



Adafruit NeoPixel Überguide

Created by Phillip Burgess



<https://learn.adafruit.com/adafruit-neopixel-uberguide>

Last updated on 2023-05-18 07:06:51 PM EDT

Table of Contents

The Magic of NeoPixels	5
<ul style="list-style-type: none">• Important Things to Know About NeoPixels in General	
Form Factors	7
NeoPixel Strips and Strands	7
<ul style="list-style-type: none">• RGB NeoPixel Strips• Mini Skinny RGB NeoPixel Strips• Side-Light NeoPixel Strips• RGBW NeoPixel Strips• “Neon-Like” NeoPixel Flex Strip• Ultraviolet NeoPixel Strips• NeoPixel Strands• Finer Details About NeoPixel Strips	
NeoPixel Rings	18
<ul style="list-style-type: none">• NeoPixel Ring Product Selector• Finer Details About NeoPixel Rings	
NeoPixel Matrices	21
<ul style="list-style-type: none">• Rigid 8x8 NeoPixel Matrices• NeoPixel Matrix Product Selector• Flexible NeoPixel Matrices• Finer Details About NeoPixel Matrices• We also have a few special-purpose matrices on the NeoPixel Shields page!	
NeoPixel Shields	25
<ul style="list-style-type: none">• NeoPixel Shield for Arduino• NeoPixel Shield Product Selector• NeoPixel FeatherWing• Pimoroni Unicorn Hat• Particle/Spark NeoPixel Ring Kit	
Other NeoPixel Shapes	29
<ul style="list-style-type: none">• NeoPixel Stick• NeoPixel Stick Product Selector• NeoPixel Jewels• NeoPixel Jewel Product Selector• 1/4 60 NeoPixel Ring• NeoPixel Ring Product Selector• Side Light NeoPixel LED PCB Bar	
Individual NeoPixels	31
<ul style="list-style-type: none">• Integrated NeoPixel Products• Flora RGB Smart NeoPixels• Breadboard-Friendly RGB Smart NeoPixels• NeoPixel Mini PCB• Discrete NeoPixel Products• Through-Hole NeoPixels• SMT NeoPixels• WS2811 Driver IC	

Basic Connections	39
Best Practices	42
<ul style="list-style-type: none">• Improper use can damage your NeoPixels. Before diving in, be aware of the following:• NeoPixels are not intended for long-term permanent use such as room lighting...but here are some tips to make the most of it:	
Powering NeoPixels	45
<ul style="list-style-type: none">• Estimating Power Requirements• Giant Power Supplies• Distributing Power• Driving 5V NeoPixels from 3.3V Microcontrollers	
Logic Level Shifting	52
Software	55
Arduino Library Installation	55
<ul style="list-style-type: none">• Install Adafruit_NeoPixel via Library Manager• Manually Install Adafruit_NeoPixel Library• A Simple Code Example: strandtest	
Arduino Library Use	58
<ul style="list-style-type: none">• HSV (Hue-Saturation-Value) Colors...• ...and Gamma Correction• Help!• Pixels Gobble RAM• Arduino Library Best Practices	
NeoMatrix Library	66
<ul style="list-style-type: none">• Adafruit_GFX Primer• Layouts• Tiled Matrices• Other Layouts• RAM Again• Gamma Correction	
Advanced Coding	73
<ul style="list-style-type: none">• FastLED Library• FAQ and Further Programming Insights• DMA NeoPixels for ARM Cortex-M0 Boards• Third-Party Libraries• Writing Your Own Library	
Python & CircuitPython	78
<ul style="list-style-type: none">• CircuitPython Microcontroller Wiring• Python Computer Wiring• CircuitPython Installation of NeoPixel Library• Python Installation of NeoPixel Library• CircuitPython & Python Usage• Full Example Code• RGB LED Colors	
Python Docs	84

MakeCode

84

- [Guide Link: NeoPixels with MakeCode](#)

Downloads

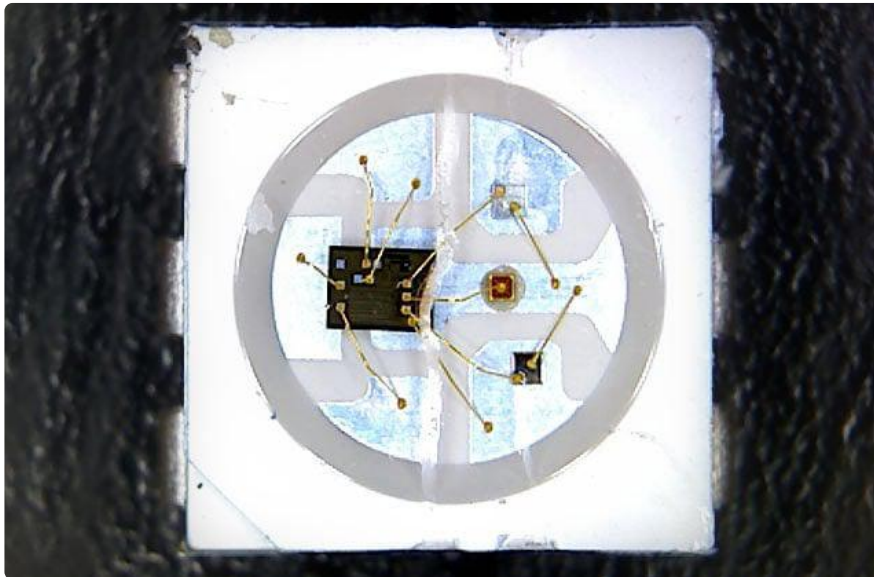
84

- [NeoPixel 12-LED Ring](#)
- [NeoPixel 16-LED Ring](#)
- [NeoPixel 24-LED Ring](#)
- [NeoPixel 1/4 60-LED Ring](#)
- [NeoPixel Jewel](#)
- [Breadboard Friendly NeoPixel Breakout](#)
- [NeoPixel NeoMatrix 8x8](#)
- [NeoPixel Arduino Shield](#)
- [NeoPixel 8 Stick](#)

The Magic of NeoPixels

Incorporating scads of LEDs into an electronic project used to be a hairy prospect, a veritable rat's nest of wires and code. The arrival of dedicated LED driver chips brought welcome relief, offloading grunt work from the microcontroller and allowing one to focus on the application. Much simpler, but still not “Christmas light” simple.

The WS2812 Integrated Light Source — or NeoPixel in Adafruit parlance — is the latest advance in the quest for a simple, scalable and affordable full-color LED. Red, green and blue LEDs are integrated alongside a driver chip into a tiny surface-mount package controlled through a single wire. They can be used individually, chained into longer strings or assembled into still more interesting form-factors.



We know you're eager to get started...but If this is your first time using NeoPixels, please at least read the “Best Practices” page before connecting anything!

Important Things to Know About NeoPixels in General

- Not all addressable LEDs are NeoPixels. “NeoPixel” is Adafruit’s brand for individually-addressable RGB color pixels and strips based on the WS2812, WS2811 and SK6812 LED/drivers, using a single-wire control protocol. Other LED products we carry — DotStars, WS2801 pixels, LPD8806 and “analog” strips — use different methodologies (and have their own tutorials). When seeking technical support in the forums, a solution can be found more quickly if the correct LED type is mentioned, i.e. avoid calling DotStars “NeoPixels.” Similar, but different!

- NeoPixels don't just light up on their own; they require a microcontroller (such as Arduino) and some programming. We provide some sample code to get you started. To create your own effects and animation, you'll need some programming practice. If this is a new experience, work through some of the beginning Arduino tutorials to get a feel for the language.
- NeoPixels aren't the answer for every project. The control signal has very strict timing requirements, and some development boards (such as Netduino or Raspberry Pi) can't reliably achieve this in every situation. This is why we continue to offer other LED types; some are more adaptable to certain tasks.

Can I use NeoPixels for POV (persistence of vision) displays?

Not recommended. The refresh rate is relatively low (about 400 Hz to 2 KHz), and in fast motion may appear "speckled." They look fine in stationary displays though (signs, decorations, jewelry, etc.). For POV use, [DotStar strips \(\)](#) will look much better (they have much higher refresh rate).

How about for light painting?

Definitely! The slower movement used for photographic light painting doesn't call attention to the limited refresh rate; [the results look great \(\)](#), especially with a light diffuser.

Is there a limit to the number of NeoPixels in a chain?

There's no inherent limit in the maximum length of a NeoPixel chain, but eventually you'll encounter any of various practical limits:

1. RAM: NeoPixels require some RAM from the host microcontroller; more pixels = more RAM. It's only a few bytes each, but as most microcontrollers are pretty resource-constrained, this becomes a very real consideration for large projects.
2. Power: each NeoPixel draws a little bit of current; more pixels = more power. Power supplies likewise have some upper limit.
3. Time: NeoPixels process data from the host microcontroller at a fixed data rate; more pixels = more time and lower animation frame rates.

Form Factors



NeoPixel products are available in a zillion form factors...from individual tiny pixels to huge matrices...plus strips, rings and everything in-between. The same software drives all of them; they're all "NeoPixels."

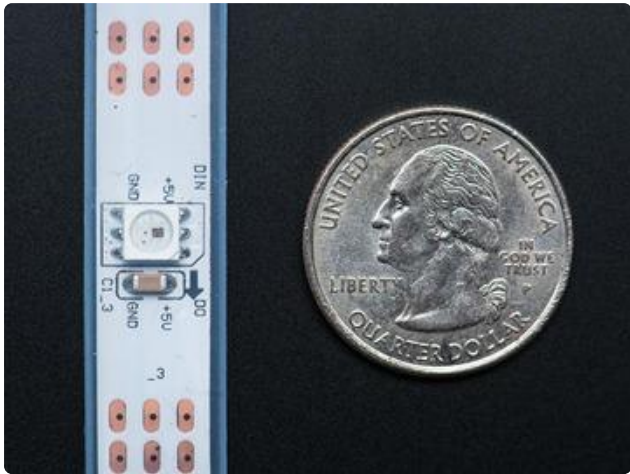
Pick a category from the left column for product links and tips & tricks specific to each type of NeoPixel.

NeoPixel Strips and Strands

The most popular type of NeoPixels are these flexible LED strips...they can be cut to length and fit into all manner of things. We've got over a dozen varieties! Two vital things to be aware of:

- Though strips are described as "flexible," they do not tolerate continuous and repeated bending. "Formable" might be a better word. A typical application is architecture, where they can be curved around columns and then stay put. Repeated flexing (as on costumes) will soon crack the solder connections. For wearable use, either affix shorter segments to a semi-rigid base (e.g. a hat, BMX armor, etc.), or use the individual sewable NeoPixels shown later.
- Watch your power draw. Though each pixel only needs a little current, it adds up fast...NeoPixel strips are so simple to use, one can quickly get carried away! We'll explain more on the "Powering NeoPixels" page.

RGB NeoPixel Strips



NeoPixel Digital RGB LED Weatherproof Strip is available in three different “densities”: 30, 60 and 144 LEDs per meter, on a white or black backing strip.

30 LEDs per meter, white strip (<http://adafru.it/1376>)

30 LEDs per meter, black strip (<http://adafru.it/1460>)

60 LEDs per meter, white strip (<http://adafru.it/1138>)

60 LEDs per meter, black strip (<http://adafru.it/1461>)

144 LEDs per meter, white strip (<http://adafru.it/1507>)

144 LEDs per meter, black strip (<http://adafru.it/1506>)

The approximate peak power use (all LEDs on at maximum brightness) per meter is:

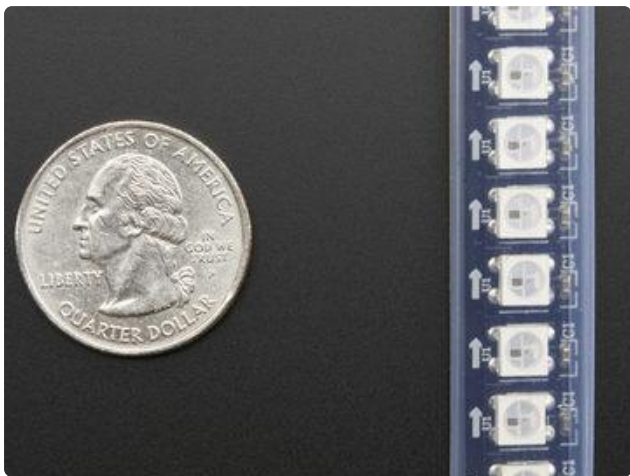
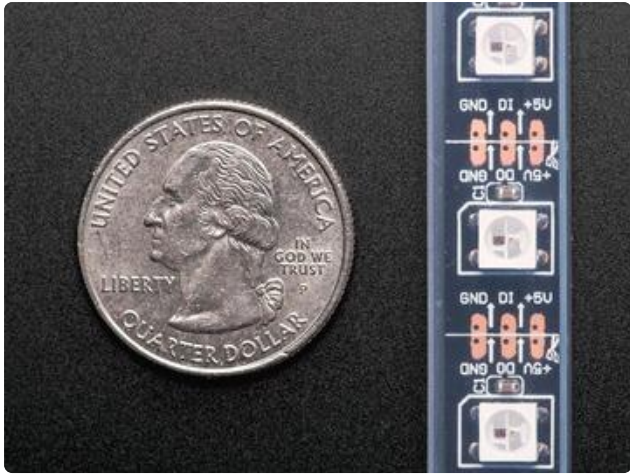
30 LEDs: 9 Watts (about 1.8 Amps at 5 Volts).

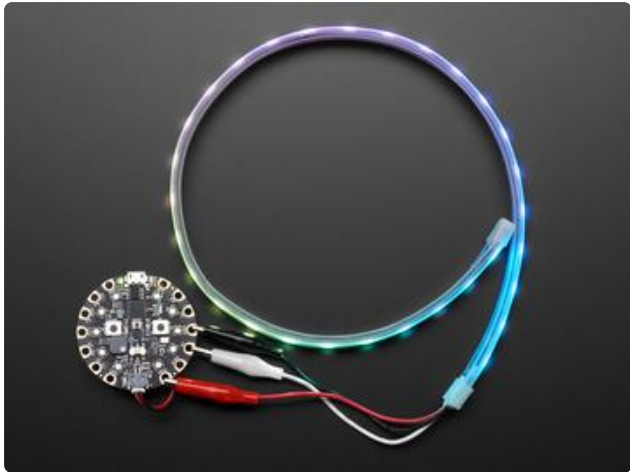
60 LEDs: 18 Watts (about 3.6 Amps at 5 Volts).

144 LEDs : 43 watts (8.6 Amps at 5 Volts).

Mixed colors and lower brightness settings will use proportionally less power.

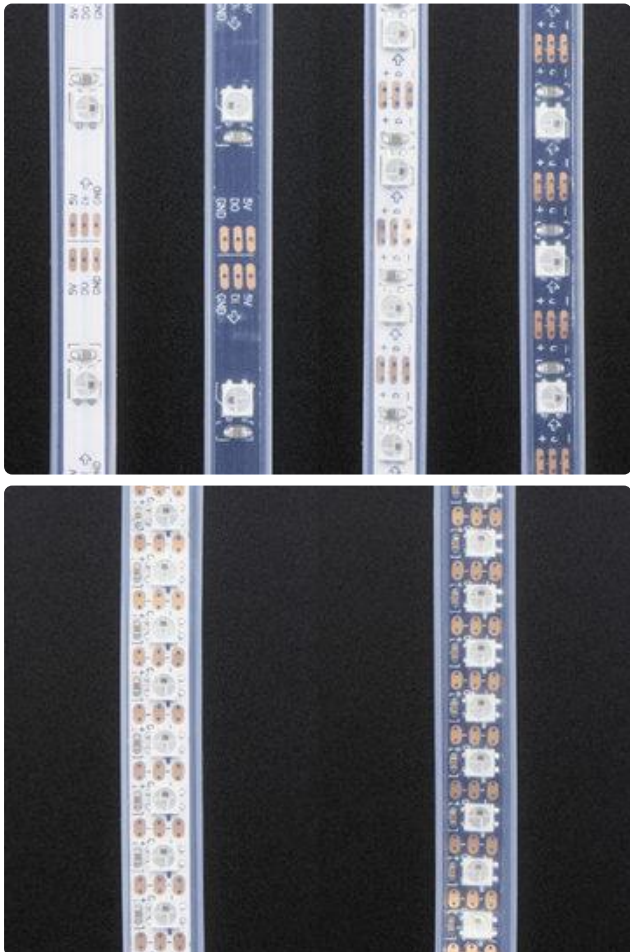






For those using [Circuit Playground Express](#) () or just needing a “no soldering” option (as in most classrooms), we have a special [half-meter, 30-LED NeoPixel strip with alligator clips attached](#) (). Easy!

Mini Skinny RGB NeoPixel Strips



Mini Skinny NeoPixel strips are about half the width of classic NeoPixel strips. They’re available in two densities and backing colors:

[30 LEDs per meter, white strip](#) (<http://adafru.it/2949>)

[30 LEDs per meter, black strip](#) (<http://adafru.it/2954>)

[60 LEDs per meter, white strip](#) ()

[60 LEDs per meter, black strip](#) (<http://adafru.it/2964>)

[144 LEDs per meter, white strip](#) ()

[144 LEDs per meter, black strip](#) ()

30 and 60 LED/meter strips are 7.5 mm wide, or 5 mm if you remove the strip from the casing (vs 12.5 mm / 10 mm for classic strips). The high-density 144/m strips are about 10 mm wide, or 7.5mm with the casing removed.

Power requirements are similar to standard-width NeoPixel strips as described above.

Side-Light NeoPixel Strips



Side-Light NeoPixel strips have the interesting property of illuminating next to the strip rather than over it. They're not quite as bright as regular NeoPixels, but may have interesting uses in tight spaces or for edge-lit acrylic. These strips are available in three densities on black flex-strip:

60 LEDs, 1 meter black strip ([link](#))

90 LEDs, 1 meter black strip ([link](#))

120 LEDs, 1 meter black strip ([link](#))

RGBW NeoPixel Strips

A recent addition is RGBW NeoPixel strips. These add a fourth LED element — pure white — which is more “true” and pleasing to the eye than white mixed from red+green+blue. Like the RGB strips, they’re available in different pixel densities and backing strip colors.

30 RGBW LEDs per meter, white strip (<http://adafru.it/2832>)

30 RGBW LEDs per meter, black strip (<http://adafru.it/2824>)

60 RGBW LEDs per meter, white strip (<http://adafru.it/2842>)

60 RGBW LEDs per meter, black strip (<http://adafru.it/2837>)

144 RGBW LEDs per meter, white strip (<http://adafru.it/2847>)

144 RGBW LEDs per meter, black strip (<http://adafru.it/2848>)

With a fourth LED per pixel, these strips may potentially draw up to 33% more current than their RGB equivalents. The maximum brightest cases are (approximately):

30 RGBW LEDs: 12 Watts (2.4 Amps at 5 Volts)

60 RGBW LEDs: 24 Watts (4.8 Amps at 5 Volts)

144 RGBW LEDs: 57 Watts (11.5 Amps at 5 Volts)

Width is the same as “classic” NeoPixel strip...these are not the “skinny” size.



“Neon-Like” NeoPixel Flex Strip



This distinctive NeoPixel flex strip has a gorgeous diffused neon-like appearance thanks to its thick silicone casing.

This strip contains 60 LED along the meter but in groups of 3-LEDs-per-pixel. So basically, in your NeoPixel program, this looks like a 20-pixel-long strand.

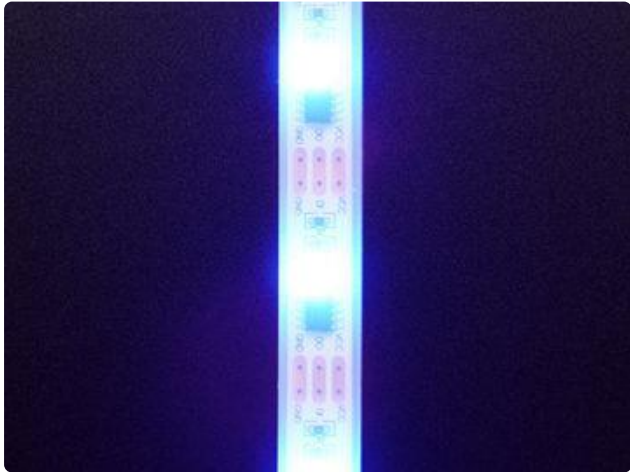
Unlike the other varieties of NeoPixel strip, this one needs to be powered from 9 Volts (minimum) to 12 Volts (ideal) DC.

[NeoPixel RGB Neon-like LED Flex Strip with Silicone Tube - 1 meter \(\)](#)



Ultraviolet NeoPixel Strips

A single-color ultraviolet variant of NeoPixel strip is available for special applications, currently one pixel density and backing color:



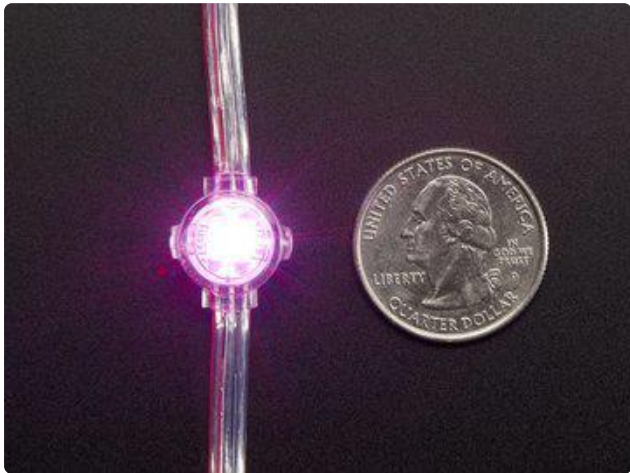
32 UV LEDs per meter, white strip ()

This can provide unusual effects when combined with UV-reactive paints, fluorescent laser-cut acrylic, etc.

The usual NeoPixel R, G and B channels translate to three individual UV diodes. So we recommend you pretty much just set all three channels to the same value, ranging from (0, 0, 0) to (255, 255, 255).

Same width as “classic” NeoPixel strip, and power draw at full brightness should be just a bit more than RGB 30/m: about 9.6W/meter (1.9A @ 5V).

NeoPixel Strands



Like NeoPixel strips, these “strands” are flexible and can be wrapped around things. The pixels are spaced further apart (2 or 4 inches) and each pixel is either sealed in its own tiny plastic capsule or epoxy-dipped, making them weatherproof (but not rated for continuous submersion). There are 20 pixels in a strand (except for the “Soft Flexible” variety, with 50).

NeoPixel Strand — 20 LED 4" Pitch ([link](#))

NeoPixel Strand — 20 LEDs at 2" Pitch ([link](#))

Slim NeoPixel Strand — 20 LEDs at 2" Pitch ([link](#))

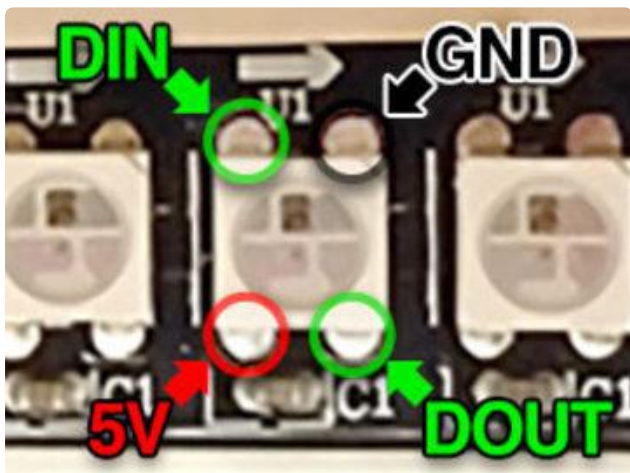
Soft Flexible NeoPixel Strand — 50 LEDs at 4" Pitch ([link](#))



Finer Details About NeoPixel Strips

- NeoPixel strips often have extra wires so they can connect to both a microcontroller and power source, but these will always correspond to one of the same three signals: +5V, ground or data. If not using the extra wires, you can clip off any exposed tips and/or insulate with some heat-shrink tube.
- 144 pixel/m NeoPixel strips and 32/m UV strips are sold in one meter lengths. Each meter is a separate strip with end connectors. Longer contiguous lengths are not offered in 144 pixels/m RGB, RGBW or 32/m UV.
- 30 and 60 pixel/m NeoPixel strips are sold in one meter multiples. Orders for multiple meters will be a single contiguous strip, up to a limit: 4 meters for 60 pixels/m strip, 5 meters for 30 pixels/m.
- For 30 and 60 pixels/meter strips, if purchasing less than a full reel (4 or 5 meters, respectively), the strip may or may not have 3-pin JST plugs soldered at one or both ends. These plugs are for factory testing and might be at either end — the plug does not always indicate the input end! Arrows printed on the strip show the actual data direction. You may need to solder your own wires or plug.

- The flex strips are enclosed in a weatherproof silicone sleeve, making them immune to rain and splashes, but are not recommended for continuous submersion. Early 144 pixel/meter strips were not weatherproof, but the current inventory now offers this feature.
- The silicone sleeve can be cut and removed for a slimmer profile, but this compromises the strip's weather resistance.
- Very few glues will adhere to the weatherproof silicone sleeve. Using zip ties for a “mechanical” bond is usually faster and easier. The only reliable glues we’ve found are Permatex 66B (or Permatex 80050) Clear RTV Silicone (not all silicone glues will work!) and Loctite Plastics Bonding System, a 2-part cyanoacrylate glue. Customers have reported excellent results with Permatex Ultra Grey Silicone Gasket Maker , Lexel® adhesive and Clear Gorilla Glue as well.
- However, do not use Permatex 66B silicone to seal the open end of a cut strip! Like many RTV silicones, 66B releases acetic acid when curing, which can destroy electronics. It’s fine on the outside of the strip, but not the inside. Use GE Silicone II for sealing strip ends, or good ol’ hot melt glue.
- 2-sided carpet tape provides a light grip on the silicone sleeve; something like a Post-It Note. Or you can try clear duct tape over the top.
- All LED strips are manufactured in 1/2 meter segments that are then joined into a longer strip. The pixel spacing across these joins is usually 2-3 millimeters different than the rest. This is not a manufacturing mistake, just physical reality.



Some batches of 144 pixel strip don't have pads between the LEDs. If you cut these into shorter sections, the only way to connect to them (except at the half-meter segments) is to carefully solder directly to the LED. The corner with the notch is the GND pin.

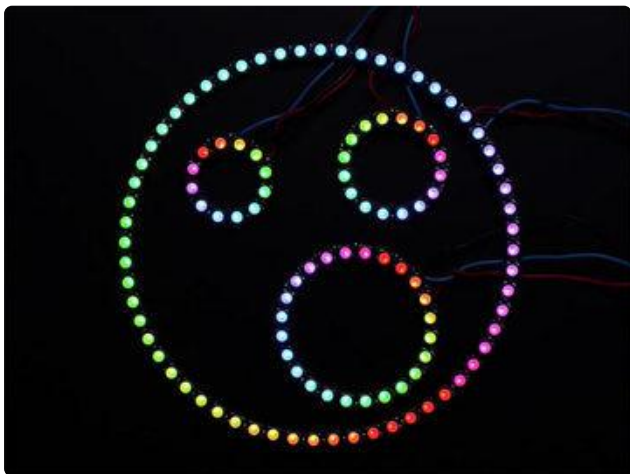
NeoPixel strips are just the start...we've got shapes too! Rings, grids, shields and more...

NeoPixel Rings

NeoPixel rings are circular rigid printed circuit boards festooned with NeoPixel LEDs. Originally designed for our [NeoPixel Goggles kit \(http://adafru.it/2221\)](http://adafru.it/2221), they proved so popular with other projects...timepieces, GPS wayfinders, jewelry, etc...that we now offer several sizes and varieties...

Rather than list a zillion different links, we have a single landing page for selecting among all the different NeoPixel ring products:

[NeoPixel Ring Product Selector \(http://adafru.it/3042\)](http://adafru.it/3042)



NeoPixel rings are offered in 12, 16, 24 and 60 pixel varieties.



60-pixel rings are actually sold as 15-pixel quarters. For a full circle, you'll need to purchase 4 and solder them together. Or you might find creative ideas for individual arcs!

Number of Pixels	Outer Diameter	Inner Diameter
12	37 mm / 1.5"	23 mm / 1"

16	44.5 mm / 1.75"	31.75 mm / 1.25"
24	65.5 mm / 2.6"	52.5 mm / 2.05"
60 (4x 15-pixel arcs)	158 mm / 6.2"	145 mm / 5.7"

All rings are about 3.6 millimeters / 0.15" thick (1.6 mm for PCB, 2 mm for NeoPixels).



RGB NeoPixels are the most affordable and can produce millions of color combinations.



RGBW NeoPixels offer an eye-pleasing “true” white in addition to RGB. These are available in three different color temperatures:

Cool white: approximately 6000 Kelvin (K).

Neutral: approx 4500K.

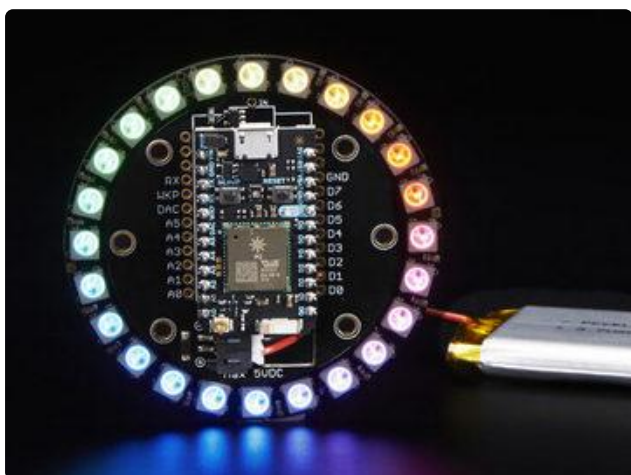
Warm white: approx. 3000K.

RGBW pixels incorporate a translucent diffuser layer to help mix and soften the light output. This makes them appear less intense than RGB pixels (which are “water clear”), but it’s really the same LEDs inside.



Finer Details About NeoPixel Rings

- When soldering wires to these rings, you need to be extra vigilant about solder blobs and short circuits. The spacing between components is very tight! It's often easiest to insert the wire from the front and solder on the back.
- If using alligator clips, we recommend first soldering short jumper wires to the ring inputs and connecting the clips to those, for similar reasons. (Some of our tutorials and product photos do show alligator clips directly connected to the rings, but we've had a lot of experience using them.)



There's also a [24-pixel RGB ring \(http://adafru.it/2268\)](http://adafru.it/2268) specifically designed for the Particle (formerly Spark) Photon development board.

This one's not "see-through" like the others — the space at the center provides a socket for the [Photon board \(http://adafru.it/2721\)](http://adafru.it/2721).

NeoPixel Matrices

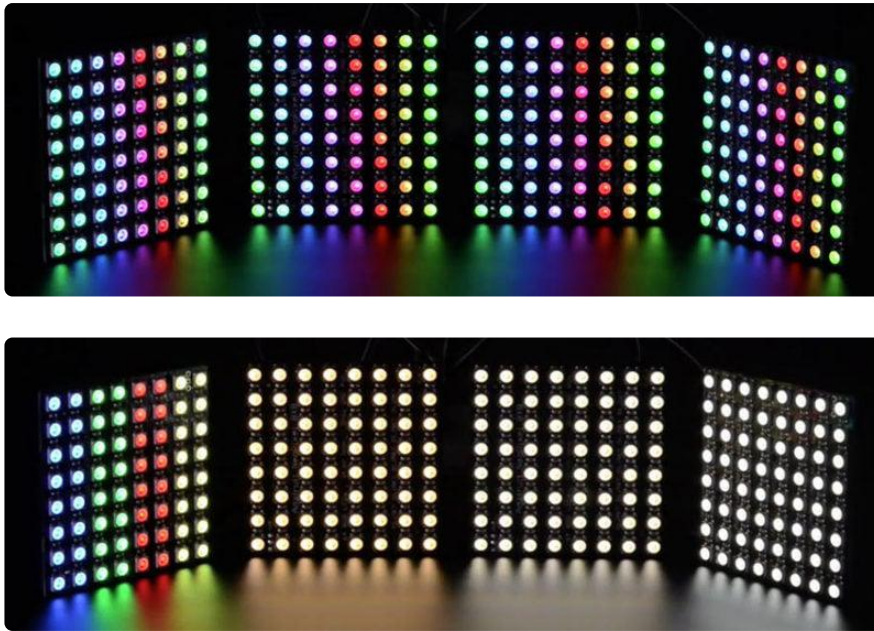
NeoPixel matrices are two-dimensional grids of NeoPixels, all controlled from a single microcontroller pin.

Rigid 8x8 NeoPixel Matrices

Like NeoPixel rings, these 64-pixel matrices are assembled on a rigid printed circuit board and are available in both RGB and RGBW varieties.

[NeoPixel Matrix Product Selector \(http://adafru.it/3052\)](http://adafru.it/3052)

All measure 71 millimeters (2.8 inches) square and about 3.6 mm thick. There are several mounting holes, and the DOUT pin allows multiple matrices to be linked in series.



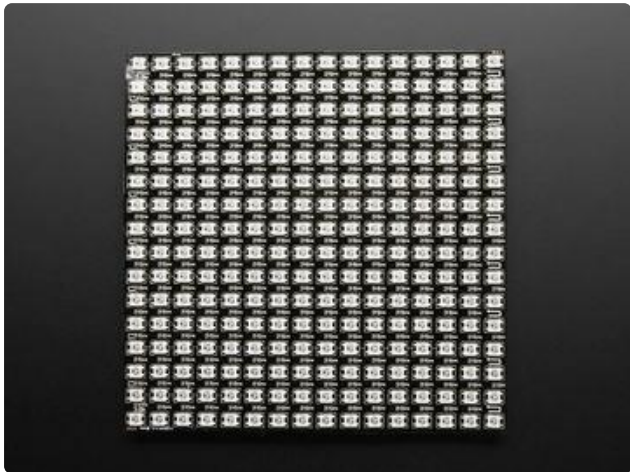
RGB NeoPixels are the most affordable and can produce millions of color combinations.

RGBW NeoPixels offer an eye-pleasing “true” white in addition to RGB. These are available in three different color temperatures:

- Cool white: approximately 6000 Kelvin (K).
- Neutral: approx 4500K.
- Warm white: approx. 3000K.

RGBW pixels incorporate a translucent diffuser layer to help mix and soften the light output. This makes them appear less intense than RGB pixels (which are “water clear”), but it’s really the same LEDs inside.

Flexible NeoPixel Matrices



Flexible NeoPixel matrices are available in three different sizes:

8x8 RGB pixels (<http://adafru.it/2612>)

16x16 RGB pixels (<http://adafru.it/2547>)

8x32 RGB pixels (<http://adafru.it/2294>)

Size	Dimensions	Total # of LEDs	Max Power Draw (approx)
------	------------	-----------------	-------------------------

8x8	80 mm / 3.15" square	64	19 Watts (3.8 Amps at 5 Volts)
16x16	160 mm / 6.3" square	256	77 Watts (15 Amps at 5 Volts)
8x32	320 mm x 80 mm / 12.6" x 3"	256	77 Watts (15 Amps at 5 Volts)

Flex matrices are about 2 millimeters (0.08 inches) thick.

Though called “flexible,” these matrices do not tolerate continuous and repeated bending. “Formable” might be a better word — they can be bent around a rigid or semi-rigid shape, like a hat. Repeated flexing (as on costumes) will soon crack the solder connections. (The videos on the product pages are to highlight just how flexible these matrices are, but this really is a “don’t try this at home” thing.)

Flex matrices are available with RGB pixels only; RGBW is not offered.

Finer Details About NeoPixel Matrices

As mentioned on the NeoPixel Strips page, keep power consumption in mind when working with NeoPixel matrices. With so many pixels at your disposal, it’s easy to get carried away.



If you need a size or shape of NeoPixel matrix that’s not offered here, it’s possible to create your own using sections of NeoPixel strip!

NeoPixel matrices don't enforce any particular "topology" — some may have rows of pixels arranged left-to-right, others may alternate left-to-right and right-to-left rows, or they could be installed in vertical columns instead. This will require some planning in your code. Our NeoMatrix library supports most matrix topologies.

We also have a few special-purpose matrices on the [NeoPixel Shields page](#)!

NeoPixel Shields

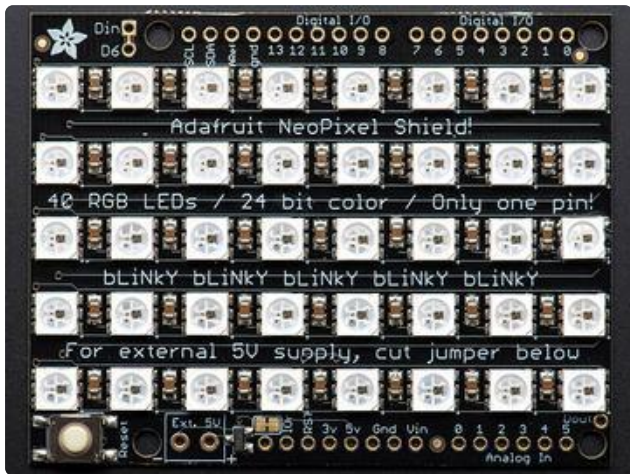
Though not all "Shields" in the strictly-speaking Arduino sense, a few NeoPixel products are designed to fit directly atop (or below) certain microcontroller boards...

NeoPixel Shield for Arduino

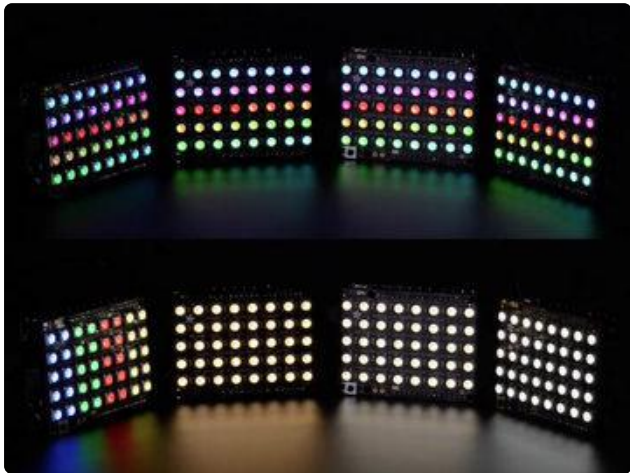


This 5x8 [NeoPixel Shield for Arduino](http://adafru.it/1430) (<http://adafru.it/1430>) fits neatly atop an Arduino Uno or compatible boards (5V logic recommended). Like many of our NeoPixel products, they're available in RGB and various RGBW pixel types:

[NeoPixel Shield Product Selector \(\)](#)

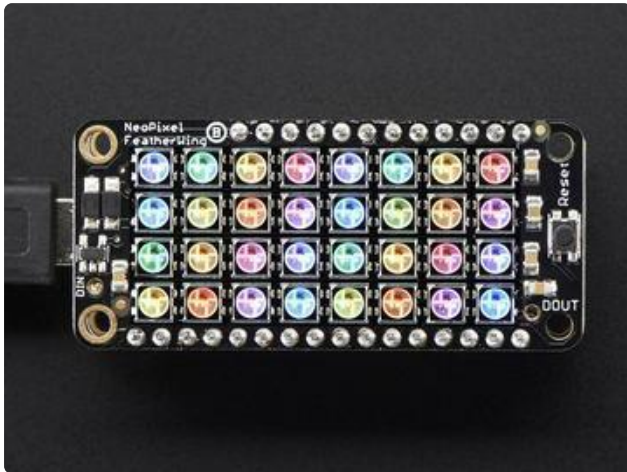


By default, the LEDs are powered from the Arduino's 5V pin. As long as you aren't lighting up all the pixels at full brightness that should be fine. Or power the shield with an external power supply by soldering the included terminal block.



The NeoPixels are controlled on digital pin 6, but with some deft wiring you could change this to any pin.

NeoPixel FeatherWing



Quite possibly The Cutest Thing in the History of Cute Little Things, the [NeoPixel FeatherWing](http://adafru.it/2945) (<http://adafru.it/2945>) is a 4x8 pixel matrix that fits perfectly atop any of our [Feather microcontroller boards](#) ().



The NeoPixels are normally controlled from digital pin 6, but pads on the bottom make this reassignable. In particular, the default pin for Feather Huzzah ESP8266 must be moved, try pin #15!

The NeoPixel Featherwing is RGB only; there's no RGBW version.

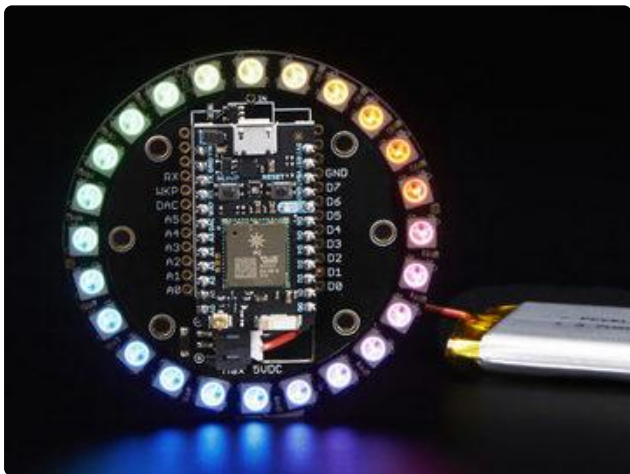
Pimoroni Unicorn Hat



The [Pimoroni Unicorn Hat \(http://adafru.it/2288\)](http://adafru.it/2288) is aptly named after a mythical animal — normally we'll say that NeoPixels don't work with the Raspberry Pi, but Pimoroni has worked up some [magical software \(\)](#) that makes this combination possible! It's an 8x8 RGB matrix that fits neatly atop the Raspberry Pi Model A+, B+ or Pi 2.

Due to the way Unicorn HAT works, you can't use your Pi's analog audio alongside it. If you see odd random colour patterns and flickering make sure analog audio is disabled.

Particle/Spark NeoPixel Ring Kit



Previously mentioned on the “Rings” page, but for posterity: this [24-pixel RGB ring \(http://adafru.it/2268\)](http://adafru.it/2268) is specifically designed for the Particle (formerly Spark) Photon development board.

Other NeoPixel Shapes

NeoPixel Stick



The simplest thing...a row of 8 NeoPixels along a rigid circuit board. These make great bargraph indicators!

Like our rings and matrices, NeoPixel sticks are available in RGB and RGBW varieties.

[NeoPixel Stick Product Selector \(http://adafru.it/3039\)](http://adafru.it/3039)



All measure 51.1 x 10.2 millimeters (2.0 x 0.4 inches).

NeoPixel Jewels



When you need more “punch” than a single NeoPixel can provide, these 7-pixel jewels provide a lot of light in a compact shape. Again, RGB and RGBW varieties are available.

[NeoPixel Jewel Product Selector \(http://adafru.it/3047\)](http://adafru.it/3047)



All measure 23 millimeters (0.9 inches) in diameter.

1/4 60 NeoPixel Ring



Though originally designed to be used in groups of four to complete a 60 NeoPixel Ring, the individual 15-pixel quarter rings can also be used to solve interesting design problems! RGB and RGBW are available.

[NeoPixel Ring Product Selector \(http://adafru.it/3042\)](http://adafru.it/3042)

Side Light NeoPixel LED PCB Bar



A half-meter rigid PCB tightly packed with 60 side-light NeoPixels. This is a strange animal but might be just the thing for compact light-painting projects or edge-lit signage. What might you make of it?

[Side Light NeoPixel LED PCB Bar - 60 LEDs - 120 LED/meter - 500mm Long \(\)](#)

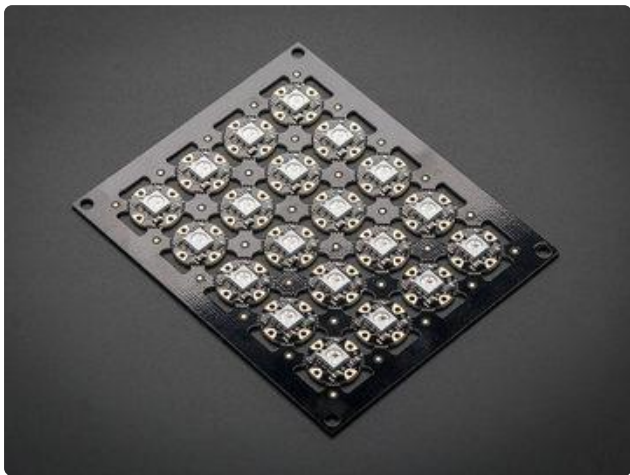
Individual NeoPixels

If you need just a small number of pixels...or if ready-made shapes and strips don't quite provide what you're after...individual NeoPixels provide the most control over placement and quantity.

Integrated NeoPixel Products

Some individual NeoPixel products come ready to use, with a small PCB holding the LED, a decoupling capacitor for power, and points for connecting wires.

Flora RGB Smart NeoPixels



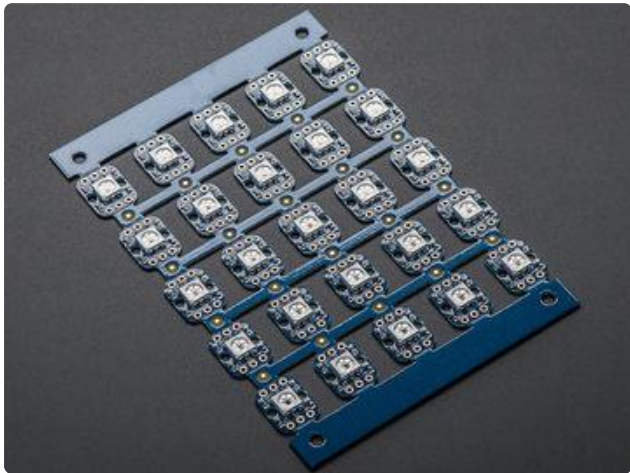
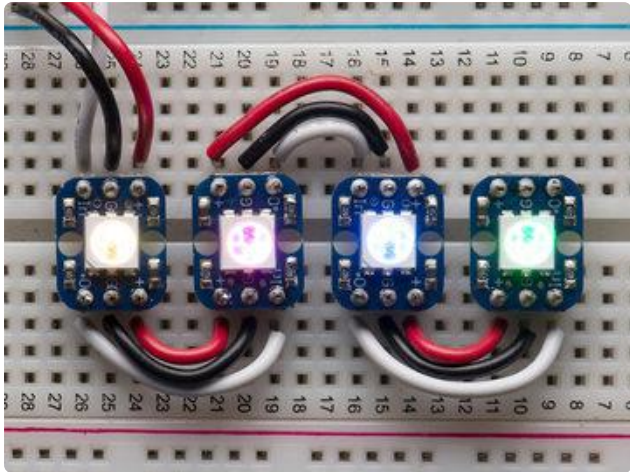
The original Adafruit NeoPixel form factor! Flora RGB Smart NeoPixels were designed for wearable projects using conductive thread, but can also be soldered normally with wires. These are available in two formats:

[Pack of 4 \(http://adafru.it/1260\)](http://adafru.it/1260), ready to use as-is.

[Sheet of 20 \(http://adafru.it/1559\)](http://adafru.it/1559), cut them off the sheet as you need them and save a little money.

Flora RGB Smart Pixels measure about 12.5 millimeters (0.5 inches) in diameter. These are RGB only; there's no RGBW version.

Breadboard-Friendly RGB Smart NeoPixels



These are similar to the sewable Flora NeoPixels, but with a pin arrangement that (with the addition of headers) fits neatly into a breadboard for prototyping. Also available in two formats:

[Pack of 4 \(http://adafru.it/1312\)](http://adafru.it/1312), ready to use as-is.

[Sheet of 25 \(http://adafru.it/1558\)](http://adafru.it/1558), cut off as needed.

For both types, [headers \(http://adafru.it/392\)](http://adafru.it/392) are optional and not included.

Breadboard-Friendly NeoPixels measure 10.2 x 12.7 millimeters (0.4 x 0.5 inches) and are RGB only; there's no RGBW version.

NeoPixel Mini PCB



NeoPixel Mini Button PCBs are the smallest ready-to-use NeoPixel format — each on its own tiny board with an input resistor and decoupling capacitor.

[NeoPixel Mini Button PCB \(\)](#) — RGB version, sold in packs of 5

[NeoPixel RGBW Mini Button PCB \(\)](#) — RGBW, sold in packs of 10

These have no mounting holes or soldering vias...wires must be soldered directly to pads on the back of the PCB.

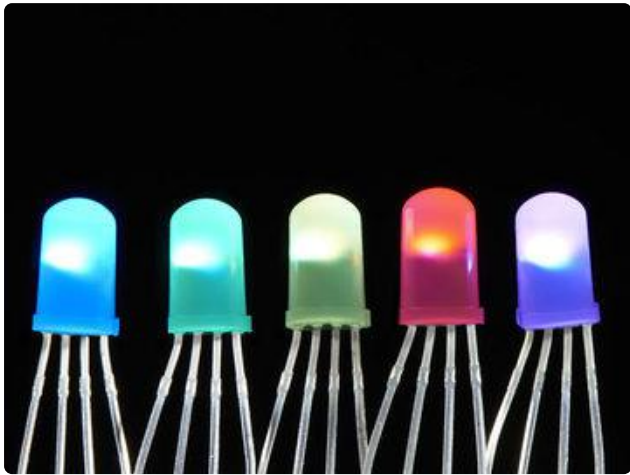
Each is about 10 millimeters (0.3 inches) in diameter.

Discrete NeoPixel Products

For advanced users needing fully customized designs, discrete NeoPixel components are available. You'll need to provide your own PCB and (depending on the pixel type) surface-mount soldering skill.

It's very strongly recommended that each NeoPixel have an accompanying 0.1 μ F capacitor between +V and ground. This prevents communication problems due to brownout of the on-pixel driver logic. It's occasionally sufficient to have one capacitor between pairs of pixels; some of our NeoPixel rings work that way.

Through-Hole NeoPixels



Discrete Through-hole NeoPixels are available in two sizes:

8mm Diffused (<http://adafru.it/1734>) – pack of 5.

5mm Diffused (<http://adafru.it/1938>) – pack of 5.

5mm Clear (<http://adafru.it/1837>) have been discontinued, but the product page is still available if you require pinout information.

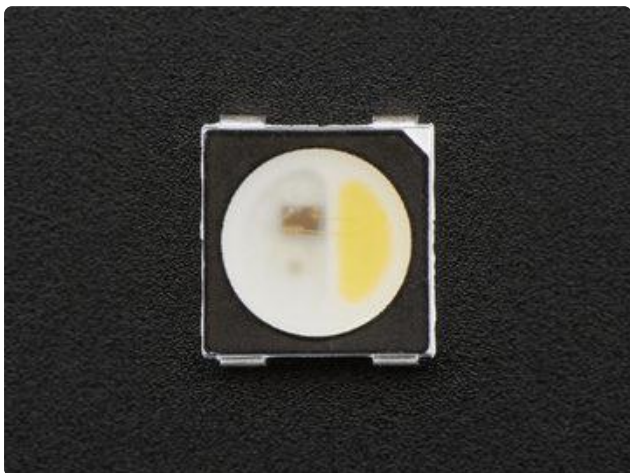
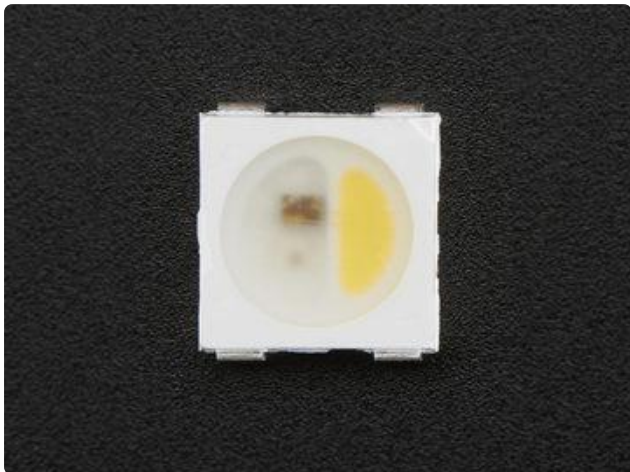
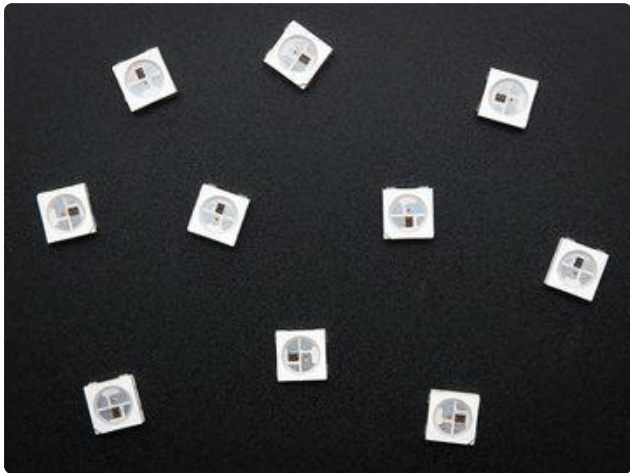
Through-hole NeoPixels are RGB only; there's no RGBW version. Use of a

0.1 μ F capacitor (<http://adafru.it/753>)

between + and ground on each pixel is strongly encouraged.

SMT NeoPixels

Surface-mount “5050” (5 millimeter square) NeoPixels are available in many varieties:



5050 RGB LED (<http://adafru.it/1655>) – pack of 10.

RGBW NeoPixel – Cool White – white case (<http://adafru.it/2759>) – pack of 10.

RGBW NeoPixel – Neutral White – white case (<http://adafru.it/2758>) – pack of 10.

RGBW NeoPixel – Warm White – white case (<http://adafru.it/2757>) – pack of 10.

RGBW NeoPixel – Cool White – black case (<http://adafru.it/2762>) – pack of 10.

RGBW NeoPixel – Neutral White – black case (<http://adafru.it/2761>) – pack of 10.

RGBW NeoPixel – Warm White – black case (<http://adafru.it/2760>) – pack of 10.

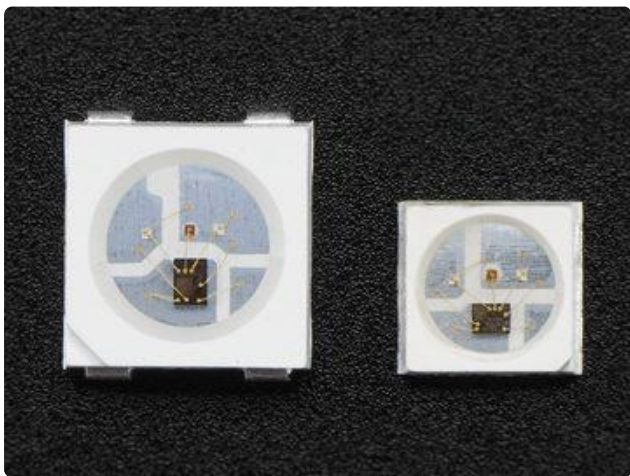
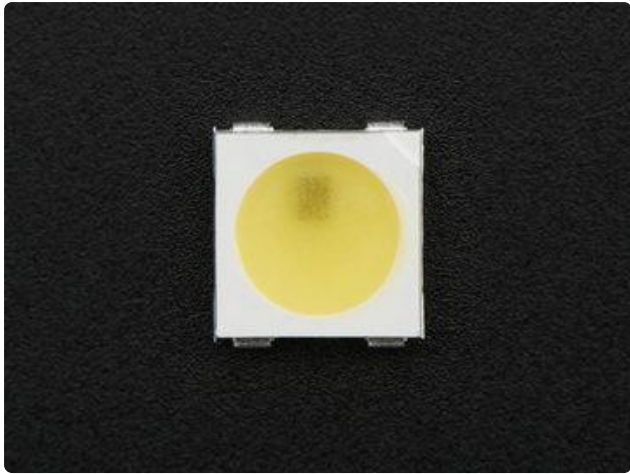
NeoPixel – Cool White (<http://adafru.it/2375>) (3X white, no RGB) – pack of 10.

NeoPixel – Warm White (<http://adafru.it/2376>) (3X white, no RGB) – pack of 10.

All measure 5 millimeters square. Adding a 0.1 μ F capacitor between + and ground is recommended for each pixel, positioned close by, though often one cap per two pixels will suffice.

The white- and black-cased pixels are functionally identical; this is purely an aesthetic choice for your design.

“Cool white” measures approximately 6000 Kelvin. “Neutral white” is approx. 4500K. “Warm White” is approx. 3000K.



Tiny surface-mount “3535” (3.5 millimeters square) NeoPixels are available in two RGB versions; no RGBW is available.

NeoPixel Mini 3535 RGB – white case (<http://adafru.it/2659>) – pack of 10.

NeoPixel Mini 3535 RGB – black case (<http://adafru.it/2686>) – pack of 10.

Decoupling capacitor recommended, as described above. As with the “5050” NeoPixels, white- and black-cased versions are functionally identical, this is an aesthetic design option.

WS2811 Driver IC



The NeoPixel driver logic is available separately (<http://adafru.it/1378>) from the LEDs, allowing power-users to create extremely customized designs...perhaps using other LED colors, or combined with [power MOSFETs](http://adafru.it/355) (<http://adafru.it/355>) to control high-current LEDs or “analog” RGB LED strips () using the NeoPixel protocol.

These require circuit design skills, custom PCBs and fine surface-mount soldering. A 0.1 uF decoupling capacitor is recommended for each chip.

Basic Connections

To get started, let's consider the example of an Arduino microcontroller connected to the computer's USB port (though it works with many other microcontroller boards as well). We'll elaborate on the finer points of powering NeoPixels later, but in general you'll usually be using a 5V DC power supply (e.g. "wall wart") or — for wearable projects — a 3.7 Volt lithium-polymer battery.

Identify the "input" end of your NeoPixel strip, pixel(s) or other device. On some, there will be a solder pad labeled "DIN" or "DI" (data input). Others will have an arrow showing the direction that data moves. The data input can originate from any digital pin on the Arduino, but all the example code is set up for digital pin 6 by default. The NeoPixel shield comes wired this way.

Some NeoPixel strips have extra wires so they can connect to both a microcontroller and power source, but these will always correspond to one of the same three inputs: +5V, ground or data. If not using the extra wires, you can clip off any exposed tips and/or insulate with some heat-shrink tube.

If there are four distinct wires, then it's probably a [DotStar \(\)](#) strip, or maybe [LPD8806 \(\)](#) (less common nowadays).

NeoPixels powered by 5V ideally require a 5V data signal. If using a 3.3V microcontroller, it's wise to use a logic level shifter such as a [74AHCT125 \(\)](#) or [74HCT245 \(<http://adafru.it/1779>\)](#). If you are powering your NeoPixels with 3.7v directly from a LiPoly cell, a 3.3v data signal is OK. See the "Logic Level Shifting" page for further information.

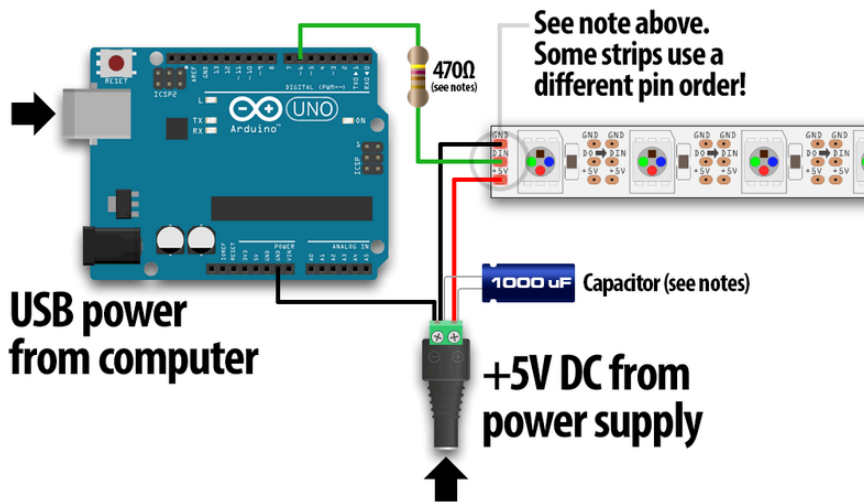
If using a Flora, Feather or other microcontroller board with an attached lithium-polymer battery: connect the +5V input on the strip to the pad labeled VBAT or BAT on the board, GND from the strip to any GND pad on the microcontroller board, and DIN to Flora pin D6. If the board doesn't have a pin #6, you'll need to modify the example code to change the pin number.

For other Arduino boards with a separate +5V DC power supply for the NeoPixels: connect the +5V input on the strip to the + (positive) terminal on the power supply (don't connect to the Arduino), DIN to digital pin 6 on the Arduino, and – (minus or GND) on the strip must connect to both the minus (–) terminal on the DC supply and a GND pin on the Arduino (there are usually several — any will do).

"DOUT" or "DO" (data out) at the end of a NeoPixel chain can be left unconnected. If adding more pixels later, data-out from one chain connects to data-in of the next.

The 144 pixel strips are so tightly packed, there's no room for labels other than -, + and the data direction arrows. Data is the third, un-labeled pad.

The order of the three pins can vary between different strip batches and densities. ALWAYS use the labels printed ON THE STRIP. Look closely, NEVER blindly follow a NeoPixel strip wiring diagram; it might be based on a different product!



When connecting NeoPixels to any LIVE power source or microcontroller, ALWAYS CONNECT GROUND (-) BEFORE ANYTHING ELSE. Conversely, disconnect ground last when separating.

On larger projects, you may need to add a capacitor (100 to 1000 μ F, 6.3V or higher) across the + and - terminals for more reliable operation. See the photo on the next page for an example.

With through-hole NeoPixels (5mm or 8mm round), add a 0.1 μ F capacitor between the + and - pins of EACH PIXEL. Individual pixels may misbehave without this "decoupling cap." Strips and boards have these caps built-in.

Today's NeoPixels are fairly robust, but if you have early ones or some "WS2812-compatible" parts, it's advisable to add a 300-500 Ohm resistor between the microcontroller's output pin and the NeoPixel input.

Can NeoPixels be powered directly from the Arduino's 5V pin?

Sometimes. The Arduino can continuously supply only about 500 milliamps to the 5V pin. Each NeoPixel can draw up to 60 milliamps at full brightness. So yes, you can skip the separate DC supply and power directly off the Arduino as long as just a few pixels are used, more if the colors and overall brightness are low. When in doubt, give the pixels a separate power supply.

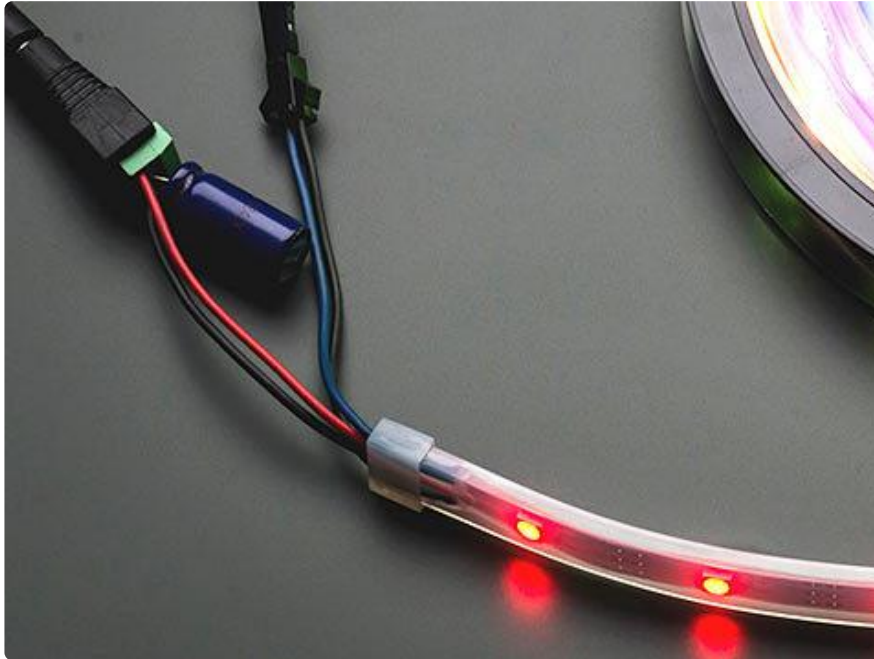
Can I power the NeoPixels and microcontroller from the same power supply?

Quite often yes — on the Arduino Uno, use the Vin pin next to GND, or the DC jack. Most well-designed boards from companies you recognize will do the right thing when both USB and external power are connected, and they incorporate a low-dropout regulator so a 5V power supply still provides very-nearly 5V to the board... close enough that everything works.

If using a super bare-bones board, perhaps an inexpensive clone or something that's less clever about handling different power inputs...you can use the 5V pin for input, but it's a smart idea in that case to not have that and USB connected at the same time; best to disconnect one or the other while iterating a design, always powering the board from just one source.

Best Practices

Improper use can damage your NeoPixels. Before diving in, be aware of the following:



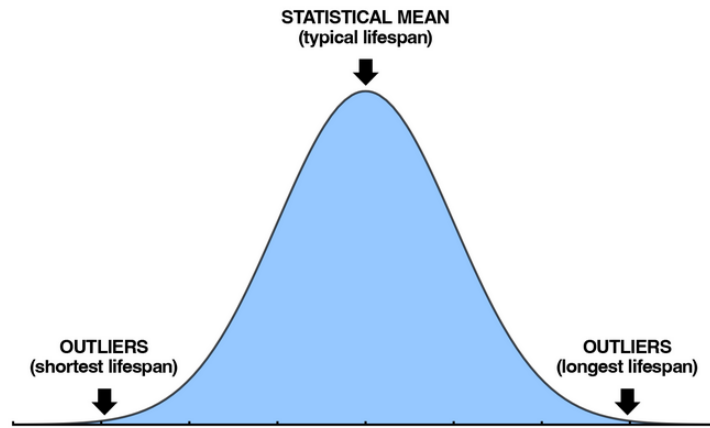
- Before connecting NeoPixels to any large power source (DC “wall wart” or even a large battery), add a capacitor (500–1000 μF at 6.3V or higher) across the + and – terminals as shown above. The capacitor buffers sudden changes in the current drawn by the strip.
- Place a [300 to 500 Ohm resistor](#) () between the Arduino data output pin and the input to the first NeoPixel. The resistor should be at the end of the wire closest to the NeoPixel(s), not the microcontroller. Some products already incorporate this resistor...if you’re not sure, add one...there’s no harm in doubling up! Also, newer NeoPixels are less picky about this. Nothing’s needed at the “out” end of a strip...you can leave the data out “floating.”
- Try to minimize the distance between the Arduino and first pixel, so the signal is clear. A meter or two is usually no problem. Much longer and things can become unreliable. Individual NeoPixels can act as repeaters for long runs.
- Avoid connecting NeoPixels to a live circuit. If you simply must, always connect ground first, then +5V, then data. Disconnect in the reverse order.
- If powering the pixels with a separate supply, apply power to the pixels before applying power to the microcontroller. Otherwise they’ll try to power “parasitically” through the data line, which could spell trouble for the microcontroller.

- Observe the same precautions as you would for any static-sensitive part; ground yourself before handling, etc.
- NeoPixels powered by 5v ideally need a 5V data signal. If using a 3.3V microcontroller you must use a logic level shifter such as a [74AHCT125 \(\)](#) or [74HCT245 \(<http://adafru.it/1779>\)](#). See the “Logic Level Shifting” page for more details. If you are powering your NeoPixels with 3.7v directly from a LiPoly cell, a 3.3v data signal is OK.
- Make sure that your connections are secure. Alligator clips do not make reliable connections to the tiny solder pads on NeoPixel rings. Better to solder a small pigtail wire to the ring and attach the alligator clips to that.
- If your microcontroller and NeoPixels are powered from two different sources (e.g. separate batteries for each), there must be a ground connection between the two.

Some of our projects don't make the above precautions...these are typically small battery-powered devices and power spikes aren't a big concern. Any project with a lot of pixels or a large power source should definitely include the power capacitor and data line resistor.

NeoPixels are not intended for long-term permanent use such as room lighting...but here are some tips to make the most of it:

It's not clear what the mean time between failures (MTBF) is for NeoPixels, but we've seen figures of $\geq 50,000$ hours quoted for similar addressable LEDs. On the surface this sounds phenomenal...that's over 5 years of 24/7 use...but it's important to understand the term “mean” here. You may have seen MTBF used in hard drive specifications, but it's also common in lighting and many other electronic components. Everything eventually has to give, and unfortunately there's just no One True Number for this. The mean time is a statistical average, a point on the bell curve where most parts will expend their usable lifetime. But any bell curve has outliers — in this case, a small number of seemingly immortal components in one direction, and those meeting an early demise in the other.



The “gotcha” with any sort of addressable LEDs is their “downstream” nature. If any pixel in a strand should fail, every pixel further down the line is also affected. Those subsequent pixels are not damaged by this...but until the failed pixel is replaced, they’re all but useless.

In a typical hobby project with a meter of LEDs or a couple of NeoPixel rings, and run for an hour or two here and there, you’ll probably never encounter this issue...most of us never do. But when projects increase in scope...hundreds or thousands of pixels running continuously...the probability of a short-lifespan outlier appearing becomes more likely, almost inevitable.

Nothing can eliminate this, but some best practices include:

- For large “wall power” installations, use high-quality power supplies with adjustment pots, configured for no higher than 5V DC output.
- Do what you can to ensure airflow. All LEDs emit a small amount of heat, and LED strips in open air seem to fare better than enclosed strips stewing in their own warmth.
- Dry environments are preferable to humidity, even with the weatherproof covering of most LED strips.
- Design for quick replaceability in sections...for example, identical 1-meter sections with JST connectors at both ends. If any section has a failure (knocking out everything downstream), swap in a known-good spare, then troubleshoot the failed section on its own, perhaps reflowing a single bad pixel and then setting this strip aside as a future spare.
- Rather than one long LED run, design for shorter parallel runs, as is done with the [Feather RP2040 SCORPIO \(\)](#), NeoPXL8 or Pixelblaze Output Expander. This will not reduce the number of failures at all, but limits the scope to a small distraction rather than the whole installation.

The first failed pixel might be a statistical outlier, but take a look at that graph and notice something: this is the start of a ramp-up. More will follow, and at shorter intervals. For large-scale architectural installations, big “art cars” and so forth, it’s not uncommon to hear after about one year that maintenance becomes near constant.

You may have seen elaborate retail window displays or theme park decorating that successfully uses large numbers of addressable LEDs. How is this done? Partly through the techniques listed above. Parallel runs localize any visual disturbance, modular replaceability allows quick overnight repairs, or a showpiece might simply be switched off until repaired. Mostly though: such setups are intended for limited runs: window displays might only be used for 2–3 months, and theme parks will redecorate annually for different seasons and events. We wish these things ran forever but large numbers just statistically work against us.

Powering NeoPixels

When connecting NeoPixels to any live power source or microcontroller, ALWAYS CONNECT GROUND (–) BEFORE ANYTHING ELSE. Conversely, disconnect ground last when separating.

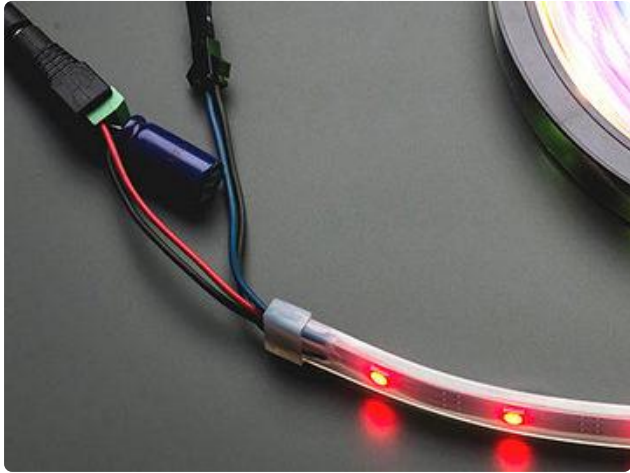
Adding a 300 to 500 Ohm resistor between your microcontroller's data pin and the data input on the first NeoPixel can help prevent voltage spikes that might otherwise damage your first pixel. Please add one between your micro and NeoPixels!

Neopixels do not light up on their own. Simply applying power will not make them light up. A microcontroller and some programming are required.

NeoPixels are usually described as “5 Volt devices,” but the reality is a little more nuanced than that.

Some (not all) NeoPixel products can work with slightly higher voltages. This depends on the additional support components around the chip, based on available space, cost and the most likely application. Refer to the specific product description page for guidance on acceptable voltage limits for each type. When in doubt, aim for 5 Volts.

Lower voltages are always acceptable, with the caveat that the LEDs may be slightly dimmer. There’s a limit below which the LED will fail to light, or will start to show the wrong color.



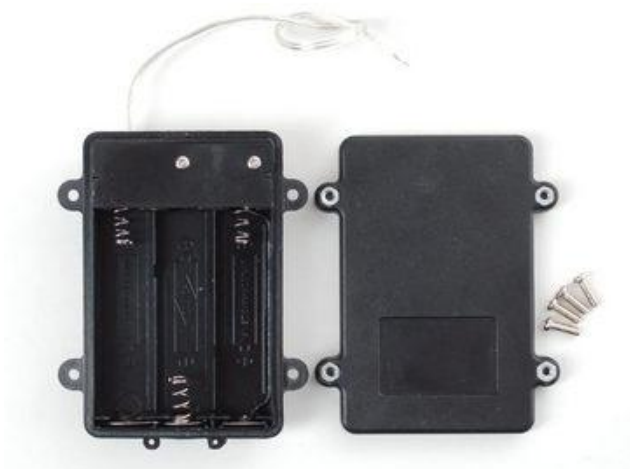
Before connecting a NeoPixel strip to ANY source of power, a large capacitor (500–1000 μF at 6.3 Volts or higher) across the + and – terminals provides a small power reservoir for abrupt changes in brightness that the power source might not otherwise handle — a common source of NeoPixel “glitching.”

For a polarized electrolytic capacitor as seen here, the negative (–) lead is usually indicated by a stripe and/or may be shorter than the + lead.

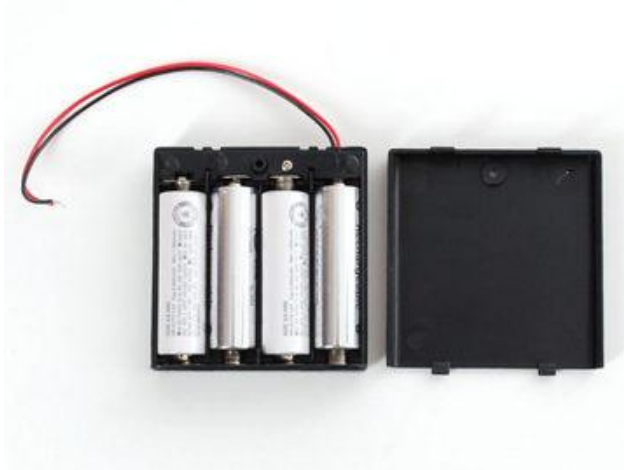
This is an item we don't stock, but they can be found from parts distributors like [Digi-Key](#) ().



For many wearable projects we recommend a [lithium-polymer battery](http://adafru.it/328) (<http://adafru.it/328>). These deliver 3.7 Volts — perfect for directly feeding low-power microcontrollers such as the Adafruit Flora, yet sufficient for a short length of NeoPixels.



Three alkaline cells (such as AA batteries) can be installed in a [battery holder](http://adafru.it/771) (<http://adafru.it/771>) to provide 4.5 Volts. Though larger and heavier than the fancy lithium-polymer pack, they're inexpensive and readily available.



Four nickel-metal hydride (NiMH) rechargeable cells can similarly be used in a [4-cell battery holder \(http://adafru.it/830\)](http://adafru.it/830) to provide 4.8 Volts.

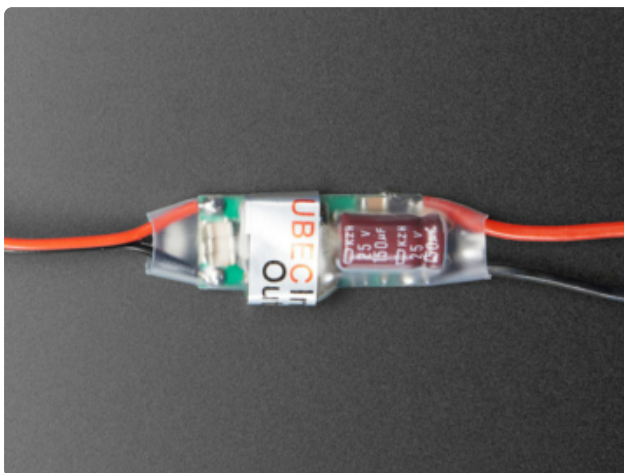
Make sure you only use NiMH cells in this configuration. Four alkaline cells (the disposable type) will output 6V total — that's too high for some NeoPixels, and definitely too much for the microcontroller!



USB power banks can sometimes be used a couple of ways...

Powering a microcontroller board through its USB port, tapping power for the NeoPixels from the board's 5V or VIN pin. Dissecting a USB cable to provide 5V directly to the pixels.

Many USB power banks shut off automatically when current draw is low (e.g. few or no pixels lit), so this requires experimentation and isn't ideal for all situations.



A [buck converter \(\)](#) can efficiently step a higher DC voltage down to 5V for NeoPixels and microcontrollers ... for example if you have a 12V power source, like a car battery.

Different converters have a specific input voltage range and output maximum current, so check specifications carefully.

Battery-operated LED project planning is discussed in greater detail in [Battery Power for LED Pixels and Strips \(\)](#).



For most non-portable “desktop” projects, a [5V DC switching power supply \(http://adafru.it/276\)](http://adafru.it/276) is ideal. This small 2 Amp supply is good for a meter or so of NeoPixel strip. We’ll explain larger projects in a moment.



Be extremely cautious with bench power supplies. Some — even reputable, well-regarded brands — can produce a large voltage spike when initially switched on, instantly destroying your NeoPixels!

If you use a bench supply, do not connect NeoPixels directly. Turn on the power supply first, let the voltage stabilize, then connect the pixels (GND first).

Estimating Power Requirements

Each individual NeoPixel draws up to 60 milliamps at maximum brightness white (red + green + blue). In actual use though, it’s rare for all pixels to be turned on that way. When mixing colors and displaying animations, the current draw will be much less. It’s impossible to estimate a single number for all circumstances, but we’ve been using 1/3 this (20 mA per pixel) as a gross rule of thumb with no ill effects. But if you know for a fact that you need every pixel on at maximum brightness, use the full 60 mA figure.

To estimate power supply needs, multiply the number of pixels by 20, then divide the result by 1,000 for the “rule of thumb” power supply rating in Amps. Or use 60 (instead of 20) if you want to guarantee an absolute margin of safety for all situations. For example:

$$60 \text{ NeoPixels} \times 20 \text{ mA} \div 1,000 = 1.2 \text{ Amps minimum}$$

$$60 \text{ NeoPixels} \times 60 \text{ mA} \div 1,000 = 3.6 \text{ Amps minimum}$$

The choice of “overhead” in your power supply is up to you. Maximum safety and

reliability are achieved with a more generously-sized power supply, and this is what we recommend. Most power supplies can briefly push a little extra current for short periods. Many contain a thermal fuse and will simply shut down if overworked. So they may technically work, but this is the electronics equivalent of abusing a rental car.

Keep in mind, 60 mA is a worst case estimate! We've written a whole separate tutorial on getting things under control: [Sipping Power with NeoPixels \(\)](#).

I estimate I need a 3.6 Amp power supply. I have a 10 Amp supply on-hand. Will this cause my NeoPixels to explode?

As long as the output is 5 Volts DC, you're golden. The LEDs will only draw as much current (Amperes) as they need. So extra Amps are OK — in fact, it can be a thing. The larger power supply will run cooler because it's not being pushed to its limit.

Excessive however, will definitely kill your LEDs.

Extra Amps = good. Extra Volts = bad.

What about batteries and “Amp hours”?

Amp-hours are current over time. A 2,600 mAh (milliamp-hour) battery can be thought of as delivering 2.6 Amps continuously for one hour, or 1.3 Amps for 2 hours, and so forth. In reality, it's not quite linear like that; most batteries have disproportionately shorter run times with a heavy load. Also, most batteries won't take kindly to being discharged in an hour — this can even be dangerous! Select a battery sufficiently large that it will take at least a couple hours to run down. It's both safer for you and better for the longevity of the battery.

I need to power LOTS of NeoPixels and don't have a power supply that large. Can I use several smaller ones?

Maybe. There are benefits to using a single supply, and large power supplies are discussed below. “Non-optimal” doesn't necessarily mean “pessimal” though, and we wouldn't discourage anyone from using what resources they have.

If you go this route, the key is to have all of the ground pins among the strips connected in common, but the +5V from each power supply should be connected only to one length of NeoPixels — those should all be joined. Every power supply is

a little different — not 5 Volts — and this keeps some from back-feeding into others.

Giant Power Supplies

Adafruit offers 5V DC power supplies up to [10 Amps](http://adafru.it/658) (<http://adafru.it/658>). This is usually sufficient for a couple hundred NeoPixels or more. For really large installations, you'll need to look elsewhere.

One possibility is to repurpose an ATX computer power supply. The nice beefy server types often provide up to 30 Amps. Some minor modifications are needed...Google around for “ATX power supply hack.” Note that the ATX 5V rail can be very unstable if there's no load on the 12V rail!

Even larger (and scarier, and much more expensive) are laboratory power supplies with ratings into the hundreds of Amps. Sometimes this is what's needed for architectural scale projects and large stage productions. And occasionally we get requests for help...

Please note that projects of this scale are potentially very dangerous, and the problems of power distribution are fundamentally different than hobby-scale projects. As much as we enjoy helping our customers in the forums, they are for product technical support and not full-on engineering services. If you're developing a project of this scope, hire a professional electrician with experience in high-power, low-voltage systems such as photovoltaics or large RVs and boats. This is no charade.

Power for large-scale NeoPixel installations is discussed further in [this NeoPixel curtain guide](#) ().

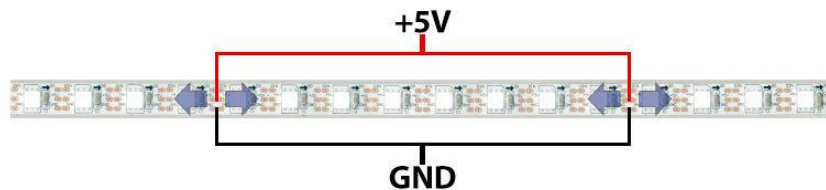
Distributing Power

The longer a wire is, the more resistance it has. The more resistance, the more voltage drops along its length. If voltage drops too far, the color of NeoPixels can be affected.

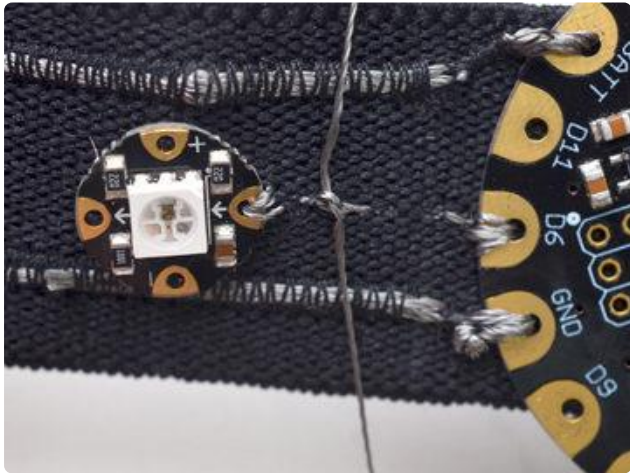
Consider a full 4 meter reel of NeoPixels. With 5V applied at one end of the strip, for those pixels closest to this end, power traverses only a few inches of copper. But at the far end of the strip, power traverses 8 meters of copper — 4 meters out on the +5V line, 4 meters back on the ground line. Those furthest pixels will be tinted brown due to the voltage drop (blue and green LEDs require higher voltage than red).



Pro Tip: NeoPixels don't care what end they receive power from. Though data moves in only one direction, electricity can go either way. You can connect power at the head, the tail, in the middle, or ideally distribute it to several points. For best color consistency, aim for 1 meter or less distance from any pixel to a power connection. With larger NeoPixel setups, think of power distribution as branches of a tree rather than one continuous line.



Resistance is just as much a concern on tiny projects too!



For wearable electronics we like conductive thread...it's flexible and withstands hand washing. Downside is that it doesn't carry much current. Here several strands of conductive thread have been grouped to provide better capacity for the + and - conductors down a pair of suspenders.

(From the [Pac Man Pixel Suspenders \(\)](#) guide.)

Driving 5V NeoPixels from 3.3V Microcontrollers

Increasingly, microcontrollers are running at 3.3 Volts instead of 5 Volts. That's great news for efficiency, but can present a communication problem with 5V NeoPixels. The 3.3V signal from the microcontroller may not be "loud" enough to register with the higher-voltage device. The manufacturer recommends a minimum signal voltage of 70% of the NeoPixel voltage.

There are two ways this can be addressed:

1. Lower the voltage to the NeoPixels so it's closer (or equal) to that of the microcontroller. This is why we recommend LiPo batteries for FLORA projects: 3.7V is enough to run a short length of pixels, and the microcontroller is comfortable at that voltage as well.
2. Use a [logic level shifter](#) () to step up the signal from the microcontroller to the first pixel.

The “Logic Level Shifting” page explains why some projects can work even without this part.

For more info on using a level shifter with your NeoPixels, [have a look at this guide.](#) ()

Logic Level Shifting

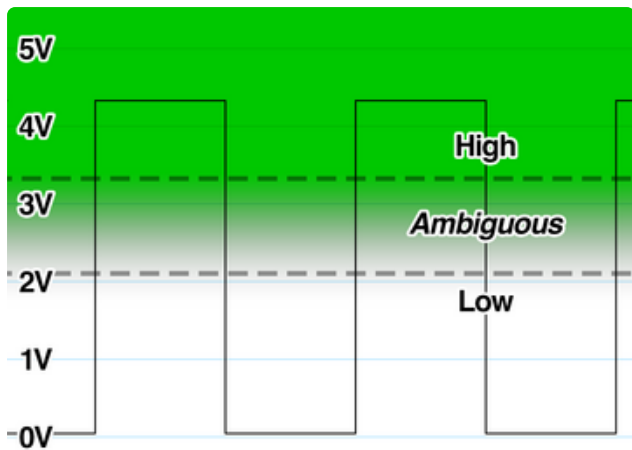
More and more contemporary microcontrollers are 3.3 Volt devices, in fact 5V is becoming a rare breed. The need for logic level shifting—boosting 3.3V signals up to the 5V supply voltage of NeoPixels—is frequently cited, but reality is more nuanced. Even in our own projects, we'll often leave this part out. What's up?

“NeoPixels require 5V logic” is a rule of thumb, a simplification that covers the vast majority of cases. When in doubt, 5V logic has you covered.

More accurately, “WS2812-compatible devices require logic above some percentage of the supply voltage, but that percentage varies from one make and model of device to another, and there are individual component tolerances and external factors like temperature to consider,” but that's a whole lot of words and unknowns, so you can see why the rule of thumb gets used. Lets unpack the whole statement though, to better understand why we sometimes don't bother with level shifting...

Percentage of Supply Voltage

The voltage level that distinguishes logic “high” from “low” is not precisely at 5.000V, but somewhere above zero and below the supply voltage (that is, whatever's powering the NeoPixels). A datasheet will often provide some number...let's say 3.5V... but that's under specific test conditions, e.g. with a specific supply voltage (which the documentation should also mention alongside such figures). That is to say...the minimum reliable “high” logic level is not a hard number, but a relative figure or a percentage of the supply voltage, like $V_{DD} - 0.5V$ or $V_{DD} \times 0.7$.



In the latter case, $V_{DD} \times 0.7$, this is saying the high logic state should be at least 70% of the supply voltage (but can be up to 100%, or even a few percent higher if pixels and controller are powered from separate supplies). That 70% figure isn't canonically true (more later), but is cited often enough; itself another rule of thumb.

70% of 5V is 3.5V, still above the 3.3V output of most microcontrollers. But one shenanigan we can pull is to use a lower supply voltage. Three AA alkaline cells provide about 4.5V, and 70% of that is 3.15V...so 3.3V logic is robust enough! The output of a lithium-polymer cell might range from 3.7–4.2V (70% = 2.6–3V), so again we're good.

NeoPixels are ostensibly 5V devices, but in reality are perfectly happy at slightly lower voltages (again, datasheet). 5 Volts is just a super common and ubiquitous voltage, provided directly by USB and many power bricks. It's an analog world and other voltages are valid.

WS2812-Compatible

WorldSemi was the original manufacturer of this sort of integrated SMD addressable LEDs. Since then, others have come along producing components that use similar signaling, but aren't 1:1 exact versions. These "WS2812-compatible" devices are often improved—better PWM frequency, more tolerant of solder reflow temperatures—and this sometimes includes a more generous logic level range, even if the datasheet isn't explicit about this. In fact, few if any Adafruit products use the "classic" WorldSemi parts anymore.

In some of these cases, that 70% rule of thumb above might be relaxed to, say, 60%. This now means that some 5V parts are perfectly happy with 3.3V logic. There are still factors to be considered, and it's a smart move to use level shifting if you're not sure...but if it's not some life-saving critical infrastructure, and if it tests reliably enough in development, we might forego level shifting in some projects for simplicity!

Tolerances and External Factors

Electronics is often depicted as being all cold, unyielding numbers...but in reality, everything is “ish.” A “5 Volt” power supply is really 5-ish Volts. 3.3V is really 3.3-ish Volts.

This is actually an argument in favor of level shifting, but explaining why sometimes a project works without, even when it shouldn't.

Mass manufacturing just isn't that precise; random factors creep in. Most consumer-grade electronic devices might have tolerances of ± 5 percent...or even a bit more. Take that 5V supply, for instance...produced by the thousands or millions...and then picture a bell curve, with 5V in the middle and most parts falling close to that. Yet you'll have a few outliers approaching 4.5 or 5.5V...but still within allowable tolerances. Anything testing outside some range would be scrapped.

It's not just power supplies. Every single electronic component, even each individual NeoPixel, will have slightly different tolerances...sometimes in opposite directions.

What can happen is that you build a project on your workbench where it tests perfectly, but later it fails when plugged into an “identical” power supply or “identical” NeoPixel strip. One or more parts...power, microcontroller, pixels, were right at the edge of some tolerance and it's only by chance that they worked together. Change just one variable and it's pushed over the edge!

EVEN STRANGER: electronics can be subtly affected by temperature, barometric pressure, ambient light and other “random” factors! Usually a very small amount, but in edge cases this can lead to unreliable operation.

For a casual “just a quick workbench demo” project, maybe the fact that it works in this one controlled setting is good enough and you can skip level shifting. But when it matters, shift!

Datasheets

You'd think that a manufacturer's own component datasheet would be The Final Word™ in settling such matters, but even this is sometimes filled with colorful metaphors.

These specifications are often written in the planning stages, and the final manufactured device might work beyond the intended bounds. Or sometimes figures are overly conservative to future-proof any changes. A part might recommend $\geq 3.5V$

logic when it really could be happy with anything over 3V, at least today. LED manufacturers sometimes make changes to improve production efficiency, even if this alters certain operating thresholds; some padding to the numbers allows wiggle room here.

Often enough, some “5V” parts (like SK6812 pixels—one of the WS2812-compatible alternatives) have proven quite reliable with 3.3V logic, even when documentation suggests otherwise. So this is another situation where we might forego level shifting.

Summary

If you want a NeoPixel project to be maximally robust and bulletproof, logic level shifting ensures reliable signaling. Our [NeoPXL8 FeatherWings and Friend \(\)](#) incorporate this, and all 3.3V [ItsyBitsy boards \(\)](#) have a 5V logic boost on pin 5.

Quite often though you can get away without level shifting, as when using a lower supply voltage. With a 5V supply and no shifting, you’re taking that small chance of unreliable operation, or “works on-desk but not in-field.” That might be good enough for casual, non-critical projects. Ask yourself if failure is an option!

[This guide shows a couple of level-shifting ICs and how to wire them up. \(\)](#)

Software

NeoPixels got their start on Arduino, but have since branched out to other boards and languages.

Pick a category from the left column for information specific to each coding environment.

Arduino Library Installation

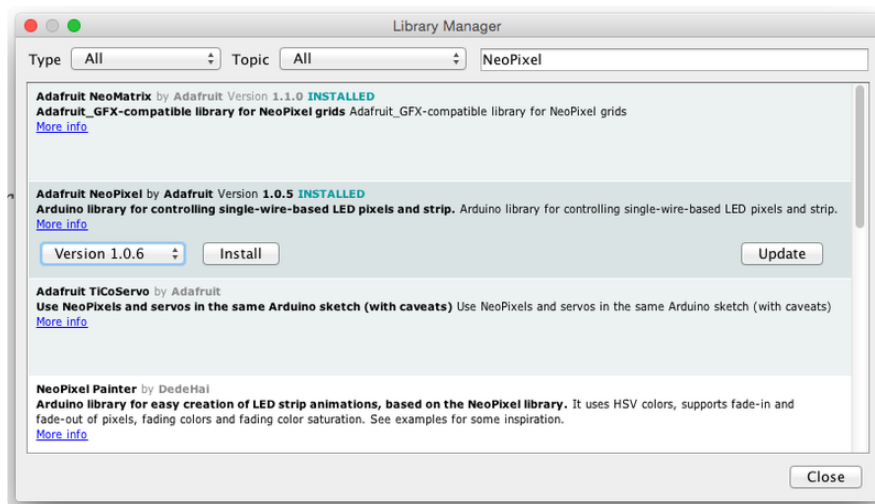
Controlling NeoPixels “from scratch” is quite a challenge, so we provide a library letting you focus on the fun and interesting bits. The library works with most mainstream Arduino boards and derivatives: Uno, Mega, Leonardo, Micro, Adafruit Flora, etc. — most anything with an Atmel AVR 8-bit processor from 8 to 16 MHz — and also works with the Arduino Due and all varieties of the PJRC Teensy boards.

Because processor-specific assembly language is used, this library does not work on Netduino, ChipKIT or other advanced “Arduino-like” boards. Others may have written code and libraries for such boards, but we’re not able to provide technical support for

any bugs or trouble you might encounter there; it's some real frontier engineering. Some of these alternative libraries are covered in the "Advanced Coding" section.

Install Adafruit_NeoPixel via Library Manager

Recent versions of the Arduino IDE (1.6.2 and later) make library installation super easy via the Library Manager interface. From the Sketch menu, > Include Library > Manage Libraries... In the text input box type in "NeoPixel". Look for "Adafruit NeoPixel by Adafruit" and select the latest version by clicking on the popup menu next to the Install button. Then click on the Install button. After it's installed, you can click the "close" button.



Manually Install Adafruit_NeoPixel Library

If you're using an older version of the IDE, or just want to set things up manually, "classic" installation of the library is as follows: you can visit the [Adafruit_NeoPixel library page \(\)](#) at Github and download from there, or just click this button:

Download Adafruit_NeoPixel for Arduino

1. Uncompress the ZIP file after it's finished downloading.
2. The resulting folder should contain the files Adafruit_NeoPixel.cpp, Adafruit_NeoPixel.h and an "examples" sub-folder. Sometimes in Windows you'll get an intermediate-level folder and need to move things around.
3. Rename the folder (containing the .cpp and .h files) to Adafruit_NeoPixel (with the underscore and everything), and place it alongside your other Arduino

libraries, typically in your (home folder)/Documents/Arduino/Libraries folder. Libraries should never be installed in the “Libraries” folder alongside the Arduino application itself...put them in the subdirectory of your home folder.

4. Re-start the Arduino IDE if it's currently running.

[Here's a tutorial \(\)](#) that walks through the process of correctly installing Arduino libraries manually.

A Simple Code Example: strandtest

Launch the Arduino IDE. From the File menu, select

Examples→Adafruit NeoPixel→strandtest

(or, in earlier IDE releases, it's File→Sketchbook→Libraries→Adafruit_NeoPixel→strandtest).

If the Adafruit_NeoPixel rollover menu is not present, the library has not been correctly installed, or the IDE needs to be restarted after installation. Check the installation steps above to confirm it's properly named and located.

Select your board type and serial port from the Tools menu, and try uploading to the board. If the NeoPixels are connected and powered as previously described, you should see a little light show.

Nothing happens!

Check your connections. The most common mistake is connecting to the output end of a strip rather than the input.

Something happens but the LEDs are blinking in a weird way!

If you are using an RGBW NeoPixel product (look at the LEDs, are they divided in half with a yellow semicircle? You have RGBW Neopixels!)

Change this line:

```
Adafruit_NeoPixel strip(LED_COUNT, LED_PIN, NEO_GRB + NEO_KHZ800);
```

to

```
Adafruit_NeoPixel strip(LED_COUNT, LED_PIN, NEO_RGBW + NEO_KHZ800);
```

and reupload the strandtest example.

I don't have RGBW LEDs and the LEDs are still blinking weird!

99% of the time this is due to not having a shared ground wire connected to the Arduino. Make sure the Ground wire from the Neopixels connects to BOTH your power supply ground AND the Arduino ground.

Arduino Library Use

[Doxygen-generated documentation for the Adafruit_NeoPixel library is available here. \(\)](#)

It's assumed at this point that you have the Adafruit_NeoPixel library for Arduino installed and have run the strandtest example sketch successfully. If not, return to the prior page for directions to set that up.

To learn about writing your own NeoPixel sketches, let's begin by dissecting the strandtest sketch...

All NeoPixel sketches begin by including the header file:

```
#include <Adafruit_NeoPixel.h>;
```

The block of code that follows is mostly descriptive comments. Only a couple lines are really doing any work:

```
// Which pin on the Arduino is connected to the NeoPixels?  
// On a Trinket or Gemma we suggest changing this to 1:  
#define LED_PIN 6  
  
// How many NeoPixels are attached to the Arduino?  
#define LED_COUNT 60  
  
// Declare our NeoPixel strip object:  
Adafruit_NeoPixel strip(LED_COUNT, LED_PIN, NEO_GRB + NEO_KHZ800);  
// Argument 1 = Number of pixels in NeoPixel strip  
// Argument 2 = Arduino pin number (most are valid)  
// Argument 3 = Pixel type flags, add together as needed:  
//   NEO_KHZ800  800 KHz bitstream (most NeoPixel products w/WS2812 LEDs)  
//   NEO_KHZ400  400 KHz (classic 'v1' (not v2) FLORA pixels, WS2811 drivers)  
//   NEO_GRB     Pixels are wired for GRB bitstream (most NeoPixel products)  
//   NEO_RGB     Pixels are wired for RGB bitstream (v1 FLORA pixels, not v2)  
//   NEO_RGBW    Pixels are wired for RGBW bitstream (NeoPixel RGBW products)
```

The first few lines assign numbers to the symbols “LED_PIN” and “LED_COUNT” for later reference. It doesn’t need to be done this way, but makes it easier to change the pin and length where the NeoPixels are connected without digging deeper into the code.

The last line declares a NeoPixel object. We’ll refer to this by name later to control the strip of pixels. There are three parameters or arguments in parenthesis:

1. The number of sequential NeoPixels in the strip. In the example this is set to LED_COUNT, which was defined as 60 above, equal to 1 meter of medium-density strip. Change this to match the actual number you’re using.
2. The pin number to which the NeoPixel strip (or other device) is connected. Normally this would be a number, but we previously declared the symbol LED_PIN to refer to it by name here.
3. A value indicating the type of NeoPixels that are connected. In most cases you can leave this off and pass just two arguments; the example code is just being extra descriptive. If you have a supply of classic “V1” Flora pixels, those require NEO_KHZ400 + NEO_RGB to be passed here. RGBW NeoPixels also require a different value here: NEO_RGBW.

For through-hole 8mm NeoPixels, use NEO_RGB instead of NEO_GRB in the strip declaration. For RGBW LEDs use NEO_RGBW (some RGBW strips use NEO_GRBW, so try that if you're getting unexpected results!)

Then, in the setup() function, call begin() to prepare the data pin for NeoPixel output:

```
void setup() {  
  strip.begin();  
  strip.show(); // Initialize all pixels to 'off'  
}
```

The second line, strip.show(), isn’t absolutely necessary, it’s just there to be thorough. That function pushes data out to the pixels...since no colors have been set yet, this initializes all the NeoPixels to an initial “off” state in case some were left lit by a prior program.

The Adafruit Trinket 5V 16 MHz board requires a little extra setup. You can see the steps required in the “strandtest” example sketch.

In the strandtest example, loop() doesn’t set any pixel colors on its own — it calls other functions that create animated effects. So let’s ignore it for now and look ahead, inside the individual functions, to see how the strip is controlled.

There are a couple different ways to set the color of a pixel. The first is:

```
strip.setPixelColor(n, red, green, blue);
```

or, if you're using RGBW strips:

```
strip.setPixelColor(n, red, green, blue, white);
```

The first argument — `n` in this example — is the pixel number along the strip, starting from 0 closest to the Arduino. If you have a strip of 30 pixels, they're numbered 0 through 29. It's a computer thing. You'll see various places in the code using a for loop, passing the loop counter variable as the pixel number to this function, to set the values of multiple pixels.

The next three arguments are the pixel color, expressed as red, green and blue brightness levels, where 0 is dimmest (off) and 255 is maximum brightness. The last optional argument is for white, which will only be used if the strip was defined during creation as an RGBW type and the strip actually is RGBW type.

To set the 12th pixel (#11, counting from 0) to magenta (red + blue), you could write:

```
strip.setPixelColor(11, 255, 0, 255);
```

to set the 8th pixel (#7 counting from 0) to half-brightness white (with an RGBW strip), with no light from red/green/blue, use:

```
strip.setPixelColor(7, 0, 0, 0, 127);
```

An alternate syntax has just two arguments:

```
strip.setPixelColor(n, color);
```

Here, `color` is a 32-bit type that merges the red, green and blue values into a single number. This is sometimes easier or faster for some (but not all) programs to work with; you'll see the strandtest code uses both syntaxes in different places.

You can also convert separate red, green and blue values into a single 32-bit type for later use:

```
uint32_t magenta = strip.Color(255, 0, 255);
```

Then later you can just pass “magenta” as an argument to `setPixelColor` rather than the separate red, green and blue numbers every time.

You can also (optionally) add a white component to the color at the end, like this:

```
uint32_t greenishwhite = strip.Color(0, 64, 0, 64);
```

`setPixelColor()` does not have an immediate effect on the LEDs. To “push” the color data to the strip, call `show()`:

```
strip.show();
```

This updates the whole strip at once, and despite the extra step is actually a good thing. If every call to `setPixelColor()` had an immediate effect, animation would appear jumpy rather than buttery smooth.

Multiple pixels can be set to the same color using the `fill()` function, which accepts one to three arguments. Typically it’s called like this:

```
strip.fill(color, first, count);
```

“color” is a packed 32-bit RGB (or RGBW) color value, as might be returned by `strip.Color()`. There is no option here for separate red, green and blue, so call the `Color()` function to pack these into one value.

“first” is the index of the first pixel to fill, where 0 is the first pixel in the strip, and `strip.numPixels() - 1` is the last. Must be a positive value or 0.

“count” is the number of pixels to fill. Must be a positive value.

If called without a count argument (only color and first), this will fill from first to the end of the strip.

If called without first or count arguments (only color), the full strip will be set to the requested color.

If called with no arguments, the strip will be filled with black or “off,” but there’s also a different syntax which might be easier to read:

```
strip.clear();
```

You can query the color of a previously-set pixel using `getPixelColor()`:

```
uint32_t color = strip.getPixelColor(11);
```

This returns a 32-bit merged RGB color value. This is always RGB, even if the “ColorHSV()” function (described below) was used.

The number of pixels in a previously-declared strip can be queried using numPixels():

```
uint16_t n = strip.numPixels();
```

The overall brightness of all the LEDs can be adjusted using setBrightness(). This takes a single argument, a number in the range 0 (off) to 255 (max brightness). For example, to set a strip to 1/4 brightness:

```
strip.setBrightness(64);
```

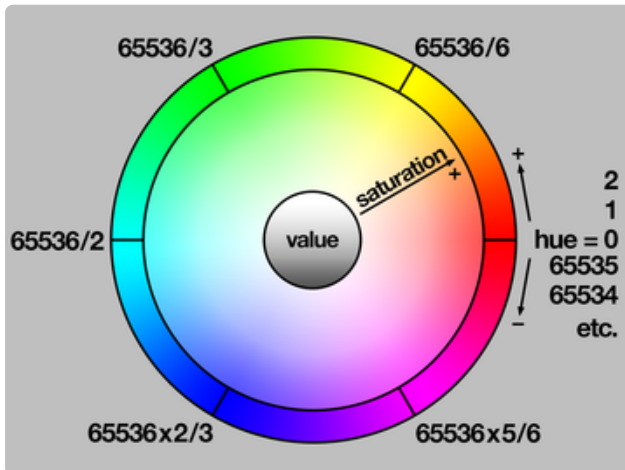
Just like setPixel(), this does not have an immediate effect. You need to follow this with a call to show().

setBrightness() was intended to be called once, in setup(), to limit the current/brightness of the LEDs throughout the life of the sketch. It is not intended as an animation effect itself! The operation of this function is “lossy” — it modifies the current pixel data in RAM, not in the show() call — in order to meet NeoPixels’ strict timing requirements. Certain animation effects are better served by leaving the brightness setting at the default maximum, modulating pixel brightness in your own sketch logic and redrawing the full strip with setPixel().

HSV (Hue-Saturation-Value) Colors...

The NeoPixel library has some support for colors in the “HSV” (hue-saturation-value) color space. This is a different way of specifying colors than the usual RGB (red-green-blue). Some folks find it easier or more “natural” to think about...or quite often it’s just easier for certain color effects (the popular rainbow cycle and such).

In the NeoPixel library, hue is expressed as a 16-bit number. Starting from 0 for red, this increments first toward yellow (around 65536/6, or 10922 give or take a bit), and on through green, cyan (at the halfway point of 32768), blue, magenta and back to red. In your own code, you can allow any hue-related variables to overflow or underflow and they'll "wrap around" and do the correct and expected thing, it's really nice.



Saturation determines the intensity or purity of the color...this is an 8-bit number ranging from 0 (no saturation, just grayscale) to 255 (maximum saturation, pure hue). In the middle, you'll start to get sort of pastel tones.

Value determines the brightness of a color...it's also an 8-bit number ranging from 0 (black, regardless of hue or saturation) to 255 (maximum brightness).

setPixelColor() and fill() both still want RGB values though, so we convert to these from HSV by using the ColorHSV() function:

```
uint32_t rgbcolor = strip.ColorHSV(hue, saturation, value);
```

If you just want a "pure color" (fully saturated and full brightness), the latter two arguments can be left off:

```
uint32_t rgbcolor = strip.ColorHSV(hue);
```

In either case, the resulting RGB value can then be passed to a pixel-setting function, e.g.:

```
strip.fill(rgbcolor);
```

There is no corresponding function to go the other way, from RGB to HSV. This is on purpose and by design, because conversion in that direction is often ambiguous — there may be multiple valid possibilities for a given input. If you look at some of the

example sketches you'll see they keep track of their own hues...they don't assign colors to pixels and then try to read them back out again.

...and Gamma Correction

Something you might observe when working with more nuanced color changes is that things may appear overly bright or washed-out. It's generally not a problem with simple primary and secondary colors, but becomes more an issue with blends, transitions, and the sorts of pastel colors you might get from the `ColorHSV()` function. Numerically the color values are correct, but perceptually our eyes make something different of it, [as explained in this guide \(\)](#).

The `gamma32()` function takes a packed RGB value (as you might get out of `Color()` or `ColorHSV()`) and filters the result to look more perceptually correct.

```
uint32_t rgbcolor = strip.gamma32(strip.ColorHSV(hue, sat, val));
```

You might notice in `strandtest` and other example sketches that we never use `ColorHSV()` without passing the result through `gamma32()` before setting a pixel's color. It's that desirable.

However, the `gamma32` operation is not built in to `ColorHSV()` — it must be called as a separate operation — for a few reasons, including that advanced programmers might want to provide a more specific color-correction function of their own design (`gamma32()` is a “one size fits most” approximation) or may need to keep around the original “numerically but not perceptually correct” numbers.

There is no corresponding reverse operation. When you set a pixel to a color filtered through `gamma32()`, reading back the pixel value yields that filtered color, not the original RGB value. It's precisely because of this sort of decimation that advanced NeoPixel programs often treat the pixel buffer as a write-only resource...they generate each full frame of animation based on their own program state, not as a series of read-modify-write operations.

Help!

I'm calling `setPixel()` but nothing's happening!

There are two main culprits for this:

1. forgetting to call `strip.begin()` in `setup()`.

2. forgetting to call `strip.show()` after setting pixel colors.

Another (less common) possibility is running out of RAM — see the last section below. If the program works but has unpredictable results, consider that.

Can I have multiple NeoPixel objects on different pins?

Certainly! Each requires its own declaration with a unique name:

```
Adafruit_NeoPixel strip_a = Adafruit_NeoPixel(16, 5);  
Adafruit_NeoPixel strip_b = Adafruit_NeoPixel(16, 6);
```

The above declares two distinct NeoPixel objects, one each on pins 5 and 6, each containing 16 pixels and using the implied default type (NEO_KHZ800 + NEO_GRB).

Can I connect multiple NeoPixel strips to the same Arduino pin?

In many cases, yes. All the strips will then show exactly the same thing. This only works up to a point though...four strips on a single pin is a good and reliable number. If you need more than that, individual NeoPixels can be used as buffers to “fan out” to more strips: connect one Arduino pin to the inputs of four separate NeoPixels, then connect each pixels’ output to the inputs of four strips (or fewer, if you don’t need quite that many). If the strips are 10 pixels long, declare the NeoPixel object as having 11 pixels. The extra “buffer” pixels will be at position #0 — just leave them turned off — and the strips then run from positions 1 through 10.

I'm getting the wrong colors. Red and blue are swapped!

When using through-hole 8mm NeoPixels (or V1 Flora pixels), use `NEO_RGB` for the third parameter in the `Adafruit_NeoPixel` declaration. For all other types of NeoPixels, use `NEO_GRB`.

The colors fall apart when I use `setBrightness()` repeatedly!

See note above; `setBrightness()` is designed as a one-time setup function, not an animation effect.

Also see the “Advanced Coding” page — there’s an alternative library that includes “nondestructive” brightness adjustment, among other features!

Pixels Gobble RAM

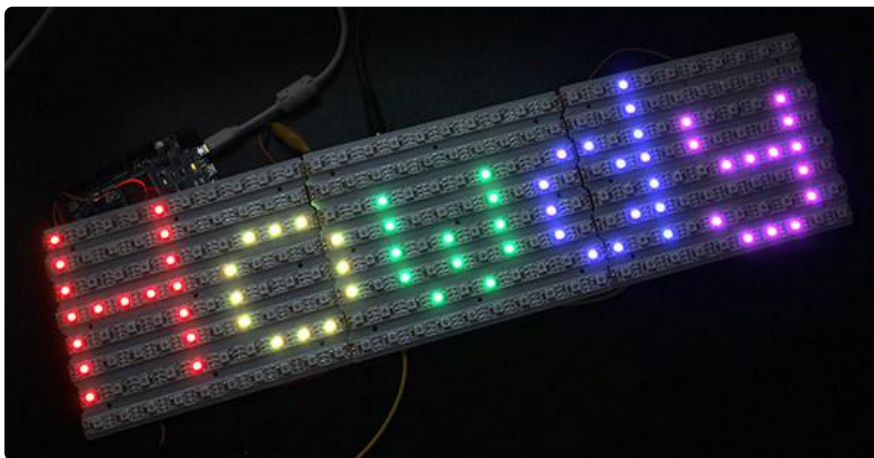
Each NeoPixel requires about 3 bytes of RAM. This doesn't sound like very much, but when you start using dozens or even hundreds of pixels, and consider that the mainstream Arduino Uno only has 2 kilobytes of RAM (often much less after other libraries stake their claim), this can be a real problem!

For using really large numbers of LEDs, you might need to step up to a more potent board like the Arduino Mega or Due. But if you're close and need just a little extra space, you can sometimes tweak your code to be more RAM-efficient. [This tutorial has some pointers on memory usage.](#) ()

Arduino Library Best Practices

[This blog post has suggestions on writing "modern" NeoPixel code](#) () to avoid old copy-and-paste bad habits and deprecated techniques.

NeoMatrix Library



The Adafruit_NeoMatrix library for Arduino builds upon Adafruit_NeoPixel to create two-dimensional graphic displays using NeoPixels. You can then easily draw shapes, text and animation without having to calculate every X/Y pixel position. Small NeoPixel matrices are available in the shop. Larger displays can be formed using sections of NeoPixel strip, as shown in the photo above.

In addition to the Adafruit_NeoPixel library (which was already downloaded and installed in a prior step), NeoMatrix requires two additional libraries:

1. [Adafruit_NeoMatrix](#) ()
2. [Adafruit_GFX](#) ()

If you've previously used any Adafruit LCD or OLED displays, you might already have the latter library installed.

Installation for both is similar to `Adafruit_NeoPixel` before: unzip, make sure the folder name matches the `.cpp` and `.h` files within, then move to your Arduino libraries folder and restart the IDE.

If using an older (pre-1.8.10) Arduino IDE, you'll also need to locate and install [Adafruit_BusIO \(\)](#).

Arduino sketches need to include all three headers just to use this library:

```
#include <Adafruit_GFX.h>;
#include <Adafruit_NeoMatrix.h>;
#include <Adafruit_NeoPixel.h>;
```

Adafruit_GFX Primer

`Adafruit_GFX` is a library providing a set of graphics functions such as lines, circles and text...all of our LCD and OLED displays use this. If you don't have prior experience, [the key concepts of Adafruit_GFX are explained in another guide \(\)](#).

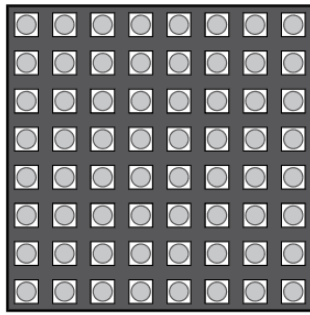
We'll focus on just the NeoMatrix particulars from here on...

Layouts

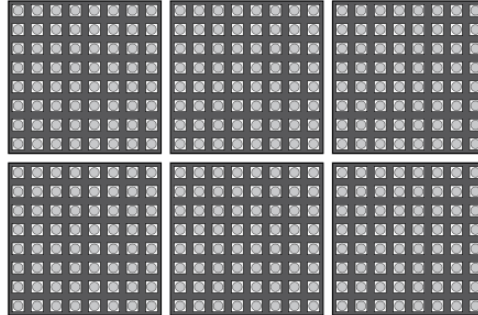
`Adafruit_NeoMatrix` uses exactly the same coordinate system, color functions and graphics commands as the `Adafruit_GFX` library. If you're new to the latter, [a separate tutorial explains its use \(\)](#). There are also example sketches included with the `Adafruit_NeoMatrix` library.

We'll just focus on the constructor here — how to declare a two-dimensional display made from NeoPixels. Powering the beast is another matter, covered on the prior page.

The library handles both single matrices — all NeoPixels in a single uniform grid — and tiled matrices — multiple grids combined into a larger display:

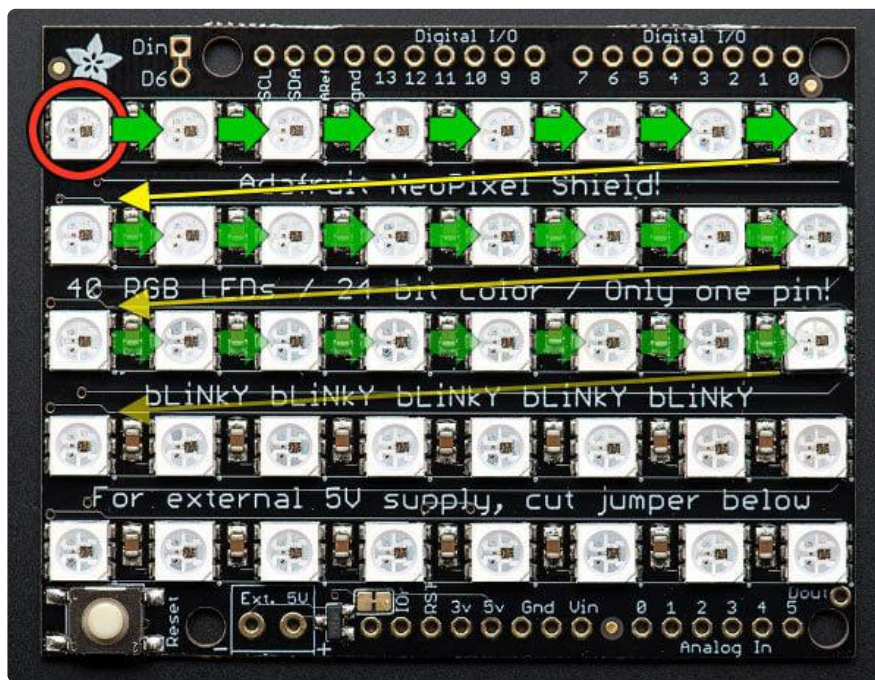


Single Matrix



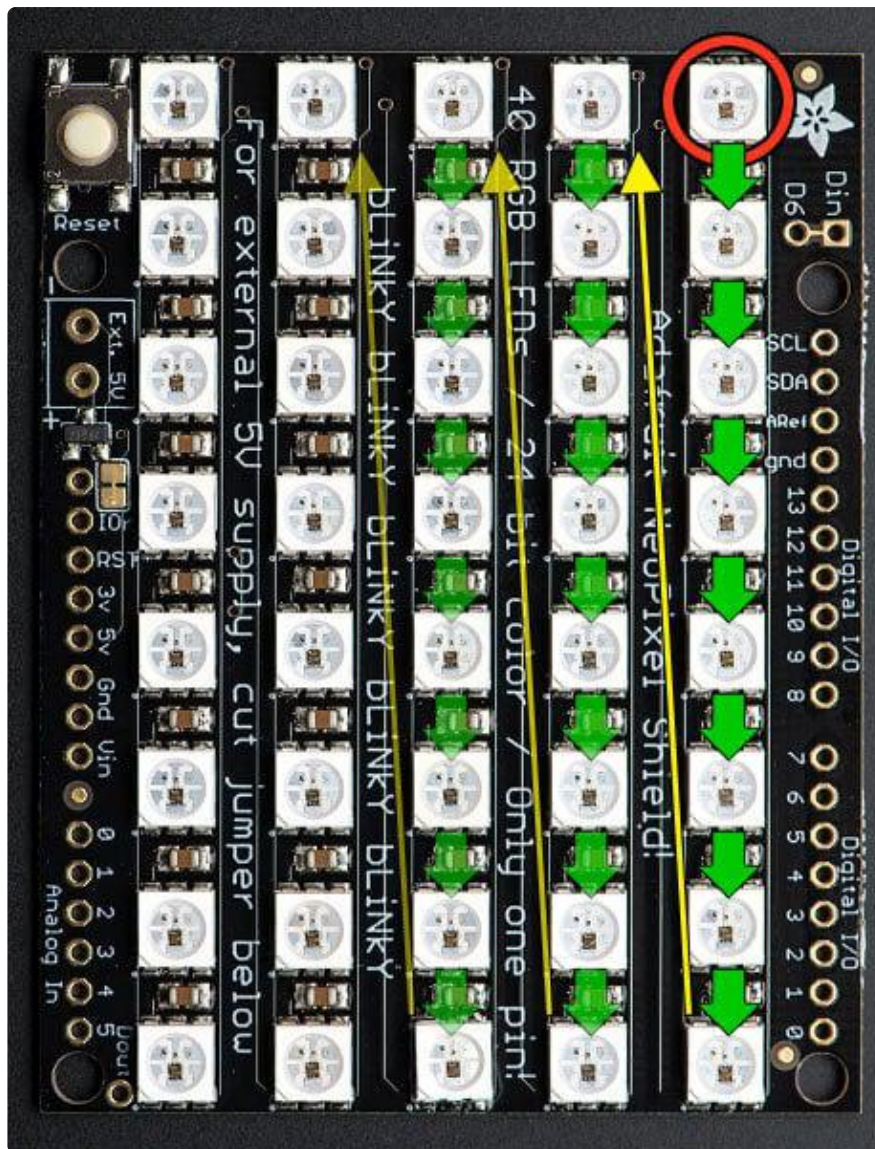
Tiled Matrices

Let's begin with the declaration for a single matrix, because it's simpler to explain. We'll be demonstrating the NeoPixel Shield for Arduino in this case — an 8x5 matrix of NeoPixels. When looking at this shield with the text in a readable orientation, the first pixel, #0, is at the top left. Each successive pixel is right one position — pixel 1 is directly to the right of pixel 0, and so forth. At the end of each row, the next pixel is at the left side of the next row. This isn't something we decide in code...it's how the NeoPixels are hard-wired in the circuit board comprising the shield.



We refer to this layout as row major and progressive. Row major means the pixels are arranged in horizontal lines (the opposite, in vertical lines, is column major). Progressive means each row proceeds in the same direction. Some matrices will reverse direction on each row, as it can be easier to wire that way. We call that a zigzag layout.

However...for this example, we want to use the shield in the “tall” direction, so the Arduino is standing up on the desk with the USB cable at the top. When we turn the board this way, the matrix layout changes...



Now the first pixel is at the top right. Pixels increment top-to-bottom — it's now column major. The order of the columns is still progressive though.

We declare the matrix thusly:

```
Adafruit_NeoMatrix matrix = Adafruit_NeoMatrix(5, 8, 6,
  NEO_MATRIX_TOP + NEO_MATRIX_RIGHT +
  NEO_MATRIX_COLUMNS + NEO_MATRIX_PROGRESSIVE,
  NEO_GRB + NEO_KHZ800);
```

The first two arguments — 5 and 8 — are the width and height of the matrix, in pixels. The third argument — 6 — is the pin number to which the NeoPixels are connected. On the shield this is hard-wired to digital pin 6, but standalone matrices are free to use other pins.

The next argument is the interesting one. This indicates where the first pixel in the matrix is positioned and the arrangement of rows or columns. The first pixel must be

at one of the four corners; which corner is indicated by adding either NEO_MATRIX_TOP or NEO_MATRIX_BOTTOM to either NEO_MATRIX_LEFT or NEO_MATRIX_RIGHT. The row/column arrangement is indicated by further adding either NEO_MATRIX_COLUMNS or NEO_MATRIX_ROWS to either NEO_MATRIX_PROGRESSIVE or NEO_MATRIX_ZIGZAG. These values are all added to form a single value as in the above code.

```
NEO_MATRIX_TOP + NEO_MATRIX_RIGHT + NEO_MATRIX_COLUMNS +  
NEO_MATRIX_PROGRESSIVE
```

The last argument is exactly the same as with the NeoPixel library, indicating the type of LED pixels being used. In the majority of cases with the latest NeoPixel products, you can simply leave this argument off...the example code is just being extra descriptive.

The point of this setup is that the rest of the sketch never needs to think about the layout of the matrix. Coordinate (0,0) for drawing graphics will always be at the top-left, regardless of the actual position of the first NeoPixel.

Why not just use the rotation feature in Adafruit_GFX?

Adafruit_GFX only handles rotation. Though it would handle our example above, it doesn't cover every permutation of rotation that may occur with certain matrix layouts, not to mention the zig-zag capability, or this next bit...

Tiled Matrices

A tiled matrix is comprised of multiple smaller NeoPixel matrices. This is sometimes easier for assembly or for distributing power. All of the sub-matrices need to be the same size, and must be ordered in a predictable manner. The Adafruit_NeoMatrix() constructor then receives some additional arguments:

```
Adafruit_NeoMatrix matrix = Adafruit_NeoMatrix(  
  matrixWidth, matrixHeight, tilesX, tilesY, pin, matrixType, ledType);
```

The first two arguments are the width and height, in pixels, of each tiled sub-matrix, n of the entire display.

The next two arguments are the number of tiles, in the horizontal and vertical direction. The dimensions of the overall display then will always be a multiple of the sub-matrix dimensions.

The fifth argument is the pin number, same as before and as with the NeoPixel library. The last argument also follows prior behaviors, and in most cases can be left off.

The second-to-last argument though...this gets complicated...

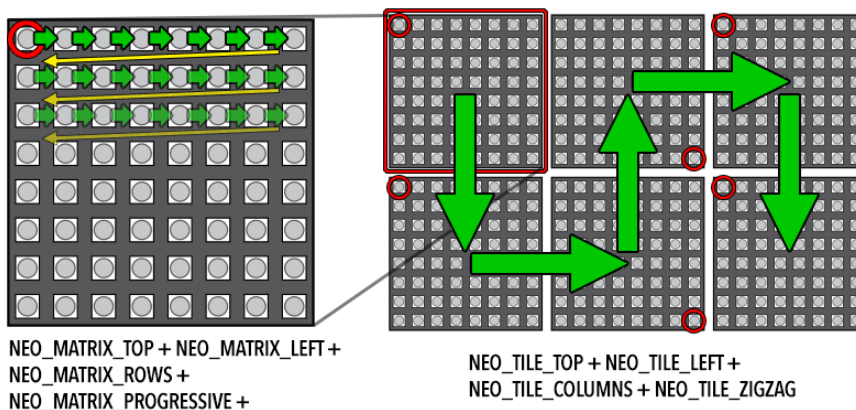
With a single matrix, there was a starting corner, a major axis (rows or columns) and a line sequence (progressive or zigzag). This is now doubled — similar information is needed both for the pixel order within the individual tiles, and the overall arrangement of tiles in the display. As before, we add up a list of symbols to produce a single argument describing the display format.

The NEO_MATRIX_* symbols work the same as in the prior single-matrix case, and now refer to the individual sub-matrices within the overall display. All tiles must follow the same format. An additional set of symbols work similarly to then describe the tile order.

The first tile must be located at one of the four corners. Add either NEO_TILE_TOP or NEO_TILE_BOTTOM and NEO_TILE_LEFT or NEO_TILE_RIGHT to indicate the position of the first tile. This is independent of the position of the first pixel within the tiles; they can be different corners.

Tiles can be arranged in horizontal rows or vertical columns. Again this is independent of the pixel order within the tiles. Add either NEO_TILE_ROWS or NEO_TILE_COLUMNS.

Finally, rows or columns of tiles may be arranged in progressive or zigzag order; that is, every row or column proceeds in the same order, or alternating rows/columns switch direction. Add either NEO_TILE_PROGRESSIVE or NEO_TILE_ZIGZAG to indicate the order. BUT...if NEO_TILE_ZIGZAG order is selected, alternate lines of tiles must be rotated 180 degrees. This is intentional and by design; it keeps the tile-to-tile wiring more consistent and simple. This rotation is not required for NEO_TILE_PROGRESSIVE.



Tiles don't need to be square! The above is just one possible layout. The display shown at the top of this page is three 10x8 tiles assembled from NeoPixel strip.

Once the matrix is defined, the remainder of the project is similar to `Adafruit_NeoPixel`. Remember to use `matrix.begin()` in the `setup()` function and `matrix.show()` to update the display after drawing. The `setBrightness()` function is also available. The library includes a couple of example sketches for reference.

Other Layouts

For any other cases that are not uniformly tiled, you can provide your own function to remap X/Y coordinates to NeoPixel strip indices. This function should accept two unsigned 16-bit arguments (pixel X, Y coordinates) and return an unsigned 16-bit value (corresponding strip index). The simplest row-major progressive function might resemble this:

```
uint16_t myRemapFn(uint16_t x, uint16_t y) {  
    return WIDTH * y + x;  
}
```

That's a crude example. Yours might be designed for pixels arranged in a spiral (easy wiring), or a Hilbert curve.

The function is then enabled using `setRemapFunction()`:

```
matrix.setRemapFunction(myRemapFn);
```

RAM Again

On a per-pixel basis, `Adafruit_NeoMatrix` is no more memory-hungry than `Adafruit_NeoPixel`, requiring 3 bytes of RAM per pixel. But the number of pixels in a two-dimensional display takes off exponentially...a 16x16 display requires four times the memory of an 8x8 display, or about 768 bytes of RAM (nearly half the available space on an Arduino Uno). It can be anywhere from tricky to impossible to combine large displays with memory-hungry libraries such as SD or `ffft`.

Gamma Correction

Because the `Adafruit_GFX` library was originally designed for LCDs (having limited color fidelity), it handles colors as 16-bit values (rather than the full 24 bits that NeoPixels are capable of). This is not the big loss it might seem. A quirk of human vision makes bright colors less discernible than dim ones. The `Adafruit_NeoMatrix` library uses gamma correction to select brightness levels that are visually (though not

numerically) equidistant. There are 32 levels for red and blue, 64 levels for green.

The `Color()` function performs the necessary conversion; you don't need to do any math. It accepts 8-bit red, green and blue values, and returns a gamma-corrected 16-bit color that can then be passed to other drawing functions.

Advanced Coding

FastLED Library

If looking to boost your NeoPixel prowess, you may find everything you need in the [FastLED library](#) (). It's an alternative to the `Adafruit_NeoPixel` library, providing more advanced features like HSV color support, nondestructive brightness setting and high-speed mathematical operations. (It works with other LED types too, such as DotStars!)

FastLED works altogether differently; it's not a drop-in replacement for `Adafruit_NeoPixel`, and existing sketches will require some rewriting.

Note: FastLED currently works only with RGB NeoPixels; RGBW pixels are not yet supported. At all. You will get incorrect and unpredictable colors.

We don't write or maintain FastLED, and can't provide software troubleshooting advice. If requesting help with a FastLED NeoPixel project in the forums, we'll usually ask that you try one of the known-working `Adafruit_NeoPixel` example sketches to narrow down whether it's a hardware or software issue.

[Visit the FastLED web site to get started.](#) ()

FAQ and Further Programming Insights

Help! My Arduino servo code stops working when combined with NeoPixels!

Unfortunately the `NeoPixel` and `Servo` libraries don't play nice together; one is dependent on periodically disabling interrupts, the other absolutely requires interrupts. There are a couple of options here:

- Use a dedicated [servo control shield](http://adafru.it/1411) (<http://adafru.it/1411>) or [breakout board](http://adafru.it/815) (<http://adafru.it/815>), offloading that task from the processor so interrupts are a non-issue.

- Use a [hardware-PWM-based servo library](#) () rather than the stock Arduino Servo library. This can provide rock-steady servo timing without interrupts, but can only control a very limited number of servos (2-3), and only on very specific pins.

When driving NeoPixels I cannot receive infrared codes on my IR receiver!

Just like servos, the infrared library uses software interrupts to poll the IR LED, while the standard NeoPixel library blocks interrupts while NeoPixel are being updated.

If you don't constantly update the NeoPixel, IR will work in between updates, but if you update them all the time, you will need to use another library and a microcontroller more capable than an Uno or Mega. Ideally one with DMA so that NeoPixels don't take up any CPU cycles.

Marc MERLIN explains how to this depending on what chip you have (Teensy, ESP8266 or ESP32):

http://marc.merlins.org/perso/arduino/post_2017-04-03_Arduino-328P-Uno-Teensy3_1-ESP8266-ESP32-IR-and-Neopixels.html ()

How fast can I refresh a string of (N) pixels?

NeoPixels receive data from a fixed-frequency 800 KHz datastream (except for “V1” Flora pixels, which use 400 KHz). Each bit of data therefore requires $1/800,000$ sec — 1.25 microseconds. One pixel requires 24 bits (8 bits each for red, green blue) — 30 microseconds. After the last pixel's worth of data is issued, the stream must stop for at least 50 microseconds for the new colors to “latch.”

For a strip of 100 pixels, that's $(100 * 30) + 50$, or 3,050 microseconds. $1,000,000 / 3,050 = 328$ updates per second, approximately.

However...

That's only the time needed to push the bits down the wire. The actual refresh rate will be something less than this, and can't be estimated as a single number for all cases. It takes time to process each “frame” of animation. How much time depends on the complexity of the math and the efficiency of the code (for example, floating-point calculations can be relatively slow). The formula above gives a maximum theoretical rate, but that's just a starting point. Reality in some cases could fall an order of magnitude (or more) below this.

For exploratory benchmarking, you can always write code as if a large number of pixels were present, and time the result. The extra output bits will simply be ignored by the strip (or you can even test with no NeoPixels connected at all).

That won't do. Now what?

Because NeoPixels use a fixed-frequency clock, options are limited. You can't switch out for a faster microcontroller and expect substantially different results.

One option is to use a different LED type, such as our DotStar or LPD8806 strips, or WS2801 pixels. These can be driven at higher data rates, though they do have some other tradeoffs with respect to NeoPixels (cost, color resolution and/or pixel density).

Another is to develop your own code on a more capable microcontroller or an FPGA that drives multiple NeoPixel strips in parallel. One such project — OctoWS2811 for the Teensy 3 microcontroller — is shown later. This sort of thing is a complex undertaking and not recommended for beginners. And even among more experienced programmers, there's often an unreasonable over-emphasis on data rates when the real bottlenecks lie elsewhere...don't dwell on this too much unless you can confirm it's the root of the problem.

Can I control NeoPixels using (Board X)?

We currently only offer an Arduino library. See the links later for other devices. For anything beyond this, if considering writing your own library, understand that some processors are better suited to the task than others. Read through the timing requirements shown below and determine if the chip in question can synthesize a signal meeting those specifications. An 8 MHz AVR can just barely keep up... anything slower may have trouble, though some hardware-specific hacks (like clever use of SPI) might make it possible. In many cases, assembly language is required.

What's up with Raspberry Pi?

This guide focuses mostly on Arduino and similar microcontroller boards where NeoPixel support is most robust. Raspberry Pi control of NeoPixels is possible, but requires special libraries and/or is limited to specific pins...it's all a bit beyond the scope of this guide. The wiring and power guidance here is valid for any device though.

We tend toward DotStar LEDs in Raspberry Pi projects, as in this [Raspberry Pi light painter \(\)](#).

DMA NeoPixels for ARM Cortex-M0 Boards

If you're using a recent "M0" development board such as the Adafruit Feather M0, Circuit Playground Express or Arduino Zero, [an alternate NeoPixel library \(\)](#) exploits these devices' direct memory access (DMA) feature to operate more smoothly. Advanced Arduino sketches can then use interrupts with impunity, and code that depends on the millis() or micros() functions will not lose time.

There's a corresponding [DMA version of the NeoMatrix library \(\)](#) as well.

Plus a super potent [8-way concurrent NeoPixel DMA library \(\)](#). We offer a companion [FeatherWing \(\)](#) and [breakout board \(\)](#) to make connections and level-shifting easier!

Third-Party Libraries

In addition to the previously-mentioned FastLED library, NeoPixel-compatible libraries have been developed for devices beyond Arduino. Please keep in mind that Adafruit did not develop any of this code and can't fix bugs or offer technical help. This is Wild West stuff.

- [OctoWS2811 \(\)](#): specifically for the PJRC Teensy 3.0 microcontroller board. Uses DMA to drive up to 8 NeoPixel strips concurrently with minimal processor load. Multiple boards can be cascaded for still larger displays.
- [FadeCandy \(\)](#): also for Teensy 3.0. Doesn't support as many pixels as OctoWS2811, but adds dithering and smooth interpolation for color purists.
- [LEDscape \(\)](#): specifically for BeagleBone Black. Although the BeagleBone is a multitasking Linux system like the not-NeoPixel-compatible Raspberry Pi, this code exploits hardware features specific to the BeagleBone Black to drive hundreds of meters of NeoPixel strip with virtually no processor load.
- [WS2812 LED Driver \(\)](#) for Parallax Propeller.
- [xCORE NeoPixel test code \(\)](#) for the XMOS xCORE startKIT.
- [light_ws2812 \(\)](#): an alternate Arduino library for AVR and ARM microcontrollers.

Some of these are 3.3V devices. See the "Powering NeoPixel" page for notes on controlling 5V NeoPixels from 3.3V microcontrollers.

WS2811? WS2812? B? SK6812? Why do I see different names mentioned?

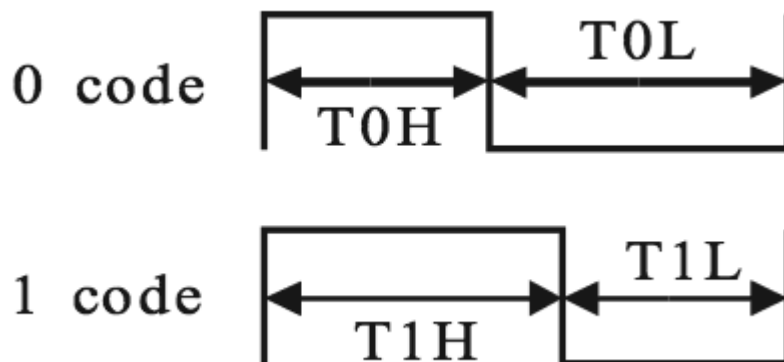
NeoPixels have evolved through several generations:

- WS2811 is a driver chip separate from the RGB LED
- WS2812 integrated the driver and RGB LED in a single package
- WS2812B is a newer generation WS2812 — the color balance is slightly different and the data signal “latch” time is longer
- SK6812 is a “WS2812-compatible” part from other manufacturers — we’ve adopted these for most NeoPixel-branded products as they better tolerate different voltages.

On the original WS2811, chip, the data stream runs at half the speed. When the WS2812 came along, a lot of code and projects had already built up around the WS2811 name — so, for instance, you’ll see projects like “OctoWS2811” and others but they’ll really work with all of these!

Writing Your Own Library

The [WS2812 datasheet \(\)](#) explains the data transmission protocol. This is a self-clocking signal — there’s only one wire, not separate data and clock lines. “1” and “0” bits are indicated by varying the duty cycle of a fixed-frequency square wave.



There’s a math goof in the datasheet’s timing values. Use these figures instead:

Data transfer time ($T_H+T_L=1.25\mu s\pm 300ns$)

T0H	0 code ,high voltage time	0.4us	$\pm 150ns$
T1H	1 code ,high voltage time	0.8us	$\pm 150ns$
T0L	0 code , low voltage time	0.85us	$\pm 150ns$
T1L	1 code ,low voltage time	0.45us	$\pm 150ns$
RES	low voltage time	Above 50 μs	

Note that there's nearly 25% "wobble room" in the timing. So if your code can't match the recommended times exactly, it's usually okay, as long as it's close.

There are three bytes of data for each pixel. These should be issued in green, red, blue order, with the most-significant bit first.

Composition of 24bit data:

G7	G6	G5	G4	G3	G2	G1	G0	R7	R6	R5	R4	R3	R2	R1	R0	B7	B6	B5	B4	B3	B2	B1	B0
----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----

The data for pixel #0 (nearest the microcontroller) is issued first, then pixel #1, and so forth to the furthest pixel. This does not operate like a traditional shift register!

All the data for the whole run must be issued. You can't address individual pixels in the middle of a run.

After all the color data is sent, the data line must be held low for a minimum of 300 microseconds for the new colors to "latch."

You may want to dig through our [Arduino library \(\)](#) for insights. The timing-critical parts are written in AVR assembly language, but it's extensively commented with C-like pseudocode.

My Microcontroller Isn't Fast Enough to Do That

The WS2812 appears to be backwardly-compatible with the 400 KHz WS2811 signal. If you can precisely match the latter chip's timing, either type will respond. The WS2811 protocol is not simply a half-speed WS2812. The duty cycle for the "0" and "1" bits is slightly different. From the [WS2811 datasheet \(\)](#):

T0H	0 code,high voltage time	0.5 μ s	\pm 150ns
T1H	1 code,high voltage time	1.2 μ s	\pm 150ns
T0L	0 code,low voltage time	2.0 μ s	\pm 150ns
T1L	1 code,low voltage time	1.3 μ s	\pm 150ns

Python & CircuitPython

It's easy to use NeoPixel LEDs with Python or CircuitPython and the [Adafruit CircuitPython NeoPixel \(\)](#) module. This module allows you to easily write Python code that controls your LEDs.

You can use these LEDs with any CircuitPython microcontroller board or with a computer that has GPIO and Python [thanks to Adafruit_Blinka, our CircuitPython-for-Python compatibility library \(\)](#).

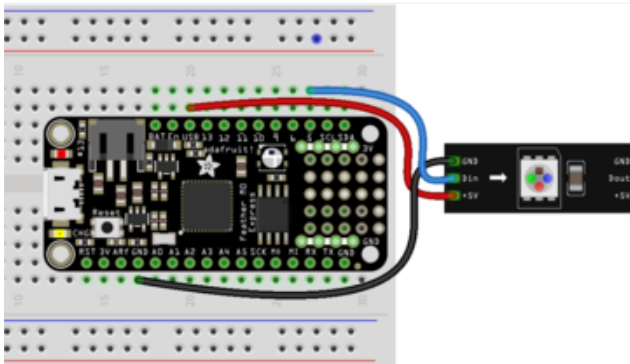
Of single boards computers, only Raspberry Pi computers have NeoPixel support at this time.

CircuitPython Microcontroller Wiring

First wire up some NeoPixels to your board exactly as shown on the previous pages. Verify your connection is on the DATA INPUT or DIN side. Plugging into the DATA OUT or DOUT side is a common mistake! The connections are labeled and some formats have arrows to indicate the direction the data must flow.

Do not use the USB pin on your microcontroller for powering more than a few LEDs! For more than that, you'll want to use an external power source. For more information, check out the Powering NeoPixels page of this guide: <https://learn.adafruit.com/adafruit-neopixel-uberguide/powering-neopixels>

Here's an example of wiring a Feather M0 to a NeoPixel strip:



Board USB to LED 5V

Board GND to LED GND

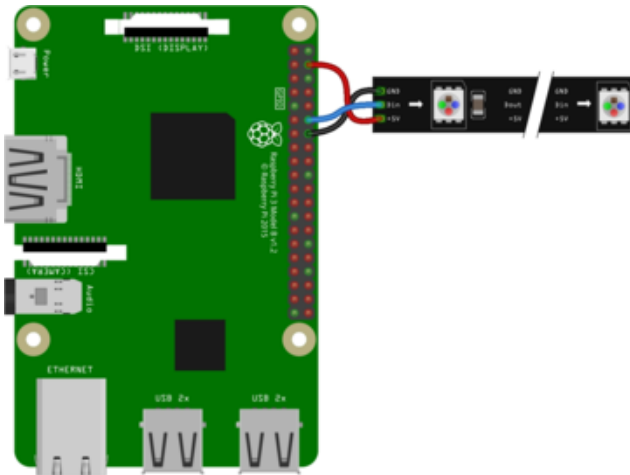
Board D5 to LED Din

The choice of digital pin 5 here is arbitrary for the sake of example. With few exceptions, most any pin will do. On ItsyBitsy boards, this pin is special and has a 5V level shifter built in. A Feather board does not, and the NeoPixel FeatherWing normally uses pin 6 for control (or is solder-jumper selectable).

Python Computer Wiring

Since there's dozens of Linux computers/boards you can use we will show wiring for Raspberry Pi. For other platforms, [please visit the guide for CircuitPython on Linux to see whether your platform is supported \(\)](#).

Here's the Raspberry Pi wired to a NeoPixel strip:



Pi 5V to LED 5V
Pi GND to LED GND
Pi GPIO18 to LED Din
On the Raspberry Pi, NeoPixels must be connected to GPIO10, GPIO12, GPIO18 or GPIO21 to work!

CircuitPython Installation of NeoPixel Library

You'll need to install the [Adafruit CircuitPython NeoPixel \(\)](#) library on your CircuitPython board.

First make sure you are running the [latest version of Adafruit CircuitPython \(\)](#) for your board.

Next you'll need to install the necessary libraries to use the hardware--carefully follow the steps to find and install these libraries from [Adafruit's CircuitPython library bundle \(\)](#). Our CircuitPython starter guide has [a great page on how to install the library bundle \(\)](#).

For non-express boards like the Trinket M0 or Gemma M0, you'll need to manually install the necessary libraries from the bundle:

- neopixel.mpy
- adafruit_bus_device

Before continuing make sure your board's lib folder or root filesystem has the neopixel.mpy, and adafruit_bus_device files and folders copied over.

Next [connect to the board's serial REPL \(\)](#) so you are at the CircuitPython >>> prompt.

Python Installation of NeoPixel Library

You'll need to install the Adafruit_Blinka library that provides the CircuitPython support in Python. This may also require verifying you are running Python 3. [Since each platform is a little different, and Linux changes often, please visit the CircuitPython on Linux guide to get your computer ready \(!\)](#)

Once that's done, from your command line run the following command:

- `sudo pip3 install adafruit-circuitpython-neopixel`

If your default Python is version 3 you may need to run 'pip' instead. Just make sure you aren't trying to use CircuitPython on Python 2.x, it isn't supported!

CircuitPython & Python Usage

To demonstrate the usage of this library with NeoPixel LEDs, we'll use the board's Python REPL.

For NeoPixels to work on Raspberry Pi, you must run the code as root! Root access is required to access the RPi peripherals.

Run the following code to import the necessary modules and initialise a NeoPixel strip with 30 LEDs. Don't forget to change the pin if your NeoPixels are connected to a different pin, and change the number of pixels if you have a different number.

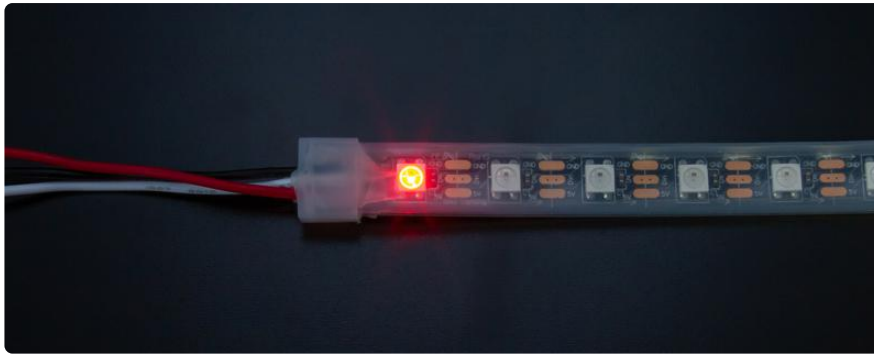
```
import board
import neopixel
pixels = neopixel.NeoPixel(board.D5, 30) # Feather wiring!
# pixels = neopixel.NeoPixel(board.D18, 30) # Raspberry Pi wiring!
```

Now you're ready to light up your NeoPixel LEDs using the following properties:

- brightness - The overall brightness of the LED
- fill - Color all pixels a given color.
- show - Update the LED colors if `auto_write` is set to `False`.

For example, to light up the first NeoPixel red:

```
pixels[0] = (255, 0, 0)
```



To light up all the NeoPixels green:

```
pixels.fill((0, 255, 0))
```



That's all there is to getting started with CircuitPython and NeoPixel LEDs!

Full Example Code

```
# SPDX-FileCopyrightText: 2021 ladyada for Adafruit Industries
# SPDX-License-Identifier: MIT

import time
import board
import neopixel

# On CircuitPlayground Express, and boards with built in status NeoPixel ->
board.NEOPIXEL
# Otherwise choose an open pin connected to the Data In of the NeoPixel strip, i.e.
board.D1
pixel_pin = board.NEOPIXEL

# On a Raspberry pi, use this instead, not all pins are supported
# pixel_pin = board.D18

# The number of NeoPixels
num_pixels = 10

# The order of the pixel colors - RGB or GRB. Some NeoPixels have red and green
reversed!
# For RGBW NeoPixels, simply change the ORDER to RGBW or GRBW.
ORDER = neopixel.GRB

pixels = neopixel.NeoPixel(
    pixel_pin, num_pixels, brightness=0.2, auto_write=False, pixel_order=ORDER
)
```

```

def wheel(pos):
    # Input a value 0 to 255 to get a color value.
    # The colours are a transition r - g - b - back to r.
    if pos < 0 or pos > 255:
        r = g = b = 0
    elif pos < 85:
        r = int(pos * 3)
        g = int(255 - pos * 3)
        b = 0
    elif pos < 170:
        pos -= 85
        r = int(255 - pos * 3)
        g = 0
        b = int(pos * 3)
    else:
        pos -= 170
        r = 0
        g = int(pos * 3)
        b = int(255 - pos * 3)
    return (r, g, b) if ORDER in (neopixel.RGB, neopixel.GRB) else (r, b, g)

def rainbow_cycle(wait):
    for j in range(255):
        for i in range(num_pixels):
            pixel_index = (i * 256 // num_pixels) + j
            pixels[i] = wheel(pixel_index & 255)
        pixels.show()
        time.sleep(wait)

while True:
    # Comment this line out if you have RGBW/GRBW NeoPixels
    pixels.fill((255, 0, 0))
    # Uncomment this line if you have RGBW/GRBW NeoPixels
    # pixels.fill((255, 0, 0, 0))
    pixels.show()
    time.sleep(1)

    # Comment this line out if you have RGBW/GRBW NeoPixels
    pixels.fill((0, 255, 0))
    # Uncomment this line if you have RGBW/GRBW NeoPixels
    # pixels.fill((0, 255, 0, 0))
    pixels.show()
    time.sleep(1)

    # Comment this line out if you have RGBW/GRBW NeoPixels
    pixels.fill((0, 0, 255))
    # Uncomment this line if you have RGBW/GRBW NeoPixels
    # pixels.fill((0, 0, 255, 0))
    pixels.show()
    time.sleep(1)

    rainbow_cycle(0.001) # rainbow cycle with 1ms delay per step

```

RGB LED Colors

RGB LED colors are set using a combination of red, green, and blue, in the form of an (R, G, B) tuple. Each member of the tuple is set to a number between 0 and 255 that determines the amount of each color present. Red, green and blue in different combinations can create all the colors in the rainbow! So, for example, to set an LED

to red, the tuple would be `(255, 0, 0)`, which has the maximum level of red, and no green or blue. Green would be `(0, 255, 0)`, etc. For the colors between, you set a combination, such as cyan which is `(0, 255, 255)`, with equal amounts of green and blue. If you increase all values to the same level, you get white! If you decrease all the values to 0, you turn the LED off.

Common colors include:

- red: `(255, 0, 0)`
- green: `(0, 255, 0)`
- blue: `(0, 0, 255)`
- cyan: `(0, 255, 255)`
- purple: `(255, 0, 255)`
- yellow: `(255, 255, 0)`
- white: `(255, 255, 255)`
- black (off): `(0, 0, 0)`

Python Docs

[Python Docs \(\)](#)

MakeCode

We've got a whole separate guide explaining the use of NeoPixels in [Microsoft MakeCode \(\)](#):

[Guide Link: NeoPixels with MakeCode \(\)](#)

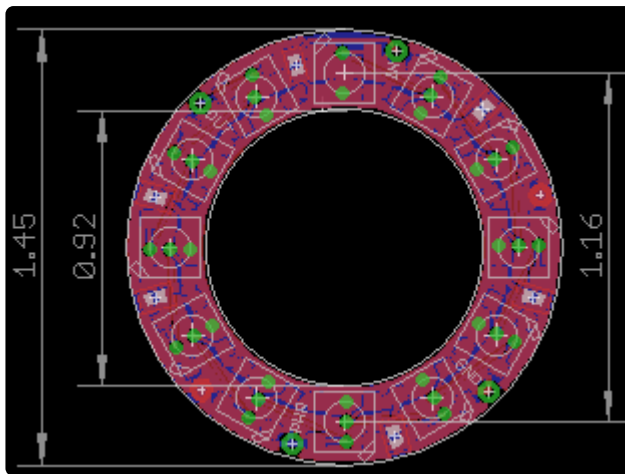
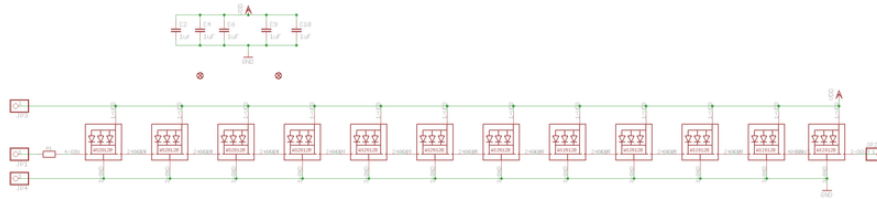
Many of the examples work right in your browser with [Circuit Playground Express \(\)](#) and its 10 built-in NeoPixel LEDs. There's even [a page explaining how to use MakeCode with external strips \(\)](#) as well.

Downloads

- [SK6812 RGBW Datasheet \(\)](#) (used in most of our RGBW products)
- [SK6812 RGB Datasheet \(\)](#) (used in most of our RGB products as of 2016)
- [WS2812B Datasheet \(\)](#) (used in some older RGB products)
- [WS2812 Datasheet \(\)](#) (used in some older RGB products)
- [Online NeoPixel Simulator \(\)](#)

NeoPixel 12-LED Ring

- [EagleCAD PCB files on GitHub \(\)](#)
- [Fritzing object in Adafruit Fritzing library \(\)](#)



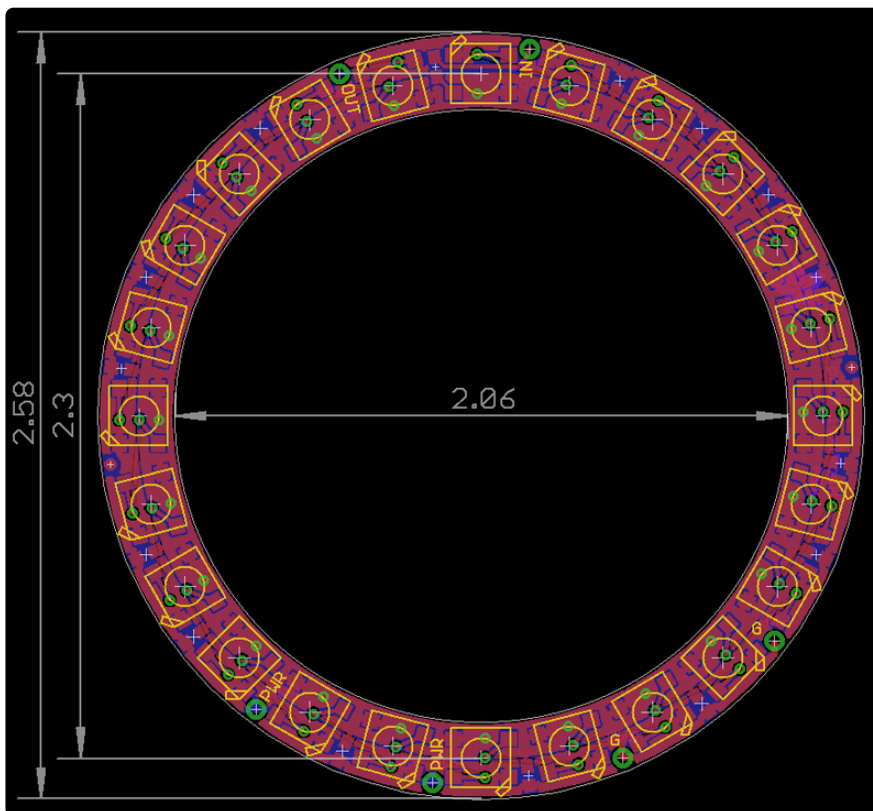
NeoPixel 16-LED Ring

- [EagleCAD PCB files on GitHub \(\)](#)
- [Fritzing object in Adafruit Fritzing library \(\)](#)



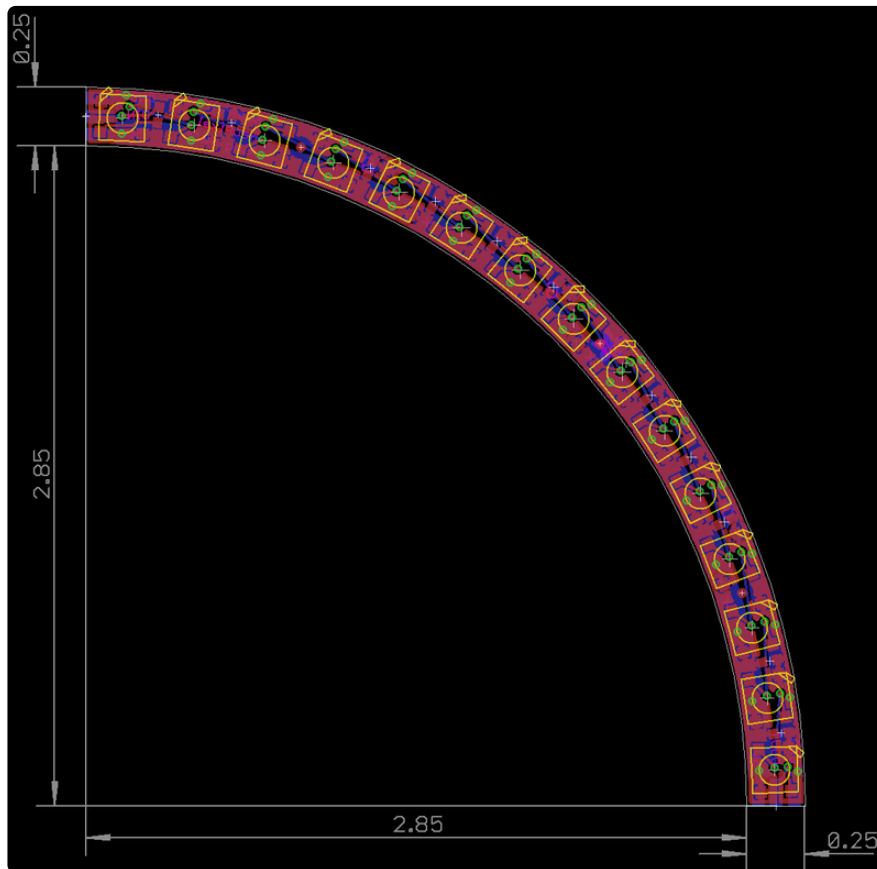
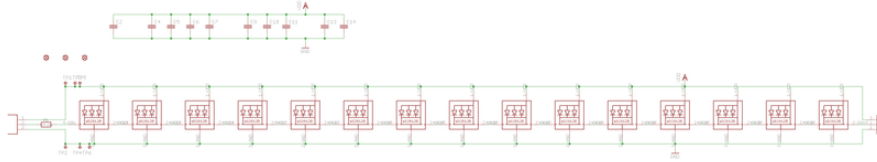
NeoPixel 24-LED Ring

- [EagleCAD PCB files on GitHub \(\)](#)
- [Fritzing object in Adafruit Fritzing library \(\)](#)



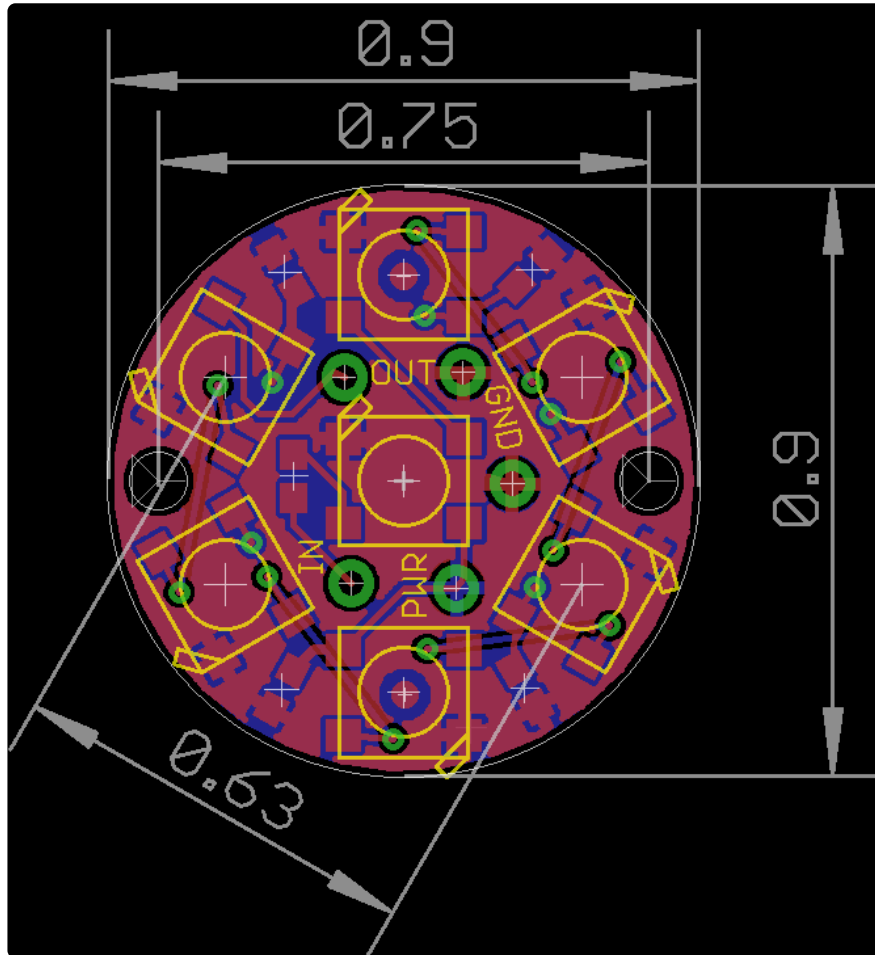
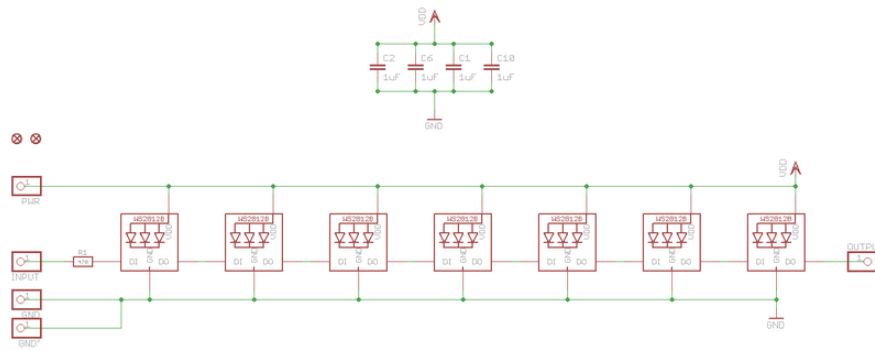
NeoPixel 1/4 60-LED Ring

- [EagleCAD PCB files on GitHub \(\)](#)
- [Fritzing object in Adafruit Fritzing library \(\)](#)



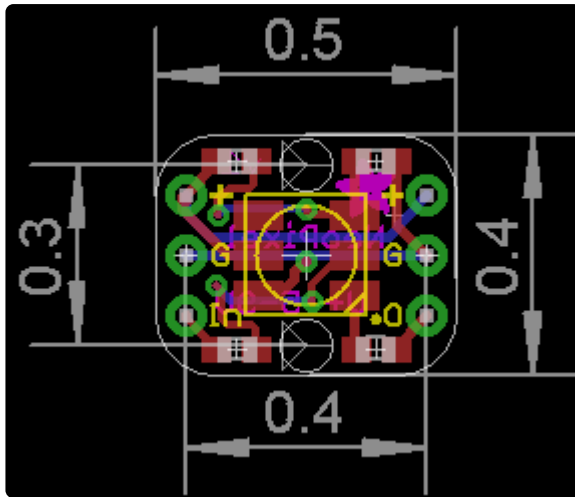
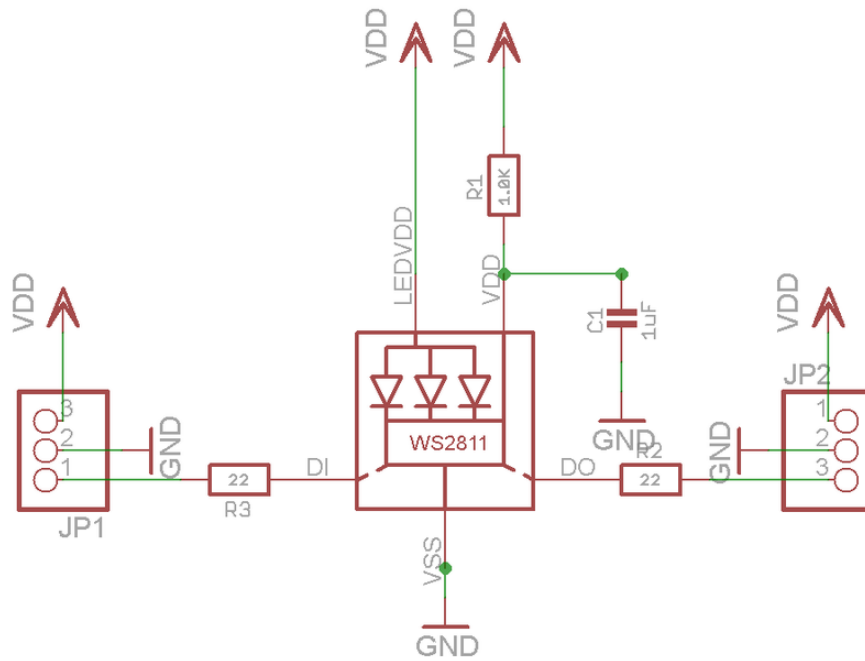
NeoPixel Jewel

- [EagleCAD PCB files on GitHub \(\)](#)
- [Fritzing object in Adafruit Fritzing library \(\)](#)



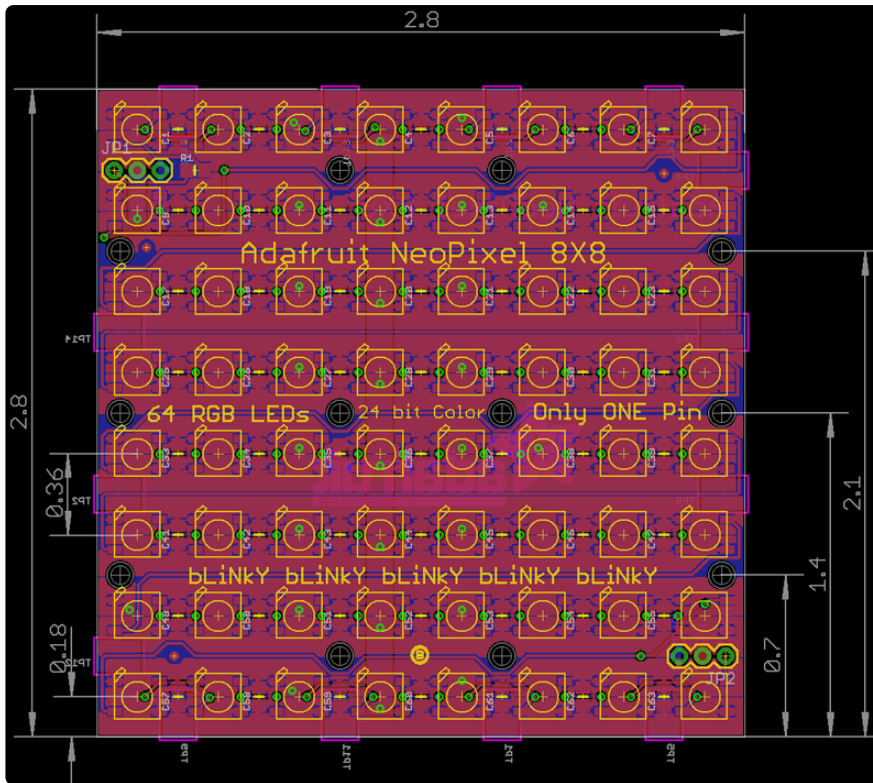
Breadboard Friendly NeoPixel Breakout

- [EagleCAD PCB files on GitHub](#) ()
- [Fritzing object in Adafruit Fritzing library](#) ()



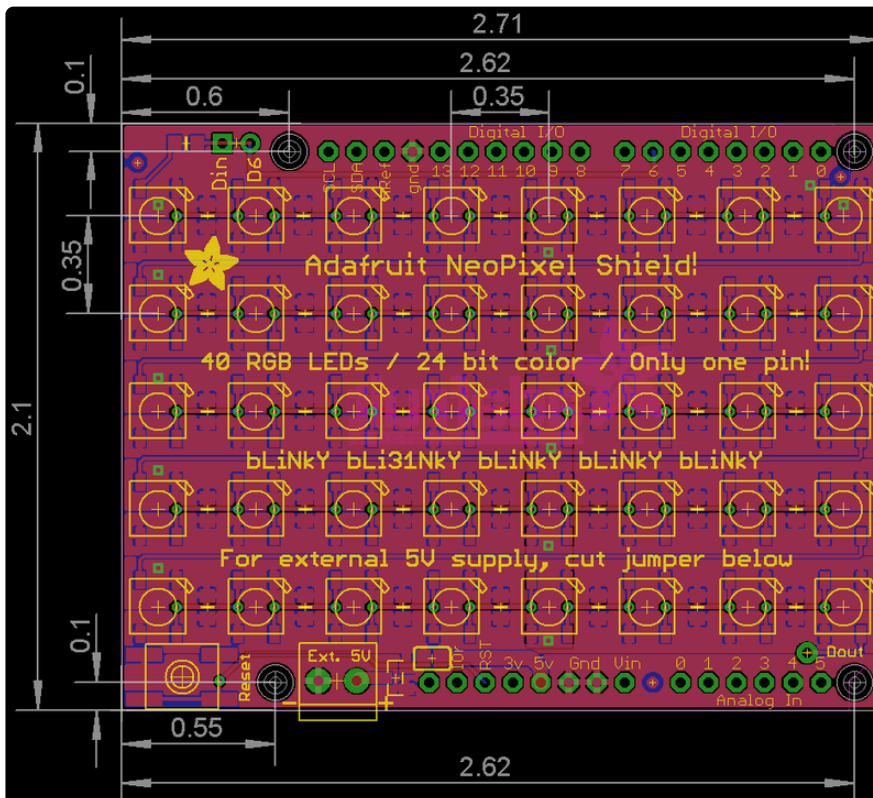
NeoPixel NeoMatrix 8x8

- [EagleCAD PCB files on GitHub \(\)](#)
- [Fritzing object in Adafruit Fritzing library \(\)](#)



NeoPixel Arduino Shield

- [EagleCAD PCB files on GitHub \(\)](#)
- [Fritzing object in Adafruit Fritzing library \(\)](#)



NeoPixel 8 Stick

- [EagleCAD PCB files on GitHub \(\)](#)
- [Fritzing object in Adafruit Fritzing library \(\)](#)

