# Table of Contents

**Overview**
- Additional Tools & Materials

**Install CircuitPython**
- Set up CircuitPython Quick Start!
- PyPortal Default Files

**PyPortal CircuitPython Setup**
- Adafruit CircuitPython Bundle

**Internet Connect!**
- What's a secrets file?
- Connect to WiFi
- Requests
- HTTP GET with Requests
- HTTP POST with Requests
- Advanced Requests Usage
- WiFi Manager

**Code PyPortal with CircuitPython**
- Open Weather Maps API Key
- Adafruit IO Time Server
- CircuitPython Code
- How It Works
- Time
- Location
- API Query and JSON
- JSON Traversal
- PyPortal Constructor
- Fetch
- Graphics
- Text Position
- Text Color

**PyPortal Weather Station Case**
- Magnets
- Magnets
Overview

With this compact PyPortal weather station on your desk or stuck to your fridge, you'll always know what mother nature is up to!

Using CircuitPython, this project queries the Open Weather Maps site API to find out the current weather for your location and displays it along with an informative icon.

3D printed case () and magnetic backing allow you to mount it to your fridge or other ferrous metal surface.
Adafruit PyPortal - CircuitPython Powered Internet Display

PyPortal, our easy-to-use IoT device that allows you to create all the things for the “Internet of Things” in minutes. Make custom touch screen interface...
https://www.adafruit.com/product/4116

5V 2A Switching Power Supply w/ USB-A Connector

Our 5V 2A USB power adapter is the perfect choice for powering single-board computers like Raspberry Pi, BeagleBone, or anything else that's power-hungry! This adapter was...

USB A/Micro Cable - 2m

This is your standard USB A-Plug to Micro-USB cable. It's 2 meters long so you'll have plenty of cord to work with for those longer extensions.
https://www.adafruit.com/product/2185

Additional Tools & Materials

You may want to create the optional mounting case for your PyPortal Weather Station. For this you'll need:

- 3D printer ()
- Filament ()
- PyPortal Case model files ()
• **M2.5 screws**

If you don't have access to a 3D printer you can optionally use an online service such as 3D Hubs to have it printed for you on demand.

**Magnetic Pin Back**
These magnetic pin backs have two pieces: a metal bar with adhesive strip and a plastic piece with two strong rare-earth magnets. Affix the metal bar to your FLORA projects-- it's...

https://www.adafruit.com/product/1170

**Black Nylon Machine Screw and Stand-off Set – M2.5 Thread**
Totaling 380 pieces, this M2.5 Screw Set is a must-have for your workstation. You'll have enough screws, nuts, and hex standoffs to fuel your maker...

https://www.adafruit.com/product/3299

**Ultimaker 2+ 3D Printer**
The Ultimaker 2+ is one of our favorite 3D printers on the market. It's a well-built open-source compact machine with an excellent UX. Every inch of the...

https://www.adafruit.com/product/2673
PLA Filament for 3D Printers - 2.85mm Diameter - Lilac - 1 Kg
Having a 3D printer without filament is sort of like having a regular printer without paper or ink. And while a lot of printers come with some filament there's a good chance...
https://www.adafruit.com/product/3733

Install CircuitPython

CircuitPython () is a derivative of MicroPython () 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 :)

Download the latest version of CircuitPython for the PyPortal via CircuitPython.org

Download the latest version of CircuitPython for the PyPortal Pynt via CircuitPython.org
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-pyporal-<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.
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:

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_portalbase - This library is the base library that adafruit_pyportal library is built on top of.
- 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 (not necessary for Titano or Pynt)
• adafruit_bus_device - low level support for I2C/SPI
• adafruit_fakerequests - This library allows you to create fake HTTP requests by using local files.

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:

```python
# 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() 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.

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:
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():
    print("%s	RSSI: %d" % (str(ap["ssid"], "utf-8").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").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 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.
In order, the example code...

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

```python
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.

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

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

```python
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:

```python
for ap in esp.scan_networks():
    print("t%stRSSI: %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)

```python
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: ", 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` - 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

```python
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)

```python
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 () library for web interfacing named Adafruit_CircuitPython_on_Requests (). 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:

```
# SPDX-FileCopyrightText: 2021 ladyada for Adafruit Industries
# SPDX-License-Identifier: MIT

# 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:
```
```python
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)

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

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, esp)

TEXT_URL = "http://wifitest.adafruit.com/testwifi/index.html"
JSON_GET_URL = "https://httpbin.org/get"
JSON_POST_URL = "https://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 `requests` object using an ESP32 `socket` and the `esp` object.

```python
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](http://wifitest.adafruit.com/testwifi/index.html).

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`.

```
HTTP GET with Requests

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

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 collects the response's data.
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`.

HTTP POST with Requests

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

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

©Adafruit Industries
# 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"), ";
RSSI:", esp.rssi)

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

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)

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

WiFi Manager

That simplestest 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:

```python
# 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")

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_wordclock = DigitalInOut(board.ESP_WDC)
esp32_mosi = DigitalInOut(board.ESP_MOSI)
esp32_micromagi = DigitalInOut(board.ESP_I2C_M"
```
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
# 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:

```python
# 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': '_yourssid_',
    'password': '_yourwifi_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**. ()

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!
Code PyPortal with CircuitPython

Open Weather Maps API Key

We'll be using OpenWeatherMaps.org to retrieve the weather info through its API. In order to do so, you'll need to register for an account and get your API key.

Go to this [link](#) and register for a free account. Once registered, you'll get an email containing your API key, also known as the "openweather token".

Copy and paste this key into your secrets.py file that is on the root level of your CIRCUITPY drive, so it looks something like this:

```python
secrets = {
    'ssid' : 'your_wifi_ssid',
    'password' : 'your_wifi_password',
    'openweather_token' : 'xxxxxxxxxxxxxxxxxxxxxxxxx'
}
```

Adafruit IO Time Server

In order to get the precise time, our project will query the Adafruit IO Internet of Things service for the time. Adafruit IO is absolutely free to use, but you'll need to log in with your Adafruit account to use it. If you don't already have an Adafruit login, create one [here](#).

If you haven't used Adafruit IO before, check [out this guide](#) for more info.

Once you have logged into your account, there are two pieces of information you'll need to place in your secrets.py file: Adafruit IO username, and Adafruit IO key. Head to [io.adafruit.com](http://io.adafruit.com) and simply click the View AIO Key link on the left hand side of the Adafruit IO page to get this information.

Then, add them to the secrets.py file like this:

```python
secrets = {
    'ssid' : 'your_wifi_ssid',
    'password' : 'your_wifi_password',
    'openweather_token' : 'xxxxxxxxxxxxxxxxxxxxxxxxx',
    'aio_username' : '_your_aio_username_',
    'aio_key' : '_your_big_huge_super_long_aio_key_'
}
```
CircuitPython Code

In the embedded code element below, click on the Download: Project Zip link, and save the .zip archive file to your computer.

Then, uncompressed the .zip file, it will unpack to a folder named PyPortal_OpenWeather.

Copy the contents of the PyPortal_OpenWeather directory to your PyPortal's CIRCUITPY drive.

This is what the final contents of the CIRCUITPY drive will look like:
This example queries the Open Weather Maps site API to find out the current weather for your location... and display it on a screen!

if you can find something that spits out JSON data, we can display it

import sys
import time
import board
from adafruit_pyportal import PyPortal
cwd = (“/”+_file_.rsplit(‘/’, 1)[0]) # the current working directory (where this file is)
sys.path.append(cwd)
import openweather_graphics # pylint: disable=wrong-import-position

# 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
# Use cityname, country code where countrycode is ISO3166 format.
# E.g. "New York, US" or "London, GB"
LOCATION = "Manhattan, US"

# Set up where we'll be fetching data from
DATA_SOURCE = "http://api.openweathermap.org/data/2.5/weather?q=\"+LOCATION
DATA_SOURCE += \"&appid=\"+secrets[\'openweather_token\']\"
# You'll need to get a token from openweathermap.org, looks like
'b6907d289e10d714a6e88b30761fae22'
DATA_LOCATION = []

# Initialize the pyportal object and let us know what data to fetch and where
# to display it
pyportal = PyPortal(url=DATA_SOURCE,
                      json_path=DATA_LOCATION,
                      status_neopixel=board.NEOPIXEL,
                      default_bg=0x000000)
gfx = openweather_graphics.OpenWeather_Graphics(pyportal.splash, am_pm=True,
celsius=False)

localtile_refresh = None
weather_refresh = None

while True:
    # only query the online time once per hour (and on first run)
    if (not localtile_refresh) or (time.monotonic() - localtile_refresh) > 3600:
        try:
            print("Getting time from internet!")
            pyportal.get_local_time()
            localtile_refresh = time.monotonic()
        except RuntimeError as e:
            print("Some error occurred, retrying! -", e)
            continue

    # only query the weather every 10 minutes (and on first run)
    if (not weather_refresh) or (time.monotonic() - weather_refresh) > 600:
        try:
            value = pyportal.fetch()
            print("Response is", value)
            gfx.display_weather(value)
            weather_refresh = time.monotonic()
        except RuntimeError as e:
            print("Some error occurred, retrying! -", e)
            continue

gfx.update_time()
time.sleep(30)  # wait 30 seconds before updating anything again

If you run into any errors, such as "ImportError: no module named `adafruit_display_text.label`" be sure to update your libraries to the latest release bundle!

How It Works

The PyPortal Weather Station has a few steps it takes to provide you with the information you desire! It has a boot-up screen, weather icons, and multiple fonts for displaying the info.
Background

First, it displays a bitmap graphic as the screen's startup background until it connects to the Open Weather Maps server to get the weather info. This is a 320 x 240 pixel RGB 16-bit raster graphic in .bmp format.

Time

Next, the program connects through the WiFi to get the local time via the adafruit.io server, which will be displayed in the upper right corner of the display.

Location

In the code.py file (which you will have renamed from openweather.py) you can change the location for which you want to display the weather in this line:

```python
# Use cityname, country code where countrycode is ISO3166 format.
# E.g. “New York, US” or “London, GB”
LOCATION = "Manhattan, US"
```

API Query and JSON

Using this information, the code can then send a query to Open Weather Maps's API that looks something like this:

```
```

(where all of those 'x's are your token).
When this query is complete, it returns a JSON file that looks like this:

```
{
    "coord": {
        "lon": -118.24,
        "lat": 34.05
    },
    "weather": [
        {
            "id": 501,
            "main": "Rain",
            "description": "moderate rain",
            "icon": "10d"
        }
    ],
    "base": "stations",
    "main": {
        "temp": 287.42,
        "pressure": 1016,
        "humidity": 50,
        "temp_min": 285.15,
        "temp_max": 289.15
    },
    "visibility": 16093,
    "wind": {
        "speed": 3.6,
        "deg": 300
    },
    "rain": {
        "1h": 1.52
    },
    "clouds": {
        "all": 75
    },
    "dt": 1552073935,
    "sys": {
        "type": 1,
        "id": 3514,
        "message": 0.0087,
        "country": "US",
        "sunrise": 1552054308,
        "sunset": 1552096542
    },
    "id": 5368361,
    "name": "Los Angeles",
    "cod": 200
}
```

Here is the same file beautified with the Firefox browser's built in tools (You can also use online code "beautifiers" such as https://codebeautify.org/jsonviewer () or http://jsonviewer.stack.hu ()):

©Adafruit Industries
The JSON file is formatted in a way that makes it easy to traverse the hierarchy and parse the data. In it, you'll see keys, such as `main`, `description`, `icon`, and `temp`, and their respective values. So, here are some key : value pairs we care about for the weather station:

- "main" : "Rain"
- "description" : "moderate rain"
- "icon" : "10d"
In order to fetch this data from the file, we need to be able to describe their locations in the file hierarchically. This is helpful, for example, in differentiating between the ‘main’ weather condition and the ‘main’ section containing temperature and other data. To avoid name clashing we rely on JSON traversal.

In the openweather_graphics.py file, you’ll see how this is done. For example, the main key is found in this hierarchy of the JSON file: ['weather'], [0], ['main']

This means there is a key at the top level of the JSON file called ‘weather’, which has a sub-tree indexed [0], and then below that is the ‘main’ key.

This process is used to cast the values of the temperature, weather, description, and which icon to display from the directory of bitmap icons.

These are the icons represented:

![Icons](https://example.com/icons)

Font

The data is displayed as text created with bitmapped fonts to overlay on top of the background. The fonts used here are bitmap fonts made from the Arial typeface. You can learn more about converting type in this guide.
PyPortal Constructor

When we set up the `pyportal` constructor, we are providing it with these things:

- **url** to query
- **json_path** to traverse and find the key:value pairs we need
- **default_bg** default background color

Fetch

With the PyPortal set up, we can then use `pyportal.fetch()` to do the query and parsing of the weather data and then display it on screen.

All of the heavy lifting of parsing that data and displaying it as text or bitmaps is done in the `openweather_graphics.py` code.

Graphics

Let’s have a look at how the `openweather_graphics.py` code places the elements on screen. Below, we can see the icon and text that are displayed. The items in quotes are the key names from the JSON file, and their values are what we see displayed using the CircuitPython `label` library.
import time
import json
import displayio
from adafruit_display_text.label import Label
from adafruit_bitmap_font import bitmap_font

cwd = ('/'+__file__).rsplit('/', 1)[0] # the current working directory (where this file is)

small_font = cwd+"/fonts/Arial-12.bdf"
medium_font = cwd+"/fonts/Arial-16.bdf"
large_font = cwd+"/fonts/Arial-Bold-24.bdf"

class OpenWeather_Graphics(displayio.Group):
    def __init__(self, root_group, *, am_pm=True, celsius=True):
        super().__init__()
        self.am_pm = am_pm
        self.celsius = celsius

        root_group.append(self)
        self._icon_group = displayio.Group()
        self.append(self._icon_group)
        self._text_group = displayio.Group()
        self.append(self._text_group)

        self._icon_sprite = None
        self._icon_file = None
        self.set_icon(cwd+"/weather_background.bmp")

        self.small_font = bitmap_font.load_font(small_font)
        self.medium_font = bitmap_font.load_font(medium_font)
        self.large_font = bitmap_font.load_font(large_font)
        glyphs = b'0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ-,.: '
        self.small_font.load_glyphs(glyphs)
        self.medium_font.load_glyphs(glyphs)
        self.large_font.load_glyphs(('°',))  # a non-ascii character we need for sure
        self.city_text = None

        self.time_text = Label(self.medium_font)
        self.time_text.x = 200
        self.time_text.y = 12
        self.time_text.color = 0xFFFFFF
        self._text_group.append(self.time_text)

        self.temp_text = Label(self.large_font)
        self.temp_text.x = 200
        self.temp_text.y = 195
        self.temp_text.color = 0xFFFFFF
        self._text_group.append(self.temp_text)

        self.main_text = Label(self.large_font)
        self.main_text.x = 10
        self.main_text.y = 195
        self.main_text.color = 0xFFFFFF
        self._text_group.append(self.main_text)

        self.description_text = Label(self.small_font)
        self.description_text.x = 10
        self.description_text.y = 225
        self.description_text.color = 0xFFFFFF
        self._text_group.append(self.description_text)

        def display_weather(self, weather):
            weather = json.loads(weather)
# set the icon/background
weather_icon = weather['weather'][0]['icon']
self.set_icon(cwd+"/icons/"+weather_icon".bmp")

city_name = weather['name'] + ", " + weather['sys']['country']
print(city_name)
if not self.city_text:
    self.city_text = Label(self.medium_font, text=city_name)
    self.city_text.x = 10
    self.city_text.y = 12
    self.city_text.color = 0xFFFFFF
    self._text_group.append(self.city_text)

self.update_time()

main_text = weather['weather'][0]['main']
print(main_text)
self.main_text.text = main_text

temperature = weather['main']['temp'] - 273.15 # its...in kelvin
print(temperature)
if self.celsius:
    self.temp_text.text = "%d °C" % temperature
else:
    self.temp_text.text = "%d °F" % ((temperature * 9 / 5) + 32)

description = weather['weather'][0]['description']
description = description[0].upper() + description[1:]
print(description)
self.description_text.text = description
# "thunderstorm with heavy drizzle"

def update_time(self):
    """Fetch the time.localtime(), parse it out and update the display text""
    now = time.localtime()
    hour = now[3]
    minute = now[4]
    format_str = "%d:%02d"
    if self.am_pm:
        if hour >= 12:
            hour -= 12
        format_str = format_str + " PM"
    else:
        format_str = format_str + " AM"
    if hour == 0:
        hour = 12
    time_str = format_str % (hour, minute)
    print(time_str)
    self.time_text.text = time_str

def set_icon(self, filename):
    """The background image to a bitmap file.
    :param filename: The filename of the chosen icon
    ""
    print("Set icon to ", filename)
    if self._icon_group:
        self._icon_group.pop()
    if not filename:
        return # we're done, no icon desired

    # CircuitPython 6 & 7 compatible
    if self._icon_file:
        self._icon_file.close()
    self._icon_file = open(filename, "rb")
    icon = displayio.OnDiskBitmap(self._icon_file)
    self._icon_sprite = displayio.TileGrid(©Adafruit Industries Page 35 of 42
# Text Position

Depending on the design of your background bitmap and the length of the text you're displaying, you may want to reposition the text and caption.

The PyPortal's display is 320 pixels wide and 240 pixels high. In order to refer to those positions on the screen, we use an x/y coordinate system, where x is horizontal and y is vertical.

The origin of this coordinate system is the upper left corner. This means that a pixel placed at the upper left corner would be (0,0) and the lower right corner would be (320, 240).

So, if you wanted to move the subscriber count text to the right and up closer to the top, your code may look like this for that part of the PyPortal constructor: `text_position=(250, 10)`
Text Color

Another way to customize your display is to adjust the color of the text. The line `text_color=0xFFFFFF` in the constructor shows how. You will need to use the hexadecimal value for any color you want to display.

You can use something like [https://htmlcolorcodes.com/](https://htmlcolorcodes.com/) to pick your color and then copy the hex value, in this example it would be `0x0ED9EE`.

Now, we'll look at mounting the PyPortal into a case for display!

**PyPortal Weather Station Case**
I decided to use the portable PyPortal case designed by the Ruiz Bros. as my enclosure. [You can go to this guide to build it](https://learn.adafruit.com) -- either by 3D printing it yourself or sending out to a service such as 3DHubs.

I didn't need this to be portable, so I omitted the PowerBoost, battery, speaker, and switch.

I used all of the same 3D parts, and screwed the mount together, then snapped the top and bottom into place.
Magnets

I wanted to mount my PyPortal Weather Station on my refrigerator, so I affixed magnets.
Magnets

Use a piece of double stick foam tape to secure the magnetic pin back to the case as shown.
I tested it on my workbench drawer and it worked great! Time to place it on the fridge and plug it in!
Now, every 30 seconds, your weather station will update with the latest info!