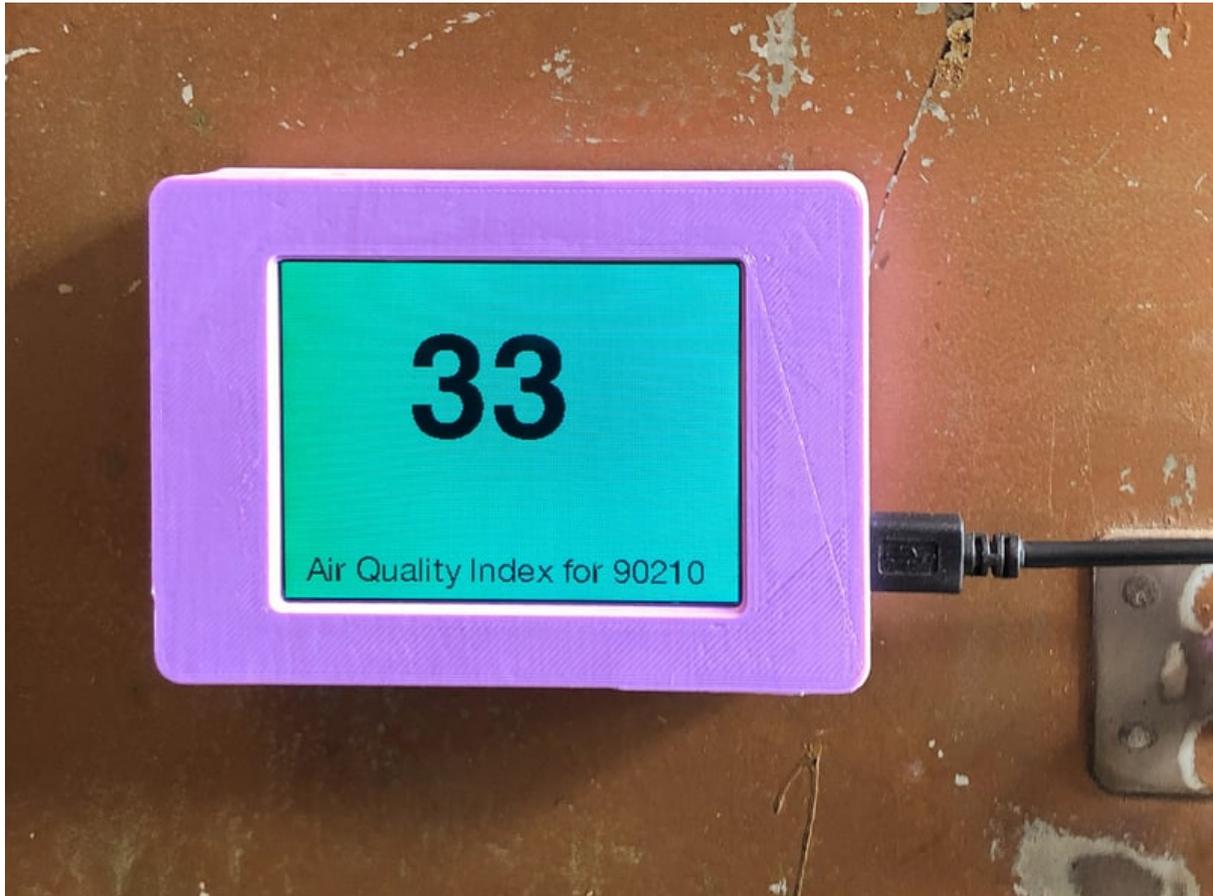




PyPortal Air Quality Display

Created by John Park



<https://learn.adafruit.com/pyportal-air-quality-display>

Last updated on 2025-02-26 02:59:04 PM EST

Table of Contents

Overview	3
<ul style="list-style-type: none">• Additional Tools & Materials	
Install CircuitPython	6
<ul style="list-style-type: none">• Set up CircuitPython Quick Start!• PyPortal Default Files	
PyPortal CircuitPython Setup	8
<ul style="list-style-type: none">• Adafruit CircuitPython Bundle	
Create Your settings.toml File	10
<ul style="list-style-type: none">• CircuitPython settings.toml File• settings.toml File Tips• Accessing Your settings.toml Information in code.py	
Internet Connect!	12
<ul style="list-style-type: none">• Connect to WiFi• Advanced Requests Usage• WiFi Manager• Further Information	
Code PyPortal with CircuitPython	21
<ul style="list-style-type: none">• AirNow API Key• Adafruit IO Time Server• CircuitPython Code• How It Works• WiFi• API Query and JSON• JSON Traversal• PyPortal Constructor• Fetch• Text Position• Text Color	
PyPortal Air Quality Display Case	30
<ul style="list-style-type: none">• Magnets• Magnets	

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 \(https://adafru.it/Eel\)](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](https://www.adafruit.com/product/4116)

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...

<https://www.adafruit.com/product/1994>



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.

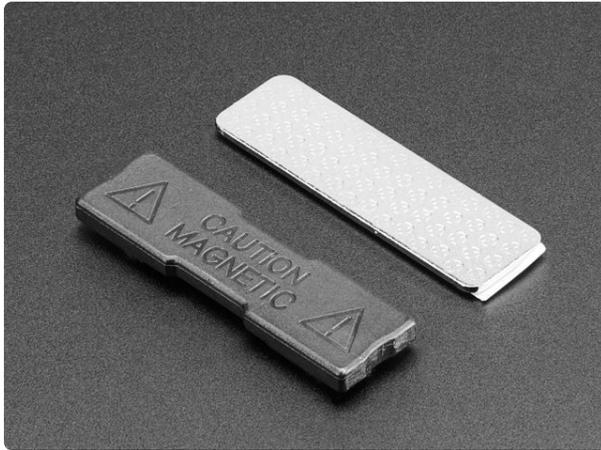
<https://www.adafruit.com/product/2185>

Additional Tools & Materials

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

- [3D printer \(http://adafru.it/2673\)](http://adafru.it/2673)
- [Filament \(http://adafru.it/3731\)](http://adafru.it/3731)
- [PyPortal Case model files \(https://adafru.it/Eem\)](https://adafru.it/Eem)
- [M2.5 screws \(http://adafru.it/3299\)](http://adafru.it/3299)

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

Totalling 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](https://adafru.it/tB7) (<https://adafru.it/tB7>) is a derivative of [MicroPython](https://adafru.it/BeZ) (<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 :)

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

<https://adafru.it/Egk>

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

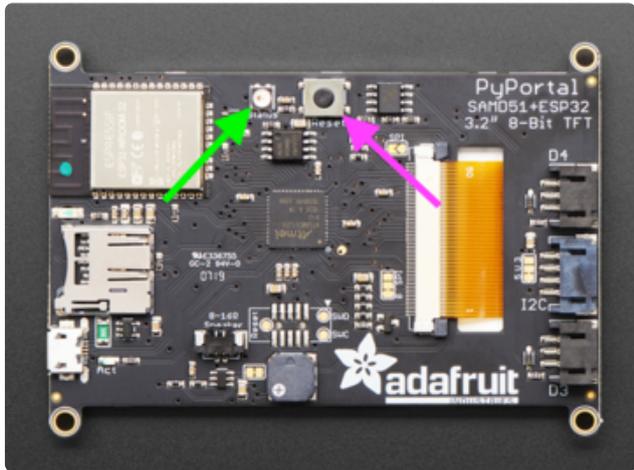
<https://adafru.it/HFd>



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).

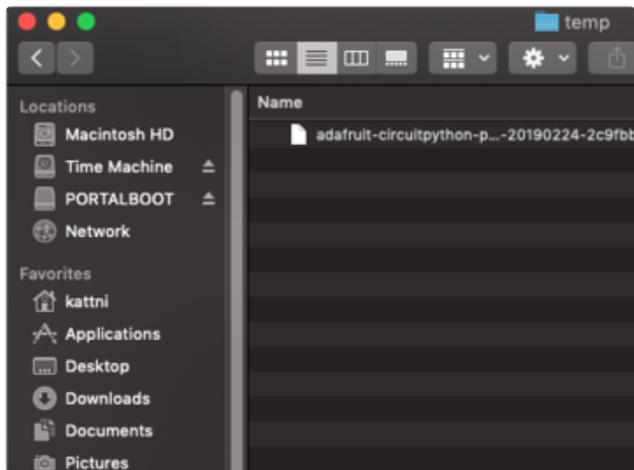
Plug your PyPortal into your computer using a known-good USB cable.



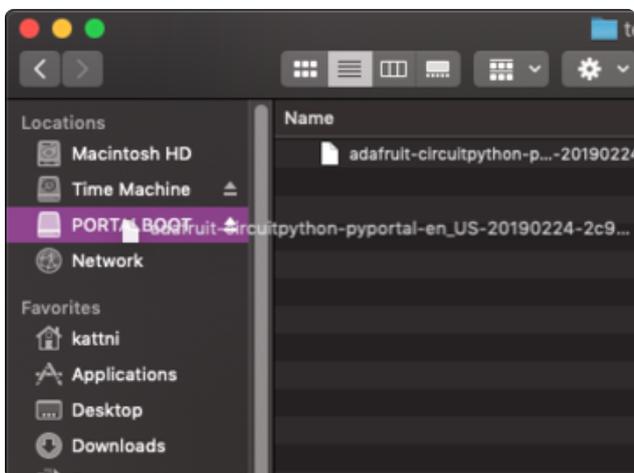
A lot of people end up using charge-only USB cables and it is very frustrating! So make sure you have a USB cable you know is good for data sync.

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!

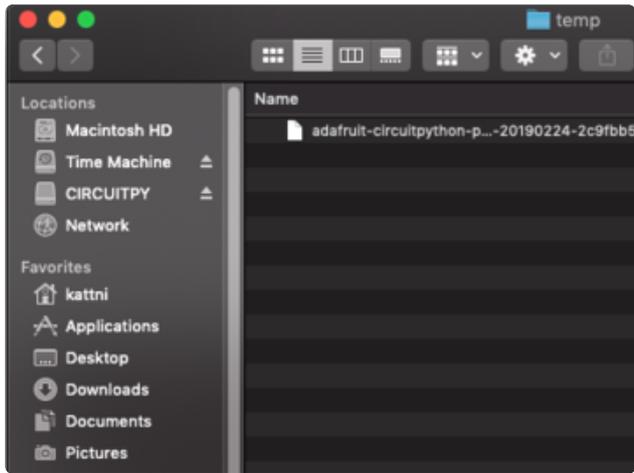
If double-clicking doesn't work the first time, try again. Sometimes it can take a few tries to get the rhythm right!



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.

PyPortal Default Files

<https://adafru.it/UF->

PyPortal Pynt Default Files

<https://adafru.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:

Latest Adafruit CircuitPython
Library Bundle

<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_connection_manager** - used by **adafruit_requests**.
- **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.

Create Your settings.toml File

CircuitPython works with WiFi-capable boards to enable you to make projects that have network connectivity. This means working with various passwords and API keys. As of [CircuitPython 8 \(https://adafru.it/Em8\)](https://adafru.it/Em8), there is support for a **settings.toml** file. This is a file that is stored on your **CIRCUITPY** drive, that contains all of your secret network information, such as your SSID, SSID password and any API keys for IoT services. It is designed to separate your sensitive information from your **code.py** file so you are able to share your code without sharing your credentials.

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

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

CircuitPython settings.toml File

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

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

- `your_wifi_ssid`
- `your_wifi_password`

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

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

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

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

Some projects use different variable names for the entries in the `settings.toml` file. For example, a project might use `ADAFRUIT_AIO_ID` in the place of `ADAFRUIT_AIO_USERNAME`. If you run into connectivity issues, one of the first things to check is that the names in the `settings.toml` file match the names in the code.

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

settings.toml File Tips

Here is an example `settings.toml` file.

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

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

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



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

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

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

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

```
CircuitPython REPL
code.py output:
this is a test

Code done running.

Press any key to enter the REPL. Use CTRL-D to reload.
```

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

Internet Connect!

Connect to WiFi

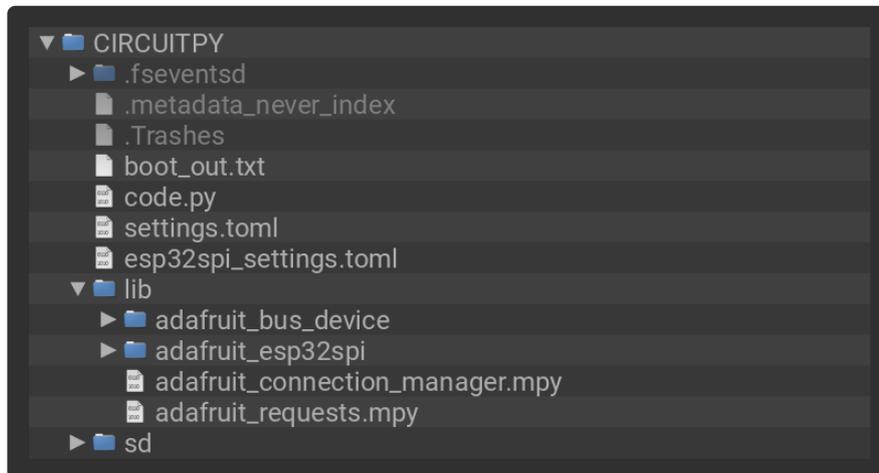
OK, now that you have your **settings.toml** file set up - you can connect to the Internet.

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

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

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

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



Update to CircuitPython 9.2.x or later to use this example.

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

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

# Get wifi details and more from a settings.toml file
# tokens used by this Demo: CIRCUITPY_WIFI_SSID, CIRCUITPY_WIFI_PASSWORD
ssid = getenv("CIRCUITPY_WIFI_SSID")
password = getenv("CIRCUITPY_WIFI_PASSWORD")

print("ESP32 SPI webclient test")

TEXT_URL = "http://wifitest.adafruit.com/testwifi/index.html"
JSON_URL = "http://wifitest.adafruit.com/testwifi/sample.json"

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

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

# If you have an AirLift Featherwing or ItsyBitsy AirLift:
```

```

# esp32_cs = DigitalInOut(board.D13)
# esp32_ready = DigitalInOut(board.D11)
# esp32_reset = DigitalInOut(board.D12)

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

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

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

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

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

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

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

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

print("Done!")

```

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

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

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

```
&gt;&gt;&gt; import wifitest
ESP32 SPI webclient test
ESP32 found and in idle mode
Firmware vers. 1.7.5
MAC addr: 24:C9:DC:BD:0F:3F
  HomeNetwork      RSSI: -46
  HomeNetwork      RSSI: -76
  Fios-12345        RSSI: -92
  FiOS-AB123        RSSI: -92
  NETGEAR53         RSSI: -93
Connecting to AP...
Connected to HomeNetwork  RSSI: -45
My IP address is 192.168.1.245
IP lookup adafruit.com: 104.20.39.240
Ping google.com: 30 ms
Fetching text from http://wifitest.adafruit.com/testwifi/index.html
-----
This is a test of Adafruit WiFi!
If you can read this, its working :)
-----

Fetching json from http://wifitest.adafruit.com/testwifi/sample.json
-----
{'fun': True, 'company': 'Adafruit', 'founded': 2005, 'primes': [2, 3, 5], 'pi':
3.14, 'mixed': [False, None, 3, True, 2.7, 'cheese']}
-----
Done!
```

Going over the example above, here's a breakdown of what the program is doing:

- Initialize the ESP32 over SPI using the SPI port and 3 control pins:

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

#...

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

- Get the socket pool and the SSL context, and then tell the `adafruit_requests` library about them.

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

- Verify 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:", ":".join("%02X" % byte for byte in esp.MAC_address))

```

- Perform a scan of all access points it can see and print out the name and signal strength.

```

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

```

- Connect to the AP we've defined here, then print out the local IP address. Then attempt 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(ssid, password)
    except OSError as e:
        print("could not connect to AP, retrying: ", e)
        continue
print("Connected to", esp.ap_info.ssid, "\tRSSI:", esp.ap_info.rssi)
print("My IP address is", esp.ipv4_address)
print(
    "IP lookup adafruit.com: %s" %
    esp.pretty_ip(esp.get_host_by_name("adafruit.com"))
)

```

Now we're getting to the really interesting part of the example program. We've written a library for web fetching web data, named [adafruit_requests](https://adafru.it/FpW) (<https://adafru.it/FpW>). It is a lot like the regular Python library named [requests](https://adafru.it/1af4) (<https://adafru.it/1af4>). This library allows you to send HTTP and HTTPS requests easily and provides helpful methods for parsing the response from the server.

- Here is the part of the example program is fetching text data from a URL.

```

TEXT_URL = "http://wifitest.adafruit.com/testwifi/index.html" # Further up in the
program

# ...

print("Fetching text from", TEXT_URL)
r = requests.get(TEXT_URL)
print('-' * 40)
print(r.text)

```

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

- Finally, here the program is fetching some JSON data. The `adafruit_requests` library will parse the JSON into a Python dictionary whose structure is the same as the structure of the JSON.

```
JSON_URL = "http://wifitest.adafruit.com/testwifi/sample.json" # Further up in
the program

# ...

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

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

import os

import adafruit_connection_manager
import board
import busio
from adafruit_esp32spi import adafruit_esp32spi
from digitalio import DigitalInOut

import adafruit_requests

# Get WiFi details, ensure these are setup in settings.toml
ssid = os.getenv("CIRCUITPY_WIFI_SSID")
password = os.getenv("CIRCUITPY_WIFI_PASSWORD")

# 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)
radio = adafruit_esp32spi.ESP_SPIcontrol(spi, esp32_cs, esp32_ready, esp32_reset)

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

# Initialize a requests session
pool = adafruit_connection_manager.get_radio_socketpool(radio)
ssl_context = adafruit_connection_manager.get_radio_ssl_context(radio)
requests = adafruit_requests.Session(pool, ssl_context)

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

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

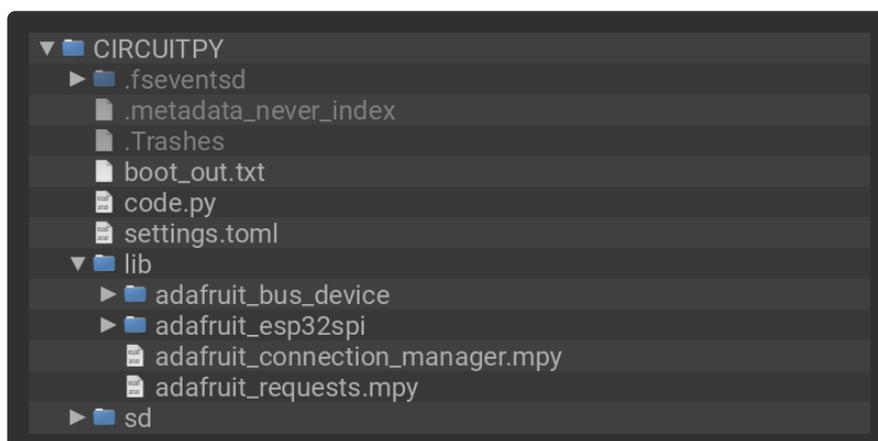
print("Fetching JSON data from %s..." % JSON_GET_URL)
with requests.get(JSON_GET_URL, headers=headers) as response:
    print("-" * 60)

    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)

```

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



WiFi Manager

The way the examples above connect to WiFi works but it's a little finicky. Since WiFi is not necessarily so reliable, you may have disconnects and need to reconnect. For more advanced uses, we recommend using the `WiFiManager` class. 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 using the `WiFiManager` and also how to fetch the current time from a web source.

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

import time
from os import getenv
import board
import busio
from digitalio import DigitalInOut
import neopixel
import rtc
from adafruit_esp32spi import adafruit_esp32spi
from adafruit_esp32spi.adafruit_esp32spi_wifimanager import WiFiManager

# Get wifi details and more from a settings.toml file
# tokens used by this Demo: CIRCUITPY_WIFI_SSID, CIRCUITPY_WIFI_PASSWORD
ssid = getenv("CIRCUITPY_WIFI_SSID")
password = getenv("CIRCUITPY_WIFI_PASSWORD")

print("ESP32 local time")

TIME_API = "http://worldtimeapi.org/api/ip"

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

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

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

"""Use below for Most Boards"""
status_pixel = neopixel.NeoPixel(board.NEOPIXEL, 1, brightness=0.2)
"""Uncomment below for ItsyBitsy M4"""
# status_pixel = dotstar.DotStar(board.APA102_SCK, board.APA102_MOSI, 1,
# brightness=0.2)
"""Uncomment below for an externally defined RGB LED (including Arduino Nano
Connect)"""
# import adafruit_rgbled
# from adafruit_esp32spi import PWMOut
# RED_LED = PWMOut.PWMOut(esp, 26)
# GREEN_LED = PWMOut.PWMOut(esp, 27)
# BLUE_LED = PWMOut.PWMOut(esp, 25)
```

```

# status_pixel = adafruit_rgbled.RGBLED(RED_LED, BLUE_LED, GREEN_LED)

wifi = WiFiManager(esp, ssid, password, status_pixel=status_pixel)

the_rtc = rtc.RTC()

response = None
while True:
    try:
        print("Fetching json from", TIME_API)
        response = wifi.get(TIME_API)
        break
    except OSError as e:
        print("Failed to get data, retrying\n", e)
        continue

json = response.json()
current_time = json["datetime"]
the_date, the_time = current_time.split("T")
year, month, mday = [int(x) for x in the_date.split("-")]
the_time = the_time.split(".")[0]
hours, minutes, seconds = [int(x) for x in the_time.split(":")]

# We can also fill in these extra nice things
year_day = json["day_of_year"]
week_day = json["day_of_week"]
is_dst = json["dst"]

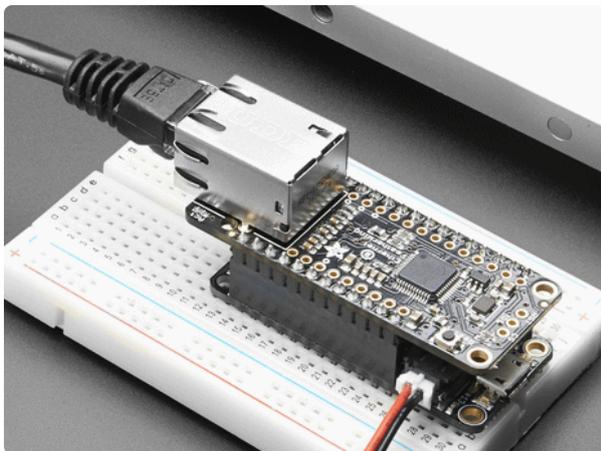
now = time.struct_time(
    (year, month, mday, hours, minutes, seconds, week_day, year_day, is_dst)
)
print(now)
the_rtc.datetime = now

while True:
    print(time.localtime())
    time.sleep(1)

```

Further Information

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



[Networking in CircuitPython](https://learn.adafruit.com/networking-in-circuitpython)

By Anne Barela

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

Code PyPortal with CircuitPython

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) (<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 **settings.toml** file that is on the root level of your **CIRCUITPY** drive, so it looks something like this:

```
CIRCUITPY_WIFI_SSID = "your_wifi_ssid"
CIRCUITPY_WIFI_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) (<https://adafru.it/dAQ>).

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

Once you have logged into your account, there are two pieces of information you'll need to place in your **settings.toml** 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 **settings.toml** file like this:

```
CIRCUITPY_WIFI_SSID = "your_wifi_ssid"
CIRCUITPY_WIFI_PASSWORD = "your_wifi_password"
AIRNOW_TOKEN = "XXX0XXXX-0000-0000-0000-XXXXXXXX0000"
```

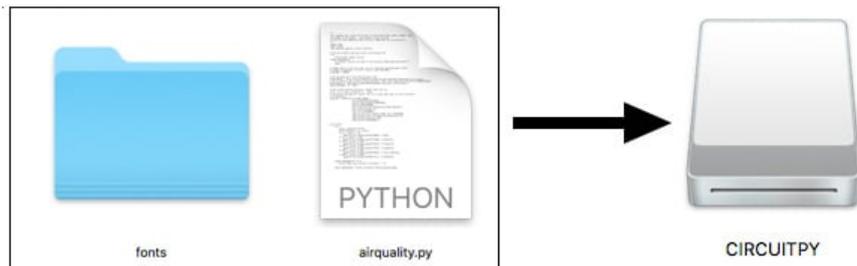
```
AIO_USERNAME = "your_aio_username"  
AIO_KEY = "your_aio_key"
```

CircuitPython Code

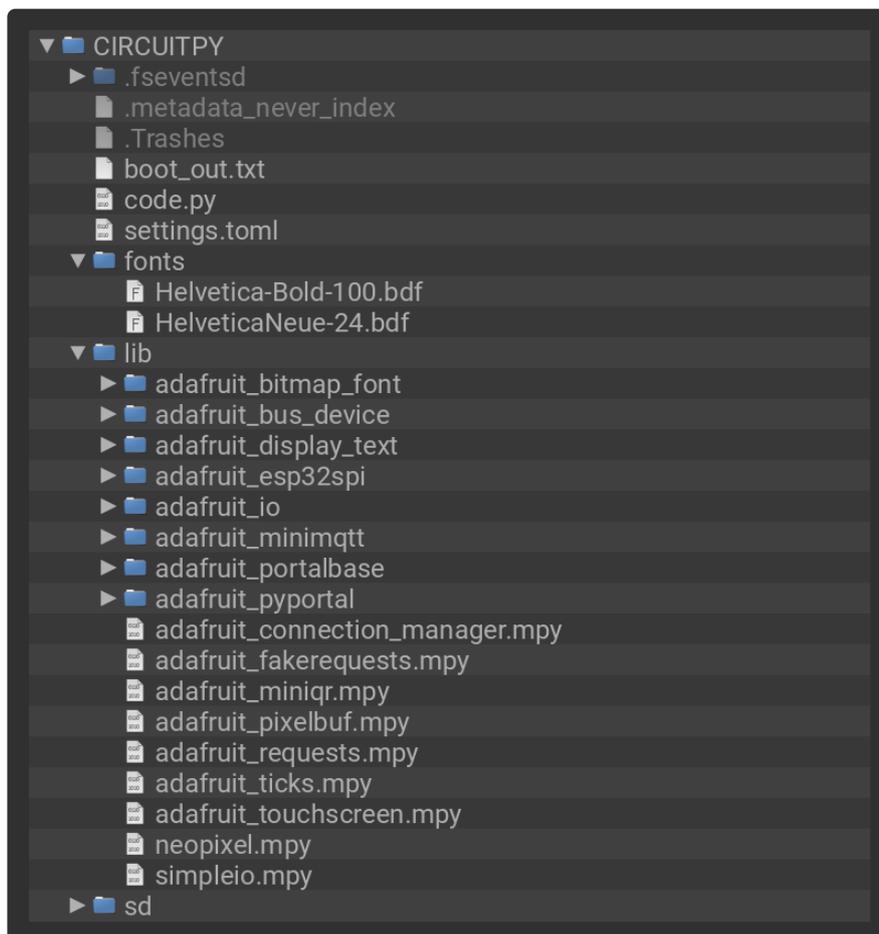
In the embedded code element below, click on the **Download Project Bundle** button, and save the .zip archive file to your computer.

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 **CIRCUITPY** drive.



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



```

# SPDX-FileCopyrightText: 2019 Limor Fried for Adafruit Industries
#
# SPDX-License-Identifier: MIT

"""
This example will access the twitter follow button API, grab a number like
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 os
import time
import board
from adafruit_pyportal import PyPortal

# 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-
e325-11cecl-08002be10318'
DATA_SOURCE += "&zipCode="+LOCATION+"&API_KEY="+os.getenv("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

    except RuntimeError as e:
        print("Some error occured, retrying! -", e)

    time.sleep(10*60) # wait 10 minutes before getting 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 Air Quality Display has a few steps it takes to provide you with the information you desire!

We'll be displaying the Air Quality Index for Particle Pollution (PM) for fine particles, 2.5 micrometers in diameter or smaller. PM2.5 particles come from all kinds of combustion.

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:

```
http://www.airnowapi.org/aq/forecast/zipCode/?format=application/
json&zipCode=90210&API_KEY=XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX
```

(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": "03",
    "AQI": 32,
    "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": "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
},
{
  "DateIssue": "2019-03-09 ",
  "DateForecast": "2019-03-10 ",
  "ReportingArea": "NW Coastal LA",
  "StateCode": "CA",
  "Latitude": 34.0505,
  "Longitude": -118.4566,
  "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,
    "ParameterName": "NO2",
    "AQI": 18,
    "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": "CO",
    "AQI": 6,
    "Category": {
      "Number": 1,
      "Name": "Good"
    },
    "ActionDay": false
  }
}
]

```

You can use online code "beautifiers" such as <https://codebeautify.org/jsonviewer> (<https://adafru.it/Eb5>) or <http://jsonviewer.stack.hu> (<https://adafru.it/Eb6>) to get a nice hierarchical view of the data in the JSON file:

```
▼ array [10]
  ► 0 {10}
  ▼ 1 {10}
    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 {2}
    Number : 1
    Name : Good
    ActionDay : false
```

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 \(https://adafru.it/E7E\)](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

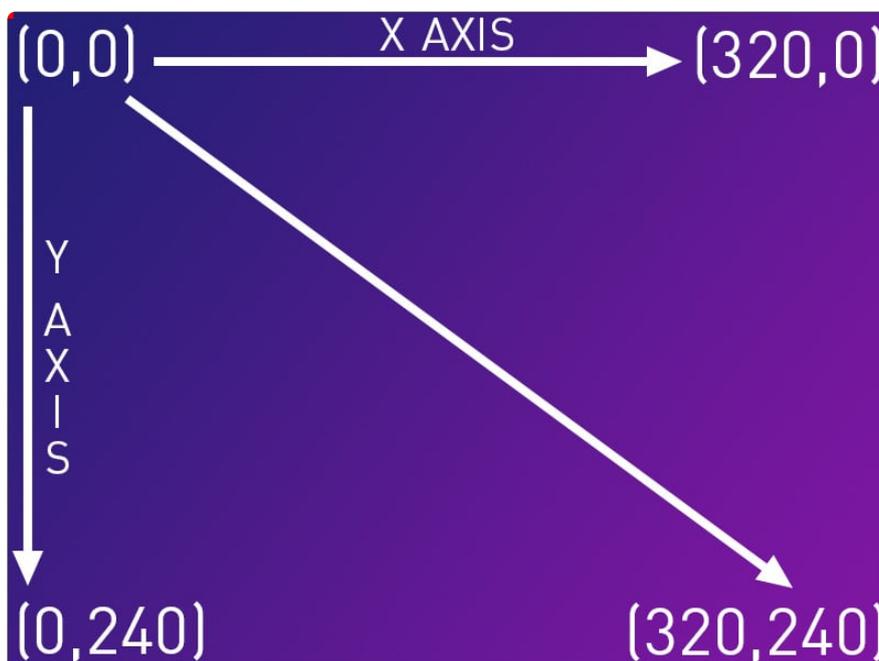
With the PyPortal set up, we can then use `pyportal.fetch()` to do the query and parsing of the AirNow data and then display it on screen.

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).



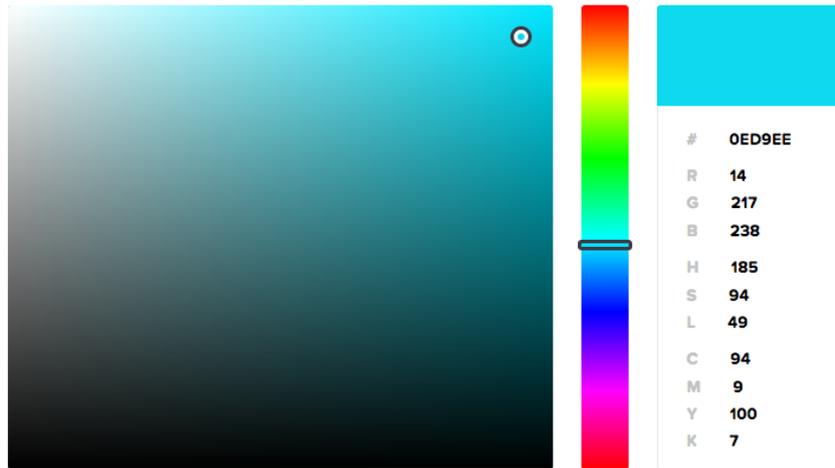
So, if you wanted to move the subscriber count text to the right and up closer to the top, your code may look like this for that part of the pyportal constructor:

```
text_position=(250, 10)
```

Text Color

Another way to customize your display is to adjust the color of the text. The line `text_color=0xFFFFFFFF` 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://adafru.it/Eb7>) 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!

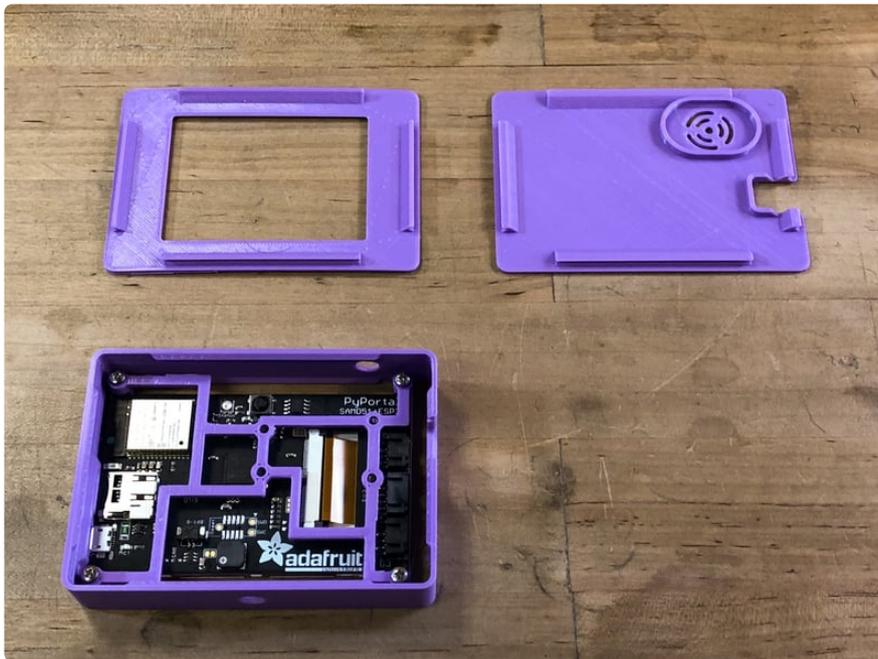
PyPortal Air Quality Display Case



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 \(https://adafru.it/Eeq\)](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.