<u>**Pretty lights!!**</u>

(or, How to Control Neopixels)

Firstly, 'Neopixel' is an Adafruit trade name but, like Hoover, it has become the common way to refer to individually addressable RGB LEDs. The ones I use are designated WS2811/WS2812 and use type 5050 LEDs. They use 3 wires; Vcc, Gnd, Data. Below is a link to an example available on eBay:

https://www.ebay.co.uk/itm/1m-5m-5V-WS2812B-5050-RGB-30-60LEDs-M-LED-Strip-ws2812-IC-Individual-Addressable/132734477057?hash=item1ee7976301:m:mQCdCQBS1-QUY2AFeu9KEyA:rk:2:pf:0

They come in lengths from 1m to 5m, can be IP65 water-resistant, and are available in a range of densities (no. of pixels per metre).

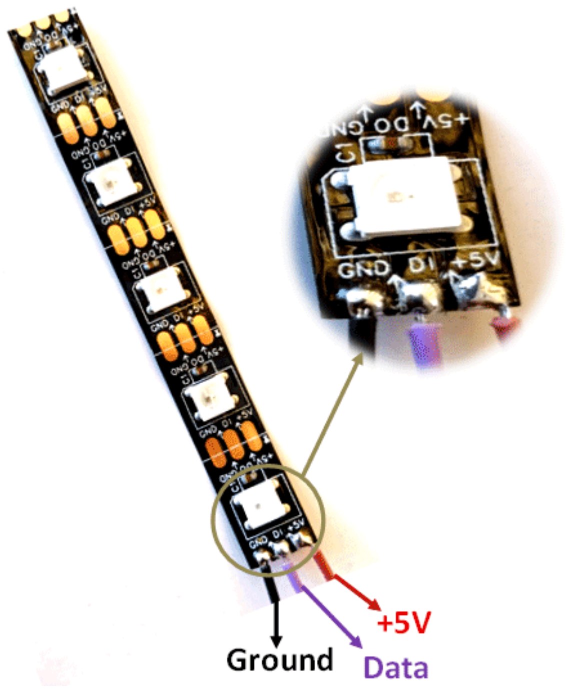
They may also have two further wires at the far end of the strip. These are duplicate Vcc and Gnd connections.

You can also buy them in solid strips of 8 pixels and in circles of varying diameters. Example below:

https://www.ebay.co.uk/itm/8-12-16-24-60-Bit-LED-RGB-5050-WS2812B-WS2812-NeoPixel-Ring-Arduino-Raspberry-Pi/173069314025?hash=item284bbc27e9:m:mNKMAt0PkpxN9KKlkxOpeww:rk:10:pf:0

Strips (or rings) of neopixels can be chained together to form longer strips or more intricate patterns. Conversely, the flexible strips can be cut to any size just by slicing through the set of pads between LEDs.

In the picture on the following page, you can see the copper contacts between each LED. If you look closely at the Data (centre) connections, the one going into the LED is marked DI or D In, and the one coming out is marked DO or D Out. If you are connected multiple strips, you need to ensure that the DO of one strip is connected to the DI of the next. No connections need to be made at the end of the strip(s).



The first LED is pixel 0 or 1, depending on the platform and programming language used. All of the methods I will show work on the same basic principal; colour data (in RGB format, ie: red would be (255, 0, 0)), is sent to a specific pixel. The rest is just loops and Maths!

Some methods allow the user to set the brightness separately, otherwise it can be done by reducing the three values in equal ratios. E.g. (255, 255, 255) would be bright (sometimes VERY bright!) white, whereas (128, 128, 128) would be roughly half as bright.

A quick word about power. None of these methods need any other components apart from your controller of choice, the neopixels, and a battery pack. All of these controllers are able to power about 8 neopixels, but a separate power supply is required for more than this due to the high current requirements.

Here's the catch: the Pi, Crumble, and Microbit all use a 3v3 logic voltage, meaning the pin that you use for Data outputs 3.3V. The datasheet for 5050 LEDs shows that, for reliable control, a logic 'High' will be detected at VCC x 0.7. Working this backwards tells us that, in order for 3v3 to be detected as 'High', the supply voltage should be no higher than 4.7v. In practice, I use either a 3xAA pack of alkaline batteries (3 x 1.5 = 4.5) or a 4xAA pack of rechargeables (4 x 1.2 = 4.8)

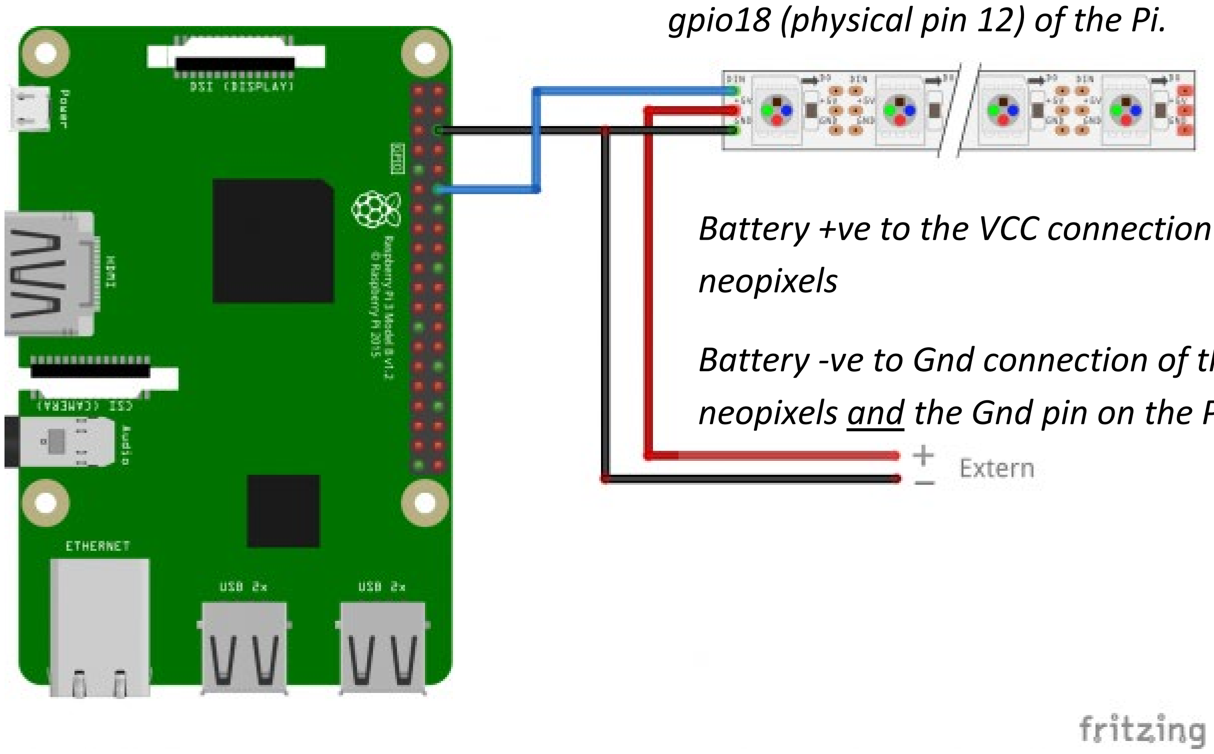
Connecting it all together

Raspberry Pi

Warning: the Python library will not work with the Pi2 or Pi3 without some changes to the config files. A great excuse to get yourself a Pi Zero!

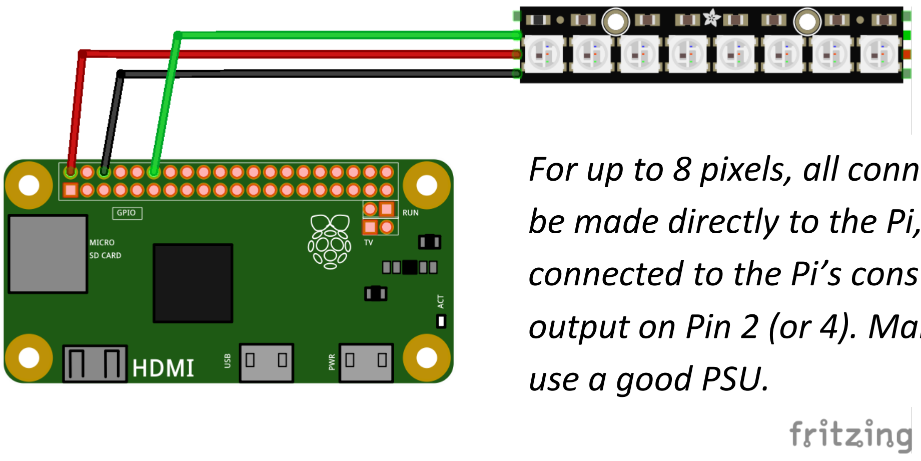
The key thing when using an external power source (with both the Pi and the Microbit) is that all three grounds need to be linked. The wiring is fairly simple:

Data In on the neopixels to gpio18 (physical pin 12) of the Pi.



Battery +ve to the VCC connection of the neopixels

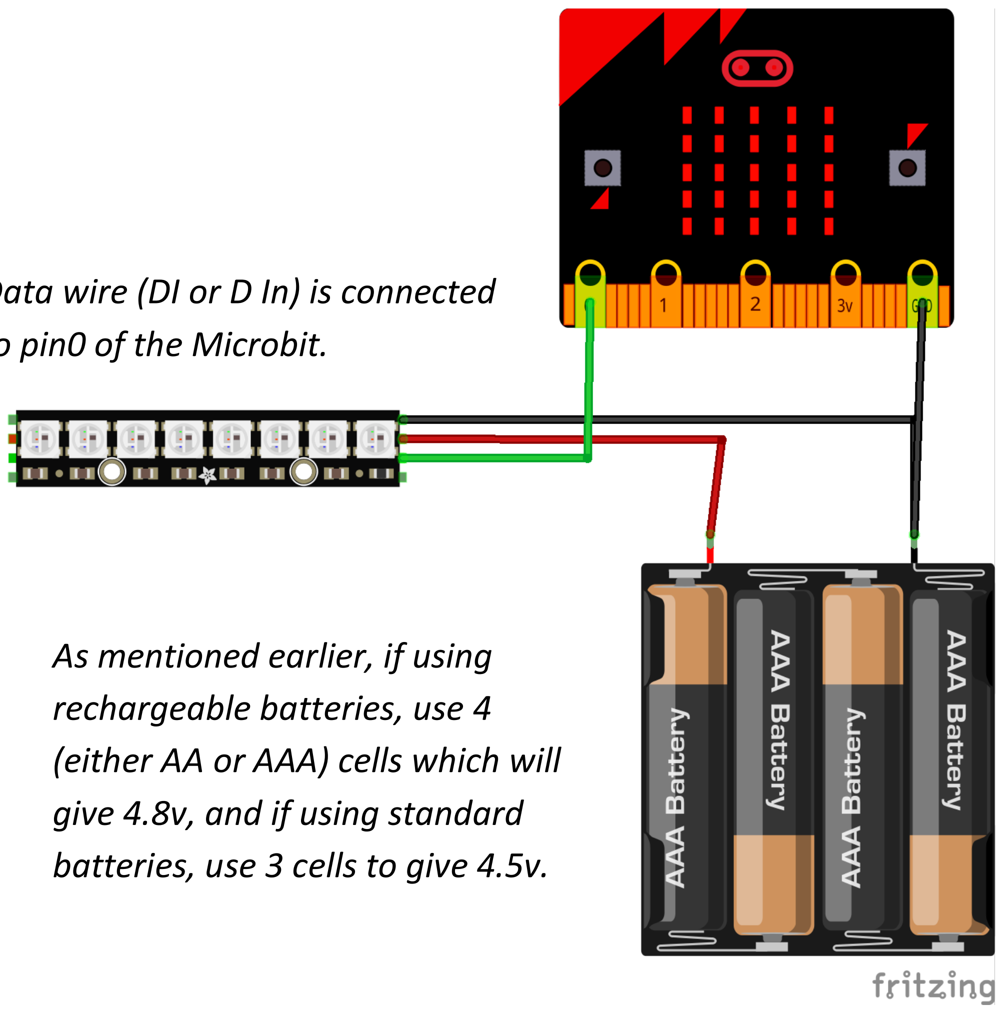
Battery -ve to Gnd connection of the neopixels <u>and</u> the Gnd pin on the Pi.



For up to 8 pixels, all connections can be made directly to the Pi, with Vcc connected to the Pi's constant 5v output on Pin 2 (or 4). Make sure you use a good PSU.

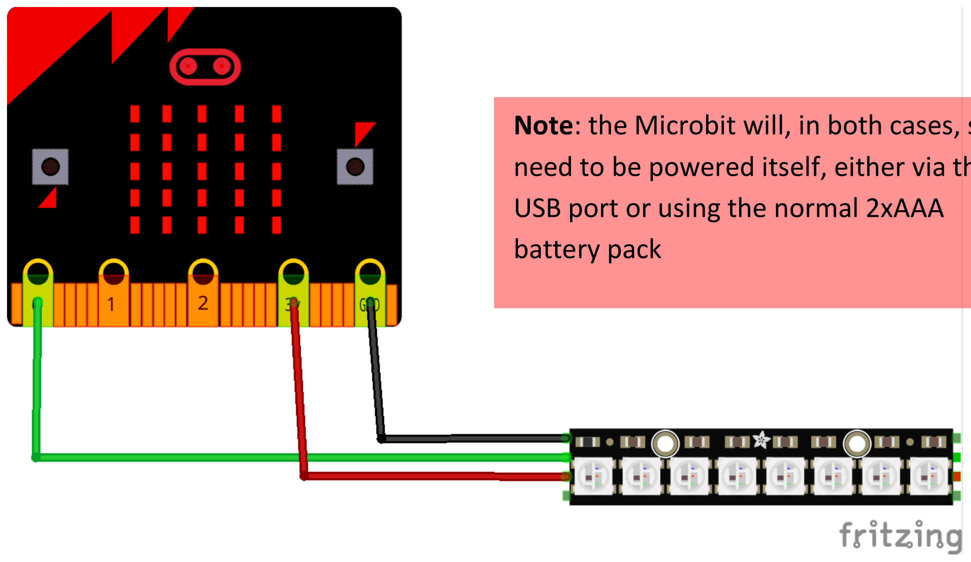
BBC Microbit

As with the Raspberry Pi, when using an external power source with the Microbit, you need to ensure that the grounds are all linked, otherwise you have an incomplete circuit.



Data wire (DI or D In) is connected to pin0 of the Microbit.

As mentioned earlier, if using rechargeable batteries, use 4 (either AA or AAA) cells which will give 4.8v, and if using standard batteries, use 3 cells to give 4.5v.



Note: the Microbit will, in both cases, still need to be powered itself, either via the USB port or using the normal 2xAAA battery pack

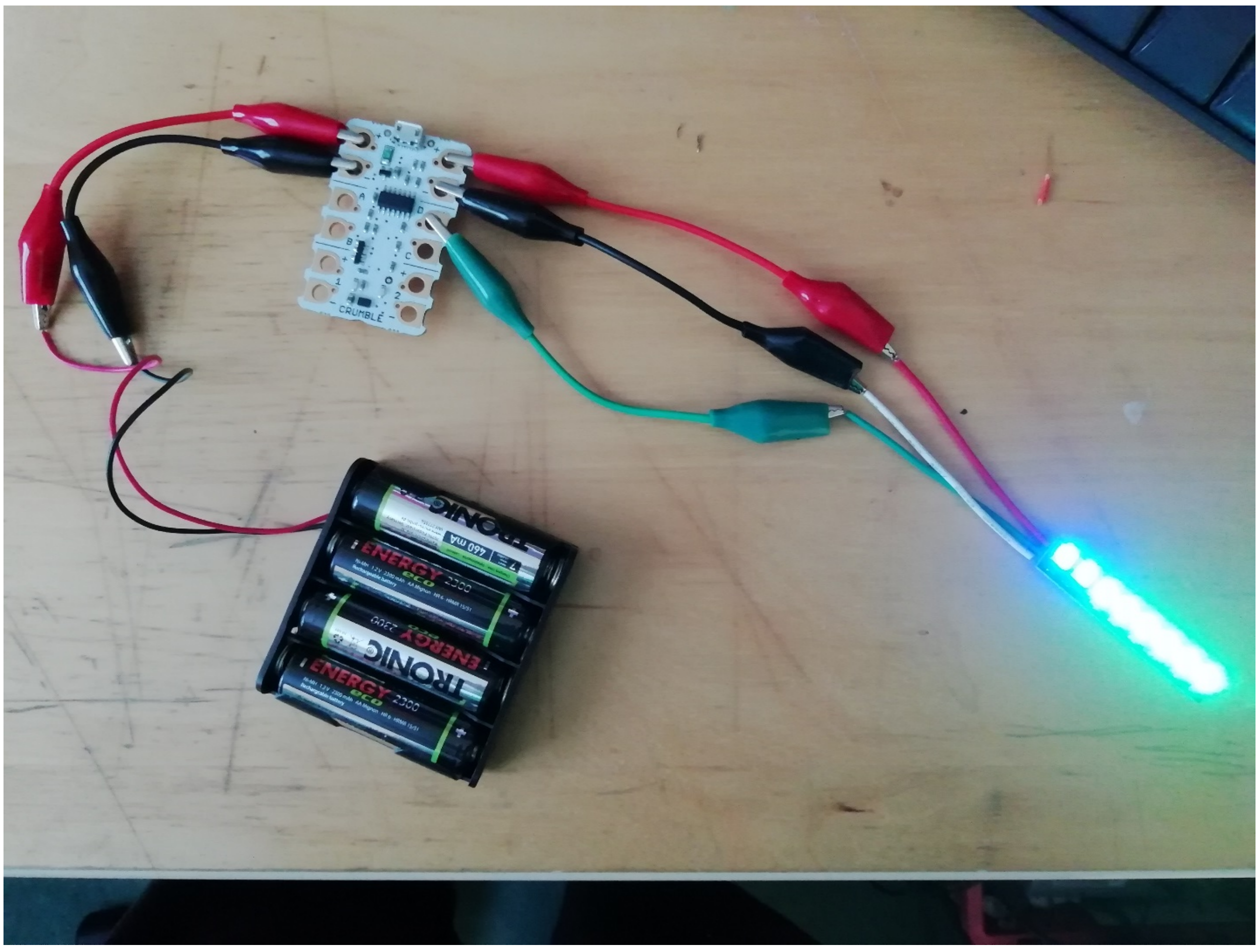
Crumble Controller

The Crumble system has its own version of neopixels, called Sparkles. These are suitable for use with croc clips, having the same round connection points as the Crumble itself. However, Sparkles and neopixels are electrically identical and the same methods and code can be used for both.

Once you have written your code, download it to the Crumble as normal. Then disconnect the Crumble from the computer and connect as per the photo below.

The connections between the Crumble and the neopixel strip are:

Red	-	Vcc (+ve)
Black/white	-	Ground (-ve)
Green	-	Data In (D)



Let’s Get Coding

There are numerous ways to program your devices.

The Crumble has its own block-based coding language.

The Microbit can be coded in MicroPython (using the excellent ‘Mu’ environment) or with Microsoft’s Makecode language.

The Raspberry Pi can be programmed in a much greater variety of languages, but we will be looking at (a special version of) Scratch 1.4 and Python.

For information on how to install/use these tools, follow the relevant link:

Crumble - https://redfernelectronics.co.uk/crumble-software/

Microbit

MakeCode	- https://makecode.microbit.org/
Mu	- https://codewith.mu/

Raspberry Pi

Scratchgpio (for V1.4)	- http://simplesi.net/scratchgpio/scratch-raspberrypi-gpio/
Python	- https://codewith.mu/

The code snippets on the following pages all do the same thing. They will light up a strip of 8 neopixels, one by one, first in red, then green, then blue, until you terminate the program or pull the power.

Have fun!

Further reading and resources

http://www.multiwingspan.co.uk/micro.php?page=pxtneo

http://www.multiwingspan.co.uk/micro.php?page=neopix

https://redfernelectronics.co.uk/projects/light-painting/

http://simplesi.net/controlling-neopixels/

Python on the Raspberry Pi

```
from rpi_ws281x import *
from time import sleep
```

rpi_ws281x is the name of the neopixel library

```
LED_COUNT      = 8
LED_PIN        = 18
LED_FREQ_HZ    = 800000
LED_DMA        = 10
LED_BRIGHTNESS = 255
LED_INVERT     = False
LED_CHANNEL    = 0
```

LED_COUNT is the number of pixels in the strip

LED_BRIGHTNESS can be changed if you want to use less power or save your eyes!

Everything else can be left as default values

```
def colorWipe(strip, color):
    for i in range(strip.numPixels()):
        strip.setPixelColor(i, color)
        strip.show()
        sleep(0.01)
```

Function to loop through each pixel in turn. The colour is set by the function call.

Important: remember to use strip.show() at the end to display the updates

```
strip = Adafruit_NeoPixel(LED_COUNT, LED_PIN,
LED_FREQ_HZ, LED_DMA, LED_INVERT, LED_BRIGHTNESS,
LED_CHANNEL)
```

Instantiates the neopixel class as ‘strip’, with the arguments set at the start of the code

```
strip.begin()
```

Strip.begin() initialises the neopixels

```
while True:
    colorWipe(strip, Color(255, 0, 0))
    colorWipe(strip, Color(0, 255, 0))
    colorWipe(strip, Color(0, 0, 255))
```

The ‘while’ loop calls the colorWipe function 3 times, changing the colour each time, from red, to blue, then green.

Scratchgpio on the Raspberry Pi

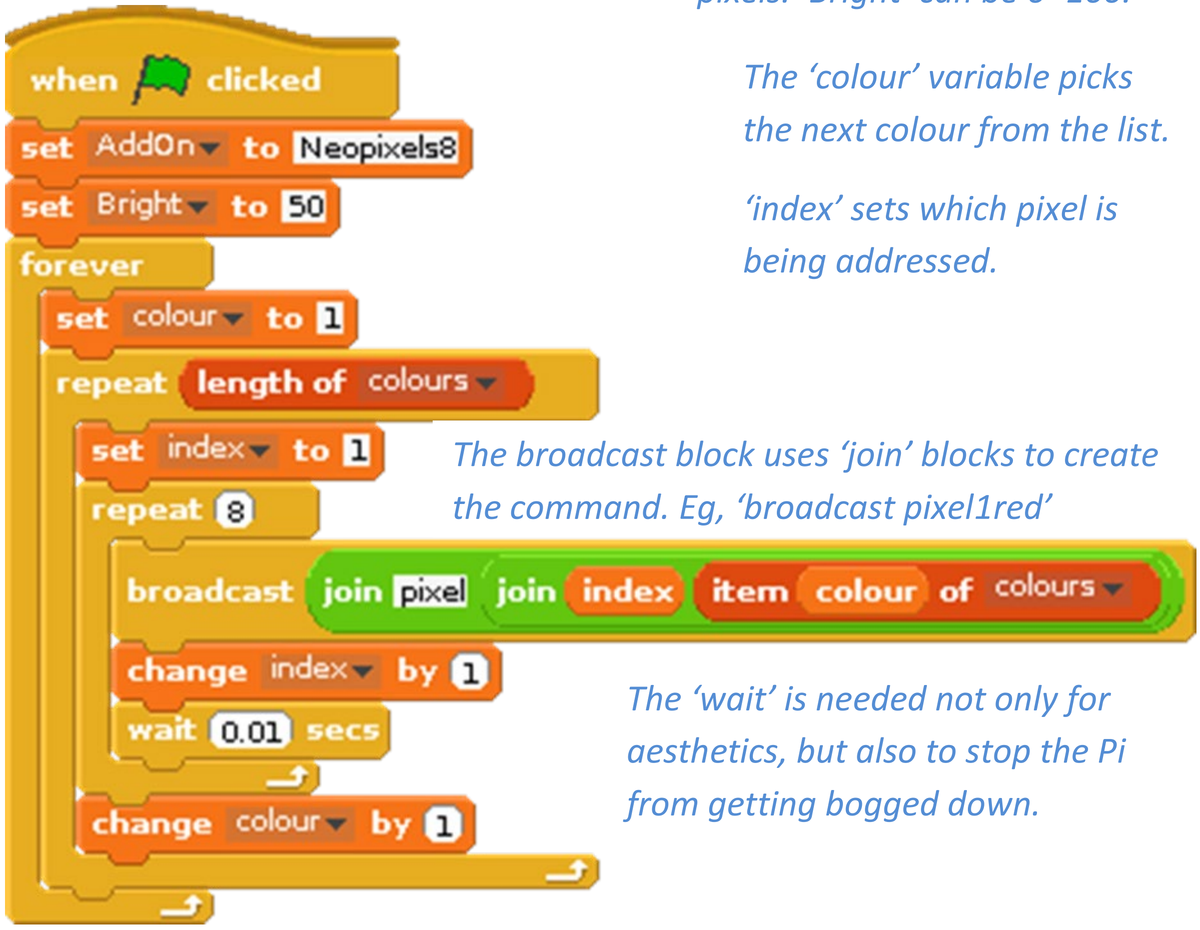
Before starting your code, you need to make a list, with the entries; red, green, *and* blue. *(Other allowable colours are: cyan, magenta, yellow, white).*

You also need to make the variables 'colour' and 'index'.

'set AddOn' sets the number of pixels. 'Bright' can be 0 -100.

The 'colour' variable picks the next colour from the list.

'index' sets which pixel is being addressed.



```
when [flag] clicked
set AddOn to Neopixels8
set Bright to 50
forever
    set colour to 1
    repeat (length of colours)
        set index to 1
        repeat 8
            broadcast (join pixel (join (index) (item (colour) of colours)))
            change index by 1
            wait 0.01 secs
        change colour by 1
```

The broadcast block uses 'join' blocks to create the command. Eg, 'broadcast pixel1red'

The 'wait' is needed not only for aesthetics, but also to stop the Pi from getting bogged down.

Scratchgpio uses Python code running in the background to interpret the 'broadcast' commands. It can be quite resource-intensive and can slow down considerably when processing lots of pixels.

Python on the BBC Micro:bit

Thanks to the excellent neopixel library for the Micro:bit, controlling neopixels is fairly straightforward.

```
from microbit import *
import neopixel
```

Import all of the Microbit commands, and the neopixel library.

```
np = neopixel.NeoPixel(pin0, 8)
```

Instantiate the neopixel module, setting the pin used and the number of pixels.

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

Set up the colours as lists to make the code more readable and simpler.

```
colours = (red, blue, green)
```

'colours' is a list of lists, which we will iterate through in the main loop.

```
while True:
    for colour in colours:
        for i in range(0, len(np)):
            np[i] = colour
            np.show()
            sleep(100)
```

For each colour, select each pixel individually and change it to that colour.

np[i] sets the pixel at index 'I' to the chosen colour.

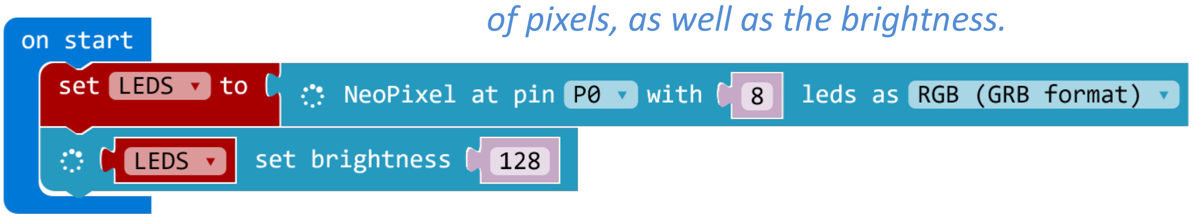
Remember, Python uses zero-indexing (it starts counting at 0) so the first pixel is number 0.

np.show() applies the updates.

Microsoft MakeCode on the BBC Micro:bit

Before starting your code, you will need to add the neopixel extension. In the coding window, click on 'Advanced' then 'Add Extension'. Select 'AdaFruit NeoPixel Driver'.
You will see a new set on blocks have been added, under the heading 'Neopixel'.

The commands in the 'on start' block initialise the neopixels, setting the pin and the number of pixels, as well as the brightness.

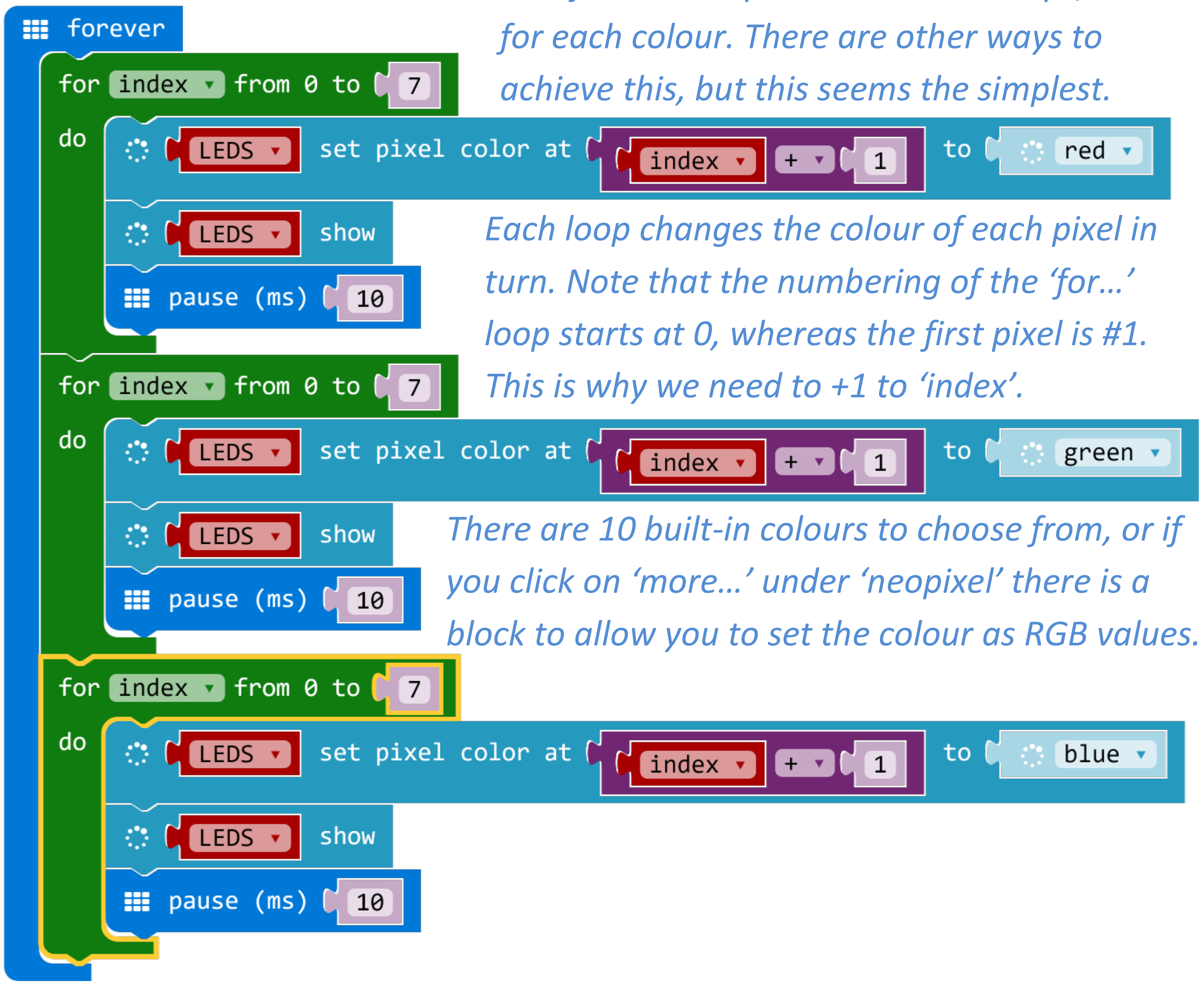


```
on start
    set LEDS to  NeoPixel at pin P0 with 8 leds as RGB (GRB format)
    LEDS set brightness 128
```

The 'forever' loop contains three loops, one for each colour. There are other ways to achieve this, but this seems the simplest.

Each loop changes the colour of each pixel in turn. Note that the numbering of the 'for…' loop starts at 0, whereas the first pixel is #1. This is why we need to +1 to 'index'.

There are 10 built-in colours to choose from, or if you click on 'more…' under 'neopixel' there is a block to allow you to set the colour as RGB values.

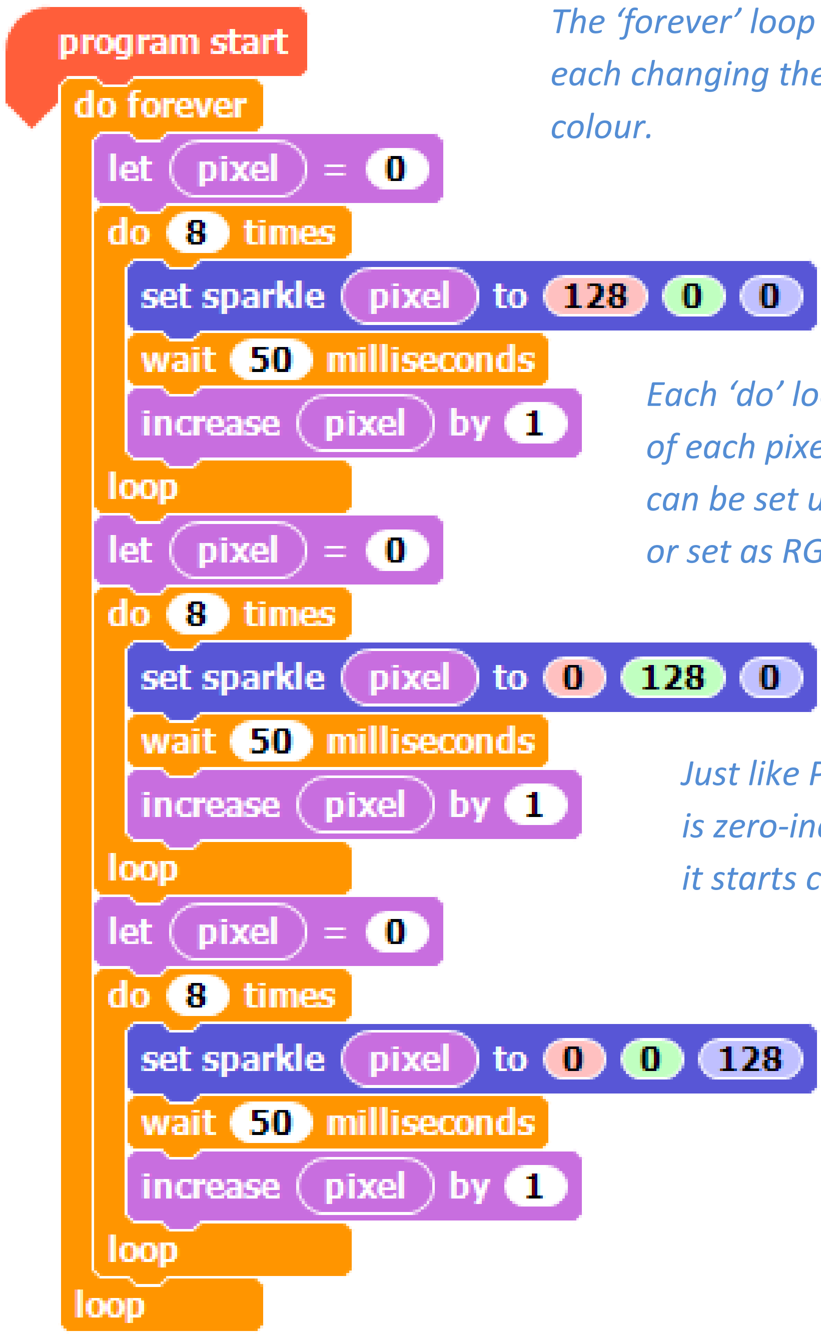


```
forever
    for index from 0 to 7
    do
        LEDS set pixel color at index + 1 to red
        LEDS show
        pause (ms) 10
    for index from 0 to 7
    do
        LEDS set pixel color at index + 1 to green
        LEDS show
        pause (ms) 10
    for index from 0 to 7
    do
        LEDS set pixel color at index + 1 to blue
        LEDS show
        pause (ms) 10
```

Redfern Crumble

The Crumble is probably the easiest platform to get started with Neopixels, which they call 'Sparkles'.

There is no setup needed with the Crumble, so you can dive straight into the code.



```
program start
do forever
    let pixel = 0
    do 8 times
        set sparkle pixel to 128 0 0
        wait 50 milliseconds
        increase pixel by 1
    loop
    let pixel = 0
    do 8 times
        set sparkle pixel to 0 128 0
        wait 50 milliseconds
        increase pixel by 1
    loop
    let pixel = 0
    do 8 times
        set sparkle pixel to 0 0 128
        wait 50 milliseconds
        increase pixel by 1
    loop
loop
```

The 'forever' loop contains three loops, each changing the pixels to a different colour.

Each 'do' loop changes the colour of each pixel in turn. The colour can be set using a 'colour picker' or set as RGB values.

Just like Python, Crumble code is zero-indexing, which means it starts counting at 0.