adafruit learning system

MagTag Sports Schedule Viewer

Created by John Park

Last updated on 2021-05-19 05:21:05 PM EDT

Guide Contents

Overview

This MagTag project puts your daily sports schedule right on your fridge. Check out the upcoming games in any sport covered by the ESPN API, including NCAA basketball, NHL hockey, NFL football, college baseball, Italia Serie A soccer, and more.

You can press a button to advance to the next game in the series, and any final games will display their score as well!

(Special thanks to my sports-watching brother-in-law Jim Kelly for consultation on this project. Go Bucks!)

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...

Out of Stock

Out of Stock

---

optional:

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...

Out of Stock

Out of Stock

Acrylic + Hardware Kit for Adafruit MagTag

Here is the perfect kit with two faceplate options for your MagTag, including a vivid Red Arrow plate and a dreamy white Cloud plate. And of course, the mounting hardware is included,...

$5.95

In Stock

Add to Cart

---

or

Adafruit MagTag Starter Kit - 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...

Out of Stock

Out of Stock

Install CircuitPython

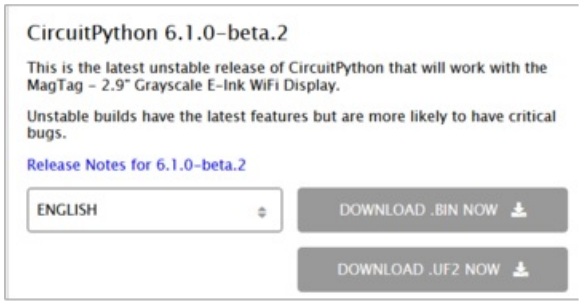
CircuitPython (https://adafru.it/tB7) is a derivative of MicroPython (https://adafru.it/BeZ) designed to simplify experimentation and education on low-cost microcontrollers. It makes it easier than ever to get prototyping by requiring no upfront desktop software downloads. Simply copy and edit files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

Set Up CircuitPython

Follow the steps to get CircuitPython installed on your MagTag.

https://adafru.it/OBd

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.

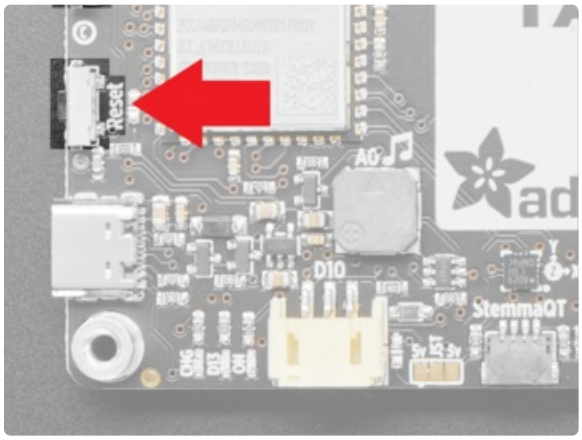
Option 1 - Load with UF2 Bootloader

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.**

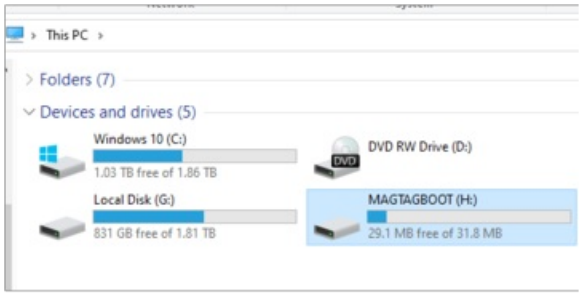
Still, try this first!

Try Launching UF2 Bootloader

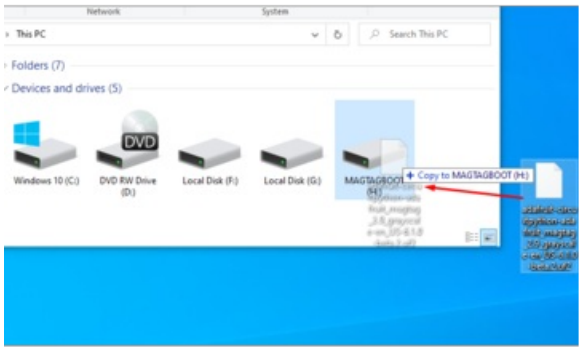
Loading CircuitPython by drag-n-drop UF2 bootloader is the easier way and we recommend it. If you have a MagTag where the front of the board is black, your MagTag came with UF2 already on it.



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.

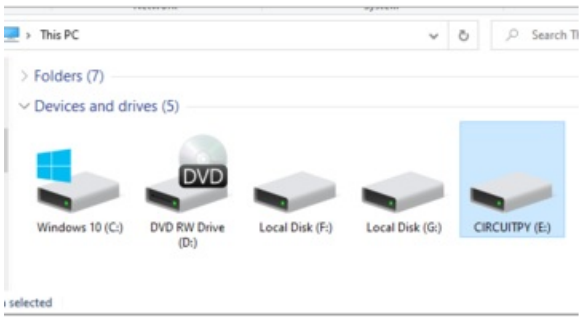


If the UF2 bootloader is installed, you will see a new disk drive appear called **MAGTAGBOOT**



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 Libraries

To use the internet-connectivity built into your ESP32-S2 with CircuitPython, you must first install a number of libraries. This page covers that process.

Adafruit CircuitPython Library Bundle

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

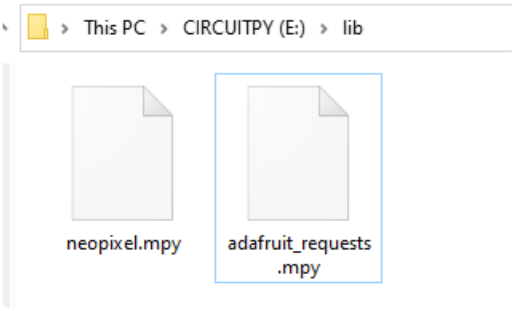
https://adafru.it/ENC

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_requests.mpy** - A requests-like library for HTTP commands.
- **neopixel.mpy** - Helper library to use NeoPixel LEDs, often built into the boards so they're great for quick feedback



Once you have added those files, please continue to the next page to set up and test Internet connectivity

CircuitPython Internet Test

Once you have CircuitPython installed and the minimum libraries installed we can get your board connected to the Internet.

To get connected, you will need to start by creating a **secrets.py** file.

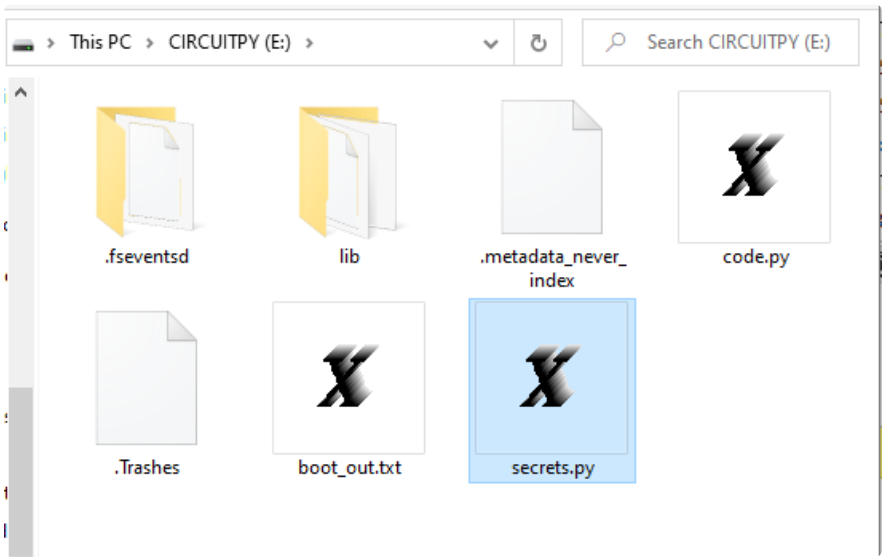
Secrets File

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 **secrets.py** file, that is in your **CIRCUITPY** drive, to hold secret/private/custom data. That way you can share your main project without worrying about accidentally sharing private stuff.

Your **secrets.py** file should look like this:

```
# 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

secrets = {
    'ssid' : 'home_wifi_network',
    'password' : 'wifi_password',
    'aio_username' : 'my_adafruit_io_username',
    'aio_key' : 'my_adafruit_io_key',
    'timezone' : "America/New_York", # http://worldtimeapi.org/timezones
    }
```

Copy and paste that text/code into a file called **secrets.py** and save it to your **CIRCUITPY** folder like so:



Inside is a python dictionary named secrets with a line for each entry. Each entry has an entry name (say `'ssid'`) and then a colon to separate it from the entry key `'home ssid'` and finally a comma ,

At a minimum you'll need to adjust the `ssid` and `password` for your local WiFi setup 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, just cause its called secrets doesn't mean you can't have general customization data in there!

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 **secrets.py** - keep that out of GitHub, Discord or other project-sharing sites.

Don't share your secrets.py file, it has your passwords and API keys in it!

Connect to WiFi

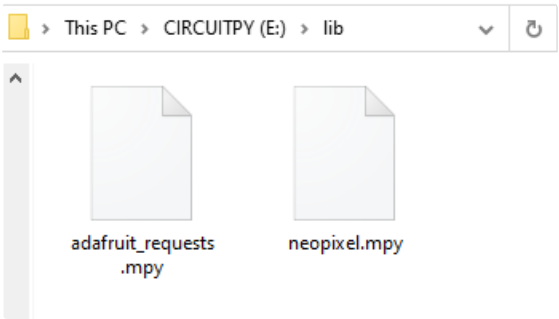
OK now you have your secrets setup - you can connect to the Internet using the Requests module.

First make sure you are running the latest version of Adafruit CircuitPython (https://adafru.it/Amd) for your board.

Next you'll need to install the necessary libraries to use the hardware--carefully follow the steps to find and install these libraries from Adafruit's CircuitPython library bundle (https://adafru.it/zdx). Our introduction guide has a great page on how to install the library bundle (https://adafru.it/ABU).

- **adafruit_requests**
- **neopixel**

Before continuing make sure your board's CIRCUITPY/lib folder or root filesystem has the above files copied over.



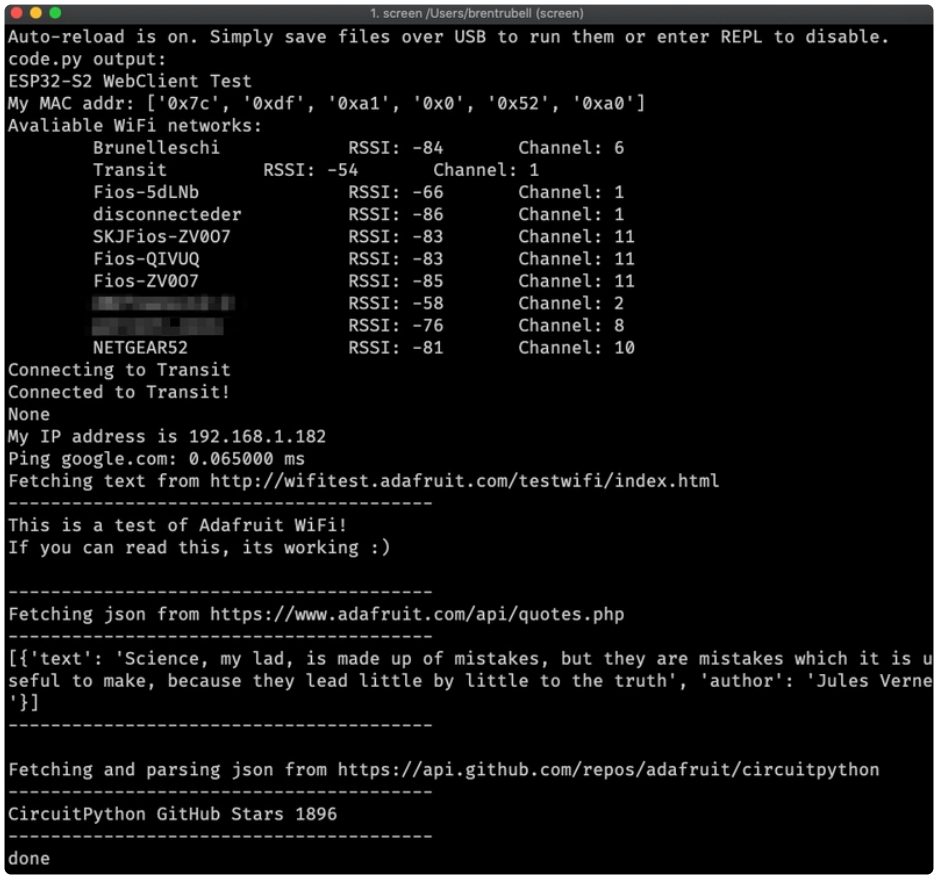
Once that's done, load up the following example using Mu or your favorite editor:

```
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"

# Get wifi details and more from a secrets.py file
try:
    from secrets import secrets
except ImportError:
    print("WiFi secrets are kept in secrets.py, please add them there!")
    raise

print("ESP32-S2 WebClient 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 %s"%secrets["ssid"])
wifi.radio.connect(secrets["ssid"], secrets["password"])
print("Connected to %s!"%secrets["ssid"])
print("My IP address is", wifi.radio.ipv4_address)

ipv4 = ipaddress.ip_address("8.8.4.4")
print("Ping google.com: %f ms" % (wifi.radio.ping(ipv4)*1000))

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

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

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

print()

print("Fetching and parsing json from", JSON_STARS_URL)
```

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

print("done")
```

And save it to your board. Make sure the file is named **code.py**.

Open up your REPL, you should see something like the following:



In order, the example code...

Checks the ESP32-S2's MAC address.

```
print("My MAC addr:", [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("Avaliable 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 **secrets.py** file, prints out its local IP address, and attempts to ping google.com to check its network connectivity.

```
print("Connecting to %s"%secrets["ssid"])
wifi.radio.connect(secrets["ssid"], secrets["password"])
print(print("Connected to %s!"%secrets["ssid"]))
print("My IP address is", wifi.radio.ipv4_address)

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

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 http **s** url for SSl connectivity.

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

OK you now have your ESP32-S2 board set up with a proper **secrets.py** file and can connect over the Internet. If not, check that your **secrets.py** file has the right ssid and password and retrace your steps until you get the Internet connectivity working!

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:

https://adafru.it/ENC

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

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 Sports Schedule Viewer

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 **magtag_sports_schedule.py** from the zip file and place on the **CIRCUITPY** drive, then rename it to **code.py**.

Copy the **/fonts** directory from the zip file linked below 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.

https://adafru.it/QF2

https://adafru.it/QF2


```
# SPDX-FileCopyrightText: 2021 Melissa LeBlanc-Williams and John Park for Adafruit Industries
# SPDX-License-Identifier: MIT

# MagTag Sports Schedule Viewer
# Be sure to add your wifi credentials to the secrets.py file

# Press D to advance to next game
# Press C to go back one game
# Press B to refresh the schedule (this takes a minute)
# Press A to advance to next sport (this takes a minute)


import time
import json
from adafruit_datetime import datetime, timedelta
from adafruit_magtag.magtag import MagTag

USE_24HR_TIME = False
TIME_ZONE_OFFSET = -8  # hours ahead or behind Zulu time, e.g. Pacific is -8
TIME_ZONE_NAME = "PST"

#  API info here https://www.gpwa.org/forum/espns-hidden-api-scores-stats-json-endpoints-
251149.html
#  Note, these will only work when the sport is in season.

SPORTS = [
    {
        "name": "NCAA Men's Basketball",
        #pylint: disable=line-too-long
        "url": "http://site.api.espn.com/apis/site/v2/sports/basketball/mens-college-
basketball/scoreboard",
    },
    {
        "name": "NCAA Wmn's Basketball",
        #pylint: disable=line-too-long
        "url": "http://site.api.espn.com/apis/site/v2/sports/basketball/womens-college-
```

```
        "url": "http://site.api.espn.com/apis/site/v2/sports/basketball/womens-college-
basketball/scoreboard",
    },
    {
        "name": "NHL Hockey",
        "url": "http://site.api.espn.com/apis/site/v2/sports/hockey/nhl/scoreboard",
    },
    {
        "name": "NFL Football",
        "url": "http://site.api.espn.com/apis/site/v2/sports/football/nfl/scoreboard",
    },
    {
        "name": "MLB Baseball",
        "url": "http://site.api.espn.com/apis/site/v2/sports/baseball/mlb/scoreboard",
    },
    {
        "name": "College Baseball",
        "url": "http://site.api.espn.com/apis/site/v2/sports/baseball/college-
baseball/scoreboard",
    },
    {
        "name": "NBA Basketball",
        "url": "http://site.api.espn.com/apis/site/v2/sports/basketball/nba/scoreboard",
    },
    {
        "name": "WNBA Basketball",
        "url": "http://site.api.espn.com/apis/site/v2/sports/basketball/wnba/scoreboard",
    },
    {
        "name": "College Football",
        "url": "http://site.api.espn.com/apis/site/v2/sports/football/college-
football/scoreboard",
    },
    {
        "name": "MLS Soccer",
        "url": "http://site.api.espn.com/apis/site/v2/sports/soccer/usa.1/scoreboard",
    },
    {
        "name": "Premiere League",
        "url": "http://site.api.espn.com/apis/site/v2/sports/soccer/eng.1/scoreboard",
    },
    {
        "name": "Italian Serie A",
        "url": "http://site.api.espn.com/apis/site/v2/sports/soccer/ita.1/scoreboard",
    },
    {
        "name": "German Bundesliga",
        "url": "http://site.api.espn.com/apis/site/v2/sports/soccer/ger.1/scoreboard",
    },
]


current_game = 0
#  You can cycle among different sports with the A button, or set it here:
current_sport = 0
sports_data = []
```

```
EVENTS_LOCATION = ["events"]
STATUS_LOCATION = ["status", "type", "description"]
BROADCAST_LOCATION = ["competitions", 0, "broadcasts"]
IS_FINAL_LOCATION = ["competitions", 0, "status", "type", "id"]
SCORES_LOCATION = ["competitions", 0, "competitors"]
SCORE_0_LOCATION = ["competitions", 0, "competitors", 0, "score"]
SCORE_1_LOCATION = ["competitions", 0, "competitors", 1, "score"]

months = [
    "Jan",
    "Feb",
    "March",
    "April",
    "May",
    "June",
    "July",
    "Aug",
    "Sept",
    "Oct",
    "Nov",
    "Dec",
]

# Set up the MagTag with the JSON data parameters
magtag = MagTag(
    url=SPORTS[current_sport]["url"],
)


def format_date(iso_formatted_date):
    if iso_formatted_date is None:
        return "When: Unavailable"
    date = datetime.fromisoformat(iso_formatted_date[:-1])
    date += timedelta(hours=TIME_ZONE_OFFSET)

    if USE_24HR_TIME:
        timestring = "%d:%02d %s" % (date.hour, date.minute, TIME_ZONE_NAME)
    elif date.hour > 12:
        timestring = "%d:%02d pm %s" % (
            abs((date.hour - 12) % 12),
            date.minute,
            TIME_ZONE_NAME,
        )
    else:
        timestring = "%d:%02d am %s" % (date.hour, date.minute, TIME_ZONE_NAME)

    return "%s %d, %s" % (months[date.month - 1], date.day, timestring)


def format_score(scores, is_final):
    home_score = scores[0]["score"]
    away_score = scores[1]["score"]
    if not home_score or not away_score:
        return "Unavailable"
    if int(is_final) == 3:
        return "%s - %s" % (home_score, away_score)
    return " "
```

```
def format_available(value):
    if value is None:
        return "Unavailable"
    return value


def format_broadcast(value):
    if not value:
        value = "N/A"
    else:
        value = magtag.network.json_traverse(value, [0, "names", 0])
    return "Airing on: " + value


def get_game_number():
    return "Game %d of %d" % (current_game + 1, len(sports_data))


def play_tone(frequency, color=None):
    magtag.peripherals.neopixel_disable = False
    if color:
        magtag.peripherals.neopixels.fill(color)
    magtag.peripherals.play_tone(frequency, 0.2)
    magtag.peripherals.neopixel_disable = True


def update_labels():
    # Set the labels for the current game data
    magtag.set_text(SPORTS[current_sport]["name"], 0, False)
    magtag.set_text(sports_data[current_game]["name"], 1, False)
    magtag.set_text(sports_data[current_game]["date"], 2, False)
    magtag.set_text(sports_data[current_game]["broadcast"], 3, False)
    magtag.set_text(sports_data[current_game]["status"], 4, False)
    magtag.set_text(get_game_number(), 5, False)
    magtag.set_text(sports_data[current_game]["score"], 6)
    # wait 2 seconds for display to complete
    time.sleep(2)


def fetch_sports_data(reset_game_number=True):
    # Fetches and parses data for all games for the current sport
    #pylint: disable=global-statement
    global sports_data, current_game, current_sport
    magtag.url = SPORTS[current_sport]["url"]
    sports_data.clear()
    raw_data = json.loads(magtag.fetch(auto_refresh=False))
    events = raw_data["events"]
    for event in events:
        game_data = {}
        game_data["name"] = format_available(event["name"])
        game_data["date"] = format_date(event["date"])
        game_data["status"] = "Game status: " + format_available(
            magtag.network.json_traverse(event, STATUS_LOCATION)
        )
        game_data["broadcast"] = format_broadcast(
```

```
        game_data["broadcast"] = format_broadcast(
            magtag.network.json_traverse(event, BROADCAST_LOCATION)
        )
        scores = magtag.network.json_traverse(event, SCORES_LOCATION)
        is_final = magtag.network.json_traverse(event, IS_FINAL_LOCATION)
        game_data["score"] = format_score(scores, is_final)
        sports_data.append(game_data)
    if reset_game_number or current_game > len(sports_data):
        current_game = 0
    update_labels()


# Sports Name
magtag.add_text(
    text_font="/fonts/Lato-Bold-ltd-25.bdf", text_position=(10, 15), is_data=False
)

# Game Name
magtag.add_text(
    text_font="/fonts/Arial-Bold-12.pcf",
    text_wrap=35,
    line_spacing=0.75,
    text_position=(10, 70),
    is_data=False,
)

# Date
magtag.add_text(text_font="/fonts/Arial-12.bdf", text_position=(10, 40), is_data=False)

# Broadcast Information
magtag.add_text(
    text_font="/fonts/Arial-Italic-12.bdf", text_position=(10, 100), is_data=False
)

# Game Status
magtag.add_text(
    text_font="/fonts/Arial-Italic-12.bdf", text_position=(10, 120), is_data=False
)

# Game Number
magtag.add_text(
    text_font="/fonts/Arial-Italic-12.bdf", text_position=(190, 38), is_data=False
)

# Score
magtag.add_text(
    text_font="/fonts/Arial-Bold-24.bdf", text_position=(170, 94), is_data=False
)

fetch_sports_data()

while True:
    if magtag.peripherals.button_a_pressed:  # switch to next sport
        play_tone(620, 0x000033)
        current_sport += 1
        if current_sport >= len(SPORTS):
            current_sport = 0
```

```
        fetch_sports_data()
    elif magtag.peripherals.button_b_pressed:  # re-fetch data
        play_tone(220, 0x330000)
        fetch_sports_data(False)
    elif magtag.peripherals.button_c_pressed:  # display previous game
        play_tone(350, 0x330000)
        current_game -= 1
        if current_game < 0:
            current_game = len(sports_data) - 1
        update_labels()
    elif magtag.peripherals.button_d_pressed:  # display next game
        play_tone(440, 0x003300)
        current_game += 1
        if current_game >= len(sports_data):
            current_game = 0
        update_labels()

    time.sleep(0.01)
```

How It Works

When you start it up, the MagTag will get online using your WiFi access point credentials you entered in the **secrets.py** file. It will then take a minute to download the entire json file from the ESPN API for the sport selected (initially NCAA Men's Basketball, but you can switch sports with the MagTag's A button, or with the `current_sport` variable in the code).

Press the D button to advance to the next game in the schedule, Press C to go back one game, in case you missed some important details!

If you want to refresh the json data for the day, press the B button.

Libraries

You'll first import the libraries `time`, `json`, `adafruit_datetime` (to deal with timezone/dateline conversion, since your timezone may differ from the Zulu time in the API), and `adafruit_magtag`.

Next, you'll set some variables related to timezone:

```
USE_24HR_TIME = False
TIME_ZONE_OFFSET = -8  # hours ahead or behind Zulu time, e.g. Pacific is -8
TIME_ZONE_NAME = "PST"
```

API JSON URLs

These are a number of the sports available from the ESPN API. You can read more details and get more sports here (https://adafru.it/QF3).

```
SPORTS = [
    {
        "name": "NCAA Men's Basketball",
        #pylint: disable=line-too-long
        "url": "http://site.api.espn.com/apis/site/v2/sports/basketball/mens-college-
basketball/scoreboard",
    },
    {
        "name": "NCAA Wmn's Basketball",
        #pylint: disable=line-too-long
        "url": "http://site.api.espn.com/apis/site/v2/sports/basketball/womens-college-
basketball/scoreboard",
    },
    {
        "name": "NHL Hockey",
        "url": "http://site.api.espn.com/apis/site/v2/sports/hockey/nhl/scoreboard",
    },
    {
        "name": "NFL Football",
        "url": "http://site.api.espn.com/apis/site/v2/sports/football/nfl/scoreboard",
    },
    {
        "name": "MLB Baseball",
        "url": "http://site.api.espn.com/apis/site/v2/sports/baseball/mlb/scoreboard",
    },
    {
        "name": "College Baseball",
        "url": "http://site.api.espn.com/apis/site/v2/sports/baseball/college-
baseball/scoreboard",
    },
    {
        "name": "NBA Basketball",
        "url": "http://site.api.espn.com/apis/site/v2/sports/basketball/nba/scoreboard",
    },
    {
        "name": "WNBA Basketball",
        "url": "http://site.api.espn.com/apis/site/v2/sports/basketball/wnba/scoreboard",
    },
    {
        "name": "College Football",
        "url": "http://site.api.espn.com/apis/site/v2/sports/football/college-
football/scoreboard",
    },
    {
        "name": "MLS Soccer",
        "url": "http://site.api.espn.com/apis/site/v2/sports/soccer/usa.1/scoreboard",
    },
    {
        "name": "Premiere League",
        "url": "http://site.api.espn.com/apis/site/v2/sports/soccer/eng.1/scoreboard",
    },
    {
        "name": "Italian Serie A",
        "url": "http://site.api.espn.com/apis/site/v2/sports/soccer/ita.1/scoreboard",
    },
    {
        "name": "German Bundesliga",
```

```
        "name": "German Bundesliga",
        "url": "http://site.api.espn.com/apis/site/v2/sports/soccer/ger.1/scoreboard",
    },
]
```

Game Variables

These variables are used to track which sport is being used and which game is currently displayed, as well as setting up the variable that will be used to parse the json date. The json files in the ESPN API are very consistent, so the locations of, say, the event scores for a final are consistent from game to game and sport to sport. Yay for standards!

```
current_game = 0
#  You can cycle among different sports with the A button, or set it here:
current_sport = 0
sports_data = []

EVENTS_LOCATION = ["events"]
STATUS_LOCATION = ["status", "type", "description"]
BROADCAST_LOCATION = ["competitions", 0, "broadcasts"]
IS_FINAL_LOCATION = ["competitions", 0, "status", "type", "id"]
SCORES_LOCATION = ["competitions", 0, "competitors"]
SCORE_0_LOCATION = ["competitions", 0, "competitors", 0, "score"]
SCORE_1_LOCATION = ["competitions", 0, "competitors", 1, "score"]
```

MagTag Setup & Date Time Function

You'll set up the MagTag device, with the URL argument, and then create a function to format the date and time.

```
magtag = MagTag(
    url=SPORTS[current_sport]["url"],
)


def format_date(iso_formatted_date):
    if iso_formatted_date is None:
        return "When: Unavailable"
    date = datetime.fromisoformat(iso_formatted_date[:-1])
    date += timedelta(hours=TIME_ZONE_OFFSET)

    if USE_24HR_TIME:
        timestring = "%d:%02d %s" % (date.hour, date.minute, TIME_ZONE_NAME)
    elif date.hour > 12:
        timestring = "%d:%02d pm %s" % (
            abs((date.hour - 12) % 12),
            date.minute,
            TIME_ZONE_NAME,
        )
    else:
        timestring = "%d:%02d am %s" % (date.hour, date.minute, TIME_ZONE_NAME)

    return "%s %d, %s" % (months[date.month - 1], date.day, timestring)
```

Functions, Functions, Functions

The next few functions are the dark inner workings for formatting all the little bits of data we need to display.

```
def format_score(scores, is_final):
    home_score = scores[0]["score"]
    away_score = scores[1]["score"]
    if not home_score or not away_score:
        return "Unavailable"
    if int(is_final) == 3:
        return "%s - %s" % (home_score, away_score)
    return " "


def format_available(value):
    if value is None:
        return "Unavailable"
    return value


def format_broadcast(value):
    if not value:
        value = "N/A"
    else:
        value = magtag.network.json_traverse(value, [0, "names", 0])
    return "Airing on: " + value
```

```
def get_game_number():
    return "Game %d of %d" % (current_game + 1, len(sports_data))


def play_tone(frequency, color=None):
    magtag.peripherals.neopixel_disable = False
    if color:
        magtag.peripherals.neopixels.fill(color)
    magtag.peripherals.play_tone(frequency, 0.2)
    magtag.peripherals.neopixel_disable = True


def update_labels():
    # Set the labels for the current game data
    magtag.set_text(SPORTS[current_sport]["name"], 0, False)
    magtag.set_text(sports_data[current_game]["name"], 1, False)
    magtag.set_text(sports_data[current_game]["date"], 2, False)
    magtag.set_text(sports_data[current_game]["broadcast"], 3, False)
    magtag.set_text(sports_data[current_game]["status"], 4, False)
    magtag.set_text(get_game_number(), 5, False)
    magtag.set_text(sports_data[current_game]["score"], 6)
    # wait 2 seconds for display to complete
    time.sleep(2)


def fetch_sports_data(reset_game_number=True):
    # Fetches and parses data for all games for the current sport
    #pylint: disable=global-statement
    global sports_data, current_game, current_sport
    magtag.url = SPORTS[current_sport]["url"]
    sports_data.clear()
    raw_data = json.loads(magtag.fetch(auto_refresh=False))
    events = raw_data["events"]
    for event in events:
        game_data = {}
        game_data["name"] = format_available(event["name"])
        game_data["date"] = format_date(event["date"])
        game_data["status"] = "Game status: " + format_available(
            magtag.network.json_traverse(event, STATUS_LOCATION)
        )
        game_data["broadcast"] = format_broadcast(
            magtag.network.json_traverse(event, BROADCAST_LOCATION)
        )
        scores = magtag.network.json_traverse(event, SCORES_LOCATION)
        is_final = magtag.network.json_traverse(event, IS_FINAL_LOCATION)
        game_data["score"] = format_score(scores, is_final)
        sports_data.append(game_data)
    if reset_game_number or current_game > len(sports_data):
        current_game = 0
    update_labels()
```

Labels

The text to be displayed on the MagTag is set up next.

```
magtag.add_text(
    text_font="/fonts/Lato-Bold-ltd-25.bdf", text_position=(10, 15), is_data=False
)

# Game Name
magtag.add_text(
    text_font="/fonts/Arial-Bold-12.pcf",
    text_wrap=35,
    line_spacing=0.75,
    text_position=(10, 70),
    is_data=False,
)

# Date
magtag.add_text(text_font="/fonts/Arial-12.bdf", text_position=(10, 40), is_data=False)

# Broadcast Information
magtag.add_text(
    text_font="/fonts/Arial-Italic-12.bdf", text_position=(10, 100), is_data=False
)

# Game Status
magtag.add_text(
    text_font="/fonts/Arial-Italic-12.bdf", text_position=(10, 120), is_data=False
)

# Game Number
magtag.add_text(
    text_font="/fonts/Arial-Italic-12.bdf", text_position=(190, 38), is_data=False
)

# Score
magtag.add_text(
    text_font="/fonts/Arial-Bold-24.bdf", text_position=(170, 94), is_data=False
)
```

Fetch

The last bit of setup is to fetch the sports data from the selected URL using `fetch_sports_data()`

This causes the data to be downloaded, parsed, formatted, and finally displayed on the MagTag screen!

Main Loop

The main loop of the program watches for button presses and changes or fetches new data depending on the button pressed.

```
while True:
    if magtag.peripherals.button_a_pressed:  # switch to next sport
        play_tone(620, 0x000033)
        current_sport += 1
        if current_sport >= len(SPORTS):
            current_sport = 0
        fetch_sports_data()
    elif magtag.peripherals.button_b_pressed:  # re-fetch data
        play_tone(220, 0x330000)
        fetch_sports_data(False)
    elif magtag.peripherals.button_c_pressed:  # display previous game
        play_tone(350, 0x330000)
        current_game -= 1
        if current_game < 0:
            current_game = len(sports_data) - 1
        update_labels()
    elif magtag.peripherals.button_d_pressed:  # display next game
        play_tone(440, 0x003300)
        current_game += 1
        if current_game >= len(sports_data):
            current_game = 0
        update_labels()

    time.sleep(0.01)
```