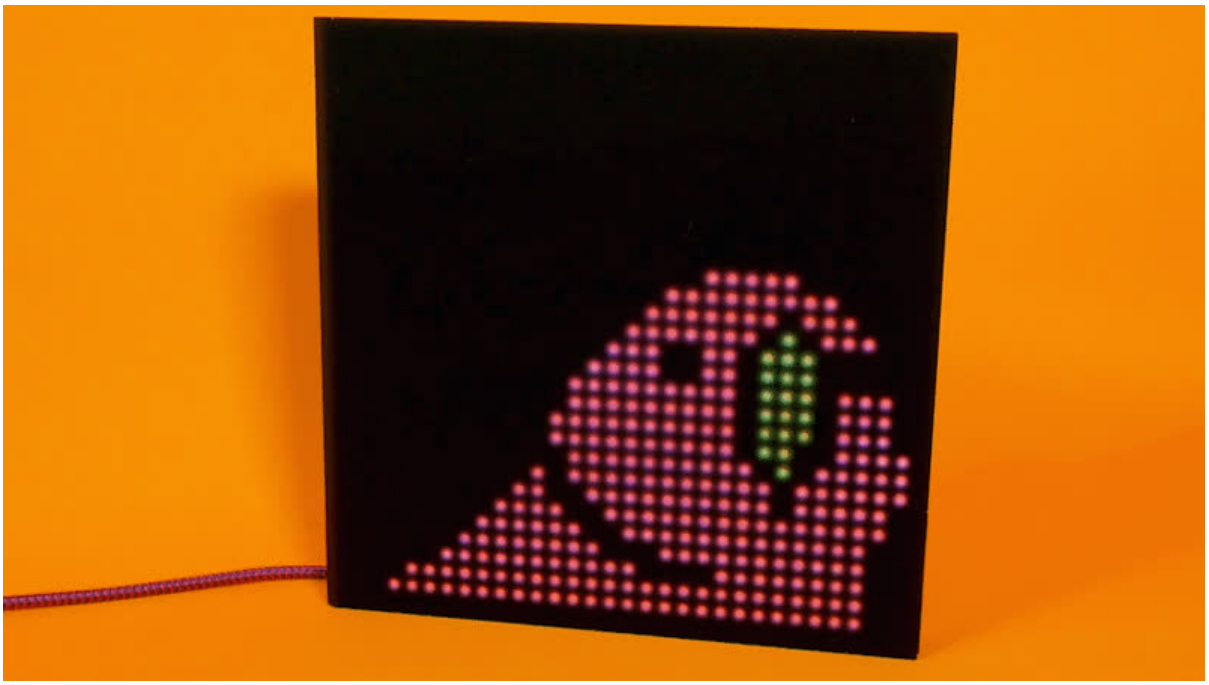
IoT Twitter Listener Party Parrot

Created by Liz Clark



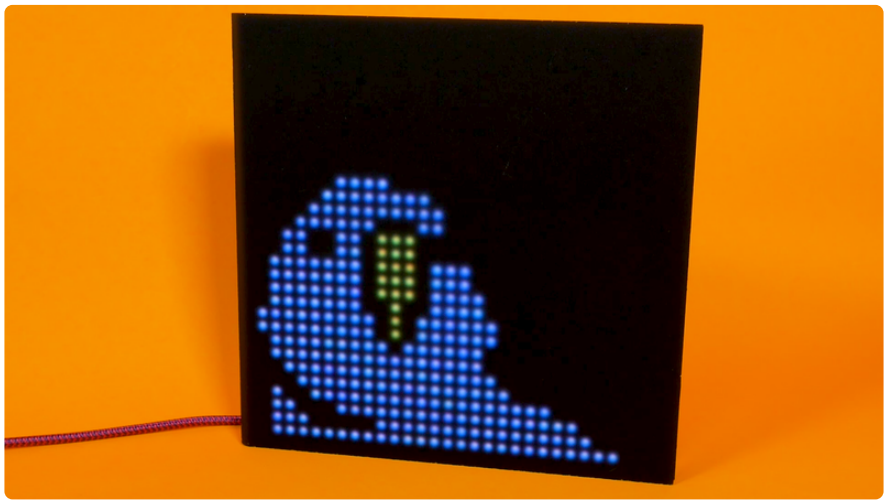
https://learn.adafruit.com/iot-twitter-listener-party-parrot

Last updated on 2025-03-25 10:00:38 AM EDT

Table of Contents

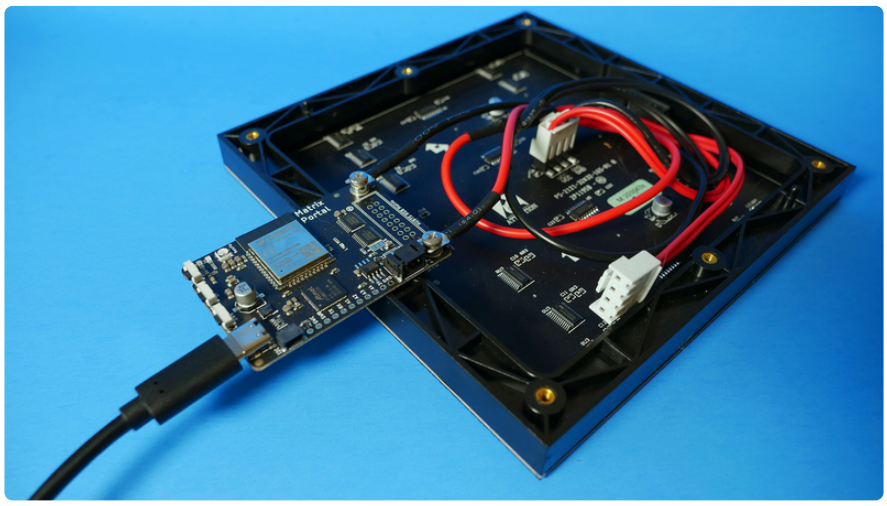
Overview

Create an IoT party parrot sprite animation that alerts you about new tweets that folks send you with the hashtag #partyparrot.

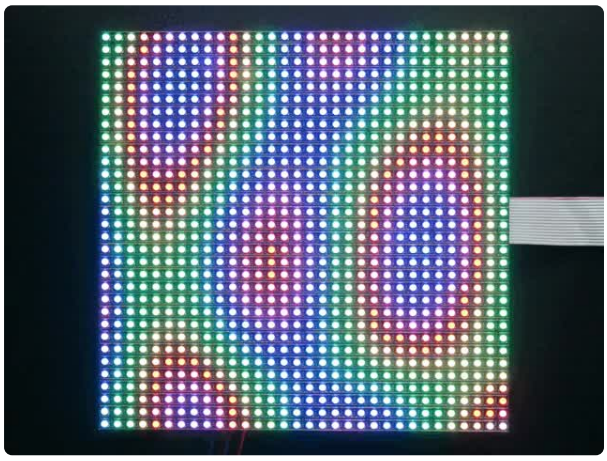


This project uses the Twitter API v2 to query tweets. You can fetch these tweets with the **adafruit_matrixportal** library and a MatrixPortal M4 plugged directly into your 32x32 RGB LED Matrix. The MatrixPortal M4 has a USB C port for power and data, which keeps the number of wires low. This makes for a nice and compact IoT project for your desk so that you never miss a virtual party.

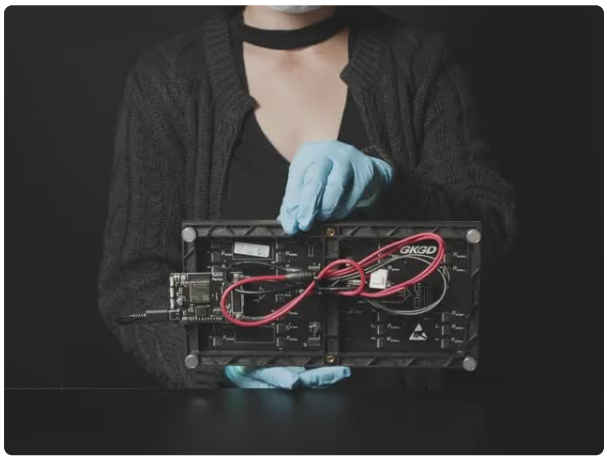
#partyparrot!



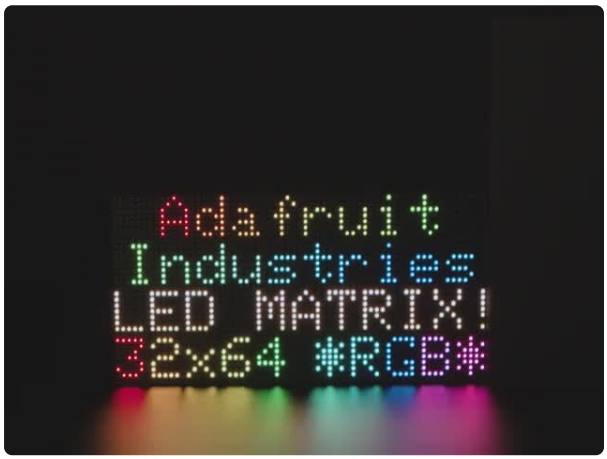
Parts List



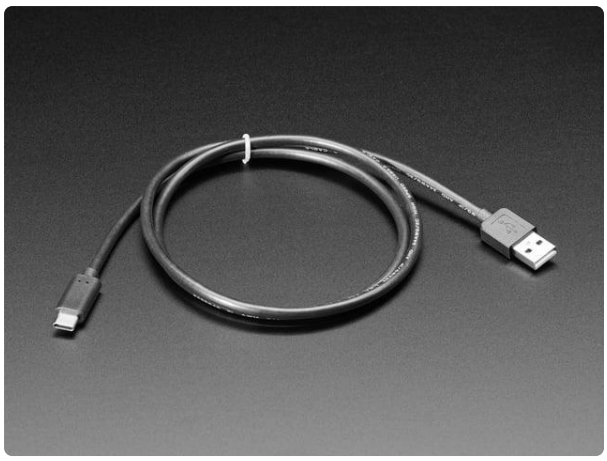
32x32 RGB LED Matrix Panel - 5mm Pitch
Bring a little bit of Times Square into your home with this sweet 32 x 32 square RGB LED matrix panel. These panels are normally used to make video walls, here in New York we see them...
https://www.adafruit.com/product/2026



Adafruit Matrix Portal - CircuitPython Powered Internet Display
Folks love our wide selection of RGB matrices and accessories, for making custom colorful LED displays... and our RGB Matrix Shields...
https://www.adafruit.com/product/4745



Black LED Diffusion Acrylic Panel 12" x 12" - 0.1" / 2.6mm thick
A nice whoppin' slab of some lovely black acrylic to add some extra diffusion to your LED Matrix project. This material is 2.6mm (0.1") thick and is made of special cast...
https://www.adafruit.com/product/4594



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

USB C to USB C Cable - USB 3.1 Gen 4 with E-Mark - 1 meter long

As technology changes and adapts, so does Adafruit! Rather than the regular USB A, this cable has USB C to USB C plugs!USB C is the latest...

https://www.adafruit.com/product/4199

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 Quick Start!

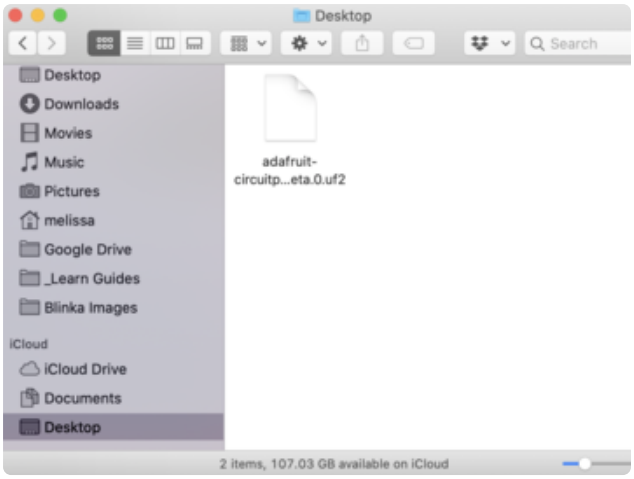
Follow this quick step-by-step for super-fast Python power :)

Download the latest version of CircuitPython for this board via circuitpython.org

https://adafru.it/Nte

Further Information

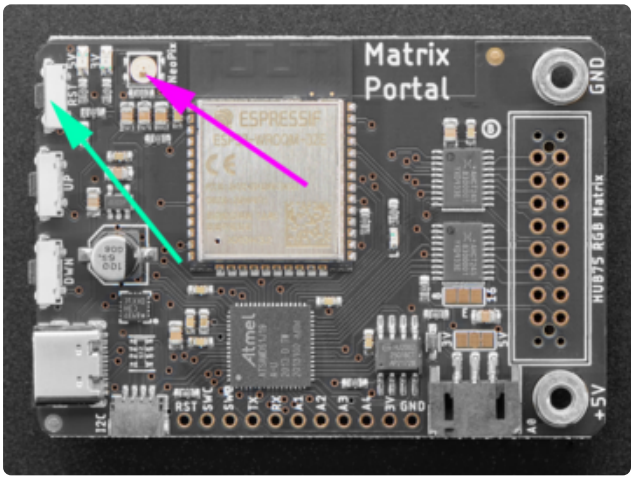
For more detailed info on installing CircuitPython, check out Installing CircuitPython (https://adafru.it/Amd).



Click the link above and download the latest UF2 file.

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

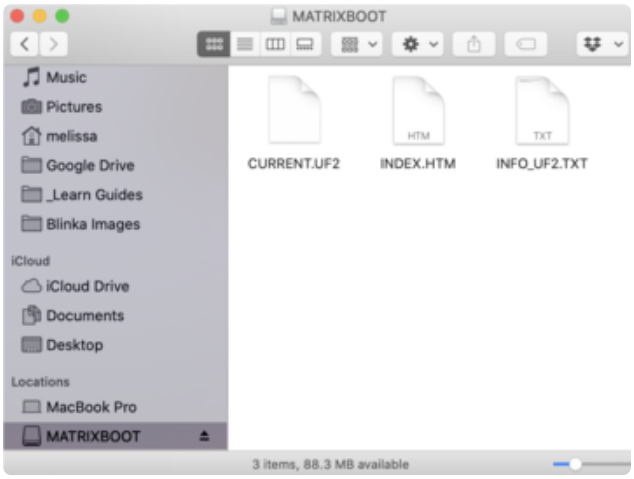
Plug your MatrixPortal M4 into your computer using a known-good USB cable.



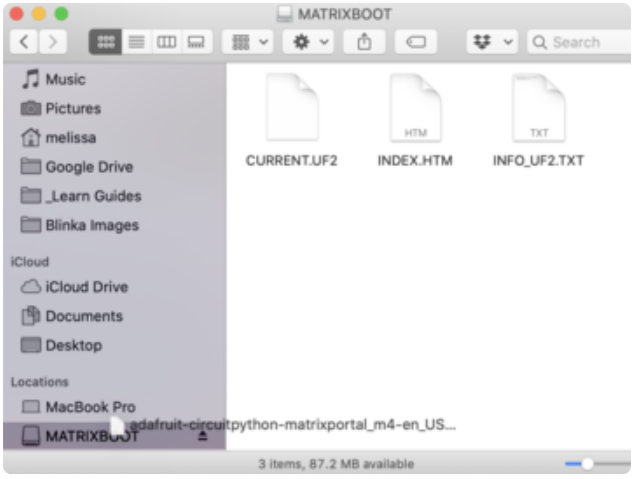
A lot of people end up using charge-only USB cables and it is very frustrating! So make sure you have a USB cable you know is good for data sync.

Double-click the **Reset** button (indicated by the green arrow) on your board, and you will see the NeoPixel RGB LED (indicated by the magenta arrow) turn green. If it turns red, check the USB cable, try another USB port, etc.

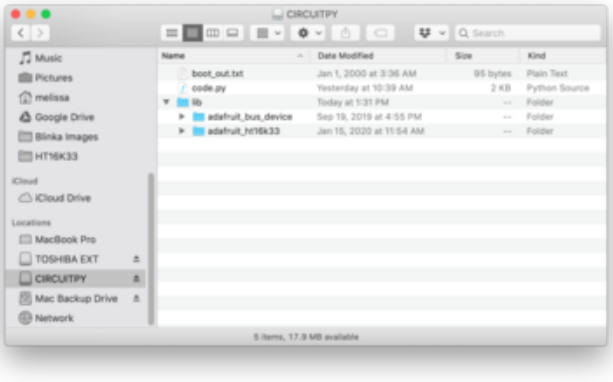
If double-clicking doesn't work the first time, try again. Sometimes it can take a few tries to get the rhythm right!



You will see a new disk drive appear called **MATRIXBOOT**.



Drag the **adafruit_circuitpython_etc.uf2** file to **MATRIXBOOT.**



The LED will flash. Then, the **MATRIXBOOT** drive will disappear and a new disk drive called **CIRCUITPY** will appear.

That's it, you're done! :)

CircuitPython Setup

To use all the amazing features of your MatrixPortal M4 with CircuitPython, you must first install a number of libraries. This page covers that process.

Adafruit CircuitPython Bundle

Download the Adafruit CircuitPython Library Bundle. You can find the latest release here:

Download latest Library Bundle

https://adafru.it/ENC

Download the **adafruit-circuitpython-bundle-version-mpy-*.zip** bundle zip file, and unzip a folder of the same name. Inside you'll find a **lib** folder. The entire collection of libraries is too large to fit on the **CIRCUITPY** drive. Instead, add each library as you need it, this will reduce the space usage but you'll need to put in a little more effort.

At a minimum we recommend the following libraries, in fact we more than recommend. They're basically required. So grab them and install them into **CIRCUITPY/lib** now!

- **adafruit_matrixportal** - this library is the main library used with the MatrixPortal.
- **adafruit_debouncer.mpy** - this library is used for debouncing a digital input pin
- **adafruit_portalbase** - This is the base library that adafruit_matrixportal is built on top of.
- **adafruit_esp32spi** - this is the library that gives you internet access via the ESP32 using (you guessed it!) SPI transport. You need this for anything Internet
- **neopixel.mpy** - for controlling the onboard neopixel
- **adafruit_bus_device** - low level support for I2C/SPI
- **adafruit_requests.mpy** - this library allows us to perform HTTP requests and get responses back from servers. GET/POST/PUT/PATCH - they're all in here!
- **adafruit_fakerequests.mpy** - This library allows you to create fake HTTP requests by using local files.
- **adafruit_io** - this library helps connect the PyPortal to our free data logging and viewing service
- **adafruit_bitmap_font** - we have fancy font support, and it's easy to make new fonts. This library reads and parses font files.
- **adafruit_display_text** - not surprisingly, it displays text on the screen
- **adafruit_lis3dh.mpy** - this library is used for the onboard accelerometer to detect the orientation of the MatrixPortal
- **adafruit_minimqtt** - this is used for communicating with MQTT servers.

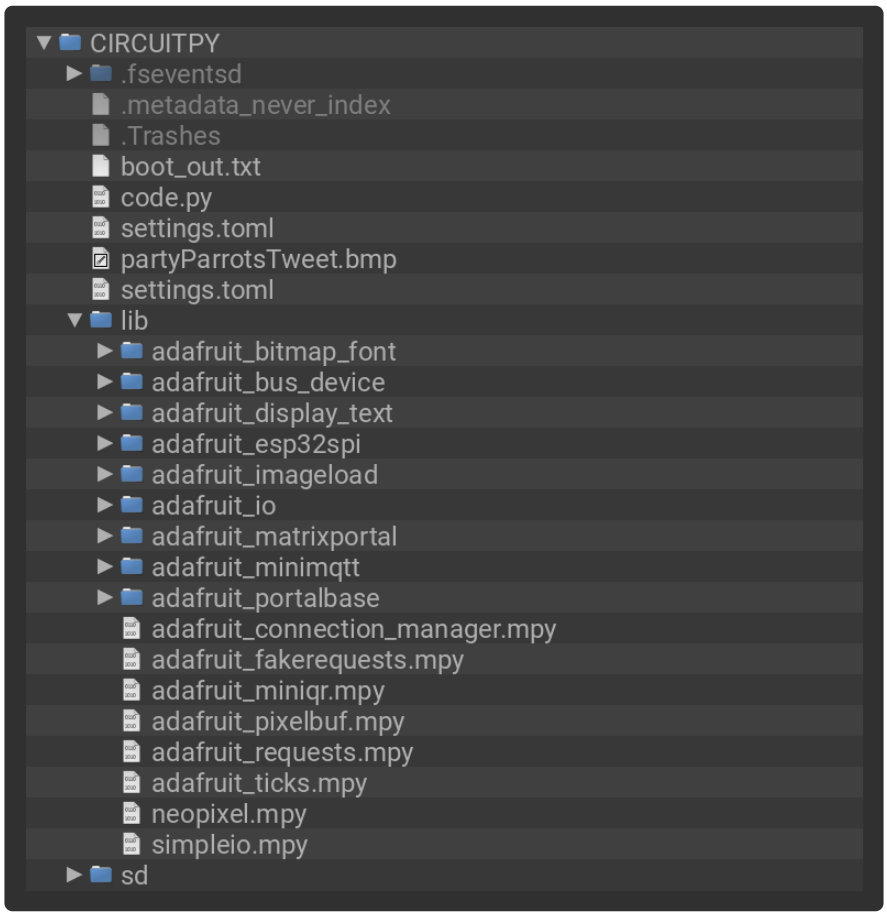
Coding the IoT Party Parrot

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 **IoT_Party_Parrot/**

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:




```
# SPDX-FileCopyrightText: 2020 Liz Clark for Adafruit Industries
#
# SPDX-License-Identifier: MIT

from os import getenv
import time
import board
import displayio
from adafruit_matrixportal.matrixportal import MatrixPortal
from adafruit_matrixportal.matrix import Matrix
import adafruit_imageload

print("Party Parrot Twitter Matrix")

# Get WiFi details, ensure these are setup in settings.toml
ssid = getenv("CIRCUITPY_WIFI_SSID")
password = getenv("CIRCUITPY_WIFI_PASSWORD")

if None in [ssid, password]:
    raise RuntimeError(
        "WiFi settings are kept in settings.toml, "
        "please add them there. The settings file must contain "
        "'CIRCUITPY_WIFI_SSID', 'CIRCUITPY_WIFI_PASSWORD', "
        "at a minimum."
    )

#  import your bearer token
```

```
bear = getenv('bearer_token')

#  query URL for tweets. looking for hashtag partyparrot sent to a specific username
DATA_SOURCE = 'https://api.twitter.com/2/tweets/search/recent?query=#partyparrot
to:blitzcitydiy'
#  json data path to get most recent tweet's ID number
DATA_LOCATION = ["meta", "newest_id"]

#  create MatrixPortal object to grab data/connect to internet
matrixportal = MatrixPortal(
    url=DATA_SOURCE,
    json_path=DATA_LOCATION,
    status_neopixel=board.NEOPIXEL
)

#  create matrix display
matrix = Matrix(width=32, height=32)
display = matrix.display

group = displayio.Group()

#  load in party parrot bitmap
parrot_bit, parrot_pal = adafruit_imageload.load("/partyParrotsTweet.bmp",
                                                 bitmap=displayio.Bitmap,
                                                 palette=displayio.Palette)

parrot_grid = displayio.TileGrid(parrot_bit, pixel_shader=parrot_pal,
                                 width=1, height=1,
                                 tile_height=32, tile_width=32,
                                 default_tile=10,
                                 x=0, y=0)

group.append(parrot_grid)

display.root_group = group

#  add bearer token as a header to request
matrixportal.set_headers({'Authorization': 'Bearer ' + bear})

last_value = 0 #  checks last tweet's ID
check = 0 #  time.monotonic() holder
parrot = False #  state to track if an animation is currently running
party = 0 #  time.monotonic() holder
p = 0 #  index for tilegrid
party_count = 0 #  count for animation cycles

while True:
    #  every 30 seconds...
    if (check + 30) < time.monotonic():
        #  store most recent tweet's ID number in value
        value = matrixportal.fetch()
        print("Response is", value)
        #  reset time count
        check = time.monotonic()
        #  compare last tweet ID and current tweet ID
        if last_value != value:
            print("new party!")
            #  if it's new, then it's a party!
            last_value = value
            parrot = True
        else:
            #  if it's not new, then the wait continues
            print("no new party... :(")
    #  when a new tweet comes in...
    if parrot:
        #  every 0.1 seconds...
        if (party + 0.1) < time.monotonic():
            #  the party parrot animation cycles
            parrot_grid[0] = p
```

```
#  p is the tilegrid index location
p += 1
party = time.monotonic()
#  if an animation cycle ends
if p > 9:
    #  index is reset
    p = 0
    #  animation cycle count is updated
    party_count += 1
    print("party parrot", party_count)
#  after 16 animations cycles...
if party_count > 15:
    #  reset states
    parrot = False
    party_count = 0
    p = 0
    #  clear the matrix so that it's blank
    parrot_grid[0] = 10
    print("the party is over")
```

Animating Sprites with CircuitPython

You can create animations with the **displayio** library using sprite sheets. Sprite sheets can be thought of as an array of images. They can be iterated through just like an array of variables, strings or integers in code.

If the sprite sheet is set up with images that are sequential, then it will appear as an animation when they're played. You can do this by splitting out of the layers of a .GIF or creating your own artwork.

To begin animating your sprites, you'll setup your matrix and display objects as usual with the **displayio** and **adafruit_matrixportal** libraries.

The bitmap sprite sheet is brought into the code with the **adafruit_imageload** library. This bitmap is sliced into a `TileGrid`, which will be how the images will be iterated through to create an animation.

```
matrix = Matrix(width=32, height=32)
display = matrix.display

group = displayio.Group()

parrot_bit, parrot_pal = adafruit_imageload.load("/partyParrotsTweet.bmp",
                                                 bitmap=displayio.Bitmap,
                                                 palette=displayio.Palette)

parrot_grid = displayio.TileGrid(parrot_bit, pixel_shader=parrot_pal,
                                 width=1, height=1,
```

```
                                    tile_height=32, tile_width=32,
                                    default_tile=10,
                                    x=0, y=0)

group.append(parrot_grid)

display.root_group = group
```

Two variables are needed to animate a sprite: one to track the index of the `TileGrid` and one to track the time. `p` will be used as the `TileGrid` index and `party` will be used to track time with `time.monotonic()`.

In the loop, every `0.1` seconds, the `TileGrid`'s index advances by `1`. You can increase or decrease the timing to adjust the speed for your animation.

When `p` reaches the end of the `TileGrid` array, it is reset to `0` so that the animation can loop.

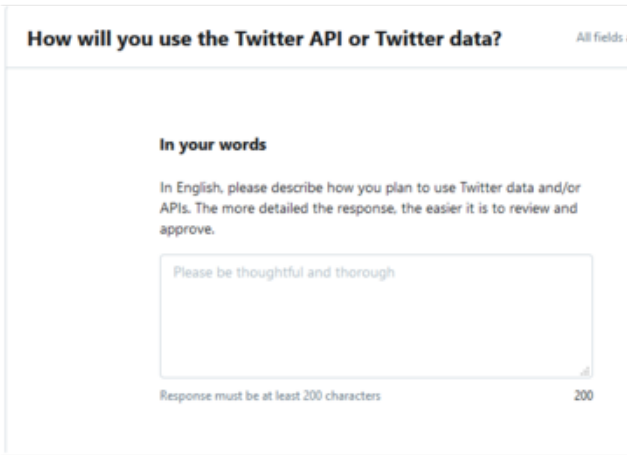
```
party = 0
p = 0

while True:
  if (party + 0.1) &lt; time.monotonic():
    parrot_grid[0] = p
    p += 1
    party = time.monotonic()
    if p &gt; 9:
      p = 0
```

And that's it! With just a few lines of code, you can get a looping animation running (or partying) on your matrix display.

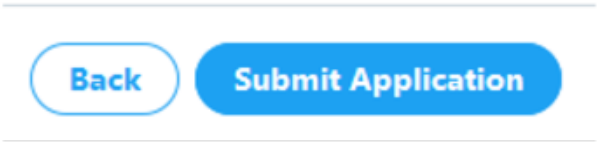
Setting up the Twitter API and settings.toml

You can access Twitter data for your projects using the Twitter API. To begin, you'll need to apply for a developer account with your Twitter account at [https://developer.twitter.com/en/apply-for-access](https://adafru.it/NFH) (https://adafru.it/NFH)

Once you click apply, you'll be prompted to select your use case. Under Hobbyist, select Exploring the API.



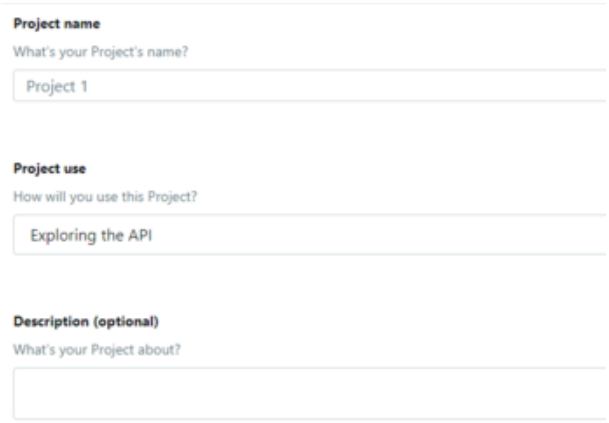
After confirming your contact details, you'll be asked to describe how you'll be using the Twitter API. Enter in information as applicable. If you're planning to use the API for other projects beyond this one, you may want to include that information here for the future.



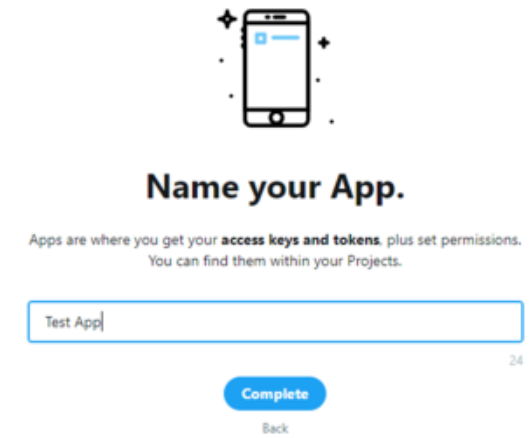
After confirming your information and agreeing to terms of service, you'll click Submit Application. Applications are usually approved fairly quickly, usually within a few minutes.

Create a Project

When you log into the developer portal, you'll see your developer dashboard. Here you can access all of your account information and documentation.

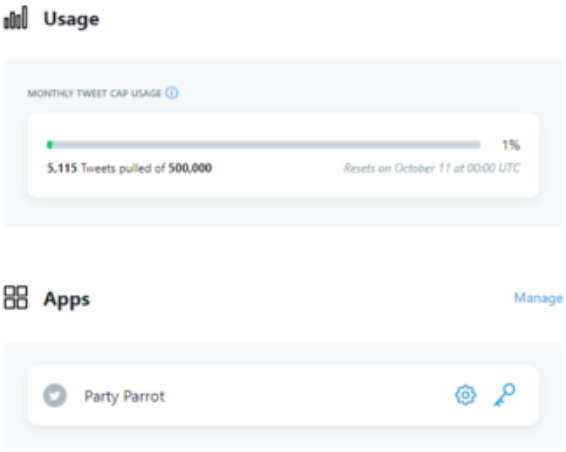


In order to pull from the Twitter API, you need to create a Project. At the bottom of the Dashboard page, click on New Project and then complete the information about your Project.



Each Project has an App, which stores your API keys and access points.

Add an App to the Project your just created. Enter your desired name for your App and then click Complete.



After you've created the Project and App, you can access the Project page which supplies Usage stats and allows access to your App.

If you click on the App Settings button (the cog wheel), you can adjust access settings for the App and access your Authentication tokens.

Do not share your Twitter API Authentication tokens with anyone!

Keep Your Secrets Secret

The project's code needs access to your Twitter API Authorization tokens. To keep this information secret, but still accessible, you can enter this information into a file called **settings.toml**.

```
CIRCUITPY_WIFI_SSID="your-wifi-ssid"
CIRCUITPY_WIFI_PASSWORD="your-wifi-password"
timezone="America/New_York"  # http://worldtimeapi.org/timezones
twitter_api_key="insert your twitter api key here"
twitter_secret_key="insert your twitter secret key here"
bearer_token="insert your bearer token here"
```

Getting and Parsing Tweets

You can access tweets with the Twitter API using a few different methods. For this project, the recent search method is used. This involves creating a hyperlink that contains the search parameters to filter the tweets that you're getting.

Twitter has documentation on the available parameters that you can use with v2 of the API: [https://developer.twitter.com/en/docs/twitter-api/tweets/search/quick-start](https://adafru.it/NFI) (https://adafru.it/NFI)

The goal for this project is to see the most recent tweets that contain a hashtag (#partyparrot) that are sent to a specific user. This means that your hyperlink query will look like this:

- `'https://api.twitter.com/2/tweets/search/recent?query=#partyparrot to:blitzcitydiy'`

Notice that there is a space between the hashtag and `to:` queries. If you want to add more options to your query, they just need to be separated by a space.

This URL is going to be used in the CircuitPython code as the data source for the JSON feed. You'll access that feed in order to pull data to trigger your sprite animation.

Introducing the Twitter API To Your CircuitPython Code

After all of the setup comes the fun part: bringing the Twitter API information into the CircuitPython code.

First, `bear` is created as a variable to hold your Bearer token. This is needed to access the API. It's imported from your **secrets.py** file.

Then `DATA_SOURCE` is setup to hold your query URL and `DATA_LOCATION` is setup to list your JSON path information, which will be important in the main loop.

A `MatrixPortal` object is created using the **adafruit_matrixportal** library. This allows your MatrixPortal M4 to connect to the internet and then access your query URL to pull down your desired data via the filtered JSON path.

```
bear = secrets['bearer_token']

DATA_SOURCE = ('https://api.twitter.com/2/tweets/search/recent?query=#partyparrot
to:blitzcitydiy')
DATA_LOCATION = ["meta", "newest_id"]

matrixportal = MatrixPortal(
    url=DATA_SOURCE,
```

```
    json_path=DATA_LOCATION,
    status_neopixel=board.NEOPIXEL
)
```

The Twitter API requires a Bearer token to authorize access to the query URL. This is sent as a header along with the request made with the **adafruit_matrixportal** library. You can setup headers using `matrixportal.set_headers()`.

After the header is created, a few states are setup for use in the loop. Their functions are commented below.

```
matrixportal.set_headers({'Authorization': 'Bearer ' + bear})

last_value = 0 #  checks last tweet's ID
check = 0 #  time.monotonic() holder
parrot = False #  state to track if an animation is currently running
party = 0 #  time.monotonic() holder
p = 0 #  index for tilegrid
party_count = 0 #  count for animation cycles
```

In the loop, the code is running `matrixportal.fetch()` to check and see if a new tweet has been sent that matches the query parameters.

This data stream is stored as `value` and holds the JSON path information. In this case, it's the most recent tweet's ID number.

This ID number is compared to the previous tweet's ID number, which is stored in `last_value`. If they are not a match, then this means that a new tweet with matching parameters has been received and the `parrot` state is updated to `True`.

```
while True:
    if (check + 30) &lt; time.monotonic():
        value = matrixportal.fetch()
        print("Response is", value)
        check = time.monotonic()
        if last_value != value:
            print("new party!")
            last_value = value
            parrot = True
        else:
            print("no new party... :(")
```

When `party` is `True`, the party parrot begins animating on the LED matrix. The entire animation cycle is played 16 times and is tracked with `party_count`. After 16 cycles, the animation stops and all of the states are reset to await the next tweet.

```
if parrot:
        if (party + 0.1) &lt; time.monotonic():
            parrot_grid[0] = p
            p += 1
            party = time.monotonic()
            if p &gt; 9:
                p = 0
                party_count += 1
```

```
            print("party parrot", party_count)
        if party_count &gt; 15:
            parrot = False
            party_count = 0
            p = 0
            parrot_grid[0] = 10
            print("the party is over")
```

Customization

You shouldn't feel limited by the parameters of this party project though. You can setup custom queries looking at a wide variety of filters for Twitter, whether it be hashtags, emoji usage, retweets, etc.

You can also include any sprite animation that you want. Perhaps you want a different animation for different Twitter notifications. There are definitely many ways that you could customize this project for your own needs.