Cheerlights Holiday Wreath with Animations

Created by Kattni Rembor

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

Last updated on 2024-11-29 10:04:58 AM EST

Table of Contents

Overview

Cheerlights (https://adafru.it/PaJ) is a project created by Hans Scharler that allows folks to sychronise LEDs across the world to one color using Twitter. It's an IoT project that listens for the word "cheerlights" on Twitter in real-time, and if there's a color name in the tweet, all of the projects connected to Cheerlights change to that color.

CircuitPython makes working with LEDs very simple. The Adafruit MagTag makes it easy to connect a strip of LED and connect to the Internet at the same time. Combined, they make doing WiFi-controlled LED projects a breeze, and the basic Cheerlights concept was no exception. However, why stop there....

Instead of simply lighting up the LEDs to the particular color, why not display an LED animation in that color? CircuitPython LED Animations turns Cheerlights into animated LED fun!

This guide will show you how to put together a festive wreath with your Adafruit MagTag and an LED strip. Then, you'll learn how to get CircuitPython set up on your MagTag, and load the necessary libraries needed to run this code. You'll save the

code to your MagTag, and before you know it, you'll be delighted by colorful LED animations.

This project requires quite a bit of power between the network connectivity and the LEDs, so it is suggested that you power it from a wall plug using an appropriate USB type C cable or adapter.

Parts

Adafruit MagTag - 2.9" Grayscale E-Ink WiFi Display

The Adafruit MagTag combines the new ESP32-S2 wireless module and a 2.9" grayscale E-Ink display to make a low-power IoT display that can show data on its screen even when power...

https://www.adafruit.com/product/4800

Mini Magnet Feet for RGB LED Matrices (Pack of 4)

Got a glorious RGB Matrix project you want to mount and display in your workspace or home? If you have one of the matrix panels listed below, you'll need a pack of these...

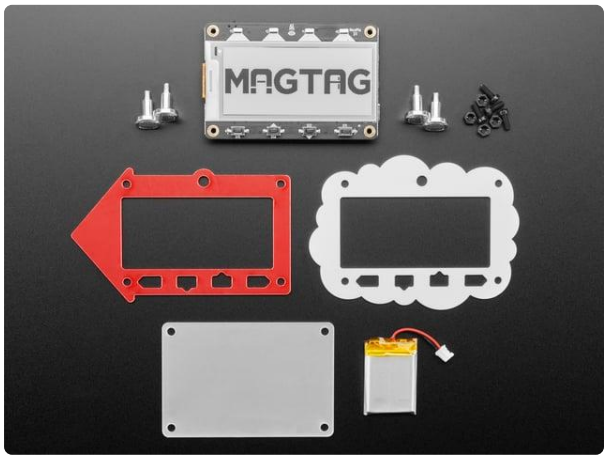
https://www.adafruit.com/product/4631

Adafruit NeoPixel LED Strip with 3-pin JST Connector - 1 meter

Plug in and glow, this Adafruit NeoPixel LED Strip with JST PH Connector has 30 total LEDs and is 1 meter long, in classy Adafruit...

https://www.adafruit.com/product/4801

Alternately, instead of purchasing the MagTag and Mini Magnet Feet separately, you could purchase the MagTag starter kit, which includes the magnet feet. Either way, you still need to purchase the LED strip separately.



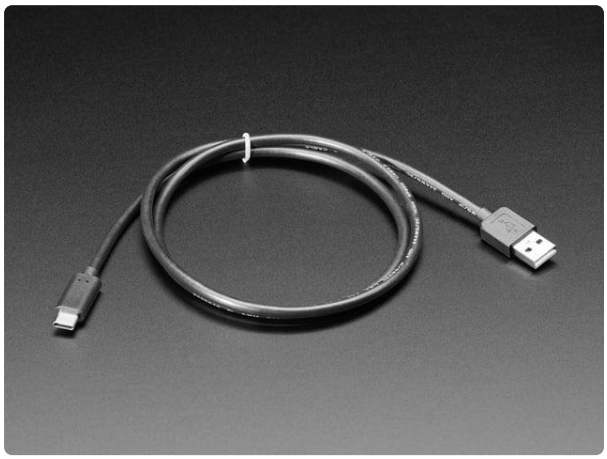
Adafruit MagTag Starter Kit - ADABOX017 Essentials

The Adafruit MagTag combines the new ESP32-S2 wireless module and a 2.9" grayscale E-Ink display to make a low-power IoT display that can show data on its screen...

https://www.adafruit.com/product/4819

Options for Power

- **Plug it into a wall plug using an appropriate cable or adapter.** One possible combination is shown below.



USB Type A to Type C Cable - approx 1 meter / 3 ft long

As technology changes and adapts, so does Adafruit. This USB Type A to Type C cable will help you with the transition to USB C, even if you're still...

https://www.adafruit.com/product/4474

5V 2A Switching Power Supply w/ USB-A Connector

Our 5V 2A USB power adapter is the perfect choice for powering single-board computers like Raspberry Pi, BeagleBone, or anything else that's power-hungry!This adapter was...

https://www.adafruit.com/product/1994

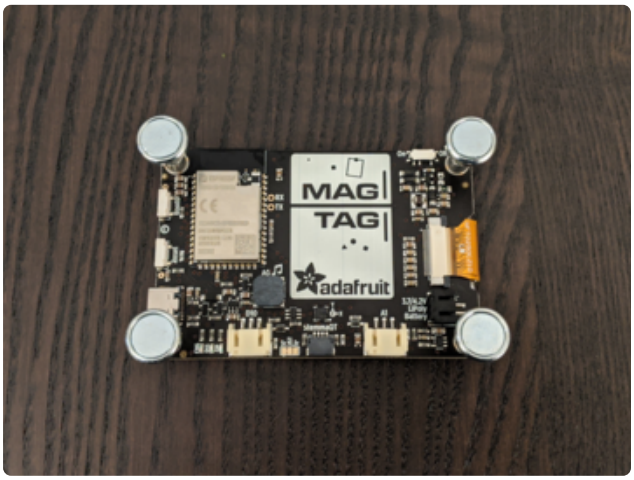
Additional Parts

- **Wreath** or other festive item to hold LEDs and MagTag. You can be creative here!
- **Zip-ties** or other appropriate mounting option to attach the LEDs and MagTag to the chosen festive item.

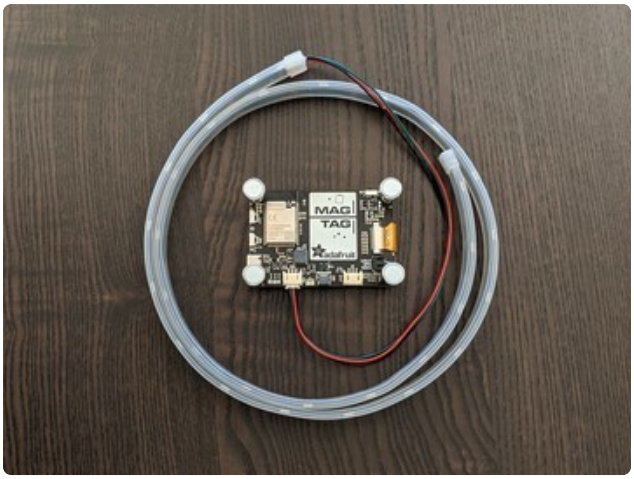
Assembly

This project is super easy to put together. Let's get started!

Gather together the necessary parts.



Attach the mini magnet feet to the MagTag by screwing them into the standoffs in the corners on the back.



Plug the **JST connector on the LED strip** into **JST port on the MagTag** labeled **D10**.

Lay the strip out how you want to attach it. This will give you an idea of how many zip ties you'll need, and where you'll want to attach them.

Start by **securing the two ends together snugly**. This will make securing the rest of the strip much easier. Then **place a zip tie intermittently around the strip** on the rest of the wreath. You can leave them relatively loose so you have some ability to move the strip around if necessary.

Secure the MagTag by nestling it into the wreath and wrapping a few of the wreath bits around the magnet feet, or if desired, you can use zip ties.

Fluff the wreath to hide the strip.

Use a wall power supply with enough current to power the project and a USB C cable - plug these into the MagTag.

Install CircuitPython

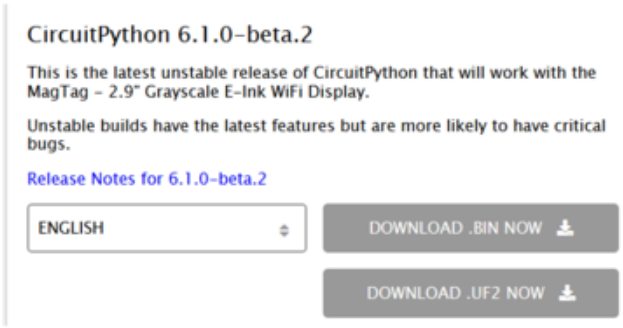
CircuitPython (https://adafru.it/tB7) is a derivative of MicroPython (https://adafru.it/BeZ) designed to simplify experimentation and education on low-cost microcontrollers. It makes it easier than ever to get prototyping by requiring no upfront desktop software downloads. Simply copy and edit files on the **CIRCUITPY** drive to iterate.

Set Up CircuitPython

Follow the steps to get CircuitPython installed on your MagTag.

Download the latest CircuitPython for your board from circuitpython.org

https://adafru.it/OBd



Click the link above and download the latest .BIN and .UF2 file

(depending on how you program the ESP32S2 board you may need one or the other, might as well get both)

Download and save it to your desktop (or wherever is handy).

Plug your MagTag into your computer using a known-good USB cable.

A lot of people end up using charge-only USB cables and it is very frustrating! So make sure you have a USB cable you know is good for data sync.

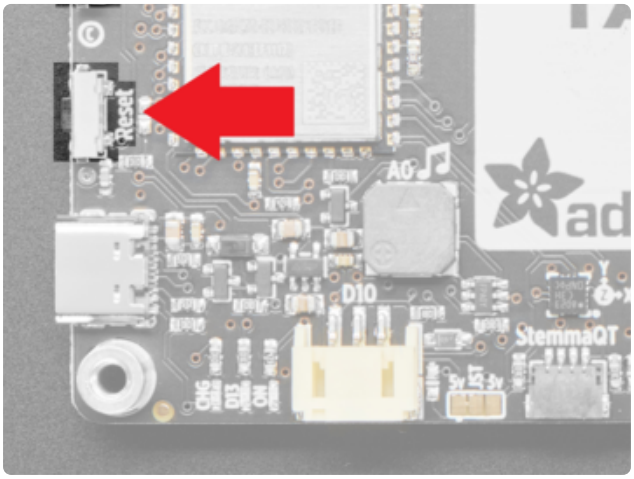
Option 1 - Load with UF2 Bootloader

This is by far the easiest way to load CircuitPython. **However it requires your board has the UF2 bootloader installed. Some early boards do not (we hadn't written UF2 yet!) - in which case you can load using the built in ROM bootloader.**

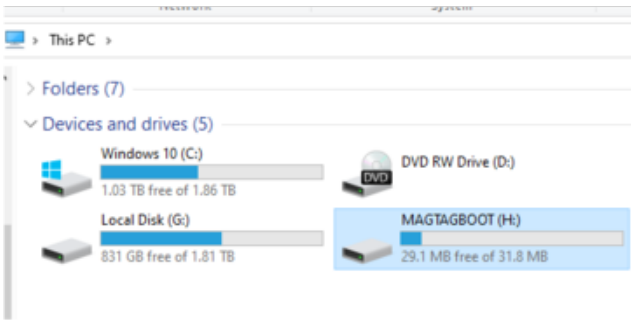
Still, try this first!

Try Launching UF2 Bootloader

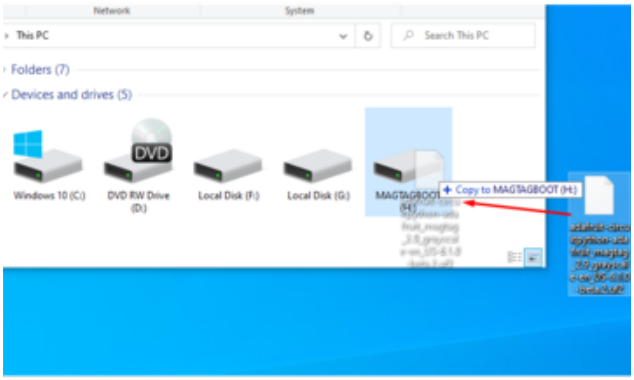
Loading CircuitPython by drag-n-drop UF2 bootloader is the easier way and we recommend it. If you have a MagTag where the front of the board is black, your MagTag came with UF2 already on it.



Launch UF2 by **double-clicking** the Reset button (the one next to the USB C port). You may have to try a few times to get the timing right.

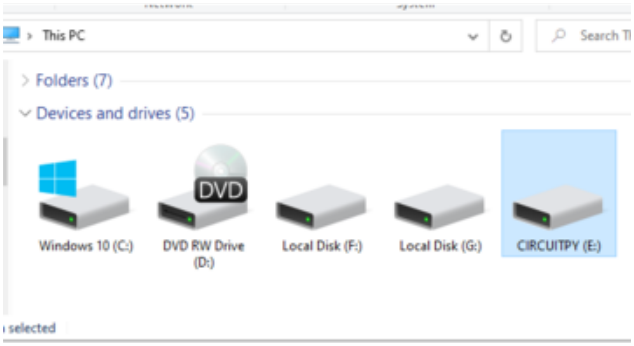


If the UF2 bootloader is installed, you will see a new disk drive appear called **MAGTAGBOOT**



Copy the **UF2** file you downloaded at the first step of this tutorial onto the **MAGTAGBOOT** drive

If you're using Windows and you get an error at the end of the file copy that says **Error from the file copy, Error 0x800701B1: A device which does not exist was specified.** You can ignore this error, the bootloader sometimes disconnects without telling Windows, the install completed just fine and you can continue. If its really annoying, you can also upgrade the bootloader (the latest version of the UF2 bootloader fixes this warning) (https://adafru.it/Pfk)

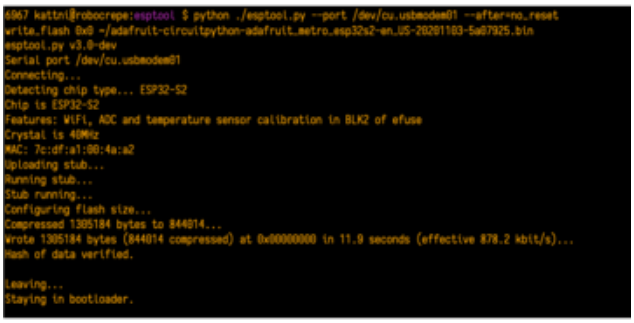


Your board should auto-reset into CircuitPython, or you may need to press reset. A **CIRCUITPY** drive will appear. You're done! Go to the next pages.

Option 2 - Use esptool to load BIN file

If you have an original MagTag with while soldermask on the front, we didn't have UF2 written for the ESP32S2 yet so it will not come with the UF2 bootloader.

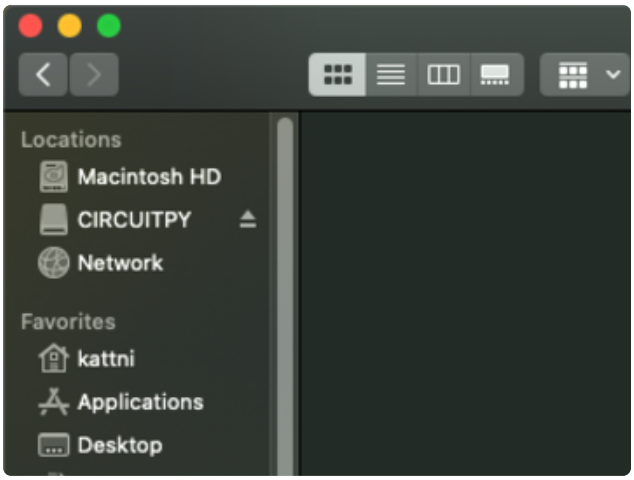
You can upload with **esptool** to the ROM (hardware) bootloader instead!



Follow the initial steps found in the Run esptool and check connection section of the ROM Bootloader page (https://adafru.it/OBc) to verify your environment is set up, your board is successfully connected, and which port it's using.

In the final command to write a binary file to the board, replace the port with your port, and replace "firmware.bin" with the the file you downloaded above.

The output should look something like the output in the image.



Press reset to exit the bootloader.

Your **CIRCUITPY** drive should appear!

You're all set! Go to the next pages.

Option 3 - Use Chrome Browser To Upload BIN file

If for some reason you cannot get esptool to run, you can always try using the Chrome-browser version of esptool we have written. This is handy if you don't have Python on your computer, or something is really weird with your setup that makes esptool not run (which happens sometimes and isn't worth debugging!) You can follow along on the Web Serial ESPTool (https://adafru.it/Pdq) page and either load the UF2 bootloader and then come back to Option 1 on this page, or you can download the CircuitPython BIN file directly using the tool in the same manner as the bootloader.

CircuitPython Internet Test

One of the great things about the ESP32 is the built-in WiFi capabilities. This page covers the basics of getting connected using CircuitPython.

The first thing you need to do is update your **code.py** to the following. Click the **Download Project Bundle** button below to download the necessary libraries and the **code.py** file in a zip file. Extract the contents of the zip file, and copy the **entire lib folder** and the **code.py** file to your **CIRCUITPY** drive.

```
# SPDX-FileCopyrightText: 2020 Brent Rubell for Adafruit Industries
#
# SPDX-License-Identifier: MIT

import os
import ipaddress
import ssl
import wifi
import socketpool
import adafruit_requests

# URLs to fetch from
TEXT_URL = "http://wifitest.adafruit.com/testwifi/index.html"
JSON_QUOTES_URL = "https://www.adafruit.com/api/quotes.php"
JSON_STARS_URL = "https://api.github.com/repos/adafruit/circuitpython"

print("ESP32-S2 WebClient Test")

print(f"My MAC address: {[hex(i) for i in wifi.radio.mac_address]}")

print("Available WiFi networks:")
for network in wifi.radio.start_scanning_networks():
    print("\t%s\t\tRSSI: %d\tChannel: %d" % (str(network.ssid, "utf-8"),
                                              network.rssi, network.channel))
wifi.radio.stop_scanning_networks()

print(f"Connecting to {os.getenv('CIRCUITPY_WIFI_SSID')}")
wifi.radio.connect(os.getenv("CIRCUITPY_WIFI_SSID"),
os.getenv("CIRCUITPY_WIFI_PASSWORD"))
print(f"Connected to {os.getenv('CIRCUITPY_WIFI_SSID')}")
print(f"My IP address: {wifi.radio.ipv4_address}")

ping_ip = ipaddress.IPv4Address("8.8.8.8")
ping = wifi.radio.ping(ip=ping_ip)

# retry once if timed out
if ping is None:
    ping = wifi.radio.ping(ip=ping_ip)

if ping is None:
    print("Couldn't ping 'google.com' successfully")
else:
    # convert s to ms
    print(f"Pinging 'google.com' took: {ping * 1000} ms")

pool = socketpool.SocketPool(wifi.radio)
requests = adafruit_requests.Session(pool, ssl.create_default_context())

print(f"Fetching text from {TEXT_URL}")
response = requests.get(TEXT_URL)
print("-" * 40)
print(response.text)
print("-" * 40)

print(f"Fetching json from {JSON_QUOTES_URL}")
response = requests.get(JSON_QUOTES_URL)
print("-" * 40)
print(response.json())
print("-" * 40)

print()
```

```
print(f"Fetching and parsing json from {JSON_STARS_URL}")
response = requests.get(JSON_STARS_URL)
print("-" * 40)
print(f"CircuitPython GitHub Stars: {response.json()['stargazers_count']}")
print("-" * 40)

print("Done")
```

Your **CIRCUITPY** drive should resemble the following.

To get connected, the next thing you need to do is update the **settings.toml** file.

The settings.toml File

We expect people to share tons of projects as they build CircuitPython WiFi widgets. What we want to avoid is people accidentally sharing their passwords or secret tokens and API keys. So, we designed all our examples to use a **settings.toml** file, that is on your **CIRCUITPY** drive, to hold secret/private/custom data. That way you can share your main project without worrying about accidentally sharing private stuff.

If you have a fresh install of CircuitPython on your board, the initial **settings.toml** file on your **CIRCUITPY** drive is empty.

To get started, you can update the **settings.toml** on your **CIRCUITPY** drive to contain the following code.

```
# SPDX-FileCopyrightText: 2023 Adafruit Industries
#
# SPDX-License-Identifier: MIT

# This is where you store the credentials necessary for your code.
# The associated demo only requires WiFi, but you can include any
# credentials here, such as Adafruit IO username and key, etc.
CIRCUITPY_WIFI_SSID = "your-wifi-ssid"
CIRCUITPY_WIFI_PASSWORD = "your-wifi-password"
```

This file should contain a series of Python variables, each assigned to a string. Each variable should describe what it represents (say `wifi_ssid`), followed by an

= (equals sign), followed by the data in the form of a Python string (such as `"my-wifi-password"` including the quote marks).

At a minimum you'll need to add/update your WiFi SSID and WiFi password, so do that now!

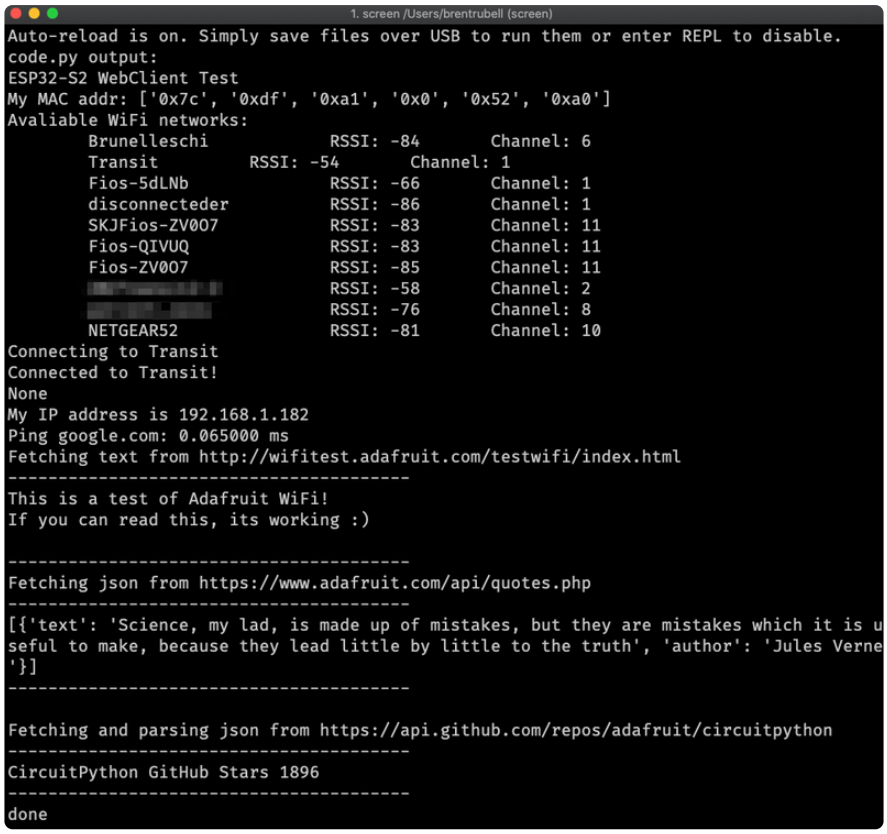
As you make projects you may need more tokens and keys, just add them one line at a time. See for example other tokens such as one for accessing GitHub or the Hackaday API. Other non-secret data like your timezone can also go here.

For the correct time zone string, look at http://worldtimeapi.org/timezones (https://adafru.it/EcP) and remember that if your city is not listed, look for a city in the same time zone, for example Boston, New York, Philadelphia, Washington DC, and Miami are all on the same time as New York.

Of course, don't share your **settings.toml** - keep that out of GitHub, Discord or other project-sharing sites.

> Don't share your settings.toml file! It has your passwords and API keys in it!

If you connect to the serial console, you should see something like the following:



In order, the example code...

Checks the ESP32's MAC address.

```
print(f"My MAC address: {[hex(i) for i in wifi.radio.mac_address]}")
```

Performs a scan of all access points and prints out the access point's name (SSID), signal strength (RSSI), and channel.

```
print("Available WiFi networks:")
for network in wifi.radio.start_scanning_networks():
    print("\t%s\t\tRSSI: %d\tChannel: %d" % (str(network.ssid, "utf-8"),
                                              network.rssi, network.channel))
wifi.radio.stop_scanning_networks()
```

Connects to the access point you defined in the **settings.toml** file, and prints out its local IP address.

```
print(f"Connecting to {os.getenv('WIFI_SSID')}")
wifi.radio.connect(os.getenv("WIFI_SSID"), os.getenv("WIFI_PASSWORD"))
print(f"Connected to {os.getenv('WIFI_SSID')}")
print(f"My IP address: {wifi.radio.ipv4_address}")
```

Attempts to ping a Google DNS server to test connectivity. If a ping fails, it returns `None`. Initial pings can sometimes fail for various reasons. So, if the initial ping is successful (`is not None`), it will print the echo speed in ms. If the initial ping fails, it

will try one more time to ping, and then print the returned value. If the second ping fails, it will result in `"Ping google.com: None ms"` being printed to the serial console. Failure to ping does not always indicate a lack of connectivity, so the code will continue to run.

```
ping_ip = ipaddress.IPv4Address("8.8.8.8")
ping = wifi.radio.ping(ip=ping_ip) * 1000
if ping is not None:
    print(f"Ping google.com: {ping} ms")
else:
    ping = wifi.radio.ping(ip=ping_ip)
    print(f"Ping google.com: {ping} ms")
```

The code creates a socketpool using the wifi radio's available sockets. This is performed so we don't need to re-use sockets. Then, it initializes a a new instance of the requests (https://adafru.it/E9o) interface - which makes getting data from the internet really really easy.

```
pool = socketpool.SocketPool(wifi.radio)
requests = adafruit_requests.Session(pool, ssl.create_default_context())
```

To read in plain-text from a web URL, call `requests.get` - you may pass in either a http, or a https url for SSL connectivity.

```
print(f"Fetching text from {TEXT_URL}")
response = requests.get(TEXT_URL)
print("-" * 40)
print(response.text)
print("-" * 40)
```

Requests can also display a JSON-formatted response from a web URL using a call to `requests.get`.

```
print(f"Fetching json from {JSON_QUOTES_URL}")
response = requests.get(JSON_QUOTES_URL)
print("-" * 40)
print(response.json())
print("-" * 40)
```

Finally, you can fetch and parse a JSON URL using `requests.get`. This code snippet obtains the `stargazers_count` field from a call to the GitHub API.

```
print(f"Fetching and parsing json from {JSON_STARS_URL}")
response = requests.get(JSON_STARS_URL)
print("-" * 40)
print(f"CircuitPython GitHub Stars: {response.json()['stargazers_count']}")
print("-" * 40)
```

OK you now have your ESP32 board set up with a proper **settings.toml** file and can connect over the Internet. If not, check that your **settings.toml** file has the right SSID and password and retrace your steps until you get the Internet connectivity working!

IPv6 Networking

Starting in CircuitPython 9.2, IPv6 networking is available on most Espressif wifi boards. Socket-using libraries like **adafruit_requests** and **adafruit_ntp** will need to be updated to use the new APIs and for now can only connect to services on IPv4.

IPv6 connectivity & privacy

IPv6 addresses are divided into many special kinds, and many of those kinds (like those starting with **FC**, **FD**, **FE**) are private or local; Addresses starting with other prefixes like **2002:** and **2001:** are globally routable. In 2024, far from all ISPs and home networks support IPv6 internet connectivity. For more info consult resources like Wikipedia (https://adafru.it/1a4z). If you're interested in global IPv6 connectivity you can use services like Hurricane Electric (https://adafru.it/1a4A) to create an "IPv6 tunnel" (free as of 2024, but requires expertise and a compatible router or host computer to set up)

It's also important to be aware that, as currently implemented by Espressif, there are privacy concerns especially when these devices operate on the global IPv6 network: The device's unique identifier (its EUI-64 or MAC address) is used by default as part of its IPv6 address. This means that the device identity can be tracked across multiple networks by any service it connects to.

Enable IPv6 networking

Due to the privacy consideration, IPv6 networking is not automatically enabled. Instead, it must be explicitly enabled by a call to `start_dhcp_client` with the `ipv6=True` argument specified:

```
wifi.start_dhcp_client(ipv6=True)
```

Check IP addresses

The read-only `addresses` property of the `wifi.radio` object holds all addresses, including IPv4 and IPv6 addresses:

```
&gt;&gt;&gt; wifi.radio.addresses
('FE80::7EDF:A1FF:FE00:518C', 'FD5F:3F5C:FE50:0:7EDF:A1FF:FE00:518C', '10.0.3.96')
```

The `wifi.radio.dns` servers can be IPv4 or IPv6:

```
&gt;&gt;&gt; wifi.radio.dns
('FD5F:3F5C:FE50::1',)
```

```
&gt;&gt;&gt; wifi.radio.dns = ("1.1.1.1",)
&gt;&gt;&gt; wifi.radio.dns
('1.1.1.1',)
```

Ping v6 networks

`wifi.radio.ping` accepts v6 addresses and names:

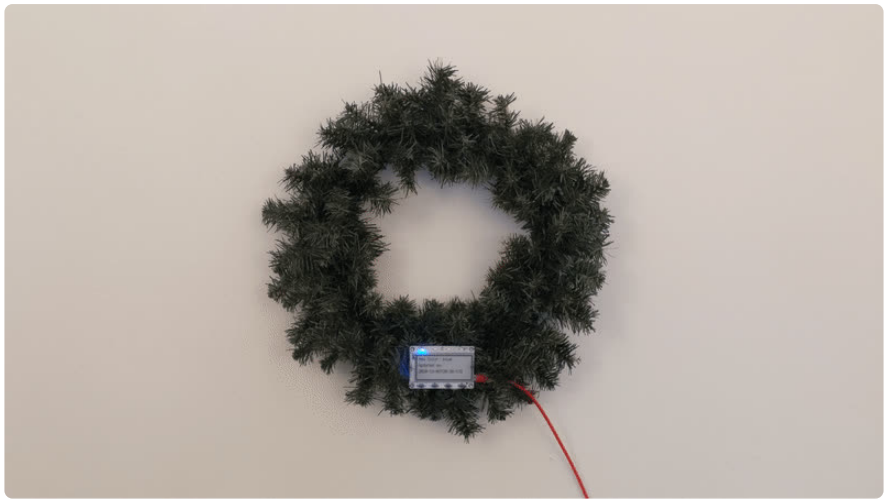
```
&gt;&gt;&gt; wifi.radio.ping("google.com")
0.043
&gt;&gt;&gt; wifi.radio.ping("ipv6.google.com")
0.048
```

Create & use IPv6 sockets

Use the address family `socket.AF_INET6`. After the socket is created, use methods like `connect`, `send`, `recfrom_into`, etc just like for IPv4 sockets. This code snippet shows communicating with a private-network NTP server; this IPv6 address will not work on your network:

```
&gt;&gt;&gt; ntp_addr = ("fd5f:3f5c:fe50::20e", 123)
&gt;&gt;&gt; PACKET_SIZE = 48
&gt;&gt;&gt;
&gt;&gt;&gt; buf = bytearray(PACKET_SIZE)
&gt;&gt;&gt; with socket.socket(socket.AF_INET6, socket.SOCK_DGRAM) as s:
...     s.settimeout(1)
...     buf[0] = 0b0010_0011
...     s.sendto(buf, ntp_addr)
...     print(s.recvfrom_into(buf))
...     print(buf)
...
48
(48, ('fd5f:3f5c:fe50::20e', 123))
bytearray(b'$\x01\x03\xeb\x00\x00\x00\x00\x00\x00\x00GGPS\x00\xeaA0h\x07s;
\xc0\x00\x00\x00\x00\x00\x00\x00\xeaA0n\xeb4\x82-\xeaA0n\xebAU\xb1')
```

Code



This project is coded using CircuitPython which makes it super simple to edit. There are a couple of customisations you can update to make this project fit your needs. If you also want to change the style and type of LED animations, there is a guide on using the CircuitPython LED Animation library (https://adafru.it/LZF) that covers all the features. This guide does not go into customising the animations.

This page will show you how to customise a couple of things in the code. However, before the code will run, you need to load the LED Animation library onto your **CIRCUITPY** drive.

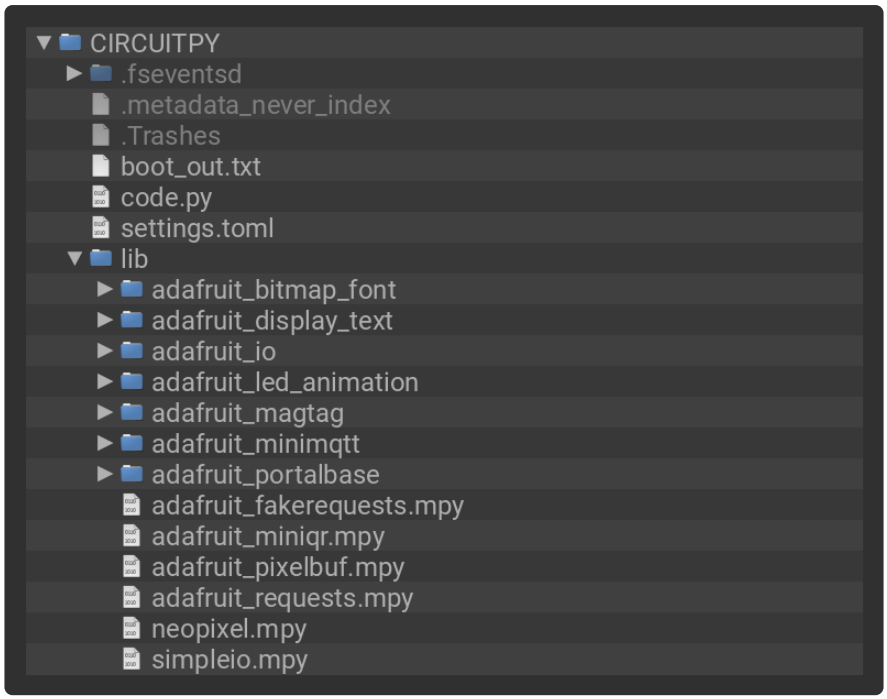
If you're having difficulty running this example, it could be because your MagTag CircuitPython firmware or library needs to be upgraded! Please be sure to follow https://learn.adafruit.com/adafruit-magtag/circuitpython to install the latest CircuitPython firmware and then also replace/update ALL the MagTag-specific libraries mentioned here https://learn.adafruit.com/adafruit-magtag/circuitpython-libraries-2

Install Code and Libraries

To use with CircuitPython, you need to first install a few libraries, into the lib folder on your **CIRCUITPY** drive. Then you need to update **code.py** with the example script.

Thankfully, we can do this in one go. In the example below, click the **Download Project Bundle** button below to download the necessary libraries and the **code.py** file in a zip file. Extract the contents of the zip file, open the directory **MagTag_Cheerlights_LED_Animations/** and then click on the directory that matches the version of CircuitPython you're using and copy the contents of that directory to your **CIRCUITPY** drive.

Your **CIRCUITPY** drive should now look similar to the following image:



Code

```
# SPDX-FileCopyrightText: 2020 Kattni Rembor, written for Adafruit Industries
#
# SPDX-License-Identifier: Unlicense
import time
import board
import neopixel
from adafruit_magtag.magtag import MagTag

from adafruit_led_animation.animation.comet import Comet
from adafruit_led_animation.animation.sparkle import Sparkle
from adafruit_led_animation.animation.chase import Chase
from adafruit_led_animation.animation.blink import Blink
from adafruit_led_animation.animation.pulse import Pulse
from adafruit_led_animation.sequence import AnimationSequence
from adafruit_led_animation.group import AnimationGroup
from adafruit_led_animation.color import RED, GREEN, BLUE, CYAN, WHITE,\
    OLD_LACE, PURPLE, MAGENTA, YELLOW, ORANGE, PINK

# ================ CUSTOMISATIONS ================
# The strip LED brightness, where 0.0 is 0% (off) and 1.0 is 100% brightness, e.g.
0.3 is 30%
strip_pixel_brightness = 1
# The MagTag LED brightness, where 0.0 is 0% (off) and 1.0 is 100% brightness, e.g.
0.3 is 30%.
magtag_pixel_brightness = 0.5

# The rate interval in seconds at which the Cheerlights data is fetched.
# Defaults to one minute (60 seconds).
refresh_rate = 60
# ================================================

# Set up where we'll be fetching data from
DATA_SOURCE = "http://api.thingspeak.com/channels/1417/field/1/last.json"
COLOR_LOCATION = ['field1']
DATE_LOCATION = ['created_at']

magtag = MagTag(
    url=DATA_SOURCE,
    json_path=(COLOR_LOCATION, DATE_LOCATION),
)
```

```
magtag.network.connect()

strip_pixels = neopixel.NeoPixel(board.D10, 30, brightness=strip_pixel_brightness)
magtag_pixels = magtag.peripherals.neopixels
magtag_pixels.brightness = magtag_pixel_brightness

# Cheerlights color
magtag.add_text(
    text_scale=2,
    text_position=(10, 15),
    text_transform=lambda x: "New Color: {}".format(x),  # pylint:
disable=unnecessary-lambda
)
# datestamp
magtag.add_text(
    text_scale=2,
    text_position=(10, 65),
    text_transform=lambda x: "Updated on:\n{}".format(x),  # pylint:
disable=unnecessary-lambda
)

timestamp = None
while True:
    if not timestamp or ((time.monotonic() - timestamp) > refresh_rate):  # Refresh
rate in seconds
        try:
            # Turn on the MagTag NeoPixels.
            magtag.peripherals.neopixel_disable = False

            # Fetch the color and datetime, and print it to the serial console.
            cheerlights_update_info = magtag.fetch()
            print("Response is", cheerlights_update_info)

            # Get the color from the fetched info for use below.
            cheerlights_color = cheerlights_update_info[0]

            # If red, blue or pink, do a comet animation in the specified color.
            if cheerlights_color in ('red', 'blue', 'pink'):
                if cheerlights_color == 'red':
                    comet_color = RED
                elif cheerlights_color == 'blue':
                    comet_color = BLUE
                elif cheerlights_color == 'pink':
                    comet_color = PINK
                animations = AnimationSequence(
                    AnimationGroup(
                        Comet(magtag_pixels, 0.3, color=comet_color, tail_length=3),
                        Comet(strip_pixels, 0.05, color=comet_color,
tail_length=15),
                    )
                )

            # If green or orange, do a pulse animation in the specified color.
            if cheerlights_color in ('green', 'orange'):
                if cheerlights_color == 'green':
                    pulse_color = GREEN
                elif cheerlights_color == 'orange':
                    pulse_color = ORANGE
                animations = AnimationSequence(
                    AnimationGroup(
                        Pulse(magtag_pixels, speed=0.1, color=pulse_color,
period=3),
                        Pulse(strip_pixels, speed=0.1, color=pulse_color, period=3),
                    )
                )

            # If oldlace, warmmwhite or magenta, do a sparkle animation in the
specified color.
            if cheerlights_color in ('oldlace', 'warmwhite', 'magenta'):
```

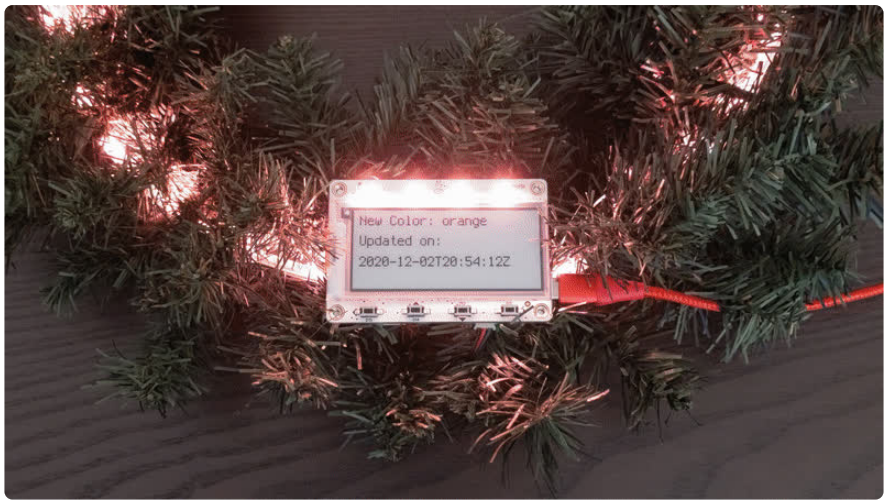
```
                if cheerlights_color in ('oldlace', 'warmwhite'):
                    sparkle_color = OLD_LACE
                elif cheerlights_color == 'magenta':
                    sparkle_color = MAGENTA
                animations = AnimationSequence(
                    AnimationGroup(
                        Sparkle(magtag_pixels, speed=0.1, color=sparkle_color,
num_sparkles=1),
                        Sparkle(strip_pixels, speed=0.01, color=sparkle_color,
num_sparkles=15),
                    )
                )

            # If cyan or purple, do a blink animation in the specified color.
            if cheerlights_color in ('cyan', 'purple'):
                if cheerlights_color == 'cyan':
                    blink_color = CYAN
                elif cheerlights_color == 'purple':
                    blink_color = PURPLE
                animations = AnimationSequence(
                    AnimationGroup(
                        Blink(magtag_pixels, speed=0.5, color=blink_color),
                        Blink(strip_pixels, speed=0.5, color=blink_color),
                    )
                )

            # If white or yellow, do a chase animation in the specified color.
            if cheerlights_color in ('white', 'yellow'):
                if cheerlights_color == 'white':
                    chase_color = WHITE
                elif cheerlights_color == 'yellow':
                    chase_color = YELLOW
                animations = AnimationSequence(
                    AnimationGroup(
                        Chase(magtag_pixels, speed=0.1, size=2, spacing=1,
color=chase_color),
                        Chase(strip_pixels, speed=0.1, size=3, spacing=2,
color=chase_color),
                    )
                )

                timestamp = time.monotonic()
            except (ValueError, RuntimeError, ConnectionError, OSError) as e:
                # Catch any random errors so the code will continue running.
                print("Some error occured, retrying! -", e)
    try:
        # This runs the animations.
        animations.animate()
    except NameError:
        # If Cheerlights adds a color not included above, the code would fail. This
allows it to
        # continue to retry until a valid color comes up.
        print("There may be a Cheerlights color not included above.")
```

Each time the MagTag fetches the current Cheerlights color, the current animation will pause. Once the information is fetched, the new animation will begin and the display will refresh. This is expected behavior. The following shows a transition from orange to white.



Customisations

There are a couple of things you can easily change in this code.

LED Brightness

You can set the brightness of both the MagTag LEDs and the strip separately. Depending on how you mounted the strip, you may want to decrease the brightness.

```
strip_pixel_brightness = 1
magtag_pixel_brightness = 0.5
```

NeoPixels can get really bright. The code defaults to `0.5`, or 50% brightness, for the MagTag LEDs, and `1`, or 100% brightness, for the strip. The strip cannot get any brighter than 100%, however you can increase the MagTag LED brightness number to make them brighter. You can decrease either of the numbers to make them dimmer, however, be aware that some of the colors look different when the LEDs are very dim.

Refresh Rate

You can also set the rate in seconds at which the MagTag fetches the Cheerlights data. Bear in mind that the animations pause during the data retrieval, so if you set the rate too low, your animations won't run for long between fetches. As well, it is not good for the display to refresh too frequently.

```
refresh_rate = 60
```

The `refresh_rate` defaults to 60 seconds, or one minute. You can increase or decrease this to fit your needs.

Walkthrough

Let's take a look at the code.

First, we set up where we'll be fetching the data from.

Cheerlights has three feeds available (https://adafru.it/PaK) - one that provides the hex value of the latest color and the update time, one that provides the name of the latest color and the update time, and one that contains the entire feed of hex and color names and update times. **This project uses the feed that provides the name of the latest color and update time because it does not require you to know hex colors, and makes the code easier to read.**

```
DATA_SOURCE = "http://api.thingspeak.com/channels/1417/field/1/last.json"
COLOR_LOCATION = ['field1']
DATE_LOCATION = ['created_at']
```

Then we initialise the MagTag library, including the information needed to fetch the Cheerlights data, and we tell the board to connect to the network.

```
magtag = MagTag(
    url=DATA_SOURCE,
    json_path=(COLOR_LOCATION, DATE_LOCATION),
)
magtag.network.connect()
```

Next, we set up the external NeoPixel strip, and set the brightness specified at the beginning of the code on both the strip NeoPixels and the MagTag NeoPixels.

```
strip_pixels = neopixel.NeoPixel(board.D10, 30, brightness=strip_pixel_brightness)
magtag_pixels = magtag.peripherals.neopixels
magtag_pixels.brightness = magtag_pixel_brightness
```

Then we set up the text that will be display on the screen. By using the `text_transform` option, we can set up the display text in the beginning, and update it each time we fetch new data in the loop.

```
magtag.add_text(
    text_scale=2,
    text_position=(10, 15),
    text_transform=lambda x: "New Color: {}".format(x),
)
magtag.add_text(
    text_scale=2,
    text_position=(10, 65),
    text_transform=lambda x: "Updated on:\n{}".format(x),
)
```

The first thing inside the loop is checking whether it's the first time the loop has started or if the `refresh_rate` seconds (set in the beginning of the code - defaults to 60 seconds) has passed.

```
if not timestamp or ((time.monotonic() - timestamp) &gt; refresh_rate):
```

The next block of code is wrapped in a `try`/`except` to avoid the code stopping if it experiences any errors with regards to fetching the data.

```
[...]
        try:
            [...]  # Main block of code is here.
        except (ValueError, RuntimeError) as e:
            print("Some error occured, retrying! -", e)
```

Inside, we first enable the NeoPixels by turning off `magtag.neopixel_disable`.

```
magtag.peripherals.neopixel_disable = False
```

Then we fetch the Cheerlights data, which returns a two-member list that includes the latest color, and the current date and time. For example, it appears as something like the following:

`['white', '2020-12-02T22:52:45Z']`.

We assign it to the `cheerlights_update_info` variable, and print that information to the serial console.

```
cheerlights_update_info = magtag.fetch()
print("Response is", cheerlights_update_info)
```

For the animations, we need only the color from the data we fetch. So, to make it easy to use the color, we assign the first member of the list (using `[0]`) to a variable called `cheerlights_color`.

```
cheerlights_color = cheerlights_update_info[0]
```

Then we begin using the colors to display animations. There are eleven colors available for Cheerlights, and five basic animations, so each animation applies to multiple colors. Each animation checks the `cheerlights_color` variable to see what the latest color is, and displays the matching animation. The same general format is used for every animation - therefore this will only cover the first animation.

This section of the code could be more succinct if the hex value feed was used. However, not everyone is familiar with hex colors, so this section is slightly longer-form to enhance simplicity and ease of understanding.

First, the code checks if the latest color is one that applies to that particular animation, in this case, is it red, blue or pink.

Then, it verifies the specific Cheerlights color, and sets the animation color to the appropriate color. In this case, for example, if the Cheerlights color is red, it sets `comet_color = RED`.

Last, it creates the animation, in this case the comet animation, along with any necessary animation settings. For more details on the animations or their settings, take a look at the CircuitPython LED Animation guide (https://adafru.it/LZF).

```
if cheerlights_color in ('red', 'blue', 'pink'):
    if cheerlights_color == 'red':
        comet_color = RED
    elif cheerlights_color == 'blue':
        comet_color = BLUE
    elif cheerlights_color == 'pink':
        comet_color = PINK
    animations = AnimationSequence(
        AnimationGroup(
            Comet(magtag_pixels, 0.3, color=comet_color, tail_length=3),
            Comet(strip_pixels, 0.05, color=comet_color, tail_length=15),
          )
        )
```

Finally, the code runs the animations. This code is also wrapped in a `try`/`except`. In the event that Cheerlights adds a new color, the code would fail - this ensures your code will continue and display the next time one of the current colors is fetched.

```
try:
    animations.animate()
except NameError:
    print("There may be a Cheerlights color not included above.")
```

That's all there is to displaying LED animations based on Cheerlights data!

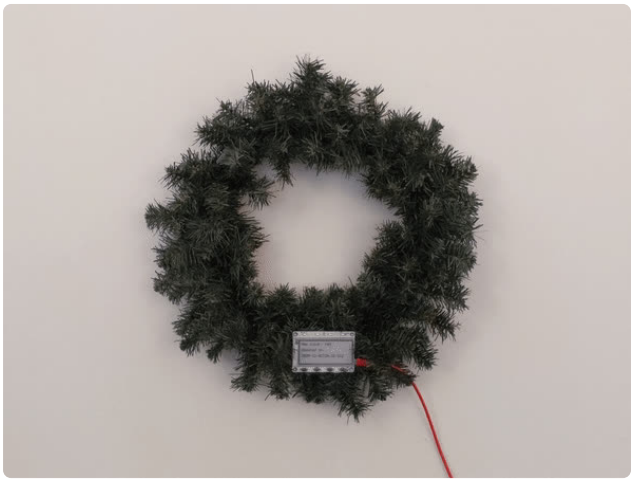
Cheerlights Animations

This project uses five different animations across the eleven available Cheerlights colors - each animation applies to multiple colors.

To activate an animation and color, tweet the color name in a message including the word "cheerlights" or send a color name and tag the @Cheerlights account.

The following is a look at one color of each animation, along with a list of the colors to which each animation applies.

Comet



The **comet** animation applies to **red**, **blue** and **pink**.

Pulse

The **pulse** animation applies to **green** and **orange**.

Sparkle

The **sparkle** animation applies to **oldlace**/**warmwhite** and **magenta**.

Blink

The **blink** animation applies to **cyan** and **purple**.

Chase

The **chase** animation applies to **white** and **yellow.**