MagTag Weekly Showtimes Event Notifier

Created by John Park

https://learn.adafruit.com/magtag-weekly-event-showtimes-display

Last updated on 2024-11-29 10:05:25 AM EST

Table of Contents

Overview 3

- Parts

Install CircuitPython 4

- Set Up CircuitPython
- Option 1 - Load with UF2 Bootloader
- Try Launching UF2 Bootloader
- Option 2 - Use esptool to load BIN file
- Option 3 - Use Chrome Browser To Upload BIN file

CircuitPython Internet Test 8

- The settings.toml File
- IPv6 Networking

Getting The Date & Time 15

- Step 1) Make an Adafruit account
- Step 2) Sign into Adafruit IO
- Step 3) Get your Adafruit IO Key
- Step 4) Upload Test Python Code

MagTag-Specific CircuitPython Libraries 18

- Get Latest Adafruit CircuitPython Bundle
- Secrets

Code the MagTag Showtimes Display 19

- Text Editor
- Code
- Events File
- How It Works
- Setup
- Event File
- Time Check
- Find Next Event
- Background Images and Text
- Sleep

Overview

Never miss an Adafruit Livestream with this Showtimes display to remind you what's coming up next/playing now!

This MagTag display project informs you of the currently playing Adafruit Livestream show, or lets you know when the next show starts! You can modify the project for your own recurring calendar events, too!

This project is a non-sync'd event reminder. Instead of connecting to a calendar system, a JSON file with event names, times and images lives on the MagTag. This is great for repeating events that don't change week-by-week. The MagTag stays in deep-sleep mode between event updates so it runs for many weeks on a charge. You can have it beep the onboard speaker to remind you an event is starting, too!

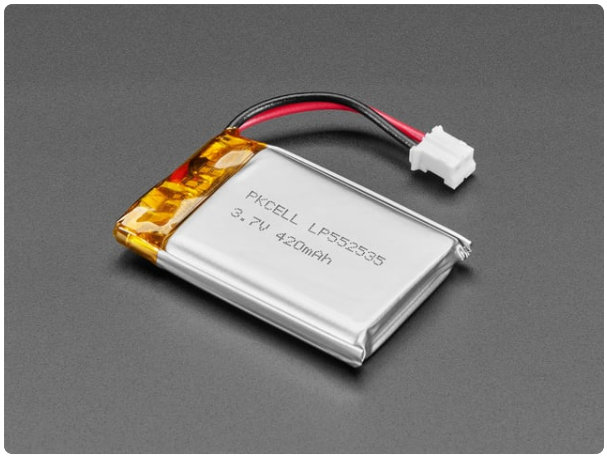
Parts

Adafruit MagTag - 2.9" Grayscale E-Ink WiFi Display

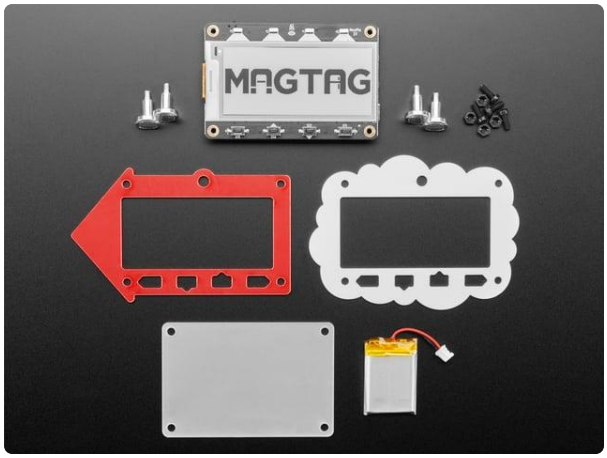
The Adafruit MagTag combines the new ESP32-S2 wireless module and a 2.9" grayscale E-Ink display to make a low-power IoT display that can show data on its screen even when power...

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



[Lithium Ion Polymer Battery with Short Cable - 3.7V 420mAh]
Lithium-ion polymer (also known as 'lipo' or 'lipoly') batteries are thin, light, and powerful. The output ranges from 4.2V when completely charged to 3.7V. This...
https://www.adafruit.com/product/4236



[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

Install CircuitPython

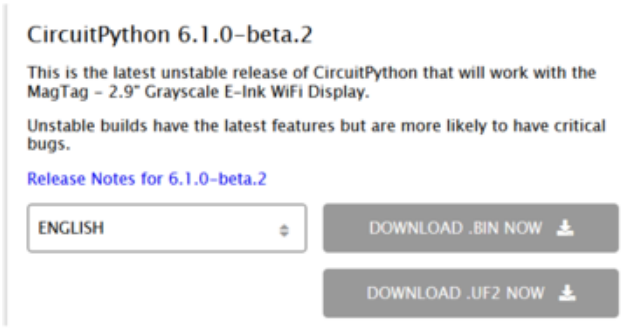
CircuitPython (https://adafru.it/tB7) is a derivative of MicroPython (https://adafru.it/BeZ) designed to simplify experimentation and education on low-cost microcontrollers. It makes it easier than ever to get prototyping by requiring no upfront desktop software downloads. Simply copy and edit files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

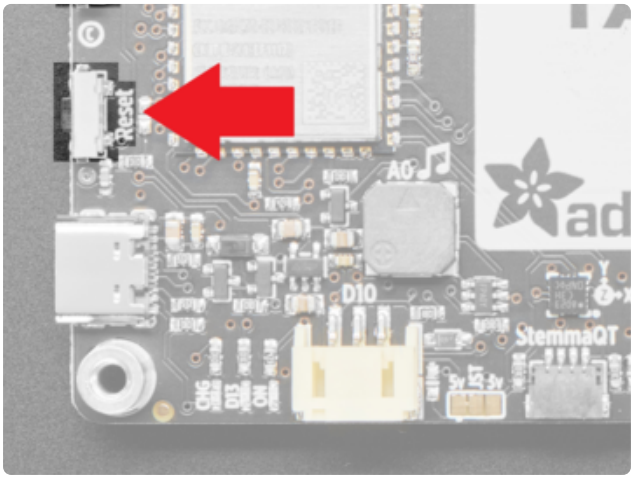
Option 1 - Load with UF2 Bootloader

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.**

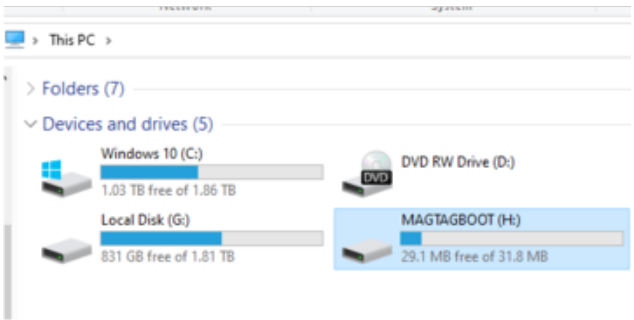
Still, try this first!

Try Launching UF2 Bootloader

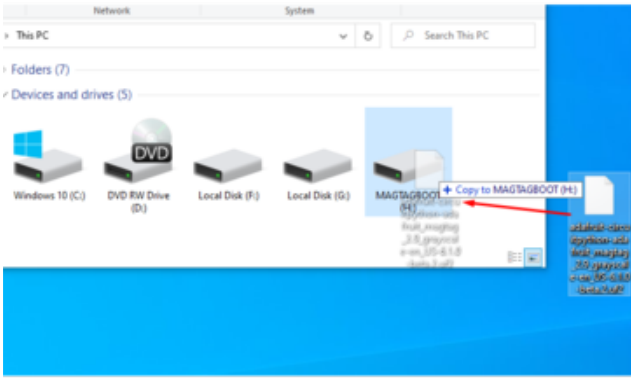
Loading CircuitPython by drag-n-drop UF2 bootloader is the easier way and we recommend it. If you have a MagTag where the front of the board is black, your MagTag came with UF2 already on it.



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.

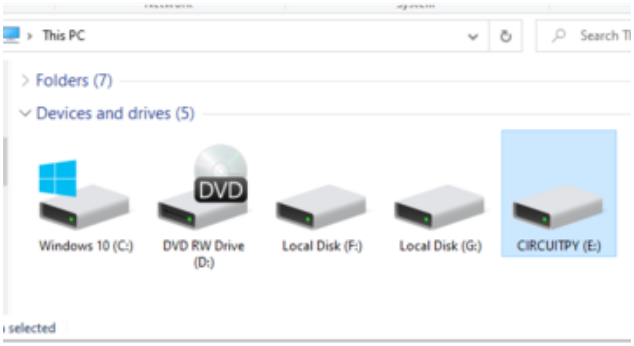


If the UF2 bootloader is installed, you will see a new disk drive appear called **MAGTAGBOOT**



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)

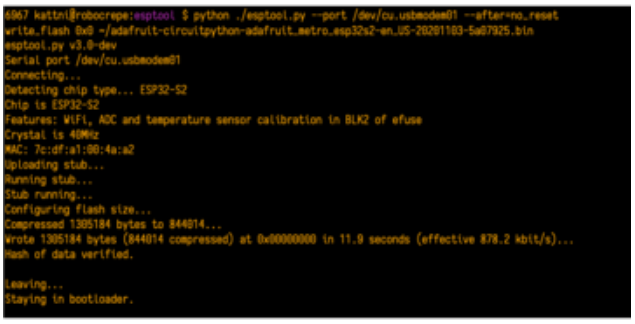


Your board should auto-reset into CircuitPython, or you may need to press reset. A **CIRCUITPY** drive will appear. You're done! Go to the next pages.

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.

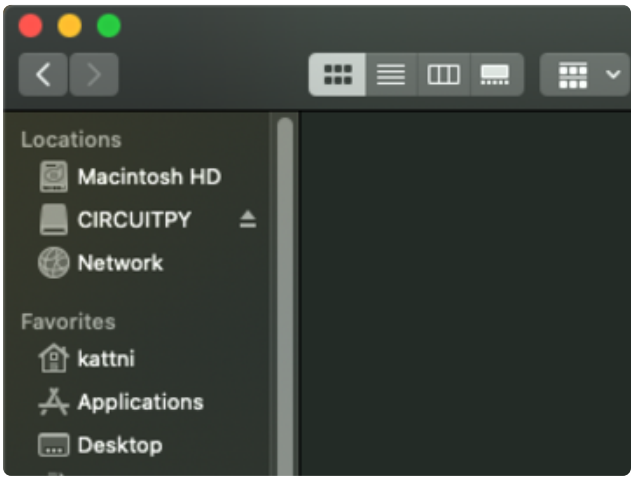
You can upload with **esptool** to the ROM (hardware) bootloader instead!



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!

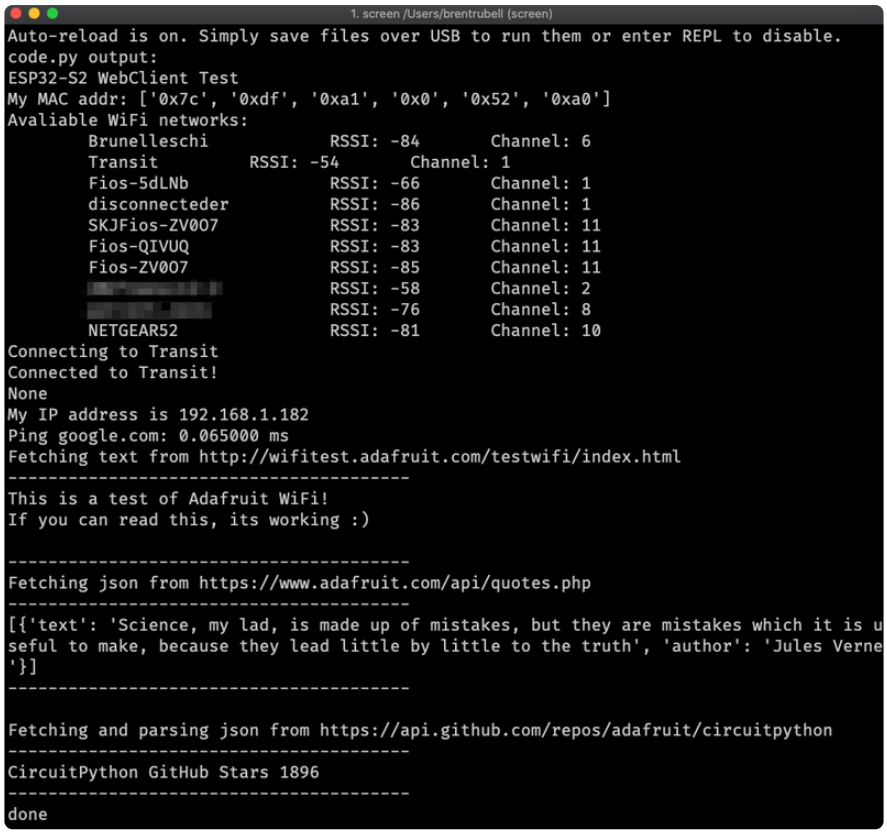
As you make projects you may need more tokens and keys, just add them one line at a time. See for example other tokens such as one for accessing GitHub or the Hackaday API. Other non-secret data like your timezone can also go here.

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')
```

Getting The Date & Time

A very common need for projects is to know the current date and time. Especially when you want to deep sleep until an event, or you want to change your display based on what day, time, date, etc. it is

Determining the correct local time is really really hard. There are various time zones, Daylight Savings dates, leap seconds, etc. Trying to get NTP time and then back-calculating what the local time is, is extraordinarily hard on a microcontroller just isn't worth the effort and it will get out of sync as laws change anyways.

For that reason, we have the free adafruit.io time service. **Free for anyone with a free adafruit.io account.** You do need an account because we have to keep accidentally mis-programmed-board from overwhelming adafruit.io and lock them out temporarily. Again, it's free!

There are other services like WorldTimeAPI, but we don't use those for our guides because they are nice people and we don't want to accidentally overload their site. Also, there's a chance it may eventually go down or also require an account.

Step 1) Make an Adafruit account

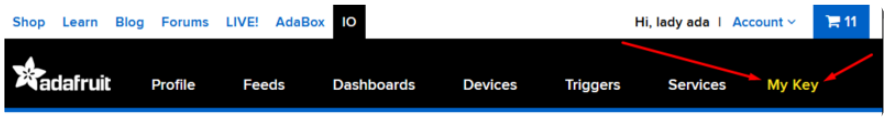
It's free! Visit https://accounts.adafruit.com/ (https://adafru.it/dyy) to register and make an account if you do not already have one

Step 2) Sign into Adafruit IO

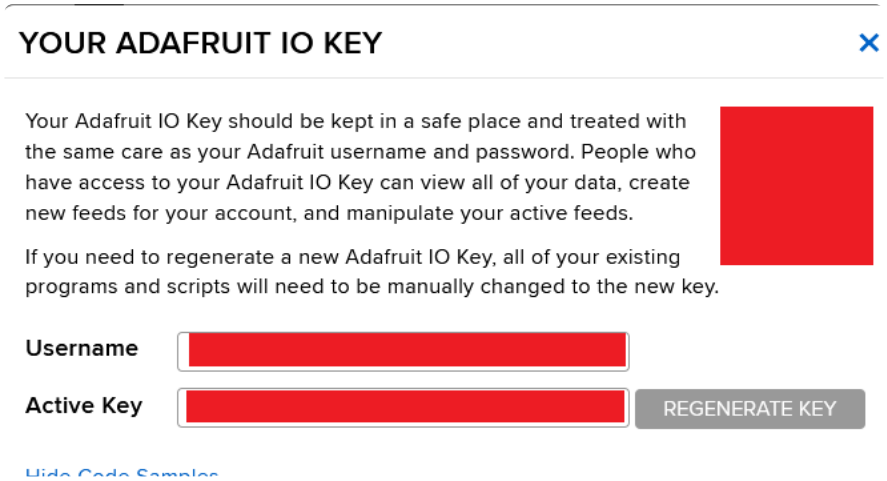
Head over to io.adafruit.com (https://adafru.it/fsU) and click **Sign In** to log into IO using your Adafruit account. It's free and fast to join.

Step 3) Get your Adafruit IO Key

Click on **My Key** in the top bar



You will get a popup with your **Username** and **Key** (In this screenshot, we've covered it with red blocks)



Go to the **settings.toml** file on your CIRCUITPY drive and add three lines for `AIO_USERNAME`, `ADAFRUIT_AIO_KEY` and `TIMEZONE` so you get something like the following:

```
# This file is where you keep secret settings, passwords, and tokens!
# If you put them in the code you risk committing that info or sharing it

CIRCUITPY_WIFI_SSID = "your-wifi-ssid"
```

```
CIRCUITPY_WIFI_PASSWORD = "your-wifi-password"
ADAFRUIT_AIO_USERNAME = "your-adafruit-io-username"
ADAFRUIT_AIO_KEY = "your-adafruit-io-key"
# Timezone names from http://worldtimeapi.org/timezones
TIMEZONE="America/New_York"
```

The timezone is optional, if you don't have that entry, adafruit.io will guess your timezone based on geographic IP address lookup. You can visit http://worldtimeapi.org/timezones (https://adafru.it/EcP) to see all the time zones available (even though we do not use Worldtime for time-keeping, we do use the same time zone table).

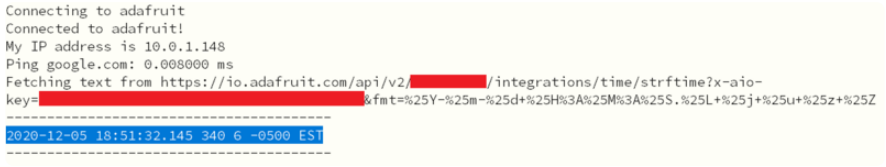
Step 4) Upload Test Python Code

This code is like the Internet Test code from before, but this time it will connect to adafruit.io and get the local time

```
import ipaddress
import os
import ssl
import wifi
import socketpool
import adafruit_requests
import secrets


# Get our username, key and desired timezone
ssid = os.getenv("CIRCUITPY_WIFI_SSID")
password = os.getenv("CIRCUITPY_WIFI_PASSWORD")
aio_username = os.getenv("ADAFRUIT_AIO_USERNAME")
aio_key = os.getenv("ADAFRUIT_AIO_KEY")
timezone = os.getenv("TIMEZONE")
TIME_URL = f"https://io.adafruit.com/api/v2/{aio_username}/integrations/time/
strftime?x-aio-key={aio_key}&amp;tz={timezone}"
TIME_URL += "&amp;fmt=%25Y-%25m-%25d+%25H%3A%25M%3A%25S.%25L+%25j+%25u+%25z+%25Z"

print("ESP32-S2 Adafruit IO Time test")

print("My MAC addr:", [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("Connecting to", ssid)
wifi.radio.connect(ssid, password)
print(f"Connected to {ssid}!")
print("My IP address is", wifi.radio.ipv4_address)

ipv4 = ipaddress.ip_address("8.8.4.4")
print("Ping google.com:", wifi.radio.ping(ipv4), "ms")

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

print("Fetching text from", TIME_URL)
response = requests.get(TIME_URL)
print("-" * 40)
```

```
print(response.text)
print("-" * 40)
```

After running this, you will see something like the below text. We have blocked out the part with the secret username and key data!



Note at the end you will get the date, time, and your timezone! If so, you have correctly configured your **settings.toml** and can continue to the next steps!

MagTag-Specific CircuitPython Libraries

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

Get Latest Adafruit CircuitPython Bundle

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

Download the latest Library Bundle from circuitpython.org

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. Therefore, you'll need to copy the necessary libraries to your board individually.

At a minimum, the following libraries are required. Copy the following folders or .mpy files to the **lib** folder on your **CIRCUITPY** drive. **If the library is a folder, copy the entire folder** to the **lib** folder on your board.

Library folders (copy the whole folder over to lib):

- **adafruit_magtag** - This is a helper library designed for using all of the features of the MagTag, including networking, buttons, NeoPixels, etc.
- **adafruit_portalbase** - This library is the base library that adafruit_magtag is built on top of.

- **adafruit_bitmap_font** - There is fancy font support, and it's easy to make new fonts. This library reads and parses font files.
- **adafruit_display_text** - This library displays text on the screen.
- **adafruit_io** - This library helps connect the MagTag to our free data logging and viewing service
- **adafruit_minimqtt** - This library provides MQTT service for Adafruit IO.

Library files:

- **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_miniqr.mpy** - QR creation library lets us add easy-to-scan 2D barcodes to the E-Ink display
- **neopixel.mpy** - This library is used to control the onboard NeoPixels.
- **simpleio.mpy** - This library is used for tone generation.

Secrets

Even if you aren't planning to go online with your MagTag, you'll need to have a **secrets.py** file in the root directory (top level) of your **CIRCUITPY** drive. If you do not intend to connect to wireless, it does not need to have valid data in it. Here's more info on the secrets.py file (https://adafru.it/P3b).

Code the MagTag Showtimes Display

Text Editor

Adafruit recommends using the Mu editor for editing your CircuitPython code. You can get more info in this guide (https://adafru.it/ANO).

Alternatively, you can use any text editor that saves simple text files.

Code

Click the Download: Project Zip File link below in the code window to get a zip file with all the files needed for the project. Copy **code.py** from the zip file and place on the **CIRCUITPY** drive.

You'll also need to place the **events.json** file at the root level of your **CIRCUITPY** drive so it can be accessed by the code.

Also copy the whole **/bmps** directory from the zip file and place and its contents it on the **CIRCUITPY** drive. These are some sample images you can start with.

Copy the **/fonts** directory from the zip file and place and its contents it on the **CIRCUITPY** drive.

Once all the files are on the MagTag **CIRCUITPY** drive, the directory structure should be the same as the listing below. If not, ensure you've got all the files noted in prior steps.


```
# SPDX-FileCopyrightText: 2020 John Park for Adafruit Industries
#
# SPDX-License-Identifier: MIT

# MagTag Showtimes Event Viewer
# Uses the events.json file to display next or current event
# Be sure to put WiFi access point info in secrets.py file to connect

import time
import json
import re
from adafruit_magtag.magtag import MagTag

# You can test by setting a time.struct here, to pretend its a different day
# (tm_year, tm_mon, tm_mday, tm_hour, tm_min, tm_sec, tm_wday, tm_yday, tm_isdst)
FAKETIME = False  # time.struct_time(2020, 12, 11,     15, 01, 00,    4, 346, -1)

BEEP_ON_EVENTSTART = True   # beep when the event begins?
EVENT_FILE = "events.json"  # file containing events
USE_24HR_TIME = False   # True for 24-hr time on display, false for 12 hour (am/pm)
time

magtag = MagTag()
magtag.add_text(
    text_font="/fonts/Arial-Bold-12.pcf",
    text_color=0xFFFFFF,
    text_position=(2, 112),
    text_anchor_point=(0, 0),
)

# According to Python, monday is index 0...this array will help us track it
```

```
day_names = ("Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday",
"Sunday")
events = None
with open(EVENT_FILE, 'r') as evfile:
    events = json.load(evfile)

# validate data
for i, event in enumerate(events):
    if not event.get('name'):
        raise RuntimeError("No name in event %d" % i)
    if not event.get('day_of_week') or event['day_of_week'] not in day_names:
        raise RuntimeError("Invalid day of week for event '%s'" % event['name'])
    r = re.compile('[0-2]?[0-9]:[0-5][0-9]')
    if not event.get('start_time') or not r.match(event['start_time']) :
        raise RuntimeError("Invalid start time for event '%s'" % event['name'])
    if not event.get('end_time') or not r.match(event['end_time']) :
        raise RuntimeError("Invalid end time for event '%s'" % event['name'])

print(events)

now = None
if not FAKETIME:
    magtag.network.connect()
    magtag.get_local_time()
    now = time.localtime()
else:
    now = FAKETIME

print("Now: ", now)

# Helper to convert times into am/pm times
def time_format(timestr):
    if USE_24HR_TIME:
        return timestr
    hr, mn = [int(x) for x in timestr.split(":")]
    if hr > 12:
        return "%d:%02d PM" % (hr-12, mn)
    elif hr > 0:
        return "%d:%02d AM" % (hr, mn)
    else:
        return "12:%02d AM" % (mn)

# find next event!
remaining_starttimes = []
remaining_endtimes = []
current_event = None
for event in events:
    days_till_event = (day_names.index(event["day_of_week"]) - now[6] + 7) % 7

    # now figure out minutes until event
    eventstart_hr, eventstart_min = event["start_time"].split(":")
    eventstart_time_in_minutes = int(eventstart_hr) * 60 + int(eventstart_min)
    # we'll also track when it ends
    eventend_hr, eventend_min = event["end_time"].split(":")
    eventend_time_in_minutes = int(eventend_hr) * 60 + int(eventend_min)

    current_time_in_minutes = now[3] * 60 + now[4]
    print(
        "\tEvent start is at",
        eventstart_time_in_minutes,
        "now is",
        current_time_in_minutes,
    )
    minutes_till_eventstart = eventstart_time_in_minutes - current_time_in_minutes
    minutes_till_eventend = eventend_time_in_minutes - current_time_in_minutes

    # add the number of days to minutes:
    minutes_till_eventstart += days_till_event * 24 * 60
    minutes_till_eventend += days_till_event * 24 * 60
```

```
    # if time is negative, that means it already happened today, so skip ahead
    if minutes_till_eventstart < 0:
        minutes_till_eventstart += 7 * 24 * 60
    if minutes_till_eventend < 0:
        minutes_till_eventend += 7 * 24 * 60

    print(
        "\t%d minutes till start, %d minutes till end"
        % (minutes_till_eventstart, minutes_till_eventend)
    )
    if (minutes_till_eventstart == 0) or (
            minutes_till_eventend < minutes_till_eventstart
    ):
        current_event = event

    # now we can back-calculate when the event is for our debugging
    days = minutes_till_eventstart // (24 * 60)
    hrs = (minutes_till_eventstart - days * (24 * 60)) // 60
    mins = minutes_till_eventstart % 60
    print(event["name"], "starts in", days, "days", hrs, "hours and", mins,
"minutes\n")

    remaining_starttimes.append(minutes_till_eventstart)
    remaining_endtimes.append(minutes_till_eventend)

mins_till_next_eventstart = min(remaining_starttimes)
mins_till_next_eventend = min(remaining_endtimes)
next_up = events[remaining_starttimes.index(mins_till_next_eventstart)]

# OK find the one with the smallest minutes remaining
sleep_time = None
if current_event:
    print("Currently: ", current_event)
    magtag.set_background("/bmps/"+current_event["graphic"])
    magtag.set_text("Currently streaming until " +
time_format(current_event["end_time"]))
    remaining_starttimes.index(mins_till_next_eventstart)
    if BEEP_ON_EVENTSTART:
        for _ in range(3):
            magtag.peripherals.play_tone(1760, 0.1)
            time.sleep(0.2)
    sleep_time = mins_till_next_eventend + 1
else:
    print("Next up! ", next_up)
    magtag.set_background("/bmps/"+next_up["graphic"])

    string = (
        "Coming up on "
        + next_up["day_of_week"]
        + " at "
        + time_format(next_up["start_time"])
    )
    magtag.set_text(string)
    sleep_time = mins_till_next_eventstart

print("Sleeping for %d minutes" % sleep_time)
time.sleep(2)
magtag.exit_and_deep_sleep(sleep_time * 60)
```

Events File

In order to make it easier to edit your events, we put all of the event data into a separate .json file that the main code will access.

Here's what the event file looks like:

```
[
  {
    "name": "JP's Product Pick of the Week",
    "day_of_week": "Tuesday",
    "graphic": "jpp.bmp",
    "start_time": "13:00",
    "end_time": "13:30"
  },
  {
    "name": "3D Hangouts",
    "day_of_week": "Wednesday",
    "graphic": "3dh.bmp",
    "start_time": "8:00",
    "end_time": "9:00"
  },
  {
    "name": "Show & Tell",
    "day_of_week": "Wednesday",
    "graphic": "snt.bmp",
    "start_time": "16:30",
    "end_time": "17:00"
  },
  {
    "name": "Ask An Engineer",
    "day_of_week": "Wednesday",
    "graphic": "aae.bmp",
    "start_time": "17:00",
    "end_time": "18:00"
  },
  {
    "name": "John Park's Workshop",
    "day_of_week": "Thursday",
    "graphic": "jpw.bmp",
    "start_time": "13:00",
    "end_time": "14:00"
  },
  {
    "name": "Scott's Deep Dive",
    "day_of_week": "Friday",
    "graphic": "dds.bmp",
    "start_time": "14:00",
    "end_time": "15:00"
  }
]
```

How It Works

Libraries

First we import some libraries to help out, including `time`, `json` (for parsing our event data from json file format), `re` (regular expression), and the `adafruit_magtag` library, which will make it simple to display graphics and text on the MagTag's e-Ink display.

```
import time
import json
import re
from adafruit_magtag.magtag import MagTag
```

Faketime

When you're testing code that is based on a date and time it can be a bit inconvenient to wait around for that exact moment when your event should begin! So, we can use a `FAKETIME` variable when necessary. In normal use, this will be set to `False`.

The comment shows how to use it when you need to fake things out.

```
# (tm_year, tm_mon, tm_mday, tm_hour, tm_min, tm_sec, tm_wday, tm_yday, tm_isdst)
FAKETIME = False  # time.struct_time(2020, 12, 11,     15, 01, 00,    4, 346, -1)
```

Variables

These other variables are set up for convenience -- you can turn the beep on and off, set the name of the **events.json** file, and decide if you're going to display 24 hour time or not here.

```
BEEP_ON_EVENTSTART = True   # beep when the event begins?
EVENT_FILE = "events.json"  # file containing events
USE_24HR_TIME = False   # True for 24-hr time on display, false for 12 hour (am/pm)
time
```

Setup

Next we'll take care of MagTag setup, and set a text object in place.

The `day_names` and `events` variables will help us print nice day names and keep track of event state.

```
# According to Python, monday is index 0...this array will help us track it
day_names = ("Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday",
"Sunday")
events = None
```

Event File

We'll open the **events.json** file and then validate the data, printing it to the serial output. Here you can see regular expressions being used to parse out some of the data. It's helpful to watch the serial output while this runs to see how the data is parsed.

```
with open(EVENT_FILE, 'r') as evfile:
    events = json.load(evfile)

# validate data
for i, event in enumerate(events):
    if not event.get('name'):
        raise RuntimeError("No name in event %d" % i)
    if not event.get('day_of_week') or event['day_of_week'] not in day_names:
        raise RuntimeError("Invalid day of week for event '%s'" % event['name'])
    r = re.compile('[0-2]?[0-9]:[0-5][0-9]')
    if not event.get('start_time') or not r.match(event['start_time']) :
```

```
        raise RuntimeError("Invalid start time for event '%s'" % event['name'])
    if not event.get('end_time') or not r.match(event['end_time']) :
        raise RuntimeError("Invalid end time for event '%s'" % event['name'])

print(events)
```

Time Check

Next we'll use the MagTag's network connection to check the local time (unless `FAKETIME` is being used). We also create a `time_format()` function to help with time formatting.

```
now = None
if not FAKETIME:
    magtag.network.connect()
    magtag.get_local_time()
    now = time.localtime()
else:
    now = FAKETIME

print("Now: ", now)
def time_format(timestr):
    if USE_24HR_TIME:
        return timestr
    hr, mn = [int(x) for x in timestr.split(":")]
    if hr &gt; 12:
        return "%d:%02d PM" % (hr-12, mn)
    elif hr &gt; 0:
        return "%d:%02d AM" % (hr, mn)
    else:
        return "12:%02d AM" % (mn)
```

Find Next Event

Now that we know when the events are and what time it is, this block of code determines which event is next.

```
remaining_starttimes = []
remaining_endtimes = []
current_event = None
for event in events:
    days_till_event = (day_names.index(event["day_of_week"]) - now[6] + 7) % 7

    # now figure out minutes until event
    eventstart_hr, eventstart_min = event["start_time"].split(":")
    eventstart_time_in_minutes = int(eventstart_hr) * 60 + int(eventstart_min)
    # we'll also track when it ends
    eventend_hr, eventend_min = event["end_time"].split(":")
    eventend_time_in_minutes = int(eventend_hr) * 60 + int(eventend_min)

    current_time_in_minutes = now[3] * 60 + now[4]
    print(
        "\tEvent start is at",
        eventstart_time_in_minutes,
        "now is",
        current_time_in_minutes,
    )
    minutes_till_eventstart = eventstart_time_in_minutes - current_time_in_minutes
    minutes_till_eventend = eventend_time_in_minutes - current_time_in_minutes

    # add the number of days to minutes:
```

```
    minutes_till_eventstart += days_till_event * 24 * 60
    minutes_till_eventend += days_till_event * 24 * 60

    # if time is negative, that means it already happened today, so skip ahead
    if minutes_till_eventstart &lt; 0:
        minutes_till_eventstart += 7 * 24 * 60
    if minutes_till_eventend &lt; 0:
        minutes_till_eventend += 7 * 24 * 60

    print(
        "\t%d minutes till start, %d minutes till end"
        % (minutes_till_eventstart, minutes_till_eventend)
    )
    if (minutes_till_eventstart == 0) or (
            minutes_till_eventend &lt; minutes_till_eventstart
    ):
        current_event = event

    # now we can back-calculate when the event is for our debugging
    days = minutes_till_eventstart // (24 * 60)
    hrs = (minutes_till_eventstart - days * (24 * 60)) // 60
    mins = minutes_till_eventstart % 60
    print(event["name"], "starts in", days, "days", hrs, "hours and", mins,
"minutes\n")

    remaining_starttimes.append(minutes_till_eventstart)
    remaining_endtimes.append(minutes_till_eventend)

mins_till_next_eventstart = min(remaining_starttimes)
mins_till_next_eventend = min(remaining_endtimes)
next_up = events[remaining_starttimes.index(mins_till_next_eventstart)]
```

Background Images and Text

Whichever event has is coming up the soonest will get its background graphic displayed, as well as the "Next up!" text. If the event is in progress, the text will display that it is "Currently streaming until..." the end time.

```
sleep_time = None
if current_event:
    print("Currently: ", current_event)
    magtag.set_background("bmps/"+current_event["graphic"])
    magtag.set_text("Currently streaming until " +
time_format(current_event["end_time"]))
    remaining_starttimes.index(mins_till_next_eventstart)
    if BEEP_ON_EVENTSTART:
        for _ in range(3):
            magtag.peripherals.play_tone(1760, 0.1)
            time.sleep(0.2)
    sleep_time = mins_till_next_eventend + 1
else:
    print("Next up! ", next_up)
    magtag.set_background("bmps/"+next_up["graphic"])

    string = (
        "Coming up on "
        + next_up["day_of_week"]
        + " at "
        + time_format(next_up["start_time"])
    )
    magtag.set_text(string)
```

Sleep

In order to save battery power, the MagTag can go into deep sleep mode. The e-Ink display will stay the same, so you'll hardly know! The code will determine how long it should sleep so that it can wake up in time to change the display for the next upcoming event!

```
sleep_time = mins_till_next_eventstart

print("Sleeping for %d minutes" % sleep_time)
time.sleep(2)
magtag.exit_and_deep_sleep(sleep_time * 60)
```

Now, you're ready to go, and you'll know when the next show starts!

Or, you can customize the `events.json` file and graphics to use with your own recurring calendar events!