



PyPortal Event Count-Up Clock

Created by John Park



<https://learn.adafruit.com/pyportal-event-count-up-clock>

Last updated on 2024-11-18 12:57:40 PM EST

Table of Contents

Overview	3
<hr/>	
<ul style="list-style-type: none">• Parts• Additional Tools & Materials	
Install CircuitPython	5
<hr/>	
<ul style="list-style-type: none">• Set up CircuitPython Quick Start!• PyPortal Default Files	
PyPortal CircuitPython Setup	8
<hr/>	
<ul style="list-style-type: none">• Adafruit CircuitPython Bundle	
Create Your settings.toml File	9
<hr/>	
<ul style="list-style-type: none">• CircuitPython settings.toml File• settings.toml File Tips• Accessing Your settings.toml Information in code.py	
Internet Connect!	12
<hr/>	
<ul style="list-style-type: none">• Connect to WiFi• Requests• HTTP GET with Requests• HTTP POST with Requests• Advanced Requests Usage• WiFi Manager	
Code PyPortal with CircuitPython	25
<hr/>	
<ul style="list-style-type: none">• Weekly Countdown• Adafruit IO Time Server• Install CircuitPython Code and Assets• How it Works• Time• Font• Customization• Graphics• Text Position• Text Color• 3D Printed Stand	

Overview



Want to commemorate the years, days, hours, and minutes passed since an important event? The lunar landing, an anniversary, or maybe the amount of time since the last accident in your workshop? The PyPortal Event Count-Up Clock has got you covered!

Build your own PyPortal Event Count-Up Clock and then customize it for the event of your choice. All coded in CircuitPython, it gets the precise time from the internet using built-in WiFi, then draws the display with the ever forward marching of time. It makes you think, doesn't it.

Parts



[Adafruit PyPortal - CircuitPython Powered Internet Display](https://www.adafruit.com/product/4116)

PyPortal, our easy-to-use IoT device that allows you to create all the things for the “Internet of Things” in minutes. Make custom touch screen interface...

<https://www.adafruit.com/product/4116>



USB cable - USB A to Micro-B

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>

Additional Tools & Materials

You may want to create the optional desk stand for your PyPortal Event Count-Up Clock. For this you'll need:

- [3D printer](http://adafru.it/2673) (<http://adafru.it/2673>)
- [Filament](http://adafru.it/3731) (<http://adafru.it/3731>)
- [PyPortal Stand model file](https://adafru.it/EcN) (<https://adafru.it/EcN>)
- [M3 standoffs and screws](http://adafru.it/3816) (<http://adafru.it/3816>)

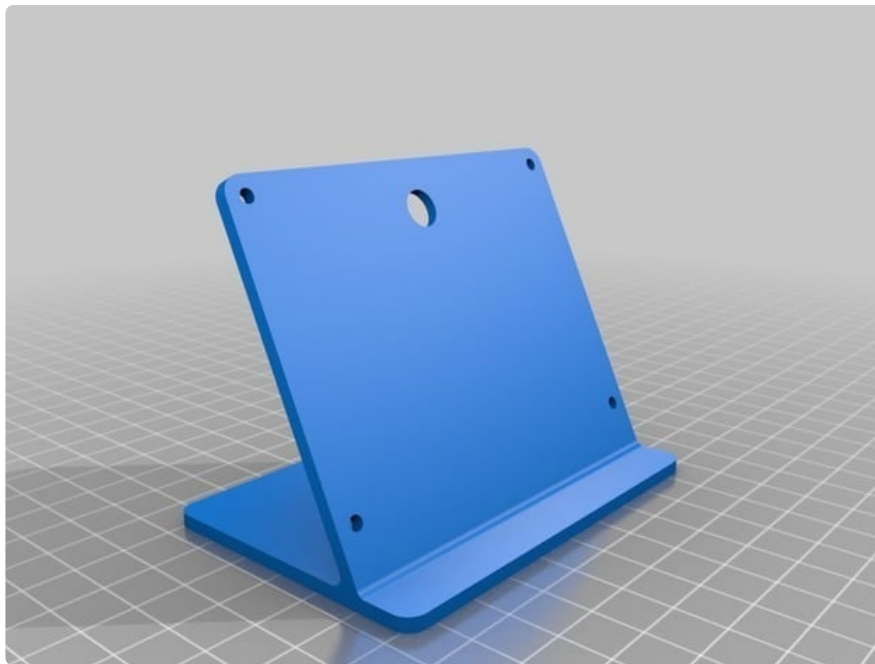
If you don't have access to a 3D printer you can optionally use an online service such as 3D Hubs to have it printed for you on demand.



Circuit Playground Bolt-On Kit

You have a Circuit Playground Express, but you need to mount it to your charming cardboard robot friend, eh? Not so easy if you...

<https://www.adafruit.com/product/3816>



Install CircuitPython

[CircuitPython](https://adafru.it/tB7) (<https://adafru.it/tB7>) is a derivative of [MicroPython](https://adafru.it/BeZ) (<https://adafru.it/BeZ>) designed to simplify experimentation and education on low-cost microcontrollers. It makes it easier than ever to get prototyping by requiring no upfront desktop software downloads. Simply copy and edit files on the **CIRCUITPY** "flash" drive to iterate.

The following instructions will show you how to install CircuitPython. If you've already installed CircuitPython but are looking to update it or reinstall it, the same steps work for that as well!

Set up CircuitPython Quick Start!

Follow this quick step-by-step for super-fast Python power :)

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

<https://adafru.it/Egk>

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

<https://adafru.it/HFd>

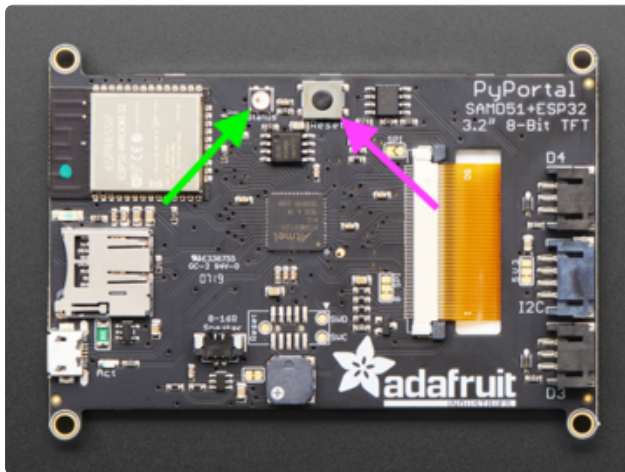


Click the link above to download the latest version of CircuitPython for the PyPortal.

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

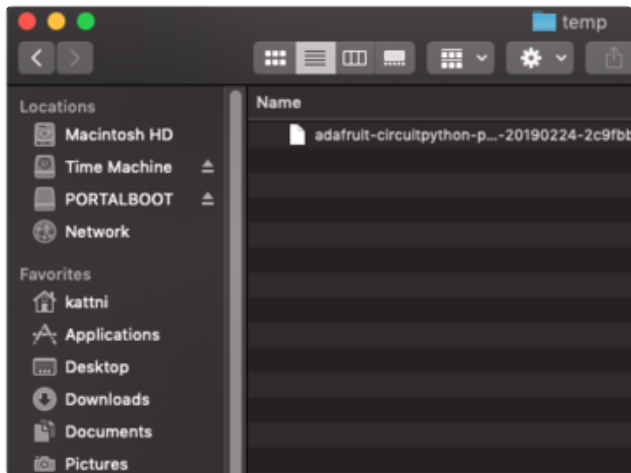
Plug your PyPortal into your computer using a known-good USB cable.

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

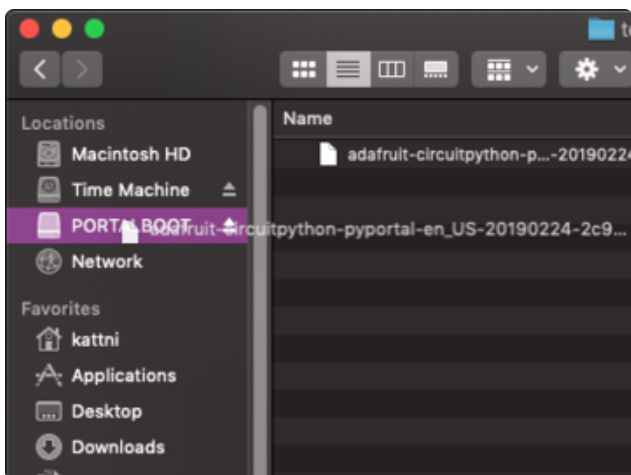


Double-click the **Reset** button on the top in the middle (magenta arrow) on your board, and you will see the NeoPixel RGB LED (green arrow) turn green. If it turns red, check the USB cable, try another USB port, etc. **Note:** The little red LED next to the USB connector will pulse red. That's ok!

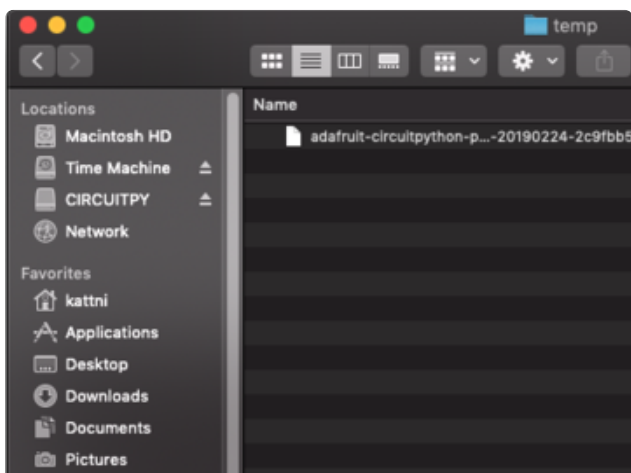
If double-clicking doesn't work the first time, try again. Sometimes it can take a few tries to get the rhythm right!



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



Drag the **adafruit-circuitpython-pyportal-
<whatever>.uf2** file to **PORTALBOOT**.



The LED will flash. Then, the **PORTALBOOT** drive will disappear and a new disk drive called **CIRCUITPY** will appear.

If you haven't added any code to your board, the only file that will be present is **boot_out.txt**. This is absolutely normal! It's time for you to add your **code.py** and get started!

That's it, you're done! :)

PyPortal Default Files

Click below to download a zip of the files that shipped on the PyPortal or PyPortal Pynt.

PyPortal Default Files

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

PyPortal CircuitPython Setup

To use all the amazing features of your PyPortal with CircuitPython, you must first install a number of libraries. This page covers that process.

Adafruit CircuitPython Bundle

Download the Adafruit CircuitPython Library Bundle. You can find the latest release here:

Latest Adafruit CircuitPython
Library Bundle

<https://adafru.it/ENC>

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

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

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

- **adafruit_esp32spi** - This is the library that gives you internet access via the ESP32 using (you guessed it!) SPI transport. You need this for anything Internet
- **adafruit_requests** - This library allows us to perform HTTP requests and get responses back from servers. GET/POST/PUT/PATCH - they're all in here!
- **adafruit_connection_manager** - used by **adafruit_requests**.
- **adafruit_pyportal** - This is our friendly wrapper library that does a lot of our projects, displays graphics and text, fetches data from the internet. Nearly all of our projects depend on it!
- **adafruit_portalbase** - This library is the base library that **adafruit_pyportal** library is built on top of.

- **adafruit_touchscreen** - a library for reading touches from the resistive touchscreen. Handles all the analog noodling, rotation and calibration for you.
- **adafruit_io** - this library helps connect the PyPortal to our free datalogging and viewing service
- **adafruit_imageload** - an image display helper, required for any graphics!
- **adafruit_display_text** - not surprisingly, it displays text on the screen
- **adafruit_bitmap_font** - we have fancy font support, and its easy to make new fonts. This library reads and parses font files.
- **adafruit_slideshow** - for making image slideshows - handy for quick display of graphics and sound
- **neopixel** - for controlling the onboard neopixel
- **adafruit_adt7410** - library to read the temperature from the on-board Analog Devices ADT7410 precision temperature sensor (not necessary for Titano or Pynt)
- **adafruit_bus_device** - low level support for I2C/SPI
- **adafruit_fakerequests** - This library allows you to create fake HTTP requests by using local files.

Create Your settings.toml File

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

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

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

CircuitPython settings.toml File

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

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

- `your_wifi_ssid`
- `your_wifi_password`

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

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

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

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

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

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

settings.toml File Tips

Here is an example **settings.toml** file.

```
# Comments are supported
CIRCUITPY_WIFI_SSID = "guest wifi"
CIRCUITPY_WIFI_PASSWORD = "guessable"
CIRCUITPY_WEB_API_PORT = 80
CIRCUITPY_WEB_API_PASSWORD = "passw0rd"
```

```
test_variable = "this is a test"
thumbs_up = "\U0001f44d"
```

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

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



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

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

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

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

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

Code done running.

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

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

Internet Connect!

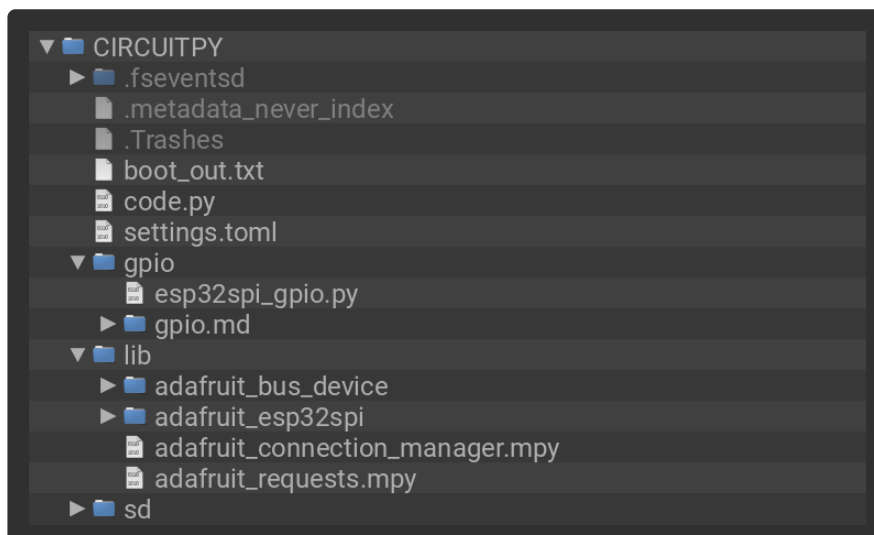
Connect to WiFi

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

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

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

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



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

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

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

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

print("ESP32 SPI webclient test")

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

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

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

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

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

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



```

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

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

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

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

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

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

print("Done!")

```

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

```

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

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

```

```

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

print("ESP32 SPI webclient test")

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

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

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

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

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

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

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

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

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

print("Connecting to AP...")
while not esp.is_connected:
    try:
        esp.connect_AP(secrets["ssid"], secrets["password"])
    except OSError as e:
        print("could not connect to AP, retrying: ", e)
        continue
print("Connected to", esp.ap_info.ssid, "\tRSSI:", esp.ap_info.rssi)
print("My IP address is", esp.ipv4_address)

```

```

print(
    "IP lookup adafruit.com: %s" %
    esp.pretty_ip(esp.get_host_by_name("adafruit.com"))
)
print("Ping google.com: %d ms" % esp.ping("google.com"))

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

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

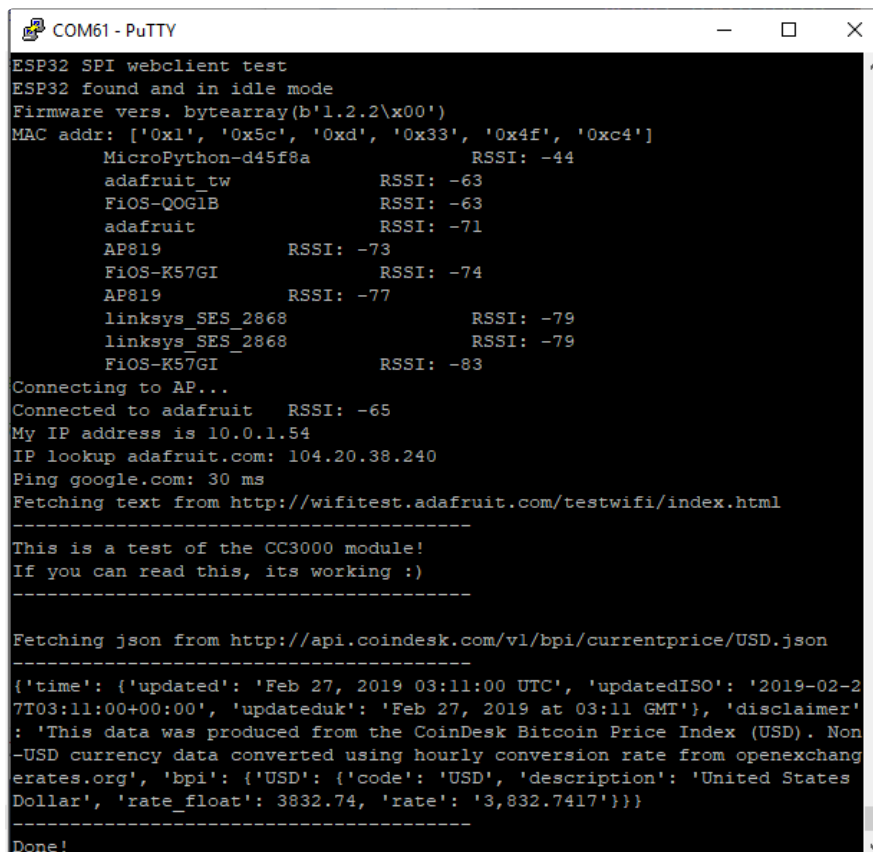
print("Done!")

```

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

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

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



```

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

```

In order, the example code...

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

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

#...

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Requests

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

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

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

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

CIRCUITPY

Temporarily unable to load content:

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

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

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

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

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

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

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

HTTP GET with Requests

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

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

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

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

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

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

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

CircuitPython_Requests can convert a **JSON-formatted** response from a server into a CPython **dict.** object.

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

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

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

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

HTTP POST with Requests

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

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

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

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

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

json_resp = response.json()
# Parse out the 'json' key from json_resp dict.
```

```
print("JSON Data received from server:", json_resp['json'])
print('- '*40)
response.close()
```

Advanced Requests Usage

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

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

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

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

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

CIRCUITPY

Temporarily unable to load content:

WiFi Manager

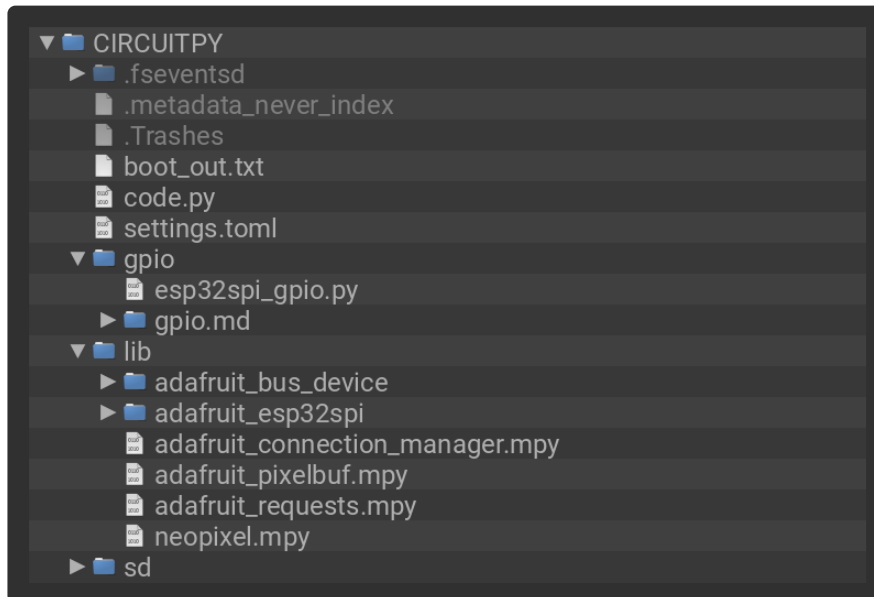
That simplest example works but it's a little finicky - you need to constantly check WiFi status and have many loops to manage connections and disconnections. For more advanced uses, we recommend using the WiFiManager object. It will wrap the connection/status/requests loop for you - reconnecting if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

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

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

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

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



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

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

print("ESP32 SPI webclient test")

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

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

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

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

```

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

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

counter = 0

while True:
    try:
        print("Posting data...", end="")
        data = counter
        feed = "test"
        payload = {"value": data}
        response = wifi.post(
            "https://io.adafruit.com/api/v2/"
            + secrets["aio_username"]
            + "/feeds/"
            + feed
            + "/data",
            json=payload,
            headers={"X-AIO-KEY": secrets["aio_key"]},
        )
        print(response.json())
        response.close()
        counter = counter + 1
        print("OK")
    except OSError as e:
        print("Failed to get data, retrying\n", e)
        wifi.reset()
        continue
    response = None
    time.sleep(15)

```

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

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

- `aio_username`
- `aio_key`

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


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

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

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

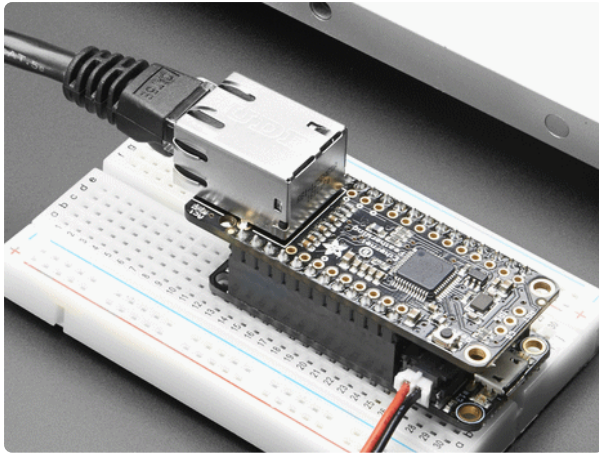
- If you do not know how to set up a feed, [follow this page and come back when you've set up a feed named **test**](https://adafru.it/f5k). (<https://adafru.it/f5k>)

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

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



For more information on the basics of doing networking in CircuitPython, see this [guide](#):



Networking in CircuitPython

By Anne Barela

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

Code PyPortal with CircuitPython

Weekly Countdown

With the PyPortal running CircuitPython code, we can set the date and time of an important event, and then have the display show how many years, days, hours, and minutes have elapsed since that time. This could be anything you want, from the lunar landing, as in our example, to the year Hydrox, the original creme-filled sandwich cookie, was introduced (1908).

The PyPortal Countdown Clock will do the following:

- Display a custom background .bmp for the event
- Determine the current local time using the WiFi connection to the Internet
- Draw out the count-up time in years, days, hours, and minutes



Adafruit IO Time Server

In order to get the precise time, our project will query the Adafruit IO Internet of Things service for the time. Adafruit IO is absolutely free to use, but you'll need to log in with your Adafruit account to use it. If you don't already have an Adafruit login, create [one here](https://adafru.it/dAQ) (<https://adafru.it/dAQ>).

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

Once you have logged into your account, there are two pieces of information you'll need to place in your **settings.toml** file: Adafruit IO username, and Adafruit IO key. Head to [io.adafruit.com](https://adafru.it/fsU) (<https://adafru.it/fsU>) and simply click the View AIO Key link on the left hand side of the Adafruit IO page to get this information.

Then, add them to the **settings.toml** file like this:

```
CIRCUITPY_WIFI_SSID = "your_wifi_ssid"
CIRCUITPY_WIFI_PASSWORD = "your_wifi_password"
AIO_USERNAME = "your_aio_username"
AIO_KEY = "your_aio_key"
```

Install CircuitPython Code and Assets

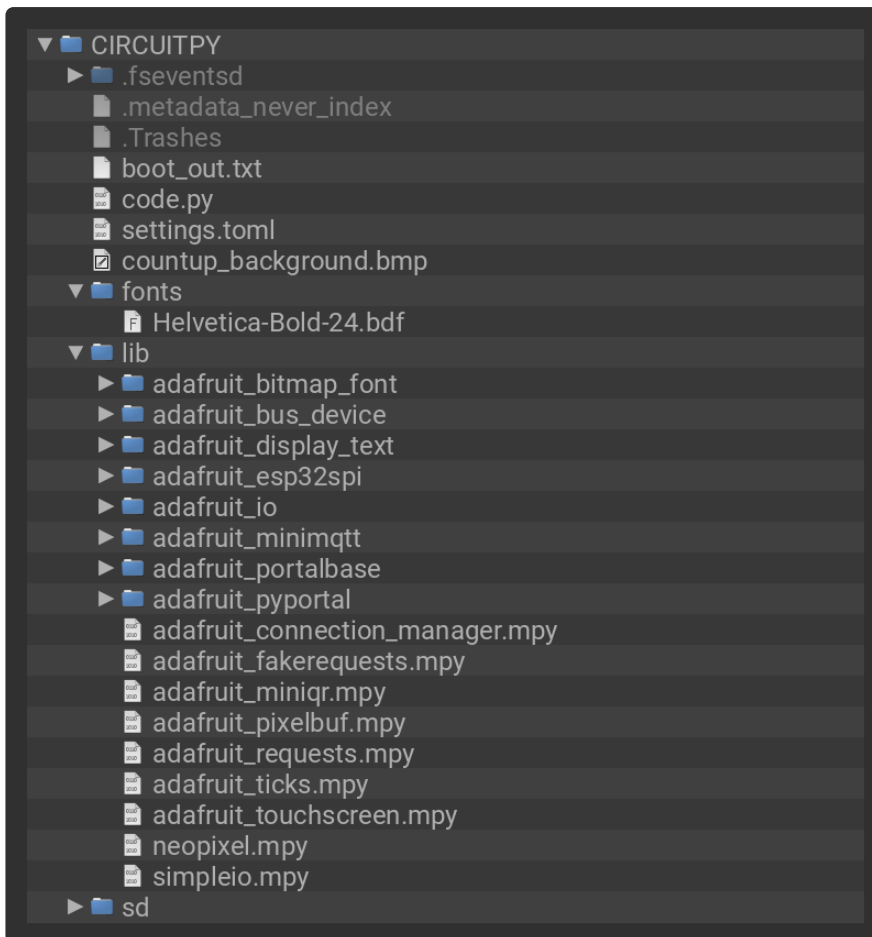
In the embedded code element below, click on the **Download Project Bundle** button, and save the .zip archive file to your computer.

Then, uncompress the .zip file, it will unpack to a folder named **PyPortal_EventCountup**.

Copy the contents of the **PyPortal_EventCountup** directory to your PyPortal **CIRCUITPY** drive.



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



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

"""
This example will figure out the current local time using the internet, and
then draw out a count-up clock since an event occurred!
Once the event is happening, a new graphic is shown
"""
import time
import board
from adafruit_pyportal import PyPortal
from adafruit_bitmap_font import bitmap_font
from adafruit_display_text.label import Label

# The time of the thing!
EVENT_YEAR = 1972
EVENT_MONTH = 12
EVENT_DAY = 7
EVENT_HOUR = 5
EVENT_MINUTE = 55
# we'll make a python-friendly structure
event_time = time.struct_time((EVENT_YEAR, EVENT_MONTH, EVENT_DAY,
                                EVENT_HOUR, EVENT_MINUTE, 0, # we don't track
                                seconds
                                -1, -1, False)) # we dont know day of week/year or
DST

# determine the current working directory
# needed so we know where to find files
cwd = ("/"+__file__).rsplit('/', 1)[0]
# Initialize the pyportal object and let us know what data to fetch and where
# to display it
```

```

pyportal = PyPortal(status_neopixel=board.NEOPIXEL,
                    default_bg=cwd+"/countup_background.bmp")

big_font = bitmap_font.load_font(cwd+"/fonts/Helvetica-Bold-24.bdf")
big_font.load_glyphs(b'0123456789') # pre-load glyphs for fast printing

years_position = (126, 15)
days_position = (13, 41)
hours_position = (118, 41)
minutes_position = (25, 68)
text_color = 0xFF0000

text_areas = []
for pos in (years_position, days_position, hours_position, minutes_position):
    textarea = Label(big_font, text=' ')
    textarea.x = pos[0]
    textarea.y = pos[1]
    textarea.color = text_color
    pyportal.splash.append(textarea)
    text_areas.append(textarea)
refresh_time = None

while True:
    # only query the online time once per hour (and on first run)
    if (not refresh_time) or (time.monotonic() - refresh_time) > 3600:
        try:
            print("Getting time from internet!")
            pyportal.get_local_time()
            refresh_time = time.monotonic()
        except RuntimeError as e:
            print("Some error occurred, retrying! -", e)
            continue

    now = time.localtime()
    print("Current time:", now)

    # We're going to do a little cheat here, since circuitpython can't
    # track huge amounts of time, we'll calculate the delta years here
    if now[0] > (EVENT_YEAR+1): # we add one year to avoid half-years
        years_since = now[0] - (EVENT_YEAR+1)
        # and then set the event_time to not include the year delta
        event_time = time.struct_time((EVENT_YEAR+years_since, EVENT_MONTH,
EVENT_DAY,
                                EVENT_HOUR, EVENT_MINUTE, 0, # we don't
track seconds
                                -1, -1, False)) # we don't know day of week/
year or DST
    else:
        years_since = 0
        print(event_time)
        since = time.mktime(now) - time.mktime(event_time)
        print("Time since not including years (in sec):", since)
        sec_since = since % 60
        since //= 60
        mins_since = since % 60
        since //= 60
        hours_since = since % 24
        since //= 24
        days_since = since % 365
        since //= 365
        years_since += since
        print("%d years, %d days, %d hours, %d minutes and %s seconds" %
            (years_since, days_since, hours_since, mins_since, sec_since))
        text_areas[0].text = '{}'.format(years_since) # set years textarea
        text_areas[1].text = '{}'.format(days_since) # set days textarea
        text_areas[2].text = '{}'.format(hours_since) # set hours textarea
        text_areas[3].text = '{}'.format(mins_since) # set minutes textarea

```



```
# update every 10 seconds  
time.sleep(10)
```

If you run into any errors, such as "ImportError: no module named `adafruit_display_text.label`" be sure to update your libraries to the latest release bundle!

How it Works

The PyPortal Event Countup is doing a couple of cool things to make your event display:

Background

First, it displays a bitmap graphic named `countup_background.bmp` as the screen's background. This is a 320 x 240 pixel RGB 16-bit raster graphic in `.bmp` format.



Time

In order to calculate the count-up value,

the PyPortal's CircuitPython code determines the local time by checking the internet time via the WiFi connection. It uses your IP address information to determine the local time. The good news is that once you've set up your timezone (or if the IP is fine) you do not have to adjust for daylight savings, leap years, etc.

In some cases, the time may not appear correctly based on your IP address, but don't fear! You can override that by manually setting the timezone in your `settings.toml` file. Plus, you can explicitly set your PyPortal to display a different time zone in case you have travel plans or a friend in Tokyo or something!

To do this, you'll add this line to your `settings.toml` file:

```
TIMEZONE = "America/Los_Angeles"
```

Here's a [great list of valid timezones \(https://adafru.it/EgK\)](https://adafru.it/EgK) from the IANA Timezone Database. Head there to find the name of the one you want. Simply find the nearest timezone to your desired location, and use that name as displayed in the **TZ database name** column.

You can adjust the following variables to make this work:

- `EVENT_YEAR`
- `EVENT_MONTH`
- `EVENT_DAY`
- `EVENT_HOUR`
- `EVENT_MINUTE`

Here's what that looks like for the precise minute of the lunar landing:

```
EVENT_YEAR = 1972
EVENT_MONTH = 12
EVENT_DAY = 7
EVENT_HOUR = 5
EVENT_MINUTE = 55
```

Font

Then, it displays the info with bitmapped fonts to overlay on top of the background. You can learn more about [converting type in this guide \(https://adafru.it/E7E\)](https://adafru.it/E7E).

Afterwards, it repeats all over again for the next week.

Customization

If you like, you can also customize the background for a different event, by making your own 320x240 16-bit RGB color .bmp file.

Then, adjust your setting to match the new event's time.



Graphics

Let's have a look at how the code places the elements on screen. Below, we can see the text items that are displayed.

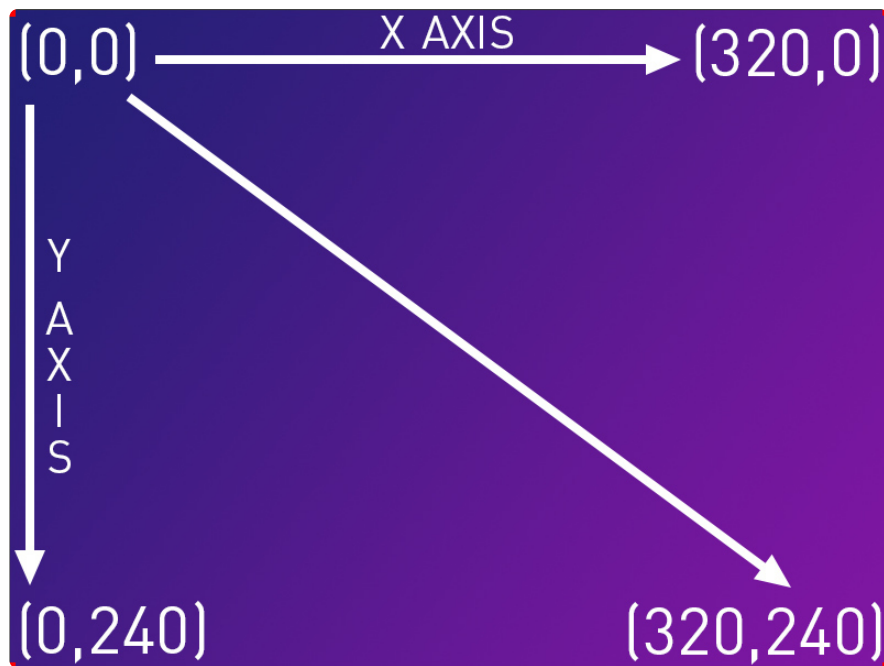


Text Position

Depending on the design of your background bitmap and the length of the text you're displaying, you may want to reposition the text and caption.

The PyPortal's display is 320 pixels wide and 240 pixels high. In order to refer to those positions on the screen, we use an x/y coordinate system, where x is horizontal and y is vertical.

The origin of this coordinate system is the upper left corner. This means that a pixel placed at the upper left corner would be (0,0) and the lower right corner would be (320, 240).



Text Color

Another way to customize your display is to adjust the color of the text. The line `text_color=0xFFFFFF` in the constructor shows how. You will need to use the hexadecimal value for any color you want to display.

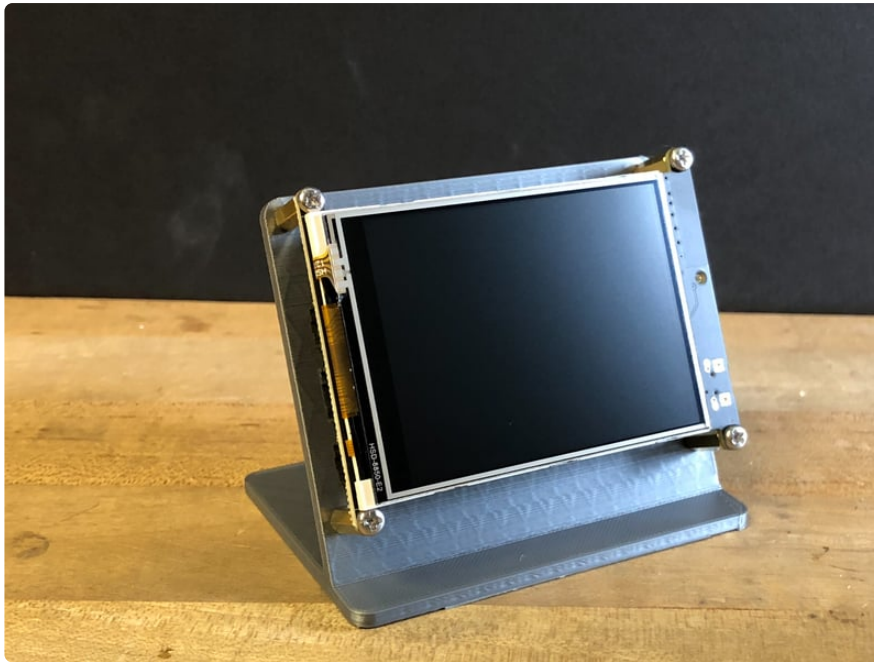
You can use something like <https://htmlcolorcodes.com/> (<https://adafru.it/Eb7>) to pick your color and then copy the hex value, in this example it would be `0x0ED9EE`

So, in order to customize the position and color of the text, you would adjust the values in these lines of code in `code.py` :

```
years_position = (126, 15)
days_position = (13, 41)
hours_position = (118, 41)
minutes_position = (25, 68)
text_color = 0xFF0000
```

3D Printed Stand

If you'd like to create a 3D printed stand for your PyPortal Countdown Clock, you can follow the [general instructions in this guide](https://adafru.it/EcO) (<https://adafru.it/EcO>), but use the [horizontal PyPortal Stand model linked here](https://adafru.it/EcN) (<https://adafru.it/EcN>).



Use the four sets of standoffs and screws to fasten them together as shown.

