PyPortal Quarantine Clock

Created by Brent Rubell

https://learn.adafruit.com/pyportal-quarantine-clock

Last updated on 2025-04-14 05:56:28 PM EDT

Table of Contents

Overview

Do you know what day it is? Perhaps you're finding yourself losing track of time and your routine is suffering. The **PyPortal Quarantine Clock** has got you covered! It only displays the essential "stay-at-home" information - the current day and a rough estimate of the time.

The **PyPortal obtains the precise time from the internet using the PyPortal's built-in WiFi module**, then updates and displays an estimate of the time and date. The quarantine clock is **programmed using CircuitPython and easily customizable**.

This beginner PyPortal project is a remix of mwfisher3's quarantine clock (https:// adafru.it/KCe). We've converted it to work with CircuitPython and made changes so it runs on the PyPortal.

Parts

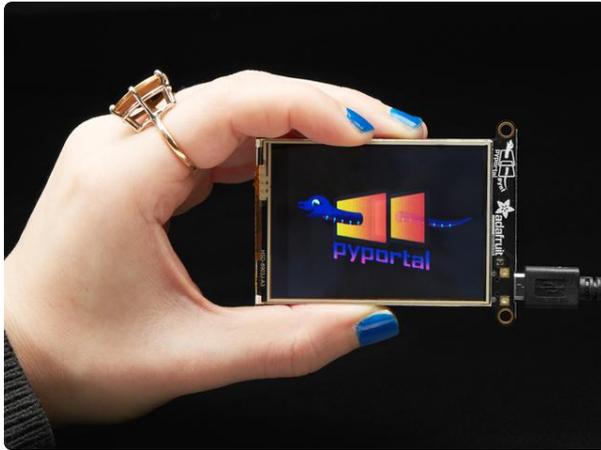Adafruit PyPortal - CircuitPython Powered Internet Display
PyPortal, our easy-to-use IoT device that allows you to create all the things for the "Internet of Things" in minutes. Make custom touch screen interface...
https://www.adafruit.com/product/4116

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
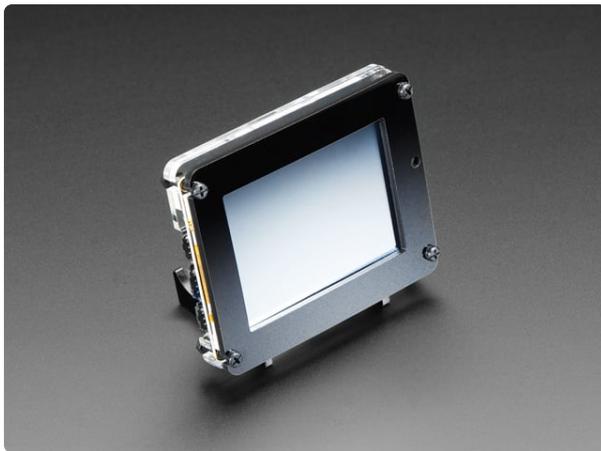
Adafruit PyPortal Desktop Stand Enclosure Kit

PyPortal is our easy-to-use IoT device that allows you to create all the things for the "Internet of Things" in minutes. Create little pocket...

https://www.adafruit.com/product/4146

USB cable - USB A to Micro-B

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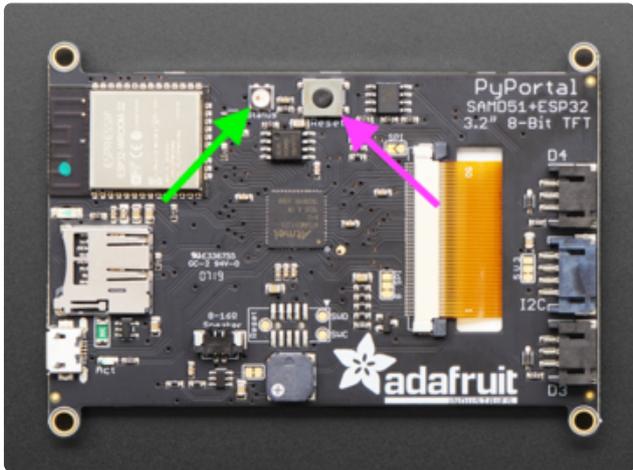
A lot of people end up using charge-only USB cables and it is very frustrating! So make sure you have a USB cable you know is good for data sync.

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!
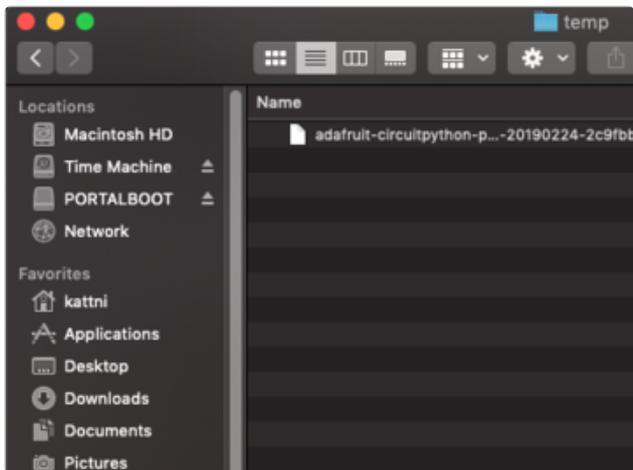
You will see a new disk drive appear called **PORTALBOOT**.
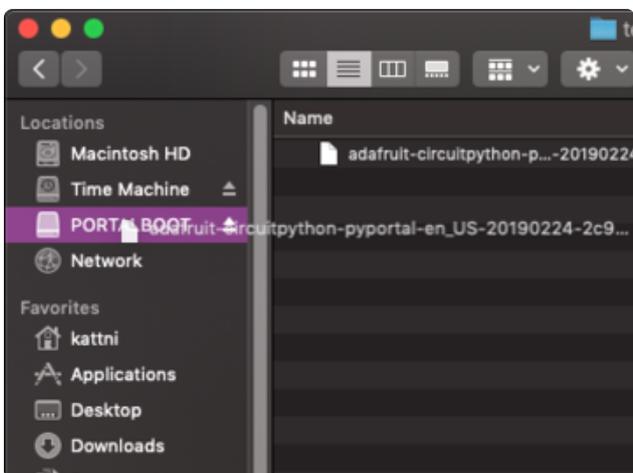
Drag the **adafruit-circuitpython-pyportal-<whatever>.uf2** file to **PORTALBOOT.**
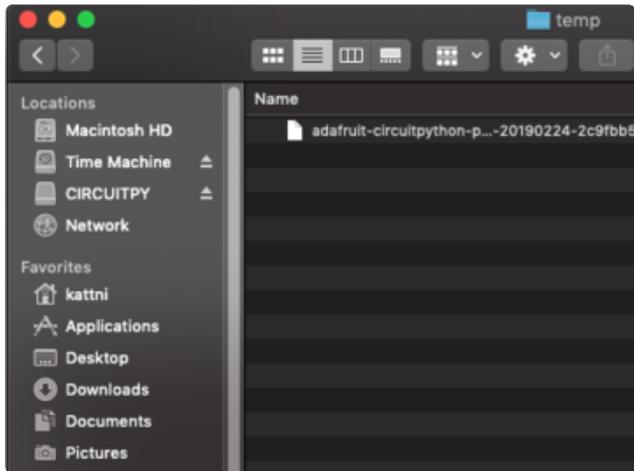
The LED will flash. Then, the **PORTALBOOT** drive will disappear and a new disk drive called **CIRCUITPY** will appear.

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.

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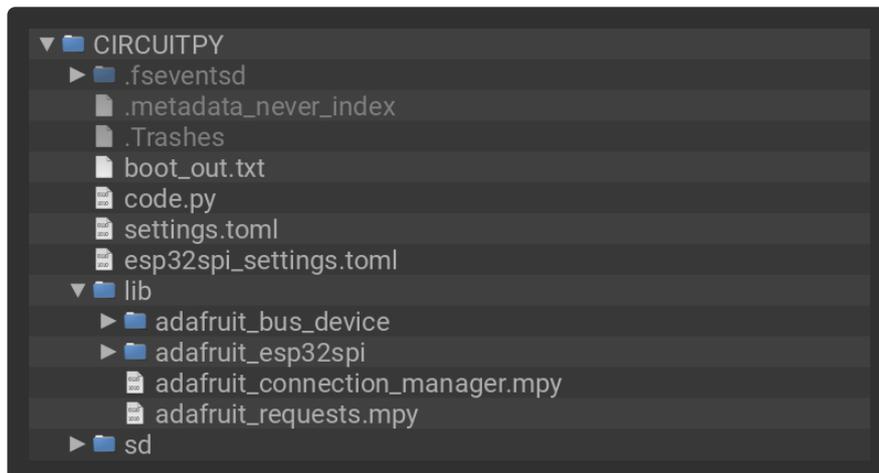
> Update to CircuitPython 9.2.x or later to use this example.

```
# 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
```

```python
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).

```
>>> 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). It is a lot like the regular Python library named requests (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.

```python
# 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.ap_info.ssid, "utf-8"), "\tRSSI:",
radio.ap_info.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)
```
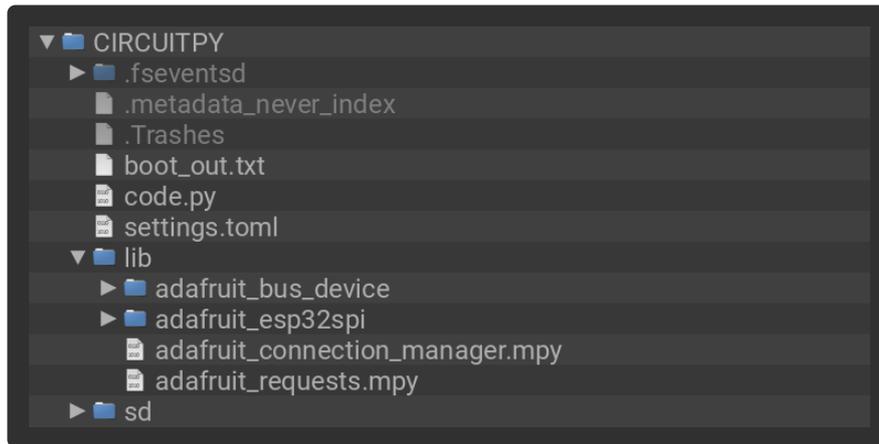
Your **CIRCUITPY** drive should now look similar to the following image:

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.

```python
# 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)
```

```python
# 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)
```
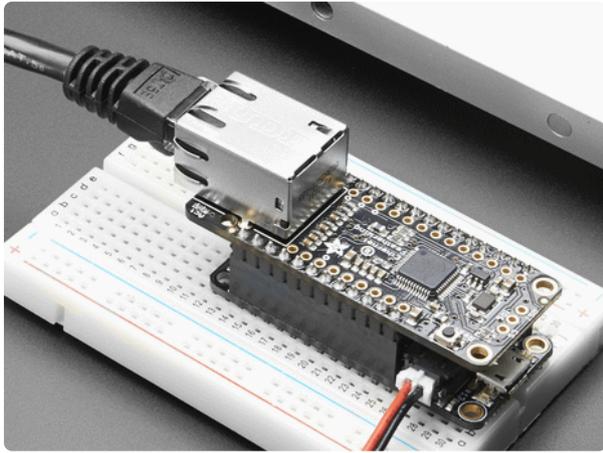
Further Information

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

Networking in CircuitPython
By Anne Barela

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

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), 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"))
```


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.

Code PyPortal with CircuitPython

Settings File Setup

In order to get the precise time, our project will query the [Adafruit IO Internet of Things service](https://adafru.it/fH9) (https://adafru.it/fH9) 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).

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) (https://adafru.it/fsU) and simply click the Adafruit IO Key on the right hand side of the Adafruit IO header to obtain this information.

> Using an online time reference means we will account for daylight savings time and never have any drift!
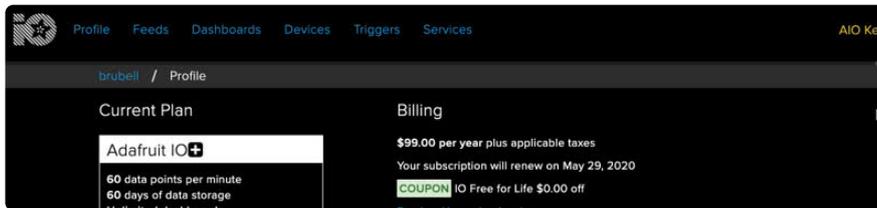
Then, add them to the **settings.toml** file:

```
CIRCUITPY_WIFI_SSID = "your_wifi_ssid"
CIRCUITPY_WIFI_PASSWORD = "your_wifi_password"
AIO_USERNAME = "your_aio_username"
AIO_KEY = "your_aio_key"
TIMEZONE = "America/New_York" # http://worldtimeapi.org/timezones
```

The PyPortal code determines the local time by checking the internet information using the WiFi connection. It uses your local timezone to fetch and determine the local time. Once you've set up your timezone, you do not need to adjust for daylight savings time or leap years.

To do this, you'll need to set this line in your **settings.toml** file to your local timezone:

```
TIMEZONE = "America/New_York"
```

Don't know your timezone? Here's [a great list from the IANA Timezone Database](https://adafru.it/EgK) (https://adafru.it/EgK). Simply find the nearest timezone to your desired location, and use that name as displayed in the **TZ database name** column.

Install CircuitPython Code and Assets

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_Quarantine_Clock**.

```python
# SPDX-FileCopyrightText: 2020 Brent Rubell for Adafruit Industries
#
# SPDX-License-Identifier: MIT

import os
import time
import board
import busio
import digitalio
import adafruit_connection_manager
from adafruit_esp32spi import adafruit_esp32spi
import adafruit_requests
from adafruit_pyportal import PyPortal
from adafruit_bitmap_font import bitmap_font
from adafruit_display_text import label

# Label colors
LABEL_DAY_COLOR = 0xFFFFFF
LABEL_TIME_COLOR = 0x2a8eba

# the current working directory (where this file is)
cwd = ("/"+__file__).rsplit('/', 1)[0]
background = None
# un-comment to set background image
# background = cwd+"/background.bmp"

# Descriptions of each hour
# https://github.com/mwfisher3/QuarantineClock/blob/master/today.html
time_names = ["midnight-ish", "late night", "late", "super late",
              "super early","really early","dawn","morning",
              "morning","mid-morning","mid-morning","late morning",
              "noon-ish","afternoon","afternoon","mid-afternoon",
              "late afternoon","early evening","early evening","dusk-ish",
              "evening","evening","late evening","late evening"]

# Days of the week
week_days = ["Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday",
"Sunday"]

esp32_cs = digitalio.DigitalInOut(board.ESP_CS)
esp32_ready = digitalio.DigitalInOut(board.ESP_BUSY)
esp32_reset = digitalio.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,
debug=False)
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)

# initialize pyportal
```

```python
pyportal = PyPortal(esp=esp,
                    external_spi=spi)

# set pyportal's backlight brightness
pyportal.set_backlight(0.7)

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])

# Set the font and preload letters
font_large = bitmap_font.load_font("/fonts/Helvetica-Bold-44.bdf")
font_small = bitmap_font.load_font("/fonts/Helvetica-Bold-24.bdf")
font_large.load_glyphs(b'abcdefghjiklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890-
')
font_small.load_glyphs(b'abcdefghjiklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890-
()')

# Set up label for the day
label_day = label.Label(font_large, color=LABEL_DAY_COLOR)
label_day.x = board.DISPLAY.width // 7
label_day.y = 80
pyportal.splash.append(label_day)

# Set up label for the time
label_time = label.Label(font_small, color=LABEL_TIME_COLOR)
label_time.x = board.DISPLAY.width // 4
label_time.y = 150
pyportal.splash.append(label_time)

refresh_time = None
while True:
    # only query the network time every hour
    if (not refresh_time) or (time.monotonic() - refresh_time) > 3600:
        try:
            print("Getting new time from internet...")
            pyportal.get_local_time(os.getenv("TIMEZONE"))
            refresh_time = time.monotonic()
            # set the_time
            the_time = time.localtime()
        except (ValueError, RuntimeError, ConnectionError, OSError) as e:
            print("Failed to get data, retrying\n", e)
            esp.reset()
            continue

    # Convert tm_wday to name of day
    weekday = week_days[the_time.tm_wday]

    # set the day label's text
    label_day.text = weekday

    # set the time label's text
    label_time.text = "({})".format(time_names[the_time.tm_hour])

    # update every minute
    time.sleep(60)
```

Copy the contents of the **PyPortal_Quarantine_Clock** directory to your PyPortal's **CIRCUITPY** drive.
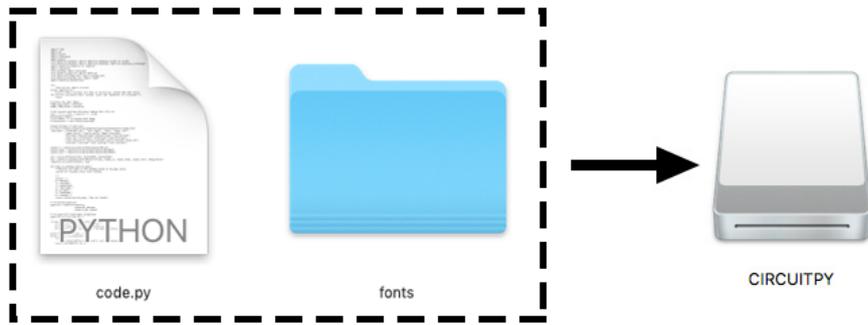
This is what the final contents of the **CIRCUITPY** drive will look like:


```
▼ 📁 CIRCUITPY
   ► 📁 .fseventsd
     📄 .metadata_never_index
     📄 .Trashes
     📄 boot_out.txt
     📄 code.py
     📄 settings.toml
   ▼ 📁 fonts
        📄 Helvetica-Bold-24.bdf
        📄 Helvetica-Bold-44.bdf
   ▼ 📁 month_clock
        📄 code.py
   ▼ 📁 lib
        ► 📁 adafruit_bitmap_font
        ► 📁 adafruit_bus_device
        ► 📁 adafruit_display_text
        ► 📁 adafruit_esp32spi
        ► 📁 adafruit_io
        ► 📁 adafruit_minimqtt
        ► 📁 adafruit_portalbase
        ► 📁 adafruit_pyportal
          📄 adafruit_connection_manager.mpy
          📄 adafruit_fakerequests.mpy
          📄 adafruit_miniqr.mpy
          📄 adafruit_pixelbuf.mpy
          📄 adafruit_requests.mpy
          📄 adafruit_ticks.mpy
          📄 adafruit_touchscreen.mpy
          📄 neopixel.mpy
          📄 simpleio.mpy
   ► 📁 sd
```

Code Usage

The code determines the local time by using the WiFi connection to query the Adafruit IO Time Service. It draws two labels to display the day of the week and the estimated time.

We use a list of time names (`time_names`) for a description of each hour. The code obtains the number of days since Sunday and calculates the weekday from it.

Each hour, the code fetches the network time from the Adafruit IO Time Service and refreshes the displayed time and date.

Customization
Set a background image

You can customize the background by setting a hex color or by making your own background .bmp file.

Using a hex color picker such as https://htmlcolorcodes.com (https://adafru.it/Eb7), pick your color and copy the hex value. change the following line of code from

```
background = None
```

to

```
background = 0xC91818
```

You can also customize the background for a different event, by making your own 320x240 16-bit RGB color .bmp file. **Name the .bmp file** background.bmp and copy it to the CIRCUITPY volume. In the code, change the following lines of code from

```
background = None
# un-comment to set background image
# background = cwd+"/background.bmp" background = None
```

to

```
# background = None
# un-comment to set background image
background = cwd+"/background.bmp"
```

Font

The font used for the date and time displayed on the PyPortal are loaded and appended on the display.

You can change the fonts used by the clock by changing the .bdf files on the two following lines in the code:

```
font_large = bitmap_font.load_font("/fonts/Helvetica-Bold-44.bdf")
font_small = bitmap_font.load_font("/fonts/Helvetica-Bold-24.bdf")
```

- If you're looking to display new fonts on your PyPortal - check out this guide for info about creating custom fonts (https://adafru.it/E7E).

Text Color

You can also adjust the colors for the time and date labels. Using a hex color picker such as https://htmlcolorcodes.com (https://adafru.it/Eb7), pick your color and then copy the hex value. Set `LABEL_DAY_COLOR` if you want to set the color of the weekday label or `LABEL_TIME_COLOR` to set the color of the estimated time label.

```
# Label colors
LABEL_DAY_COLOR = 0xFFFFFF
LABEL_TIME_COLOR = 0x8f42f4
```

Quarantine Month Clock

A user on the project's Reddit thread commented (https://adafru.it/KCq):

> This is way too specific for a quarantine clock. Can you make it say "April-ish"?

Instead of displaying the day of the week, you can edit the code a little bit to make the code display the month instead of the date:

```python
# SPDX-FileCopyrightText: 2020 Brent Rubell for Adafruit Industries
#
# SPDX-License-Identifier: MIT

import time
import board
import busio
import digitalio
import adafruit_connection_manager
from adafruit_esp32spi import adafruit_esp32spi
import adafruit_requests
from adafruit_pyportal import PyPortal
from adafruit_bitmap_font import bitmap_font
from adafruit_display_text import label

try:
    from secrets import secrets
except ImportError:
    print("""WiFi settings are kept in secrets.py, please add them there!
the secrets dictionary must contain 'ssid' and 'password' at a minimum""")
    raise

# Label colors
LABEL_DAY_COLOR = 0xFFFFFF
LABEL_TIME_COLOR = 0x2a8eba

# the current working directory (where this file is)
cwd = ("/"+__file__).rsplit('/', 1)[0]
background = None
# un-comment to set background image
# background = cwd+"/background.bmp"

# Descriptions of each hour
# https://github.com/mwfisher3/QuarantineClock/blob/master/today.html
time_names = ["midnight-ish", "late night", "late", "super late",
              "super early","really early","dawn","morning",
              "morning","mid-morning","mid-morning","late morning",
              "noon-ish","afternoon","afternoon","mid-afternoon",
              "late afternoon","early evening","early evening","dusk-ish",
              "evening","evening","late evening","late evening"]

# Months of the year
months = ["January", "January", "February", "March", "April",
          "May", "June", "July", "August",
          "September", "October", "November", "December"]

# Dictionary of tm_mon and month name.
# note: tm_mon structure in CircuitPython ranges from [1,12]
months = {
    1: "January",
    2: "February",
    3: "March",
    4: "April",
    5: "May",
    6: "June",
    7: "July",
    8: "August",
    9: "September",
    10: "October",
    11: "November",
```

```
        12: "December"
}


esp32_cs = digitalio.DigitalInOut(board.ESP_CS)
esp32_ready = digitalio.DigitalInOut(board.ESP_BUSY)
esp32_reset = digitalio.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,
debug=False)
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)

# initialize pyportal
pyportal = PyPortal(esp=esp,
                    external_spi=spi,
                    default_bg = None)

# set pyportal's backlight brightness
pyportal.set_backlight(0.2)

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])

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

# Set the font and preload letters
font_large = bitmap_font.load_font("/fonts/Helvetica-Bold-44.bdf")
font_small = bitmap_font.load_font("/fonts/Helvetica-Bold-24.bdf")
font_large.load_glyphs(b'abcdefghjiklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890-
')
font_small.load_glyphs(b'abcdefghjiklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890-
()')

# Set up label for the month
label_month = label.Label(font_large, color=LABEL_DAY_COLOR)
label_month.x = board.DISPLAY.width // 10
label_month.y = 80
pyportal.splash.append(label_month)

# Set up label for the time
label_time = label.Label(font_small, color=LABEL_TIME_COLOR)
label_time.x = board.DISPLAY.width // 3
label_time.y = 150
pyportal.splash.append(label_time)

refresh_time = None
while True:
    # only query the network time every hour
    if (not refresh_time) or (time.monotonic() - refresh_time) > 3600:
        try:
            print("Getting new time from internet...")
            pyportal.get_local_time(secrets['timezone'])
            refresh_time = time.monotonic()
            # set the_time
            the_time = time.localtime()
        except (ValueError, RuntimeError, ConnectionError, OSError) as e:
            print("Failed to get data, retrying\n", e)
            esp.reset()
```
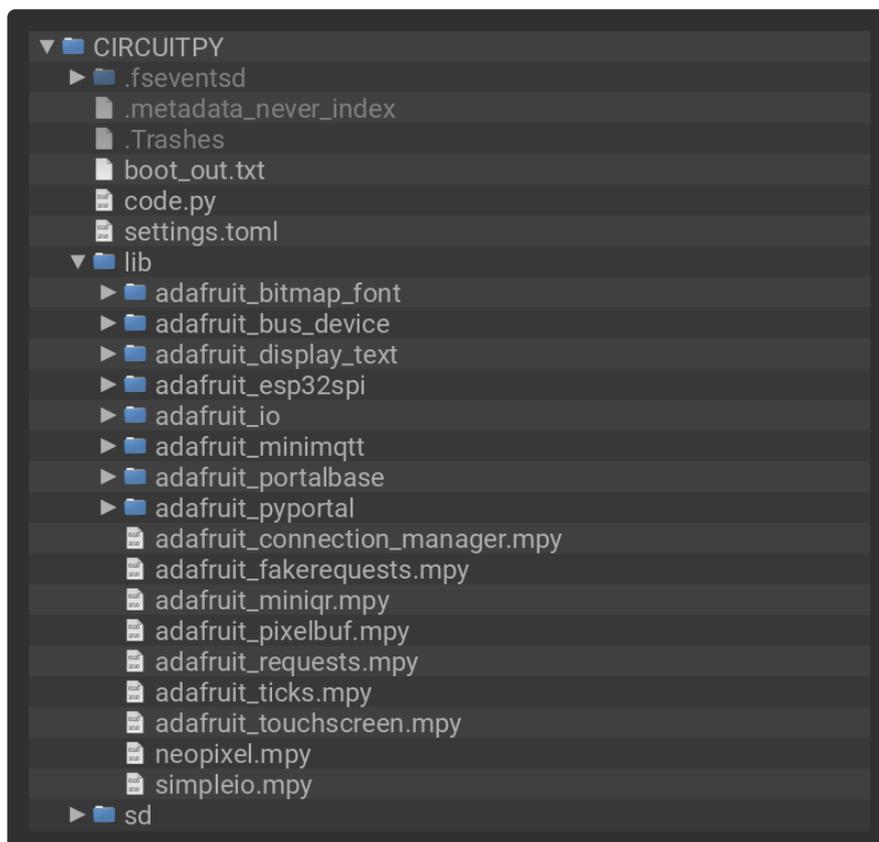
```
            continue
    # convert tm_mon value to month name
    month = months[the_time.tm_mon]

    # determine and display how far we are in the month
    if 1 <= the_time.tm_mday <= 14:
        label_month.text = "Early %s-ish"%month
    elif 15 <= the_time.tm_mday <= 24:
        label_month.text = "Mid %s-ish"%month
    else:
        label_month.text = "Late %s-ish"%month

    # set the time label's text
    label_time.text = "({})".format(time_names[the_time.tm_hour])

    # update every minute
    time.sleep(60)
```

```
▼ 📁 CIRCUITPY
  ► 📁 .fseventsd
    📄 .metadata_never_index
    📄 .Trashes
    📄 boot_out.txt
    📄 code.py
    📄 settings.toml
  ▼ 📁 lib
    ► 📁 adafruit_bitmap_font
    ► 📁 adafruit_bus_device
    ► 📁 adafruit_display_text
    ► 📁 adafruit_esp32spi
    ► 📁 adafruit_io
    ► 📁 adafruit_minimqtt
    ► 📁 adafruit_portalbase
    ► 📁 adafruit_pyportal
      📄 adafruit_connection_manager.mpy
      📄 adafruit_fakerequests.mpy
      📄 adafruit_miniqr.mpy
      📄 adafruit_pixelbuf.mpy
      📄 adafruit_requests.mpy
      📄 adafruit_ticks.mpy
      📄 adafruit_touchscreen.mpy
      📄 neopixel.mpy
      📄 simpleio.mpy
  ► 📁 sd
```

Code Walkthrough

When the code obtains the time from the internet, it stores the time returned in a specific structure type used to capture a date and a time called struct_time (https://adafru.it/Eri) .

```
Getting new time from internet...
Getting time for timezone America/New_York
struct_time(tm_year=2020, tm_mon=4, tm_mday=23, tm_hour=9, tm_min=38, tm_sec=41,
tm_wday=4, tm_yday=114, tm_isdst=None)
```

In the `struct_time`, `tm_mon` holds the the numerical representation of the month from 1 (January) to 12 (December). The updated code uses `tm_mon` to select an entry in the `months` dictionary

```
# convert tm_mon value to month name
month = months[the_time.tm_mon]
```

The conditional statement determines how far we are in the month: early month, in the middle of the month, or late in the month.

Then, the label is set to a string which is concatenates the time, month, and the phrase. i.e: "early + march + -ish"

```
# determine and display how far we are in the month
if 1 &lt;= the_time.tm_mday &lt;= 14:
  label_month.text = "Early %s-ish"%month
elif 15 &lt;= the_time.tm_mday &lt;= 24:
  label_month.text = "Mid %s-ish"%month
else:
  label_month.text = "Late %s-ish"%month
```