PyPortal 2FA TOTP Authentication Friend

Created by Brent Rubell

https://learn.adafruit.com/pyportal-2fa-totp-authentication-friend

Last updated on 2023-08-29 04:04:28 PM EDT

Table of Contents

Overview

What is TOTP?

Having 2 Factor Authentication on all your accounts is a good way to keep your data more secure. With 2FA logins, not only is a username and password needed, but also a one-time-use code. There are a few different ways to get that code, such as by email, phone or SMS. But my favorite way is to do it is via a 'Google Authenticator' time-based OTP (one time password), also known as a TOTP.

Using an app on your phone like Authy () or Authenticator, you set up a secret code given to you by the service, then every 30 seconds, a new code is generated for you. What's extra nice is that the Google Authenticator protocol is supported by just about every service and phone/tablet.

So What's The Problem?

I could use my phone, but it's not always at my desk. It also may be charging or dead. Or maybe someone doesnt own a phone?

A Solution!

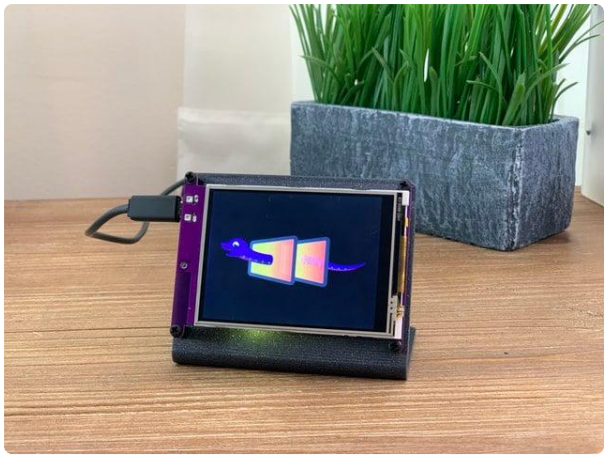
Luckily for us, the Google Authenticator protocol is really simple - You just need to be able to know the current time, and run a SHA1 hash.

Ladyada previously built a device which uses a Feather ESP8266 to display her TOTP codes. ()

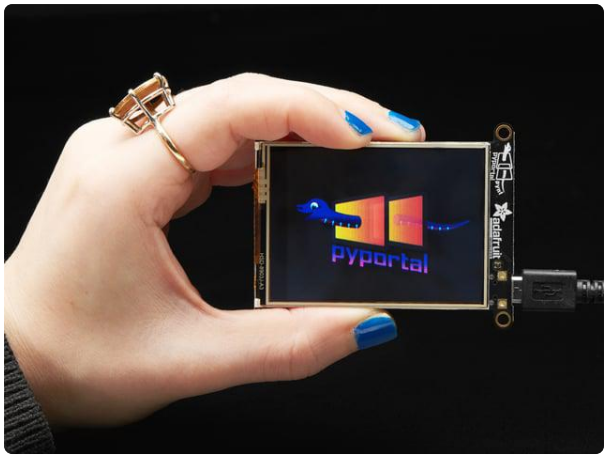
This guide is a version of this device which uses the PyPortal. The PyPortal has WiFi so it can connect to NTP to get the current time at startup and a full-color touchscreen display.

The code supports displaying up to 5 keys which you can select by tapping the buttons on the PyPortal's display.

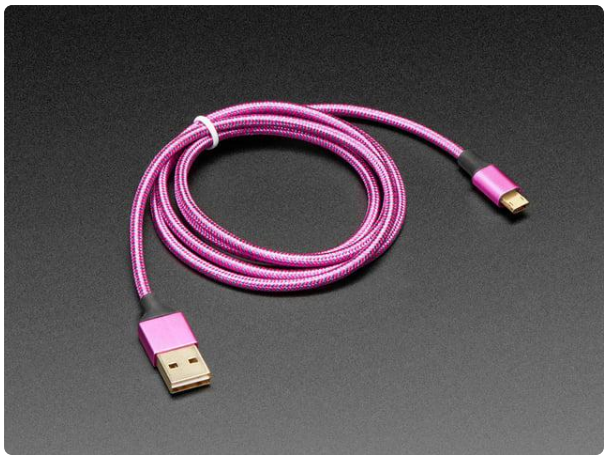
Parts



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



Adafruit PyPortal Pynt - CircuitPython Powered Internet Display
The PyPortal Pynt is the little sister to our popular PyPortal - zapped with a shrink ray to take the design...
https://www.adafruit.com/product/4465



Fully Reversible Pink/Purple USB A to micro B Cable - 1m long
This cable is not only super-fashionable, with a woven pink and purple Blinka-like pattern, it's also fully reversible! That's right, you will save seconds a day by...
https://www.adafruit.com/product/4111

FAQ

Where did you get that awesome PyPortal Case?

The PyPortal case used in this guide is the PyPortal Retro Case designed by the Ruiz Brothers. Click here to learn more and visit the guide... ()

A Note on Security

THIS IS NOT A QUESTION MORE OF A COMMENT. YOU ARE PROGRAMMING THE TOTP SECRET INTO THE FLASH OF THE MICROCONTROLLER AND ITS NOT ENCRYPTED OR PROTECTED AT ALL ANYONE COULD BREAK INTO YOUR APARTMENT, GO TO YOUR BEDROOM, LOOK ON YOUR DESK, FIND THIS AND THEN CONNECT IT UP TO THEIR HACKER LAPTOP TO GRAB YOUR SECRET KEY THEN IF THEY HAD YOUR USERNAME AND PASSWORD THEY WOULD BE ABLE TO LOG IN AS YOU AND THIS IS REALLY INSECURE ITS SO IRRESPONSIBLE TO CONSIDER PUBLISHING A PROJECT LIKE THIS BY THE WAY DID YOU SEE THAT SNOWDEN APP? MAYBE YOU CAN RUN THAT ON A PHONE SO YOU CAN WATCH YOUR DESK REMOTELY AND MAKE SURE NOBODY BROKE IN TO STEAL YOUR PYPORTAL? OH WAIT YOU JUST SAID YOU DON'T HAVE A PHONE. OK I DONT KNOW WHAT MY QUESTION IS

This project is probably not for you

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

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

- You can add the lib folder to your CIRCUITPY drive. This will ensure you have all the drivers. But it will take a bunch of space on the 8 MB disk
- Add each library as you need it, this will reduce the space usage but you'll need to put in a little more effort.

At a minimum we recommend the following libraries, in fact we more than recommend. They're basically required. So grab them and install them into CIRCUITPY/lib now!

- adafruit_esp32spi - This is the library that gives you internet access via the ESP32 using (you guessed it!) SPI transport. You need this for anything Internet
- adafruit_requests - This library allows us to perform HTTP requests and get responses back from servers. GET/POST/PUT/PATCH - they're all in here!
- adafruit_pyportal - This is our friendly wrapper library that does a lot of our projects, displays graphics and text, fetches data from the internet. Nearly all of our projects depend on it!
- adafruit_portalbase - This library is the base library that adafruit_pyportal library is built on top of.
- adafruit_touchscreen - a library for reading touches from the resistive touchscreen. Handles all the analog noodling, rotation and calibration for you.
- adafruit_io - this library helps connect the PyPortal to our free datalogging and viewing service
- adafruit_imageload - an image display helper, required for any graphics!
- adafruit_display_text - not surprisingly, it displays text on the screen
- adafruit_bitmap_font - we have fancy font support, and its easy to make new fonts. This library reads and parses font files.
- adafruit_slideshow - for making image slideshows - handy for quick display of graphics and sound
- neopixel - for controlling the onboard neopixel
- adafruit_adt7410 - library to read the temperature from the on-board Analog Devices ADT7410 precision temperature sensor (not necessary for Titano or Pynt)
- adafruit_bus_device - low level support for I2C/SPI
- adafruit_fakerequests - This library allows you to create fake HTTP requests by using local files.

Internet Connect!

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

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

What's a secrets file?

We expect people to share tons of projects as they build CircuitPython WiFi widgets. What we want to avoid is people accidentally sharing their passwords or secret tokens and API keys. So, we designed all our examples to use a secrets.py file, that is in your CIRCUITPY drive, to hold secret/private/custom data. That way you can share your main project without worrying about accidentally sharing private stuff.

Your secrets.py file should look like this:

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

secrets = {
    'ssid' : 'home ssid',
    'password' : 'my password',
    'timezone' : "America/New_York", # http://worldtimeapi.org/timezones
    'github_token' : 'fawfj23rakjnfawiefa',
    'hackaday_token' : 'h4xx0rs3kret',
    }
```

Inside is a python dictionary named secrets with a line for each entry. Each entry has an entry name (say `'ssid'`) and then a colon to separate it from the entry key `'home ssid'` and finally a comma ,

At a minimum you'll need the `ssid` and `password` for your local WiFi setup. As you make projects you may need more tokens and keys, just add them one line at a time. See for example other tokens such as one for accessing github or the hackaday API. Other non-secret data like your timezone can also go here, just cause it's called secrets doesn't mean you can't have general customization data in there!

For the correct time zone string, look at http://worldtimeapi.org/timezones () and remember that if your city is not listed, look for a city in the same time zone, for example Boston, New York, Philadelphia, Washington DC, and Miami are all on the same time as New York.

Of course, don't share your secrets.py - keep that out of GitHub, Discord or other project-sharing sites.

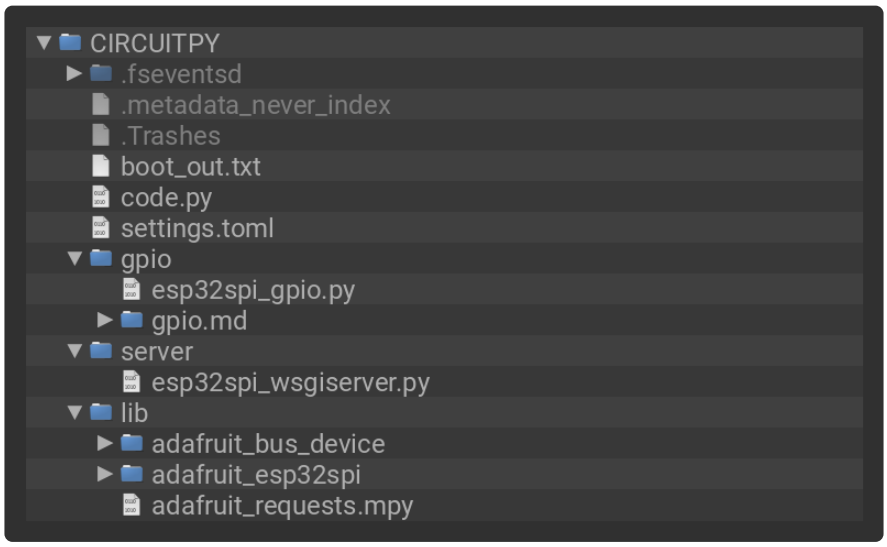
Connect to WiFi

OK now you have your secrets setup - you can connect to the Internet.

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

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

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




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

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

# Get wifi details and more from a secrets.py file
try:
    from secrets import secrets
except ImportError:
    print("WiFi secrets are kept in secrets.py, please add them there!")
    raise
```

```
print("ESP32 SPI webclient test")

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


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

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

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

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

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

requests.set_socket(socket, esp)

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

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

print("Connecting to AP...")
while not esp.is_connected:
    try:
        esp.connect_AP(secrets["ssid"], secrets["password"])
    except 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)
```

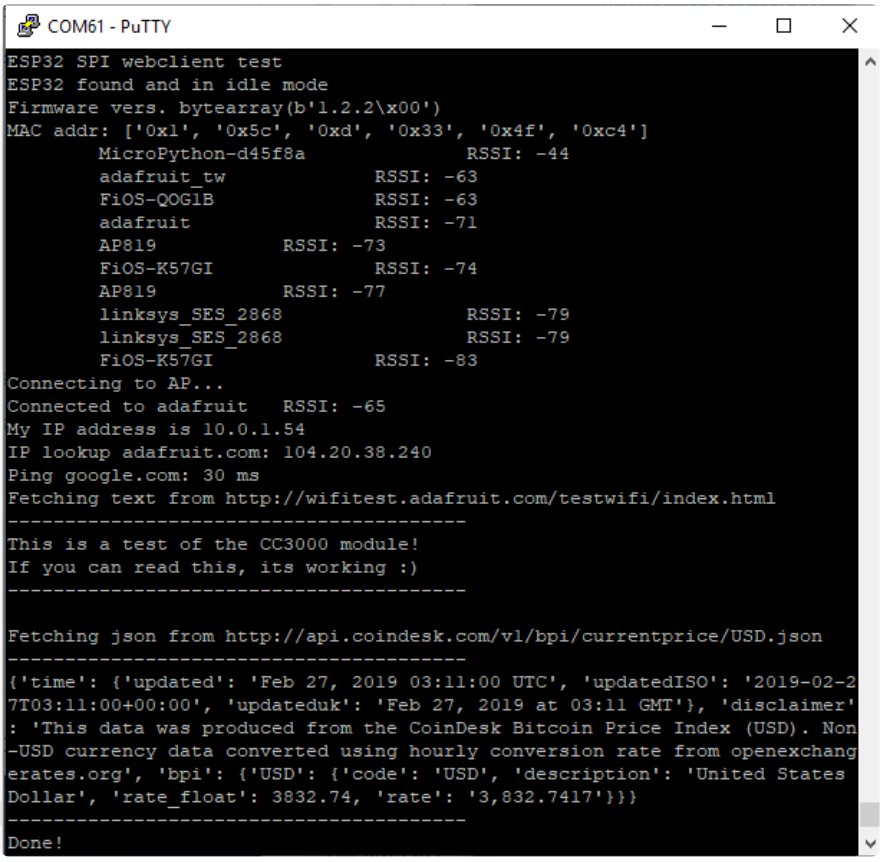
```
r.close()

print("Done!")
```

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

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

In a serial console, you should see something like the following. For more information about connecting with a serial console, view the guide Connecting to the Serial Console ().



In order, the example code...

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

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

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

Tells our `requests` library the type of socket we're using (socket type varies by connectivity type - we'll be using the `adafruit_esp32spi_socket` for this example).

We'll also set the interface to an `esp` object. This is a little bit of a hack, but it lets us use `requests` like CPython does.

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

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

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

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

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

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

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

OK now we're getting to the really interesting part. With a SAMD51 or other large-RAM (well, over 32 KB) device, we can do a lot of neat tricks. Like for example we can implement an interface a lot like requests () - 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()
```

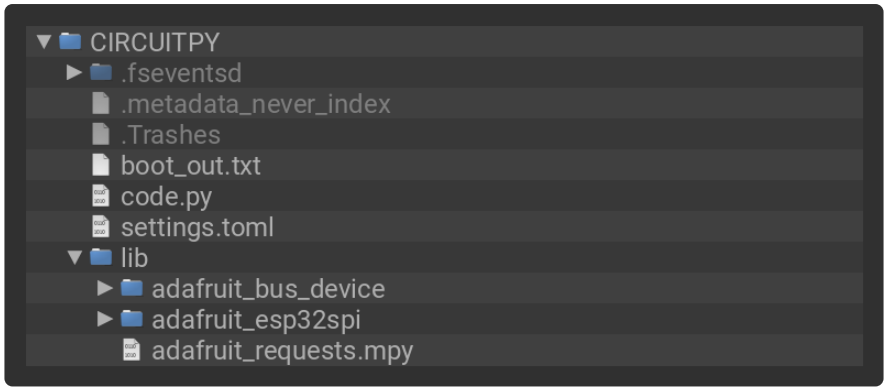
Requests

We've written a requests-like () library for web interfacing named Adafruit_CircuitPython_Requests (). This library allows you to send HTTP/1.1 requests without "crafting" them and provides helpful methods for parsing the response from the server.

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

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

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




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

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

```
from adafruit_esp32spi import adafruit_esp32spi
import adafruit_requests as requests

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

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

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

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

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

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

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

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

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

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

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

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

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

```
print("-" * 40)

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

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

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

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

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

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

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

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

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

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

HTTP GET with Requests

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

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

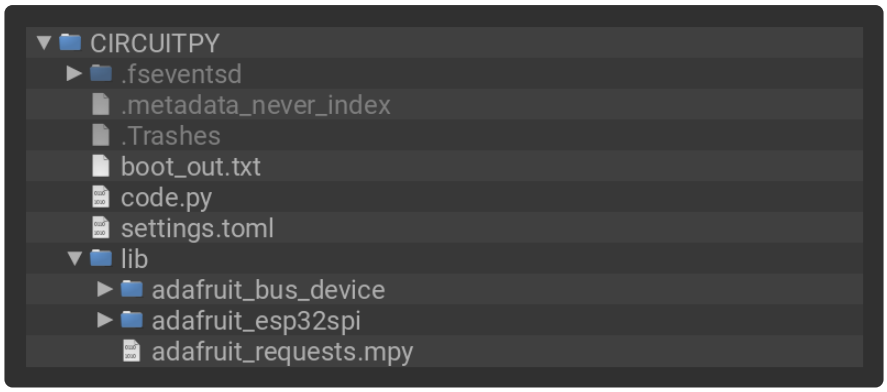
Want to send custom HTTP headers, parse the response as raw bytes, or handle a response's http status code in your CircuitPython code?

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:




```
# 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
from adafruit_esp32spi import adafruit_esp32spi
import adafruit_requests as requests

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

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

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

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

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

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

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

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

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

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

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

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

WiFi Manager

That simpletest example works but it's a little finicky - you need to constantly check WiFi status and have many loops to manage connections and disconnections. For more advanced uses, we recommend using the WiFiManager object. It will wrap the connection/status/requests loop for you - reconnecting if WiFi drops, resetting the ESP32 if it gets into a bad state, etc.

Here's a more advanced example that shows the WiFi manager and also how to POST data with some extra headers:

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
import board
import busio
from digitalio import DigitalInOut
import neopixel
from adafruit_esp32spi import adafruit_esp32spi
from adafruit_esp32spi import adafruit_esp32spi_wifimanager

print("ESP32 SPI webclient test")

# Get wifi details and more from a secrets.py file
try:
    from secrets import secrets
except ImportError:
    print("WiFi secrets are kept in secrets.py, please add them there!")
    raise

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

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

spi = busio.SPI(board.SCK, board.MOSI, board.MISO)
esp = adafruit_esp32spi.ESP_SPIcontrol(spi, esp32_cs, esp32_ready, esp32_reset)
"""Use below for Most Boards"""
status_light = neopixel.NeoPixel(
    board.NEOPIXEL, 1, brightness=0.2
)  # Uncomment for Most Boards
"""Uncomment below for ItsyBitsy M4"""
# status_light = dotstar.DotStar(board.APA102_SCK, board.APA102_MOSI, 1,
brightness=0.2)
# Uncomment below for an externally defined RGB LED
# import adafruit_rgbled
# from adafruit_esp32spi import PWMOut
# 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`. ()

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

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

Code Setup

Install the Mu Editor

This guide requires you to edit and interact with CircuitPython code. While you can use any text editor of your choosing, Mu is a simple code editor that works with the Adafruit CircuitPython boards. It's written in Python and works on Windows, MacOS, Linux and Raspberry Pi. The serial console is built right in, so you get immediate feedback from your board's serial output!

Before proceeding, click the button below to install the Mu Editor. There are versions for PC, mac, and Linux.

Install Mu Editor

CircuitPython Library Installation

First make sure you are running the latest version of Adafruit CircuitPython () for your board.

Next you'll need to install the necessary libraries to use the hardware--carefully follow the steps to find and install these libraries from Adafruit's CircuitPython library bundle () matching your version of CircuitPython. PyPortal requires at least CircuitPython version 4.0.0.

Before continuing make sure your board's lib folder has the following files and folders copied over:

- adafruit_binascii.mpy
- adafruit_esp32spi
- adafruit_pyportal.mpy
- adafruit_bitmap_font
- adafruit_hashlib
- adafruit_requests.mpy
- adafruit_bus_device
- adafruit_imageload
- adafruit_button.mpy
- adafruit_io
- adafruit_touchscreen.mpy
- adafruit_display_shapes
- adafruit_ntp.mpy
- neopixel.mpy
- adafruit_display_text
- adafruit_progressbar
- simpleio.mpy

Add 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_TOTP_Friend.

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

```
# SPDX-FileCopyrightText: 2017 Limor Fried for Adafruit Industries
#
# SPDX-License-Identifier: MIT

import time

import board
import busio
from digitalio import DigitalInOut
import displayio
import terminalio
from simpleio import map_range
import adafruit_hashlib as hashlib
import adafruit_touchscreen
from adafruit_button import Button
from adafruit_progressbar.progressbar import ProgressBar
from adafruit_display_text.label import Label
from adafruit_esp32spi import adafruit_esp32spi
from adafruit_pyportal import PyPortal
import rtc


# Background Color
BACKGROUND = 0x0

# Button color
BTN_COLOR = 0xFFFFFF

# Button text color
BTN_TEXT_COLOR = 0x0

# Set to true if you never want to go to sleep!
ALWAYS_ON = True

# How long to stay on if not in always_on mode
ON_SECONDS = 60

# 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

# Initialize PyPortal Display
display = board.DISPLAY

WIDTH = board.DISPLAY.width
HEIGHT = board.DISPLAY.height
ts = adafruit_touchscreen.Touchscreen(board.TOUCH_XL, board.TOUCH_XR,
                                      board.TOUCH_YD, board.TOUCH_YU,
                                      calibration=(
                                          (5200, 59000),
                                          (5800, 57000)
                                          ),
                                      size=(WIDTH, HEIGHT))

# Create a SHA1 Object
SHA1 = hashlib.sha1

# PyPortal ESP32 AirLift Pins
esp32_cs = DigitalInOut(board.ESP_CS)
esp32_ready = DigitalInOut(board.ESP_BUSY)
```

```
esp32_reset = DigitalInOut(board.ESP_RESET)

# Initialize PyPortal ESP32 AirLift
spi = busio.SPI(board.SCK, board.MOSI, board.MISO)
esp = adafruit_esp32spi.ESP_SPIcontrol(spi, esp32_cs, esp32_ready, esp32_reset)

def HMAC(k, m):
    """# HMAC implementation, as hashlib/hmac wouldn't fit
    From https://en.wikipedia.org/wiki/Hash-based_message_authentication_code

    """
    SHA1_BLOCK_SIZE = 64
    KEY_BLOCK = k + (b'\0' * (SHA1_BLOCK_SIZE - len(k)))
    KEY_INNER = bytes((x ^ 0x36) for x in KEY_BLOCK)
    KEY_OUTER = bytes((x ^ 0x5C) for x in KEY_BLOCK)
    inner_message = KEY_INNER + m
    outer_message = KEY_OUTER + SHA1(inner_message).digest()
    return SHA1(outer_message)

def base32_decode(encoded):
    missing_padding = len(encoded) % 8
    if missing_padding != 0:
        encoded += '=' * (8 - missing_padding)
    encoded = encoded.upper()
    chunks = [encoded[i:i + 8] for i in range(0, len(encoded), 8)]

    out = []
    for chunk in chunks:
        bits = 0
        bitbuff = 0
        for c in chunk:
            if 'A' <= c <= 'Z':
                n = ord(c) - ord('A')
            elif '2' <= c <= '7':
                n = ord(c) - ord('2') + 26
            elif c == '=':
                continue
            else:
                raise ValueError("Not base32")
            # 5 bits per 8 chars of base32
            bits += 5
            # shift down and add the current value
            bitbuff <<= 5
            bitbuff |= n
            # great! we have enough to extract a byte
            if bits >= 8:
                bits -= 8
                byte = bitbuff >> bits  # grab top 8 bits
                bitbuff &= ~(0xFF << bits)  # and clear them
                out.append(byte)  # store what we got
    return out

def int_to_bytestring(int_val, padding=8):
    result = []
    while int_val != 0:
        result.insert(0, int_val & 0xFF)
        int_val >>= 8
    result = [0] * (padding - len(result)) + result
    return bytes(result)


def generate_otp(int_input, secret_key, digits=6):
    """ HMAC -> OTP generator, pretty much same as
    https://github.com/pyotp/pyotp/blob/master/src/pyotp/otp.py

    """
    if int_input < 0:
        raise ValueError('input must be positive integer')
    hmac_hash = bytearray(
```

```
        HMAC(bytes(base32_decode(secret_key)),
             int_to_bytestring(int_input)).digest()
    )
    offset = hmac_hash[-1] & 0xf
    code = ((hmac_hash[offset] & 0x7f) << 24 |
            (hmac_hash[offset + 1] & 0xff) << 16 |
            (hmac_hash[offset + 2] & 0xff) << 8 |
            (hmac_hash[offset + 3] & 0xff))
    str_code = str(code % 10 ** digits)
    while len(str_code) < digits:
        str_code = '0' + str_code

    return str_code

def display_otp_key(secret_name, secret_otp):
    """Updates the displayio labels to display formatted OTP key and name.

    """
    # display the key's name
    label_title.text = secret_name
    # format and display the OTP
    label_secret.text = "{} {}".format(str(secret_otp)[0:3], str(secret_otp)[3:6])
    print("OTP Name: {}\nOTP Key: {}".format(secret_name, secret_otp))

print("==========================================")

# GFX Font
font = terminalio.FONT

# Initialize new PyPortal object
pyportal = PyPortal(esp=esp,
                    external_spi=spi)

# Root DisplayIO
root_group = displayio.Group()
display.show(root_group)

BACKGROUND = BACKGROUND if isinstance(BACKGROUND, int) else 0x0
bg_bitmap = displayio.Bitmap(display.width, display.height, 1)
bg_palette = displayio.Palette(1)
bg_palette[0] = BACKGROUND
background = displayio.TileGrid(bg_bitmap, pixel_shader=bg_palette)

# Create a new DisplayIO group
splash = displayio.Group()

splash.append(background)

key_group = displayio.Group(scale=5)
# We'll use a default text placeholder for this label
label_secret = Label(font, text="000 000")
label_secret.x = (display.width // 2) // 13
label_secret.y = 17
key_group.append(label_secret)

label_title = Label(font)
label_title.text = "  Loading.."
label_title.x = 0
label_title.y = 5
key_group.append(label_title)

# append key_group to splash
splash.append(key_group)

# Show the group
display.show(splash)

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 ", secrets['ssid'])

# get_time will raise ValueError if the time isn't available yet so loop until
# it works.
now_utc = None
while now_utc is None:
    try:
        now_utc = time.localtime(esp.get_time()[0])
    except ValueError:
        pass
rtc.RTC().datetime = now_utc

# Get the current time in seconds since Jan 1, 1970
t = time.time()
print("Seconds since Jan 1, 1970: {} seconds".format(t))

# Instead of using RTC which means converting back and forth
# we'll just keep track of seconds-elapsed-since-NTP-call
mono_time = int(time.monotonic())
print("Monotonic time", mono_time)

# Add buttons to the interface
assert len(secrets['totp_keys']) < 6, "This code can only display 5 keys at a time"

# generate buttons
buttons = []

btn_x = 5
for i in secrets['totp_keys']:
    button = Button(name=i[0], x=btn_x,
                    y=175, width=60,
                    height=60, label=i[0].strip(" "),
                    label_font=font, label_color=BTN_TEXT_COLOR,
                    fill_color=BTN_COLOR, style=Button.ROUNDRECT)
    buttons.append(button)
    # add padding btween buttons
    btn_x += 63

# append buttons to splash group
for b in buttons:
    splash.append(b)

# refrsh timer label
label_timer = Label(font)
label_timer.x = (display.width // 2) // 13
label_timer.y = 15
splash.append(label_timer)

# create a new progress bar
progress_bar = ProgressBar(display.width//5, 125,
                           200, 30, bar_color = 0xAAAAAA)

splash.append(progress_bar)

# how long to stay on if not in always_on mode
countdown = ON_SECONDS

# current button state, defaults to first item in totp_keys
current_button = secrets['totp_keys'][0][0]
buttons[0].selected = True

while ALWAYS_ON or (countdown > 0):
    # Calculate current time based on NTP + monotonic
```

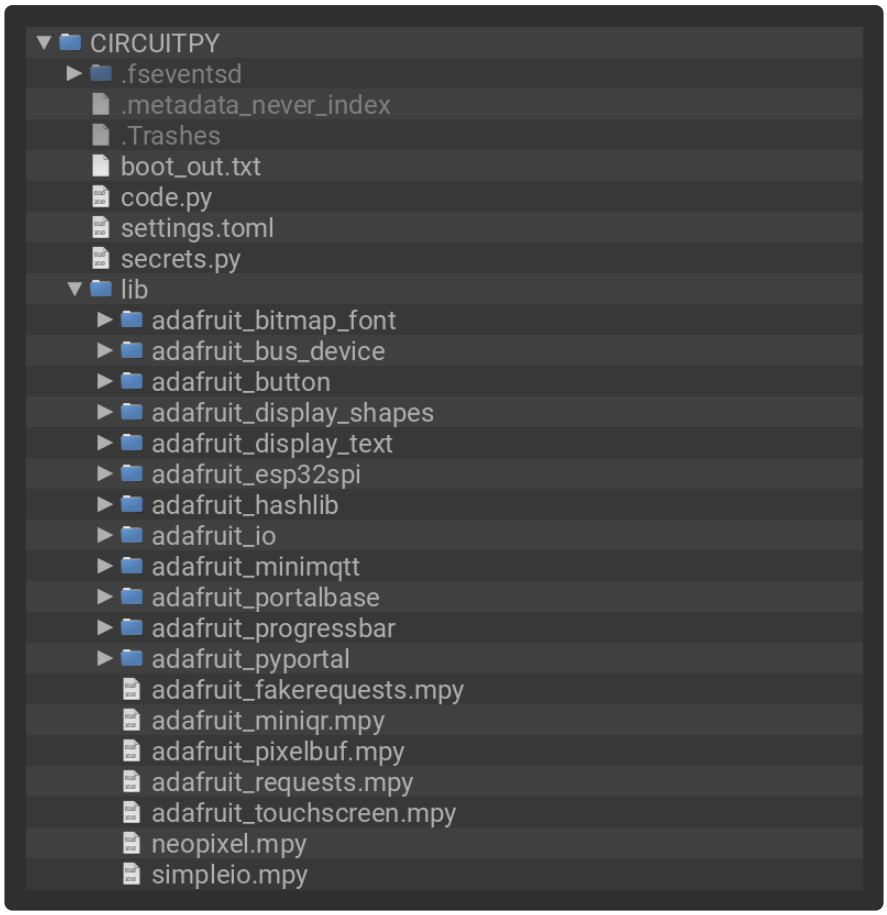
```
    unix_time = t - mono_time + int(time.monotonic())

    # Update the key refresh timer
    timer = time.localtime(time.time()).tm_sec
    # timer resets on :00/:30
    if timer > 30:
        countdown = 60 - timer
    else:
        countdown = 30 - timer
    print('NTP Countdown: {}%'.format(countdown))
    # change the timer bar's color if text is about to refresh
    progress_bar.fill = 0xFFFFFF
    if countdown < 5:
        progress_bar.fill = 0xFF0000

    # update the progress_bar with countdown
    countdown = map_range(countdown, 0, 30, 0.0, 1.0)
    progress_bar.progress = countdown

    # poll the touchscreen
    p = ts.touch_point
    # if the touchscreen was pressed
    if p:
        for i, b in enumerate(buttons):
            if b.contains(p):
                b.selected = True
                for name, secret in secrets['totp_keys']:
                    # check if button name is the same as a key name
                    if b.name == name:
                        current_button = name
                        # Generate OTP
                        otp = generate_otp(unix_time // 30, secret)
                        display_otp_key(name, otp)
            else:
                b.selected = False
    else:
        for name, secret in secrets['totp_keys']:
            if current_button == name:
                # Generate OTP
                otp = generate_otp(unix_time // 30, secret)
                display_otp_key(name, otp)
    # We'll update every 1/4 second, we can hash very fast so its no biggie!
    countdown -= 0.25
    time.sleep(0.25)
```

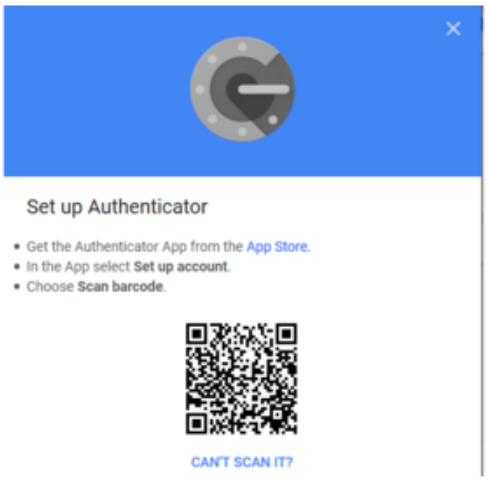
Once all the files are copied from your computer to the PyPortal, you should have the following files on your CIRCUITPY drive:



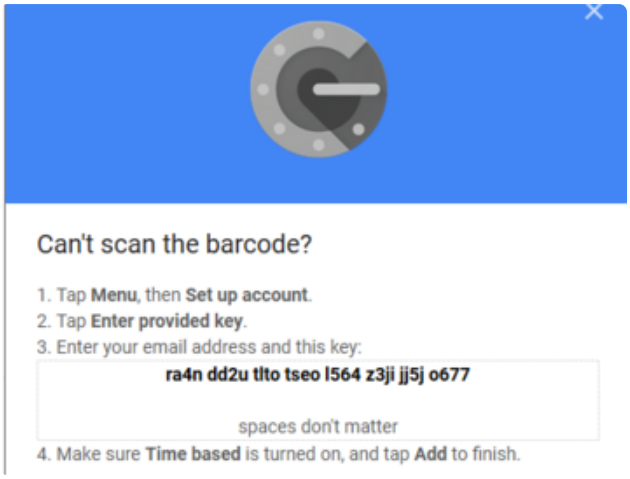
Set Up Tokens

You'll also need to get 2 factor "authenticator tokens/secrets". Each site is a little different about how it does this.

For example, when you set up Gmail for 2FA it will show you a QR code like this:



For example, when you set up Gmail for 2FA it will show you a QR code like this



Which is great for phones. For us, we need the base32-encoded token. Click the Can't Scan It? link or otherwise request the text token. You'll get a page like this.

(Don't freak out - this isnt a real key)

That string of letters and numbers may be uppercase or lower case, it may also be 16 digits or 24 or 32 or some other qty. It doesn't matter! Grab that string, and remove the spaces so its one long string like `"ra4ndd2utltotseol564z3jijj5jo677"` Note that the number 0 and number 1 never appear so anything that looks like an `O`, `l` or an `I` is a letter.

Secrets File Setup

Open the secrets.py file on your CircuitPython device using Mu. You're going to edit this file to enter your WiFi credentials along with your keys.

- Change **`yourwifissid`** to the name of your WiFi network
- Change **`yourwifipassword`** to your WiFi network's password

```
# 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' : 'yourwifissid',
    'password' : 'yourwifipassword',
    'timezone' : "America/New_York", # http://worldtimeapi.org/timezones
    # https://github.com/pyotp/pyotp example
    'totp_keys' : [("Discord ", "JBSWY3DPEHPK3PXP"),
                   ("Gmail", "JBSWY3DPEHPK3PZP"),
                   ("GitHub", "JBSWY5DZEHPK3PXP"),
                   ("Adafruit", "JBSWY6DZEHPK3PXP"),
                   ("Outlook", "JBSWY7DZEHPK3PXP")]
    }
```

This code displays up to five keys. If you have less than 5 keys to display, you may remove items from `totp_keys`. The buttons on the PyPortal display are dynamically generated based on how many items are in `totp_keys`:

```
'totp_keys' : [("Discord ", "JBSWY3DPEHPK3PXP"),
               ("Gmail", "JBSWY3DPEHPK3PZP"),
               ("GitHub", "JBSWY5DZEHPK3PXP")]
```

Once you've added your keys, save the modified secrets.py file.

Code Usage

The PyPortal fetches and sets the microcontroller's current UTC time. This may take up to 15 seconds.

Formatting the Name Label

Add spaces to the left of the TOTP key name in the `totp_keys` list to pad the left side:

When the timer is five seconds away from refreshing, the progress bar will change from white to red.

Once the time hits zero, the progress bar resets and the key regenerates.

Customizing the Authentication Friend

Everyone's desk is different and we can do a bit of customization to our PyPortal authentication friend.

Background

In the code.py file, change `BACKGROUND = 0x0` to a new hexadecimal color value and save the file.

Customizing the Buttons

We can change the button's fill color and text color.

To change the button's fill color: in the code.py file, change `BTN_COLOR = 0xFFFFFF` to a hexadecimal color value.

To change the button's text color: in the code.py file, change `BTN_TEXT_COLOR` to a hexadecimal color value.

Formatting the Name Label

The x-location for the name label is zero by default. You may need to modify the `totp_keys` list to center shorter TOTP key names.

In secrets.py, add spaces to the left of the the key in `totp_keys` to center the names, if desired:

```
'totp_keys' : [("  Discord", "JBSWY3DPEHPK3PXP"),
               ("   Gmail", "JBSWY3DPEHPK3PZP"),
               ("  GitHub", "JBSWY5DZEHPK3PXP"),
               ("  Adafruit", "JBSWY6DZEHPK3PXP"),
               ("  Outlook", "JBSWY7DZEHPK3PXP")]
```