CircuitPython LED Animations

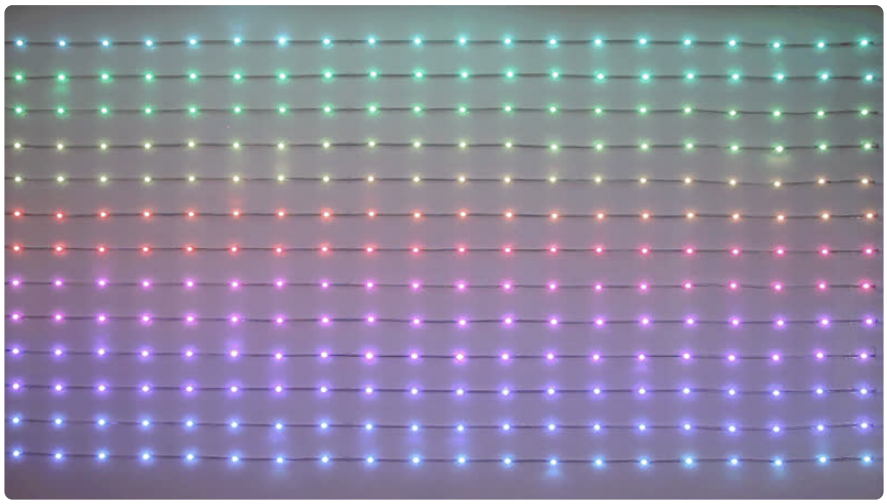
Created by Kattni Rembor

https://learn.adafruit.com/circuitpython-led-animations

Last updated on 2024-08-15 09:34:34 AM EDT

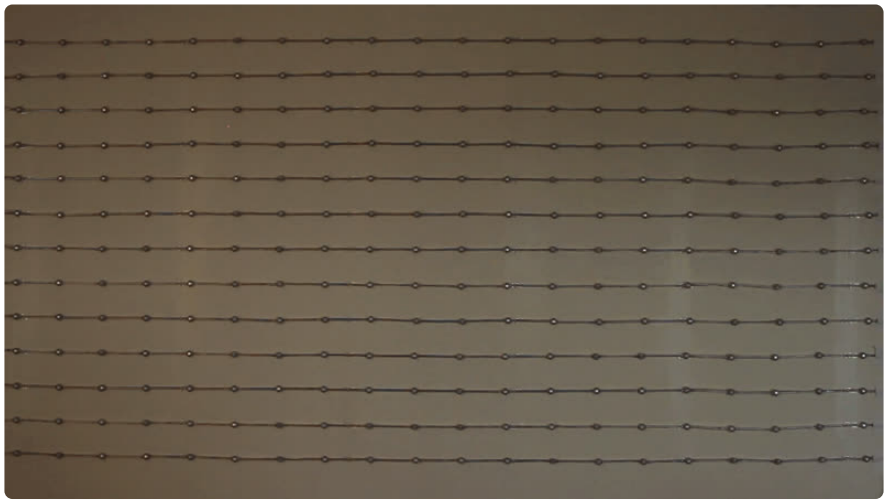
Table of Contents

Overview



One of the first things many people do with CircuitPython is blink an LED. We even consider it our version of "Hello, world!" It's fairly simple code with the little red LED - set the LED to `True`, wait for the desired on period, and set it to `False` for the desired off period. It gets a little more complicated with NeoPixels or DotStars - you have to set the LED to a color, wait, and then set the color to `0` to turn it off. Regardless, blinking is pretty easy. But what if you want to do more?

Creating a beautiful animated display on RGB LEDs, like NeoPixels and DotStars, is simple using the **Adafruit CircuitPython LED Animation** library. This library enables you to display a number of animations including comet, theatre chase, pulse, blink, color cycle, rainbow, sparkle and more.



This library also includes pixel mapping and animation group helpers. The pixel mapping helper allows you to work with strips of LEDs arranged in a grid or other shape to display animations across the grid or shape horizontally and vertically. It also allows you to combine multiple sets of LEDs, e.g. two matrices, and treat them as one

for animation purposes. The animation group helper allows you to keep multiple animations synchronised, or to display two separate animations on two separate pixel objects, e.g. two separate strips.

This guide will walk you through the key features of each animation, such as timing and and other animation-specific customisations. It will cover the basics of pixel mapping to show you how to easily treat LEDs in a strip or series of strips as a grid to display animations, as well as the basics of animation groups to keep multiple animations in sync or display multiple animations across multiple sets of pixels.

Before we get animating, the first thing we'll do is look at what the basic import and setup looks like with the CircuitPython LED Animation library. Let's go!

Import and Setup

The LED Animation library is designed to make displaying LED animations super simple. The first thing you need to do is import the necessary modules from the LED Animation library and create your initial pixel object. Each animation is a separate module to ensure you only import exactly what you need.

CircuitPython LED Animation Library

To get the necessary libraries for this guide, download the latest CircuitPython library bundle from circuitpython.org.

Download the latest CircuitPython library bundle

https://adafru.it/ENC

Open the downloaded zip and find the following folder and file within the **lib** folder:

- **adafruit_led_animation**
- **neopixel.mpy**

Drag this folder and file to the **lib** folder on your **CIRCUITPY** drive.

Import and Setup

The rest of the guide will reference this page. When you are introduced to each animation, the code snippet will not include the entire setup found below. It is

assumed that you have included the rest of the import and setup necessary to make the code run. If you find an example is not working, make sure you've included the entire import and setup found on this page.

An example of import and setup for the NeoPixel FeatherWing is as follows:

```
import board
import neopixel
from adafruit_led_animation.animation.solid import Solid
from adafruit_led_animation.color import RED

pixel_pin = board.D6
pixel_num = 32

pixels = neopixel.NeoPixel(pixel_pin, pixel_num, brightness=0.2, auto_write=False)
```

First you import `board` and `neopixel`. Next, you import the `Solid` module and the color `RED`.

Next you identify the pin to which you've connected to your NeoPixels, `board.D6` in this case, and the number of pixels connected, `32`. This example uses the NeoPixel FeatherWing. If you're using some other NeoPixel form factor, you would change these variables to match the pin you chose and the number of pixels you connected.

Finally, you create the pixel object.

This guide will use NeoPixels for all the examples, but the LED Animation library works equally well with DotStar LEDs. If you are using DotStars, you'll need to load the **adafruit_dotstar.mpy** file onto your CIRCUITPY drive. As well, your import and setup will differ in your code.

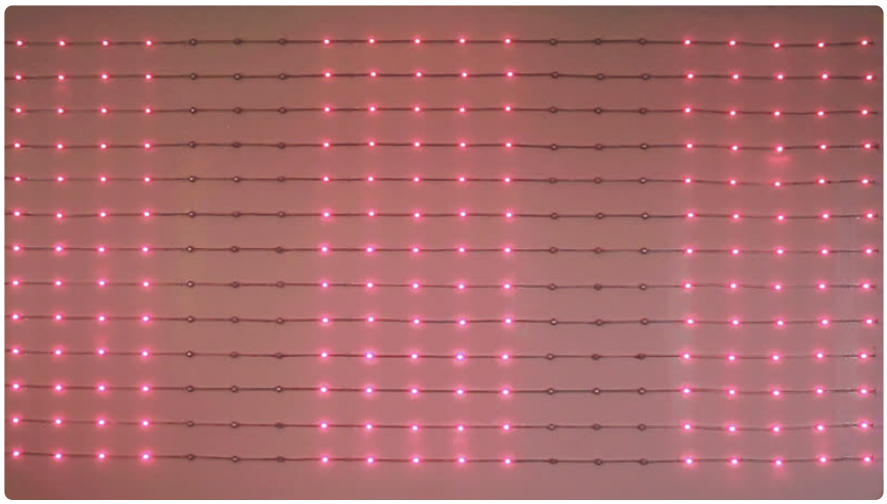
For example:

```
import board
import adafruit_dotstar
from adafruit_led_animation.animation.solid import Solid
from adafruit_led_animation.color import RED

clock_pin = board.D12
data_pin = board.D11
pixel_num = 144

pixels = adafruit_dotstar.DotStar(clock_pin, data_pin, num_pixels,
                                  brightness=0.2, auto_write=False)
```

This example imports the necessary modules and assigns the appropriate pins and number of pixels to use 144 DotStar LEDs connected to D12 and D11.

These are very basic examples of what your import and setup may look like. It will likely end up far more complicated than that as you begin to work with multiple animations and so on. Regardless, this gives you an idea of what to expect. Now it's time to start animating!

Colors



The LED Animation library has a `color` module to make assigning LED colors much simpler. In this module, the RGB value for a color is assigned to a color name variable. In your code, simply import the colors you'd like to use from the module, such as `RED` or `BLUE`, and then use them anywhere you would use an RGB tuple, e.g. `(r, g, b)`.

Available Colors

The current complete list of colors can be found in the color module documentation (https://adafru.it/Rqa). Available colors include:

- `RED`
- `YELLOW`
- `ORANGE`
- `GREEN`
- `TEAL`
- `CYAN`
- `BLUE`
- `PURPLE`
- `MAGENTA`
- `WHITE`
- `BLACK`
- `GOLD`

- `PINK`
- `AQUA`
- `JADE`
- `AMBER`
- `OLD_LACE`

To see the RGB value for each color name, check the documentation (https://adafru.it/Rqa).

Usage

Using the colors with the LED Animation library is easy. Simply import the colors you want to use, and then include them anywhere you would use an RGB value.

This example uses red.

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

"""
This simpletest example displays the Blink animation.

For NeoPixel FeatherWing. Update pixel_pin and pixel_num to match your wiring if
using
a different form of NeoPixels.
"""
import board
import neopixel
from adafruit_led_animation.animation.blink import Blink
from adafruit_led_animation.color import RED

# Update to match the pin connected to your NeoPixels
pixel_pin = board.D6
# Update to match the number of NeoPixels you have connected
pixel_num = 32

pixels = neopixel.NeoPixel(pixel_pin, pixel_num, brightness=0.5, auto_write=False)

blink = Blink(pixels, speed=0.5, color=RED)

while True:
    blink.animate()
```

Included at the top is the following line:

```
from adafruit_led_animation.color import RED
```

Then, when the animation is set up, instead of using `(255, 0, 0)`, you can use `RED`.

```
blink = Blink(pixels, speed=0.5, color=RED)
```

You can import more than one color at the same time as well. This example uses purple, amber and jade.

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

"""
This example uses AnimationsSequence to display multiple animations in sequence, at
a five second
interval.

For NeoPixel FeatherWing. Update pixel_pin and pixel_num to match your wiring if
using
a different form of NeoPixels.
"""
import board
import neopixel

from adafruit_led_animation.animation.blink import Blink
from adafruit_led_animation.animation.comet import Comet
from adafruit_led_animation.animation.chase import Chase
from adafruit_led_animation.sequence import AnimationSequence
from adafruit_led_animation.color import PURPLE, AMBER, JADE

# Update to match the pin connected to your NeoPixels
pixel_pin = board.D6
# Update to match the number of NeoPixels you have connected
pixel_num = 32

pixels = neopixel.NeoPixel(pixel_pin, pixel_num, brightness=0.5, auto_write=False)

blink = Blink(pixels, speed=0.5, color=JADE)
comet = Comet(pixels, speed=0.01, color=PURPLE, tail_length=10, bounce=True)
chase = Chase(pixels, speed=0.1, size=3, spacing=6, color=AMBER)


animations = AnimationSequence(blink, comet, chase, advance_interval=3,
auto_clear=True)

while True:
    animations.animate()
```

Included at the top is a similar import line, but this time, there are multiple color names, separated by commas.

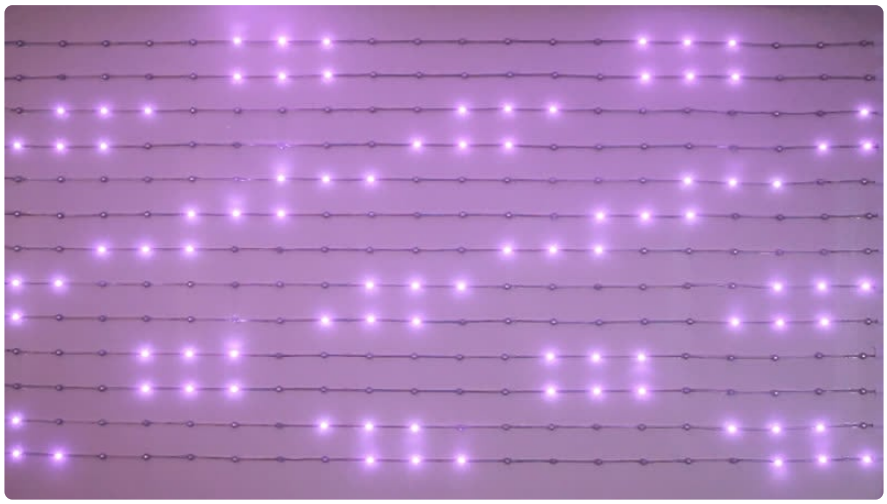
```
from adafruit_led_animation.color import PURPLE, AMBER, JADE
```

Then each color is used in various places later in the code.

```
blink = Blink(pixels, speed=0.5, color=JADE)
comet = Comet(pixels, speed=0.01, color=PURPLE, tail_length=10, bounce=True)
chase = Chase(pixels, speed=0.1, size=3, spacing=6, color=AMBER)
```

Using these colors is not limited to use with the LED Animation library. They can be used anywhere you would otherwise use an RGB tuple value. Load the LED Animation library onto your CIRCUITPY drive, and you can import these colors into any code.

Basic Animations



The CircuitPython LED Animation library provides many animations. This section will cover the basic animations: **solid**, **blink**, **colorcycle**, **chase**, **comet**, and **pulse**. Most of these animations are displayed in a single color, with colorcycle cycling through a list of colors. Each animation has features you can adjust. We'll show the basics of using the animations, and look at the specific features for each one. Let's get animating!

Most animations will run individually on the SAMD21 (M0) microcontroller boards, but some combinations of animations and the most complex animations will not. Check out the FAQ for details. If you're interested in running all the animations, or many animations together, consider using at least a SAMD51 (M4) microcontroller.

Solid

Solid is the simplest of all the animations. It displays a single color. While this is easy enough to do alone without the LED Animation library, you may want to include a solid color in a series of animations, so we made it available.

First you import the `Solid` module and a color for it. See Import and Setup (https://adafru.it/LfT) for the rest of the necessary imports and pixel object creation.

```
from adafruit_led_animation.animation.solid import Solid
from adafruit_led_animation.color import PINK
```

Next, you create the `Solid` animation. `Solid` requires two arguments:

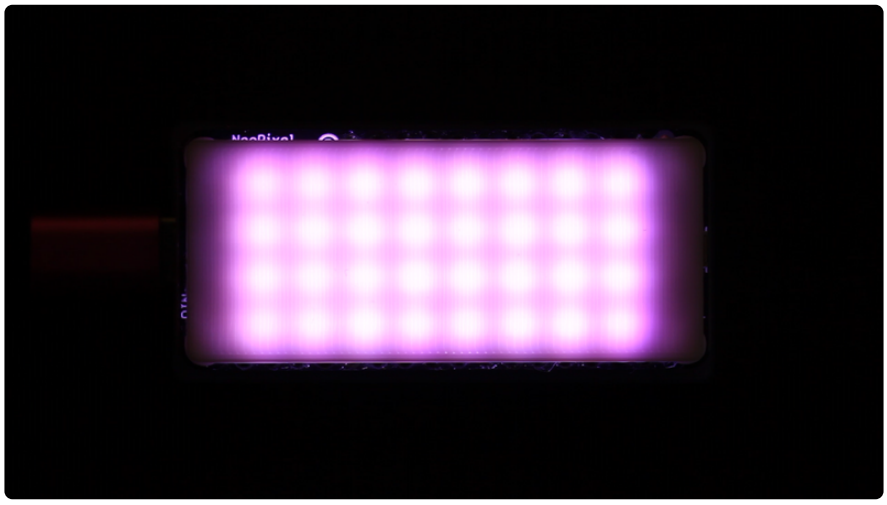
- `pixel_object`: The pixel object, e.g. `pixels`.

- `color`: The color to display, e.g. `PINK`. Can also be a color tuple, e.g. `(255, 0, 0)`, or a hex color value, e.g. `0xFF0000`.

```
solid = Solid(pixels, color=PINK)
```

Then you need to display the animation.

```
while True:
    solid.animate()
```



That's all there is to displaying a solid color using the LED Animation library! Let's take a look at the next animation.

Blink

The blink animation flashes a single color on and off at a specified speed.

First, you import the `Blink` module and a color for it. See Import and Setup (https://adafru.it/LfT) for the rest of the necessary imports and pixel object creation.

```
from adafruit_led_animation.animation.blink import Blink
from adafruit_led_animation.color import JADE
```

Next you create the `Blink` animation. `Blink` requires three arguments:

- `pixel_object`: The pixel object, e.g. `pixels`.
- `speed`: The speed of the blinking in seconds, e.g. `0.5`.
- `color`: The color to display, e.g. `JADE`. Can also be a color tuple, e.g. `(255, 0, 0)`, or a hex color value, e.g. `0xFF0000`.

Once created, you display the animation.

```
blink = Blink(pixels, speed=0.5, color=JADE)

while True:
    blink.animate()
```

That's all there is to blinking a color on and off using the LED Animation library! Let's take a look at the next animation.

ColorCycle

The ColorCycle animation allows you to provide a list of colors to cycle through at a specified speed.

First you import the `ColorCycle` module and one or more colors for it. See Import and Setup (https://adafru.it/LfT) for the rest of the necessary imports and pixel object creation.

```
from adafruit_led_animation.animation.colorcycle import ColorCycle
from adafruit_led_animation.color import MAGENTA, ORANGE, TEAL
```

Next you create the `ColorCycle` animation. `ColorCycle` requires two arguments, and has an optional third. You'll likely want to specify the third, but the animation will run without specifying it.

Required:

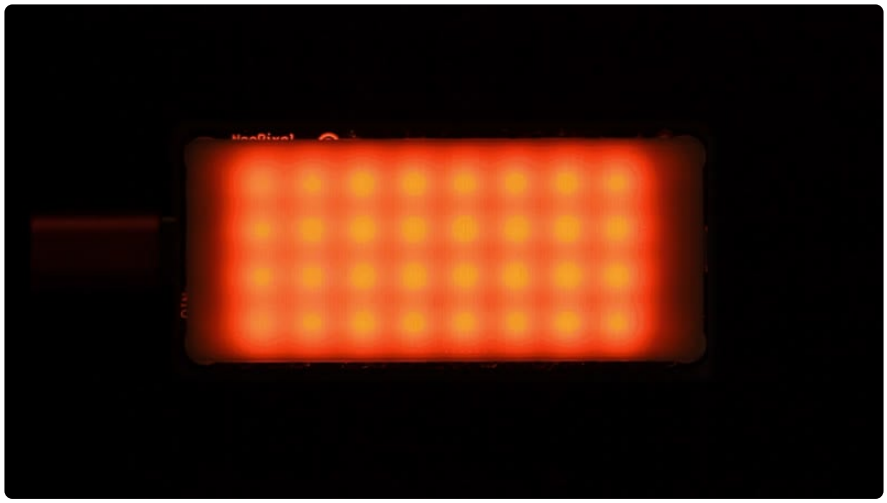
- `pixel_object`: The pixel object, e.g. `pixels`.
- `speed`: The speed of color cycle in seconds, e.g. `0.5`.

Optional:

- `colors`: The list of colors to display, e.g. `[MAGENTA, ORANGE, TEAL]`. This must be a list! Lists are one or more items in `[]`. If no colors are provided, it defaults to cycling through rainbow colors. Can also be a list of color tuples, e.g. `(255, 0, 0)`, or a list of hex color values, e.g. `0xFF0000`.

Once created, you display the animation.

```
colorcycle = ColorCycle(pixels, 0.5, colors=[MAGENTA, ORANGE, TEAL])

while True:
    colorcycle.animate()
```



That's all there is to cycling through a list of colors using the LED Animation library! Let's take a look at the next animation.

Chase

This is a theatre marquee type chase animation, with definable length of lit LEDs and dark gap between lit LEDs.

First you import the `Chase` module and a color for it. See Import and Setup (https://adafru.it/LfT) for the rest of the necessary imports and pixel object creation.

```
from adafruit_led_animation.animation.chase import Chase
from adafruit_led_animation.color import WHITE
```

Next you create the `Chase` animation. `Chase` requires three arguments, and has an optional three more. This animation will run without the optional arguments, but you'll likely want to specify `size` and `spacing` as well.

Required:

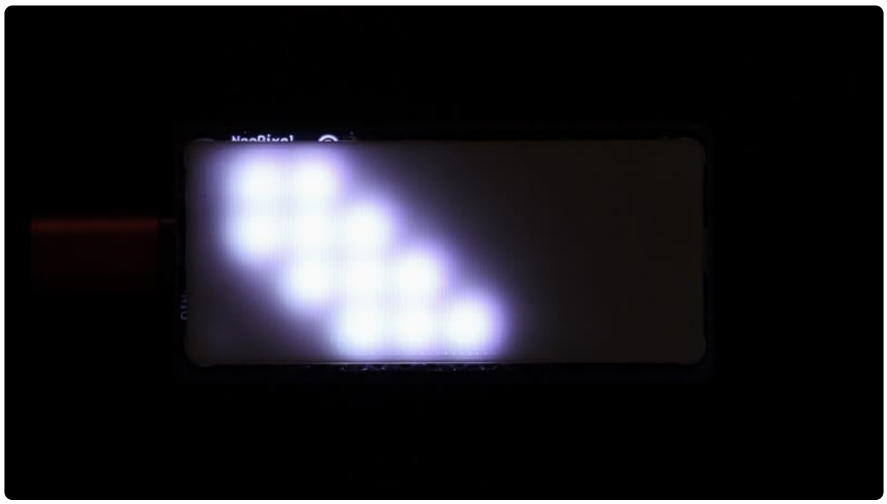
- `pixel_object`: The pixel object, e.g. `pixels`.
- `speed`: The speed of the chase movement in seconds, e.g. `0.1`.
- `color`: The color to display, e.g. `WHITE`. Can also be a color tuple, e.g. `(255, 0, 0)`, or a hex color value, e.g. `0xFF0000`.

Optional:

- `size`: The number of pixels to turn on in a row, e.g. `3`. Defaults to `2` if no size is provided.
- `spacing`: The number of pixels to turn off in a row, e.g. `6`. Defaults to `3` if no size is provided.
- `reverse`: Optionally reverses the movement of the animation. Set to `True` to enable. Defaults to `False`.

Once created, you display the animation.

```
chase = Chase(pixels, speed=0.1, color=WHITE, size=3, spacing=6)

while True:
    chase.animate()
```



That's all there is to creating your own theatre chase animation using the LED Animation library! Let's take a look at the next animation.

Comet

This animation creates a comet of a specified speed, with a dimming tail of specified length.

First you import the `Comet` module and a color for it. See Import and Setup (https://adafru.it/LfT) for the rest of the necessary imports and pixel object creation.

```
from adafruit_led_animation.animation.comet import Comet
from adafruit_led_animation.color import PURPLE
```

Next you create the `Comet` animation. `Comet` requires three arguments, and has an optional three more. You'll likely want to specify at least `tail_length`.

Required:

- `pixel_object`: The pixel object, e.g. `pixels`.
- `speed`: The speed of the comet in seconds, e.g. `0.1`.
- `color`: The color to display, e.g. `PURPLE`. Can also be a color tuple, e.g. `(255, 0, 0)`, or a hex color value, e.g. `0xFF0000`.

Optional:

- `tail_length`: The length of the comet in pixels. Defaults to 25% of the length of the `pixel_object` if no length is provided. Automatically compensates for a minimum length of `2` and a maximum of the length of the `pixel_object`.
- `reverse`: Optionally reverses the movement of the animation. Set to `True` to enable. Defaults to `False`.
- `bounce`: Optionally "bounces" the comet along the strip by displaying it from the beginning of the strip to the end, and then reversing the movement once it reaches the end of the strip. Set to `True` to enable. Defaults to `False`.
- `ring`: Optionally continues the animation from the end back to the beginning without disappearing into the void. Especially nice on NeoPixel rings. Set to `True` to enable. Defaults to `False`.

Once created, you display the animation.

```
comet = Comet(pixels, speed=0.01, color=PURPLE, tail_length=10, bounce=True)

while True:
    comet.animate()
```

That's all there is to displaying a comet using the LED Animation library! Let's take a look at the next animation.

Pulse

This animation pulses all of the LEDs simultaneously a single color at a specified speed.

First you import the `Pulse` module and a color for it. See Import and Setup (https://adafru.it/LfT) for the rest of the necessary imports and pixel object creation.

```
from adafruit_led_animation.animation.pulse import Pulse
from adafruit_led_animation.color import AMBER
```

Next you create the `Pulse` animation. `Pulse` requires three arguments, and has an optional fourth.

Required:

- `pixel_object`: The pixel object, e.g. `pixels`.
- `speed`: The speed of the pulse in seconds, e.g. `0.1`.
- `color`: The color to display, e.g. `AMBER`. Can also be a color tuple, e.g. `(255, 0, 0)`, or a hex color value, e.g. `0xFF0000`.

Optional:

- `period`: The number of seconds over which to pulse the LEDs. Defaults to `5` if no period is provided.

Once created, you display the animation.

```
pulse = Pulse(pixels, speed=0.1, color=AMBER, period=3)

while True:
    pulse.animate()
```

That's all there is to pulsing a single color using the LED Animation library!

Next up, we'll look at running multiple animations in a sequence. Let's go!

Full Example Code

This is the simpletest example from the LED Animation library.

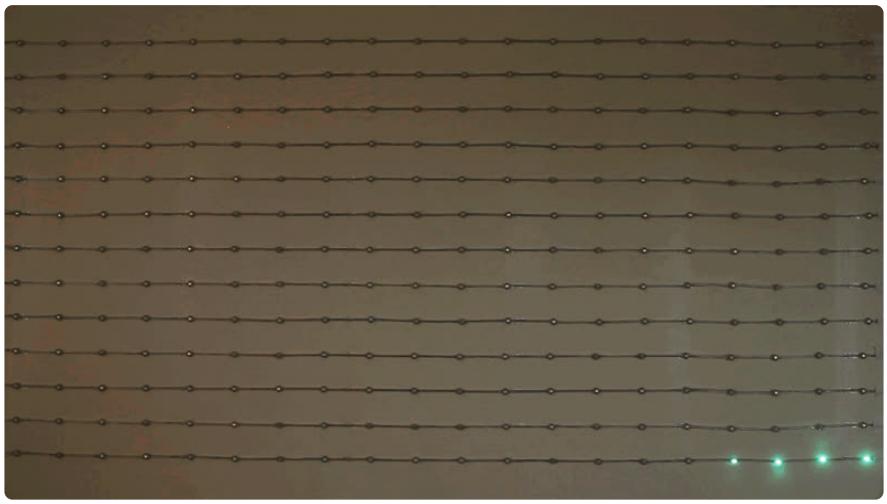
```
# SPDX-FileCopyrightText: 2021 Kattni Rembor for Adafruit Industries
# SPDX-License-Identifier: MIT

"""
This simpletest example displays the Blink animation.

For NeoPixel FeatherWing. Update pixel_pin and pixel_num to match your wiring if
using
a different form of NeoPixels.
"""
import board
import neopixel
from adafruit_led_animation.animation.blink import Blink
from adafruit_led_animation.color import RED

# Update to match the pin connected to your NeoPixels
pixel_pin = board.D6
# Update to match the number of NeoPixels you have connected
pixel_num = 32

pixels = neopixel.NeoPixel(pixel_pin, pixel_num, brightness=0.5, auto_write=False)

blink = Blink(pixels, speed=0.5, color=RED)
```

```
while True:
    blink.animate()
```

This is an example that runs all of the basic animations in a sequence.

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

"""
This example displays the basic animations in sequence, at a five second interval.

For NeoPixel FeatherWing. Update pixel_pin and pixel_num to match your wiring if
using
a different form of NeoPixels.

This example may not work on SAMD21 (M0) boards.
"""
import board
import neopixel

from adafruit_led_animation.animation.solid import Solid
from adafruit_led_animation.animation.colorcycle import ColorCycle
from adafruit_led_animation.animation.blink import Blink
from adafruit_led_animation.animation.comet import Comet
from adafruit_led_animation.animation.chase import Chase
from adafruit_led_animation.animation.pulse import Pulse
from adafruit_led_animation.sequence import AnimationSequence
from adafruit_led_animation.color import (
    PURPLE,
    WHITE,
    AMBER,
    JADE,
    TEAL,
    PINK,
    MAGENTA,
    ORANGE,
)

# Update to match the pin connected to your NeoPixels
pixel_pin = board.D6
# Update to match the number of NeoPixels you have connected
pixel_num = 32

pixels = neopixel.NeoPixel(pixel_pin, pixel_num, brightness=0.5, auto_write=False)

solid = Solid(pixels, color=PINK)
blink = Blink(pixels, speed=0.5, color=JADE)
colorcycle = ColorCycle(pixels, speed=0.4, colors=[MAGENTA, ORANGE, TEAL])
chase = Chase(pixels, speed=0.1, color=WHITE, size=3, spacing=6)
comet = Comet(pixels, speed=0.01, color=PURPLE, tail_length=10, bounce=True)
pulse = Pulse(pixels, speed=0.1, color=AMBER, period=3)


animations = AnimationSequence(
    solid,
    blink,
    colorcycle,
    chase,
    comet,
    pulse,
    advance_interval=5,
    auto_clear=True,
)

while True:
    animations.animate()
```

Animation Sequence



The LED Animation library makes displaying animations on LEDs super simple. You've gone through the basic animations and how to use each of them individually. What if you want to run multiple animations in sequence? The LED Animation library has you covered with `AnimationSequence`.

`AnimationSequence` allows you to display multiple animations in a sequence, with a definable interval and a few other customisation options including clearing the pixels between animations and displaying them in a random order.

To use it, you'll want to include the following in your imports.

```
from adafruit_led_animation.sequence import AnimationSequence
```

The rest of the imports and setup is the same, however, you'll want to include multiple animations this time. This example for the NeoPixel FeatherWing includes blink, comet, and chase. You'll create the pixel object and the animations the same way you did in the previous sections.

```
import board
import neopixel

from adafruit_led_animation.animation.blink import Blink
from adafruit_led_animation.animation.comet import Comet
from adafruit_led_animation.animation.chase import Chase
from adafruit_led_animation.sequence import AnimationSequence
from adafruit_led_animation.color import PURPLE, AMBER, JADE

pixel_pin = board.D6
pixel_num = 32

pixels = neopixel.NeoPixel(pixel_pin, pixel_num, brightness=0.2, auto_write=False)

blink = Blink(pixels, speed=0.5, color=JADE)
```

```
comet = Comet(pixels, speed=0.01, color=PURPLE, tail_length=10, bounce=True)
chase = Chase(pixels, speed=0.1, size=3, spacing=6, color=AMBER)
```

Next, you'll create the `AnimationSequence` object. `AnimationSequence` takes up to six arguments, but you're most commonly going to use a combination these four:

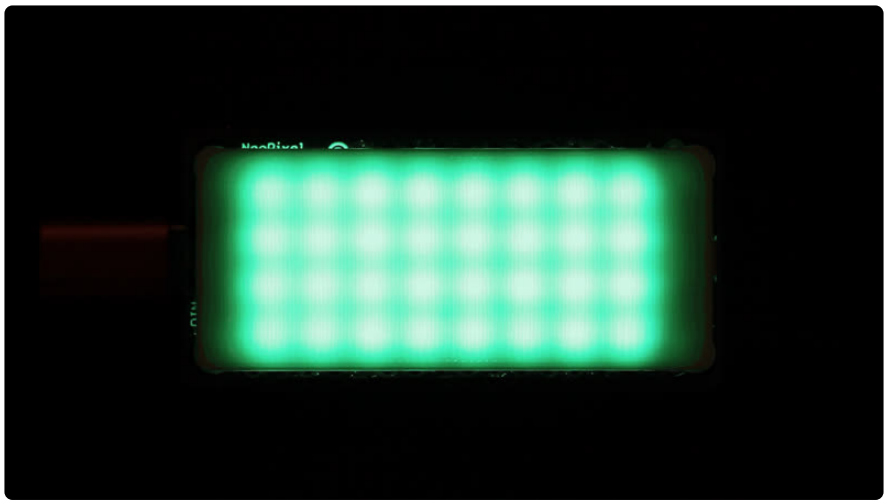
- `members`: The animation objects or groups, e.g. `comet, blink, chase`.
- `advance_interval`: Time in seconds between animations if cycling automatically, e.g. `5`. Defaults to `None` - it will not advance if an interval is not provided.
- `auto_clear`: Clear the pixels between animations. Set to `True` to enable. Defaults to `False`.
- `random_order`: Activate the animations in a random order. Set to `True` to enable. Defaults to `False`.

Check out the API documentation (https://adafru.it/LcO) for information on the other two.

```
animations = AnimationSequence(
    comet, blink, chase, advance_interval=5, auto_clear=True, random_order=True
)
```

Finally, you'll display the animations.

```
while True:
    animations.animate()
```



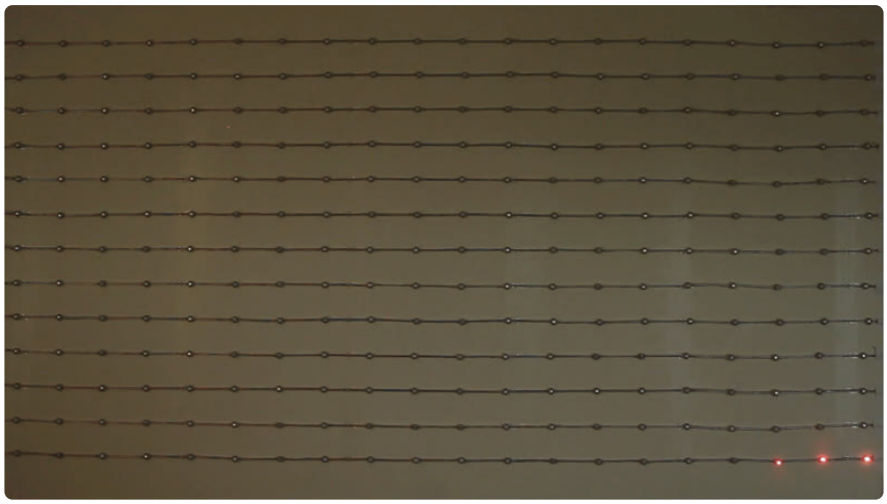
Now you can display multiple animations in a sequence! Now we'll take a look at some more animations. Next up: rainbows!

Full Example Code

This example displays three animations in a sequence, at a 5 second interval, and in a random order.

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

"""
This example uses AnimationsSequence to display multiple animations in sequence, at
a five second
interval.

For NeoPixel FeatherWing. Update pixel_pin and pixel_num to match your wiring if
using
a different form of NeoPixels.
"""
import board
import neopixel

from adafruit_led_animation.animation.blink import Blink
from adafruit_led_animation.animation.comet import Comet
from adafruit_led_animation.animation.chase import Chase
from adafruit_led_animation.sequence import AnimationSequence
from adafruit_led_animation.color import PURPLE, AMBER, JADE

# Update to match the pin connected to your NeoPixels
pixel_pin = board.D6
# Update to match the number of NeoPixels you have connected
pixel_num = 32

pixels = neopixel.NeoPixel(pixel_pin, pixel_num, brightness=0.5, auto_write=False)

blink = Blink(pixels, speed=0.5, color=JADE)
comet = Comet(pixels, speed=0.01, color=PURPLE, tail_length=10, bounce=True)
chase = Chase(pixels, speed=0.1, size=3, spacing=6, color=AMBER)


animations = AnimationSequence(blink, comet, chase, advance_interval=3,
auto_clear=True)

while True:
    animations.animate()
```

Rainbows



The CircuitPython LED Animation library includes a series of rainbow animations: **rainbow**, **rainbowchase**, **rainbowcomet**, and **rainbowsparkle**. This section will cover these animations and the available customisations for each of them. Let's make some rainbows!

Most animations will run individually on the SAMD21 (M0) microcontroller boards, but some combinations of animations and the most complex animations will not. Check out the FAQ for details. If you're interested in running all the animations, or many animations together, consider using at least a SAMD51 (M4) microcontroller.

Rainbow

This animation displays a shifting rainbow across all the pixels with a number of customization options.

First, you import the `Rainbow` module. See Import and Setup (https://adafru.it/LfT) for the rest of the necessary imports and pixel object creation.

```
from adafruit_led_animation.animation.rainbow import Rainbow
```

Next you create the `Rainbow` animation. `Rainbow` requires two arguments, and has an optional three more.

Required:

- `pixel_object`: The pixel object, e.g. `pixels`.

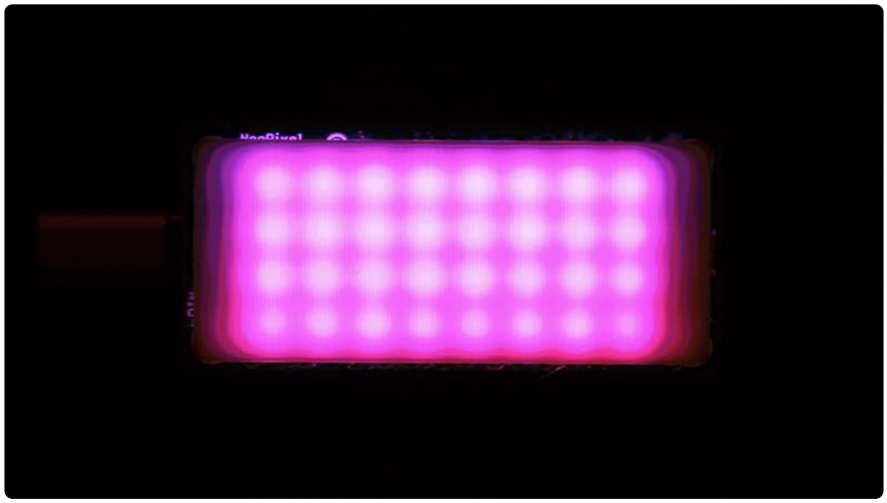
- `speed`: The refresh rate of the rainbow in seconds, e.g. `0.1`.

Optional:

- `period`: The period over which to cycle the rainbow in seconds, e.g. `2`. Defaults to 5 if no period is provided.
- `step`: The steps to take through the colorwheel (0-255). A step of `1` means cycling through the entire colorwheel, a step of `2` means it cycles through every other possible value. Defaults to `1` if no step is provided.
- `precompute_rainbow`: Precompute the rainbow which increases its speed, but uses more memory. Set to `False` to disable if you are running into memory limitations. Defaults to `True`.

Once created, you display the animation.

```
rainbow = Rainbow(pixels, speed=0.1, period=2)

while True:
    rainbow.animate()
```



That's all there is to displaying a rainbow using the LED Animation library! Let's take a look at the next animation.

RainbowChase

This is a rainbow version of the theatre marquee type chase animation, with definable length of lit LEDs and the dark gap between lit LEDs.

First, you import the `RainbowChase` module. See Import and Setup (https://adafru.it/LfT) for the rest of the necessary imports and pixel object creation.

```
from adafruit_led_animation.animation.rainbowchase import RainbowChase
```

Next you create the `RainbowChase` animation. `RainbowChase` requires two arguments, and has an optional five more. You'll likely want to at least specify `size` and `spacing`.

Required:

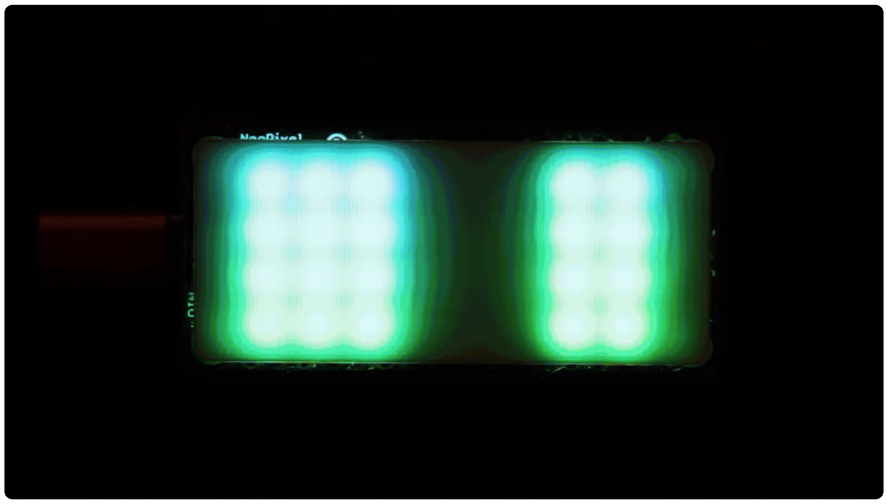
- `pixel_object`: The pixel object, e.g. `pixels`.
- `speed`: The refresh rate of the rainbow in seconds, e.g. `0.1`.

Optional:

- `size`: The number of pixels to turn on in a row, e.g. `3`. Defaults to `2` if no size is provided.
- `spacing`: The number of pixels to turn off in a row, e.g. `6`. Defaults to `3` if no size is provided.
- `reverse`: Optionally reverses the movement of the animation. Set to `True` to enable. Defaults to `False`.
- `step`: The steps to take through the colorwheel (0-255). A step of `1` means cycling through the entire colorwheel, a step of `2` means it cycles through every other possible value. Defaults to `8` if no step is provided.

Once created, you display the animation.

```
rainbow_chase = RainbowChase(pixels, speed=0.1, size=5, spacing=3)

while True:
    rainbow_chase.animate()
```



That's all there is to displaying a rainbow theatre chase animation using the LED Animation library! Let's take a look at the next animation.

RainbowComet

This is a rainbow version of a comet of a specified speed, with a dimming tail of specified length.

First, you import the `RainbowComet` module. See Import and Setup (https://adafru.it/LfT) for the rest of the necessary imports and pixel object creation.

```
from adafruit_led_animation.animation.rainbowcomet import RainbowComet
```

Next you create the `RainbowComet` animation. `RainbowComet` requires two arguments, and has an optional four more. You'll likely want to specify at least `tail_length`.

Required:

- `pixel_object`: The pixel object, e.g. `pixels`.
- `speed`: The speed of the comet in seconds, e.g. `0.1`.

Optional:

- `tail_length`: The length of the comet in pixels. Defaults to 25% of the length of the `pixel_object` if no length is provided. Automatically compensates for a minimum length of `2` and a maximum of the length of the `pixel_object`.
- `reverse`: Optionally reverses the movement of the animation. Set to `True` to enable. Defaults to `False`.
- `bounce`: Optionally "bounces" the comet along the strip by displaying it from the beginning of the strip to the end, and then reversing the movement once it reaches the end of the strip. Set to `True` to enable. Defaults to `False`.
- `colorwheel_offset`: Offset from the start of the colorwheel. Provide a value of `0`-`255` where `0` is red, `85` is blue and `170` is green, wrapping back to `255` being red. Defaults to `0`.

Once created, you display the animation.

```
rainbow_comet = RainbowComet(pixels, speed=0.1, tail_length=7, bounce=True)
```

```
while True:
    rainbow_comet.animate()
```

RainbowSparkle

This is a shifting rainbow that sparkles.

First, you import the `RainbowSparkle` module. See Import and Setup (https://adafru.it/LfT) for the rest of the necessary imports and pixel object creation.

```
from adafruit_led_animation.animation.rainbowsparkle import RainbowSparkle
```

Next you create the `RainbowSparkle` animation. `RainbowSparkle` requires two arguments, and has an optional five more.

Required:

- `pixel_object`: The pixel object, e.g. `pixels`.
- `speed`: The refresh rate of the rainbow in seconds, e.g. `0.1`.

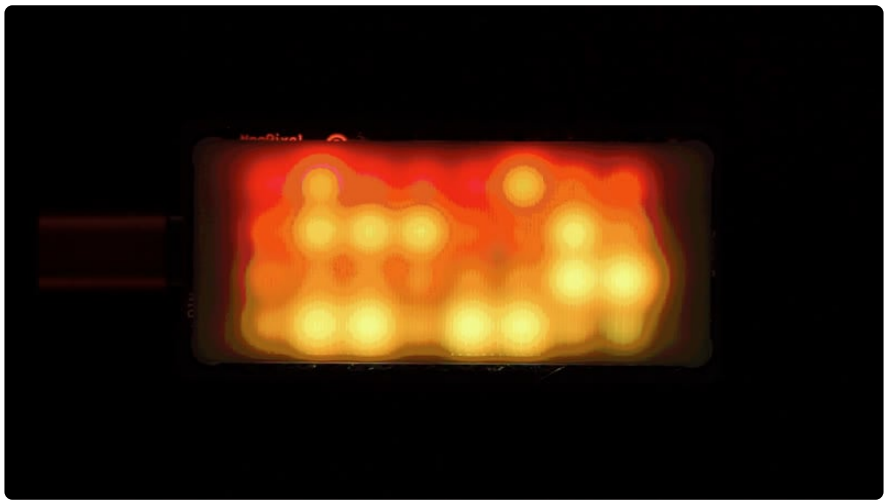
Optional:

- `period`: The period over which to cycle the rainbow in seconds, e.g. `2`. Defaults to 5 if no period is provided.
- `num_sparkles`: The number of sparkles. Defaults to 5% of the length of the `pixel_object`.
- `step`: The steps to take through the colorwheel (0-255). A step of `1` means cycling through the entire colorwheel, a step of `2` means it cycles through every other possible value. Defaults to `1` if no step is provided.

- `precompute_rainbow`: Precompute the rainbow which increases its speed, but uses more memory. Set to `False` to disable if you are running into memory limitations. Defaults to `True`.

Once created, you display the animation.

```
rainbow_sparkle = RainbowSparkle(pixels, speed=0.1, num_sparkles=15)

while True:
    rainbow_sparkle.animate()
```



That's how to display a rainbow sparkle animation using the LED Animation library! Next up: sparkles!

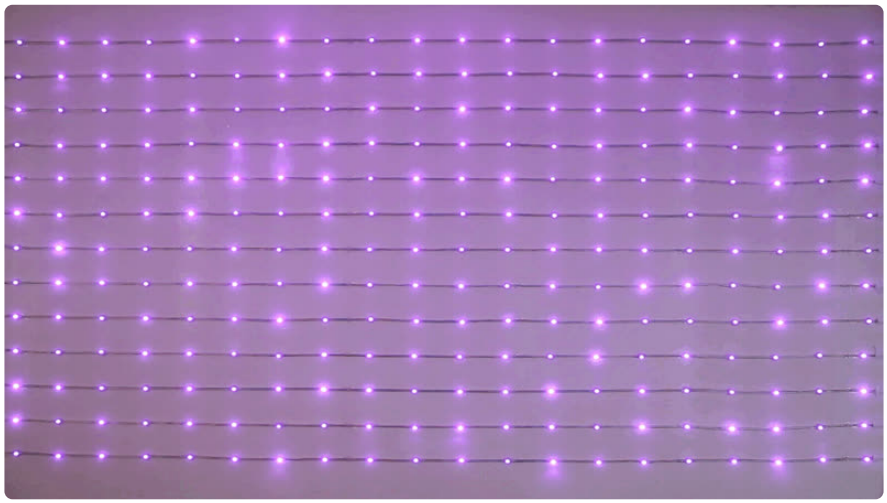
Full Example Code

This example runs all the rainbow animations in sequence.

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

"""
This example uses AnimationsSequence to display multiple animations in sequence, at
a five second
interval.

For NeoPixel FeatherWing. Update pixel_pin and pixel_num to match your wiring if
using
a different form of NeoPixels.

This example does not work on SAMD21 (M0) boards.
"""
import board
import neopixel

from adafruit_led_animation.animation.rainbow import Rainbow
from adafruit_led_animation.animation.rainbowchase import RainbowChase
from adafruit_led_animation.animation.rainbowcomet import RainbowComet
```

```
from adafruit_led_animation.animation.rainbowsparkle import RainbowSparkle
from adafruit_led_animation.sequence import AnimationSequence

# Update to match the pin connected to your NeoPixels
pixel_pin = board.D6
# Update to match the number of NeoPixels you have connected
pixel_num = 32

pixels = neopixel.NeoPixel(pixel_pin, pixel_num, brightness=0.5, auto_write=False)

rainbow = Rainbow(pixels, speed=0.1, period=2)
rainbow_chase = RainbowChase(pixels, speed=0.1, size=5, spacing=3)
rainbow_comet = RainbowComet(pixels, speed=0.1, tail_length=7, bounce=True)
rainbow_sparkle = RainbowSparkle(pixels, speed=0.1, num_sparkles=15)


animations = AnimationSequence(
    rainbow,
    rainbow_chase,
    rainbow_comet,
    rainbow_sparkle,
    advance_interval=5,
    auto_clear=True,
)

while True:
    animations.animate()
```

Sparkle



The CircuitPython LED Animation library includes a series of sparkle animations: **sparkle** and **sparklepulse**. This section will cover these animations and the available customizations for each of them.

Most animations will run individually on the SAMD21 (M0) microcontroller boards, but some combinations of animations and the most complex animations will not. Check out the FAQ for details. If you're interested in running all the animations, or many animations together, consider using at least a SAMD51 (M4) microcontroller.

Sparkle

This animation sparkles across all of the pixels in a single color.

First, you import the `Sparkle` module and a color for it. See Import and Setup (https://adafru.it/LfT) for the rest of the necessary imports and pixel object creation.

```
from adafruit_led_animation.animation.sparkle import Sparkle
from adafruit_led_animation.color import AMBER
```

Next you create the `Sparkle` animation. `Sparkle` requires two arguments, and has an optional third.

Required:

- `pixel_object`: The pixel object, e.g. `pixels`.
- `speed`: The refresh rate of the sparkle in seconds, e.g. `0.05`.
- `color`: The color to display, e.g. `AMBER`. Can also be a color tuple, e.g. `(255, 0, 0)`, or a hex color value, e.g. `0xFF0000`.

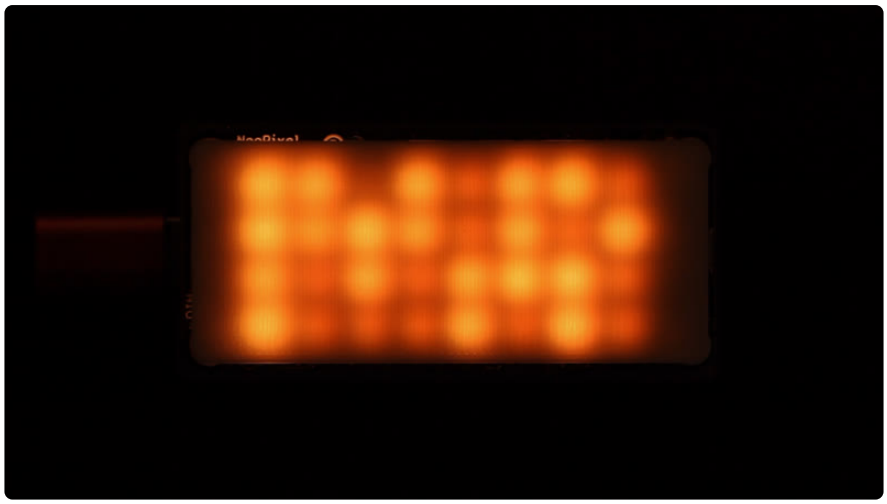
Optional:

- `num_sparkles`: The number of sparkles. Defaults to 5% of the length of the `pixel_object`.

Once created, you display the animation.

```
sparkle = Sparkle(pixels, speed=0.05, color=AMBER, num_sparkles=10)

while True:
    sparkle.animate()
```

It is difficult to make a faithful video of this effect because of video aliasing issues. The example video below has slower transitions than one sees in real life. See the video at the top of this page for a better example of what Sparkle looks like.



SparklePulse

This is a version of sparkle that uses pulse to determine the brightness of each pixel.

First, you import the `SparklePulse` module and a color for it. See Import and Setup (https://adafru.it/LfT) for the rest of the necessary imports and pixel object creation.

```
from adafruit_led_animation.animation.SparklePulse import SparklePulse
from adafruit_led_animation.color import JADE
```

Next you create the `SparklePulse` animation. `SparklePulse` requires three arguments, and has an optional three more.

Required:

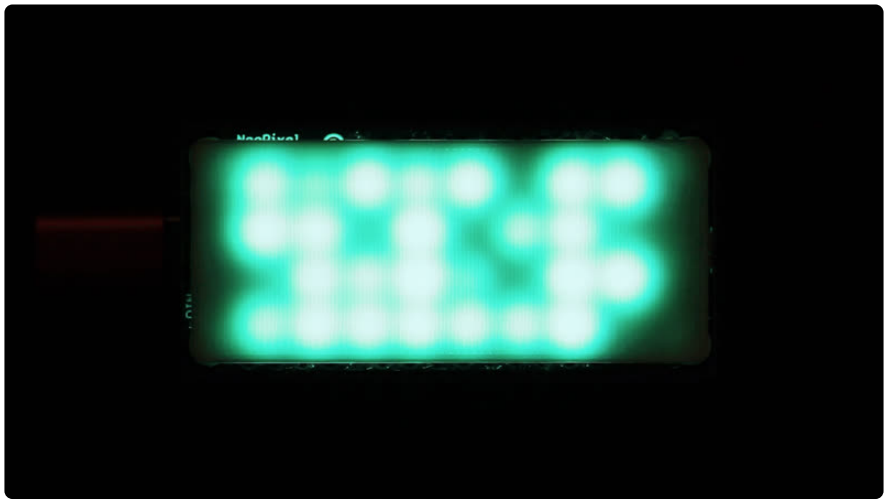
- `pixel_object`: The pixel object, e.g. `pixels`.
- `speed`: The speed of the pulse in seconds, e.g. `0.05`.
- `color`: The color to display, e.g. `JADE`. Can also be a color tuple, e.g. `(255, 0, 0)`, or a hex color value, e.g. `0xFF0000`.

Optional:

- `period`: The number of seconds over which to pulse the LEDs. Defaults to `5` if no period is provided.
- `max_intensity`: The maximum intensity to pulse. Provide a value between `0` and `1.0`. Defaults to `1`.
- `min_intensity`: The minimum intensity to pulse. Provide a value between `0` and `1.0`. Defaults to `0`.

Once created, you display the animation.

```
sparkle_pulse = SparklePulse(pixels, speed=0.05, period=3, color=JADE)

while True:
    sparkle_pulse.animate()
```



Next we'll look at using the pixel mapping helpers to create a grid from a single LED strip.

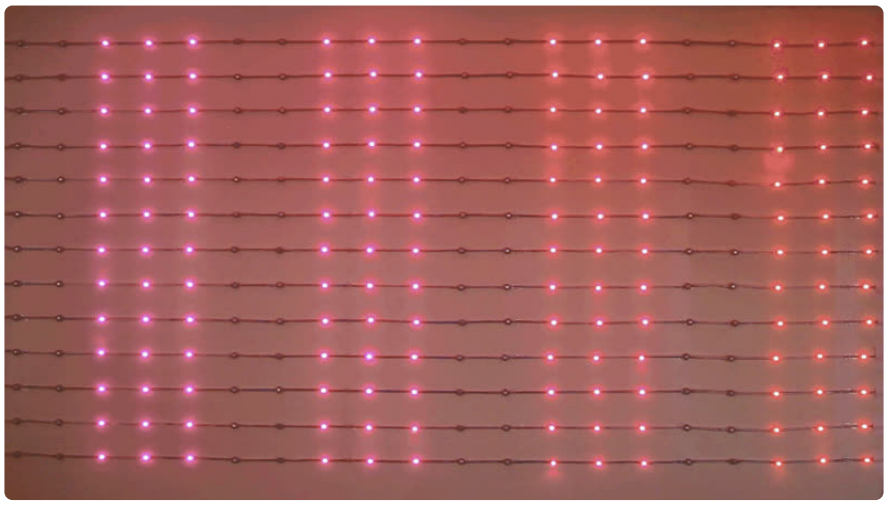
Full Example Code

This example displays the sparkle animations in sequence.

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

"""
This example uses AnimationsSequence to display multiple animations in sequence, at
a five second
interval.

For NeoPixel FeatherWing. Update pixel_pin and pixel_num to match your wiring if
using
a different form of NeoPixels.
"""
import board
import neopixel

from adafruit_led_animation.animation.sparkle import Sparkle
from adafruit_led_animation.animation.sparklepulse import SparklePulse
from adafruit_led_animation.sequence import AnimationSequence
from adafruit_led_animation.color import AMBER, JADE

# Update to match the pin connected to your NeoPixels
pixel_pin = board.D6
# Update to match the number of NeoPixels you have connected
pixel_num = 32

pixels = neopixel.NeoPixel(pixel_pin, pixel_num, brightness=0.5, auto_write=False)
```

```
sparkle = Sparkle(pixels, speed=0.05, color=AMBER, num_sparkles=10)
sparkle_pulse = SparklePulse(pixels, speed=0.05, period=3, color=JADE)

animations = AnimationSequence(
    sparkle,
    sparkle_pulse,
    advance_interval=5,
    auto_clear=True,
)

while True:
    animations.animate()
```

Pixel Mapping



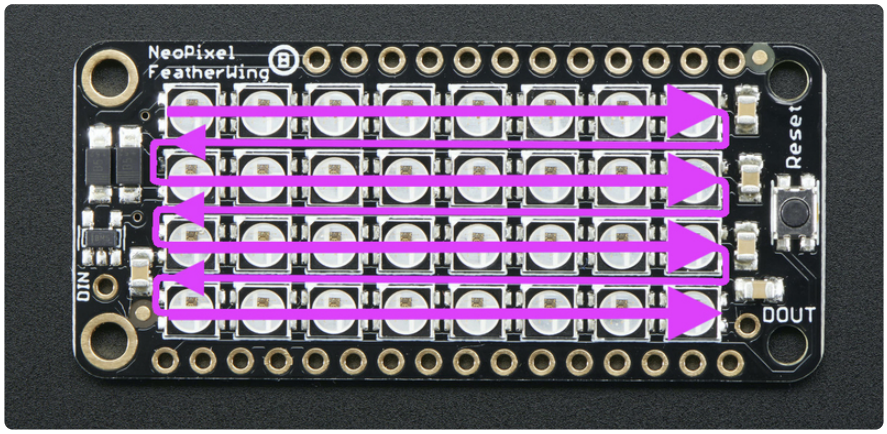
The CircuitPython LED Animation library includes pixel mapping helpers to make it super simple to treat a single LED strip as a grid. This is handy when you are using an LED matrix that is essentially one strip in series, or if you have a series of strips arranged in a grid. This section will go over the basics of the pixel mapping feature of the library and show you how to use it with animations.

Most animations will run individually on the SAMD21 (M0) microcontroller boards, but some combinations of animations, the most complex animations, and complex pixel mapping will not. Check out the FAQ for details. If you're interested in running all the animations, many animations together, or a complex pixel map, consider using at least a SAMD51 (M4) microcontroller.

LED Matrices

Many LED matrices look like a matrix at first glance, but they're actually a strip of pixels arranged as a grid. The NeoPixel FeatherWing is a set of NeoPixels made up of 32 pixels that are arranged in an 8x4 pixel grid. It's arranged left to right, beginning in the upper left corner near the "NeoPixel FeatherWing" silk print, and ending in the

bottom right near the DOUT pin, with each row beginning on the left and ending on the right.



If you want to see the pixel arrangement, try animating a comet with the initial pixel object. It will follow the pixel "strip". But what if you want to treat it as a matrix to display animations across it horizontally or vertically? `PixelMap` has helpers to do exactly that.

PixelMap

The `PixelMap` helper enables you to treat a strip or strips of LEDs as a grid for animation purposes. It also works great with LED matrices that are actually a strip of LEDs arranged in a matrix, such as the NeoPixel FeatherWing. The example on this page uses the NeoPixel FeatherWing, but should be quick to adapt to any grid or matrix.

First, you'll need to import the `helper` module. You'll also import the other required libraries, a number of animations and colors for some of them, and `AnimationSequence`.

```
import board
import neopixel
from adafruit_led_animation.animation.comet import Comet
from adafruit_led_animation.animation.rainbowcomet import RainbowComet
from adafruit_led_animation.animation.rainbowchase import RainbowChase
from adafruit_led_animation.animation.chase import Chase
from adafruit_led_animation.animation.rainbow import Rainbow
from adafruit_led_animation.sequence import AnimationSequence
from adafruit_led_animation import helper
from adafruit_led_animation.color import PURPLE, JADE, AMBER
```

Next you create the initial pixel object. This is identical to the pixel object used in all the previous examples - it sets up the pixels for use by the code.

If you're using some other NeoPixel form factor, update `pixel_pin` and `pixel_num` to match your NeoPixel setup. However, be aware that this example is designed for 32 pixels in a 8x4 matrix, and will require other changes to run properly if using a different setup.

```
pixel_pin = board.D6
pixel_num = 32
pixels = neopixel.NeoPixel(pixel_pin, pixel_num, brightness=0.2, auto_write=False)
```

Next you're going to create two new pixel objects using the `PixelMap` and `horizontal_strip_gridmap` helpers. One will be to create grid on which to animate vertically, and the other to animate horizontally.

PixelMap has two grid options for creating a grid: `vertical_lines` and `horizontal_lines`. The first generates a pixel map of vertical lines on a strip arranged in a grid, and the second generates a pixel map of horizontal lines on a strip arranged in a grid. Both of these helpers have four required arguments.

- `pixel_object`: The initial pixel object, e.g. `pixels`.
- `width`: The width of the grid, e.g. `8`.
- `height`: The height of the grid, e.g. `4`.
- `gridmap`: A function to map x and y coordinates to the grid, e.g. `horizontal_strip_gridmap` or `vertical_strip_gridmap`.

As the NeoPixel FeatherWing is arranged horizontally, you'll be using the `horizontal_strip_gridmap` helper. It has one required argument and one optional argument.

Required:

- `width`: The grid width in pixels, e.g. `8`.

Optional:

- `alternating`: Whether or not the lines in the grid are running alternating directions in a zigzag. Defaults to `True`.

The NeoPixel FeatherWing lines do not run in alternating directions.

```
pixel_wing_vertical = helper.PixelMap.vertical_lines(
    pixels, 8, 4, helper.horizontal_strip_gridmap(8, alternating=False)
)
```

```
pixel_wing_horizontal = helper.PixelMap.horizontal_lines(
    pixels, 8, 4, helper.horizontal_strip_gridmap(8, alternating=False)
)
```

Now that you've created these pixel objects, you can use them with the animations, the same way you used the initial pixel object.

To create a purple comet that is the width of the grid and animates top to bottom, you would create a comet animation as follows:

```
comet_h = Comet(
    pixel_wing_horizontal, speed=0.1, color=PURPLE, tail_length=3, bounce=True
)
```

To create an amber comet that is the height of the grid and animates left to right, you would create a comet as follows:

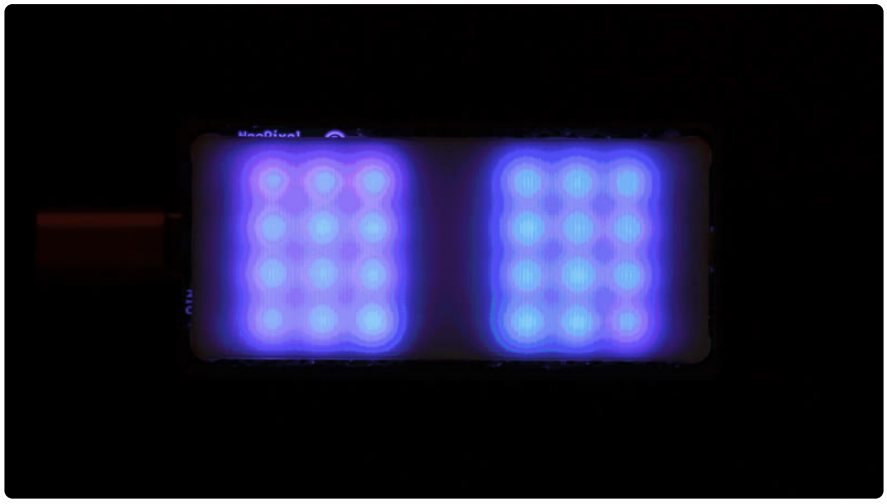
```
comet_v = Comet(pixel_wing_vertical, speed=0.1, color=AMBER, tail_length=6,
bounce=True)
```

To create a jade chase animation that animates from top to bottom:

```
chase_h = Chase(pixel_wing_horizontal, speed=0.1, size=3, spacing=6, color=JADE)
```

To create a rainbow chase animation that animates from left to right:

```
rainbow_chase_v = RainbowChase(
    pixel_wing_vertical, speed=0.1, size=3, spacing=2, wheel_step=8
)
```

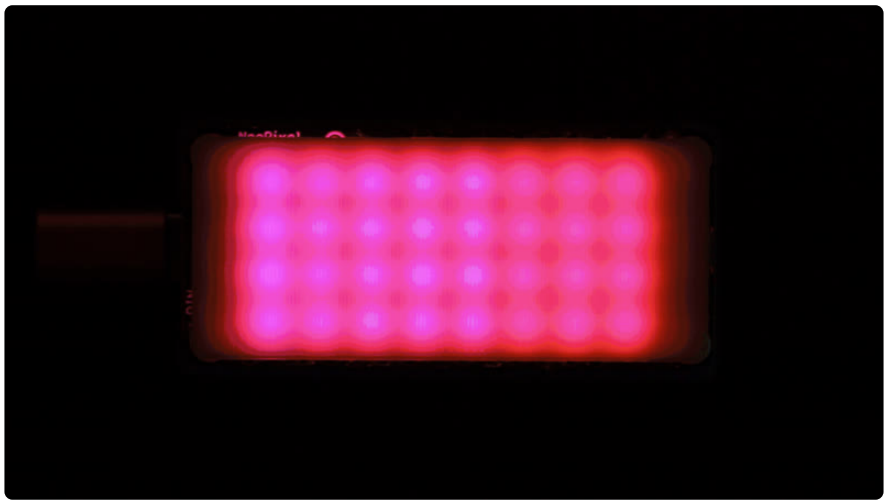


To create a rainbow comet that animates across the grid from left to right:

```
rainbow_comet_v = RainbowComet(
    pixel_wing_vertical, speed=0.1, tail_length=7, bounce=True
)
```

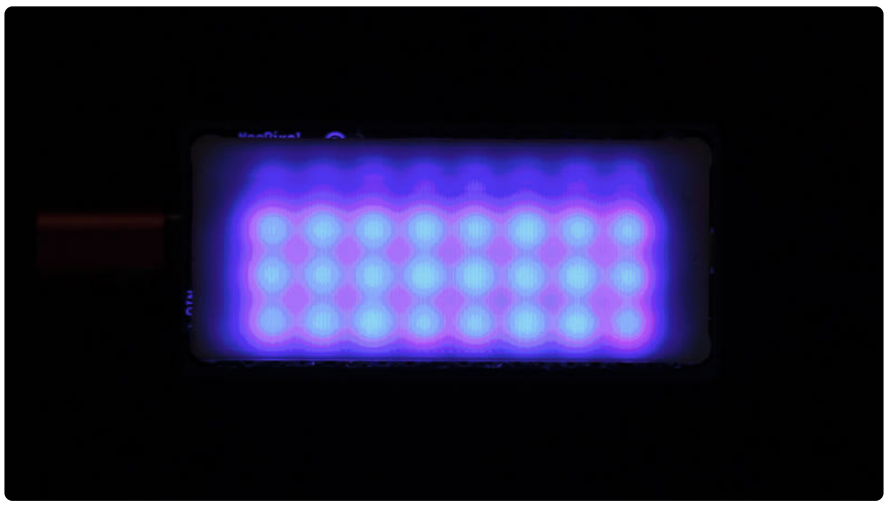
To create a rainbow that cycles across the grid vertically:

```
rainbow_v = Rainbow(pixel_wing_vertical, speed=0.1, period=2)
```



And finally, to create a rainbow chase that animates from top to bottom:

```
rainbow_chase_h = RainbowChase(pixel_wing_horizontal, speed=0.1, size=3, spacing=3)
```



Then you would display these animations the same way you did previously.

```
animations = AnimationSequence(
    rainbow_v,
    comet_h,
    rainbow_comet_v,
    chase_h,
    rainbow_chase_v,
    comet_v,
    rainbow_chase_h,
    advance_interval=5,
)

while True:
    animations.animate()
```

The same follows for any other animation. Some animations do not make sense to use with these helpers like blink, colorcycle, sparkle and pulse, as they use all the LEDs and the arrangement is irrelevant.

Now you can create animations that display horizontally and vertically across a grid!

Full Example Code

This example displays five different animations horizontally and vertically across a grid.

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

"""
This example shows usage of the PixelMap helper to easily treat a single strip as a
horizontal or
vertical grid for animation purposes.

For NeoPixel FeatherWing. Update pixel_pin and pixel_num to match your wiring if
using
a different form of NeoPixels. Note that if you are using a number of pixels other
than 32, you
will need to alter the PixelMap values as well for this example to work.

This example does not work on SAMD21 (M0) boards.
"""
import board
import neopixel

from adafruit_led_animation.animation.comet import Comet
from adafruit_led_animation.animation.rainbowcomet import RainbowComet
from adafruit_led_animation.animation.rainbowchase import RainbowChase
from adafruit_led_animation.animation.chase import Chase
from adafruit_led_animation.animation.rainbow import Rainbow
from adafruit_led_animation.sequence import AnimationSequence
from adafruit_led_animation import helper
from adafruit_led_animation.color import PURPLE, JADE, AMBER

# Update to match the pin connected to your NeoPixels
pixel_pin = board.D6
# Update to match the number of NeoPixels you have connected
pixel_num = 32
```

```
pixels = neopixel.NeoPixel(pixel_pin, pixel_num, brightness=0.5, auto_write=False)

pixel_wing_vertical = helper.PixelMap.vertical_lines(
    pixels, 8, 4, helper.horizontal_strip_gridmap(8, alternating=False)
)
pixel_wing_horizontal = helper.PixelMap.horizontal_lines(
    pixels, 8, 4, helper.horizontal_strip_gridmap(8, alternating=False)
)

comet_h = Comet(
    pixel_wing_horizontal, speed=0.1, color=PURPLE, tail_length=3, bounce=True
)
comet_v = Comet(pixel_wing_vertical, speed=0.1, color=AMBER, tail_length=6,
bounce=True)
chase_h = Chase(pixel_wing_horizontal, speed=0.1, size=3, spacing=6, color=JADE)
rainbow_chase_v = RainbowChase(
    pixel_wing_vertical, speed=0.1, size=3, spacing=2, step=8
)
rainbow_comet_v = RainbowComet(
    pixel_wing_vertical, speed=0.1, tail_length=7, bounce=True
)
rainbow_v = Rainbow(pixel_wing_vertical, speed=0.1, period=2)
rainbow_chase_h = RainbowChase(pixel_wing_horizontal, speed=0.1, size=3, spacing=3)

animations = AnimationSequence(
    rainbow_v,
    comet_h,
    rainbow_comet_v,
    chase_h,
    rainbow_chase_v,
    comet_v,
    rainbow_chase_h,
    advance_interval=5,
)

while True:
    animations.animate()
```

Animation Group

The CircuitPython LED Animation library includes an animation group helper that enables you to synchronize groups of animations. This section will walk through the basics of the animation group feature of the library and show you how to use it with animations and pixel objects.

Most animations will run individually on the SAMD21 (M0) microcontroller boards, but some combinations of animations, the most complex animations, and animation groups will not. Check out the FAQ for details. If you're interested in running all the animations, many animations together, or an animation group, consider using at least a SAMD51 (M4) microcontroller.

AnimationGroup

The `AnimationGroup` helper enables you to keep multiple animations in sync, whether or not the same animation or pixel object is used. It can be used with multiple animations or pixel objects, including pixel subsets. The example on this page is written for Circuit Playground Bluefruit and a 30-pixel NeoPixel LED strip, connected to pad A1.

First, you'll need to import the `AnimationGroup` module. You'll also import the other required libraries, the CircuitPlayground library, a number of animations and colors for them, and `AnimationSequence`.

```
import board
import neopixel
from adafruit_circuitplayground import cp
from adafruit_led_animation.animation.blink import Blink
from adafruit_led_animation.animation.comet import Comet
from adafruit_led_animation.animation.chase import Chase
from adafruit_led_animation.group import AnimationGroup
from adafruit_led_animation.sequence import AnimationSequence
import adafruit_led_animation.color as color
```

Next you'll create the pixel object for the strip, and specifically set the brightness for the Circuit Playground Bluefruit NeoPixels to `0.5`.

```
strip_pixels = neopixel.NeoPixel(board.A1, 30, brightness=0.5, auto_write=False)
cp.pixels.brightness = 0.5
```

Then you create an animation sequence. But this time, instead of simply adding animations, you're also going to add animation groups.

This section goes over each animation group individually, however this code is designed to run all of them in a single example. Do not try to use these code snippets alone. See the end of the page for the full example.

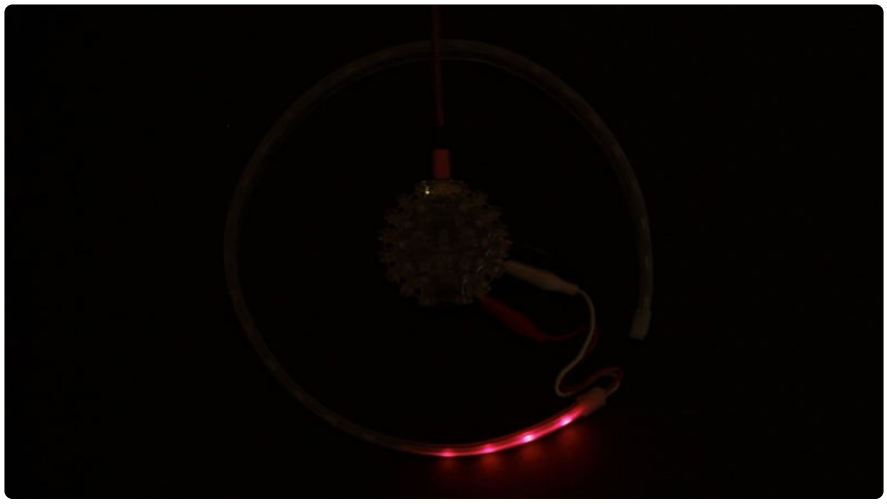
The first group is made up of the same animation on both the CPB and the strip, but we'll set each animation to a different speed. Then, we'll set `sync=True`. This means that when the animations are displayed, the different speed of the second animation is ignored, and the speed of the two animations is synced to the speed specified in the first one.

```
animations = AnimationSequence(
    AnimationGroup(
        Blink(cp.pixels, 0.5, color.CYAN),
        Blink(strip_pixels, 3.0, color.AMBER),
        sync=True,
```

```
    ),
[...]  # Means there's code below here in this code block.
```

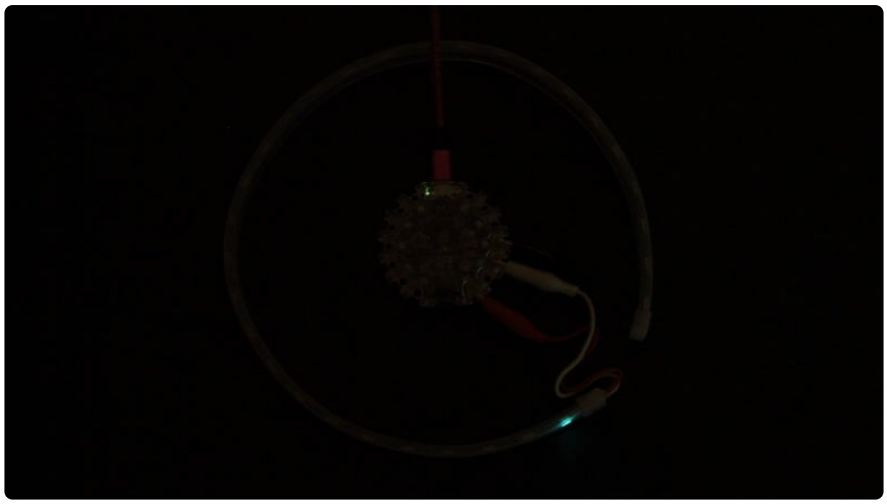
The second group is also made up of the same animation on both, and we set each one to a different speed. This time, we won't sync them.

```
[...]  # Means there's code above here in this code block.
    AnimationGroup(
        Comet(cp.pixels, 0.1, color.MAGENTA, tail_length=5),
        Comet(strip_pixels, 0.01, color.MAGENTA, tail_length=15),
    ),
[...]
```



The third group is made up of two different animations, one on the CPB and one on the strip. It displays two different animations on two different pixel objects simultaneously.

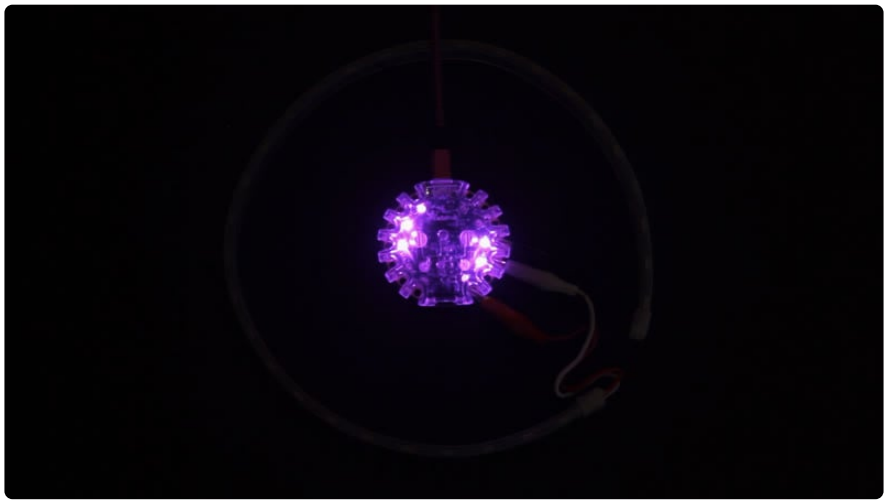
```
[...]
    AnimationGroup(
        Blink(cp.pixels, 0.5, color.JADE),
        Comet(strip_pixels, 0.05, color.TEAL, tail_length=15),
    ),
[...]
```



And finally, you include two animations in the sequence that will display sequentially, first on the CPB and then on the strip.

The advance interval is set to 3 seconds, and `auto_clear` and `auto_reset` are set to `True`.

```
[...]
    Chase(cp.pixels, 0.05, size=2, spacing=3, color=color.PURPLE),
    Chase(strip_pixels, 0.05, size=2, spacing=3, color=color.PURPLE),
    advance_interval=3.0,
    auto_clear=True,
    auto_reset=True,
)
```



You display the animations the same way you have in the previous sections.

```
while True:
    animations.animate()
```

That's all there is to using AnimationGroup to display and synchronise groups of animations using the LED Animation library!

Full Example Code

This example uses animation groups to display multiple animations simultaneously.

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

"""
This example shows three different ways to use AnimationGroup: syncing two
animations, displaying
two animations at different speeds, and displaying two animations sequentially,
across two separate
pixel objects such as the built-in NeoPixels on a Circuit Playground Bluefruit and
a NeoPixel strip.

This example is written for Circuit Playground Bluefruit and a 30-pixel NeoPixel
strip connected to
pad A1. It does not work on Circuit Playground Express.
"""
import board
import neopixel
from adafruit_circuitplayground import cp

from adafruit_led_animation.animation.blink import Blink
from adafruit_led_animation.animation.comet import Comet
from adafruit_led_animation.animation.chase import Chase

from adafruit_led_animation.group import AnimationGroup
from adafruit_led_animation.sequence import AnimationSequence

from adafruit_led_animation import color

strip_pixels = neopixel.NeoPixel(board.A1, 30, brightness=0.5, auto_write=False)
cp.pixels.brightness = 0.5

animations = AnimationSequence(
    # Synchronized to 0.5 seconds. Ignores the second animation setting of 3
seconds.
    AnimationGroup(
        Blink(cp.pixels, 0.5, color.CYAN),
        Blink(strip_pixels, 3.0, color.AMBER),
        sync=True,
    ),
    # Different speeds
    AnimationGroup(
        Comet(cp.pixels, 0.1, color.MAGENTA, tail_length=5),
        Comet(strip_pixels, 0.01, color.MAGENTA, tail_length=15),
    ),
    # Different animations
    AnimationGroup(
        Blink(cp.pixels, 0.5, color.JADE),
        Comet(strip_pixels, 0.05, color.TEAL, tail_length=15),
    ),
    # Sequential animations on the built-in NeoPixels then the NeoPixel strip
    Chase(cp.pixels, 0.05, size=2, spacing=3, color=color.PURPLE),
    Chase(strip_pixels, 0.05, size=2, spacing=3, color=color.PURPLE),
    advance_interval=3.0,
    auto_clear=True,
    auto_reset=True,
)

while True:
    animations.animate()
```

FAQs

These are the answers to some frequently asked questions regarding the CircuitPython LED Animation library.

Does the LED Animation library run on the SAMD21 microcontroller?

Technically, yes.

However, the entire library does not fit on SAMD21 non-Express boards. Any SAMD21-based microcontroller that does not have external flash available is considered a SAMD21 non-Express board, such as Trinket or NeoTrinkey. If you want to run LED Animations on SAMD21 non-Express boards, you must load only the parts of the library you intend to use.

Further, due to the memory limitations of the SAMD21, it is not possible to run all of the animations available in the LED Animation library. **The following animations will not run:**

- `rainbow_sparkle`
- `sparkle_pulse`

All animations **not listed above** will work standalone on the SAMD21.

It is not possible to run a significant number of animations together in sequence. Simpler animations can be run together. For example, you can use `AnimationSequence` to run `blink` and `chase` together in sequence. Adding more animations to the sequence, or adding more complicated animations to the sequence may fail. If you intend to run multiple animations, consider using a SAMD51 based microcontroller board or similar.

Animation groups do not run on the SAMD21.

On a SAMD21 non-Express board, why does my animation slow down if I leave it running for a while?

The LED Animation library uses `time.monotonic()` for animation timing. This allows for the animations to be non-blocking (https://adafru.it/BIT), meaning you are able to do other things in your code while animating your LEDs. See this link (https://adafru.it/

BIT) for more details - but, basically, at any given point in time, `time.monotonic()` is equal to the number seconds since your board was last power-cycled. (The soft-reboot that occurs with the auto-reload when you save changes to your CircuitPython code, or enter and exit the REPL, does not start it over.)

Due to the limitations of CircuitPython on a SAMD21 (M0) non-Express microcontroller board, the `time.monotonic()` value begins to lose accuracy after about an hour (1.165 hours to be exact) has passed. It is like a clock that functions initially, but after running for an hour, only ticks every two seconds in two second intervals, and after another two hours, ticks every four seconds in four second intervals, and so on. Using this device to keep track of time in seconds would be quite frustrating! You can hard-reset the board manually to resolve this - but you would have to do this each time it reaches the loss of accuracy to keep your animation running properly. It is simpler to reset your board using code.

This example uses CircuitPython to reset the board every time an hour passes since the last time the board was power-cycled. Save the following as **code.py** to your **CIRCUITPY** drive:

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

"""
This example shows how to reset the microcontroller to avoid the animation slowing
down over time
due to the limitations of CircuitPython for the SAMD21 (M0) microcontroller. The
example
animates a purple comet that bounces from end to end of the strip, and resets the
board if the
specified amount of time has passed since the board was last reset.

See this FAQ for details:
https://learn.adafruit.com/circuitpython-led-animations/faqs#on-the-samd21-non-
express-board-why-does-my-animation-slow-down-if-i-leave-it-running-for-a-
while-3074335-3

For QT Py Haxpress and a NeoPixel strip. Update pixel_pin and pixel_num to match
your wiring if
using a different board or form of NeoPixels.

This example will run on SAMD21 (M0) Express boards (such as Circuit Playground
Express or QT Py
Haxpress), but not on SAMD21 non-Express boards (such as QT Py or Trinket).
"""
import time
import microcontroller
import board
import neopixel

from adafruit_led_animation.animation.comet import Comet
from adafruit_led_animation.color import PURPLE

# Update to match the pin connected to your NeoPixels
pixel_pin = board.A3
# Update to match the number of NeoPixels you have connected
pixel_num = 30
```

```
pixels = neopixel.NeoPixel(pixel_pin, pixel_num, brightness=0.5, auto_write=False)

comet = Comet(pixels, speed=0.02, color=PURPLE, tail_length=10, bounce=True)

while True:
    comet.animate()

    if time.monotonic() > 3600:  # After an hour passes, reset the board.
        microcontroller.reset()  # pylint: disable=no-member
```

The relevant parts of this example are:

```
import time
import microcontroller

while True:
    if time.monotonic() &gt; 3600:
        microcontroller.reset()
```

The code above checks the value of `time.monotonic()`, and when it is greater than `3600` seconds, it resets the board. That's it! Include this with your animation code to keep your animations running at the speed you expect.

API Documentation

API Documentation (https://adafru.it/LcO)