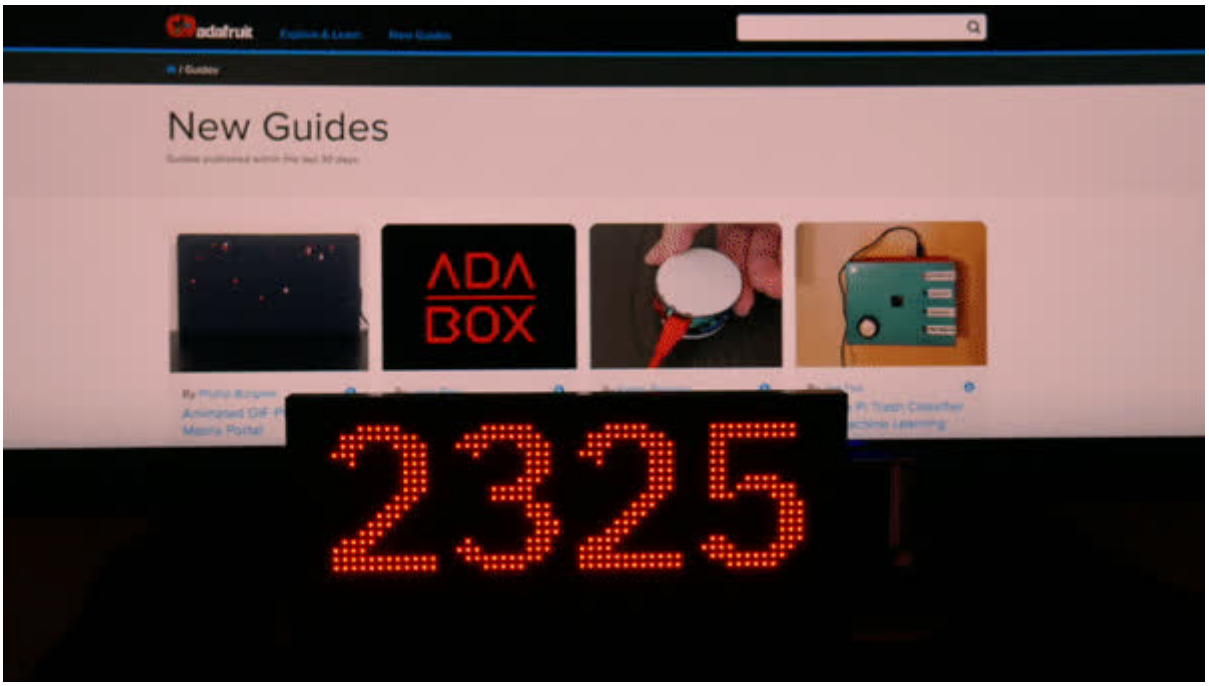
Matrix Portal New Guide Scroller

Created by Brent Rubell

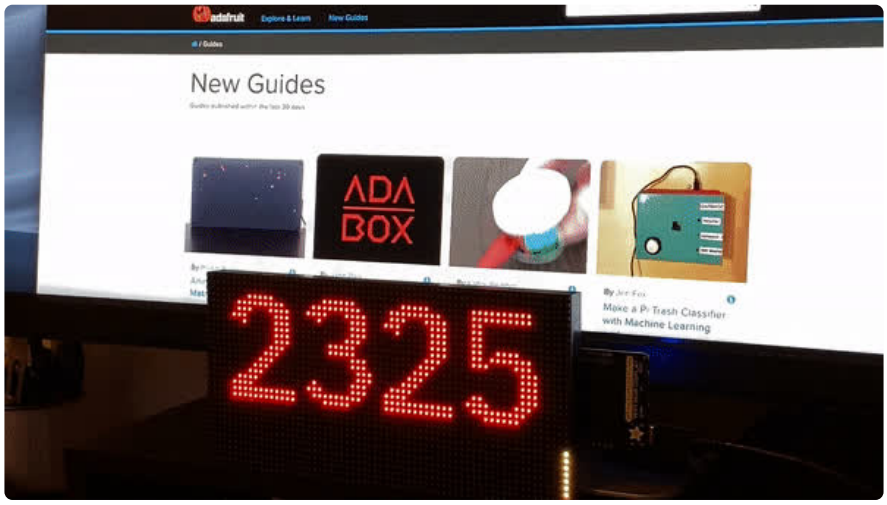


https://learn.adafruit.com/matrix-portal-new-guide-scroller

Last updated on 2024-11-18 01:00:23 PM EST

Table of Contents

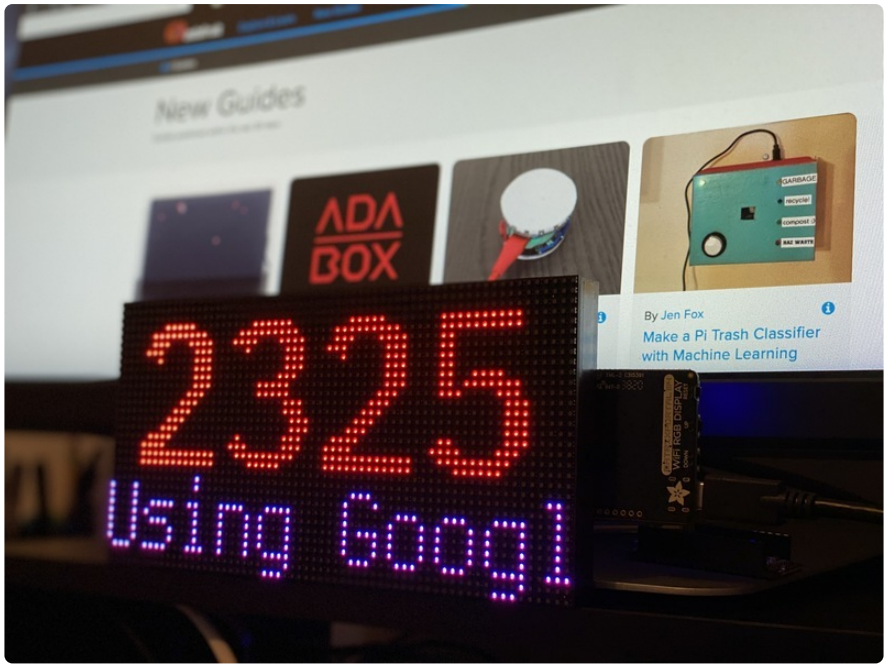
Overview



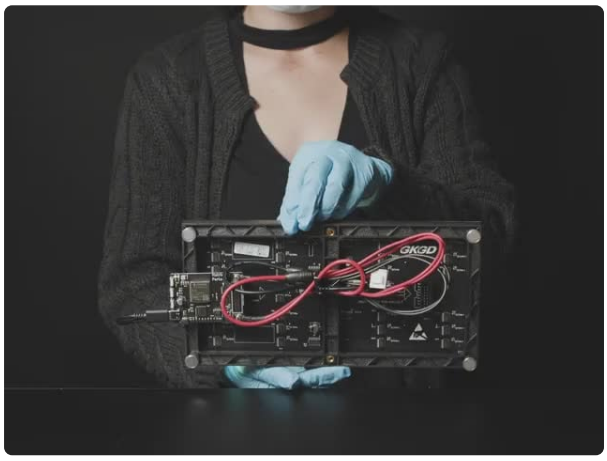
Is there a new guide on the Adafruit Learning System? Glance over at this attention-grabbing Matrix Portal New Guide scroller. **Every hour, this project fetches the latest guides from the Adafruit Learning System and displays them on the Matrix Portal.**

You'll use **CircuitPython and the Adafruit Matrix Portal** to fetch JSON data from the Adafruit Learning System's public API, parse the data, and display it scrolling across a RGB Matrix.

This internet-connected project can easily be **adapted to a REST API of your choice to scroll data of your choice:** negative COVID-19 tests, sports scores, positive news headlines, or stock prices.



Parts

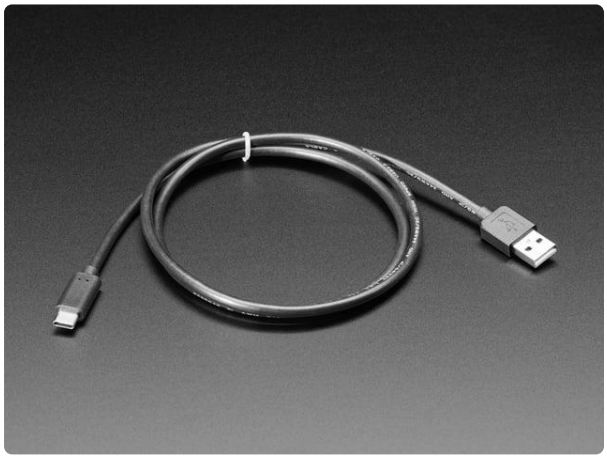


Adafruit Matrix Portal - CircuitPython Powered Internet Display

Folks love our wide selection of RGB matrices and accessories, for making custom colorful LED displays... and our RGB Matrix Shields...

https://www.adafruit.com/product/4745

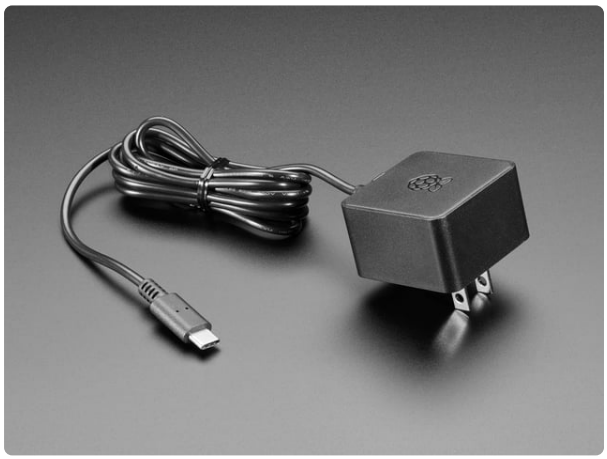
You can use a USB C power supply or a USB micro B with a micro B to C adapter (http://adafru.it/4299)



USB Type A to Type C Cable - approx 1 meter / 3 ft long

As technology changes and adapts, so does Adafruit. This USB Type A to Type C cable will help you with the transition to USB C, even if you're still...

https://www.adafruit.com/product/4474



Official Raspberry Pi Power Supply 5.1V 3A with USB C

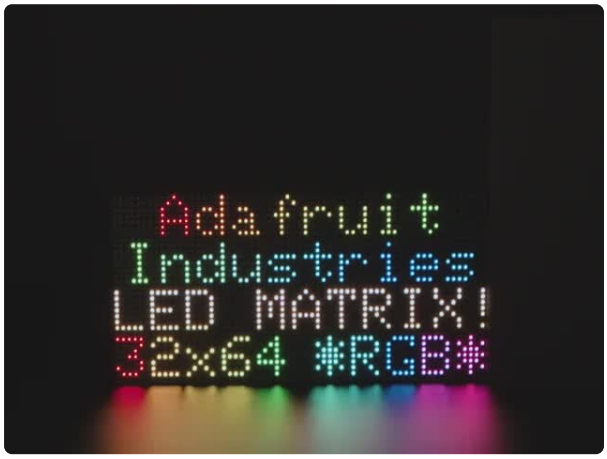
The official Raspberry Pi USB-C power supply is here! And of course, we have 'em in classic Adafruit black! Superfast with just the right amount of cable length to get your Pi 4...

https://www.adafruit.com/product/4298

[5V 2.5A Switching Power Supply with 20AWG MicroUSB Cable]
Our all-in-one 5V 2.5 Amp + MicroUSB cable power adapter is the perfect choice for powering single-board computers like Raspberry Pi, BeagleBone, or anything else that's...
https://www.adafruit.com/product/1995

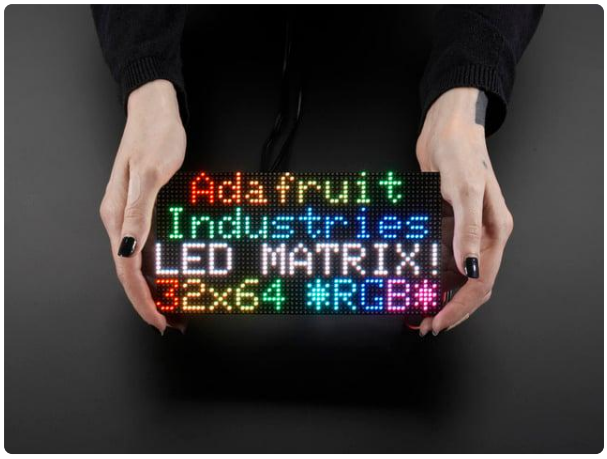
[Micro B USB to USB C Adapter]
As technology changes and adapts, so does Adafruit, and speaking of adapting, this adapter has a Micro B USB jack and a USB C...
https://www.adafruit.com/product/4299

If you'd like your LEDs diffused (and if your LED matrix is 4mm pitch or smaller), some acrylic may help:



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

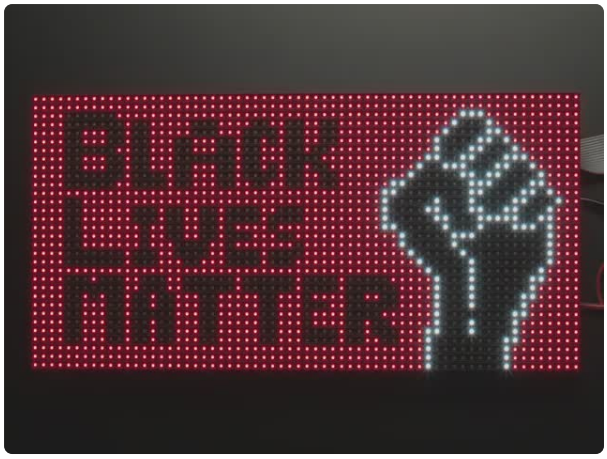
Adafruit carries a number of 64x32 RGB LED Matrices, varying between the space between LEDs (pitch) and whether rigid or flexible. Choose your favorite - larger pitch means the display is larger, width and height-wise but with the same number of pixels, and larger may be easier to read further away. Smaller for near your desk, for example.



64x32 RGB LED Matrix - 3mm pitch

Bring a little bit of Times Square into your home with this sweet 64 x 32 square RGB LED matrix panel. These panels are normally used to make video walls, here in New York we see them...

https://www.adafruit.com/product/2279



64x32 RGB LED Matrix - 4mm pitch

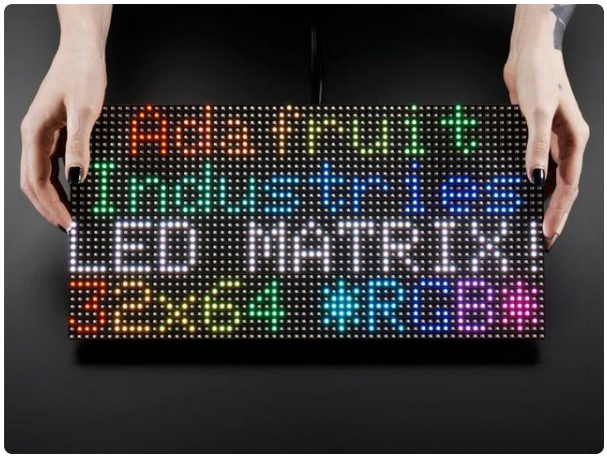
Bring a little bit of Times Square into your home with this sweet 64 x 32 square RGB LED matrix panel. These panels are normally used to make video walls, here in New York we see them...

https://www.adafruit.com/product/2278

64x32 RGB LED Matrix - 5mm pitch

Bring a little bit of Times Square into your home with this sweet 64x32 square RGB LED matrix panel. These panels are normally used to make video walls, here in New York we see them on...

https://www.adafruit.com/product/2277



64x32 RGB LED Matrix - 6mm pitch

Bring a little bit of Times Square into your home with this sweet 64x32 square RGB LED matrix panel. These panels are normally used to make video walls, here in New York we see them on...

https://www.adafruit.com/product/2276

64x32 Flexible RGB LED Matrix - 4mm Pitch

If you've played with multiplexed RGB matrices, you may have wondered "hey, could we possibly manufacture these on a thin enough PCB, so it's flexible?" and the...

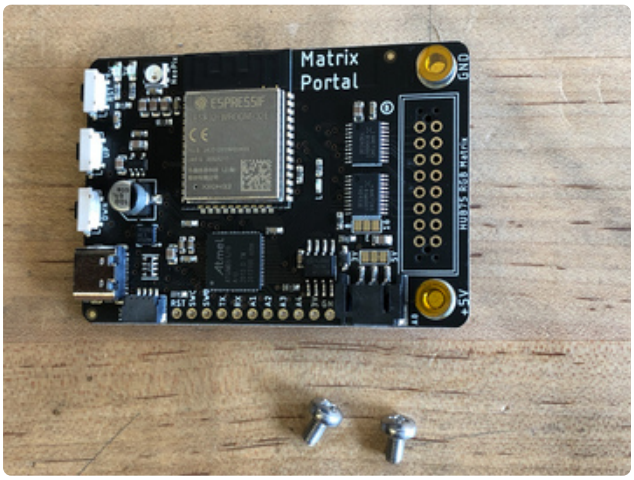
https://www.adafruit.com/product/3826

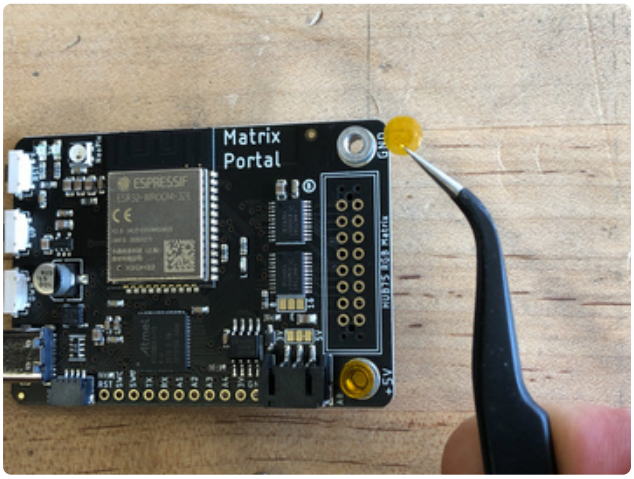
64x32 Flexible RGB LED Matrix - 5mm Pitch

If you've played with multiplexed RGB matrices, you may have wondered "hey, could we possibly manufacture these on a thin enough PCB so it's flexible?" and the answer...

https://www.adafruit.com/product/3803

Prep the MatrixPortal

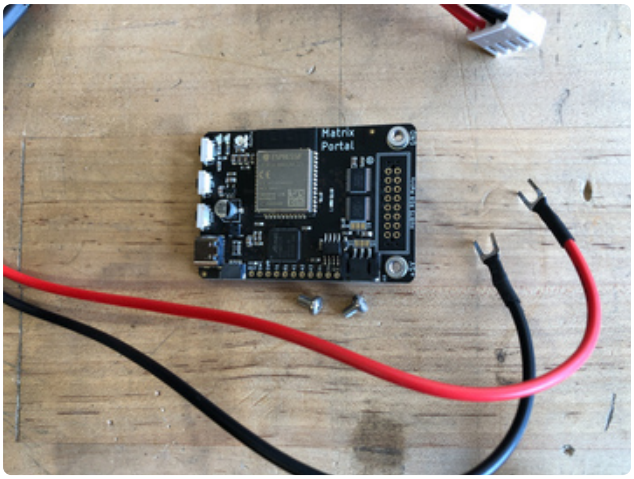


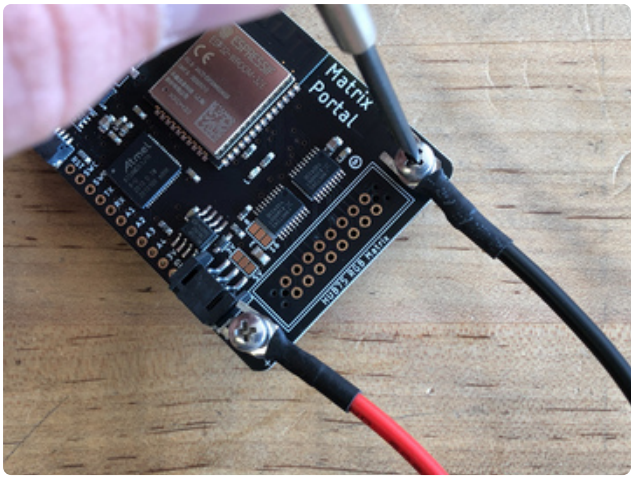


Power Prep

The MatrixPortal supplies power to the matrix display panel via two standoffs. These come with protective tape applied (part of our manufacturing process) which MUST BE REMOVED!

Use some tweezers or a fingernail to remove the two amber circles.

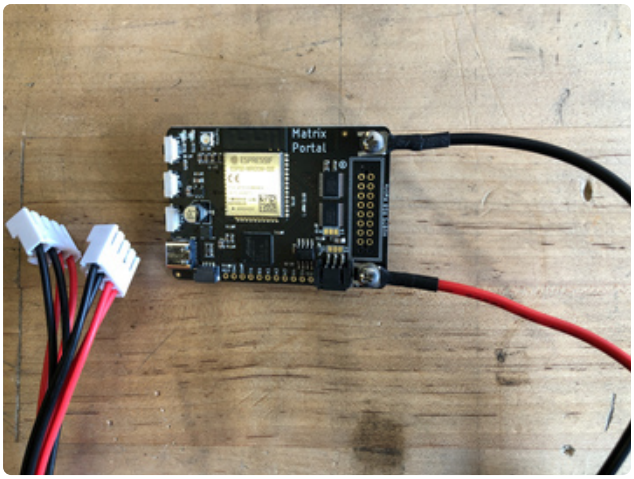


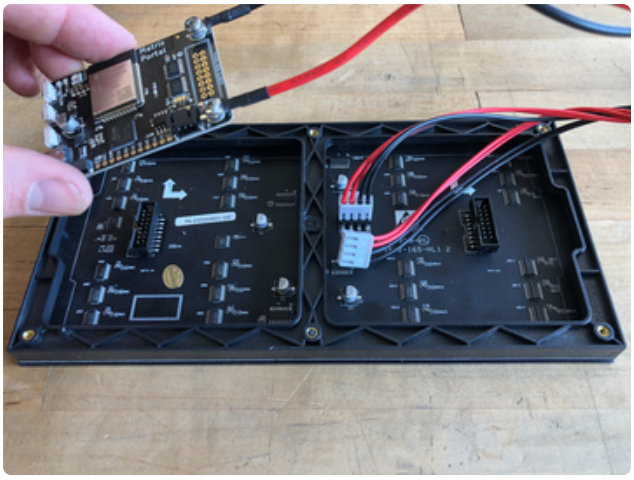


Power Terminals

Next, screw in the spade connectors to the corresponding standoff.

red wire goes to **+5V**
black wire goes to **GND**



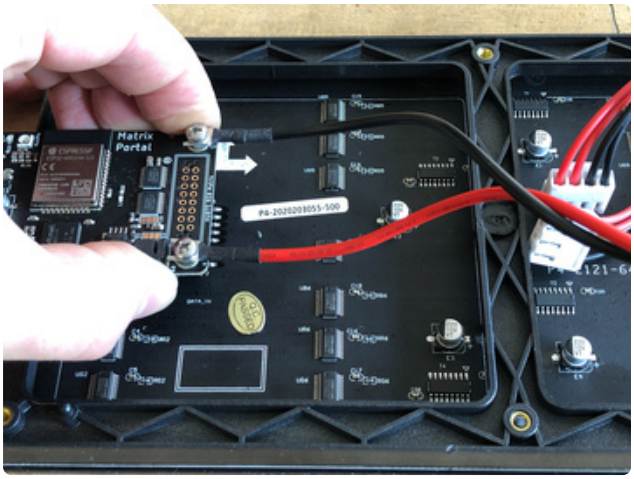


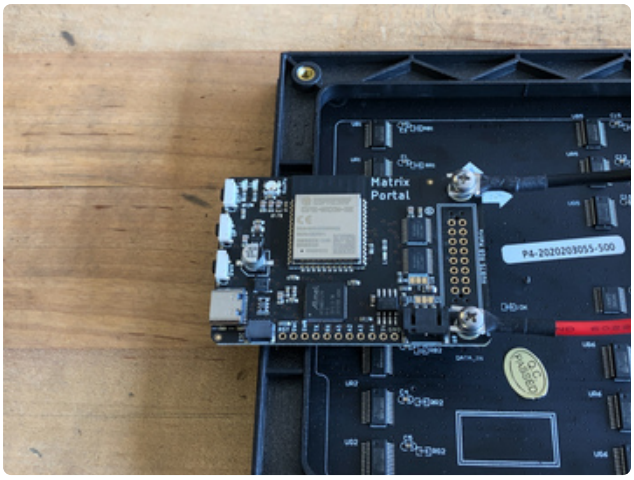
Panel Power

Plug either one of the four-conductor power plugs into the power connector pins on the panel. The plug can only go in one way, and that way is marked on the board's silkscreen.

Dual Matrix Setup

If you're planning to use a 64x64 matrix, follow these instructions on soldering the Address E Line jumper (https://adafru.it/OdJ).

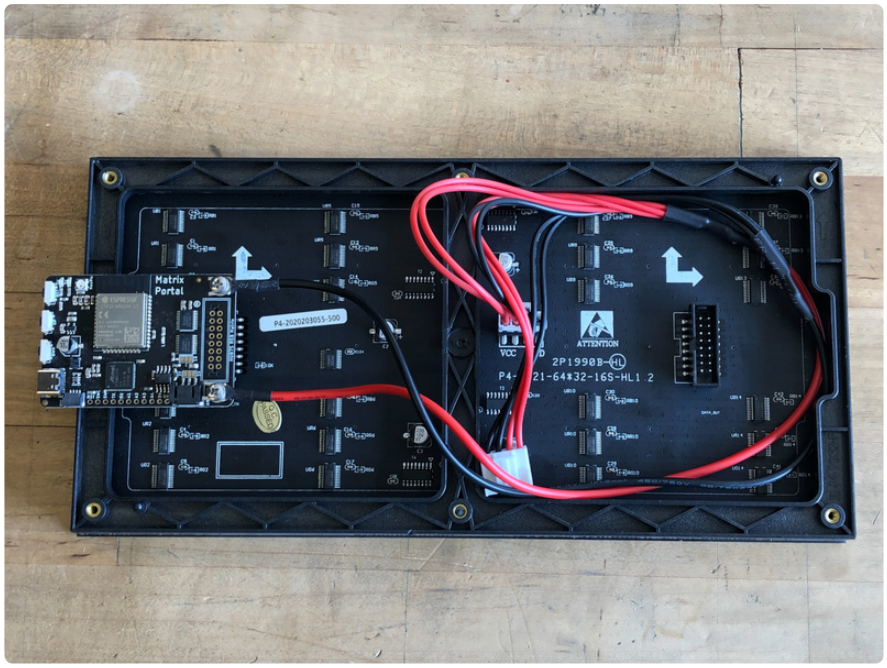




Board Connection

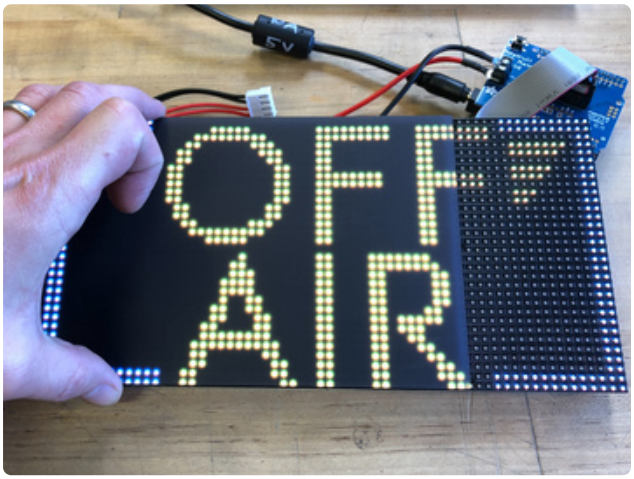
Now, plug the board into the left side shrouded 8x2 connector as shown. The orientation matters, so take a moment to confirm that the **white indicator arrow on the matrix panel is oriented pointing up and right** as seen here and the MatrixPortal overhangs the edge of the panel when connected. This allows you to use the edge buttons from the front side.

Check nothing is impeding the board from plugging in firmly. If there's a plastic nub on the matrix that's keeping the Portal from sitting flat, cut it off with diagonal cutters



For info on adding LED diffusion acrylic, see the page LED Matrix Diffuser.

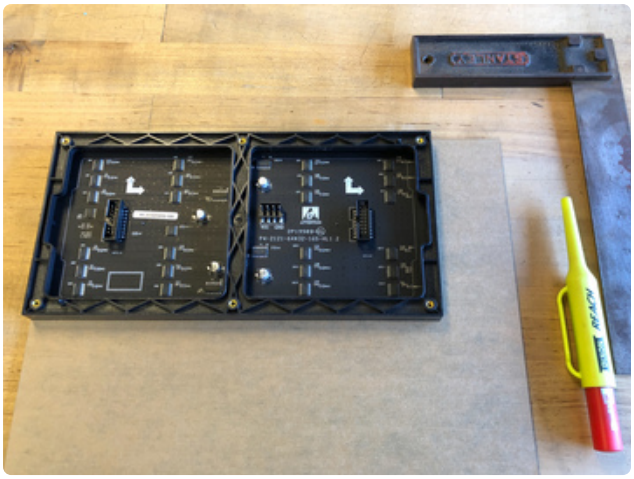
LED Matrix Diffuser

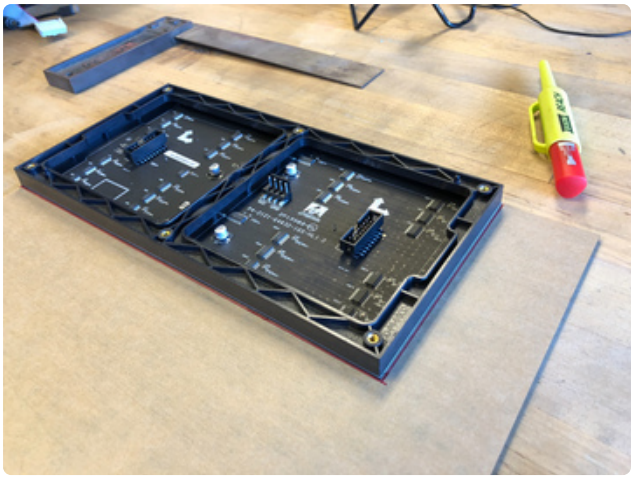


LED Diffusion Acrylic

You can add an LED diffusion acrylic faceplate (http://adafru.it/4594) to the your LED matrix display. (Pictured here with the ON AIR project (https://adafru.it/MPE))

This can help protect the LEDs as well as enhance the look of the sign both indoors and out by reducing glare and specular highlights of the plastic matrix grid.



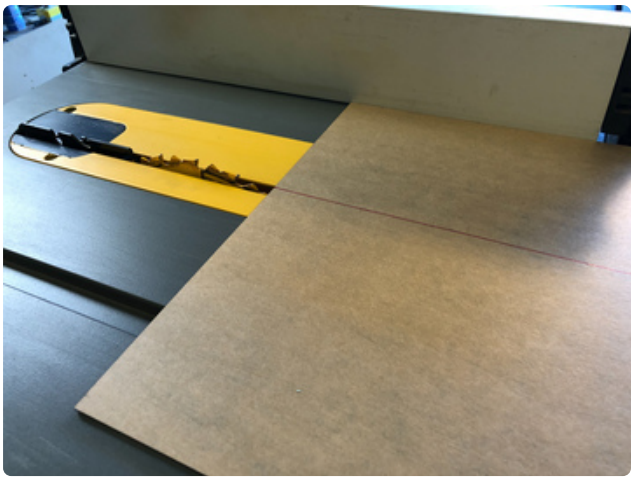


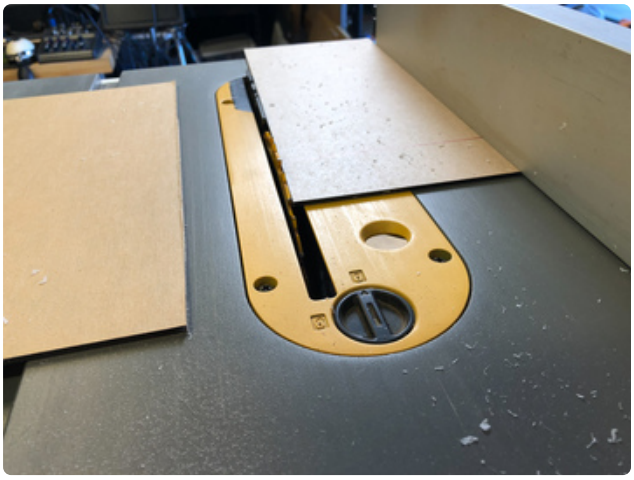
Measure and Cut the Plastic

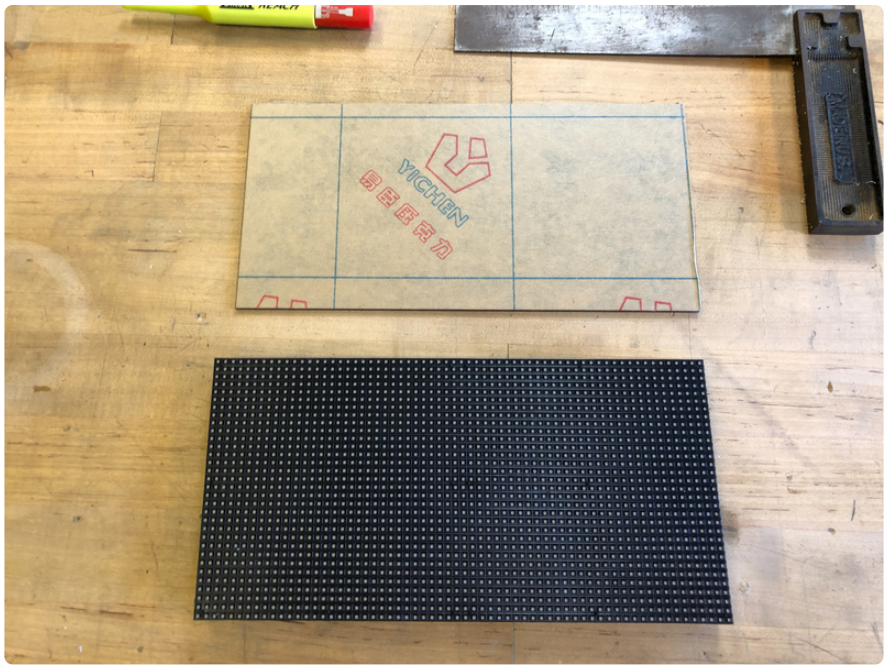
You can use the sign to measure and mark cut lines on the paper backing of the acrylic sheet.

Then, use a tablesaw or bandsaw with a fine toothed blade and a guide or sled to make the cuts.

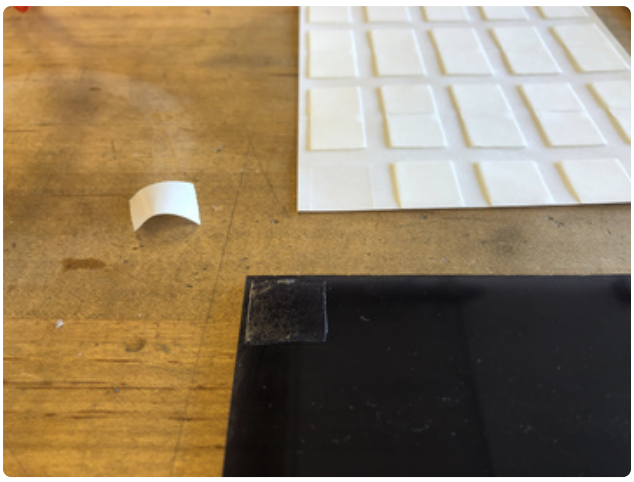
Note: it is possible to score and snap acrylic, but it can be very tricky to get an even snap without proper clamping.

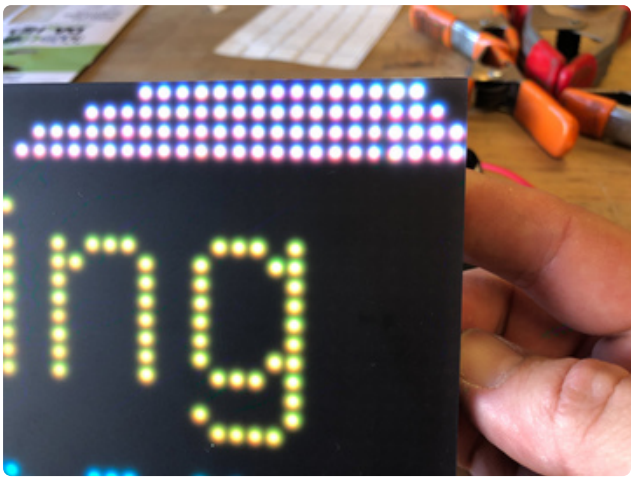






Peel away the paper backing from both sides and set the acrylic onto your matrix display with the matte finished side facing out.



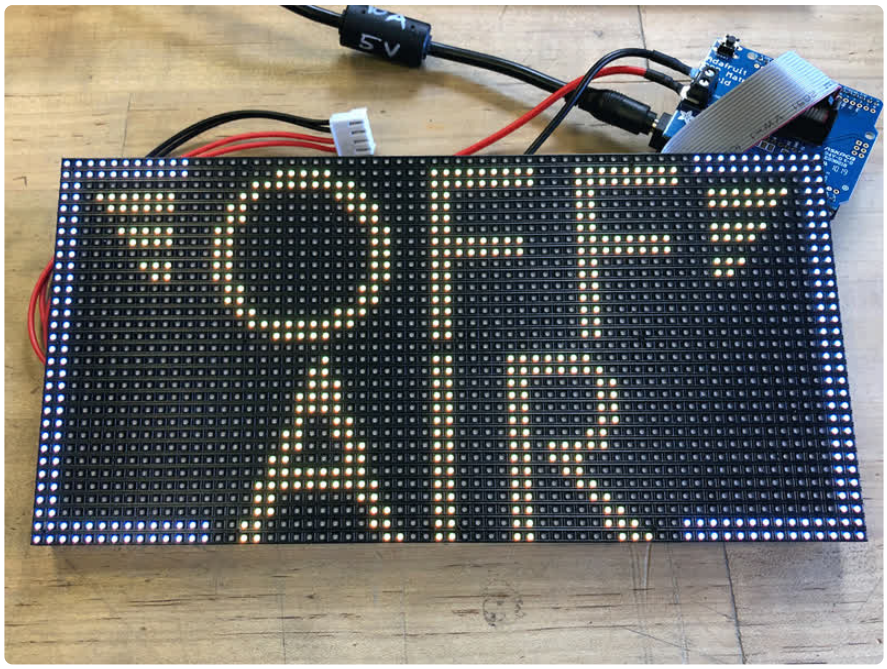


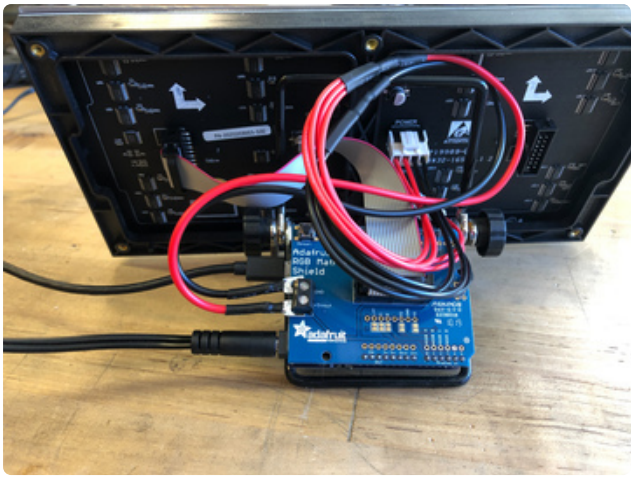
Uglu Dashes

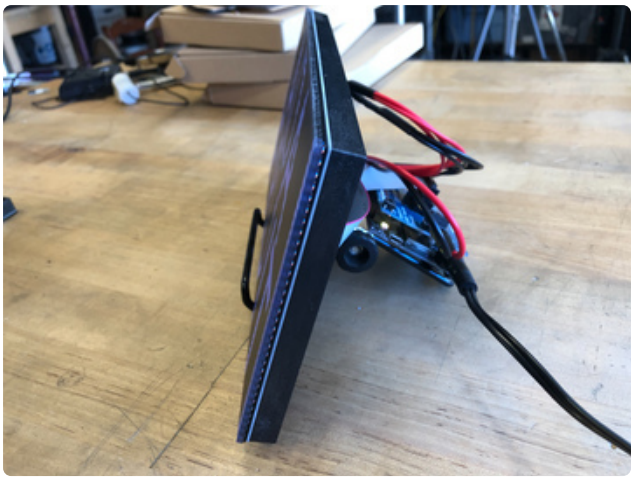
The best method we've found for adhering acrylic to the matrix display is to use Uglu Dashes clear adhesive rectangles from Pro Tapes (https://adafru.it/NcP). They are incredibly strong (although can be removed if necessary), easy to apply, and are invisible once attached.

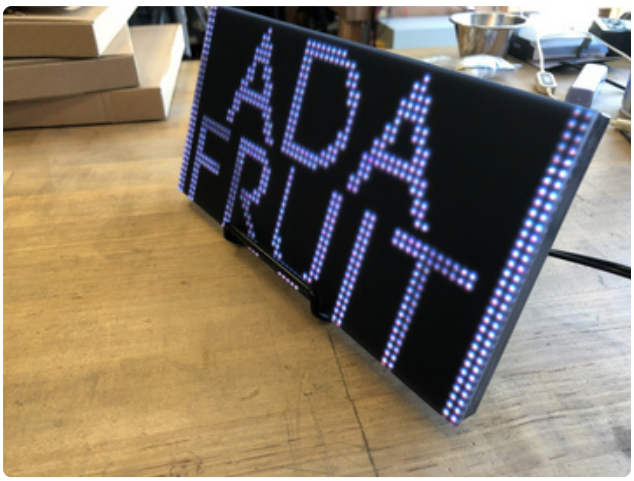
Use one at each corner and one each at the halfway point of the long edges, then press the acrylic and matrix panel together for about 20 seconds.

Here you can see the impact of using the diffusion acrylic. (Pictured here with the ON AIR sign project)


5V







Stand

A very simple and attractive way to display your matrix is with the adjustable bent-wire stand (http://adafru.it/1679).

Alternately, you can use a frame, 3D printed brackets (https://adafru.it/MZf), tape, glue, or even large binder clips to secure the acrylic to the sign and then mount it on on a wall, shelf, or display cabinet.

These mini-magnet feet (http://adafru.it/4631) can be used to stick the sign to a ferrous surface.

Install CircuitPython

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

Set up CircuitPython Quick Start!

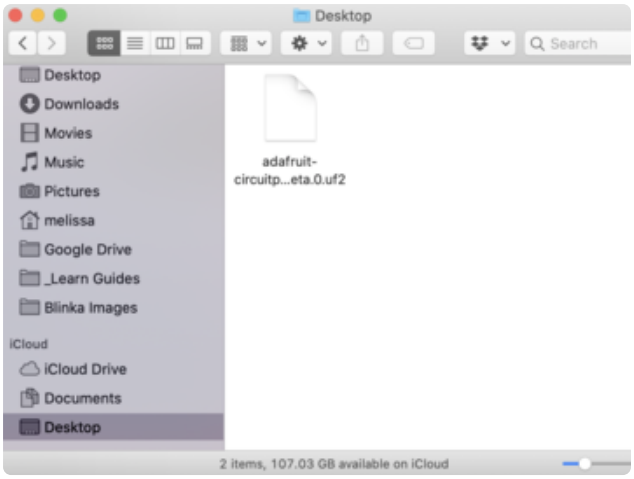
Follow this quick step-by-step for super-fast Python power :)

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

https://adafru.it/Nte

Further Information

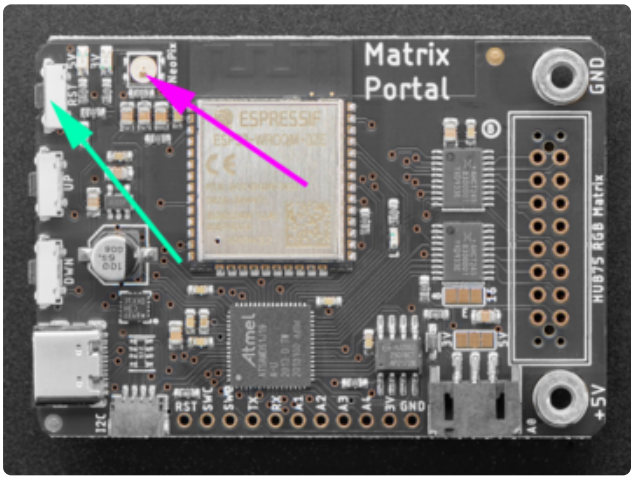
For more detailed info on installing CircuitPython, check out Installing CircuitPython (https://adafru.it/Amd).



Click the link above and download the latest UF2 file.

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

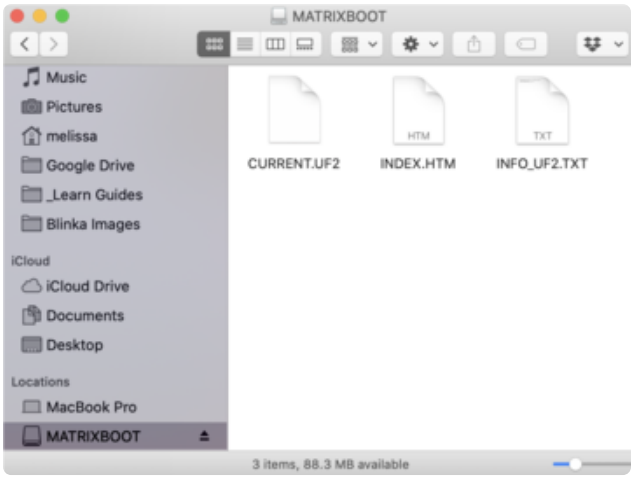
Plug your MatrixPortal M4 into your computer using a known-good USB cable.



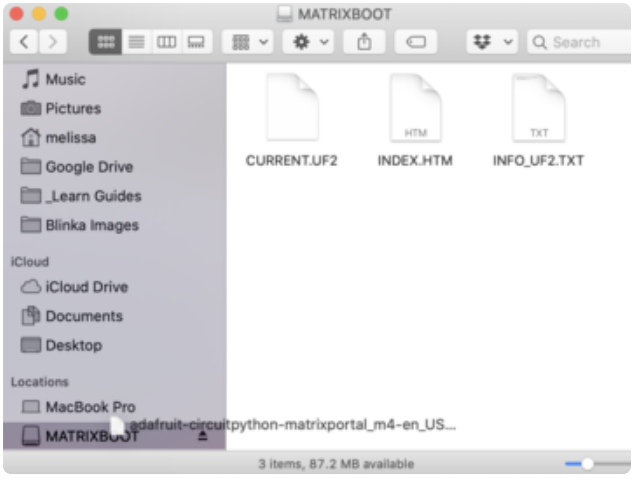
A lot of people end up using charge-only USB cables and it is very frustrating! So make sure you have a USB cable you know is good for data sync.

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

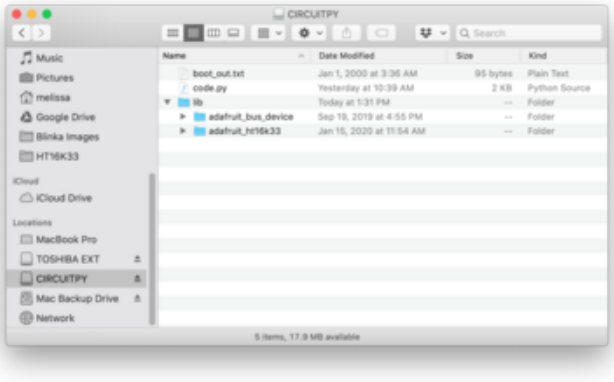
If double-clicking doesn't work the first time, try again. Sometimes it can take a few tries to get the rhythm right!



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



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



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

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

Create Your settings.toml File

CircuitPython works with WiFi-capable boards to enable you to make projects that have network connectivity. This means working with various passwords and API keys. As of CircuitPython 8 (https://adafru.it/Em8), there is support for a **settings.toml** file. This is a file that is stored on your **CIRCUITPY** drive, that contains all of your secret network information, such as your SSID, SSID password and any API keys for IoT

services. It is designed to separate your sensitive information from your **code.py** file so you are able to share your code without sharing your credentials.

CircuitPython previously used a **secrets.py** file for this purpose. The **settings.toml** file is quite similar.

> Your settings.toml file should be stored in the main directory of your CIRCUITPY drive. It should not be in a folder.

CircuitPython **settings.toml** File

This section will provide a couple of examples of what your **settings.toml** file should look like, specifically for CircuitPython WiFi projects in general.

The most minimal **settings.toml** file must contain your WiFi SSID and password, as that is the minimum required to connect to WiFi. Copy this example, paste it into your **settings.toml**, and update:

- `your_wifi_ssid`
- `your_wifi_password`

```
CIRCUITPY_WIFI_SSID = "your_wifi_ssid"
CIRCUITPY_WIFI_PASSWORD = "your_wifi_password"
```

Many CircuitPython network-connected projects on the Adafruit Learn System involve using Adafruit IO. For these projects, you must also include your Adafruit IO username and key. Copy the following example, paste it into your settings.toml file, and update:

- `your_wifi_ssid`
- `your_wifi_password`
- `your_aio_username`
- `your_aio_key`

```
CIRCUITPY_WIFI_SSID = "your_wifi_ssid"
CIRCUITPY_WIFI_PASSWORD = "your_wifi_password"
ADAFRUIT_AIO_USERNAME = "your_aio_username"
ADAFRUIT_AIO_KEY = "your_aio_key"
```

Some projects use different variable names for the entries in the **settings.toml** file. For example, a project might use `ADAFRUIT_AIO_ID` in the place of

`ADAFRUIT_AIO_USERNAME`. **If you run into connectivity issues, one of the first things to check is that the names in the settings.toml file match the names in the code.**

> Not every project uses the same variable name for each entry in the settings.toml file! Always verify it matches the code.

settings.toml File Tips

Here is an example **settings.toml** file.

```
# Comments are supported
CIRCUITPY_WIFI_SSID = "guest wifi"
CIRCUITPY_WIFI_PASSWORD = "guessable"
CIRCUITPY_WEB_API_PORT = 80
CIRCUITPY_WEB_API_PASSWORD = "passw0rd"
test_variable = "this is a test"
thumbs_up = "\U0001f44d"
```

In a **settings.toml** file, it's important to keep these factors in mind:

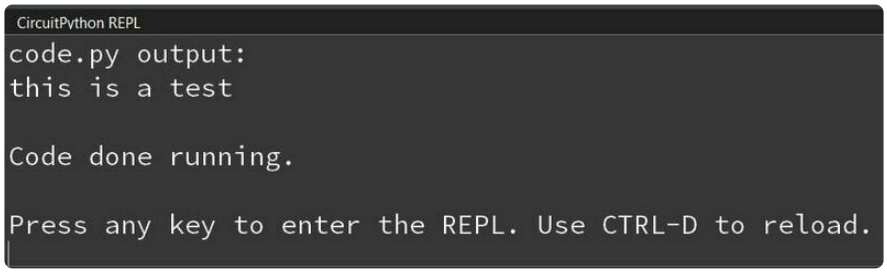
- Strings are wrapped in double quotes; ex: `"your-string-here"`
- Integers are **not** quoted and may be written in decimal with optional sign (`+1`, `-1`, `1000`) or hexadecimal (`0xabcd`).
    - Floats, octal (`0o567`) and binary (`0b11011`) are not supported.
- Use `\u` escapes for weird characters, `\x` and `\ooo` escapes are not available in **.toml** files
    - Example: `\U0001f44d` for (thumbs up emoji) and `\u20ac` for € (EUR sign)
- Unicode emoji, and non-ASCII characters, stand for themselves as long as you're careful to save in "UTF-8 without BOM" format

When your **settings.toml** file is ready, you can save it in your text editor with the **.toml** extension.

Accessing Your **settings.toml** Information in **code.py**

In your **code.py** file, you'll need to `import` the `os` library to access the **settings.toml** file. Your settings are accessed with the `os.getenv()` function. You'll pass your settings entry to the function to import it into the **code.py** file.

```
import os

print(os.getenv("test_variable"))
```



In the upcoming CircuitPython WiFi examples, you'll see how the **settings.toml** file is used for connecting to your SSID and accessing your API keys.

Internet Connect!

Connect to WiFi

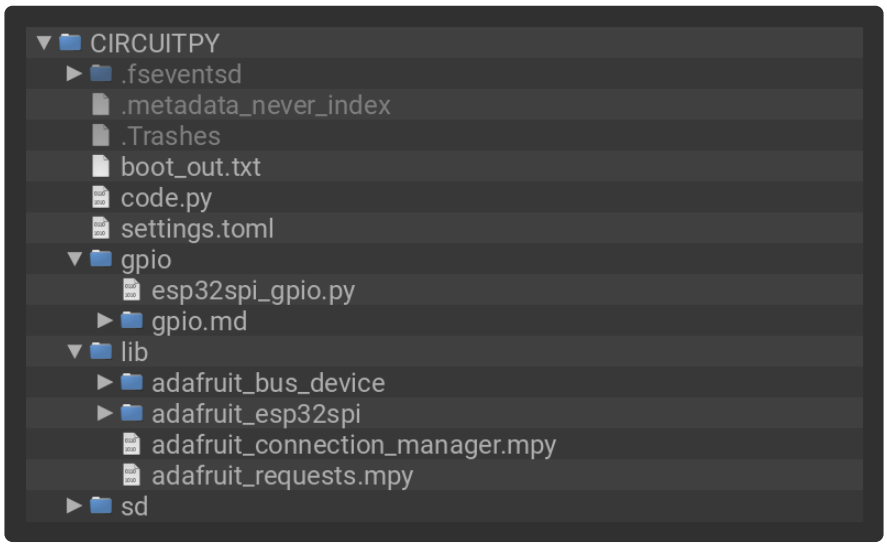
OK, now that you have your **settings.toml** file set up - you can connect to the Internet.

To do this, you need to first install a few libraries, into the lib folder on your **CIRCUITPY** drive. Then you need to update **code.py** with the example script.

Thankfully, we can do this in one go. In the example below, click the **Download Project Bundle** button below to download the necessary libraries and the **code.py** file in a zip file. Extract the contents of the zip file, open the directory **examples/** and then

click on the directory that matches the version of CircuitPython you're using and copy the contents of that directory to your **CIRCUITPY** drive.

Your **CIRCUITPY** drive should now look similar to the following image:



If you are using CircuitPython 9.0.x on a board with frozen libraries, such the Matrix Portal M4, use this version of the "Internet Connect" program. If you are using CircuitPython 9.1.0 or later, use the second version below.

```
# SPDX-FileCopyrightText: 2019 ladyada for Adafruit Industries
# SPDX-License-Identifier: MIT

from os import getenv
import board
import busio
from digitalio import DigitalInOut
import adafruit_connection_manager
import adafruit_requests
from adafruit_esp32spi import adafruit_esp32spi

# Get wifi details and more from a settings.toml file
# tokens used by this Demo: CIRCUITPY_WIFI_SSID, CIRCUITPY_WIFI_PASSWORD
secrets = {
    "ssid": getenv("CIRCUITPY_WIFI_SSID"),
    "password": getenv("CIRCUITPY_WIFI_PASSWORD"),
}
if secrets == {"ssid": None, "password": None}:
    try:
        # Fallback on secrets.py until depreciation is over and option is removed
        from secrets import secrets
    except ImportError:
        print("WiFi secrets are kept in settings.toml, please add them there!")
        raise

print("ESP32 SPI webclient test")

TEXT_URL = "http://wifitest.adafruit.com/testwifi/index.html"
JSON_URL = "http://api.coindesk.com/v1/bpi/currentprice/USD.json"
```

```
# If you are using a board with pre-defined ESP32 Pins:
esp32_cs = DigitalInOut(board.ESP_CS)
esp32_ready = DigitalInOut(board.ESP_BUSY)
esp32_reset = DigitalInOut(board.ESP_RESET)

# If you have an AirLift Shield:
# esp32_cs = DigitalInOut(board.D10)
# esp32_ready = DigitalInOut(board.D7)
# esp32_reset = DigitalInOut(board.D5)

# If you have an AirLift Featherwing or ItsyBitsy Airlift:
# esp32_cs = DigitalInOut(board.D13)
# esp32_ready = DigitalInOut(board.D11)
# esp32_reset = DigitalInOut(board.D12)

# If you have an externally connected ESP32:
# NOTE: You may need to change the pins to reflect your wiring
# esp32_cs = DigitalInOut(board.D9)
# esp32_ready = DigitalInOut(board.D10)
# esp32_reset = DigitalInOut(board.D5)

# Secondary (SCK1) SPI used to connect to WiFi board on Arduino Nano Connect RP2040
if "SCK1" in dir(board):
    spi = busio.SPI(board.SCK1, board.MOSI1, board.MISO1)
else:
    spi = busio.SPI(board.SCK, board.MOSI, board.MISO)
esp = adafruit_esp32spi.ESP_SPIcontrol(spi, esp32_cs, esp32_ready, esp32_reset)

pool = adafruit_connection_manager.get_radio_socketpool(esp)
ssl_context = adafruit_connection_manager.get_radio_ssl_context(esp)
requests = adafruit_requests.Session(pool, ssl_context)

if esp.status == adafruit_esp32spi.WL_IDLE_STATUS:
    print("ESP32 found and in idle mode")
print("Firmware vers.", esp.firmware_version.decode("utf-8"))
print("MAC addr:", ":".join("%02X" % byte for byte in esp.MAC_address))

for ap in esp.scan_networks():
    print("\t%-23s RSSI: %d" % (str(ap["ssid"], "utf-8"), ap["rssi"]))

print("Connecting to AP...")
while not esp.is_connected:
    try:
        esp.connect_AP(secrets["ssid"], secrets["password"])
    except OSError as e:
        print("could not connect to AP, retrying: ", e)
        continue
print("Connected to", str(esp.ssid, "utf-8"), "\tRSSI:", esp.rssi)
print("My IP address is", esp.pretty_ip(esp.ip_address))
print(
    "IP lookup adafruit.com: %s" %
esp.pretty_ip(esp.get_host_by_name("adafruit.com"))
)
print("Ping google.com: %d ms" % esp.ping("google.com"))

# esp._debug = True
print("Fetching text from", TEXT_URL)
r = requests.get(TEXT_URL)
print("-" * 40)
print(r.text)
print("-" * 40)
r.close()

print()
print("Fetching json from", JSON_URL)
r = requests.get(JSON_URL)
print("-" * 40)
print(r.json())
```

```
print("-" * 40)
r.close()

print("Done!")
```

If you are using CircuitPython 9.1.0, or using the latest version of the ESP32SPI library, using the version below. If you are using CircuitPython 9.0.x on a board with frozen libraries, such as the Matrix Portal M4, use the first version above.

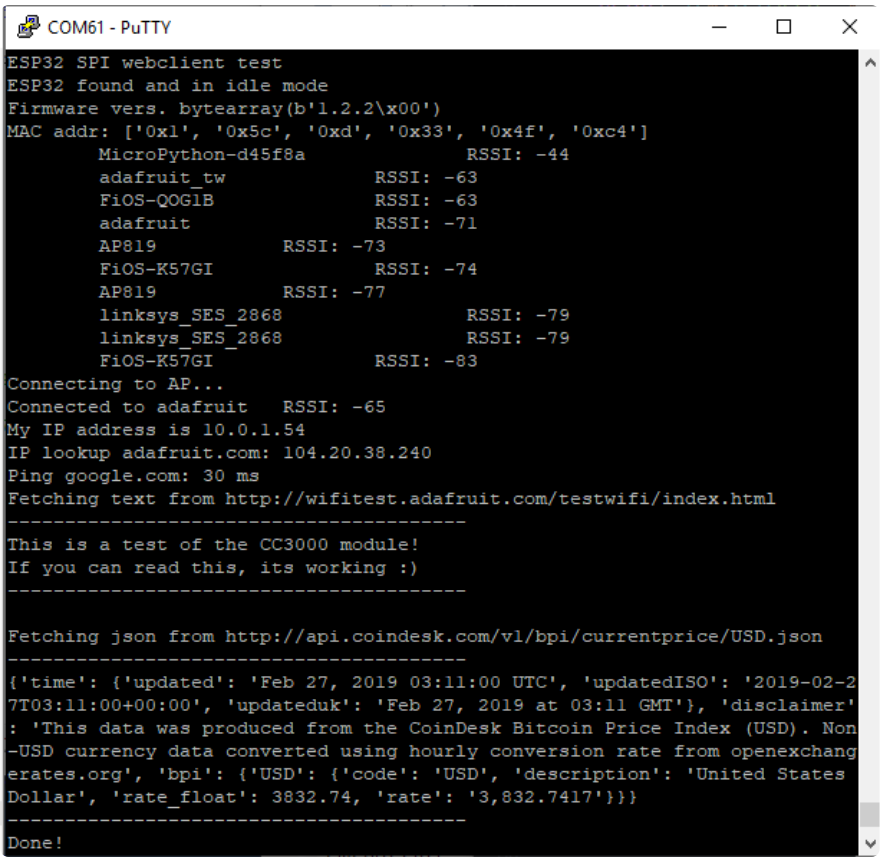
```
# SPDX-FileCopyrightText: 2019 ladyada for Adafruit Industries
# SPDX-License-Identifier: MIT

from os import getenv
import board
import busio
from digitalio import DigitalInOut
import adafruit_connection_manager
import adafruit_requests
from adafruit_esp32spi import adafruit_esp32spi

# Get wifi details and more from a settings.toml file
# tokens used by this Demo: CIRCUITPY_WIFI_SSID, CIRCUITPY_WIFI_PASSWORD
secrets = {
    "ssid": getenv("CIRCUITPY_WIFI_SSID"),
    "password": getenv("CIRCUITPY_WIFI_PASSWORD"),
}
if secrets == {"ssid": None, "password": None}:
    try:
        # Fallback on secrets.py until depreciation is over and option is removed
        from secrets import secrets
    except ImportError:
        print("WiFi secrets are kept in settings.toml, please add them there!")
        raise

print("ESP32 SPI webclient test")

TEXT_URL = "http://wifitest.adafruit.com/testwifi/index.html"
JSON_URL = "http://api.coindesk.com/v1/bpi/currentprice/USD.json"


# If you are using a board with pre-defined ESP32 Pins:
esp32_cs = DigitalInOut(board.ESP_CS)
esp32_ready = DigitalInOut(board.ESP_BUSY)
esp32_reset = DigitalInOut(board.ESP_RESET)

# If you have an AirLift Shield:
# esp32_cs = DigitalInOut(board.D10)
# esp32_ready = DigitalInOut(board.D7)
# esp32_reset = DigitalInOut(board.D5)

# If you have an AirLift Featherwing or ItsyBitsy Airlift:
# esp32_cs = DigitalInOut(board.D13)
# esp32_ready = DigitalInOut(board.D11)
# esp32_reset = DigitalInOut(board.D12)

# If you have an externally connected ESP32:
# NOTE: You may need to change the pins to reflect your wiring
# esp32_cs = DigitalInOut(board.D9)
# esp32_ready = DigitalInOut(board.D10)
# esp32_reset = DigitalInOut(board.D5)
```

```
# Secondary (SCK1) SPI used to connect to WiFi board on Arduino Nano Connect RP2040
if "SCK1" in dir(board):
    spi = busio.SPI(board.SCK1, board.MOSI1, board.MISO1)
else:
    spi = busio.SPI(board.SCK, board.MOSI, board.MISO)
esp = adafruit_esp32spi.ESP_SPIcontrol(spi, esp32_cs, esp32_ready, esp32_reset)

pool = adafruit_connection_manager.get_radio_socketpool(esp)
ssl_context = adafruit_connection_manager.get_radio_ssl_context(esp)
requests = adafruit_requests.Session(pool, ssl_context)

if esp.status == adafruit_esp32spi.WL_IDLE_STATUS:
    print("ESP32 found and in idle mode")
print("Firmware vers.", esp.firmware_version)
print("MAC addr:", ":".join("%02X" % byte for byte in esp.MAC_address))

for ap in esp.scan_networks():
    print("\t%-23s RSSI: %d" % (ap.ssid, ap.rssi))

print("Connecting to AP...")
while not esp.is_connected:
    try:
        esp.connect_AP(secrets["ssid"], secrets["password"])
    except OSError as e:
        print("could not connect to AP, retrying: ", e)
        continue
print("Connected to", esp.ap_info.ssid, "\tRSSI:", esp.ap_info.rssi)
print("My IP address is", esp.ipv4_address)
print(
    "IP lookup adafruit.com: %s" %
esp.pretty_ip(esp.get_host_by_name("adafruit.com"))
)
print("Ping google.com: %d ms" % esp.ping("google.com"))

# esp._debug = True
print("Fetching text from", TEXT_URL)
r = requests.get(TEXT_URL)
print("-" * 40)
print(r.text)
print("-" * 40)
r.close()

print()
print("Fetching json from", JSON_URL)
r = requests.get(JSON_URL)
print("-" * 40)
print(r.json())
print("-" * 40)
r.close()

print("Done!")
```

And save it to your board, with the name **code.py**.

Don't forget you'll also need to create the **settings.toml** file as seen above, with your WiFi ssid and password.

In a serial console, you should see something like the following. For more information about connecting with a serial console, view the guide Connecting to the Serial Console (https://adafru.it/Bec).



In order, the example code...

Initializes the ESP32 over SPI using the SPI port and 3 control pins:

```
esp32_cs = DigitalInOut(board.ESP_CS)
esp32_ready = DigitalInOut(board.ESP_BUSY)
esp32_reset = DigitalInOut(board.ESP_RESET)

#...

else:
    spi = busio.SPI(board.SCK, board.MOSI, board.MISO)
esp = adafruit_esp32spi.ESP_SPIcontrol(spi, esp32_cs, esp32_ready, esp32_reset)
```

Gets the socket pool and the SSL context, and then tells the `adafruit_requests` library about them.

```
pool = adafruit_connection_manager.get_radio_socketpool(esp)
ssl_context = adafruit_connection_manager.get_radio_ssl_context(esp)
requests = adafruit_requests.Session(pool, ssl_context)
```

Verifies an ESP32 is found, checks the firmware and MAC address

```
if esp.status == adafruit_esp32spi.WL_IDLE_STATUS:
    print("ESP32 found and in idle mode")
print("Firmware vers.", esp.firmware_version)
print("MAC addr:", [hex(i) for i in esp.MAC_address])
```

Performs a scan of all access points it can see and prints out the name and signal strength:

```
for ap in esp.scan_networks():
    print("\t%s\t\tRSSI: %d" % (str(ap['ssid'], 'utf-8'), ap['rssi']))
```

Connects to the AP we've defined here, then prints out the local IP address, attempts to do a domain name lookup and ping google.com to check network connectivity (note sometimes the ping fails or takes a while, this isn't a big deal)

```
print("Connecting to AP...")
while not esp.is_connected:
    try:
        esp.connect_AP(secrets["ssid"], secrets["password"])
    except RuntimeError as e:
        print("could not connect to AP, retrying: ", e)
        continue
print("Connected to", str(esp.ssid, "utf-8"), "\tRSSI:", esp.rssi)
print("My IP address is", esp.pretty_ip(esp.ip_address))
print(
    "IP lookup adafruit.com: %s" %
esp.pretty_ip(esp.get_host_by_name("adafruit.com"))
```

OK now we're getting to the really interesting part. With a SAMD51 or other large-RAM (well, over 32 KB) device, we can do a lot of neat tricks. Like for example we can implement an interface a lot like requests (https://adafru.it/E9o) - which makes getting data really really easy

To read in all the text from a web URL call `requests.get` - you can pass in `https` URLs for SSL connectivity

```
TEXT_URL = "http://wifitest.adafruit.com/testwifi/index.html"
print("Fetching text from", TEXT_URL)
r = requests.get(TEXT_URL)
print('-'*40)
print(r.text)
print('-'*40)
r.close()
```

Or, if the data is in structured JSON, you can get the json pre-parsed into a Python dictionary that can be easily queried or traversed. (Again, only for nRF52840, M4 and other high-RAM boards)

```
JSON_URL = "http://api.coindesk.com/v1/bpi/currentprice/USD.json"
print("Fetching json from", JSON_URL)
r = requests.get(JSON_URL)
print('-'*40)
print(r.json())
print('-'*40)
r.close()
```

Requests

We've written a requests-like (https://adafru.it/Kpa) library for web interfacing named Adafruit_CircuitPython_Requests (https://adafru.it/FpW). This library allows you to send HTTP/1.1 requests without "crafting" them and provides helpful methods for parsing the response from the server.

To use with CircuitPython, you need to first install a few libraries, into the lib folder on your **CIRCUITPY** drive. Then you need to update **code.py** with the example script.

Thankfully, we can do this in one go. In the example below, click the **Download Project Bundle** button below to download the necessary libraries and the **code.py** file in a zip file. Extract the contents of the zip file, open the directory **examples/** and then click on the directory that matches the version of CircuitPython you're using and copy the contents of that directory to your **CIRCUITPY** drive.

Your **CIRCUITPY** drive should now look similar to the following image:

CIRCUITPY

```
Temporarily unable to load content:
```

The code first sets up the ESP32SPI interface. Then, it initializes a `request` object using an ESP32 `socket` and the `esp` object.

```
import board
import busio
from digitalio import DigitalInOut
import adafruit_esp32spi.adafruit_esp32spi_socket as socket
from adafruit_esp32spi import adafruit_esp32spi
import adafruit_connection_manager
import adafruit_requests as requests

# If you are using a board with pre-defined ESP32 Pins:
esp32_cs = DigitalInOut(board.ESP_CS)
esp32_ready = DigitalInOut(board.ESP_BUSY)
esp32_reset = DigitalInOut(board.ESP_RESET)

# If you have an externally connected ESP32:
# esp32_cs = DigitalInOut(board.D9)
# esp32_ready = DigitalInOut(board.D10)
# esp32_reset = DigitalInOut(board.D5)

spi = busio.SPI(board.SCK, board.MOSI, board.MISO)
esp = adafruit_esp32spi.ESP_SPIcontrol(spi, esp32_cs, esp32_ready, esp32_reset)

print("Connecting to AP...")
while not esp.is_connected:
    try:
        esp.connect_AP(b'MY_SSID_NAME', b'MY_SSID_PASSWORD')
    except RuntimeError as e:
        print("could not connect to AP, retrying: ",e)
        continue
print("Connected to", str(esp.ssid, 'utf-8'), "\tRSSI:", esp.rssi)
```

```
pool = adafruit_connection_manager.get_radio_socketpool(esp)
ssl_context = adafruit_connection_manager.get_radio_ssl_context(esp)
requests = adafruit_requests.Session(pool, ssl_context)
```

HTTP GET with Requests

The code makes a HTTP GET request to Adafruit's WiFi testing website - http://wifitest.adafruit.com/testwifi/index.html (https://adafru.it/Fp-).

To do this, we'll pass the URL into `requests.get()`. We're also going to save the response from the server into a variable named `response`.

Having requested data from the server, we'd now like to see what the server responded with. Since we already saved the server's `response`, we can read it back. Luckily for us, **requests automatically decodes the server's response into human-readable text,** you can read it back by calling `response.text`.

Lastly, we'll perform a bit of cleanup by calling `response.close()`. This closes, deletes, and collect's the response's data.

```
print("Fetching text from %s"%TEXT_URL)
response = requests.get(TEXT_URL)
print('-'*40)

print("Text Response: ", response.text)
print('-'*40)
response.close()
```

While some servers respond with text, some respond with json-formatted data consisting of attribute–value pairs.

CircuitPython_Requests can convert a JSON-formatted response from a server into a CPython `dict.` object.

We can also fetch and parse **json** data. We'll send a HTTP get to a url we know returns a json-formatted response (instead of text data).

Then, the code calls `response.json()` to convert the response to a CPython `dict`.

```
print("Fetching JSON data from %s"%JSON_GET_URL)
response = requests.get(JSON_GET_URL)
print('-'*40)

print("JSON Response: ", response.json())
print('-'*40)
response.close()
```

HTTP POST with Requests

Requests can also **POST data to a server by calling the `requests.post` method, passing it a `data` value.**

```
data = '31F'
print("POSTing data to {0}: {1}".format(JSON_POST_URL, data))
response = requests.post(JSON_POST_URL, data=data)
print('-'*40)

json_resp = response.json()
# Parse out the 'data' key from json_resp dict.
print("Data received from server:", json_resp['data'])
print('-'*40)
response.close()
```

You can also post json-formatted data to a server by passing `json_data` into the `requests.post` method.

```
    json_data = {"Date" : "July 25, 2019"}
print("POSTing data to {0}: {1}".format(JSON_POST_URL, json_data))
response = requests.post(JSON_POST_URL, json=json_data)
print('-'*40)

json_resp = response.json()
# Parse out the 'json' key from json_resp dict.
print("JSON Data received from server:", json_resp['json'])
print('-'*40)
response.close()
```

Advanced Requests Usage

Want to send custom HTTP headers, parse the response as raw bytes, or handle a response's http status code in your CircuitPython code?

We've written an example to show advanced usage of the requests module below.

To use with CircuitPython, you need to first install a few libraries, into the lib folder on your **CIRCUITPY** drive. Then you need to update **code.py** with the example script.

Thankfully, we can do this in one go. In the example below, click the **Download Project Bundle** button below to download the necessary libraries and the **code.py** file in a zip file. Extract the contents of the zip file, open the directory **examples/** and then click on the directory that matches the version of CircuitPython you're using and copy the contents of that directory to your **CIRCUITPY** drive.

Your **CIRCUITPY** drive should now look similar to the following image:

CIRCUITPY

```
Temporarily unable to load content:
```

WiFi Manager

That simpletest example works but it's a little finicky - you need to constantly check WiFi status and have many loops to manage connections and disconnections. For more advanced uses, we recommend using the WiFiManager object. It will wrap the connection/status/requests loop for you - reconnecting if WiFi drops, resetting the ESP32 if it gets into a bad state, etc.

Here's a more advanced example that shows the WiFi manager and also how to POST data with some extra headers:

To use with CircuitPython, you need to first install a few libraries, into the lib folder on your **CIRCUITPY** drive. Then you need to update **code.py** with the example script.

Thankfully, we can do this in one go. In the example below, click the **Download Project Bundle** button below to download the necessary libraries and the **code.py** file in a zip file. Extract the contents of the zip file, open the directory **examples/** and then click on the directory that matches the version of CircuitPython you're using and copy the contents of that directory to your **CIRCUITPY** drive.

Your **CIRCUITPY** drive should now look similar to the following image:


```
# SPDX-FileCopyrightText: 2019 ladyada for Adafruit Industries
# SPDX-License-Identifier: MIT

import time
from os import getenv
import board
import busio
from digitalio import DigitalInOut
```

```
import neopixel
from adafruit_esp32spi import adafruit_esp32spi
from adafruit_esp32spi import adafruit_esp32spi_wifimanager

print("ESP32 SPI webclient test")

# Get wifi details and more from a settings.toml file
# tokens used by this Demo: CIRCUITPY_WIFI_SSID, CIRCUITPY_WIFI_PASSWORD
#                           CIRCUITPY_AIO_USERNAME, CIRCUITPY_AIO_KEY
secrets = {}
for token in ["ssid", "password"]:
    if getenv("CIRCUITPY_WIFI_" + token.upper()):
        secrets[token] = getenv("CIRCUITPY_WIFI_" + token.upper())
for token in ["aio_username", "aio_key"]:
    if getenv("CIRCUITPY_" + token.upper()):
        secrets[token] = getenv("CIRCUITPY_" + token.upper())

if not secrets:
    try:
        # Fallback on secrets.py until depreciation is over and option is removed
        from secrets import secrets
    except ImportError:
        print("WiFi secrets are kept in settings.toml, please add them there!")
        raise

# If you are using a board with pre-defined ESP32 Pins:
esp32_cs = DigitalInOut(board.ESP_CS)
esp32_ready = DigitalInOut(board.ESP_BUSY)
esp32_reset = DigitalInOut(board.ESP_RESET)

# If you have an externally connected ESP32:
# esp32_cs = DigitalInOut(board.D9)
# esp32_ready = DigitalInOut(board.D10)
# esp32_reset = DigitalInOut(board.D5)

# Secondary (SCK1) SPI used to connect to WiFi board on Arduino Nano Connect RP2040
if "SCK1" in dir(board):
    spi = busio.SPI(board.SCK1, board.MOSI1, board.MISO1)
else:
    spi = busio.SPI(board.SCK, board.MOSI, board.MISO)
esp = adafruit_esp32spi.ESP_SPIcontrol(spi, esp32_cs, esp32_ready, esp32_reset)
"""Use below for Most Boards"""
status_light = neopixel.NeoPixel(board.NEOPIXEL, 1, brightness=0.2)
"""Uncomment below for ItsyBitsy M4"""
# status_light = dotstar.DotStar(board.APA102_SCK, board.APA102_MOSI, 1,
brightness=0.2)
"""Uncomment below for an externally defined RGB LED (including Arduino Nano
Connect)"""
# import adafruit_rgbled
# from adafruit_esp32spi import PWMOut
# RED_LED = PWMOut.PWMOut(esp, 26)
# GREEN_LED = PWMOut.PWMOut(esp, 27)
# BLUE_LED = PWMOut.PWMOut(esp, 25)
# status_light = adafruit_rgbled.RGBLED(RED_LED, BLUE_LED, GREEN_LED)

wifi = adafruit_esp32spi_wifimanager.ESPSPI_WiFiManager(esp, secrets, status_light)

counter = 0

while True:
    try:
        print("Posting data...", end="")
        data = counter
        feed = "test"
        payload = {"value": data}
        response = wifi.post(
            "https://io.adafruit.com/api/v2/"
            + secrets["aio_username"]
            + "/feeds/"
```

```
            + feed
            + "/data",
            json=payload,
            headers={"X-AIO-KEY": secrets["aio_key"]},
        )
        print(response.json())
        response.close()
        counter = counter + 1
        print("OK")
    except OSError as e:
        print("Failed to get data, retrying\n", e)
        wifi.reset()
        continue
    response = None
    time.sleep(15)
```

You'll note here we use a secrets.py file to manage our SSID info. The wifimanager is given the ESP32 object, secrets and a neopixel for status indication.

Note, you'll need to add a some additional information to your secrets file so that the code can query the Adafruit IO API:

- `aio_username`
- `aio_key`

You can go to your adafruit.io View AIO Key link to get those two values and add them to the secrets file, which will now look something 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' : '_your_ssid_',
    'password' : '_your_wifi_password_',
    'timezone' : "America/Los_Angeles", # http://worldtimeapi.org/timezones
    'aio_username' : '_your_aio_username_',
    'aio_key' : '_your_aio_key_',
    }
```

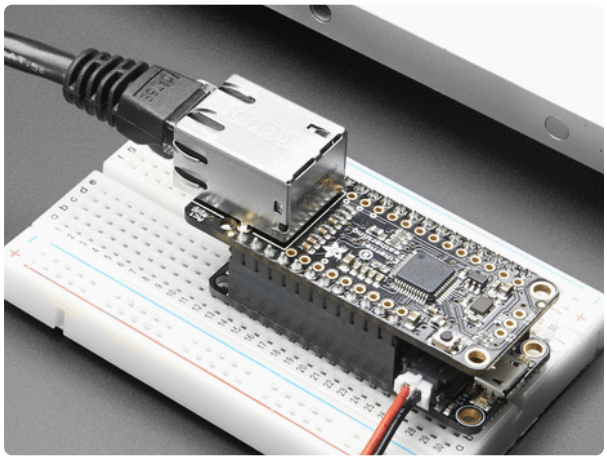
Next, **set up an Adafruit IO feed named `test`**

- If you do not know how to set up a feed, follow this page and come back when you've set up a feed named `test`. (https://adafru.it/f5k)

We can then have a simple loop for posting data to Adafruit IO without having to deal with connecting or initializing the hardware!

Take a look at your **test** feed on Adafruit.io and you'll see the value increase each time the CircuitPython board posts data to it!

For more information on the basics of doing networking in CircuitPython, see this guide:



Networking in CircuitPython
By Anne Barela

https://learn.adafruit.com/networking-in-circuitpython

Code the Matrix Portal

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.

Add Font

Instead of the standard terminalio typeface, this project uses a lovely typeface converted to a bitmap font for use on the matrix display.

- If you want to make your own font, follow this excellent guide (https://adafru.it/EFI).

We'll be using a 64x32 version of the IBM Plex Mono Medium (https://adafru.it/MIA) typeface. Download and uncompress the zip file and then drag it onto the board's **CIRCUITPY** drive.

IBMPlexMono-Medium-24_jep.bdf.zip

https://adafru.it/MIB

Secrets Setup

Instead of relying on a real-time-clock or the microcontroller's software timers, this code uses Adafruit IO's time service to query an exact time for your location. You will need an Adafruit IO account to use this service. If you don't already have an Adafruit login, create one here (https://adafru.it/dAQ).

Once you have logged into your account, there are two pieces of information you'll need to place in your **settings.toml** file: your **Adafruit IO username**, and **Adafruit IO key**. Head to io.adafruit.com (https://adafru.it/fsU) and simply click the **Adafruit IO Key** link on the left hand side of the Adafruit IO page to obtain this information.

Then, add them to the **settings.toml** file:

```
CIRCUITPY_WIFI_SSID = "your_wifi_ssid"
CIRCUITPY_WIFI_PASSWORD = "your_wifi_password"
AIO_USERNAME = "your_aio_username"
AIO_KEY = "your_aio_key"
```

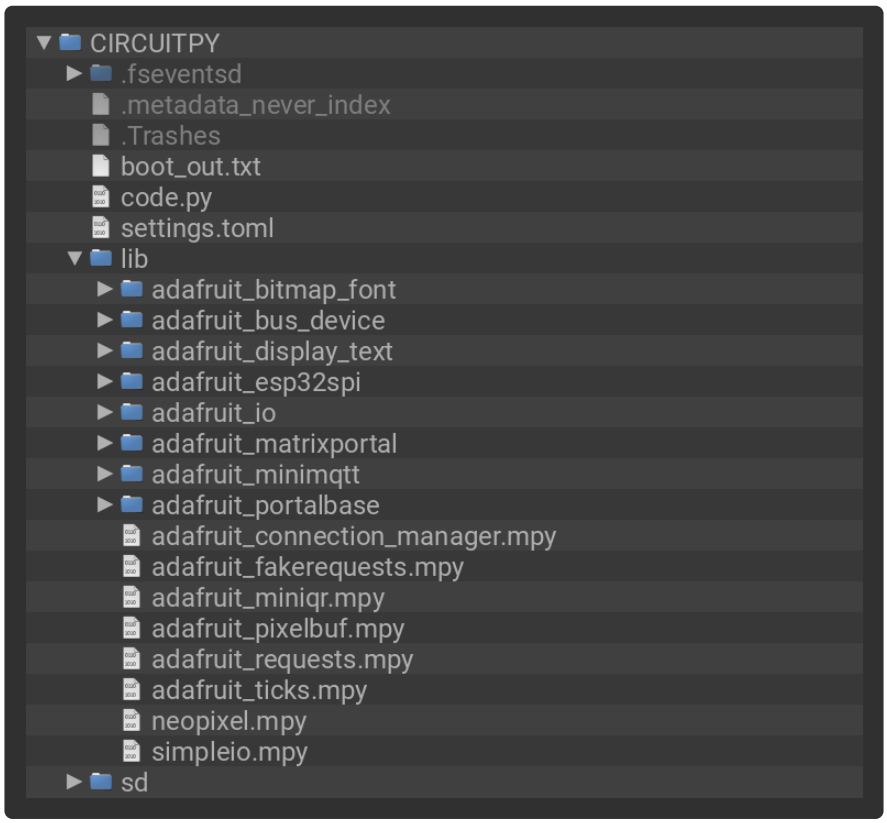
Install Code

To use with CircuitPython, you need to first install a few libraries, into the lib folder on your **CIRCUITPY** drive. Then you need to update **code.py** with the example script.

Thankfully, we can do this in one go. In the example below, click the **Download Project Bundle** button below to download the necessary libraries and the **code.py** file in a zip file. Extract the contents of the zip file, open the directory **Matrix_Portal_Learn_Stats/** and then click on the directory that matches the version

of CircuitPython you're using and copy the contents of that directory to your **CIRCUITPY** drive.

Your **CIRCUITPY** drive should now look similar to the following image:



Connect to the Internet

Once you have CircuitPython setup and libraries installed, next is to get your board connected to the Internet. The process for connecting can be found here (https://adafru.it/NFK). Once you've connected to WiFi using the code on that guide, come back to this page.

Code

If you copied a different **code.py** over when testing the internet connection, make sure to copy the one from this project over again.

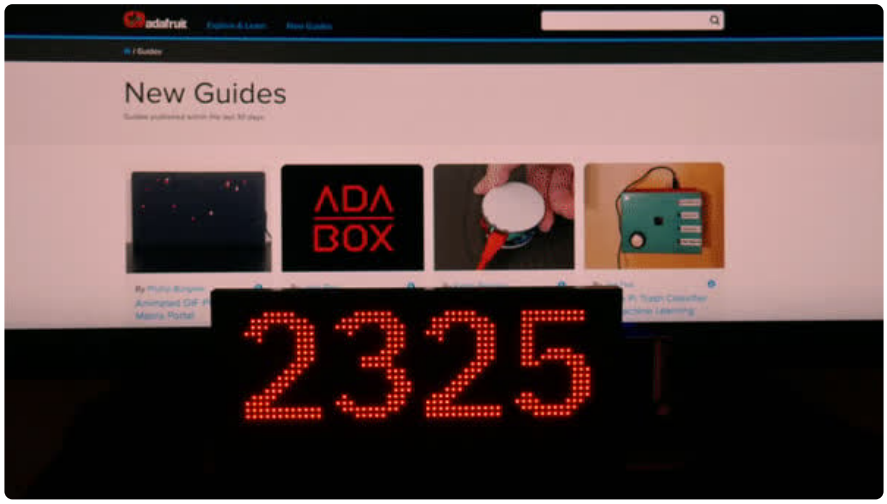
```
# SPDX-FileCopyrightText: 2020 Brent Rubell for Adafruit Industries
#
# SPDX-License-Identifier: MIT

import time
from random import randrange
import board
import terminalio
from adafruit_matrixportal.matrixportal import MatrixPortal

# --- Data Setup --- #
# Number of guides to fetch and display from the Adafruit Learning System
DISPLAY_NUM_GUIDES = 5
```

```
# Data source URL
DATA_SOURCE = (
    "https://learn.adafruit.com/api/guides/new.json?count=%d" % DISPLAY_NUM_GUIDES
)
TITLE_DATA_LOCATION = ["guides"]

matrixportal = MatrixPortal(
    url=DATA_SOURCE,
    json_path=TITLE_DATA_LOCATION,
    status_neopixel=board.NEOPIXEL,
)

# --- Display Setup --- #

# Colors for guide name
colors = [0xFFA500, 0xFFFF00, 0x008000, 0x0000FF, 0x4B0082, 0xEE82EE]

# Delay for scrolling the text
SCROLL_DELAY = 0.03

FONT = "/IBMPlexMono-Medium-24_jep.bdf"

# Learn guide count (ID = 0)
matrixportal.add_text(
    text_font=FONT,
    text_position=(
        (matrixportal.graphics.display.width // 12) - 1,
        (matrixportal.graphics.display.height // 2) - 8,
    ),
    text_color=0x800000,
)
matrixportal.preload_font("0123456789")

# Learn guide title (ID = 1)
matrixportal.add_text(
    text_font=terminalio.FONT,
    text_position=(2, 25),
    text_color=0x000080,
    scrolling=True,
)


def get_guide_info(index):
    """Parses JSON data returned by the DATA_SOURCE
    to obtain the ALS guide title and number of guides and
    sets the text labels.
    :param int index: Guide index to display

    """
    if index > DISPLAY_NUM_GUIDES:
        raise RuntimeError("Provided index may not be larger than
DISPLAY_NUM_GUIDES.")
    print("Obtaining guide info for guide %d..." % index)

    # Traverse JSON data for title
    guide_count = matrixportal.network.json_traverse(als_data.json(),
["guide_count"])

    # Set guide count
    matrixportal.set_text(guide_count, 0)

    guides = matrixportal.network.json_traverse(als_data.json(),
TITLE_DATA_LOCATION)
    guide_title = guides[index]["guide"]["title"]
    print("Guide Title", guide_title)

    # Select color for title text
    color_index = randrange(0, len(colors))
```

```
    # Set the title text color
    matrixportal.set_text_color(colors[color_index], 1)

    # Set the title text
    matrixportal.set_text(guide_title, 1)


refresh_time = None
guide_idx = 0
prv_hour = 0
while True:
    if (not refresh_time) or (time.monotonic() - refresh_time) > 900:
        try:
            print("obtaining time from adafruit.io server...")
            matrixportal.get_local_time()
            refresh_time = time.monotonic()
        except RuntimeError as e:
            print("Unable to obtain time from Adafruit IO, retrying - ", e)
            continue

    if time.localtime()[3] != prv_hour:
        print("New Hour, fetching new data...")
        # Fetch and store guide info response
        als_data = matrixportal.network.fetch(DATA_SOURCE)
        prv_hour = time.localtime()[3]

    # Cycle through guides retrieved
    if guide_idx < DISPLAY_NUM_GUIDES:
        get_guide_info(guide_idx)

        # Scroll the scrollable text block
        matrixportal.scroll_text(SCROLL_DELAY)
        guide_idx += 1
    else:
        guide_idx = 0
    time.sleep(0.05)
```

Code Usage

Every hour, the code will fetch and scroll the five latest guides from the Adafruit Learning System. The number of guides on the Adafruit Learning System will be displayed on top of the scrolling text.



Customize Colors

You can change the colors of the scrolling text. In the code, the scrolling text's color is defined as a list of hex color values.

```
# Colors for guide name
colors = [0xffa500, 0xffff00,
          0x008000, 0x0000ff,
          0x4b0082, 0xee82ee]
```

To add a new color, convert a RGB color to a hex color (https://adafru.it/Oep) and add it to the `colors` list.

Customize Fonts

This code uses the IBMPlexMono Medium typeface to display the number of guides in the Adafruit Learning System. To change the font, you'll first need to convert it to a single bitmap font in a size that works for the 64x32 pixel matrix display.

- Read this guide for more information about making a custom font for the Matrix Portal (https://adafru.it/Oeq).

Then, change the following code in this guide to the name of your new font:

`FONT = "/IBMPlexMono-Medium-24_jep.bdf"`

Change the Scrolling Speed

You can also change the speed of text scrolling across the bottom of the MatrixPortal by editing the following line. The scrolling delay is measured in seconds.

`# Delay for scrolling the text`

`SCROLL_DELAY = 0.03`

Code Walkthrough

Import Libraries

The code first imports the libraries it needs. The `terminalio` library is used for the built-in terminal-style font. The `adafruit_matrixportal` library handles the matrix portal's graphics and networking.

```
import time
import board
from random import randrange
import terminalio
from adafruit_matrixportal.matrixportal import MatrixPortal
```

Data Setup

The next chunk of code configures the data source displayed by the Matrix Portal. You can access the Adafruit Learning System's public API endpoint to display data about new guides by navigating to https://learn.adafruit.com/api/guides/new.json?count=5 (https://adafru.it/Oer).

This API is unique in that the URL contains a count parameter at the end of it. You can increase/decrease this number and see how many guides are loaded on the webpage:

https://learn.adafruit.com/api/guides/new.json**?count=5** (https://adafru.it/Oer)

The code stores the number of guides to display using the `DISPLAY_NUM_GUIDES` variable. The `DATA_SOURCE` variable contains the the URL along with the number of guides to display. The `TITLE_DATA_LOCATION` is list of json traversals used to obtain the data we'll want.

Finally, a new matrixportal object is created using the `DATA_SOURCE` and `TITLE_DATA_LOCATION`.

```
# --- Data Setup --- #
# Number of guides to fetch and display from the Adafruit Learning System
DISPLAY_NUM_GUIDES = 5
# Data source URL
DATA_SOURCE = (
    "https://learn.adafruit.com/api/guides/new.json?count=%d" % DISPLAY_NUM_GUIDES
)
TITLE_DATA_LOCATION = ["guides"]

matrixportal = MatrixPortal(
    url=DATA_SOURCE,
    json_path=TITLE_DATA_LOCATION,
    status_neopixel=board.NEOPIXEL,
)
```

Display Setup

The scrolling text displaying guide names changes colors as it scrolls by. Colors are defined by the `colors` list. Then, the scrolling delay measured in seconds is defined as `SCROLL_DELAY` and the path for the custom font is also defined here.

```
# Colors for guide name
colors = [0xffa500, 0xffff00,
          0x008000, 0x0000ff, 0x4b0082,
          0xee82ee]

# Delay for scrolling the text
SCROLL_DELAY = 0.03

FONT = "/IBMPlexMono-Medium-24_jep.bdf"
```

The matrix portal library includes handy functions to make controlling the graphics on the Matrix Portal easy. The `add_text` function adds text labels to the Matrix Portal's display. You can also pass it settings including the label's font, position, color, and animation.

The first label displays the number of guides currently in the Adafruit Learning system and uses the custom font defined earlier in the code. The `preload_font` function is used to pre-load the glyphs used by the custom font so it doesn't need to be rendered each time. Since the code is only displaying the number of guides in the learning system, the code preloads all the possible numbers glyphs which could be displayed.

The second label displays the learn guide title. Since this text label scrolls, the scrolling parameter is set to `True`.

```
# Learn guide count (ID = 0)
matrixportal.add_text(
    text_font=FONT,
    text_position=((matrixportal.graphics.display.width // 12) - 1,
(matrixportal.graphics.display.height // 2) - 4),
    text_color=0x800000,
)
matrixportal.preload_font("0123456789")

# Learn guide title (ID = 1)
matrixportal.add_text(
    text_font=terminalio.FONT,
    text_position=(2, 25),
    text_color=0x000080,
    scrolling = True
)
```

Main Loop

Every 15 minutes, the code fetches the time from the Adafruit IO Server. If a new hour is elapsed, the `DATA_SOURCE` is queried and the resulting JSON data blob is stored into the `als_data` variable. Instead of fetching and parsing the URL it each time we want to display data, the code uses the `als_data` variable as a simple way of caching the data from the learning system API. We also update the `prv_hour` variable to reflect the numerical value of the new hour.

```
while True:
    if (not refresh_time) or (time.monotonic() - refresh_time) &gt; 900:
        try:
            print("obtaining time from adafruit.io server...")
            matrixportal.get_local_time()
            refresh_time = time.monotonic()
        except RuntimeError as e:
            print("Unable to obtain time from Adafruit IO, retrying - ", e)
            continue

    if time.localtime()[3] != prv_hour:
        print("New Hour, fetching new data...")
        # Fetch and store guide info response
```

```
        als_data = matrixportal.network.fetch(DATA_SOURCE)
        prv_hour = time.localtime()[3]
```

The next chunk of code cycles through the data returned by the Adafruit Learning System API. A call to `get_guide_info` is made using the current guide number.

```
# Cycle through guides retrieved
    if guide_idx &lt; DISPLAY_NUM_GUIDES:
        get_guide_info(guide_idx)
```

The `get_guide_info` method parses the JSON data returned by the `DATA_SOURCE` to obtain the number of guides and current guide title. Then, it sets up the text labels and displays them on the Matrix Portal.

```
def get_guide_info(index):
    """Parses JSON data returned by the DATA_SOURCE
    to obtain the ALS guide title and number of guides and
    sets the text labels.
    :param int index: Guide index to display

    """
    if index &gt; DISPLAY_NUM_GUIDES:
        raise RuntimeError("Provided index may not be larger than
DISPLAY_NUM_GUIDES.")
    print("Obtaining guide info for guide %d..."%index)

    # Traverse JSON data for title
    guide_count = matrixportal.network.json_traverse(als_data.json(),
["guide_count"])

    # Set guide count
    matrixportal.set_text(guide_count, 0)

    # Reset prv_num_guides to the current guide count
    prv_num_guides = guide_count

    guides = matrixportal.network.json_traverse(als_data.json(),
TITLE_DATA_LOCATION)
    guide_title = guides[index]["guide"]["title"]
    print("Guide Title", guide_title)

    # Select color for title text
    color_index = randrange(0, len(colors))

    # Set the title text color
    matrixportal.set_text_color(colors[color_index], 1)

    # Set the title text
    matrixportal.set_text(guide_title, 1)
```

Back in the main loop, the code scrolls the scrollable text label and increments the guide index.

```
# Scroll the scrollable text blocks
        matrixportal.scroll_text(SCROLL_DELAY)
        guide_idx += 1
    else:
        guide_idx = 0
    time.sleep(0.05)
```