adafruit learning system

What’s Fresh Today? In-Season Produce Reminder for Adafruit MagTag

Created by Phillip Burgess

Last updated on 2021-05-19 05:20:41 PM EDT

Guide Contents

Overview

We live in a bright, if somewhat detached, future. In just about any city, 365 days a year, snow or desert heat, you can buy a pineapple. Not canned...an actual tropical pineapple fruit from across the globe, in your hand. It's amazing when you think about it. Behind the scenes, *it takes an army* to get that pineapple (or any other peculiar food) to you alive...inordinate energy and labor, and all that entails. *So* tasty, but it does give one pause.

Consider the benefits of **locally-grown fruits and vegetables** sold at farmers' markets, produce stands and smaller independent grocers:

- Vine-ripened foods taste *amazing,* so much more flavorful than canned, frozen or store-shelf ripened. If you've ever wondered "why are tomatoes considered a fruit?", now you'll know. *Sweetness and flavor!*
- Local growers often have *nifty varieties* too delicate for industrial-scale agriculture and distribution. Flavors, colors, sizes and shapes you seldom see at big grocers. *Romanesco broccoli looks exactly like 1990s computer graphics.*
- *Fewer middlemen* means farm workers have a better chance for a living wage.
- *A lower carbon footprint* as your food arrives from a nearby county rather than four states away, and with less processing, packaging and waste.

Not to push an agenda — let's be practical, pineapple is *delicious,* among other fruits and veggies you might otherwise miss year 'round. The notion is to *prioritize local seasonal selections,* for some good reasons above, then round out anything else during routine grocery trips.

Okay then, how to start? It's not just about *what* can grow locally, but *when?* Without freezers or cans, just fresh off the land, your options *change throughout the year.* Meal planning suddenly isn't so simple. This knowledge goes back *generations*...we've started with data from the venerable ***Farmers' Almanac*** *(https://adafru.it/Pa6)* and fully automated it. Punch in your location and MagTag keeps track of the seasons and offers suggestions. High tech, but pragmatic.

This currently only works for the lower 48 United States. Information for Alaska, Hawaii (Pineapple!), and non-US countries is not present in the dataset.

Parts Required

The MagTag starter kit includes an e-ink development board, LiPoly battery and magnetic feet...bring-your-own USB type A to type C cable. Or the individual pieces can be rounded up separately...

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

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

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

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 battery...

$6.95

In Stock

Add to Cart

Also needed:

- **WiFi network** (802.11 b/g/n)
- A desktop or laptop **computer** is required for initial setup: any **text editor** will suffice
- **An** Adafruit IO **(https://adafru.it/BRB) account.** If you've previously purchased from Adafruit and created an account, this is automagic (and free)!

Install CircuitPython

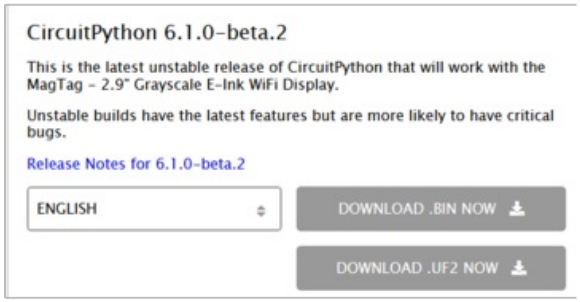
CircuitPython (https://adafru.it/tB7) is a derivative of MicroPython (https://adafru.it/BeZ) designed to simplify experimentation and education on low-cost microcontrollers. It makes it easier than ever to get prototyping by requiring no upfront desktop software downloads. Simply copy and edit files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

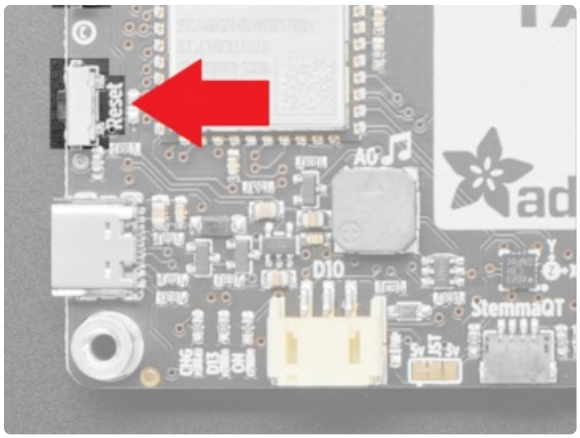
Option 1 - Load with UF2 Bootloader

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!

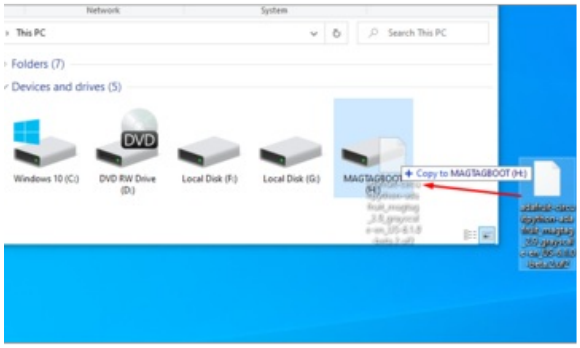
Try Launching UF2 Bootloader

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:

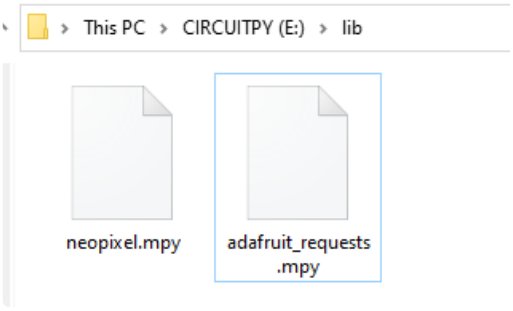
https://adafru.it/ENC

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.

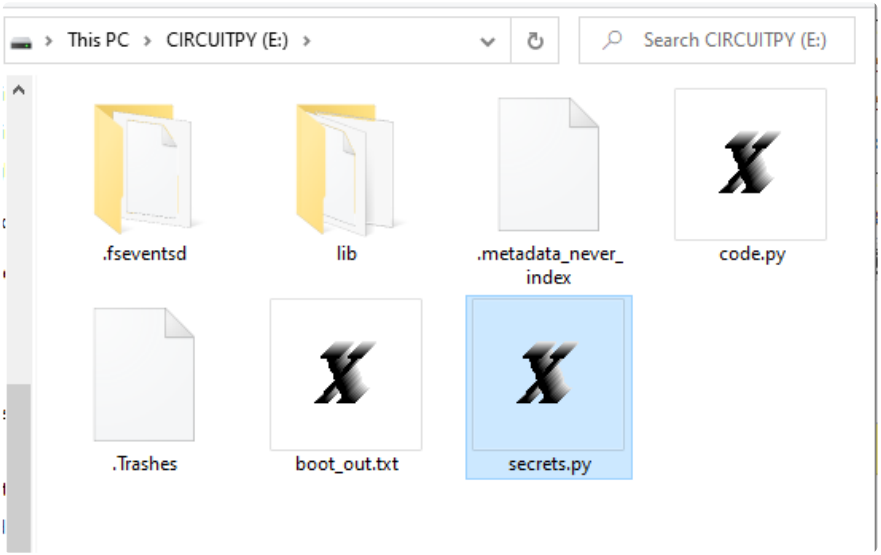
Secrets File

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

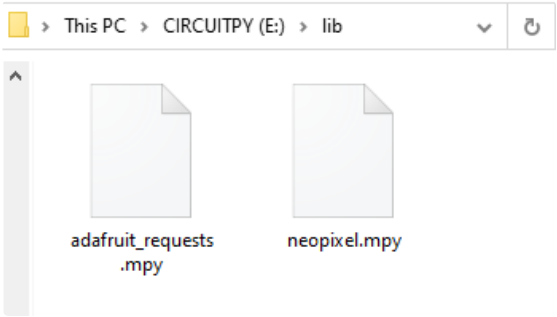
OK now you have your secrets setup - you can connect to the Internet using the Requests module.

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.



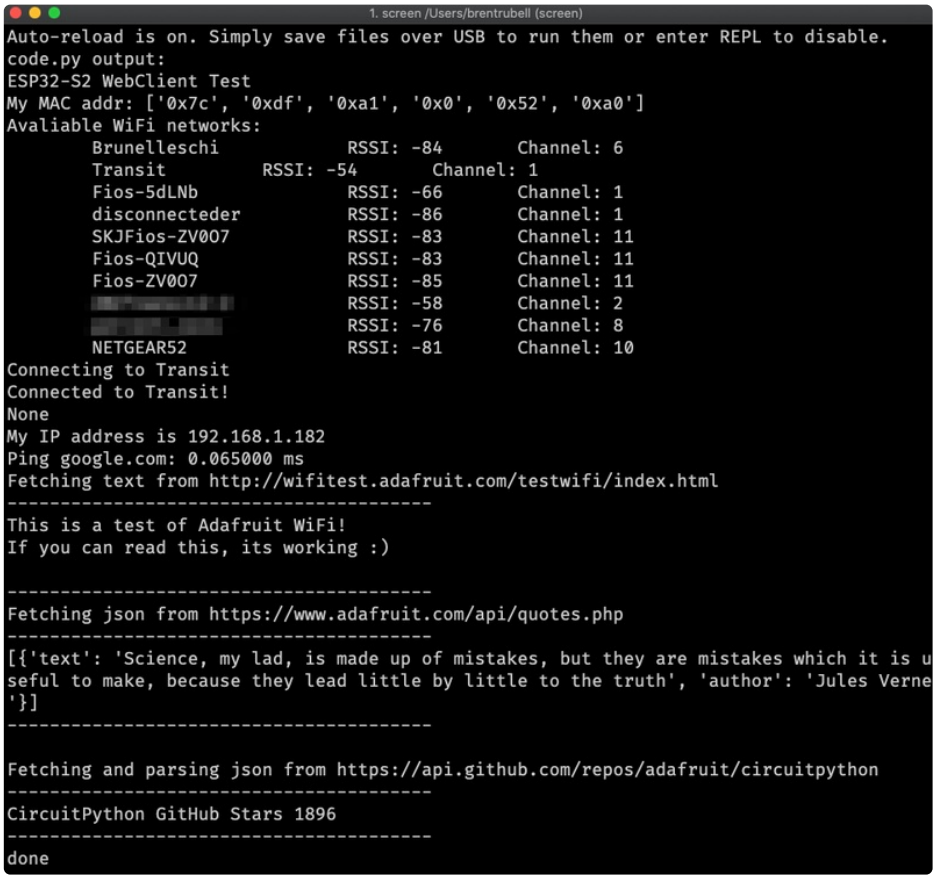
Once that's done, load up the following example using Mu or your favorite editor:

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

```
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!

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

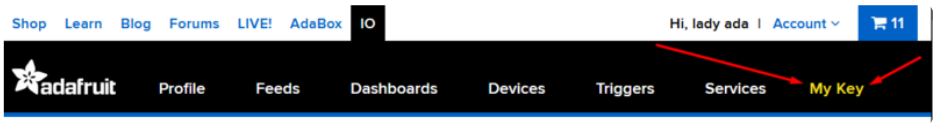
It's free! Visit https://accounts.adafruit.com/ (https://adafru.it/dyy) to register and make an account if you do not already have one

Step 2) Sign into Adafruit IO

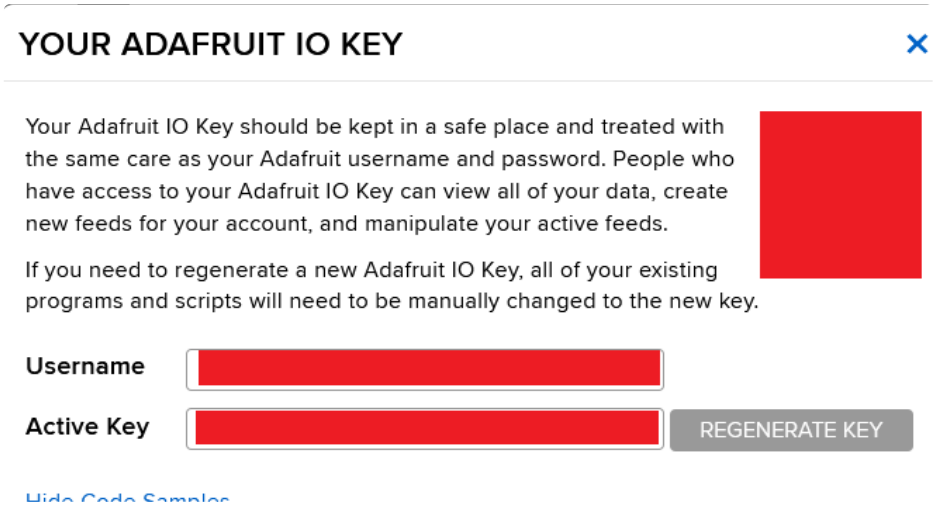
Head over to io.adafruit.com (https://adafru.it/fsU) and click **Sign In** to log into IO using your Adafruit account. It's free and fast to join.

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 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!

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

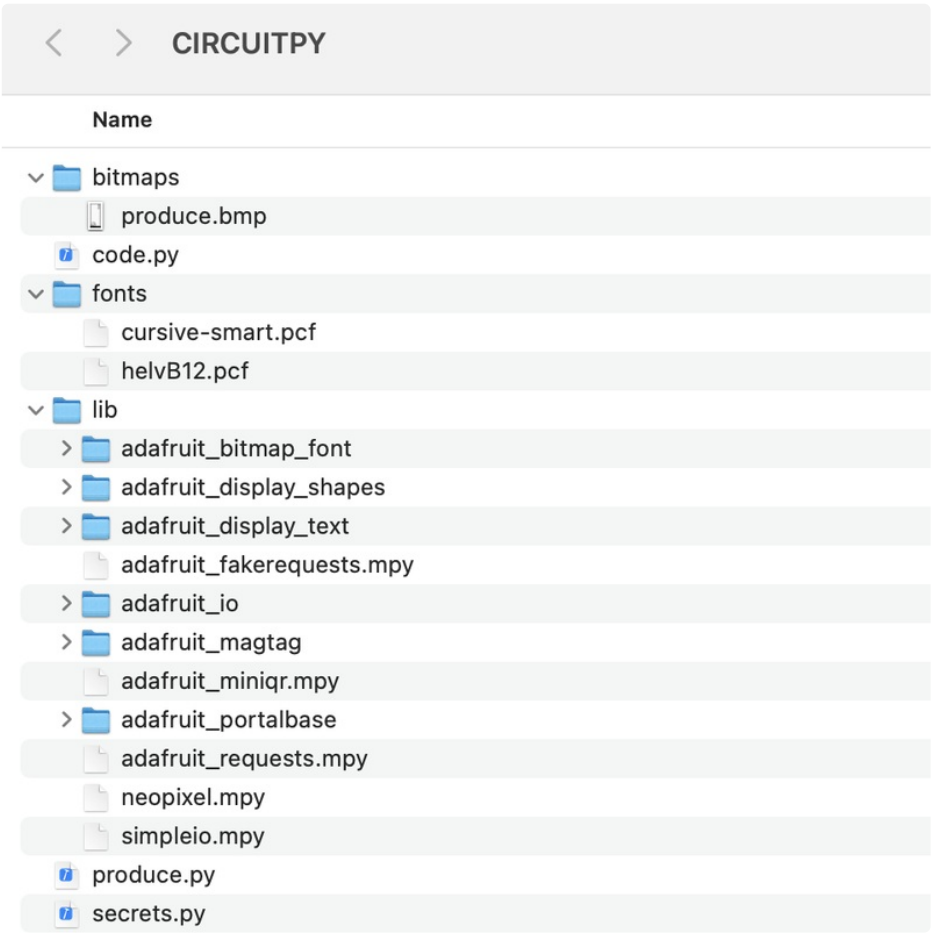
directory (top level) of your **CIRCUITPY** drive. If you do not intend to connect to wireless, it does not need to have valid data in it. Here's more info on the secrets.py file (https://adafru.it/P3b).

Install Code and Graphics

Back up any existing code or files you want to keep from your MagTag CIRCUITPY drive.

Fetch the files for our Seasonal Produce project from Github by clicking "**Download Project Zip**" at the top of the code below - this will fetch all the necessary files in one zip file which you can then use in the project.

Here's a map of all this project's required images, fonts and code on the **CIRCUITPY** drive:




```
"""
Seasonal produce finder for Adafruit MagTag w/CircuitPython 6.1 or later.
Lists in-season fruits and vegetables for a user's location and season.
"Smart cursive" BDF font by Thomas A. Fine, helvB12 from Xorg fonts.
"""

# pylint: disable=import-error, line-too-long
import time
from secrets import secrets
import rtc
```

```
from adafruit_display_shapes.rect import Rect
from adafruit_magtag.magtag import MagTag
from produce import Produce


# CONFIGURABLE SETTINGS and ONE-TIME INITIALIZATION -----------------------

TWELVE_HOUR = True # If set, show 12-hour vs 24-hour (e.g. 3:00 vs 15:00)
DD_MM = False      # If set, show DD/MM instead of MM/DD dates
# Location of produce data (file:// or http:// or https://):
JSON_URL =
'https://raw.githubusercontent.com/adafruit/Adafruit_Learning_System_Guides/master/MagTag_Seasonal
oduce/produce.json'

# Location is configured in secrets.py. If location is not contained there,
# default value below will be used.
LOCATION = secrets.get('location', 'NY') # default to 'NY'

PRODUCE = Produce(JSON_URL, LOCATION)
MAGTAG = MagTag(rotation=0) # Portrait (vertical) display


# SOME UTILITY FUNCTIONS --------------------------------------------------

def hh_mm(time_struct, twelve_hour=True):
    """ Given a time.struct_time, return a string as H:MM or HH:MM, either
        12- or 24-hour style depending on twelve_hour flag.
    """
    if twelve_hour:
        if time_struct.tm_hour > 12:
            hour_string = str(time_struct.tm_hour - 12) # 13-23 -> 1-11 (pm)
        elif time_struct.tm_hour > 0:
            hour_string = str(time_struct.tm_hour) # 1-12
        else:
            hour_string = '12' # 0 -> 12 (am)
    else:
        hour_string = '{hh:02d}'.format(hh=time_struct.tm_hour)
    return hour_string + ':{mm:02d}'.format(mm=time_struct.tm_min)


# GRAPHICS INITIALIZATION -------------------------------------------------

MAGTAG.graphics.set_background('bitmaps/produce.bmp')

# Produce list is inserted at this position
MAGTAG.add_text(
    text_font='/fonts/cursive-smart.pcf',
    text_position=(3, 2),
    line_spacing=1.0,
    text_anchor_point=(0, 0),  # top left
    is_data=False, # we'll set this text manually
)

# Add 14-pixel-tall black bar at bottom of display. It's a distinct layer
# (not just background) to appear on top of produce list if it runs long.
MAGTAG.graphics.splash.append(Rect(0, MAGTAG.graphics.display.height - 14,
                                   MAGTAG.graphics.display.width,
```

```
                                    MAGTAG.graphics.display.height, fill=0x0))

# Center white text label over black bar to show last update time
# (Initially a placeholder, string is not assigned to label until later)
MAGTAG.add_text(
    text_font='/fonts/helvB12.pcf',
    text_position=(MAGTAG.graphics.display.width // 2,
                   MAGTAG.graphics.display.height - 1),
    text_color=0xFFFFFF,
    text_anchor_point=(0.5, 1),
    is_data=False, # we'll set this text manually later
)


# MAIN LOOP ----------------------------------------------------------------

# FYI: Not really a "loop" -- deep sleep makes the whole system restart on
# wake, this only needs to run once.

try:
    MAGTAG.network.connect() # Sneak this in last, as WiFi uses power
    print('Updating time')
    MAGTAG.get_local_time()
    NOW = rtc.RTC().datetime
    print(NOW)

    print('Updating produce')
    PRODUCE.fetch(MAGTAG)

    # Set the "Updated" date and time label
    if DD_MM:
        DATE = '%d/%d' % (NOW.tm_mday, NOW.tm_mon)
    else:
        DATE = '%d/%d' % (NOW.tm_mon, NOW.tm_mday)
    MAGTAG.set_text('Updated %s %s' % (DATE, hh_mm(NOW, TWELVE_HOUR)), 1,
                    auto_refresh=False)

    # Look up the matching produce data (returned as list of strings)
    PRODUCE_LIST = PRODUCE.in_season(NOW.tm_mon)
    NUM_ITEMS = len(PRODUCE_LIST)
    print('Produce list: ', PRODUCE_LIST)
    # List one item per line since some may be long
    VEGGIE_LIST = ''
    for item in PRODUCE_LIST:
        VEGGIE_LIST += '\n'.join(MAGTAG.wrap_nicely(item, 15)) + '\n'
    MAGTAG.set_text(VEGGIE_LIST) # Update list on the display

    time.sleep(2) # Allow refresh to finish before deep sleep
    print('Zzzz time')
    MAGTAG.exit_and_deep_sleep(24 * 60 * 60) # 24 hour snooze

except RuntimeError as error:
    # If there's an error above, no harm, just try again in ~15 minutes.
    # Usually it's a common network issue or time server hiccup.
    print('Retrying in 15 min - ', error)
    MAGTAG.exit_and_deep_sleep(15 * 60) # 15 minute snooze
```

Unzip this file after downloading.

The **bitmaps** and **fonts** folders should be copied into the **CIRCUITPY** root directory. If folders with these names already exist, copy the individual .BMP and .BDF files *into* the corresponding folders.

code.py and **produce.py** should also be copied to the **CIRCUITPY** root directory.

One of the files in the project folder — **produce.json** — **does *not* get copied**. Our code will pull the latest from the internet each time it's needed, so it always has the latest information.

One additional file — **secrets.py** — isn't distributed in the project folder...if you don't already have this file from a prior MagTag project, we'll create this file on the next page.

If you run out of space when copying items to CIRCUITPY: make a **backup** of any files currently on that drive, then delete files that aren't related to this project to free up space.

*The following page gets mirrored into several MagTag guides...being a catch-all for many things, it kind of goes off into the weeds. Only the **first part** is important — getting the **settings.py** file created if you haven't before, setting up WiFi network credentials and timezone in there — then you can skip ahead.*

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.

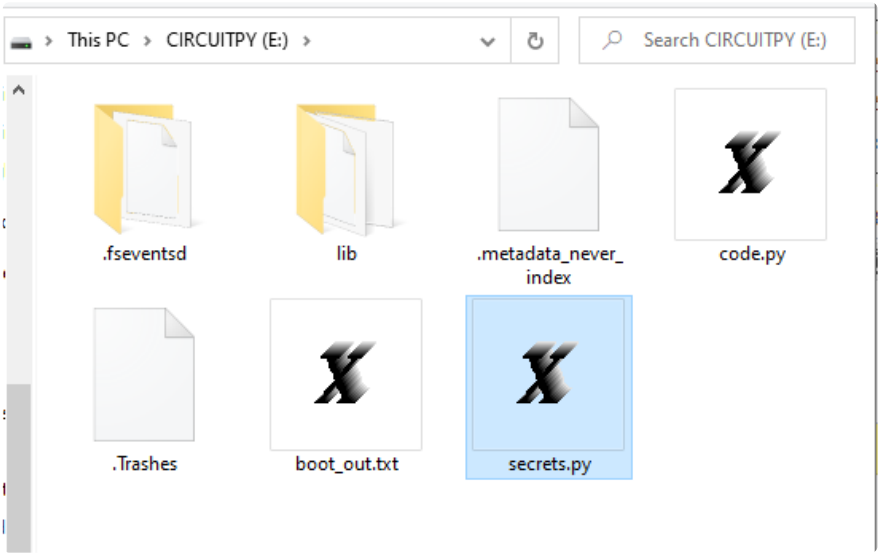
Secrets File

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

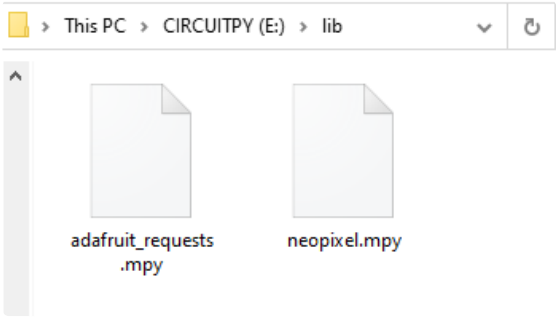
OK now you have your secrets setup - you can connect to the Internet using the Requests module.

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.



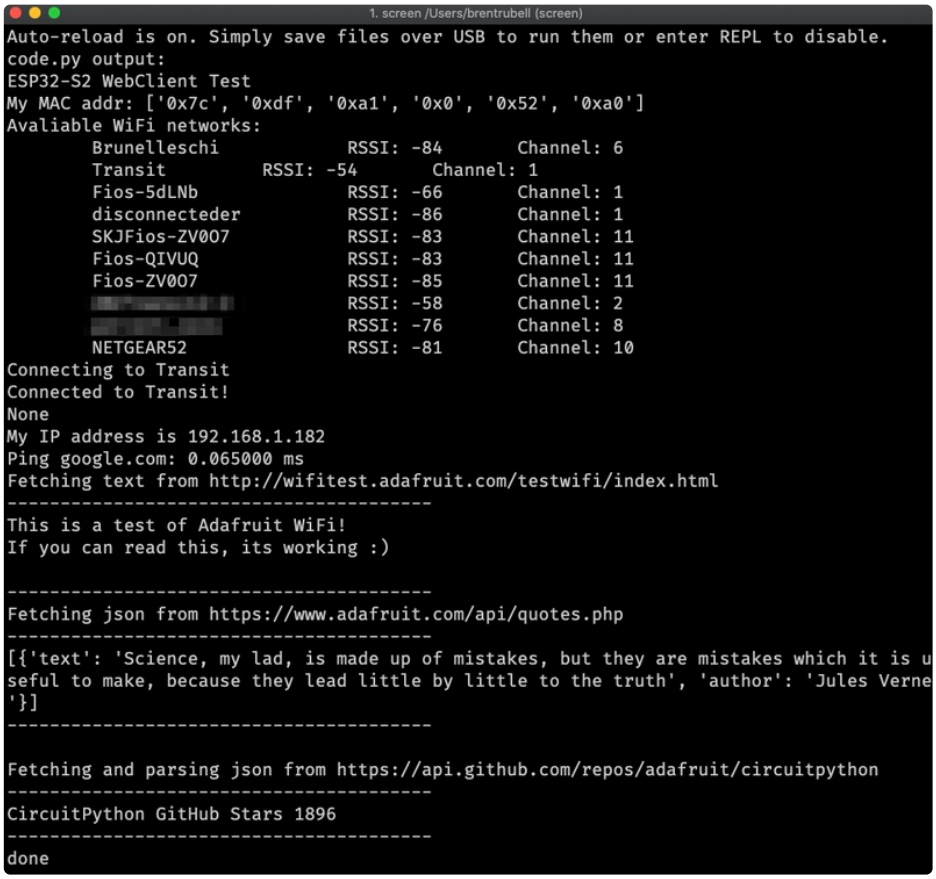
Once that's done, load up the following example using Mu or your favorite editor:

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

```
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!

Configure Location

A few more items need to go in the **secrets.py** file described on the prior page.

In addition to **WiFi network credentials** and **timezone**, our veggie code also needs:

- `aio_username` and `aio_key` matching your Adafruit IO **(https://adafru.it/BRB) credentials**. If logged into your account at adafruit.com, click the "**IO**" link in the top navigation bar, then "**My Key**."
- `location` is the **2-letter postal code** for your state, in quotes: e.g. **'NY'** or **'TX'**.

This currently only works for the lower 48 United States. Information for Alaska, Hawaii, and non-US countries is not present in the dataset.

```
secrets = {
    'ssid'         : 'WiFi-Network-Name',
    'password'     : 'WiFi-Network-Password',
    'aio_username' : 'Your_Adafruit_IO_Username',
    'aio_key'      : 'Your_Adafruit_IO_Key',
    'timezone'     : "America/New_York",
    'location'     : 'NY'
}
```

After editing and saving **secrets.py**, tap the reset button and within a minute or so you'll see some veggies!

If the project does NOT start up: most likely the WiFi credentials are incorrect, or something is wrong with the **secrets.py** file syntax...make sure every quote, comma and colon is there and in the right place.

Usage

Once configured for your location, there's literally **nothing to do**. The device will quietly go about its business, checking for changes in the seasons and updating the list once a day. There are no buttons to press or menus to navigate.

When it's time to go shopping, you can bring the MagTag list with you, or snap a picture with your smartphone before heading out.

Other Configurable Settings

Aside from the location (in **secrets.py**), there are some other configurable settings that can be adjusted in the code itself.

In **code.py**, starting around line 18 are a few global variables. Each of these is briefly commented in the code, but for posterity...

`TWELVE_HOUR` selects whether to display the "Updated" time as 12-hour vs. 24-hour time (e.g. 3:00 vs. 15:00). Default is `True`, use 12-hour format.

`DD_MM` selects whether the "Updated" date is displayed as day/month vs. month/day. Default is `False` — use month/day formatting — since the data is U.S.-centric for now, and that's the more common date format here.

`JSON_URL` is the internet location of the agricultural data file. The default is a long URL pointing to a file hosted on Github (https://adafru.it/P9C) that might see updates from time to time. It's possible to use a `file://` URL instead, and place this data on the **CIRCUITPY** drive...but that won't automatically pick up any future changes to the data.