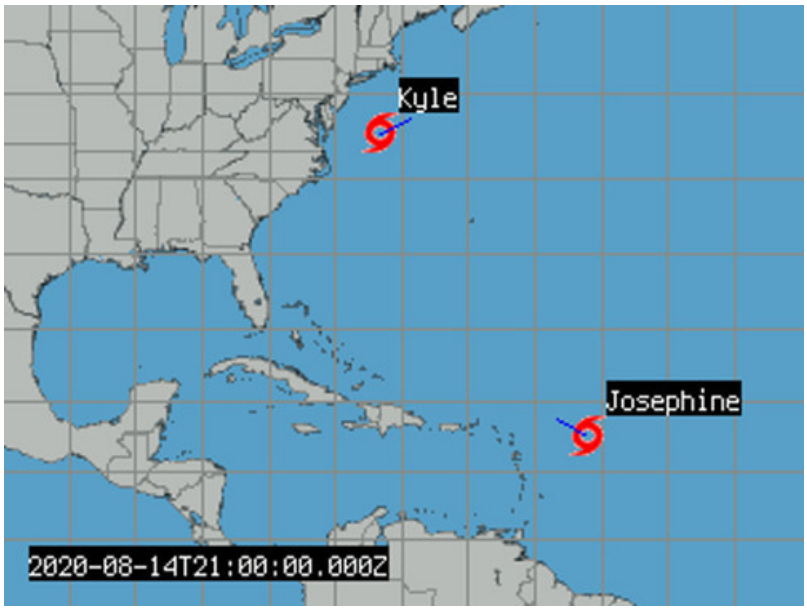
adafruit learning system

# PyPortal Hurricane Tracker

Created by Carter Nelson

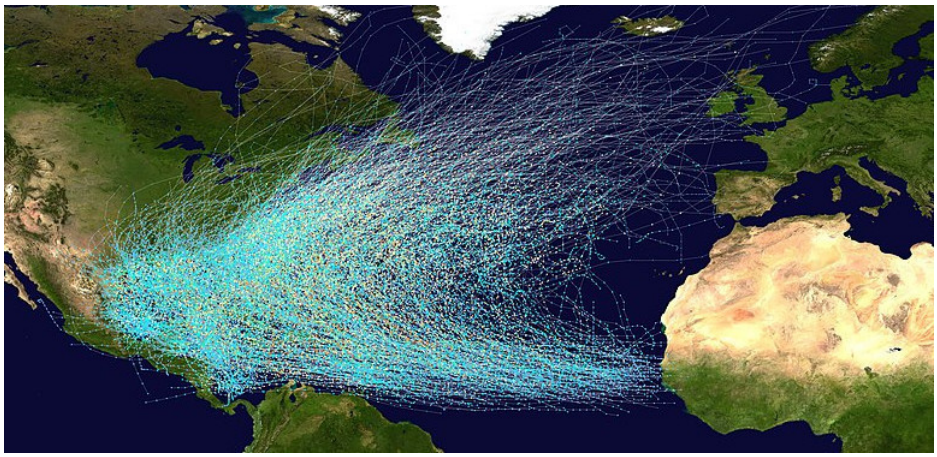



Last updated on 2020-10-28 03:51:24 PM EDT

# Overview

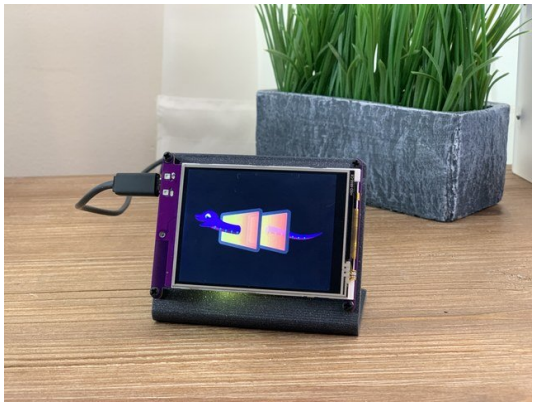

Every summer out in the Atlantic Ocean, storms form off the West coast of Africa and start moving towards North America. Sometimes they dissipate and are gone. Other times they keep growing as they move and can become hurricanes (https://adafru.it/MHe).

This plot of historical storm tracks shows how everything from Panama to Nova Scotia is a potential target.



Hurricanes are very powerful, so when they hit land, it's a serious event. Typical preparation involves boarding up windows, stocking provisions, and getting ready for possible evacuation. So keeping an eye on storm progress is a key part of dealing with hurricane season.

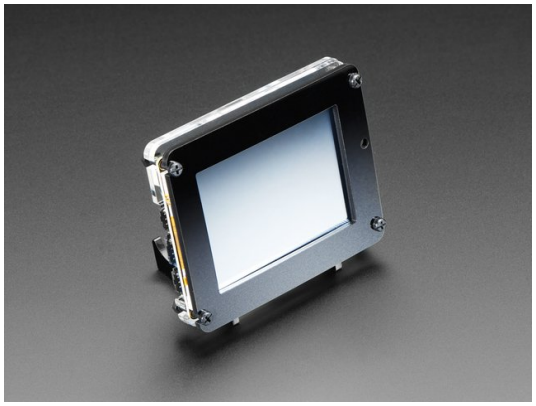
In this guide, we'll show you how you can use your Adafruit PyPortal to display current storm locations. The code is written in CircuitPython (https://adafru.it/EFq).



Adafruit PyPortal - CircuitPython Powered Internet Display

$54.95
IN STOCK

Add To Cart



Adafruit PyPortal Desktop Stand Enclosure Kit

$9.95
IN STOCK

Add To Cart

# 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** "flash" drive to iterate.

The following instructions will show you how to install CircuitPython. If you've already installed CircuitPython but are looking to update it or reinstall it, the same steps work for that as well!

## Set up CircuitPython Quick Start!

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

https://adafru.it/Egk

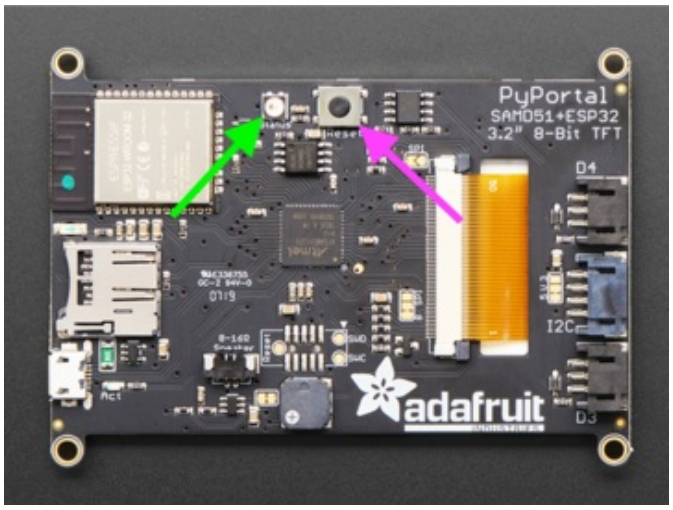
https://adafru.it/Egk

https://adafru.it/HFd

https://adafru.it/HFd


**Click the link above to download the latest version of CircuitPython for the PyPortal.**

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

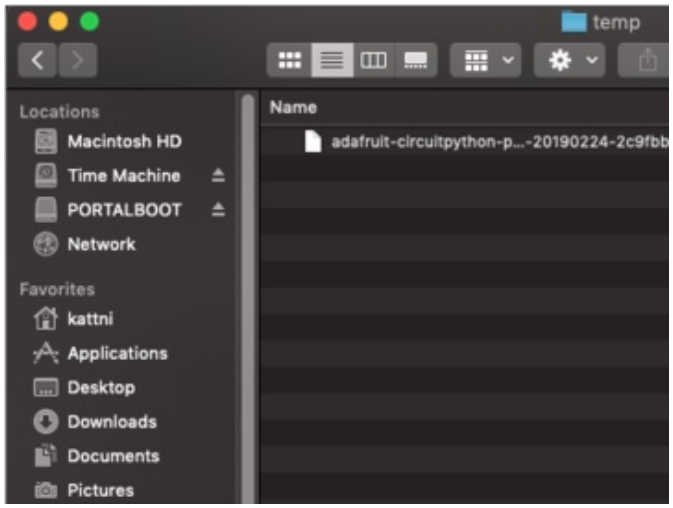


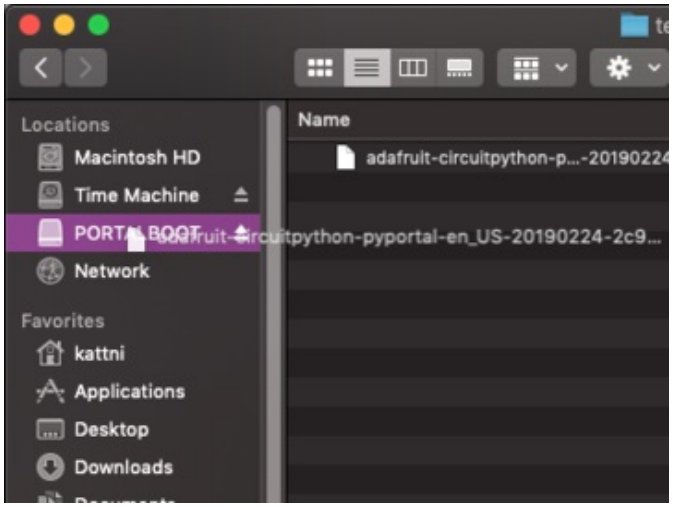

Plug your PyPortal into your computer using a known-good USB cable.

**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 on the top in the middle (magenta arrow) on your board, and you will see the NeoPixel RGB LED (green arrow) turn green. If it turns red, check the USB cable, try another USB port, etc. **Note:** The little red LED next to the USB connector will pulse red. That's ok!

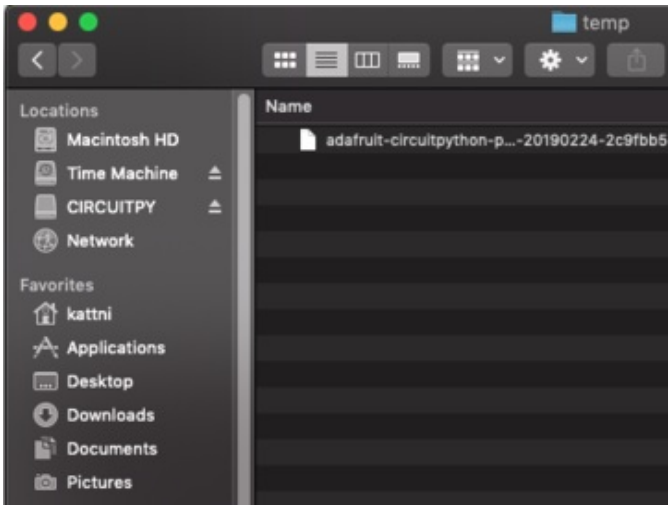
If double-clicking doesn't work the first time, try again. Sometimes it can take a few tries to get the rhythm right!




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

Drag the **adafruit-circuitpython-pyportal-<whatever>.uf2** file to **PORTALBOOT.**




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

If you haven't added any code to your board, the only file that will be present is **boot_out.txt**. This is absolutely normal! It's time for you to add your **code.py** and get started!

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

## PyPortal Default Files

Click below to download a zip of the files that shipped on the PyPortal or PyPortal Pynt.

https://adafru.it/Env

https://adafru.it/Env

https://adafru.it/HFf

https://adafru.it/HFf

# PyPortal CircuitPython Setup

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

## 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-*.x-mpy-*.zip** bundle zip file where ***.x MATCHES THE VERSION OF CIRCUITPYTHON YOU INSTALLED**, and unzip a folder of the same name. Inside you'll find a **lib** folder. You have two options:

- You can add the **lib** folder to your **CIRCUITPY** drive. This will ensure you have *all the drivers*. But it will take a bunch of space on the 8 MB disk
- 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_esp32spi** - This is the library that gives you internet access via the ESP32 using (you guessed it!) SPI transport. You need this for anything Internet
- **adafruit_requests** - This library allows us to perform HTTP requests and get responses back from servers. GET/POST/PUT/PATCH - they're all in here!
- **adafruit_pyportal** - This is our friendly wrapper library that does a lot of our projects, displays graphics and text, fetches data from the internet. Nearly all of our projects depend on it!
- **adafruit_touchscreen** - a library for reading touches from the resistive touchscreen. Handles all the analog noodling, rotation and calibration for you.
- **adafruit_io** - this library helps connect the PyPortal to our free datalogging and viewing service
- **adafruit_imageload** - an image display helper, required for any graphics!
- **adafruit_display_text** - not surprisingly, it displays text on the screen
- **adafruit_bitmap_font** - we have fancy font support, and its easy to make new fonts. This library reads and parses font files.
- **adafruit_slideshow** - for making image slideshows - handy for quick display of graphics and sound
- **neopixel** - for controlling the onboard neopixel
- **adafruit_adt7410** - library to read the temperature from the on-board Analog Devices ADT7410 precision temperature sensor
- **adafruit_sdcard** - support for reading/writing data from the onboard SD card slot.
- **adafruit_bus_device** - low level support for I2C/SPI

# Internet Connect!

Once you have CircuitPython setup and libraries installed we can get your board connected to the Internet. Note that access to enterprise level secured WiFi networks is not currently supported, only WiFi networks that require SSID and password.

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

## What's a 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 ssid',
    'password' : 'my password',
    'timezone' : "America/New_York", # http://worldtimeapi.org/timezones
    'github_token' : 'fawfj23rakjnfawiefa',
    'hackaday_token' : 'h4xx0rs3kret',
    }
```

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 the `ssid` and `password` for your local WiFi setup. 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 it's 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.

## Connect to WiFi

OK now you have your secrets setup - you can connect to the Internet. Lets use the ESP32SPI and the Requests libraries - you'll need to visit the CircuitPython bundle and install (https://adafru.it/ENC):

- **adafruit_bus_device**
- **adafruit_esp32spi**
- **adafruit_requests**

- neopixel

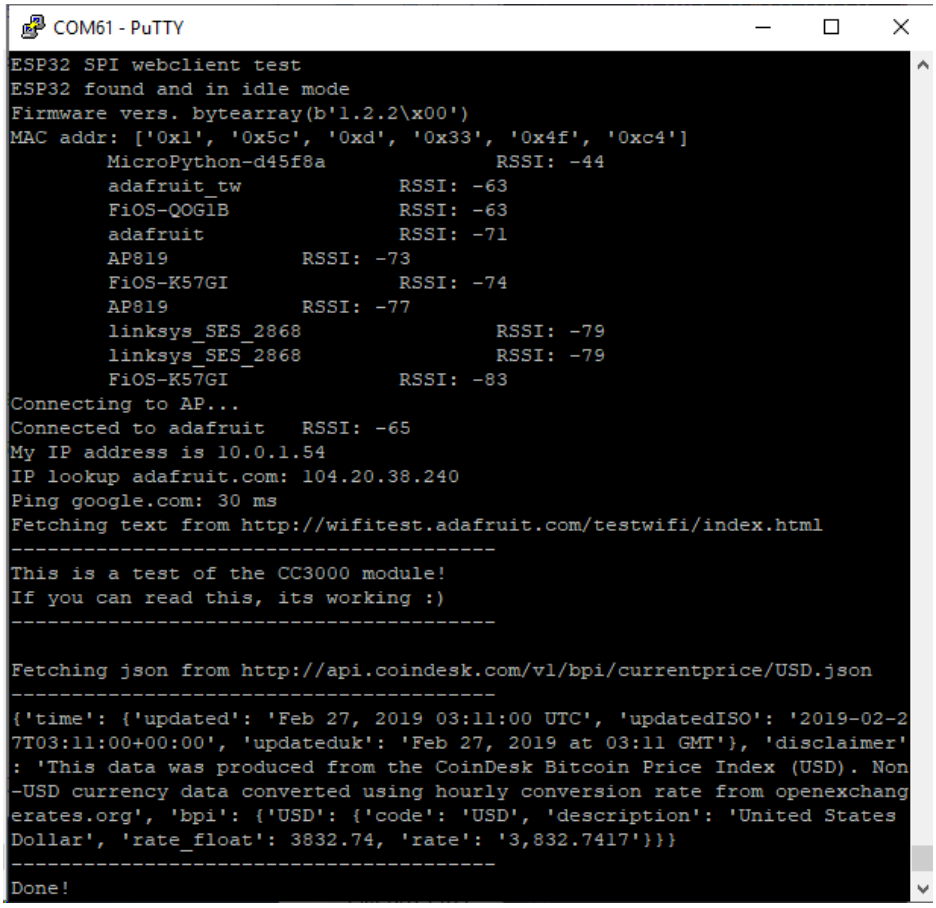
Into your `lib` folder. Once that's done, load up the following example using Mu or your favorite editor:

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

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

# 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 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)

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

requests.set_socket(socket, esp)

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])

for ap in esp.scan_networks():
```

```
for ap in esp.scan_networks():
    print("\t%s\t\tRSSI: %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 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"))
)
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 `secrets.py` file as seen above, with your WiFi ssid and password.




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)

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

Tells our `requests` library the type of socket we're using (socket type varies by connectivity type - we'll be using the `adafruit_esp32spi_socket` for this example). We'll also set the interface to an `esp` object. This is a little bit of a hack, but it lets us use `requests` like CPython does.

```
requests.set_socket(socket, esp)
```

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.

```
# adafruit_requests usage with an esp32spi_socket
```

```
# adafruit_requests usage with an esp32spi_socket
import board
import busio
from digitalio import DigitalInOut
import adafruit_esp32spi.adafruit_esp32spi_socket as socket
from adafruit_esp32spi import adafruit_esp32spi
import adafruit_requests as requests

# Add a secrets.py to your filesystem that has a dictionary called secrets with "ssid" and
# "password" keys with your WiFi credentials. DO NOT share that file or commit it into Git or other
# source control.
# pylint: disable=no-name-in-module,wrong-import-order
try:
    from secrets import secrets
except ImportError:
    print("WiFi secrets are kept in secrets.py, 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)

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(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)

# Initialize a requests object with a socket and esp32spi interface
socket.set_interface(esp)
requests.set_socket(socket)

TEXT_URL = "http://wifitest.adafruit.com/testwifi/index.html"
JSON_GET_URL = "http://httpbin.org/get"
JSON_POST_URL = "http://httpbin.org/post"

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

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

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()

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()

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()
```

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_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)

# Initialize a requests object with a socket and esp32spi interface
requests.set_socket(socket, esp)
```

## 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/FpZ).

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`.

While we requested data from the server, we'd 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.

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

# Add a secrets.py to your filesystem that has a dictionary called secrets with "ssid" and
# "password" keys with your WiFi credentials. DO NOT share that file or commit it into Git or other
# source control.
# pylint: disable=no-name-in-module,wrong-import-order
try:
    from secrets import secrets
except ImportError:
    print("WiFi secrets are kept in secrets.py, 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)

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(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)

# Initialize a requests object with a socket and esp32spi interface
socket.set_interface(esp)
requests.set_socket(socket)

JSON_GET_URL = "http://httpbin.org/get"

# Define a custom header as a dict.
headers = {"user-agent": "blinka/1.0.0"}

print("Fetching JSON data from %s..." % JSON_GET_URL)
response = requests.get(JSON_GET_URL, headers=headers)
print("-" * 60)

json_data = response.json()
headers = json_data["headers"]
print("Response's Custom User-Agent Header: {0}".format(headers["User-Agent"]))
print("-" * 60)

# Read Response's HTTP status code
print("Response HTTP Status Code: ", response.status_code)
print("-" * 60)
```

```
print("-" * 60)

# Read Response, as raw bytes instead of pretty text
print("Raw Response: ", response.content)

# Close, delete and collect the response data
response.close()
```

## 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:

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

import time
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 secrets.py file
try:
    from secrets import secrets
except ImportError:
    print("WiFi secrets are kept in secrets.py, 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)

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 for Most Boards
"""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
# import adafruit_rgbled
# from adafruit_esp32spi import PWMOut
```

```
# 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 (ValueError, RuntimeError) 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_',
    }
```

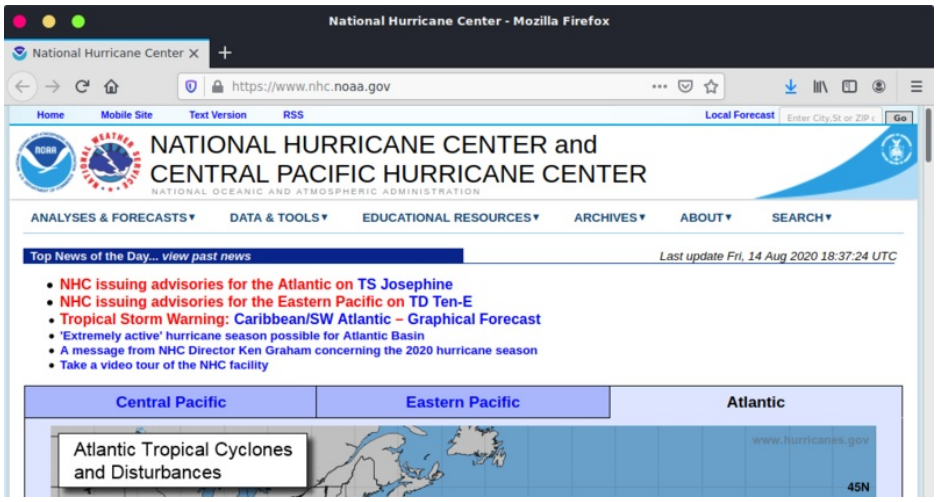
Next, **set up an Adafruit IO feed named `test`**

- 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!

# Current Storm Information




Getting the information about current storms is super easy. The National Hurricane Center (https://adafru.it/MHf), which is part of NOAA (https://adafru.it/ExM), makes various products available. There's a list here:

https://adafru.it/MHA

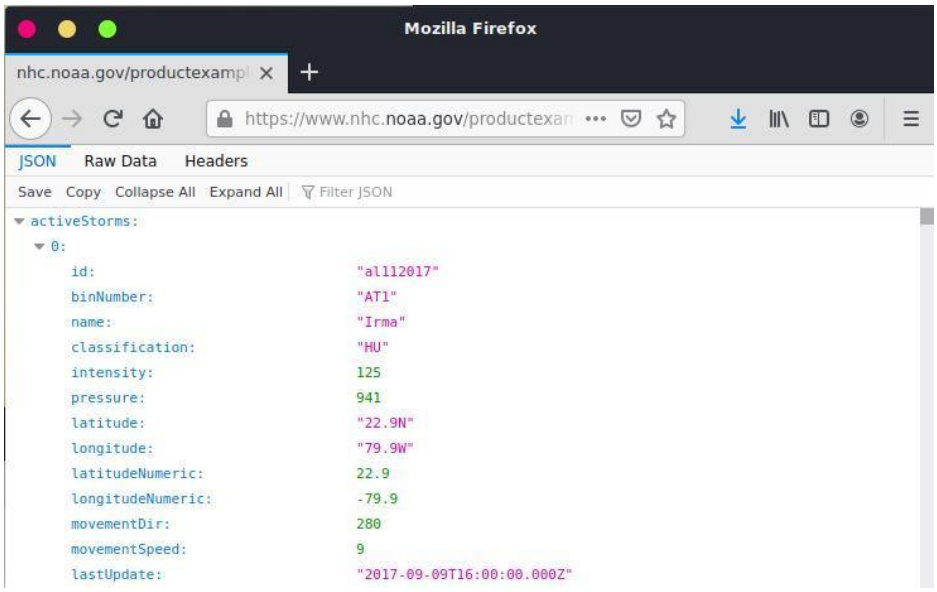
https://adafru.it/MHA

At the very end of the "Text Products" list, there is a link to this JSON source:

https://adafru.it/MHB

https://adafru.it/MHB

This has everything we need! You can open the link in a web browser to see the JSON data:




For each storm, there's a **name**, a **location** in terms of latitude and logitude, a **classification**, as well as other ancillary

information. And the CircuitPython PyPortal library makes grabbing and parsing this data easy.

This PDF has more information about the JSON data source:

https://adafru.it/MHC

https://adafru.it/MHC

# Hurricane Tracker

OK, let's load up our PyPortal with the hurricane tracker code. You'll need a few additional libraries, as mentioned below. You'll also need the BMP files for the map and icons. And finally, there's the code itself.

Note - the hurricane tracker version provided here is for the **Atlantic Ocean only**. The NOAA JSON data source also covers the Eastern and Central Pacific. We think adapting this code for those regions would make for a fun knowledge building exercise.

## Libraries

In addition to all the libraries needed for the PyPortal (see PyPortal CircuitPython Setup), you'll also need these libraries:

- **adafruit_display_shapes**
- **simpleio**

Make sure your **CIRCUITPY/lib** folder contains them.

## Bitmaps

This is the basemap used. Save this as **map.bmp** in your **CIRCUITPY** folder:

https://adafru.it/MIc

https://adafru.it/MIc

This is the sprite sheet bitmap used for the storm icons. Save this as **storm_icons.bmp** in your **CIRCUITPY** folder:

https://adafru.it/MId

https://adafru.it/MId

## Code

Here's the complete code for the PyPortal Hurricane Tracker:

```
import time
import math
import board
import displayio
import terminalio
from simpleio import map_range
import adafruit_imageload
from adafruit_pyportal import PyPortal
from adafruit_display_text.label import Label
from adafruit_display_shapes.line import Line

# --| User Config |------------------------------------------------
UPDATE_RATE = 60  # minutes
MAX_STORMS = 3  # limit storms
NAME_COLOR = 0xFFFFFF  # label text color
NAME_BG_COLOR = 0x000000  # label background color
ARROW_COLOR = 0x0000FF  # movement direction arrow color
```

```
ARROW_LENGTH = 15  # movement direction arrow length
LAT_RANGE = (45, 5)  # set to match map
LON_RANGE = (-100, -40)  # set to match map
# ---------------------------------------------------------------------

# setup pyportal
pyportal = PyPortal(
    url="https://www.nhc.noaa.gov/CurrentStorms.json",
    json_path=["activeStorms"],
    status_neopixel=board.NEOPIXEL,
    default_bg="/map.bmp",
)

# setup display group for storms
icons_bmp, icons_pal = adafruit_imageload.load(
    "/storm_icons.bmp", bitmap=displayio.Bitmap, palette=displayio.Palette
)
for i, c in enumerate(icons_pal):
    if c == 0xFFFF00:
        icons_pal.make_transparent(i)
storm_icons = displayio.Group(max_size=MAX_STORMS)
pyportal.splash.append(storm_icons)
STORM_CLASS = ("TD", "TS", "HU")

# setup info label
info_update = Label(
    terminalio.FONT,
    text="1984-01-01T00:00:00.000Z",
    color=NAME_COLOR,
    background_color=NAME_BG_COLOR,
)
info_update.anchor_point = (0.0, 1.0)
info_update.anchored_position = (10, board.DISPLAY.height - 10)
pyportal.splash.append(info_update)

# these are need for lat/lon to screen x/y mapping
VIRTUAL_WIDTH = board.DISPLAY.width * 360 / (LON_RANGE[1] - LON_RANGE[0])
VIRTUAL_HEIGHT = board.DISPLAY.height * 360 / (LAT_RANGE[0] - LAT_RANGE[1])
Y_OFFSET = math.radians(LAT_RANGE[0])
Y_OFFSET = math.tan(math.pi / 4 + Y_OFFSET / 2)
Y_OFFSET = math.log(Y_OFFSET)
Y_OFFSET = (VIRTUAL_WIDTH * Y_OFFSET) / (2 * math.pi)
Y_OFFSET = VIRTUAL_HEIGHT / 2 - Y_OFFSET


def update_display():
    # clear out existing icons
    while len(storm_icons):
        _ = storm_icons.pop()

    # get latest storm data
    try:
        storm_data = pyportal.fetch()
    except RuntimeError:
        return
    print("Number of storms:", len(storm_data))

    # parse the storm data
    for storm in storm_data:
```

```
        # don't exceed max
        if len(storm_icons) >= MAX_STORMS:
            continue
        # get lat/lon
        lat = storm["latitudeNumeric"]
        lon = storm["longitudeNumeric"]
        # check if on map
        if (
            not LAT_RANGE[0] >= lat >= LAT_RANGE[1]
            or not LON_RANGE[0] <= lon <= LON_RANGE[1]
        ):
            continue
        # OK, let's make a group for all the graphics
        storm_gfx = displayio.Group(max_size=3)  # icon + label + arrow
        # convert to sreen coords
        x = int(map_range(lon, LON_RANGE[0], LON_RANGE[1], 0, board.DISPLAY.width - 1))
        y = math.radians(lat)
        y = math.tan(math.pi / 4 + y / 2)
        y = math.log(y)
        y = (VIRTUAL_WIDTH * y) / (2 * math.pi)
        y = VIRTUAL_HEIGHT / 2 - y
        y = int(y - Y_OFFSET)
        # icon type
        if storm["classification"] in STORM_CLASS:
            storm_type = STORM_CLASS.index(storm["classification"])
        else:
            storm_type = 0
        # create storm icon
        icon = displayio.TileGrid(
            icons_bmp,
            pixel_shader=icons_pal,
            width=1,
            height=1,
            tile_width=16,
            tile_height=16,
            default_tile=storm_type,
            x=x - 8,
            y=y - 8,
        )
        # add storm icon
        storm_gfx.append(icon)
        # add a label
        name = Label(
            terminalio.FONT,
            text=storm["name"],
            color=NAME_COLOR,
            background_color=NAME_BG_COLOR,
        )
        name.anchor_point = (0.0, 1.0)
        name.anchored_position = (x + 8, y - 8)
        storm_gfx.append(name)
        # add direction arrow
        angle = math.radians(storm["movementDir"])
        xd = x + int(ARROW_LENGTH * math.sin(angle))
        yd = y - int(ARROW_LENGTH * math.cos(angle))
        arrow = Line(x, y, xd, yd, color=ARROW_COLOR)
        storm_gfx.append(arrow)
        # add the storm graphics
        storm_icons.append(storm_gfx)
        # update time
```

```
        info_update.text = storm["lastUpdate"]
        # debug
        print(
            "{} @ {},{}".format(
                storm["name"], storm["latitudeNumeric"], storm["longitudeNumeric"]
            )
        )

    # no storms? at least say something
    if not len(storm_icons):
        print("No storms in map area.")
        storm_icons.append(
            Label(
                terminalio.FONT,
                scale=4,
                x=50,
                y=110,
                text="NO STORMS\n IN AREA",
                color=NAME_COLOR,
                background_color=NAME_BG_COLOR,
            )
        )


# ------------------------------------------------------------------
# M A I N
# ------------------------------------------------------------------
update_display()
last_update = time.monotonic()
while True:
    now = time.monotonic()
    if now - last_update > UPDATE_RATE * 60:
        print("Updating...")
        update_display()
        last_update = now
```

# How It Works

## Getting Storm Information

Having a JSON source makes things really easy. And the PyPortal library makes getting that data easy as well. The location is provided when we created the PyPortal object:

```
# setup pyportal
pyportal = PyPortal(
    url="https://www.nhc.noaa.gov/CurrentStorms.json",
    json_path=["activeStorms"],
    status_neopixel=board.NEOPIXEL,
    default_bg="/map.bmp",
)
```

We give it the URL as well as the path location to where we'll find the data. Note that the background map BMP is also specified.

Then, to actually go fetch the data, we just call **fetch()**:

```
storm_data = pyportal.fetch()
```

And if everything works (network connection, etc.), then we should get back a Python list. The list will have a dictionary for each storm. Each dictionary has all the info we want, and we simply access the data via its associated key. For example, the **name**:

```
storm["name"]
```

## Computing Screen Coordinates

The storm location in the JSON file is provided in terms of **latitude** and **longitude**. So some work must be done to convert that into screen **(x, y)** coordinates. See this other guide for a more in depth discussion of what is involved:

https://adafru.it/MHD

https://adafru.it/MHD

The same general approach is used for the hurricane tracker.

## Storm Icons

The storm icons are contained in a single BMP. This uses the concept of a **sprite sheet**, which can break down the single BMP into tiles, with each tile containing a single icon. For more information about sprite sheets, see here:

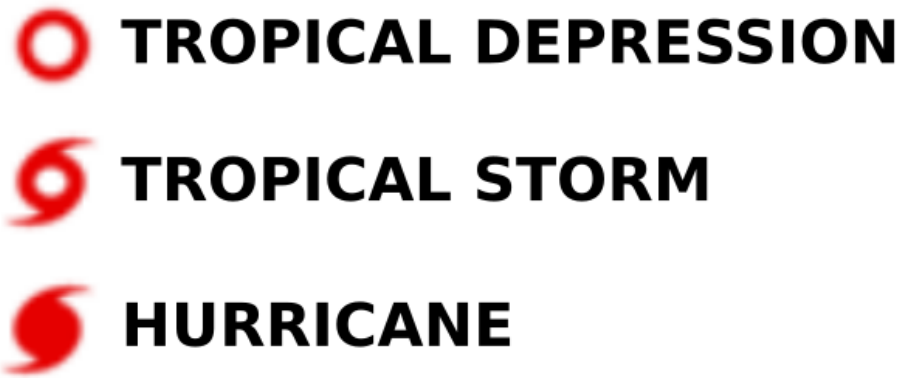
https://adafru.it/GC4

https://adafru.it/GC4

In this case, each icon is 16x16 pixels. There are 3 of them, so the total bitmap size is 16x48 pixels. A background color of yellow is used so that it can easily be found and set as the transparency color. That's what these lines of code do:

```
for i, c in enumerate(icons_pal):
    if c == 0xFFFF00:
        icons_pal.make_transparent(i)
```

Here's a summary of what the icons mean:




## Storm Graphics Group

For each storm found, the tracker will display 3 things at the storm location:

1. An icon based on storm classification
2. The storm name
3. An arrow indicating storm movement direction

To make it easy to place all of these items, a new **displayio.Group** is created for each storm. That's what this line does:

```
storm_gfx = displayio.Group(max_size=3)  # icon + label + arrow
```

Then each of the items are added via the **append()** function. And then the group itself is added to the main display ground via this line:

```
storm_icons.append(storm_gfx)
```

This is a good example of how nesting groups within groups can be useful.

## Putting It All Together

How all this comes together is pretty simple. The initial setup of the PyPortal specifies the JSON data source and background map. That, along with your **secrets.py** file for connecting to your network, sets up most of the hardware.

Then, the current JSON data is fetched. For each storm found within the map region, it's (x, y) location on the screen is computed and the proper icon and other graphics are shown.

And that's it. With a simple time check, this same process is then repeated at regular intervals.