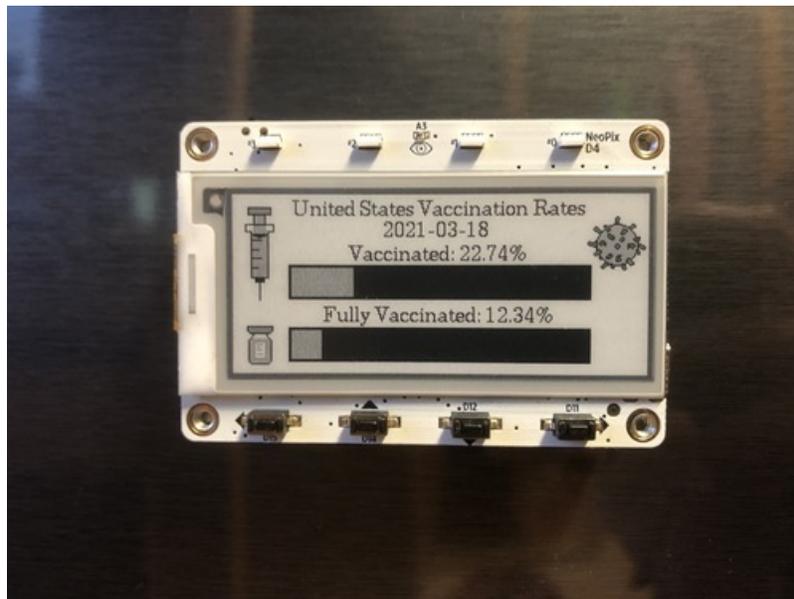


## Adafruit MagTag COVID Vaccination Percent Tracker

Created by Dylan Herrada

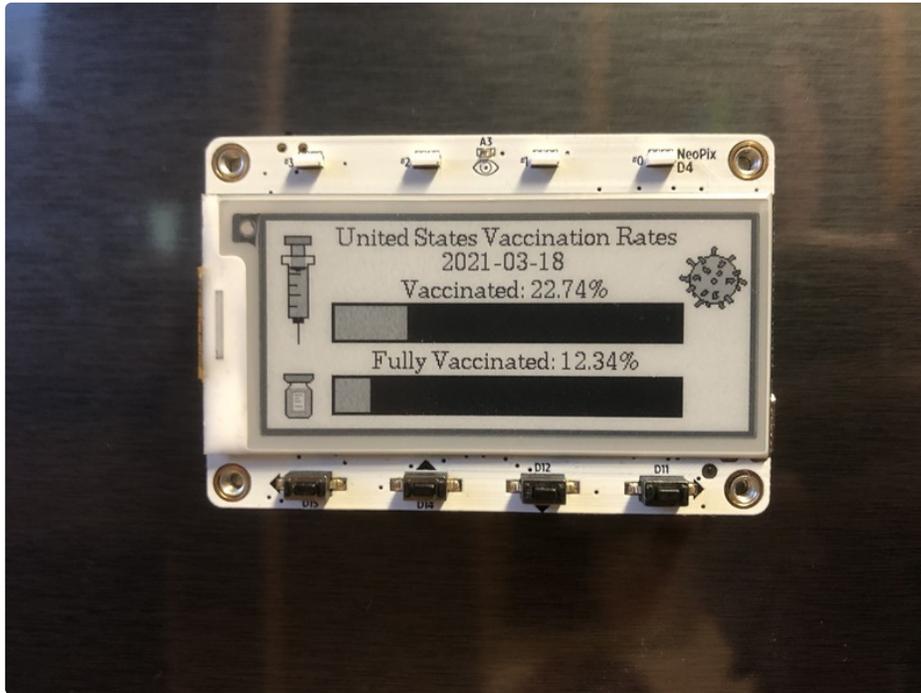


Last updated on 2021-10-22 11:46:03 AM EDT

## Guide Contents

Guide Contents	2
Overview	3
Parts	3
Install CircuitPython	5
Set Up CircuitPython	5
Option 1 - Load with UF2 Bootloader	6
Try Launching UF2 Bootloader	6
Option 2 - Use esptool to load BIN file	7
Option 3 - Use Chrome Browser To Upload BIN file	8
CircuitPython Internet Libraries	9
Adafruit CircuitPython Library Bundle	9
CircuitPython Internet Test	10
Secrets File	10
Connect to WiFi	11
Getting The Date & Time	16
Step 1) Make an Adafruit account	16
Step 2) Sign into Adafruit IO	16
Step 3) Get your Adafruit IO Key	16
Step 4) Upload Test Python Code	17
MagTag-Specific CircuitPython Libraries	20
Get Latest Adafruit CircuitPython Bundle	20
Secrets	20
Code the Vaccination Tracker	22
Installing the Project Code	22
Code Run Through	25

# Overview



Vaccinations for COVID are well underway in most countries, and what better way to keep track of vaccinations for your country or region than with a battery-powered, refrigerator magnet that's connected to the internet. This project makes use of the eInk display and the internet capabilities of the ESP32-S2 to automatically display and update the latest vaccination numbers on your Adafruit MagTag.

## Parts

This kit contains all the parts except for a cable:

### [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...

\$44.95

In Stock

Add to Cart

Or get the pieces separately:

### [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...

\$34.95

In Stock

Add to Cart

---

### 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...

\$2.50

In Stock

Add to Cart

---

### 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...

\$6.95

In Stock

Add to Cart

---

Get a USB-A to USB-C cable to connect your computer to the MagTag:

### USB Type A to Type C Cable - 1ft - 0.3 meter

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...

\$3.95

In Stock

Add to Cart

---

### 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...

\$4.95

In Stock

Add to Cart

---

# Install CircuitPython

[CircuitPython \(https://adafru.it/tB7\)](https://adafru.it/tB7) is a derivative of [MicroPython \(https://adafru.it/BeZ\)](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.

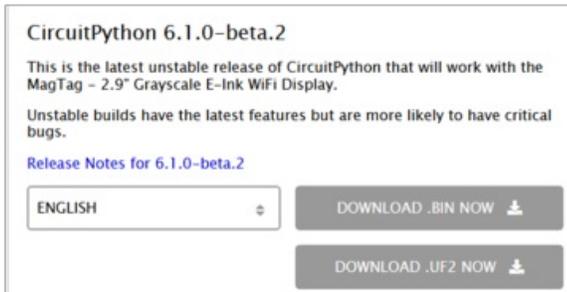
<https://adafru.it/OBd>

<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

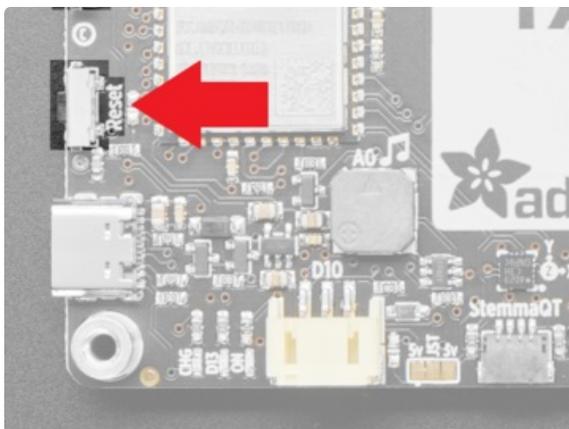
This is by far the easiest way to load CircuitPython. However it requires your board has the UF2 bootloader installed. Some early boards do not (we hadn't written UF2 yet!) - in which case you can load using the built in ROM bootloader.

Still, try this first!



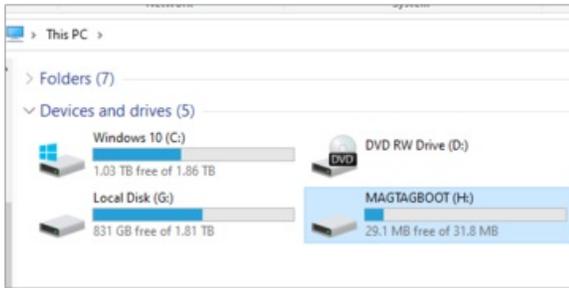
### Try Launching UF2 Bootloader

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.

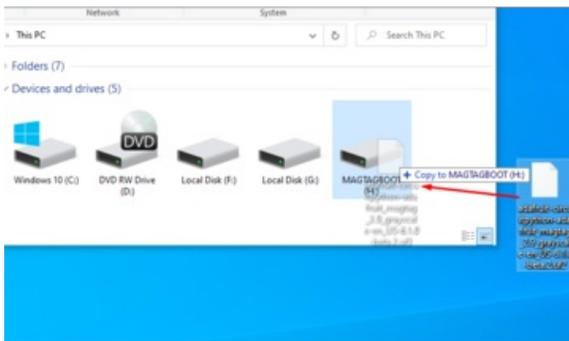


Launch UF2 by **double-clicking** the Reset button (the one next to the USB C port). You may have to try a few times to get the timing right.

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 \(the latest version of the UF2 bootloader fixes this warning\) \(https://adafruit.it/Pfk\)](#)

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!

```
kattni@obscure:~$ python ./esptool.py --port /dev/cu.usbmodem01 --afterno.reset
write_flash 0x0 ~/adafruit-circuitpython-adafruit_astro_esp32s2-en_US-20201103-5a7925_bin
esptool.py v3.0-dev
Serial port /dev/cu.usbmodem01
Connecting...
Detecting chip type... ESP32-S2
Chip is ESP32-S2
Features: WiFi, ADC and temperature sensor calibration in BLK2 of eFuse
Crystal is 40MHz
MAC: 7c:d:f:a1:00:4a:a2
Uploading stub...
Running stub...
Stub running...
Configuring flash size...
Compressed 1305184 bytes to 844014...
Wrote 1305184 bytes (844014 compressed) at 0x00000000 in 11.8 seconds (effective 878.2 kbit/s)...
Hash of data verified.
Leaving...
Staying in bootloader.
```

Follow the initial steps found in the [Run esptool and check connection section of the ROM Bootloader page](https://adafru.it/OBc) (<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) (<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:

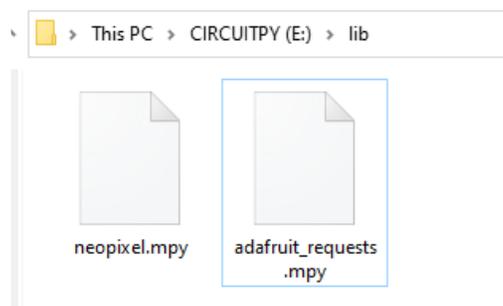
<https://adafru.it/ENC>

<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

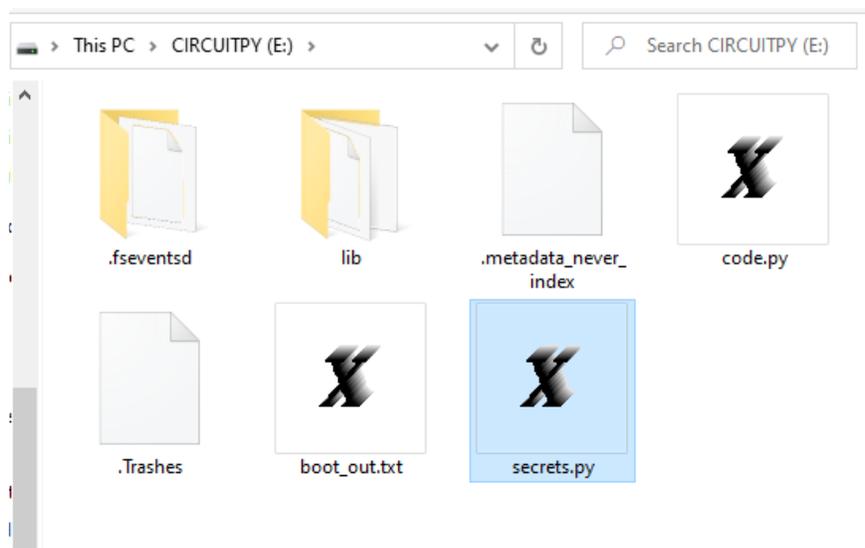
We expect people to share tons of projects as they build CircuitPython WiFi widgets. What we want to avoid is people accidentally sharing their passwords or secret tokens and API keys. So, we designed all our examples to use a **secrets.py** file, that is in your **CIRCUITPY** drive, to hold secret/private/custom 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

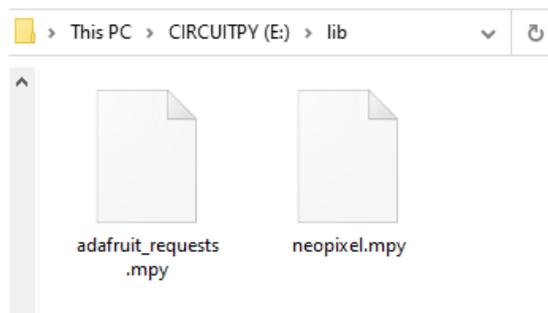
OK now you have your secrets setup - you can connect to the Internet using the Requests module.

First make sure you are running the [latest version of Adafruit CircuitPython](https://adafru.it/Amd) (<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) (<https://adafru.it/zdx>). Our introduction guide has [a great page on how to install the library bundle](https://adafru.it/ABU) (<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)
response = requests.get(JSON_STARS_URL)
print("-" * 40)
print("Github Python Github Stars")

```

```
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:

```
1. screen /Users/brentrubell (screen)
Auto-reload is on. Simply save files over USB to run them or enter REPL to disable.
code.py output:
ESP32-S2 WebClient Test
My MAC addr: ['0x7c', '0xdf', '0xa1', '0x0', '0x52', '0xa0']
Availible WiFi networks:
  Brunelleschi          RSSI: -84      Channel: 6
  Transit              RSSI: -54      Channel: 1
  Fios-5dLNb           RSSI: -66      Channel: 1
  disconnectededer     RSSI: -86      Channel: 1
  SKJFios-ZV007       RSSI: -83      Channel: 11
  Fios-QIVUQ          RSSI: -83      Channel: 11
  Fios-ZV007          RSSI: -85      Channel: 11
  [REDACTED]           RSSI: -58      Channel: 2
  [REDACTED]           RSSI: -76      Channel: 8
  NETGEAR52           RSSI: -81      Channel: 10
Connecting to Transit
Connected to Transit!
None
My IP address is 192.168.1.182
Ping google.com: 0.065000 ms
Fetching text from http://wifitest.adafruit.com/testwifi/index.html
-----
This is a test of Adafruit WiFi!
If you can read this, its working :)
-----
Fetching json from https://www.adafruit.com/api/quotes.php
-----
[{'text': 'Science, my lad, is made up of mistakes, but they are mistakes which it is u
seful to make, because they lead little by little to the truth', 'author': 'Jules Verne
'}]
-----
Fetching and parsing json from https://api.github.com/repos/adafruit/circuitpython
-----
CircuitPython GitHub Stars 1896
-----
done
```

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 new instance of the [requests](https://adafru.it/E9o) (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("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!

# 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

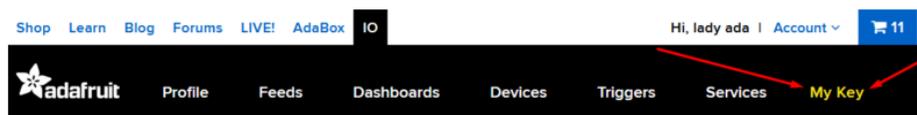
It's free! Visit <https://accounts.adafruit.com/> (<https://adafru.it/dyy>) to register and make an account if you do not already have one

## Step 2) Sign into Adafruit IO

Head over to [io.adafruit.com](https://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)

## YOUR ADAFRUIT IO KEY



Your Adafruit IO Key should be kept in a safe place and treated with the same care as your Adafruit username and password. People who have access to your Adafruit IO Key can view all of your data, create new feeds for your account, and manipulate your active feeds.



If you need to regenerate a new Adafruit IO Key, all of your existing programs and scripts will need to be manually changed to the new key.

Username

Active Key

REGENERATE KEY

[Hide Code Sample](#)

Go to your secrets.py file on your CIRCUITPY drive and add three lines for `aio_username`, `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

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
}
```

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 worldtimeapi 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 ssl
import wifi
import socketpool
import adafruit_requests
import secrets

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

# Get our username, key and desired timezone
aio_username = secrets["aio_username"]
aio_key = secrets["aio_key"]
location = secrets.get("timezone", None)
TIME_URL = "https://io.adafruit.com/api/v2/%s/integrations/time/strftime?x-aio-key=%s" %
(aio_username, aio_key)
TIME_URL += "&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 %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))

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!

```
Connecting to adafruit
Connected to adafruit!
My IP address is 10.0.1.148
Ping google.com: 0.008000 ms
Fetching text from https://io.adafruit.com/api/v2/[REDACTED]/integrations/time/strftime?x-aio-
key=[REDACTED]&fmt=%25Y-%25m-%25d+%25H%3A%25M%3A%25S.%25L+%25j+%25u+%25z+%25Z
-----
2020-12-05 18:51:32.145 340 6 -0500 EST
-----
```

Note at the end you will get the date, time, and your timezone! If so, you have correctly configured your **secrets.py** 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:

<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 \(https://adafru.it/P3b\)](https://adafru.it/P3b).

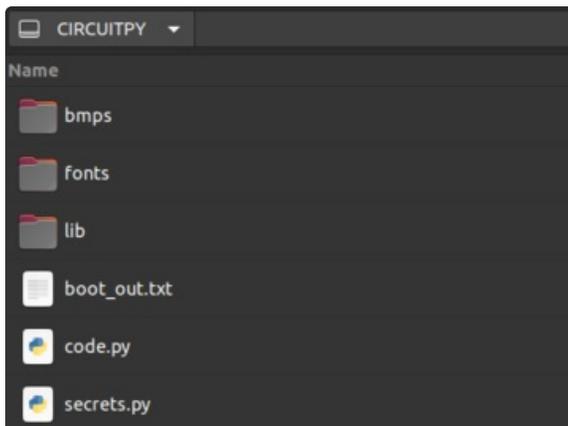
# Code the Vaccination Tracker

## Installing the Project Code

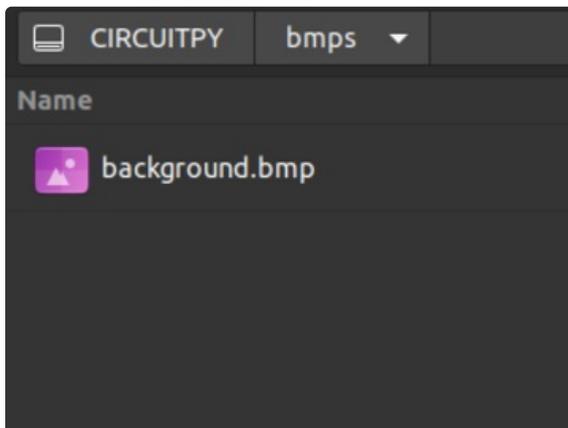
Download a zip of the project by clicking 'Download: Project Zip' in the preview of `code.py` below.

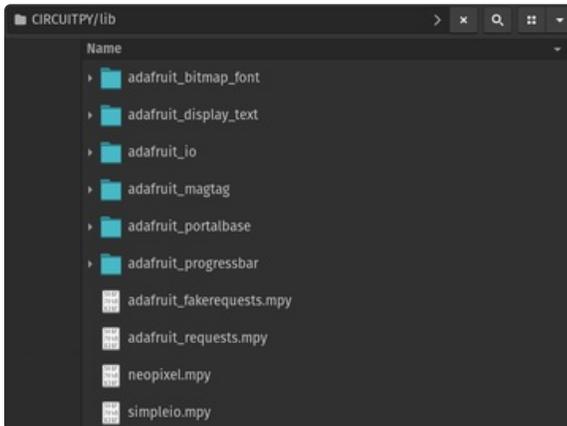
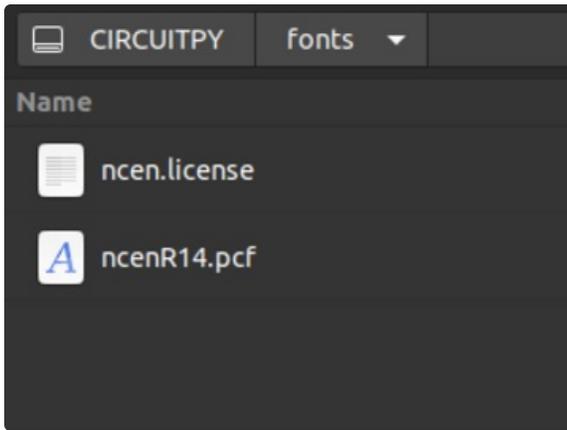
After unzipping the file, copy its contents to the **CIRCUITPY** drive which appears when the MagTag is connected to your computer via a USB cable and turned on via a small on/off switch onboard.

In addition to the libraries you installed in the 'MagTag-Specific CircuitPython Libraries' section, you'll also have to install `adafruit_progressbar`.



After you've copied everything over, this is what the **CIRCUITPY** drive should look like.





Be sure you have the libraries and secrets.py file complete and loaded onto the CIRCUITPY drive also or the code will not run properly.

```
from adafruit_magtag.magtag import MagTag
from adafruit_progressbar.progressbar import ProgressBar

# Set up where we'll be fetching data from
DATA_SOURCE = "https://raw.githubusercontent.com/owid/covid-19-
data/master/public/data/vaccinations/country_data/United%20States.csv" # pylint: disable=line-
too-long
# Find data for other countries/states here:
# https://github.com/owid/covid-19-data/tree/master/public/data/vaccinations

magtag = MagTag(url=DATA_SOURCE)
magtag.network.connect()

magtag.add_text(
    text_font="/fonts/ncenR14.pcf",
    text_position=(
        (magtag.graphics.display.width // 2) - 1,
        8,
    ),
    text_anchor_point=(0.5, 0.5),
    is_data=False,
) # Title
```

```

magtag.add_text(
    text_font="/fonts/ncenR14.pcf",
    text_position=(
        (magtag.graphics.display.width // 2) - 1,
        23,
    ),
    text_anchor_point=(0.5, 0.5),
    is_data=False,
) # Date

magtag.add_text(
    text_font="/fonts/ncenR14.pcf",
    text_position=(
        (magtag.graphics.display.width // 2) - 1,
        40,
    ),
    text_anchor_point=(0.5, 0.5),
    is_data=False,
) # Vaccinated text

magtag.add_text(
    text_font="/fonts/ncenR14.pcf",
    text_position=(
        (magtag.graphics.display.width // 2) - 1,
        85,
    ),
    text_anchor_point=(0.5, 0.5),
    is_data=False,
) # Fully vaccinated text

BAR_WIDTH = magtag.graphics.display.width - 80
BAR_HEIGHT = 25
BAR_X = magtag.graphics.display.width // 2 - BAR_WIDTH // 2

progress_bar = ProgressBar(
    BAR_X, 50, BAR_WIDTH, BAR_HEIGHT, 1.0, bar_color=0x999999, outline_color=0x000000
)

progress_bar_1 = ProgressBar(
    BAR_X, 95, BAR_WIDTH, BAR_HEIGHT, 1.0, bar_color=0x999999, outline_color=0x000000
)

magtag.graphics.splash.append(progress_bar)
magtag.graphics.splash.append(progress_bar_1)
magtag.graphics.set_background("/bmps/background.bmp")

def l_split(line):
    line_list = []
    print(line)
    while "," in line:
        if line[0] == "'":
            temp = line.split("'", 2)[1]
            line_list.append(temp)
            line = line.split("'", 2)[2][1:]
        else:
            temp = line.split(", ", 1)

```

```

        temp, line = line.split(", ", 1)
        line_list.append(temp)
    line_list.append(line)
    return line_list

try:
    table = magtag.fetch().split("\n")
    columns = l_split(table[0])
    latest = l_split(table[-2])
    print(columns)
    print(latest)
    value = dict(zip(columns, latest))
    print("Response is", value)
    print(value)

    vaccinated = int(value["people_vaccinated"]) / 331984513
    fully_vaccinated = int(value["people_fully_vaccinated"]) / 331984513

    magtag.set_text(f"{value['location']} Vaccination Rates", 0, False)
    magtag.set_text(value["date"], 1, False)
    magtag.set_text("Vaccinated: {:.2f}%".format(vaccinated * 100), 2, False)
    magtag.set_text(
        "Fully Vaccinated: {:.2f}%".format(fully_vaccinated * 100), 3, False
    )

    progress_bar.progress = vaccinated
    progress_bar_1.progress = fully_vaccinated

    magtag.refresh()

    SECONDS_TO_SLEEP = 24 * 60 * 60 # Sleep for one day

except (ValueError, RuntimeError) as e:
    print("Some error occured, retrying in one hour! -", e)
    seconds_to_sleep = 60 * 60 # Sleep for one hour

print(f"Sleeping for {SECONDS_TO_SLEEP} seconds")
magtag.exit_and_deep_sleep(SECONDS_TO_SLEEP)

```

## Code Run Through

First, the code imports the two required libraries.

```

from adafruit_magtag.magtag import MagTag
from adafruit_progressbar import ProgressBar

```

Next, the code defines where it'll be getting the data from, initializes the `MagTag` object and tells it to connect to the network defined in `secrets.py`.

```
# Set up where we'll be fetching data from
DATA_SOURCE = "https://raw.githubusercontent.com/owid/covid-19-
data/master/public/data/vaccinations/country_data/United%20States.csv"
# Find data for other countries/states here:
# https://github.com/owid/covid-19-data/tree/master/public/data/vaccinations

magtag = MagTag(url=DATA_SOURCE)
magtag.network.connect()
```

Then, the four text fields are defined. These are individually set by passing the `set_text` method a number that corresponds to the order the specific text field was created.

```
magtag.add_text(
    text_font="/fonts/ncenR14.pcf",
    text_position=((magtag.graphics.display.width // 2) - 1, 8,),
    text_anchor_point=(0.5, 0.5),
    is_data=False,
) # Title

magtag.add_text(
    text_font="/fonts/ncenR14.pcf",
    text_position=((magtag.graphics.display.width // 2) - 1, 23,),
    text_anchor_point=(0.5, 0.5),
    is_data=False,
) # Date

magtag.add_text(
    text_font="/fonts/ncenR14.pcf",
    text_position=((magtag.graphics.display.width // 2) - 1, 40,),
    text_anchor_point=(0.5, 0.5),
    is_data=False,
) # Vaccinated text

magtag.add_text(
    text_font="/fonts/ncenR14.pcf",
    text_position=((magtag.graphics.display.width // 2) - 1, 85,),
    text_anchor_point=(0.5, 0.5),
    is_data=False,
) # Fully vaccinated text
```

After that, the progress bars are set up. The top one will track the percent of people who have been partially vaccinated and the bottom one will track the percent of people who have been fully vaccinated.

```

BAR_WIDTH = magtag.graphics.display.width - 80
BAR_HEIGHT = 25
BAR_X = magtag.graphics.display.width // 2 - BAR_WIDTH // 2

progress_bar = ProgressBar(
    BAR_X, 50, BAR_WIDTH, BAR_HEIGHT, 1.0, bar_color=0x999999, outline_color=0x000000
)

progress_bar_1 = ProgressBar(
    BAR_X, 95, BAR_WIDTH, BAR_HEIGHT, 1.0, bar_color=0x999999, outline_color=0x000000
)

magtag.graphics.splash.append(progress_bar)
magtag.graphics.splash.append(progress_bar_1)
magtag.graphics.set_background("/bmps/background.bmp")

```

The code now enters the try part of the try/except block. It first gets the csv data from the URL defined above, then splits it by line, gets the latest line, and splits it into a list so it can be used easier. After that, the percent of people who have been partially and fully vaccinated is calculated. The four text fields are then all set to their respective values and the progress bars are updated. Assuming everything has gone well so far, the display is then refreshed.

Finally, the MagTag sleeps for a day, at which point this code will run again.

Please note that if you change the country you are getting the data for, you will also have to change the population (the denominators in vaccinated and fully\_vaccinated) to match that.

```

try:
    value = magtag.fetch().split("\n")[-2].split(",")
    print("Response is", value)

    vaccinated = int(value[-2]) / 331984513
    fully_vaccinated = int(value[-1]) / 331984513

    magtag.set_text(f"{value[0]} Vaccination Rates", 0, False)
    magtag.set_text(value[1], 1, False)
    magtag.set_text("Vaccinated: {:.2f}%".format(vaccinated * 100), 2, False)
    magtag.set_text(
        "Fully Vaccinated: {:.2f}%".format(fully_vaccinated * 100), 3, False
    )

    progress_bar.progress = vaccinated
    progress_bar_1.progress = fully_vaccinated

    magtag.refresh()

    seconds_to_sleep = 24 * 60 * 60 # Sleep for one day
    print(f"Sleeping for {seconds_to_sleep} seconds")
    magtag.exit_and_deep_sleep(seconds_to_sleep)

```

However, if an issue has occurred, the code prints it out and tries again.

```
except (ValueError, RuntimeError) as e:  
    print("Some error occurred, retrying! -", e)
```

