PyPortal Air Quality Display
Created by John Park

Last updated on 2021-09-13 05:46:29 PM EDT
# Guide Contents

Guide Contents .............................................. 2
Overview ...................................................... 3
  Additional Tools & Materials ......................... 4
Install CircuitPython ......................................... 6
  Set up CircuitPython Quick Start! .................... 6
  PyPortal Default Files .................................. 8
PyPortal CircuitPython Setup ............................. 9
Adafruit CircuitPython Bundle ............................ 9
Internet Connect! .......................................... 11
What's a secrets file? ..................................... 11
Connect to WiFi ............................................ 11
Requests ..................................................... 16
  HTTP GET with Requests ............................... 18
  HTTP POST with Requests ............................. 19
  Advanced Requests Usage ............................. 20
WiFi Manager ................................................ 21
Code PyPortal with CircuitPython ...................... 25
  AirNow API Key ......................................... 25
  Adafruit IO Time Server ............................... 25
  CircuitPython Code ..................................... 26
  How It Works ............................................ 28
  WiFi ....................................................... 28
  API Query and JSON .................................... 28
  JSON Traversal ......................................... 32
    Background ........................................... 32
    Font ................................................... 32
  PyPortal Constructor .................................... 33
  Fetch .................................................... 33
  Text Position .......................................... 33
  Text Color .............................................. 34
PyPortal Air Quality Display Case ....................... 35
  Magnets .................................................. 36
  Magnets .................................................. 37

© Adafruit Industries
https://learn.adafruit.com/pyportal-air-quality-display
Page 2 of 40
Overview

With the PyPortal air quality display stuck to your fridge, you can keep track of the quality of the air you breathe.

Using CircuitPython, this project queries the AirNow site API to find out the current air quality index for your location and displays it along with an informative color.

3D printed case (https://adafru.it/Eel) and magnetic backing allow you to mount it to your fridge or other ferrous metal surface.

Note that the Air Quality information used in this tutorial is from https://airnow.gov/ which only provides information for the United States and then only certain geographic areas (by postal Zip Code). If you wish to use this project in other areas, you may have to adapt the project to another data source which is not covered in this guide.

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...
$54.95
In Stock
Add to Cart

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...

$7.95  
In Stock  
Add to Cart

**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.  
$4.95  
In Stock  
Add to Cart

### Additional Tools & Materials

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

- [3D printer](https://adafru.it/zef)
- [Filament](https://adafru.it/EcM)
- [PyPortal Case model files](https://adafru.it/Eem)
- [M2.5 screws](https://adafru.it/wsc)

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...

$1.95  
In Stock  
Add to Cart

**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...

$16.95  
In Stock  
Add to Cart

**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...
Out of Stock

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...
Install CircuitPython

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

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://adafruit.it/UFZ
https://adafruit.it/UGa
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

Download the `adafruit-circuitpython-bundle-*-x-mpy-*-zip` bundle zip file where `*` 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_sdcards** - support for reading/writing data from the onboard SD card slot.
- **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](https://adafruit.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. Let's use the ESP32SPI and the Requests libraries - [you'll need to visit the CircuitPython bundle and install](https://adafruit.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:

```python
# 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)
```

© Adafruit Industries  https://learn.adafruit.com/pyportal-air-quality-display
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("t%s\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"), ", RSSI: ", 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 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://adafruit.com/Bec).
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
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("%s		RSSI: %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"), ", RSSI:", 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 - 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)
Requests

We’ve written a requests-like (https://adafruit.it/Kpa) library for web interfacing named Adafruit_CircuitPython_Requests (https://adafruit.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.

```
# 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:
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)
es = 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)

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 request 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](https://adafruit.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.
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.

```python
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.

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

import board
import busio
from digitalio import DigitalInOut
import adafruit_esp32spi.adafruit_esp32spi_socket as socket
import 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, esp)

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

# Define a custom header as a dict.
headers = {"user-agent": "blinky/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 simplest 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:

```python
import time
import board
import busio
from digitalio import DigitalInOut
import neopixel
from adafruit_esp32spi import adafruit_esp32spi
```
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:
estp32_cs = DigitalInOut(board.ESP_CS)
estp32_ready = DigitalInOut(board.ESP_BUSY)
estp32_reset = DigitalInOut(board.ESP_RESET)

# If you have an externally connected ESP32:
# estp32_cs = DigitalInOut(board.D9)
# estp32_ready = DigitalInOut(board.D10)
# estp32_reset = DigitalInOut(board.D5)

spi = busio.SPI(board.SCK, board.MOSI, board.MISO)
estp32 = adafruit_esp32spi.ESP_SPIcontrol(spi, estp32_cs, estp32_ready, estp32_reset)

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(estp32, 26)
# GREEN_LED = PWMOut.PWMOut(estp32, 27)
# BLUE_LED = PWMOut.PWMOut(estp32, 25)
# status_light = adafruit_rgbled.RGBLED(RED_LED, BLUE_LED, GREEN_LED)

wifi = adafruit_esp32spi_wifimanager.ESPSPI_WiFiManager(estp32, 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:

```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' : '_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://adafruit.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!
<table>
<thead>
<tr>
<th>Created at</th>
<th>Value</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>2019-11-27 12:07am</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>2019-11-27 14:17pm</td>
<td>29</td>
<td></td>
</tr>
<tr>
<td>2019-11-27 19:35pm</td>
<td>38</td>
<td></td>
</tr>
<tr>
<td>2019-11-28 11:10am</td>
<td>37</td>
<td></td>
</tr>
<tr>
<td>2019-11-28 10:17am</td>
<td>36</td>
<td></td>
</tr>
</tbody>
</table>
AirNow API Key

We'll be using the US government's site [airnow.gov](https://adafru.it/EgC) to retrieve the air quality index 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](https://adafru.it/Eep) and register. Once registered, you'll get an email containing your API key, also known as the "airnow 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',
    'airnow_token' : 'XXX0XXXX-0000-0000-0000-XXXXXXXX0000'
}
```

Airnow.Gov only provides data for the United States and only for specific postal Zip codes. Please check to ensure data for your area is available. While this guide can be used as a template for using other data sources, changes to do so are not in the scope of this guide and Adafruit cannot assist in changes for other data sources.

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](https://adafru.it/dAQ).

If you haven't used Adafruit IO before, [check out this guide for more info](https://adafru.it/Ef8).

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](https://adafru.it/fsU) 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:
secrets = {
    'ssid' : 'your_wifi_ssid',
    'password' : 'your_wifi_password',
    'airnow_token' : 'XXX0XXXX-0000-0000-0000-XXXXXXXX0000',
    '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, uncompress the .zip file, it will unpack to a folder named PyPortal_AirQuality.

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

This is what the final contents of the CIRCUITPY drive will look like:

This example will access the twitter follow button API, grab a number like the number of followers... and display it on a screen!
the number of followers... and display it on a screen!
if you can find something that spits out JSON data, we can display it

import time
import board
from adafruit_pyportal import PyPortal

# 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

# Change this to your zip code, not all locations have AQI data. Check
# https://airnow.gov/ to see if there's data available!
LOCATION = "90210"

# Set up where we'll be fetching data from
DATA_SOURCE = "http://www.airnowapi.org/aq/forecast/zipCode/?format=application/json"
# You'll need to get a token from airnow.gov, looks like '4d36e978-e325-11ec-c1-08002be10318'
DATA_SOURCE += "+&zipCode=\"+LOCATION+\"&API_KEY=\"+secrets[\'airnow_token\']
DATA_LOCATION = [1, "AQI"]

# the current working directory (where this file is)
cwd = ("/\"+_file_\).rsplit('/', 1)[0]
# 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,
    text_font=cwd+"/fonts/Helvetica-Bold-100.bdf",
    text_position=(90, 100),
    text_color=0x000000,
    caption_text="Air Quality Index for "+LOCATION,
    caption_font=cwd+"/fonts/HelveticaNeue-24.bdf",
    caption_position=(15, 220),
    caption_color=0x000000,)

while True:
    try:
        value = pyportal.fetch()
        print("Response is ", value)
        if 0 <= value <= 50:
            pyportal.set_background(0x66bb6a)  # good
        if 51 <= value <= 100:
            pyportal.set_background(0xffeb3b)  # moderate
        if 101 <= value <= 150:
            pyportal.set_background(0xf39c12)  # sensitive
        if 151 <= value <= 200:
            pyportal.set_background(0xff5722)  # unhealthy
        if 201 <= value <= 300:
            pyportal.set_background(0x8e24aa)  # very unhealthy
        if 301 <= value <= 500:
            pyportal.set_background(0xb71c1c )  # hazardous
How It Works

The PyPortal Air Quality Display has a few steps it takes to provide you with the information you desire!

**WiFi**

First, the program will connect to the WiFi network specified in the `secrets.py` file.

**API Query and JSON**

Then, it will send a request to the AirNow URL using your LOCATION United States Zip code specified in the code, as well as your AirNow token from the `secrets.py` file.

The query 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:

```
[
  {
    "DateIssue": "2019-03-09 ",
    "DateForecast": "2019-03-10 ",
    "ReportingArea": "NW Coastal LA",
    "StateCode": "CA",
    "Latitude": 34.0505,
    "Longitude": -118.4566,
    "ParameterName": "O3",
    "AQI": 32,
    "Category": {
      "Number": 1,
      "Description": "Good"
    }
  }
]```

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!
"Number": 1,
"Name": "Good"
},
"ActionDay": false
},
{
"DateIssue": "2019-03-09 ",
"DateForecast": "2019-03-10 ",
"ReportingArea": "NW Coastal LA",
"StateCode": "CA",
"Latitude": 34.0505,
"Longitude": -118.4566,
"ParameterName": "PM2.5",
"AQI": 33,
"Category": {
"Number": 1,
"Name": "Good"
},
"ActionDay": false
},
{
"DateIssue": "2019-03-09 ",
"DateForecast": "2019-03-10 ",
"ReportingArea": "NW Coastal LA",
"StateCode": "CA",
"Latitude": 34.0505,
"Longitude": -118.4566,
"ParameterName": "PM10",
"AQI": 7,
"Category": {
"Number": 1,
"Name": "Good"
},
"ActionDay": false
},
{
"DateIssue": "2019-03-09 ",
"DateForecast": "2019-03-10 ",
"ReportingArea": "NW Coastal LA",
"StateCode": "CA",
"Latitude": 34.0505,
"Longitude": -118.4566,
"ParameterName": "NO2",
"AQI": 22,
"Category": {
"Number": 1,
"Name": "Good"
},
"ActionDay": false
},
{
"ParameterName": "CO",
"AQI": 5,
"Category": {
  "Number": 1,
  "Name": "Good"
},
"ActionDay": false
},
{
  "DateIssue": "2019-03-09 ",
  "DateForecast": "2019-03-11 ",
  "ReportingArea": "NW Coastal LA",
  "StateCode": "CA",
  "Latitude": 34.0505,
  "Longitude": -118.4566,
  "ParameterName": "O3",
  "AQI": 29,
  "Category": {
    "Number": 1,
    "Name": "Good"
  },
  "ActionDay": false
},
{
  "DateIssue": "2019-03-09 ",
  "DateForecast": "2019-03-11 ",
  "ReportingArea": "NW Coastal LA",
  "StateCode": "CA",
  "Latitude": 34.0505,
  "Longitude": -118.4566,
  "ParameterName": "PM2.5",
  "AQI": 33,
  "Category": {
    "Number": 1,
    "Name": "Good"
  },
  "ActionDay": false
},
{
  "DateIssue": "2019-03-09 ",
  "DateForecast": "2019-03-11 ",
  "ReportingArea": "NW Coastal LA",
  "StateCode": "CA",
  "Latitude": 34.0505,
  "Longitude": -118.4566,
  "ParameterName": "PM10",
  "AQI": 9,
  "Category": {
    "Number": 1,
    "Name": "Good"
  },
  "ActionDay": false
},
{
  "DateIssue": "2019-03-09 ",
  "DateForecast": "2019-03-11 ",
  "ReportingArea": "NW Coastal LA",
  "StateCode": "CA",
  "Latitude": 34.0505,
  "Longitude": -118.4566,
You can use online code "beautifiers" such as [https://codebeautify.org/jsonviewer](https://codebeautify.org/jsonviewer) or [http://jsonviewer.stack.hu](http://jsonviewer.stack.hu) to get a nice hierarchical view of the data in the JSON file:
JSON Traversal

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 DateIssue, ReportingArea, ParameterName, and AQI, and their respective values. The key : value pair we care about for the Air Quality Display is the Air Quality Index of particle pollution (PM2.5), or AQI. This is the pair:

- "AQI": "33"

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 'AQI' in the 'PM2.5' section vs. the 'AQI' in the 'NO2' section. To avoid name clashing we rely on JSON traversal.

In the airquality.py file, you'll see how this is done. For example, the key we want for PM2.5 is found in this hierarchy of the JSON file: [1, "AQI"]

This means there is a sub-tree to the top level of the file indexed [1], which is the second section, as the first is indexed zero. Then, below that level is the 'AQI' key we want.

The value of that key in this case is '33', which is considered to be in the "good" range of particle pollution.

Background

For the background, it displays a solid color based on the current air quality index value. 0-50 is considered good, for example, and has a green background. Here is the full set:

- 0-50, "good", green
- 51-100, "moderate", yellow
- 101-150, "sensitive", orange
- 151-200 "unhealthy", red
- 210-300, "very unhealthy", purple
- 301-500, "hazardous", maroon

Font

The AQI value is displayed as text created with a bitmapped font to overlay on top of the background. The fonts used here are bitmap fonts made from the Helvetica typeface. You can learn more about converting type in this guide (https://adafru.it/E7E).

At the bottom of the screen, there is a caption text that types out "Air Quality Index for " followed by the United States Zip code you entered.
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 pair we need
- `default_bg` default background color
- details of text and caption fonts, positions, and colors

Fetch

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

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. You can do this with the `text_position` and `caption_position` options.

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`.

![Color picker](https://adafruit.it/Eb7)

Now, we'll look at mounting the PyPortal into a case for display!
PyPortal Air Quality Display 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://adafru.it/Eeq) -- 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 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 ten minutes, your display will update with the latest Air Quality Index data. Pay attention if it turns orange or red for the background color -- you may want to change your air filters and be careful outside.