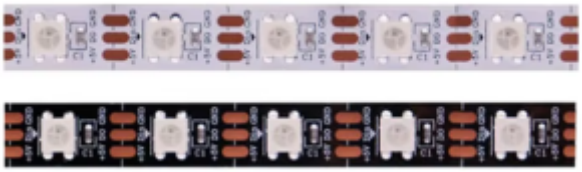
12 NeoPixel LED Strip

Aug 2026

Introduction



Strips of NeoPixels are very cool to use. You might have heard of them being referred to as Addressable RGBs (aRGB) LEDs. While neopixels look a bit like Xmas tree lights, they are in fact individually addressable, so you can turn on one, none, all or any combination of the LEDs, and they can all be differently coloured.

Please note. The first part of this wiki page is about the NeoPixels and the code required to run them. It seems quite long, but this is only because the code has loads of useful comments in it. It is quite simple code, and when you remove all the comments, it is also rather short.

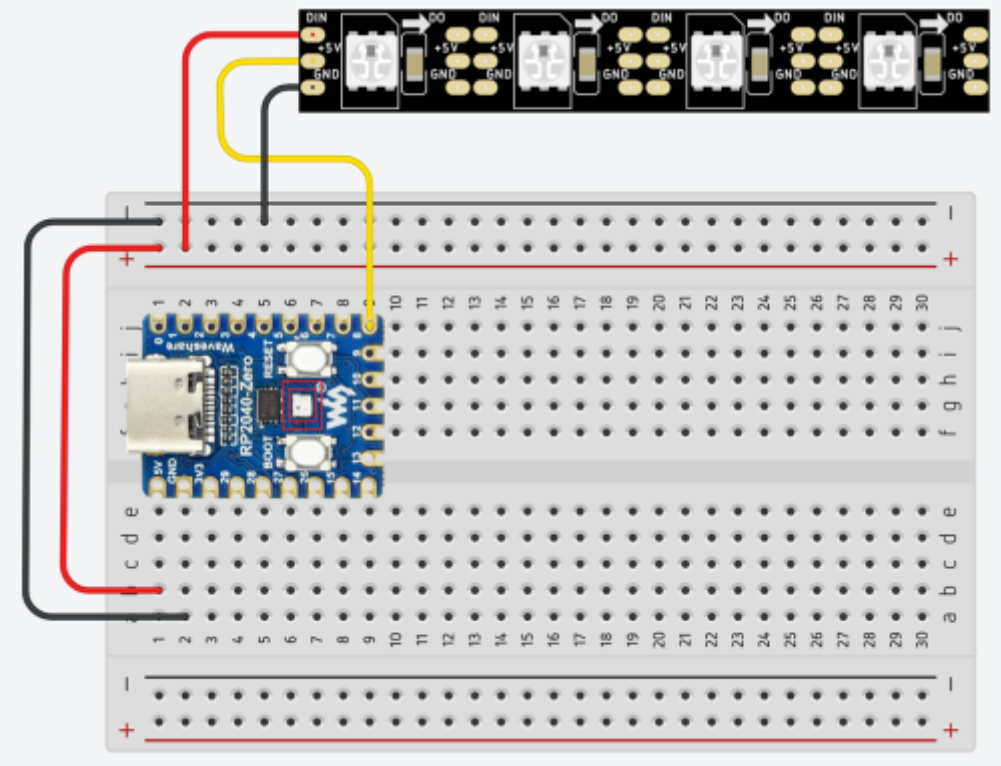
After this, there is some other projects you can follow along with about the different ways you could use the NeoPixel LEDs, they all use exactly the same wiring, so trying them out is simply a case of running a new piece of code, enjoy :)

The Circuit

The circuit for the NeoPixels is very simple. All the electronics that you need are built in to the strip, so you don't need to worry about things like current limiting resistors etc.

NeoPixels come in two main types that I have seen (ignoring waterproofing levels) and that is 5V and 12V. So make sure you have the correct type.

Please see below for setup I am using.



I am using a 5v strip of neopixels, so I can power everything I need from a USB connector. When we plug the USB-C connector in to this device, it will power both the Pico and the NeoPixels.

Please note, this will work with either the Raspberry Pi or Waveshare zero.

A quick note about setting the LED Colour and Brightness

We set the colour of RGB LEDs by sending an RGB value to each chip LED. For example, I can send the following:
Remember (R, G, B) (some NeoPixels have the RGB in a different order, beware!)

download

```
(255, 0, 0),    # Red
(0, 255, 0),    # Green
(0, 0, 255),    # Blue
(255, 255, 0),  # Yellow
(255, 0, 255),  # Magenta
(0, 255, 255),  # Cyan
(255, 255, 255),# White
(0, 0, 0)       # Off
```

So the value 255 turns on the respective LED. 255, 0, 0 is Red on, Green off, Blue off. So you just see a RED coloured LED.

The value 255 is the brightness, so if you use lower values, say 128, 0, 0 then you get a much dimmer Red LED. I personally use the value 8, as scorched retinas is really not my thing.

Micro Python Test Code

Before we let our imagination run away with what we can do with NeoPixels, lets run some simple test code to ensure that everything is working as expected.

Here is a simple test set for the LEDs.

| download

```
# WS2812 / NeoPixel Example
# Raspberry Pi Pico + MicroPython
#
# Controls a strip of 12 LEDs connected to GP0.

from machine import Pin
import neopixel
import time

# ------------------------------
# Configuration
# ------------------------------

LED_PIN = 8          # GPIO connected to DIN
NUM_LEDS = 12        # Number of LEDs, 0-11 is used, not 1-12
LUMAR = 8            # Red Brightness
LUMAG = 8            # Green Brightness
LUMAB = 8            # Blue Brightness

# Create the NeoPixel object
np = neopixel.NeoPixel(Pin(LED_PIN), NUM_LEDS)
print (np)

# ------------------------------
# Helper Functions
# ------------------------------

def clear():
    """Turn all LEDs off."""
    for i in range(NUM_LEDS):
        np[i] = (0, 0, 0)
    np.write()

def fill(color):
    """Set every LED to the same colour."""
    for i in range(NUM_LEDS):
        np[i] = color
    np.write()

# ------------------------------
# Main Program
# ------------------------------

while True:

    print ("Set every LED to Red")
    # Red
    fill((LUMAR, 0, 0))
    time.sleep(1)

    print ("Set every LED to Green")
    # Green
    fill((0, LUMAG, 0))
    time.sleep(1)

    print ("Set every LED to Blue")
    # Blue
    fill((0, 0, LUMAB))
    time.sleep(1)

    print ("Set every LED to White")
    # White
    fill((LUMAR, LUMAG, LUMAB))
    time.sleep(1)

    print ("Clear all LEDs")
    # Off
    clear()
    time.sleep(0.5)
```

What the code above does, is to turn off all LEDs, then set all LEDs to Red, Green, Blue then repeat the sequence. Lets break down the code to see what each section does.

This first part

| download

```
# Controls a strip of 12 LEDs connected to GP0.

#This imports the Pin class, which lets you control the Raspberry Pi Pico's GPIO pins.
#You need this because NeoPixels require a digital output pin (like GP0) to send data.
from machine import Pin

#This loads the NeoPixel driver library.
#It gives you the NeoPixel class with methods like:

#np[i] = (r, g, b) – set one LED
#np.fill((r, g, b)) – set all LEDs
#np.write() – send the data to the strip

#Without this import, you can't talk to the LEDs.
import neopixel

#This gives you access to timing functions like time.sleep().
#This allows us to keep the LEDs on or off for a set time.
import time
```

So the first block of code sets up the Pico hardware.

Next we can set some variables.

| download

```
LED_PIN = 8          # GPIO connected to DIN. This is the GPIO Pin that you are using on your
Pico.
NUM_LEDS = 12        # Number of LEDs, 0-11 is used, not 1-12. Set this to the number of LEDs on
your strip/ring.

#  each time you write to an LED, you can use a fixed set of values, say (255,0,0) for red,
however,
#  if you want to change the brightness at any time, you have to edit every value in your code. So
by using a
#  variable here, we can # set the brightness once and it is used every time we write to the LEDs
(LUMAR, 0, 0)
#  for example
LUMAR = 8
LUMAG = 8
LUMAB = 8
```

We have set our variable values.

Next, we are defining a couple of functions.

| download

```
# In your code, def clear(): means you are defining a function named clear.
# We have to call the function for it to work. By having the function outside of our main code
# we can call this function from anywhere in our program.
def clear():
    """Turn all LEDs off."""
    for i in range(NUM_LEDS):    #At the start of the code, a variable NUM_LEDS = 12 tells the
code we have 12 LEDs
        np[i] = (0, 0, 0)        #np (neopixel) [i] the current value of i (from 0 to 11) for
NUM_LEDS = 12
        print (i)                #prints current value of i to Thonny output.
    np.write()                   #writes (0, 0, 0) to currently selected aRGB (defined by current
value of i)
```

```
# fill is a function to set the LED colour.
# color is a built in python variable that holds an RGB value.
# when you call the fill function (fill(color)) you have to set the value of color.
# Ex: fill((255, 0, 0)) will set all LEDs red. 'fill' sets all LEDs, not just selected LEDs.
def fill(color):
    """Set every LED to the same colour."""
    for i in range(NUM_LEDS):
        np[i] = color
    np.write()
```

So above we have created two functions, one to clear() (turn off) the LEDs and one to set the LED Color fill(color). These are both called from our main body of code.

Now for the main code body, this is actually the simplest part, because all we are doing here is calling the code we have already written.

| download

```
# ------------------------------
# Main Program
# ------------------------------

while True:    # run this loop for ever.

    print ("Set every LED to Red")    # this is just a comment
    # Red                             # as is this
    fill((LUMAR, 0, 0))               # fill (LUMAR, 0, 0) calls the fill function we defined
above and sets the
                                      # data value passed to the color variable to LUMAR (which we
defined as a
                                      # value of 8 at the start of this code so color now = (8, 0
,0) as these
                                      # LEDs are R, G, B this means set LEDs to all red, no green,
no blue.

    time.sleep(1)                     # Wait for 1 second.

#   Now we will repeat what we did for Red (255, 0, 0) for Green (0, 255, 0), Blue (0, 0, 255),
#   White (255, 255, 255) then Off (0, 0, 0).

    print ("Set every LED to Green")
    # Green
    fill((0, LUMAG, 0))
    time.sleep(1)

    print ("Set every LED to Blue")
    # Blue
    fill((0, 0, LUMAB))
    time.sleep(1)

    print ("Set every LED to White")
    # White
    fill((LUMAR, LUMAG, LUMAB))
    time.sleep(1)

    print ("Clear all LEDs")
    # Off
    clear()
    time.sleep(0.5)
```

Using this knowledge, we can have some fun working with these LED strips, check out below for some ideas.

Using NeoPixel Strip for Traffic Lights

So this is a great idea for several reasons.

- Less components, no Transistors or resistors (9 components saved).
- Less wiring, just 3 wires, 5v, data, 0v.
- Simpler control of LED Colour and Brighness.
- Can double up LEDs to get an even better light output

So there are some clear reasons to use a NeoPixel strip over discrete components. The code is very similar, and just as easy to understand. If you are using the same connection as above, then use this code below. The only thing you might have to change is the LED_PIN example as you might not be using the same Pico GPIO pin as the example.

| download

```
from machine import Pin
import time
import neopixel

LED_PIN = 8
NUM_LEDS = 12

np = neopixel.NeoPixel(Pin(LED_PIN), NUM_LEDS)

# --- Helper functions ---------------------------------------------

def clear():
    """Turn off all LEDs."""
    np.fill((0, 0, 0))
    np.write()

def set_pixels(indices, color):
    """Set multiple LEDs to the same colour."""
    for i in indices:
        np[i] = color
    np.write()

# --- Traffic light colours ----------------------------------------

RED     = (255, 0, 0)
AMBER   = (100, 25, 0)
GREEN   = (0, 255, 0)

# --- Main loop ----------------------------------------------------

while True:
    # Red
    clear()
    set_pixels([0, 1], RED)
    time.sleep(3)

    # Red + Amber
    set_pixels([4, 5], AMBER)
    time.sleep(2)

    # Green
    clear()
    set_pixels([8, 9], GREEN)
    time.sleep(3)

    # Amber
    clear()
    set_pixels([4, 5], AMBER)
    time.sleep(1)
```

So if you set this up correctly, you should have something resembling the video below:

neopixel-traffic-light.mp4

I cannot wait to build this in to an actual traffic light ;)

Here is the NeoPixel Traffic Light project built in the Wokwi simulator.

wokwi-neopixel-traffic-lights.mp4

The NeoPixel in Wokwi isn't perfect, but it illustrates that the project will function as expected.

NeoPixel LED Chase Sequence

This next code scrolls down the line of NeoPixels. The most important line of code in this example is this:

- for i in range(NUM_LEDS):

What this does is creates a range of numbers from the value of the NUM_LEDS (12 in our example) and this looks like this:

- 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11

So this for loop starts at '0' and does the following:

| download

```
for i in range(NUM_LEDS):    # get the num range (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11)
        clear()              # call the clear function to turn off all leds
        np[i] = color        # set the color of first LED
        np.write()           # write the LED status
        time.sleep(delay)    # wait for delay time.

        # now the loop will do the same but for LED2, 3, 4 etc.
        # we clear the LEDs each time so that only 1 LED is ever illuminated.
```

We call this chase function 3 times:

| download

```
    # Chase effect
    chase((255, 0, 0), 0.08)
    chase((0, 255, 0), 0.08)
    chase((0, 0, 255), 0.08)
```

First we set LEDs to Red, then second time it's Green then lastly we set Blue.

Here is the full code:

| download

```
# WS2812 / NeoPixel Example
# Raspberry Pi Pico + MicroPython
#
# Controls a strip of 12 LEDs connected to GP0.

from machine import Pin
import neopixel
import time

# ------------------------------
# Configuration
# ------------------------------

LED_PIN = 8          # GPIO connected to DIN
NUM_LEDS = 12        # Number of LEDs, 0-11 is used, not 1-12

# Create the NeoPixel object
np = neopixel.NeoPixel(Pin(LED_PIN), NUM_LEDS)

# ------------------------------
# Helper Functions
# ------------------------------

def clear():
```

```
    """Turn all LEDs off."""
    for i in range(NUM_LEDS):
        np[i] = (0, 0, 0)
    np.write()

def chase(color, delay=1):
    """Move one LED along the strip."""
    clear()

    for i in range(NUM_LEDS):
        clear()
        np[i] = color
        np.write()
        time.sleep(delay)

# ------------------------------
# Main Program
# ------------------------------

while True:

    # Chase effect
    chase((255, 0, 0), 0.08)
    chase((0, 255, 0), 0.08)
    chase((0, 0, 255), 0.08)
```

Here is a short video showing the sequence.

neopixel-chase-led-example-001.mp4

It's a nice effect to see the LEDs chasing like this.

Knight Rider Lights

Who remembers these?

knight-rider-001.mp4

Well I do :) and we can replicate this effect with our Pico and NeoPixel LEDs. The code for this is almost identical to the previous project where we have the running LEDs.

The major change is the addition of this part:

| download

```
    # Backward direction
    for i in range(NUM_LEDS - 2, 0, -1):
        clear()
        np[i] = color
        np.write()
        time.sleep(delay)
```

In our previous project, the LEDs ran forward, then cycled colour, in this project we are not cycling the colour and the additional code runs the LEDs in reverse.

Here is the full code, explanation for how this works is at the end of this wiki page.

| download

```
# WS2812 / NeoPixel Example
# Raspberry Pi Pico + MicroPython
#
# Controls a strip of 12 LEDs connected to GP8.
```

```
from machine import Pin
import neopixel
import time

# --------------------------------
# Configuration
# --------------------------------

LED_PIN = 8          # GPIO connected to DIN
NUM_LEDS = 12        # Number of LEDs, 0-11 is used, not 1-12

# Create the NeoPixel object
np = neopixel.NeoPixel(Pin(LED_PIN), NUM_LEDS)

# --------------------------------
# Helper Functions
# --------------------------------

def clear():
    """Turn all LEDs off."""
    for i in range(NUM_LEDS):
        np[i] = (0, 0, 0)
    np.write()

def chase_bounce(color, delay=0.1):
    """Move one LED up and down the strip."""
    # Forward direction
    for i in range(NUM_LEDS):
        clear()
        np[i] = color
        np.write()
        time.sleep(delay)

    # Backward direction
    for i in range(NUM_LEDS - 2, 0, -1):
        clear()
        np[i] = color
        np.write()
        time.sleep(delay)

# --------------------------------
# Main Program
# --------------------------------

while True:

    # Chase effect
    chase_bounce((255, 0, 0), 0.08)
```

So, what does this extra code do? and how?

Here is an explanation of the reverse code block in general.

| download

```
# Backward direction (move the lit LED from right to left)
# range(NUM_LEDS - 2, 0, -1) generates:
# For 12 LEDs → 10, 9, 8, 7, 6, 5, 4, 3, 2, 1
# - Start at LED 10 (second-to-last)
# - Stop at LED 1 (LED 0 is skipped to avoid repeating the end LED)
# - Step backwards by -1 each time
for i in range(NUM_LEDS - 2, 0, -1):

    clear()              # Turn all LEDs off before lighting the next one

    np[i] = color        # Light the current LED at position i

    np.write()           # Send the updated LED data to the strip
```

```
    time.sleep(delay)    # Pause so the movement is visible
```

And this is what the line that makes the count to backwards actually does.

| download

```
range(NUM_LEDS - 2, 0, -1)
#Breakdown for NUM_LEDS = 12:

Start: NUM_LEDS - 2 → 10
#(LED 11 was already lit in the forward chase)

Stop: 0
#(but Python stops before 0, so last value is 1)

Step: -1
#(count backwards)
```

For the reverse sequence, we don't need to worry about the first (0) or last (11) LEDs as these are already set by the forward sequence.

That's it! Oh, and don't worry, I won't be fitting these to my car :)
