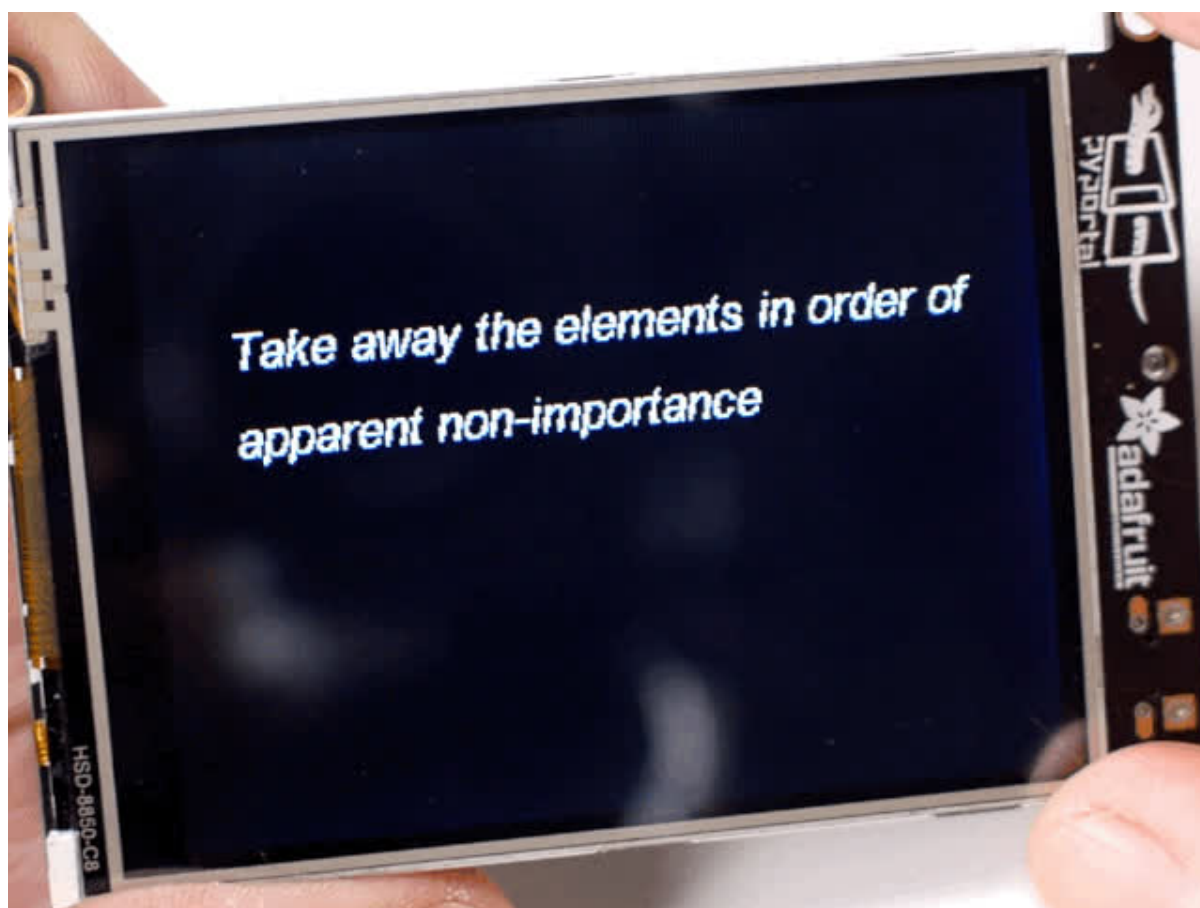




# PyPortal Oblique Strategies

Created by Collin Cunningham



<https://learn.adafruit.com/pyportal-oblique-strategies>

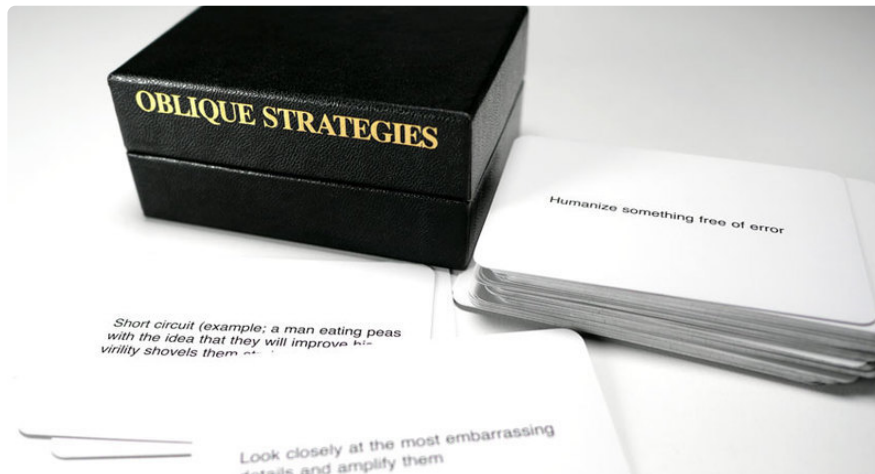
Last updated on 2024-11-18 12:58:22 PM EST

# Table of Contents

Overview	3
<hr/>	
• Parts	
Install CircuitPython	4
<hr/>	
• Set up CircuitPython Quick Start!	
• PyPortal Default Files	
Create Your settings.toml File	7
<hr/>	
• CircuitPython settings.toml File	
• settings.toml File Tips	
• Accessing Your settings.toml Information in code.py	
Internet Connect!	9
<hr/>	
• Connect to WiFi	
• Requests	
• HTTP GET with Requests	
• HTTP POST with Requests	
• Advanced Requests Usage	
• WiFi Manager	
Project Files	22
<hr/>	
• How it works	
• Import strategies text	
• PyPortal object	
• Startup Text	
• Main loop	
• Debounce	
Usage	25
<hr/>	
• Customize it	

---

# Overview



[Oblique Strategies](https://adafru.it/Eo2) (<https://adafru.it/Eo2>) is a deck of cards created by a composer (**Brian Eno**) and a painter (**Peter Schmidt**), originally released in 1975. Each card contains text intended to aid in the creative process through question, instruction, or simply presenting a concept. You can purchase a physical deck of Oblique Strategies from [Enoshop here](https://adafru.it/Eo3) (<https://adafru.it/Eo3>).



This project uses the **Pyportal** to create an electronic deck of Oblique Strategies you can keep at the ready on your desktop. You'll be able view a **random strategy** by simply **pressing** the Pyportal's **touchscreen**.

## Parts



### [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>



### [USB cable - USB A to Micro-B](https://www.adafruit.com/product/592)

This here is your standard A to micro-B USB cable, for USB 1.1 or 2.0. Perfect for connecting a PC to your Metro, Feather, Raspberry Pi or other dev-board or...

<https://www.adafruit.com/product/592>

---

## 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)

<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.

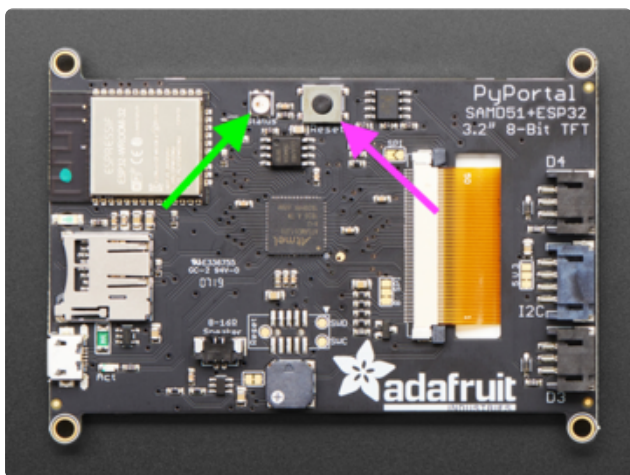
Download and save it to your desktop (or wherever is handy).

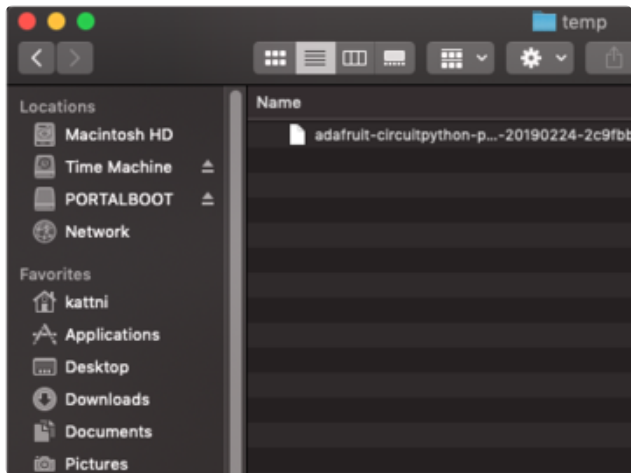
Plug your PyPortal into your computer using a known-good USB cable.

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.

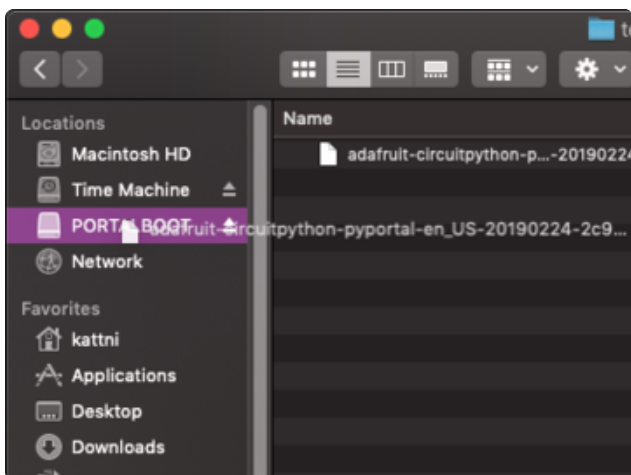
Double-click the **Reset** button on the top in the middle (magenta arrow) on your board, and you will see the NeoPixel RGB LED (green arrow) turn green. If it turns red, check the USB cable, try another USB port, etc. **Note:** The little red LED next to the USB connector will pulse red. That's ok!

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

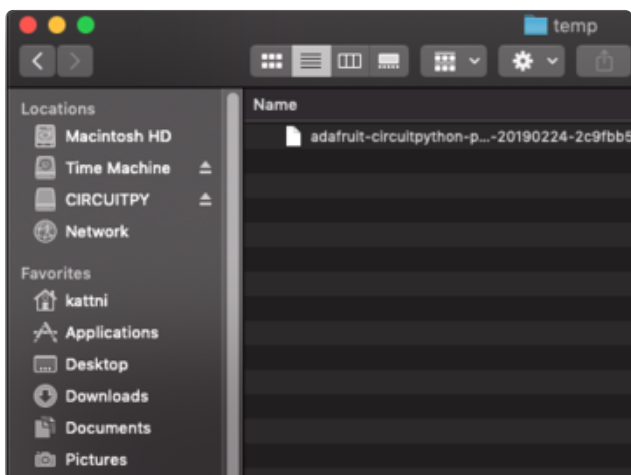




You will see a new disk drive appear called **PORTALBOOT**.



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->

## 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:



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

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

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

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

```

print("ESP32 SPI webclient test")

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

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

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

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

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

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

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

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

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

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

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

```

```

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

print("Done!")

```

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

```

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

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

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

print("ESP32 SPI webclient test")

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

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

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

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

# If you have an externally connected ESP32:

```

```

# NOTE: You may need to change the pins to reflect your wiring
# esp32_cs = DigitalInOut(board.D9)
# esp32_ready = DigitalInOut(board.D10)
# esp32_reset = DigitalInOut(board.D5)

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

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

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

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

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

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

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

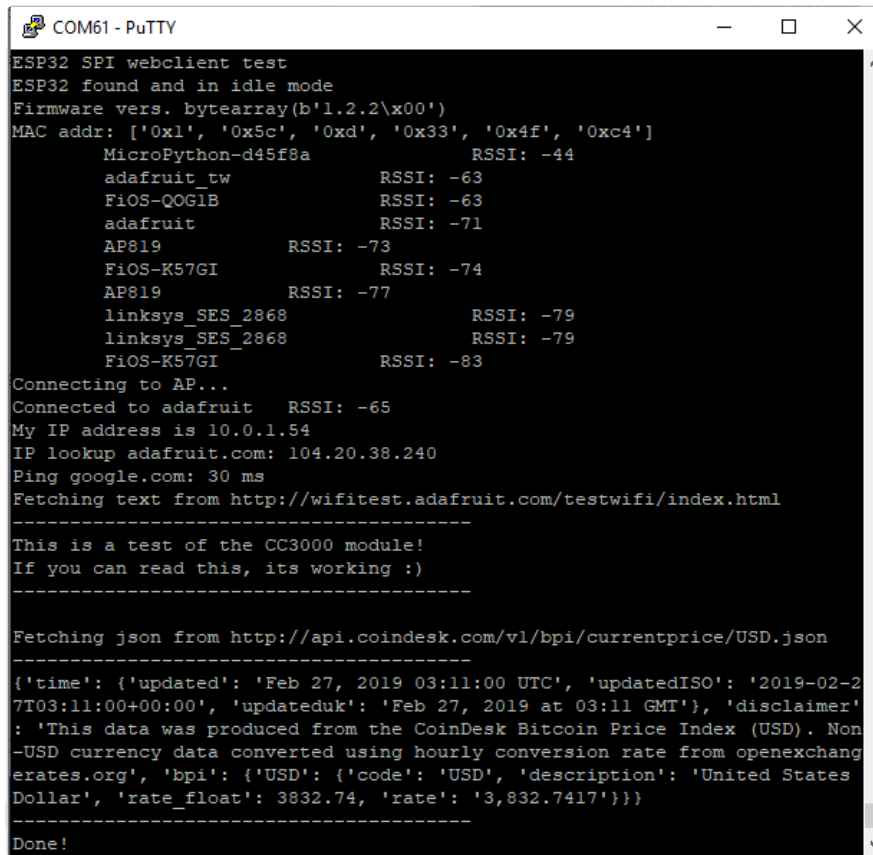
print("Done!")

```

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

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

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



```
COM61 - PuTTY
ESP32 SPI webclient test
ESP32 found and in idle mode
Firmware vers. bytearray(b'1.2.2\x00')
MAC addr: ['0x1', '0x5c', '0xd', '0x33', '0x4f', '0xc4']
MicroPython-d45f8a      RSSI: -44
adafruit tw             RSSI: -63
FIOS-QOGLB              RSSI: -63
adafruit                RSSI: -71
AP819                   RSSI: -73
FIOS-K57GI              RSSI: -74
AP819                   RSSI: -77
linksys_SES_2868        RSSI: -79
linksys_SES_2868        RSSI: -79
FIOS-K57GI              RSSI: -83
Connecting to AP...
Connected to adafruit    RSSI: -65
My IP address is 10.0.1.54
IP lookup adafruit.com: 104.20.38.240
Ping google.com: 30 ms
Fetching text from http://wifitest.adafruit.com/testwifi/index.html
-----
This is a test of the CC3000 module!
If you can read this, its working :)
-----
Fetching json from http://api.coindesk.com/v1/bpi/currentprice/USD.json
-----
{'time': {'updated': 'Feb 27, 2019 03:11:00 UTC', 'updatedISO': '2019-02-27T03:11:00+00:00', 'updateduk': 'Feb 27, 2019 at 03:11 GMT'}, 'disclaimer': 'This data was produced from the CoinDesk Bitcoin Price Index (USD). Non-USD currency data converted using hourly conversion rate from openexchange.org', 'bpi': {'USD': {'code': 'USD', 'description': 'United States Dollar', 'rate_float': 3832.74, 'rate': '3,832.7417'}}}
-----
Done!
```

In order, the example code...

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

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

#...

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

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

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

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

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

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

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

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

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

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

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

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

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

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



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

# Requests

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

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

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

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

CIRCUITPY

Temporarily unable to load content:

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

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

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

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

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

print("Connecting to AP...")
while not esp.is_connected:
    try:
        esp.connect_AP(b'MY_SSID_NAME', b'MY_SSID_PASSWORD')
```

```

except RuntimeError as e:
    print("could not connect to AP, retrying: ",e)
    continue
print("Connected to", str(esp.ssid, 'utf-8'), "\tRSSI:", esp.rssi)

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

```

## HTTP GET with Requests

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

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

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

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

```

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

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

```

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

**CircuitPython\_Requests** can convert a JSON-formatted response from a server into a CPython `dict` object.

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

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

```

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

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

```

## HTTP POST with Requests

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

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

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

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

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

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

## Advanced Requests Usage

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

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

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

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

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

CIRCUITPY

Temporarily unable to load content:

## WiFi Manager

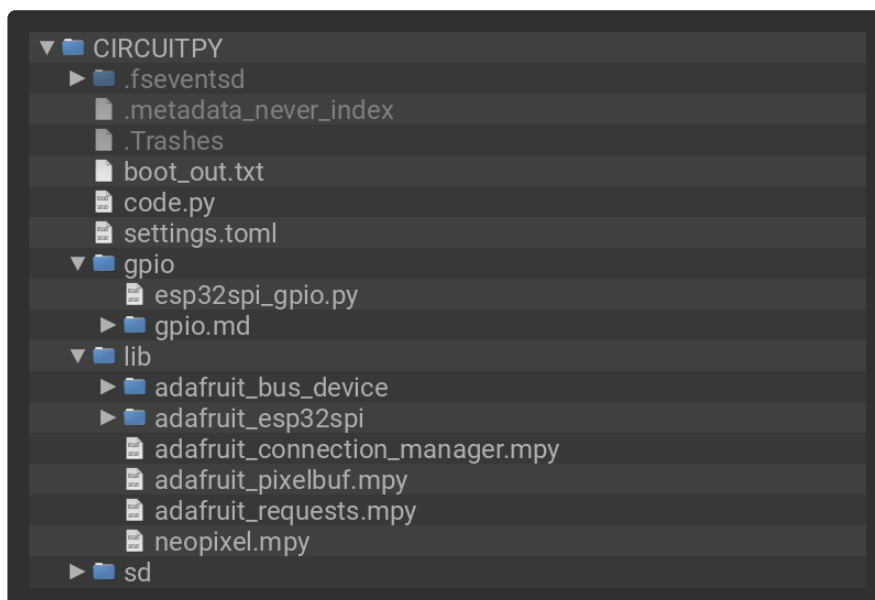
That simplest example works but it's a little finicky - you need to constantly check WiFi status and have many loops to manage connections and disconnections. For more advanced uses, we recommend using the WiFiManager object. It will wrap the connection/status/requests loop for you - reconnecting if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

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

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

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



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

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

```

import neopixel
from adafruit_esp32spi import adafruit_esp32spi
from adafruit_esp32spi import adafruit_esp32spi_wifimanager

print("ESP32 SPI webclient test")

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

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

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

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

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

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

counter = 0

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

```

```

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

```

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

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

- `aio_username`
- `aio_key`

You can go to your adafruit.io View AIO Key link to get those two values and add them to the secrets file, which will now look something like this:

```

# This file is where you keep secret settings, passwords, and tokens!
# If you put them in the code you risk committing that info or sharing it

secrets = {
    'ssid' : '_your_ssid_',
    'password' : '_your_wifi_password_',
    'timezone' : "America/Los_Angeles", # http://worldtimeapi.org/timezones
    'aio_username' : '_your_aio_username_',
    'aio_key' : '_your_aio_key_',
}

```

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

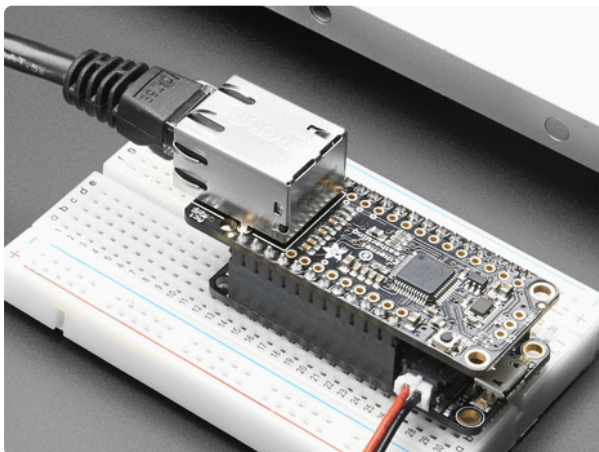
- If you do not know how to set up a feed, [follow this page and come back when you've set up a feed named `test`](https://adafru.it/f5k). (<https://adafru.it/f5k>)

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

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



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



## Networking in CircuitPython

By Anne Barela

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

---

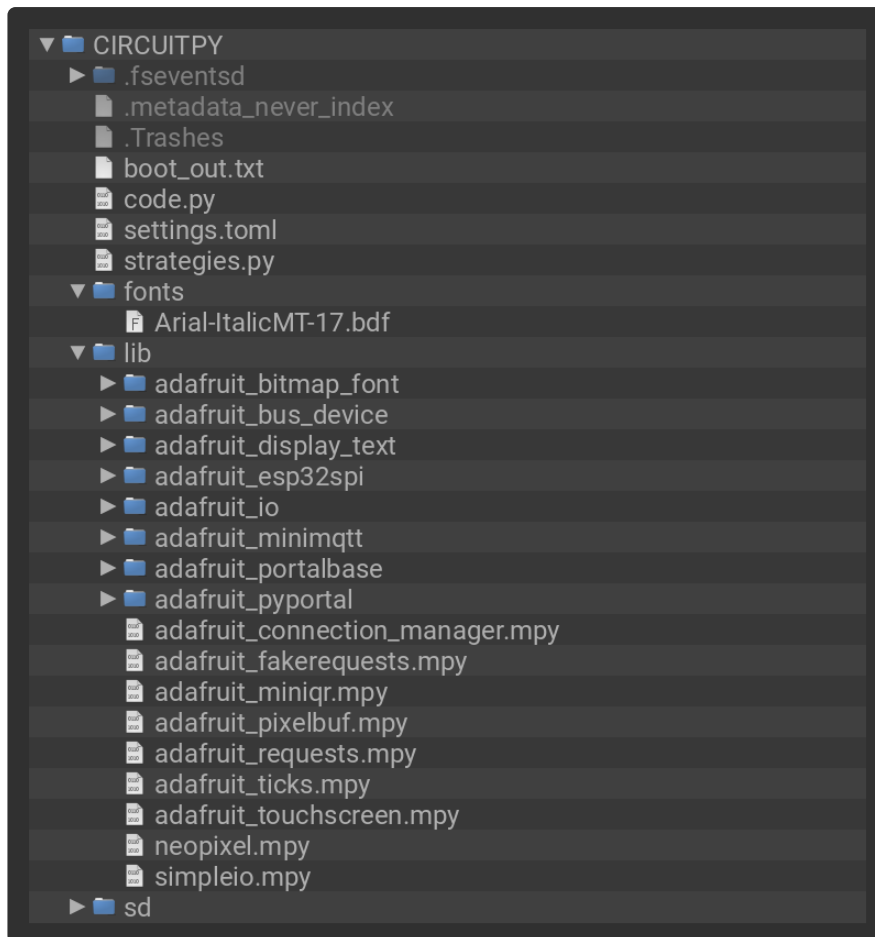
## Project Files

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\_ObliqueStrategies**.

Connect your PyPortal to your computer via a known good USB data+power cable. It will show up in your file system as a drive named **CIRCUITPY**. Copy the contents of the **PyPortal\_ObliqueStrategies** directory to your PyPortal **CIRCUITPY** drive.





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

"""
This code will display a random strategy from strategies.py when the
PyPortal screen is pressed. See the original Oblique Strategies
by Brian Eno & Peter Schmidt here: https://www.enoshop.co.uk/product/oblique-strategies
"""
import random
import board
from strategies import strategies
from adafruit_pyportal import PyPortal

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

# create pyportal object w no data source (we'll feed it text later)
pyportal = PyPortal(url = None,
                    json_path = None,
                    status_neopixel = board.NEOPIXEL,
                    default_bg = None,
                    text_font = cwd+"fonts/Arial-ItalicMT-17.bdf",
                    text_position = (30, 120),
                    text_color = 0xFFFFFF,
                    )

pyportal.set_text("loading ...") # display while user waits
pyportal.preload_font() # speed things up by preloading font
pyportal.set_text("OBLIQUE STRATEGIES\nBrian Eno / Peter Schmidt") # show title

while True:
```

```

if pyportal.touchscreen.touch_point:
    # get random string from array and wrap w line breaks
    strat = pyportal.wrap_nicely(random.choice(strategies), 35)
    outstring = '\n'.join(strat)
    # display new text
    pyportal.set_text(outstring, 0)
    # don't repeat until a new touch begins
    while pyportal.touchscreen.touch_point:
        continue

```

## How it works

The **code.py** file is relatively simple, performing a few basic operations ...

## Import strategies text

```
from strategies import strategies
```

All of the strategy text that will be displayed is stored in **strategies.py**, so we'll import it all as an array of strings at the start.

## PyPortal object

```
pyportal = PyPortal
```

The pyportal object is created with **font path** and **text formatting** info, but **no URL** or **json path** - we omit these because we'll be **setting** the display text within the **main loop**.

## Startup Text

```
pyportal.set_text("loading ...")
```

Next, we display **loading ...** on the PyPortal and begin **preloading the font** so all characters can be displayed quickly from now on. Once preloading is finished, **title & author text** is displayed.

## Main loop

```
while True:
```

In the **repeating** main loop, the following actions are performed:

1. Check to see if the screen was **touched**
2. If screen was touched, get a **random string** from the **strategies array** and **format** it so it wraps properly on the display.
3. Display the text on PyPortal

## Debounce

Lastly, you'll see these two lines at the bottom:

```
while pyportal.touchscreen.touch_point:  
    continue
```

This ensures that the code loads **only one new strategy** when the user touches the screen - even if they keep their finger held down.

---

## Usage



When starting up, you'll first see a **loading ...** screen while PyPortal boots and preloads the font. Once finished, you'll see the title screen. **Press** anywhere on the resistive touchscreen to display a **random strategy**. Repeat until satisfied.

## Customize it

If you'd like to **add your own** strategies, or **repurpose** your PyPortal as a random **recipe/quote/joke/affirmation** viewer, you can easily do so by editing one file.

```
strategies = [  
    "Abandon normal instruments",  
    "Accept advice",  
    "Accretion",  
    "THIS RIGHT HERE IS A NEW THING",  
    "A line has two sides",  
    "Allow an easement (an easement is the abandonment of a stricture)",  
    "Are there sections? Consider transitions",  
    "Ask people to work against their better judgement",  
    "Ask your body",  
    "Assemble some of the instruments in a group and treat the group",  
    "Balance the consistency principle with the inconsistency principle"
```

Open **strategies.py** in your favorite text editor and add new entries to the **strategies array**. Just remember - each entry must be surrounded by **quotes** and followed by a **comma**.

The entire array **must be preceded** by **strategies = [** and **followed** by **]**.

When you're done, **save the file** and the PyPortal will **reboot**. Your new text entries should be displayed at **random** on the PyPortal.