of a programming construct known as a function, or procedure. Functions are used to define a sequence of instructions that would otherwise be repeated many times in our code. Functions and procedures also allow us to pass parameters to them in order to control the way that they work

So, let's look at an alternative way of writing this code. We're going to write a function, called `switchLights`, that switches the 3 LEDs on or off, depending on 3 parameters which are passed to the function, and then sleeps for a specified amount of time. The code below shows how we define this function in our code.

```

def switchLights (greenLight, yellowLight, redLight, sleepTime):
    if greenLight:
        green.on()
    else:
        green.off()

    if yellowLight:
        yellow.on()
    else:
        yellow.off()

    if redLight:
        red.on()
    else:
        red.off()

    sleep(sleepTime)

```

The full code to make use of this function is as follows:-

```
from gpiozero import LED
from time import sleep

green = LED(17)
yellow = LED(27)
red = LED(22)

def switchLights (greenLight, yellowLight, redLight, sleepTime):
    if greenLight:
        green.on()
    else:
        green.off()

    if yellowLight:
        yellow.on()
    else:
        yellow.off()

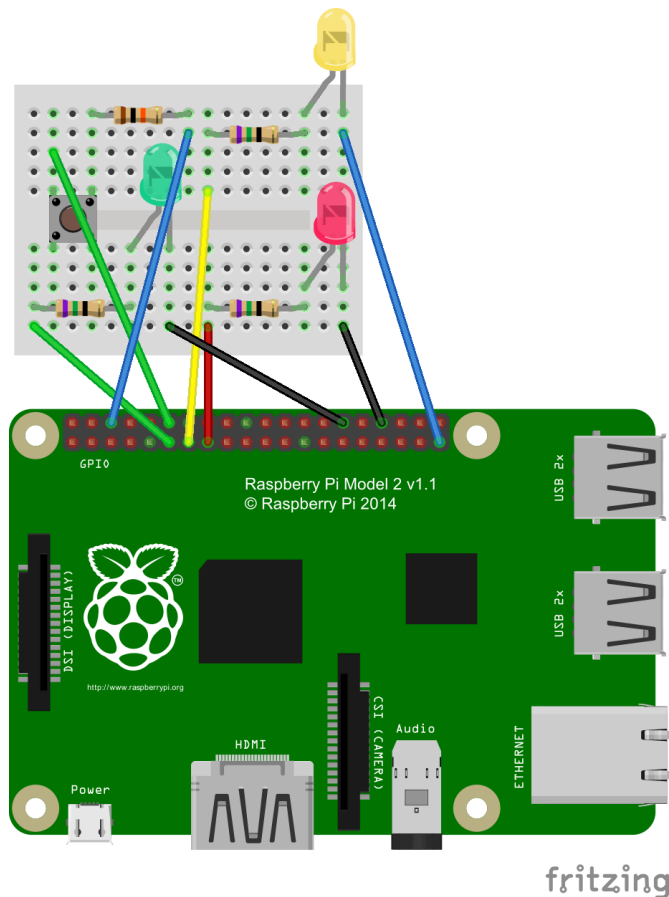
    if redLight:
        red.on()
    else:
        red.off()

    sleep(sleepTime)

while True:
    switchLights (True, False, False, 10)
    switchLights (False, True, False, 1)
    switchLights (False, False, True, 10)
    switchLights (False, True, True, 1)
```

## Project Four – Adding A Switch Into The Traffic Light Circuit

So, we have our traffic lights working, so let's move on again. Again, we're going to build upon our previous project, so all the LEDs, resistors, and jumper wires that you've already got in place can stay where they are for now. For this project all we're going to do is to add a simple push button so that we can add a little more control over how our traffic lights work. However, when we add a push button to a circuit for the Raspberry Pi, we also need to add a resistor. The resistor we will add is a 10k resistor, which has a brown band, a black band and an orange band. Connect these new components as shown in the diagram below.



Before looking at the code for this project, let's try to understand the circuit connections a little first; for example, why do we need the resistor? You'll notice that one of the leads of the 10k resistor is connected to one of the ground (0v) GPIO pins (physical pin 6) on the Raspberry Pi through the blue jumper wire. The other lead of the resistor is connected to one of the pins of the push button. We have also connected the adjacent pole of the push button to GPIO pin 18 (physical pin 12) using the green jumper wire. We are going to use GPIO pin 18 as an input to determine whether or not the push button has been pressed. Because the 10k resistor is connected to 0v, when the button is pressed, the voltage at GPIO pin 18 will be pulled down to 0v. When the button is not pressed, you might think that the voltage input to GPIO pin 18 is indeterminate (floating); however, the Raspberry Pi has internal pull-up and pull-down resistors, which can be configured through software, and we'll look at this when we look at our program.

So, why do we need the resistor? The 10k resistor is there to protect us from damaging our Raspberry Pi if we were to accidentally make a mistake in our code. For example, if we were to

configure GPIO pin 18 as an output, and to set pin 18 to +3.3v in our code, and then pressed the push button, GPIO pin 18 (+3.3v) would effectively become shorted straight through to the ground pin. Without the 10k resistor, this would cause damage to the Raspberry Pi because of the current flowing directly between these 2 pins. The 10k resistor would limit this current to a tolerable level were these 2 pins to become shorted together in this way.

The code needed to detect when the button is pressed is really simple, thanks to the GPIO zero library. The following lines of code show how to import the necessary class (Button) from the GPIO zero library, and how to define a button object.

```
from gpiozero import LED, Button

button = Button(18, pull_up = True)
```

The 'pull\_up = True' parameter tells GPIO zero to configure the Raspberry Pi's internal pull-up/pull-down resistors so that when the button is not pressed, pin18 will be pulled high by default, and will thus have a voltage of +3.3v. Because the other pole of the push button is connected to ground by the 10k resistor, when the button is pressed pin18 will have a voltage of 0v.

The full code for this project is as follows:-

```
from gpiozero import LED, Button
from time import sleep

green = LED(17)
yellow = LED(27)
red = LED(22)
button = Button(18, pull_up = True)

def switchLights (greenLight, yellowLight, redLight, sleepTime):
    if greenLight:
        green.on()
    else:
        green.off()

    if yellowLight:
        yellow.on()
    else:
        yellow.off()

    if redLight:
        red.on()
    else:
        red.off()
    sleep(sleepTime)

while True:
    switchLights (True, False, False, 10)
    print ("Waiting for button press ...")
    button.wait_for_press()
    print ("Button pressed")
    switchLights (False, True, False, 1)
    switchLights (False, False, True, 10)
    switchLights (False, True, True, 1)
```

You'll see from this code that the green LED will light for 10 seconds, following which the code will wait for the button to be pressed. When the button is pressed, the lights will immediately start to switch until they turn back to green again and then wait for a further button press.

As it stands, this is not very realistic for a set of button controlled traffic lights, so we'll look at modifying our code so that it waits for a random amount of time after the button is pressed before starting to switch the lights. In order to do this, we'll need to import another Python library and generate a random wait time; as follows:-

```
import random

wait = random.randint(10,15)
```

This code imports the Python library that allows us to generate a random number, and then generates a random wait time of between 10 and 15 seconds. The full code for our more realistic set of traffic lights is shown below:-

```
from gpiozero import LED, Button
from time import sleep
import random

green = LED(17)
yellow = LED(27)
red = LED(22)
button = Button(18, pull_up = True)

def switchLights (greenLight, yellowLight, redLight, sleepTime):
    if greenLight:
        green.on()
    else:
        green.off()

    if yellowLight:
        yellow.on()
    else:
        yellow.off()

    if redLight:
        red.on()
    else:
        red.off()
    sleep(sleepTime)

while True:
    switchLights (True, False, False, 1)
    print ("Waiting for button press ...")
    button.wait_for_press()
    wait = random.randint(10, 15)
    print ("Button pressed - wait for %d seconds ..." % wait)
    switchLights (True, False, False, wait)
    switchLights (False, True, False, 1)
    switchLights (False, False, True, 10)
    switchLights (False, True, True, 1)
```